

**NEW ALGORITHMS FOR QUADRATIC  
UNCONSTRAINED BINARY OPTIMIZATION  
(QUBO) WITH APPLICATIONS IN ENGINEERING  
AND SOCIAL SCIENCES**

**BY GABRIEL TAVARES**

A dissertation submitted to the  
Graduate School—New Brunswick  
Rutgers, The State University of New Jersey  
in partial fulfillment of the requirements  
for the degree of  
Doctor of Philosophy  
Graduate Program in Operations Research

Written under the direction of

**DR. PETER L. HAMMER and DR. ENDRE BOROS**

and approved by

---

---

---

---

---

---

New Brunswick, New Jersey

May, 2008

© 2008

Gabriel Tavares

**ALL RIGHTS RESERVED**

## ABSTRACT OF THE DISSERTATION

# New algorithms for Quadratic Unconstrained Binary Optimization (QUBO) with applications in engineering and social sciences

by Gabriel Tavares

Dissertation Director:

DR. PETER L. HAMMER and DR. ENDRE BOROS

This dissertation investigates the Quadratic Unconstrained Binary Optimization (QUBO) problem, i.e. the problem of minimizing a quadratic function in binary variables. QUBO is studied at two complementary levels. First, there is an algorithmic aspect that tells how to preprocess the problem, how to find heuristics, how to get improved bounds and how to solve the problem with all the above ingredients. Second, there is a practical aspect that uses QUBO to solve various applications from the engineering and social sciences fields including: via minimization, 2D/3D Ising models, 1D Ising chain models, image binarization, hierarchical clustering, greedy graph coloring/partitioning, MAX-2-SAT, MIN-VC, MAX-CLIQUE, MAX-CUT, graph stability and minimum  $k$ -partition.

Several families of fast heuristics for QUBO are analyzed, which include a novel probabilistic based class of methods.

It is shown that there is a unique maximal set of persistencies for the linearization model for QUBO. This set is determined in polynomial time by a maximum flow followed by the computation of the strong components of a network that has  $2n+2$  nodes, where

$n$  is the number of variables. The identification of the above persistencies leads to a unique decomposition of the function, such that each component can be optimized separately. To find further persistencies, two additional techniques are proposed: one is based on the second order derivatives of Hammer et al. [121]; the other technique is a probing procedure on the two possible values of the variables. These preprocessing tools work remarkably well for certain classes of problems.

We improved the Iterated Roof–Dual bound (IRD) of [51] by proposing two combinatorial methods: one was named the squeezed IRD; and the second was called the project–and–lift IRD method.

The cubic–dual bound can be found by means of linear programming by adding a set of triangle inequalities to the standard linearization, whose number is cubic in the number of variables. We show that this set can be reduced depending on the coefficients of the terms of the function. This leads to the possibility of computing the cubic–duals of larger QUBOs.

## Acknowledgements

I would like to begin by mentioning how fortunate I was to have Peter L. Hammer as a co-advisor. I will not forget his advice and exceptional wisdom to solve real life problems. He was truly a role model in many aspects of teaching and research excellence, humility, kindness, and sympathy. I learned so much from working with him and from observing the way he lived his life.

I wish to extend my deepest thanks to my co-advisor Dr. Endre Boros for all of the help, motivation, support and guidance that he has provided to me over all these years. He allowed me to have great freedom in exploring different methods and ideas, but was always there to provide top-notch insight and advice.

Many other people at Rutgers have also been generous in helping me throughout all these years. I would especially like to thank the RUTCOR Faculty and in particular Dr. András Prékopa, Dr. Jonathan Eckstein and Dr. Vladimir Gurvich, for their teachings and for their help. I would like to thank my fellow Graduate class mates who enriched my graduate school experience. I am especially thankful to Tibérius Bonates and Cem Iyigun. I am very thankful to Clare Smietana, Terry Hart, Lynn Agre and Katie D'Agosta for assisting me so efficiently in any administrative related requests, but more importantly because of their constant presence and advice.

I would like to thank many other people that I have been fortunate to collaborate in several fronts related to the topic of this dissertation. In particular I would like to thank to Dr. Bruno Simeone, Dr. Ramin Zabih and Dr. Bela Vizvari.

I would like to thank Alkis Vazacopoulos that gave me a chance to apply my knowledge in the industry during this work, and was always supportive of this work.

I would like to acknowledge and thank the financial support I was awarded by the Portuguese FCT through the FSE in the context of the III Quadro Comunitário de

Apoio.

I would like to thank all the members of my family who have been so supportive of all my efforts. I would like to thank my father Lino Tavares, my mother Maria Tavares, and my sister and bother-in-law Mariline and Lino Tavares, for their support and lodging when I started my studies at Rutgers. I would like to especially thank my mother-in-law Mariline Pinto who was so supportive and helpful all along these years.

Finally, I would like to thank my wife Lisa and my children David and Sara the most, for their endless support and belief in me, and more especially for the time they have sacrificed without me, and that I hope to recover as soon as possible.

Thank you to all.

## Dedication

To my dear wife, Lisa, and children, David and Sara, for all their love and support.

In Memory of my grand-father José Proença.

In Memory of my father-in-law António Pinto.

In Memory of my mentor Peter L. Hammer.

# Table of Contents

<b>Abstract</b> . . . . .	ii
<b>Acknowledgements</b> . . . . .	iv
<b>Dedication</b> . . . . .	vi
<b>List of Tables</b> . . . . .	xiv
<b>List of Figures</b> . . . . .	xxi
<b>1. Introduction</b> . . . . .	1
1.1. Definitions and notation . . . . .	5
1.1.1. Pseudo-Boolean functions . . . . .	6
1.1.2. Quadratic pseudo-Boolean functions . . . . .	8
1.2. A generic illustrative example . . . . .	9
<b>2. Classic Combinatorial Optimization Models</b> . . . . .	10
2.1. Pseudo-Boolean optimization . . . . .	10
2.2. Graph theory . . . . .	11
2.2.1. Maximum clique . . . . .	11
2.2.2. Minimum vertex cover . . . . .	13
2.2.3. Maximum cut . . . . .	14
2.3. Maximum satisfiability . . . . .	15
<b>3. Test Beds</b> . . . . .	17
3.1. Benchmarks with prescribed density . . . . .	19
3.1.1. Benchmark families . . . . .	21
3.1.2. Randomly generated test problems . . . . .	24

3.2.	Graphs for maximum clique . . . . .	28
3.3.	Planar graphs for minimum vertex cover . . . . .	36
3.4.	Graphs for MAX-CUT . . . . .	37
3.4.1.	Benchmark families . . . . .	38
3.4.2.	Graphs with $m$ -Hamiltonian random cycles . . . . .	43
3.5.	Maximum 2-satisfiability test problems . . . . .	45
3.5.1.	Benchmark families . . . . .	45
3.5.2.	Randomly generated MAX-2-SAT formulas . . . . .	47
<b>4.</b>	<b>Basic Tools and Concepts . . . . .</b>	<b>50</b>
4.1.	Persistency . . . . .	51
4.2.	First order partial derivatives . . . . .	51
4.3.	Second order derivatives . . . . .	54
4.4.	Locotope . . . . .	57
4.5.	Implication graph . . . . .	61
4.6.	Posiform minimization . . . . .	64
4.6.1.	Standard quadratic posiforms . . . . .	65
4.7.	Rounding procedures and derandomization . . . . .	66
4.8.	Best linear Euclidean approximations . . . . .	70
<b>5.</b>	<b>Roof-Duality and New Persistency Results . . . . .</b>	<b>81</b>
5.1.	Minorization . . . . .	82
5.2.	Linearization . . . . .	85
5.2.1.	Persistency of linearizations . . . . .	86
5.3.	Implication networks . . . . .	92
5.4.	Computational results . . . . .	104
5.4.1.	Network flow model versus linear programming . . . . .	106
5.4.2.	Application of roof-duality to VLSI design . . . . .	108
5.4.2.1.	Via minimization . . . . .	109
5.4.2.2.	Cell flipping in standard cell technology . . . . .	110

<b>6. Heuristics</b>	116
6.1. One-pass heuristics	117
6.1.1. DDT methods	117
6.1.2. Greedy heuristics	120
6.1.2.1. Best linear approximation methods	121
6.1.2.2. Probabilistic methods	126
6.1.2.3. Rounding methods	131
6.1.3. Measuring heuristics performance	139
6.1.4. Computational results	141
6.1.4.1. Computing time	144
6.1.4.2. Quality of solutions	146
6.1.5. Comparing proposed methods to other results from the literature	148
6.2. Local-search heuristics	151
6.2.1. Methods	152
6.2.1.1. Basic concepts and notations	152
6.2.1.2. Algorithms	155
6.2.1.3. Implementation details	157
6.2.2. Algorithm selection	159
6.2.3. Parametric analysis of heuristic ACSIOM	166
6.2.3.1. Quality of solutions	167
6.2.3.2. Computational efficiency	171
6.2.4. Comparing ACSIOM to other results from the literature	173
6.2.5. Conclusions	173
6.3. One-pass heuristics enhancement by local-search	174
6.3.1. Computing time	176
6.3.2. Quality of solutions	177
6.3.3. Concluding remarks	177
<b>7. Preprocessing</b>	180

7.1.	Basic preprocessing tools . . . . .	182
7.1.1.	First order derivatives . . . . .	182
7.1.2.	Second order derivatives and co-derivatives . . . . .	182
7.2.	Roof-duality . . . . .	183
7.2.1.	Strong persistency . . . . .	184
7.2.2.	Weak persistency . . . . .	185
7.2.3.	Decomposition . . . . .	188
7.3.	Combining basic tools . . . . .	190
7.3.1.	Enhancing roof-duality by probing . . . . .	190
7.3.2.	Consensus . . . . .	194
7.4.	Algorithm and implementation . . . . .	194
7.5.	Test problems . . . . .	198
7.5.1.	Benchmarks with prescribed density . . . . .	199
7.5.2.	Maximum cliques of graphs . . . . .	199
7.5.3.	Minimum vertex cover problems of planar graphs . . . . .	199
7.5.4.	Graphs for MAX-CUT . . . . .	200
7.5.5.	MAX-2-SAT formulas . . . . .	200
7.6.	Computational experiments . . . . .	200
7.6.1.	Test environment . . . . .	200
7.6.2.	Results . . . . .	201
7.7.	Optimal vertex covers of planar graphs . . . . .	206
7.7.1.	Deriving minimum vertex cover from QUBO's optimum . . . . .	206
7.7.2.	Preprocessing . . . . .	207
7.7.3.	Optimization . . . . .	208
7.7.4.	Minimum vertex covers of very large planar graphs . . . . .	209
7.8.	Final remarks . . . . .	211
<b>8.</b>	<b>Lower Bounds to the Minimum . . . . .</b>	<b>214</b>
8.1.	Bi-forms and packing of cycles . . . . .	219

8.2.	Optimal rooted noose packings relationship to the network model . . . .	229
8.3.	Rooted noose packing structure and decomposition . . . . .	235
8.4.	Iterated roof-duality . . . . .	237
8.4.1.	Computational results . . . . .	238
8.4.1.1.	MAX-CUT . . . . .	242
8.4.1.2.	Randomly generated quadratic binary optimization prob- lems . . . . .	244
8.5.	Squeezed iterated roof-duality . . . . .	248
8.5.1.	Computational results . . . . .	252
8.6.	Project-and-lift iterated roof-duality . . . . .	256
8.6.1.	Computational results . . . . .	263
8.7.	Linearization models for sparse QUBOs . . . . .	265
8.7.1.	Computational results . . . . .	270
8.8.	A closer look at $C_4$ . . . . .	275
8.8.1.	Computational results . . . . .	283
<b>9.</b>	<b>Exact Methods</b> . . . . .	<b>287</b>
9.1.	Background . . . . .	293
9.2.	Enumerative approaches . . . . .	295
9.2.1.	Computational results . . . . .	300
9.3.	Branch-and-bound with network flows . . . . .	302
9.3.1.	Computational results . . . . .	305
9.3.1.1.	MIN-VC of planar graphs . . . . .	305
9.3.1.2.	Benchmarks with prescribed density . . . . .	306
9.3.1.3.	MAX-2-SAT . . . . .	309
9.3.1.4.	MAX-CUT . . . . .	314
9.3.1.5.	One dimensional Ising chains . . . . .	314
9.4.	Linearization enhanced with logical cuts . . . . .	317
9.4.1.	Computational results . . . . .	319

9.4.1.1.	Benchmarks with prescribed density . . . . .	319
9.4.1.2.	MAX-2-SAT . . . . .	321
9.4.1.3.	MAX-CUT . . . . .	321
9.4.1.4.	Minimum 3-partition . . . . .	325
<b>10.</b>	<b>Applications . . . . .</b>	<b>327</b>
10.1.	Minimum vertex cover problem . . . . .	327
10.1.1.	Solving MIN-WVC using QUBO . . . . .	331
10.1.2.	Struction . . . . .	336
10.1.3.	Simplified reduction techniques for MIN-VC . . . . .	338
10.1.4.	Implementation algorithms . . . . .	341
10.1.5.	QUBO solvers considered for testing . . . . .	344
10.1.6.	Minimum vertex cover of planar graphs . . . . .	345
10.1.6.1.	Randomly generated planar graphs . . . . .	346
10.1.6.2.	Two dimensional grid graphs . . . . .	350
10.1.6.3.	Regular graphs consisting of hexagons . . . . .	353
10.1.6.4.	Regular graphs consisting of triangles . . . . .	355
10.1.6.5.	A family of planar graphs with small diameter . . . . .	356
10.1.6.6.	Planar graphs with Delaunay triangulations . . . . .	358
10.1.7.	Preventing Internet DDoS attacks . . . . .	363
10.1.7.1.	Route-based distributed packet filtering . . . . .	363
10.1.8.	Maximum independent set of real world graphs . . . . .	365
10.2.	Clustering . . . . .	368
10.2.1.	A QUBO model to 2-partitioning . . . . .	369
10.2.1.1.	Hierarchical clustering . . . . .	370
10.2.2.	Greedy graph coloring and partitioning . . . . .	372
10.2.2.1.	The protein-protein interaction map of Helicobacter Py- lori . . . . .	373
10.3.	A QUBO approach to image binarization . . . . .	376

10.3.1. Computational results . . . . .	380
<b>11. Conclusions . . . . .</b>	<b>385</b>
<b>Appendix A. Test Problems Characteristics . . . . .</b>	<b>387</b>
<b>Appendix B. Heuristics Statistics . . . . .</b>	<b>402</b>
<b>Appendix C. Preprocessing Statistics . . . . .</b>	<b>405</b>
<b>Appendix D. Iterated Roof–Duality Experiments . . . . .</b>	<b>413</b>
<b>References . . . . .</b>	<b>419</b>
<b>Vita . . . . .</b>	<b>436</b>

## List of Tables

3.1. Statistics of the QUBO problems used for testing the proposed algorithms.	18
3.2. QUBO benchmarks with prescribed density. . . . .	22
3.3. Characteristics of QUBO problems in the <i>Small</i> family. . . . .	25
3.4. Characteristics of QUBO problems in the <i>Medium</i> family. . . . .	26
3.5. Characteristics of QUBO problems in the <i>Large</i> family. . . . .	27
3.6. Characteristics of QUBO problems in the <i>Massive</i> family. . . . .	28
3.7. DIMACS graphs for the maximum clique problem (Part I). . . . .	29
3.8. DIMACS graphs for the maximum clique problem (Part II). . . . .	30
3.9. Additional graphs ([79, 196, 237]) for the maximum clique problem. . .	35
3.10. Graphs of Pardalos and Desai [192] for the weighted maximum clique problem. . . . .	36
3.11. Comparative statistical numbers about the LEDA benchmarks. . . . .	37
3.12. G-graphs of Helmberg and Rendl [140] for MAX-CUT. . . . .	39
3.13. $G_{\pm 1}$ -graphs of Helmberg and Rendl [140] for MAX-CUT. . . . .	40
3.14. Cubic lattice graphs of Burer et al. [74] for MAX-CUT. . . . .	41
3.15. DIMACS torus graphs for MAX-CUT. . . . .	42
3.16. Graphs of Homer and Peinado [145] for MAX-CUT. . . . .	43
3.17. Graphs of Kim, Kim and Moon [157] for MAX-CUT. . . . .	44
3.18. Graphs with $m$ -Hamiltonian randomly generated cycles for MAX-CUT.	44
3.19. MAX-2-SAT instances of Borchers and Furman [47]. . . . .	46
3.20. Weighted MAX-2-SAT instances of Borchers and Furman [47]. . . . .	46
3.21. Profiles of probabilities for a clause to belong to a (weighted) MAX-2- SAT formula. . . . .	48
3.22. Randomly generated (weighted) MAX-2-SAT formulas. . . . .	49

5.1. Network flow model versus LP to find the roof-duals of the $G_1$ problems proposed by Glover et al. [109]. . . . .	107
5.2. Average relative gap ( $g$ ) to the best known lower bound ( $z$ ) and average computing times of the roof-duals of 10% dense QUBO problems (Beasley [37]). . . . .	108
5.3. Via minimization problems of Homer and Peinado [145]. . . . .	110
5.4. Impact of roof-duality on large 2-pin cell flipping randomly generated problems having 1 000 000 nets each. . . . .	113
5.5. Average relative gap of roof-duality on randomly generated 2-pin cell flipping problems having 10 000 nets. . . . .	114
6.1. Probability distributions of random variables used to characterize the probability distribution of partial derivatives of quadratic pseudo-Boolean functions. . . . .	129
6.2. Starting points considered in the computational experiments. . . . .	134
6.3. One-pass heuristics for QUBO considered in the computational experiments. . . . .	143
6.4. Families of QUBO problems used to evaluate the proposed one-pass heuristics. . . . .	144
6.5. Computing time of the one-pass heuristics across several families of QUBO problems. . . . .	145
6.6. Quality of the one-pass heuristics across several families of QUBO problems. . . . .	147
6.7. One-pass heuristics that minimize the approximate errors of several families of QUBO problems. . . . .	148
6.8. Quality of solutions comparison between the proposed methods and the one-pass heuristics from the literature. . . . .	150
6.9. List of target sets considered in the computational experiments. . . . .	157
6.10. Criteria list of pivot selection considered in the computational experiments.	157
6.11. Performance ratio sets $R_{i,p,c;\mathcal{F}}$ for algorithms $A_{I,P,C}$ . . . . .	162

6.12. Computing times $T_{i,p,c;\mathcal{F}}$ for algorithms $A_{I,P,C}$ ( $I = \{1, 2, 3, 4, 6\}$ , $P = \{1, 2\}$ , and $C = \{1, \dots, 4\}$ ). . . . .	164
6.13. Correlations between quality of solutions ( $r_{i,1,1}$ , $i = 2, 3, 4$ ), computing times ( $t_{i,1,1}$ , $i = 2, 3, 4$ ) and input parameters ( $n$ , $d$ , $\rho$ and $p$ ) of the test problems in $\mathcal{S}$ . . . . .	165
6.14. Algorithms returning on average the highest value for the test problems, according to the $\rho$ parameter. . . . .	166
6.15. Correlations between quality of solutions ( $r_{ACSIOM}$ ), computing times ( $t_{ACSIOM}$ ) and input parameters ( $n$ , $d$ , $\rho$ and $p$ ) of the test problems in $\mathcal{T}$ . . . . .	168
6.16. Number of variables ( $n$ ) versus density ( $d$ ) analysis on the heuristic computation times ( $t_{ACSIOM}$ ) for the <i>large</i> family of test problems. . . . .	172
6.17. Average relative error of local maximization heuristics within several <i>benchmark</i> sub-families. . . . .	174
6.18. Performance gain by locally improving the solutions of one-pass heuristics. . . . .	177
6.19. Percentage number of tests where the STEEPEST-ASCENT( $\frac{1}{2}, \dots, \frac{1}{2}$ ) heuristic gives better solution values than STEEPEST-ASCENT(Bernoulli( $\frac{1}{2}$ ), $\dots$ , Bernoulli( $\frac{1}{2}$ )), for the $G_1$ QUBO problems ([109]). . . . .	179
7.1. Average QUBO simplifications and decomposition after preprocessing. . . . .	203
7.2. Preprocessing strategies recommended for the benchmarks. . . . .	204
7.3. Comparative preprocessing results for minimum vertex cover problems in planar graphs. . . . .	208
7.4. Average computing times of optimal vertex covers for graphs belonging to the LEDA benchmarks. . . . .	209
7.5. Average computing times over 3 experiments of optimal vertex covers for graphs belonging to the PVC RUDY benchmark. . . . .	210
8.1. Characteristics of computer systems used for testing the algorithms. . . . .	242
8.2. Bounding MAX-CUT. . . . .	245
8.3. Iterated roof-duals of QUBO problems with 100 variables (Glover et al. [108]). . . . .	246

8.4. Iterated roof-duals of 10% dense quadratic unconstrained binary optimization problems (Beasley [37]). . . . .	248
8.5. Squeezed iterated roof-duals of QUBO problems with 100 variables (Glover et al. [108]). . . . .	254
8.6. Squeezed iterated roof-duals of 10% dense quadratic unconstrained binary optimization problems (Beasley [37]). . . . .	255
8.7. Project-and-lift and squeezed iterated roof-duals of QUBO problems with 100 variables (Glover et al. [108]). . . . .	266
8.8. Project-and-lift and squeezed iterated roof-duals of MAX-2-SAT (Bonami and Minoux [46]). . . . .	267
8.9. Linear programming bounds of MAX-2-SAT (Bonami and Minoux [46]).	272
8.10. Upper bounds based on linear programming for MAX-CUT graphs from the Hamilton family. . . . .	274
8.11. Lower bounds for M3P problems proposed by Anjos et al. [20]. . . . .	286
9.1. Exact solutions of some quadratic pseudo-Boolean functions from family $\mathcal{N}$ . . . . .	291
9.2. Computing times to find the optimum values of some $F_2$ and $G_2$ problems of Kochenberger et al. [158]. . . . .	300
9.3. Statistics of DEPTH-FIRST to find optimal solutions of some $F_2$ and $G_2$ problems of Kochenberger et al. [158]. . . . .	301
9.4. Optimal solutions of the family $C$ problems proposed by Pardalos and Rodgers [195]. . . . .	302
9.5. Using B&B to find minimum vertex covers of planar graphs. . . . .	306
9.6. Proving optimality to Beasley [37] QUBO problems with 250 variables. .	307
9.7. Proving optimality to Beasley [37] QUBO problems with 250 variables. .	310
9.8. Proving optimality to the Borchers and Furman [47] MAX-2-SAT instances. . . . .	312
9.9. Proving optimality to the Ibaraki et al. [148] break minimization (MAX-2-SAT) problems. . . . .	313

9.10. Proving optimality to some Resende [209] MAX-CUT problems. . . . .	314
9.11. Computing times to find the minimum energy state of the one-dimensional Ising chains proposed by Rendl et al. [206, 208]. . . . .	316
9.12. Average computing times to find the minimum energy state of larger one-dimensional Ising chains. . . . .	317
9.13. Maximum of QUBO problems with 250 variables and 10% density (Beasley [37]). . . . .	320
9.14. Optimal solutions for the Bonami and Minoux [46] MAX-2-SAT formulas.	322
9.15. MAX-CUT of $5 \times 5 \times 5$ spin glass Ising models (Burer et al. [74]). . . .	324
9.16. Finding the MAX-CUT for the DIMACS torus graphs. . . . .	325
9.17. MAX-CUT of the toroidal $G_{\pm 1}$ -graphs of Helmberg and Rendl [140]. . .	325
9.18. Optimal solutions for the M3P problems proposed by Anjos et al. [20]. .	326
10.1. Average computing times of MIN-VC-USING-ROOF-DUALITY applied to the PVC RUDY benchmark. . . . .	347
10.2. Average number of variables of QUBOs solved by PREPRO* within MIN- VC-USING-ROOF-DUALITY, when applied to the PVC RUDY benchmark.	348
10.3. Minimum vertex covers of planar graphs randomly generated by LEDA. . . .	349
10.4. Minimum vertex covers of 2 dimensional grid graphs. . . . .	352
10.5. MIN-VC-USING-ROOF-DUALITY computing times of minimum vertex covers of some $G_{m,n}$ graphs. . . . .	352
10.6. Minimum vertex covers of regular planar graphs consisting of hexagons (generated by GenGraph). . . . .	354
10.7. MIN-VC-USING-ROOF-DUALITY computing times of minimum vertex covers of some $H_{m,n}$ graphs. . . . .	355
10.8. Minimum vertex covers of regular planar graphs consisting of triangles (generated by GenGraph). . . . .	356
10.9. Minimum vertex covers of regular dense planar graphs with low diameter (generated by GenGraph). . . . .	359

10.10	Minimum vertex covers of planar graphs with Delaunay triangulations (generated by GenGraph). . . . .	361
10.11	Minimum vertex covers of planar graphs with Delaunay triangulations (generated by GenGraph). . . . .	361
10.12	Minimum ASes covers of NLANR routing data ([10]). . . . .	364
10.13	Real world graphs. . . . .	367
10.14	Combinatorics of real world graphs. . . . .	368
10.15	Combinatorics of the H. Pylori map. . . . .	374
10.16	Large stable sets of the H. Pylori map. . . . .	374
10.17	Large 2–distance cliques of the H. Pylori map. . . . .	375
10.18	Large 3–distance cliques of the H. Pylori map. . . . .	375
10.19	Large 4–distance cliques of the H. Pylori map. . . . .	376
A.1.	QUBO problems of Beasley [37]. . . . .	387
A.2.	QUBO problems of Glover, Kochenberger and Alidaee [108]. . . . .	389
A.3.	QUBO problems of Glover, Kochenberger, Alidaee and Amini [109]. . . . .	390
A.4.	QUBO submodular problems of Glover, Alidaee, Rego and Kochenberger [107]. . . . .	390
A.5.	QUBO problems of Palubeckis and Tomkevièius [189]. . . . .	391
A.6.	Minimum values of the <i>Small</i> family QUBO problems. . . . .	392
A.7.	Best known values of the <i>Large</i> family QUBO problems. . . . .	393
A.8.	Best known cuts of the <i>Hamilton</i> graphs. . . . .	396
A.9.	Best known values of the randomly generated MAX–2–SAT problems. . . . .	398
B.1.	Least squares fitting of performance ratios ( $r_{ACSIOM}$ ) in the <i>medium</i> test problems with $\rho \leq 0.525$ . . . . .	402
B.2.	Number of variables ( $n$ ) versus density ( $d$ ) analysis on the performance ratios values ( $r_{ACSIOM}$ ) for the <i>medium</i> problems. . . . .	403
B.3.	Number of variables ( $n$ ) versus density ( $d$ ) analysis of number of round- ings per variable necessary by $A_{ACSIOM}$ in the problems of the <i>medium</i> family. . . . .	404

C.1. PREPRO statistical report on the QUBO problems of Glover, Kochen- berger and Alidaee [108]. . . . .	407
C.2. PREPRO statistical report on the QUBO problems of Beasley [37]. . . .	408
C.3. PREPRO statistical report on the $c$ -fat and Hamming graphs. . . . .	409
C.4. PREPRO statistical report on the minimum vertex cover of the RUDY planar graphs. . . . .	410
C.5. PREPRO statistical report on the MAX-CUT graphs. . . . .	411
C.6. PREPRO statistical report on the MAX-2-SAT formulas of Borchers and Furman [47]. . . . .	412
D.1. MAX-CUT (Kim et al. [157]). . . . .	414
D.2. MAX-CUT (Homer and Peinado [145]). . . . .	415
D.3. MAX-CUT on cubic lattice graphs (Burer et al. [74]). . . . .	416
D.4. MAX-CUT for torus graphs (7th DIMACS Implementation Challenge).	417
D.5. Upper bounds of 10% dense QUBO maximization problems (Beasley [37]).	418

## List of Figures

4.1. Locotope Tightening Algorithm (LTA). . . . .	59
4.2. Standard posiform algorithm. . . . .	67
5.1. The network $G_{\phi_g}$ corresponding to the posiform $\phi_g$ of Example 5.2. . . .	93
5.2. The network $G_{\psi_g}$ corresponding to the posiform $\psi_g$ of Example 5.3. . .	96
5.3. Capacitated networks of Example 5.4. . . . .	101
6.1. The DDT heuristics. . . . .	118
6.2. Algorithm description of one-pass heuristics. . . . .	121
6.3. Average computing time of the (fastest, slowest, average case) one-pass QUBO heuristics according to the number of variables ( $n$ ). . . . .	146
6.4. Description of the details shown in a cell of a cross-analysis table. . . . .	161
6.5. $\text{Exp}[r_{i,1,1}]$ , $i \in \{2, 3, 4\}$ , values as a function of $\rho$ in the random test problems. . . . .	165
6.6. Distribution of problems according to $r_{ACSIOM}$ . . . . .	168
6.7. Average computing times of one-pass heuristics and of their local im- provement by steepest ascent. . . . .	176
7.1. The network $G_\phi$ corresponding to the posiform $\phi$ of Example 7.1. We indicate only those arcs which have positive capacities. . . . .	188
7.2. The network $G_\phi$ corresponding to the posiform $\phi$ of Example 7.2. We disregarded the values of the capacities, and indicated only those arcs which have positive capacities. The dashed arcs represent arcs connecting the strong components of $G_\phi$ . . . . .	190
7.3. PREPRO algorithm. . . . .	195
7.4. Planar graph for which PREPRO does not find any persistent result. . .	206
8.1. The graph $G_g$ of the bi-form given in Example 8.1. . . . .	223

8.2. The residual graph $G'_g$ of the bi-form given in Example 8.1. . . . .	226
8.3. ITERATED-ROOF-DUAL algorithm. . . . .	239
8.4. SQUEEZED-ITERATED-ROOF-DUAL algorithm. . . . .	253
8.5. PROJECT&LIFT-ITERATED-ROOF-DUAL algorithm. . . . .	264
9.1. DEPTH-FIRST algorithm. . . . .	297
9.2. FIND-PERSISTENCIAS algorithm. . . . .	298
9.3. UPDATE-BOUND&VECTORS algorithm. . . . .	299
9.4. XPRESS-QUBO algorithm. . . . .	318
10.1. Representing a MIN-WVC problem using the implication network model.	333
10.2. Residual network of the implication network model of a MIN-WVC prob- lem. . . . .	335
10.3. On iteration of struction of a graph. . . . .	338
10.4. Data reduction algorithm for MIN-VC. . . . .	342
10.5. Exact method for MIN-VC using data reduction techniques and roof- duality. . . . .	343
10.6. Exact method for MIN-VC using data reduction techniques and struction.	344
10.7. Planar graph with 50 vertices and 140 edges, generated by Gengraph us- ing LEDA. The MIN-VC size is 34. A minimum vertex cover is indicated by the set of nodes with white color. . . . .	347
10.8. Graph $G_{20,10}$ . The MIN-VC size is 100. A minimum vertex cover is indicated by the set of nodes with white (black) color. . . . .	351
10.9. Graph $H_{15,10}$ , generated by Gengraph. The MIN-VC size is 175. A minimum vertex cover is indicated by the set of nodes with white color.	353
10.10 Family of regular graphs $T_s$ ( $s = 2, \dots, 7$ ) consisting of triangles gener- ated by Gengraph. A minimum vertex cover is indicated by the set of nodes with white color. . . . .	355
10.11 Regular graph $T_{44}$ consisting of triangles with 990 vertices and 2838 edges, generated by Gengraph. The MIN-VC size is 660. A minimum vertex cover is indicated by the set of nodes with white color. . . . .	357

10.12	Family of regular graphs $D_s$ ( $s = 1, \dots, 5$ ) with small diameter generated by Gengraph. A minimum vertex cover is indicated by the set of nodes with white color. . . . .	358
10.13	Planar graph with Delaunay triangulations having 100 vertices and 285 edges, generated by Gengraph. The MIN-VC size is 68. A minimum vertex cover is indicated by the set of nodes with white color. . . . .	360
10.14	Planar graph with Delaunay triangulations having 1 000 vertices and 2 979 edges, generated by Gengraph. The MIN-VC size is 686. A minimum vertex cover is indicated by the set of nodes with white color. . . . .	362
10.15	Minimum ASes cover for 15 months of daily NLANR routing data. . . . .	364
10.16	QUBO hierarchical clustering in the Euclidean space of 6 000 objects. . . . .	371
10.17	The protein–protein interaction map of the H. Pylori produced by Balasundaram et al. [28]. . . . .	373
10.18	Image binarization weights used to define the neighborhood average assignment. . . . .	377
10.19	Image binarizations found by the one-pass heuristic applied to $f_{\alpha,\beta,\gamma}$ . Original image is a 24-bit bitmap $254 \times 300$ image of a child with a dark background. . . . .	381
10.20	Image binarizations found by the one-pass heuristic applied to $f_{\alpha,\beta,\gamma}$ . Original image is a 24-bit $1280 \times 755$ bitmap image of a project design of a house. . . . .	382
10.21	Image binarizations found by the one-pass heuristic applied to $f_{\alpha,\beta,\gamma}$ . Original image is a 24-bit $200 \times 199$ bitmap image of an x-ray. . . . .	383

# Chapter 1

## Introduction

*Quadratic Unconstrained Binary Optimization* (or QUBO in short) is a mathematical programming problem, whose objective is to find the minimum (or the maximum) value of a quadratic function with a finite number of binary variables.

Numerous hard combinatorial optimization problems arise naturally or can easily be reformulated as QUBO problems, including VLSI design (e.g., [32, 52, 55, 76, 152, 161, 217]), statistical mechanics (e.g., [32, 223, 198]), reliability theory and statistics (e.g., [190, 204, 205]), economics and finance (e.g., [132, 144, 166, 167, 175]), operations research and management science (e.g., [103, 118, 201, 232, 234, e.g.,]), manufacturing (e.g., [15, 25, 26, 91, 162]), data mining (e.g., [231]), vision (e.g. [69, 70, 159, 160, 203]), as well as numerous algorithmic problems of discrete mathematics (e.g., [101, 119, 131, 196, 200, 197, 222]).

This dissertation covers a wide range of problems related to QUBO, both from theoretical and practical perspectives. The various chapters of this work were written on a natural sequence in terms of how a person should study this problem.

The following three chapters introduce basic materials for the subsequent more elaborated topics related to QUBO. Chapter 2 presents various classic models that can be solved by QUBO. Chapter 3 describes a long list of QUBO problems used for testing the proposed algorithms. Chapter 4 introduces several definitions and basic concepts, including: persistency, first and second order derivatives, the concept of locotope, the implication graph, basic concepts about posiform minimization, rounding and derandomization, and best linear approximations to pseudo-Boolean functions.

The roof-duality approach of Hammer et al. [123] is an essential tool embedded in many of the algorithms that we propose. Chapter 5 reviews the *roof-duality*, and

presents new persistency results, by using linear algebra and network flows arguments.

Chapter 6 describes in detail how to find heuristic solutions to QUBO. The heuristic approaches suggested are one-pass or local optimization procedures. An innovative probabilistic approach based on continuous extensions of pseudo-Boolean functions is proposed and related to a more traditional approach based on rounding or switching steps.

Chapter 7 describes a preprocessing routine for QUBO. It involves various components, namely roof-duality persistency, decomposition and probing. The results obtained with this preprocessing method are impressive for several applications, resulting in the complete solution of many cases, such as vertex cover minimization of sparse or real world graphs, via minimization problems in VLSI or the 1-dimensional Ising chain models.

Chapter 8 describes several innovative ways to improve the roof-dual bound and its iterated version proposed by Boros and Hammer [51, 52]. The new approach is based on network flows and combinatorics and leads to substantially improved bounds for many benchmarks. Linear programming enhanced with certain specific families of cuts is also a very promising way to improve bounds and get a closer characterization of the integer polytope, which is especially fast for sparse QUBOs.

After having all the components (roof-duality, preprocessing, heuristics and bounds) then the next natural step is to use all of them to attempt to prove optimality for those more difficult QUBOs. Chapter 9 covers three exact approaches to solve QUBO problems: the first one is based on enumerative approaches, the second is a branch-and-bound solver based on the network flows model, and the third approach is based on Mixed Integer Programming (MIP). We shall demonstrate that in practice each one of the approaches has its own advantages compared to the others.

The final chapter covers three applications of QUBO in more detail. It starts by looking at the minimum vertex cover problem, extends to it the persistency and decomposition results for QUBO. A special preprocessing routine is proposed and tested among various classes of graphs. The combination of these data reduction techniques with those implemented for QUBO leads to a very efficient solver for many instances.

The next application that is considered is how to do clustering using QUBO. Two approaches are considered: one is a hierarchical clustering algorithm based on the graph balancing problem; the other applies to graphs only and is based on the greedy coloring (or partitioning) of the graph.

The main contributions of this dissertation are the following:

- If there are variables with integral optimal values for the relaxation of the classical linearization model for QUBO ([123]), then it is well known that there is an optimal solution to QUBO where these variables have the same values. This subset of variables with known optimal values are called persistencies. We will show that there is a *unique maximal* set of persistencies for the linearization model (see Chapter 5). This set can be determined in polynomial time by computing a maximum flow followed by the computation of the strong components of a capacitated network that has  $2n$  nodes, where  $n$  is the number of variables of the function. This procedure results in a  $O(n^3)$  time algorithm to determine all the persistencies for the linearization model.
- The identification of the above persistencies leads to a *decomposition* of the function (if any exists). This occurs in such a way that each component of the function can be optimized separately from the others, since the variables of each component do not participate in the other components (see Chapter 7).
- To find further persistencies, we propose two additional techniques: one is based on the second order derivatives of Hammer et al. [121] and its generalization (we called it *coordination*); the other one is a *probing* procedure on the two possible values of the variables, which when used in combination with a Boolean consensus algorithm, it is able to derive additional persistencies. These two preprocessing tools run also in polynomial time, and in practice can work remarkably well for certain classes of problems (e.g. minimum vertex cover of planar or power-law graphs, via minimization, 1-dimensional Ising chains, problems derived from vision) (see Chapter 7).

- We propose several families of one-pass (polynomial time) heuristics for QUBO. One family is based on probabilistic assumptions. The second group of heuristics is based on rounding methods and the third approach is based on best linear approximations. We also studied several families of local (search) optimization heuristics. The results indicate that our methods perform better than those considered in the literature (see Chapter 6).
- Roof-duality gives a very well known bound to the minimum of a quadratic pseudo-Boolean function. Boros and Hammer [51, 52] proposed an improved bound that they called as the iterated roof-duality bound. We improved further this bound by proposing two methods which are entirely combinatorial: one is called the *squeezed* iterated roof-duality; and the other one is called the *project-and-lift* iterated roof-duality method (see Chapter 8).
- Boros et al. [49] showed a hierarchy of bounds for QUBO. One end of the hierarchy corresponds to the roof-dual, and the other end corresponds to the actual optimum. If the roof-dual bound is not the optimum, then the next level of the hierarchy corresponds to the cubic-dual. The cubic-dual can be found by means of linear programming by adding to the standard linearization a set of triangle inequalities, whose number is cubic in the number of variables. We will show that this set can be reduced depending on the coefficients of the terms of the function. This leads to the possibility of computing a good approximation (or even the exact value) to the cubic-duals of larger QUBOs. This will be particularly advantageous for sparse QUBOs, i.e. problems which have a reduced number of quadratic terms per variable), which are common in real applications (see Chapter 8).
- After the cubic-dual, the next bound level in the hierarchy of Boros et al. [49] is characterized for the first time here, in terms of the generators description required to represent the function in the space of the cone of positive quadratic pseudo-Boolean functions (see Chapter 8).

- We propose the use of three exact approaches for QUBO: the first is an enumerative approach, the second is a branch-and-bound method based on roof-duality and network flows, and the third method is based on 0–1 linear optimization (see Chapter 9). For various applications we demonstrate that each approach has its advantages and disadvantages depending on the type of problem that they are trying to solve. We also compare the proposed methods to those state-of-the-art solvers for QUBO. In certain cases, the proposed methods are able to solve problems in a few seconds that no other (known) solvers can solve in several hours or more of computing time. The proposed methods could prove optimality for some publicly available unsolved problems (to our best knowledge), and in some cases they could find (substantially) better solutions than the meta-heuristics for QUBO (using about the same or less computing time).
- Along this dissertation we will cover many applications derived from engineering and social sciences: via minimization, 2d and 3D Ising models, 1D Ising chain models, image binarization, hierarchical clustering, greedy graph coloring, greedy graph partitioning, weighted MAX-2-SAT, MIN-VC, MAX-CLIQUE, MAX-CUT, weighted maximum stable set, minimum  $k$ -partition, etc. This wide practical view in combination with the various tools proposed to solve QUBO problems, will give a substantially better understanding about how to attack the algorithmic solution approach to any given QUBO problem.

Some basic definitions are introduced in the following section. The last section describes an example that is widely used in this dissertation to demonstrate the various techniques proposed.

## 1.1 Definitions and notation

Let  $\mathbb{R}$  denote the set of reals,  $\mathbb{Z}$  the set of integers, and let  $\mathbb{B} = \{0, 1\}$  and  $\mathbb{U} = [0, 1]$ . Further, let  $n$  denote a positive integer, and let  $\mathbf{V} = (1, \dots, n)$ . For a subset  $\mathbf{S} \subseteq \mathbf{V}$ ,

denote by  $\mathbf{1}^S \in \mathbb{B}^n$  its characteristic vector, i.e.

$$\mathbf{1}_j^S = \begin{cases} 1 & \text{if } j \in \mathbf{S} \\ 0 & \text{otherwise.} \end{cases}$$

Functions in  $n$  binary variables, denoted by  $x_1, x_2, \dots, x_n$ , are considered, and  $\mathbf{x} = (x_1, \dots, x_n)$  is used to denote a binary vector or to denote a vector of these variables.

The *complements* of the binary variables are denoted by  $\bar{x}_i \stackrel{\text{def}}{=} 1 - x_i$ . A variable or its complement is called a *literal*. Let  $\mathbf{L} \stackrel{\text{def}}{=} \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$  denote the set of literals.

In the sequel, the letters  $x, y$  and  $z$  are used to refer to variables,  $u, v$  and  $w$  are used to refer to literals, and bold face letters  $\mathbf{x}, \mathbf{y}, \mathbf{p}$ , etc., are used to denote vectors.

### 1.1.1 Pseudo-Boolean functions

*Pseudo-Boolean* functions are real valued mappings  $f : \mathbb{B}^n \mapsto \mathbb{R}$  from the set of binary  $n$ -vectors. It is well-known that such a mapping has a unique multi-linear polynomial expression in terms of the variables  $x_1, \dots, x_n$  (see [129, 131]):

$$f(x_1, \dots, x_n) = \sum_{S \subseteq \mathbf{V}} c_S \prod_{j \in S} x_j. \quad (1.1)$$

By convention, it is assumed that  $\prod_{j \in \emptyset} x_j = 1$ .

The size of the largest subset of variables  $S \subseteq V$  for which  $c_S \neq 0$  is called the *degree* of  $f$ , and is denoted by  $\deg(f)$ . A pseudo-Boolean function  $f$  is called *linear* (*quadratic*, *cubic*, *quartic*, etc.) if  $\deg(f) \leq 1$  (2, 3, 4, etc.)

The family of pseudo-Boolean functions of degree at most  $k$  is denoted by  $\mathfrak{F}_k$ :

$$\mathfrak{F}_k \stackrel{\text{def}}{=} \{(f : \mathbb{B}^n \mapsto \mathbb{R}) \mid \deg(f) \leq k\}.$$

The *size* of a pseudo-Boolean  $f$  function represented by (1.1) is the total number

of variable occurrences in it, i.e

$$\text{size}(f) \stackrel{\text{def}}{=} \sum_{S: S \subseteq \mathbf{V}, c_S \neq 0} |S|.$$

Pseudo-Boolean functions represented as *posiforms* are also considered, i.e. polynomial expressions in terms of all literals, having the form

$$\phi(\mathbf{x}) = \sum_{T \subseteq \mathbf{L}} a_T \prod_{u \in T} u, \quad (1.2)$$

where  $a_T \geq 0$  whenever  $T \neq \emptyset$ . If  $\{u, \bar{u}\} \subseteq T$  for some  $u \in \mathbf{L}$ , then  $\prod_{u \in T} u$  is identically zero over  $\mathbb{B}^n$  and, therefore, it is customary to assume that  $a_T = 0$ .

Similarly to the case of polynomial expressions, the size of the largest subset  $T \subseteq \mathbf{L}$  of literals for which  $a_T \neq 0$  is called the *degree* of the posiform  $\phi$ , and is denoted by  $\text{deg}(\phi)$ . The posiform  $\phi$  is called *linear* (*quadratic*, *cubic*, *quartic*, etc.) if  $\text{deg}(\phi) \leq 1$  (2, 3, 4, etc.)

The *size* of a posiform is the total number of occurrences in it, i.e.

$$\text{size}(\phi) \stackrel{\text{def}}{=} \sum_{T \subseteq \mathbf{L}: a_T \neq 0} |T|.$$

For the purpose of analyzing algorithms, the sum of the coefficients

$$A(\phi) \stackrel{\text{def}}{=} \sum_{T \neq \emptyset} a_T.$$

will also be needed.

It is simple to note that a posiform (1.2) uniquely determines a pseudo-Boolean function. However, the reverse is not true; a pseudo-Boolean function can have several distinct posiforms representing it. Furthermore, while it is computationally easy to generate a posiform expression from a polynomial expression (1.1), it might be computationally difficult to generate the unique polynomial expression corresponding to a given posiform. We denote the unique multilinear polynomial associated to the posiform  $\phi$  by  $f_\phi$ , and the set of all *posiforms of degree at most  $k$*  representing a pseudo-Boolean

function  $f$  by  $\mathcal{P}_k(f)$ . We also denote the constant term  $a_0$  of a posiform  $\phi$  by  $C(\phi)$ .

Consider the pseudo-Boolean minimization problem

$$\nu_S(f) = \min_{\mathbf{x} \in S} f(\mathbf{x}), \quad (1.3)$$

where  $f$  is a given pseudo-Boolean function and  $S \subseteq \mathbb{B}^n$ .

The focus of this dissertation is in minimization problems. However, pseudo-Boolean maximization problems are also considered:

$$\tau_S(f) = \max_{\mathbf{x} \in S} f(\mathbf{x}). \quad (1.4)$$

If  $S = \mathbb{B}^n$ , then the subscript  $S$  in  $\nu_S(f)$  or  $\tau_S(f)$  may be disregarded in the text. Thus,  $\nu(f)$  (or simply  $\nu_f$ ) and  $\tau(f)$  (or simply  $\tau_f$ ), respectively denote the minimum and maximum of the pseudo-Boolean function  $f$ .

Let  $\text{Argmin}_S(f)$  denote the subset of points belonging to  $S$ , which are minimizing solutions of problem (1.3). Similarly, let  $\text{Argmax}_S(f)$  denote the subset of points belonging to  $S$ , which are maximizing solutions of problem (1.4).

In the sequel, the letters  $f, g$  and  $h$  will usually denote pseudo-Boolean functions as well as their unique multi-linear polynomial expressions, while the greek letters  $\phi, \beta$  and  $\psi$  will denote posiforms.

### 1.1.2 Quadratic pseudo-Boolean functions

A pseudo-Boolean function  $f$  is called *quadratic* if its unique multi-linear polynomial is quadratic (i.e.  $\deg(f) \leq 2$ .) Specializing the notations given earlier, it is assumed that the quadratic pseudo-Boolean functions are represented either by their (unique) multi-linear quadratic polynomial expression

$$f(x_1, \dots, x_n) = c_0 + \sum_{i=1}^n c_i x_i + \sum_{1 \leq i < j \leq n} c_{ij} x_i x_j \quad (1.5)$$

or by a quadratic posiform

$$f(x_1, \dots, x_n) = a_0 + \sum_{u \in \mathbf{L}} a_u u + \sum_{u, v \in \mathbf{L}} a_{uv} uv, \quad (1.6)$$

where as before,  $L$  denotes the set of literals,  $a_u \geq 0$  and  $a_{uv} \geq 0$  for all  $u, v \in \mathbf{L}$ .

It should be remarked that among the posiforms representing a quadratic pseudo-Boolean function there may also exist some having degrees higher than two.

## 1.2 A generic illustrative example

During this dissertation, to illustrate certain concepts and algorithms for QUBO, we shall use the quadratic function,

$$\begin{aligned} & f_6(x_1, x_2, x_3, x_4, x_5, x_6) \\ = & 2x_1 + x_2 - 2x_3 - x_4 + x_5 - x_6 \\ & -x_1x_2 + 2x_1x_3 - 2x_1x_4 + 2x_1x_5 - x_1x_6 \\ & +x_2x_3 - x_2x_4 - x_2x_5 + x_2x_6 \\ & +2x_3x_4 - 2x_3x_5 + x_3x_6 \\ & +2x_4x_5 - x_4x_6 \\ & +2x_5x_6 \end{aligned}$$

of six variables.

The minimum and maximum values of  $f_6$  in  $\mathbb{B}^6$  are respectively  $\nu(f_6) = -4$  and  $\tau(f_6) = 5$ . The corresponding minima and maxima are

$$\text{Argmin}_{\mathbb{B}^6}(f) = \{(1, 0, 0, 1, 0, 1), (1, 1, 0, 1, 0, 1)\}$$

and

$$\text{Argmax}_{\mathbb{B}^6}(f) = \{(1, 0, 0, 0, 1, 0), (1, 0, 0, 0, 1, 1), (1, 1, 0, 0, 1, 1), (1, 1, 1, 0, 1, 1)\}.$$

## Chapter 2

### Classic Combinatorial Optimization Models

The purpose of this chapter is to present a set of combinatorial optimization problems and show how QUBO arises to solve many of them, either naturally or by reformulation. Throughout this dissertation we will show additional families of problems that could be solved by QUBO. This chapter describes the most common models that appear in the literature.

#### 2.1 Pseudo-Boolean optimization

Rosenberg [213] shown that the optimization of a pseudo-Boolean  $f$  with  $\deg(f) \geq 3$  can be reduced in polynomial time to the optimization of a quadratic pseudo-Boolean function with additional variables. The basic idea consists in replacing a product  $x_i x_j$  appearing in a non-quadratic term of  $f$ , by a new binary variable  $y_{ij}$ .

**Lemma 2.1.** *Let  $x_i, x_j, y_{ij} \in \mathbb{B}$ . Then the following equivalences hold:*

$$y_{ij} = x_i x_j \quad \text{iff} \quad (a + b + c) y_{ij} - (a + c) x_i y_{ij} - (b + c) x_j y_{ij} + c x_i x_j = 0,$$

*and*

$$y_{ij} \neq x_i x_j \quad \text{iff} \quad (a + b + c) y_{ij} - (a + c) x_i y_{ij} - (b + c) x_j y_{ij} + c x_i x_j > 0,$$

*for any positive values  $a, b$  and  $c$ .*

The above result can be used to transform a pseudo-Boolean optimization problem into a QUBO. The transformation is polynomial time in size and time. The dimension of the problem may increase substantially, but all the engineering tools available for QUBO could be used to solve those general pseudo-Boolean optimization problems.

An interesting new approach, for the same purpose of the above idea, as been

introduced by Buchheim and Rinaldi [72]. This method also requires the introduction of additional variables or constraints, but compared to the previous method, this approach produces smaller optimization problems, typically making it more tractable from a practical point of view.

## 2.2 Graph theory

Let  $G = (V, E)$  be an undirected graph, with vertex set  $V$  and edge set  $E$ . The *complement*  $\overline{G}$  of  $G$  is the graph  $\overline{G} = (V, \binom{V}{2} \setminus E)$ , i.e. the graph having the same vertex set  $V$  and having as edge set the complement of the edge set  $E$  of  $G$ . Sometimes, we associate to each edge of the graph a real weight  $w_e \in \mathbb{R}, e \in E$ . We may also associate to a vertex, a real weight  $c_j \in \mathbb{R}, j \in V$ .

### 2.2.1 Maximum clique

A *clique* of the graph  $G = (V, E)$  is a set of pairwise adjacent vertices. Naturally, a *maximum clique* (or MAX-Clique in short) is a clique of maximum cardinality. The size of a maximum clique is commonly called the *clique number* of  $G$ . We shall denote it as  $\theta(G)$ .

A subset  $S \subseteq V$  is called *independent* (or *stable*), if no edge of  $G$  has both endpoints in  $S$ . A *maximum independent set* is a largest cardinality independent set; the cardinality of a maximum independent set will be denoted by  $\alpha(G)$  and called the *stability number* of  $G$ .

An independent set in a graph  $G$  is a clique in the complement graph  $\overline{G}$ . Thus,  $\alpha(G) = \theta(\overline{G})$ . Moreover, since the complement graph  $\overline{G}$  has a polynomial size representation of the input size of  $G$ , and since it can be obtained in a polynomial time function of the size of  $G$ , then there is a strong equivalence between the maximum clique and the maximum independent set problems. If a solution to one of the problems is available, a solution to the other problem can be obtained immediately.

**Theorem 2.1.** *The cardinality of a maximum independent set of  $G = (V, E)$  is equal*

to the optimum value of the QUBO

$$\alpha(G) = \max_{\mathbf{x} \in \mathbb{B}^V} \left( \sum_{i \in V} x_i - \sum_{(i,j) \in E} (1 + \epsilon_{(i,j)}) x_i x_j \right), \quad (2.1)$$

where  $\epsilon_{(i,j)} (\geq 0)$  are arbitrary nonnegative reals for all  $(i,j) \in E$ . Furthermore, if  $\mathbf{x}^* = \mathbf{1}^S$  is a maximizing binary vector of (2.1), then a maximum cardinality set  $S^*$  ( $\subseteq S$ ) of  $G$  can be obtained in  $O(|E|)$  time.

*Proof.* Let us associate to  $G$  the quadratic pseudo-Boolean function  $f(x_1, \dots, x_n) = \sum_{i \in V} x_i - \sum_{(i,j) \in E} (1 + \epsilon_{(i,j)}) x_i x_j$ . Let further  $f(x_1^*, \dots, x_n^*)$  be a maximum of  $f$ , and let  $S^*$  be the set of vertices  $j \in V$  for which  $x_j^* = 1$ . We will show next that the size of a maximum independent set is  $f(x_1^*, \dots, x_n^*)$ , and the set  $S^*$  can be reduced to a maximum independent set  $\widehat{S} \subseteq S^*$  in  $O(|E|)$  time.

If  $S^*$  is an independent set, it is clearly maximal, and since  $x_i^* x_j^* = 0$  for every  $(i,j) \in E$ , the problem is solved. Let us assume that  $S^*$  is not a stable set, and let  $h$  and  $k$  be two adjacent vertices with  $x_h^* = x_k^* = 1$ . Let us consider now the vector  $(x_1^{**}, \dots, x_n^{**})$  defined by

$$x_l^{**} = \begin{cases} x_l^* & \text{if } l \neq k, \\ 0 & \text{if } l = k. \end{cases}$$

If  $N_k^*$  is the neighborhood of  $k$  in the set  $V \setminus S^*$ , then clearly

$$f(x_1^{**}, \dots, x_n^{**}) = f(x_1^*, \dots, x_n^*) - 1 + \sum_{t \in N_k^*} (1 + \epsilon_{(i,j)}) x_t^*,$$

and since  $h \in N_k^*$ , the set  $N_k^*$  is not empty, and therefore

$$f(x_1^{**}, \dots, x_n^{**}) \geq f(x_1^*, \dots, x_n^*).$$

On the other hand, from the maximality of  $f$  in  $(x_1^*, \dots, x_n^*)$  it follows that

$$f(x_1^{**}, \dots, x_n^{**}) \leq f(x_1^*, \dots, x_n^*).$$

By repeating this transformation several times (at most  $|V \setminus S^*|$  times), the set  $S^*$  will be eventually transformed to an independent set  $\widehat{S}$  of the graph  $G$ , which must have a maximum size.  $\square$

From the previous proof, it can be easily seen that when all coefficients  $\epsilon_e$ ,  $e \in E$ , are restricted to be positive, then a set  $S$  associated to the characteristic vector of an optimal solution of (2.1), is a maximum cardinality independent set of  $G$ .

### 2.2.2 Minimum vertex cover

The complement  $V \setminus S$  of an independent set  $S$  of  $G$  is called a *vertex cover* of the graph. Let us denote the size of the smallest vertex cover of  $G$  as  $\tau(G) = |V| - \alpha(G)$ . Then using (2.1) it can be shown that

$$\tau(G) = \min_{\mathbf{x} \in \mathbb{B}^V} \left( \sum_{i \in V} x_i + \sum_{(i,j) \in E} (1 + \epsilon_{(i,j)}) \bar{x}_i \bar{x}_j \right). \quad (2.2)$$

The knowledge of any one of the three numbers  $\theta(\overline{G})$ ,  $\alpha(G)$  or  $\tau(G)$  implies that the other two values can be immediately determined. In general finding any of these numbers is a NP-hard optimization problem ([104]). It is important to note that even for planar graphs it is known that solving the minimum vertex cover problem is NP-hard ([105]).

The concepts introduced above can be extended analogously to the weighted variants of these problems. For instance, the *weighted stability number* of a graph  $G = (V, E)$  with nonnegative costs  $\mathbf{c}^V$ , associated to the vertices of the graph, is equal to the optimal solution of problem

$$\max_{\mathbf{x} \in \mathbb{B}^V} \left( \sum_{i \in V} c_i x_i - \sum_{(i,j) \in E} (\max(c_i, c_j) + \epsilon_{(i,j)}) x_i x_j \right). \quad (2.3)$$

In a similar way, the above formulations can also be extended to hypergraphs. For instance, given a hypergraph  $\mathcal{H} \subseteq 2^V$ , a subset  $S \subseteq V$  is called a *vertex cover* of  $\mathcal{H}$  (known also as a *hitting set*) if  $S \cap H \neq \emptyset$  for all hyperedges  $H \in \mathcal{H}$ . If the size of the

smallest such set is denoted by  $\tau(\mathcal{H})$ , then

$$\tau(\mathcal{H}) = \min_{\mathbf{x} \in \mathbb{B}^V} \left( \sum_{i \in V} x_i + \sum_{H \in \mathcal{H}} (1 + \epsilon_H) \prod_{i \in H} \bar{x}_i \right). \quad (2.4)$$

This optimization problem can also be viewed as a pseudo-Boolean formulation of the *set covering* problem over the transposed hypergraph. The formulation of the hitting set problem requires optimization of non-quadratic pseudo-Boolean functions, but in theory, as shown in subsection 2.1, a degree reduction technique can be used to get a quadratic problem with the same optimal solution as the original problem with a higher degree.

### 2.2.3 Maximum cut

A *cut* of the graph  $G = (V, E)$  is defined by a partition of the vertex set  $V$  into two subsets  $S$  and  $\bar{S}$ , and consists of the set of edges with exactly one endpoint in  $S$  and another in  $\bar{S}$ . We shall denote the cut defined by  $S$  as  $(S, \bar{S})$ .

The *maximum cut* (or MAX-CUT in short) problem is to find a cut  $(S, \bar{S})$  with the largest cardinality. If  $\mathbf{x} = \mathbf{1}^S$  is the characteristic vector representing  $S$ , then it can be shown that

$$\max_{S \subseteq V} |(S, \bar{S})| = \max_{\mathbf{x} \in \mathbb{B}^V} \left( \sum_{(i,j) \in E} (x_i \bar{x}_j + \bar{x}_i x_j) \right). \quad (2.5)$$

The *weighted MAX-CUT* problem in a graph  $G = (V, E)$  with weights  $\mathbf{w}^E$  is to find a cut  $(S, \bar{S})$  for which the sum of the weights of the corresponding edges is maximum. If the total weight of a cut is denoted by  $W(S, \bar{S})$ , then the weighted maximum cut can be found by solving the problem

$$\max_{S \subseteq V} W(S, \bar{S}) = \max_{\mathbf{x} \in \mathbb{B}^V} \left( \sum_{(i,j) \in E} w_{ij} (x_i \bar{x}_j + \bar{x}_i x_j) \right). \quad (2.6)$$

It should be remarked that any optimal solution  $\mathbf{x} = (x_1^*, \dots, x_n^*)$  of problem (2.6) has a complementary solution  $(\bar{x}_1^*, \dots, \bar{x}_n^*)$ .

### 2.3 Maximum satisfiability

The *maximum satisfiability problem* (or MAX-SAT in short) is a popular subject in applied mathematics and computer science, and it has a natural pseudo-Boolean formulation. The input of a MAX-SAT instance (usually called a *formula*) consists of a family  $\mathcal{C}$  of subsets  $C \subseteq \mathbf{L}$  of literals, called *clauses*. A binary assignment  $\mathbf{x} \in \mathbb{B}^V$  *satisfies* a clause  $C$ , if at least one literal in  $C$  takes value 1 (true) for this assignment. The maximum satisfiability problem consists in finding a binary assignment satisfying the maximum number of clauses in  $\mathcal{C}$ . It is easy to see that a clause  $C$  is satisfied by  $\mathbf{x} \in \mathbb{B}^V$  if and only if  $\prod_{u \in C} \bar{u} = 0$ . Thus, MAX-SAT is equivalent to the problem

$$\max_{x \in \mathbb{B}^V} \sum_{C \in \mathcal{C}} \left( 1 - \prod_{u \in C} \bar{u} \right).$$

In the *weighted* maximum satisfiability problem there is also a nonnegative weight  $a_C$  associated with each clause  $C \in \mathcal{C}$ , and the objective is to maximize the total weight of satisfied clauses:

$$\max_{x \in \mathbb{B}^V} \sum_{C \in \mathcal{C}} a_C \left( 1 - \prod_{u \in C} \bar{u} \right).$$

If the clauses  $C$  have at most  $k$  literals, then the (weighted) MAX-SAT problem is called the MAX- $k$ -SAT problem. In particular, the weighted MAX-2-SAT problem can be formulated as the optimization of a special quadratic *negaform*<sup>1</sup>:

$$\tau(\psi) = \max_{x \in \mathbb{B}^V} \psi = \max_{x \in \mathbb{B}^V} \left( \sum_{\{u\} \in \mathcal{C}} a_{\{u\}} (1 - \bar{u}) + \sum_{\{u,v\} \in \mathcal{C}} a_{\{u,v\}} (1 - \bar{u}\bar{v}) \right).$$

If the previous negaform is denote by  $\psi$ , then we consider the minimum of the quadratic posiform  $\phi = A(-\psi) - \psi$ , i.e.

$$\tau(\phi) = \min_{x \in \mathbb{B}^V} \phi = \sum_{\{u\} \in \mathcal{C}} a_{\{u\}} \bar{u} + \sum_{\{u,v\} \in \mathcal{C}} a_{\{u,v\}} \bar{u}\bar{v}. \quad (2.7)$$

---

<sup>1</sup>In a similar way to a posiform, a negaform of a pseudo-Boolean function  $f(x_1, \dots, x_n)$  is defined as a polynomial  $g(x_1, \bar{x}_1, \dots, x_n, \bar{x}_n)$ , taking the same values as  $f$  in every binary  $n$ -vector, and having the property that all its coefficients (with the possible exception of the free term) are non-positive.

Since for any assignment of the formula  $\mathcal{C}$ ,  $\tau(\psi)$  is the maximum number of true clauses and  $\nu(\phi)$  is the minimum number of false clauses, then it is simple to notice that  $\tau(\psi) + \nu(\phi) = A(-\psi) = A(\phi)$ .

## Chapter 3

### Test Beds

A solution method that works for a particular class of problems, may not work for another. This is particularly important if the problem that has to be solved is a “hard” optimization problem, whose optimal solution is difficult to be found or certified. Therefore, it is important to have a set of benchmarks, which would provide a way to compare the quality of solutions and the efficiency of the proposed algorithms.

A set of QUBO benchmarks is listed and described in this chapter. Each test problem is described by enhancing its basic structural properties, and by showing its origin, best known solution values, and generation parameters (when available).

Since we were dealing with the QUBO, it was natural to search for test problems, previously studied by other researchers. Many of these publicly available problems have heuristic solutions that are not known to be optimal. The best known solutions of the “large” benchmarks were found using meta-heuristics. Every benchmark problem has been investigated by several authors, and has been tested using different methods. The high degree of sophistication put into these algorithms, gives a high degree of confidence to the fact that these best known solutions are at least very close to the optimum.

Each benchmark with a known optimal solution was included in a particular set of problems, which we called the *optimal dataset* for QUBO. The optimal dataset gives to the researchers, a way of measuring with accuracy, the quality of solutions returned by their proposed algorithms.

By providing information about this set and about the best known solutions, we hope to motivate the scientific community to make contributions to enlarge the number of benchmark problems which have provably optimal solutions, or simply to improve the best known solutions of the problems.



Several QUBO applications representative of different classes of problems were introduced in Chapter 2. To test the algorithm’s “sensitivity”, it is desirable to have test cases representing different classes of problems.

### 3.1 Benchmarks with prescribed density

The class of benchmarks with prescribed density consists of quadratic multilinear polynomials, which are randomly generated with a predetermined expected density<sup>1</sup>  $d$ , a uniform distribution of values of the linear coefficients in an interval  $[c^-, c^+]$ , and a uniform distribution of values of the quadratic coefficients in an interval  $[q^-, q^+]$ . The constant term of the function is zero. Since a quadratic term’s probability to have a nonzero coefficient is  $d$ , the expected number of quadratic terms which include a specific variable and have a nonzero coefficient is  $(n - 1)d$ .

When reporting computational results about QUBO, the standard practice followed by many authors was to generate random instances with prescribed density. An inconvenience of this approach was that the information was insufficient to allow the replication of test problems by other researchers. In order to overcome these difficulties, Pardalos and Rodgers (P&R) [195] have introduced a test problem generator for QUBO. In their notation, the QUBO is formulated as  $\max \{ \mathbf{x}^T \mathbf{Q} \mathbf{x} : \mathbf{x} \in \mathbb{B}^n \}$  where  $q_{ji} = q_{ij} = c_{ij}/2$ ,  $i \neq j$ ,  $1 \leq i < j \leq n$  and  $q_{ii} = c_i$ ,  $1 \leq i \leq n$ . The “standard” parameters in this approach include:  $n$ ,  $\bar{d}$ ,  $c^-$ ,  $c^+$ ,  $q^-$ ,  $q^+$ , and a seed to initialize a random number generator. The P&R routine generates symmetric integer matrices  $\mathbf{Q}$ , such that the expected density is  $\bar{d}$ , the distribution of the coefficients  $q_{ii}$  ( $1 \leq i \leq n$ ) is *discrete uniform* in  $[c^-, c^+]$ , and the distribution of the nonzero coefficients  $q_{ij} = q_{ji}$  ( $1 \leq i < j \leq n$ ) is *discrete uniform* in  $[q^-, q^+]$ .

Before presenting the list of problems considered in this section, we introduce some parameters, which are strongly related to the quality of solutions returned by the traditional algorithms proposed to solve QUBO problems.

---

<sup>1</sup>The density  $d$  of a quadratic pseudo-Boolean function represented by the polynomial expression (1.5) is defined as the number of nonzero coefficients  $c_{ij}$  ( $1 \leq i < j \leq n$ ) divided by  $\binom{n}{2}$ .

Let

$$\begin{aligned}
 P &\stackrel{\text{def}}{=} C^+ + Q^+ & \text{where } C^+ &\stackrel{\text{def}}{=} \sum_{\substack{j \in \mathbf{V} \\ c_j > 0}} c_j & \text{and } Q^+ &\stackrel{\text{def}}{=} \sum_{\substack{1 \leq i < j \leq n \\ c_{ij} > 0}} c_{ij}, \\
 N &\stackrel{\text{def}}{=} C^- + Q^- & \text{where } C^- &\stackrel{\text{def}}{=} \sum_{\substack{j \in \mathbf{V} \\ c_j < 0}} c_j & \text{and } Q^- &\stackrel{\text{def}}{=} \sum_{\substack{1 \leq i < j \leq n \\ c_{ij} > 0}} c_{ij},
 \end{aligned}$$

i.e.  $P$  (respectively,  $N$ ) is the sum of the positive (respectively, negative) coefficients of the quadratic pseudo-Boolean function  $f$  given as (1.5)

Let  $\rho$  be proportion of the sum positive coefficients in the total weight associated to the set of coefficients of  $f$ , i.e.

$$\rho \stackrel{\text{def}}{=} \frac{P}{P-N}. \quad (3.1)$$

$f$  is said to have  $p$  *diagonal dominance* if

$$p \stackrel{\text{def}}{=} \frac{C^+ - C^-}{Q^+ - Q^-},$$

or equivalently

$$p = \frac{\bar{c}}{\bar{q}(n-1)},$$

where  $\bar{c}$  is the average of the absolute values of linear coefficients, and  $\bar{q}$  is the average of the absolute values of the quadratic coefficients.

Since problems become easier with growing diagonal dominance  $p$  ([33, 44, 80]), this parameter has been used to characterize the complexity of finding an optimal solution to a given QUBO. It is interesting to notice that another parameter ( $\rho$ ) is highly relevant in the study of heuristics for the solution of QUBO. The relationship of the quality of solutions given by several heuristics on the parameter  $\rho$  will be discussed on Chapter 6.

Using the above definitions, it is not difficult to prove that the following results hold for all problems generated with P&R.

**Lemma 3.1.** *If a QUBO is randomly generated with  $n$  variables, expected density  $\bar{d}$ ,*

diagonal coefficients with discrete uniform distribution in  $[c^-, c^+]$ ,  $c^- \in \mathbb{Z}$ ,  $c^+ \in \mathbb{Z}$ ,  
off-diagonal coefficients with discrete uniform distribution in  $[q^-, q^+]$ ,  $q^- \in \mathbb{Z}$ ,  $q^+ \in \mathbb{Z}$ ,  
all coefficients mutually independent, then

$$\begin{aligned}
E[\bar{c}] &= \begin{cases} \frac{c^-+c^+}{2}, & c^- \in \mathbb{Z}^+ \\ -\frac{c^-+c^+}{2}, & c^+ \in \mathbb{Z}^- \\ \frac{c^+(c^++1)+c^-(c^- -1)}{2(c^+-c^-+1)}, & c^- \in \mathbb{Z}_0^-, c^+ \in \mathbb{Z}_0^+ \end{cases} \\
E[\bar{q}] &= \begin{cases} \frac{q^-+q^+}{2}\bar{d}, & q^- \in \mathbb{Z}^+ \\ -\frac{q^-+q^+}{2}\bar{d}, & q^+ \in \mathbb{Z}^- \\ \frac{q^+(q^++1)+q^-(q^- -1)}{2(q^+-q^-+1)}\bar{d}, & q^- \in \mathbb{Z}_0^-, q^+ \in \mathbb{Z}_0^+ \end{cases} \\
E[\rho] &= \frac{E[\bar{c}]}{E[\bar{q}](n-1)}.
\end{aligned}$$

**Lemma 3.2.** *If a QUBO is randomly generated with:  $n$  variables, expected density  $\bar{d}$ , diagonal coefficients with discrete uniform distribution in  $[c^-, c^+]$ ,  $c^- \in \mathbb{Z}_0^-$ ,  $c^+ \in \mathbb{Z}_0^+$ , off-diagonal coefficients with discrete uniform distribution in  $[q^-, q^+]$ ,  $q^- \in \mathbb{Z}_0^-$ ,  $q^+ \in \mathbb{Z}_0^+$ , all coefficients mutually independent, then*

i)

$$E[\rho] = \frac{\frac{c^+(c^++1)}{2(c^+-c^-+1)} + \frac{q^+(q^++1)}{2(q^+-q^-+1)}(n-1)\bar{d}}{E[\bar{c}] + E[\bar{q}](n-1)};$$

ii) and as  $n\bar{d} \rightarrow \infty$ ,

$$\rho \rightarrow \bar{\rho} \equiv \frac{1}{1 + \frac{q^-(q^- -1)}{q^+(q^++1)}}. \quad (3.2)$$

In the following subsections, 143 publicly available benchmark datasets are described, as well as the 4900 randomly generated test problems used in evaluating the efficiency of the different variants of the proposed algorithms.

### 3.1.1 Benchmark families

In the past literature about QUBO, 143 test problems (see Table 3.2) were frequently used (see e.g., [18, 37, 107, 108, 177, 178, 189]) for testing QUBO algorithms.

The basic generation parameters of the sub-families containing these problems can

be seen in Table 3.2, while the individual characteristics of the problems appear in Tables A.1 to A.5 of the Appendix.

Table 3.2: QUBO benchmarks with prescribed density.

Family	Sub-Family	Variables ( $n$ )	Number Problems	Density ( $\bar{d}$ %)	Linear Coef.		Quadr. Coef.		$\bar{p}$
					( $c^-$ )	( $c^+$ )	( $q^-$ )	( $q^+$ )	
GKA	<i>A</i>	30 to 100	8	6.5 to 50	-100	100	-100	100	0.5
	<i>B</i>	20 to 125	10	100	0	63	-100	0	0
	<i>C</i>	40 to 100	7	10 to 80	-50	50	-100	100	0.5
	<i>D</i>	100	10	6.5 to 50	-50	50	-75	75	0.5
	<i>E</i>	200	5	10 to 50	-50	50	-100	100	0.5
F	$F_1$	500	5	10 to 100	-75	75	-50	50	0.5
	$F_2$	500	5	10 to 100	0	100	-50	0	0
G	$G_1$	1 000	10	10 to 100	-75	75	-50	50	0.5
	$G_2$	1 000	5	10 to 100	0	100	-50	0	0
Beasley	B-50	50	10	10	-100	100	-100	100	0.5
	B-100	100	10	10	-100	100	-100	100	0.5
	B-250	250	10	10	-100	100	-100	100	0.5
	B-500	500	10	10	-100	100	-100	100	0.5
	B-1000	1 000	10	10	-100	100	-100	100	0.5
	B-2500	2 500	10	10	-100	100	-100	100	0.5
Palubeckis	P-3000	3 000	5	50 to 100	-100	100	-100	100	0.5
	P-4000	4 000	5	50 to 100	-100	100	-100	100	0.5
	P-5000	5 000	5	50 to 100	-100	100	-100	100	0.5
	P-6000	6 000	3	50 to 100	-100	100	-100	100	0.5

The test problems in the families *A*, *B*, *D*, *E* and  $F_1$ , were proposed by Glover et al. [108]. They were generated using the P&R routine ([195]), with the basic parameters shown in Table 3.2. The corresponding best known solutions, and the information on which solutions are known to be optimal, can be seen in Table A.2 of the Appendix.

The group of problem in the *B* family define *submodular* quadratic pseudo-Boolean functions, 100% dense problems with the number of variables ranging from 20 to 125. The seven small sized datasets in group *C* were proposed by Pardalos and Rodgers [195], as being the most “challenging” problems that they could solve optimally with their enumerative procedure. Later, these problems were adopted by Glover et al. [108] and Beasley [37].

The 10 problems of the group  $G_1$  were proposed by Glover et al. [109]. The P&R routine was used to generate these problems, which have 1 000 variables, and various densities ranging from 10% to 100%. The details of the  $G_1$  problems can be seen in Table A.3 of the Appendix. Later, these test problems were used as benchmarks in

several studies of QUBO ([18, 177, 178]).

The datasets in the groups  $F_2$  and  $G_2$  were proposed by Kochenberger et al. [158]. They define *submodular* quadratic pseudo-Boolean functions, respectively having 500 and 1000 variables, and densities varying from 10% to 100%. The general characteristics of these problems can be seen in Table A.4. These problems were downloaded from the Hearin Center for Enterprise Science website [2].

For the benefit of those who may want to replicate the experiments, we would like to make the following remarks. Some of the recently published papers ([37, 107, 108, 178]) report on heuristic solutions for the *maximization* problem. However, it should be kept in mind that the  $GKA_c$  datasets were proposed initially by Pardalos and Rodgers [195] as minimization problems. Subsequently, these problems were changed by Glover et al. [108] to the maximization of the negatives of the functions minimized in [195]. Similarly, the  $GKA_{a,b,d,e}$  and  $F_1$  datasets were created by using the generator of [195], and modified afterwards to maximization problems, as explained above. Since then, the literature (e.g., [37, 107, 178]) reporting on these datasets, as well as the present study, followed the format of the Glover et al. [108] problems.

The *Beasley* family of test problems for QUBO consists of a set of 60 randomly generated test problems, where the number of variables  $n$  varies from 50 to 2 500; 10 problems for each value of  $n$ . They were proposed by Beasley [37], and since then several other studies of QUBO (e.g., [18, 107, 178]) have reported heuristic values on the maximum value of the corresponding quadratic pseudo-Boolean functions. The basic parameters of the Beasley problems can be seen in Table 3.2, and the information on best known values is listed in Table A.1 of the Appendix. The Beasley datasets were downloaded from the *OR-Library* website [38] (see also [36, 25])

All the Beasley test problems are reported to be 10% dense, but the definition of expected density in this family (given there as the probability that the coefficient of a *linear* or quadratic term is nonzero) differs from the definition used in this study, making these problems somewhat more difficult.

More recently, Palubeckis [187, 189] introduced 18 new test problems generated

from a similar routine to the one provided by P&R. They use the same random number generator, but they differ in the way how the seeds provided to the random number generator, are handled. In the P&R case, two seeds are used, one for decisions on density and the other to generate the function's coefficients. In the Palubeckis case only one seed is used for both cases.

The generation parameters of the Palubeckis problems can be seen in Table 3.2, and the information on best known values is listed in Table A.5 of the Appendix. The sizes of these problems vary from 3 000 to 6 000 in the number of variables, and from 50% to 100% in density.

Let us remark the following facts on the benchmark problems for QUBO:

- Only 69 out of the 143 problems in the benchmark families have a solution which is provably optimal;
- All the best known solutions (including optimal ones) refer to the *maximum* values of the corresponding functions;
- Except for the *submodular* sub-families  $B$ ,  $F_2$  and  $G_2$  (with  $\bar{\rho} = 0.0$ ), all the other sub-families have a  $\bar{\rho}$  value of 0.5. This fact implies that the *expected* sum of all coefficients of the functions in these particular datasets is zero.

### 3.1.2 Randomly generated test problems

In order to increase the number of test problems with prescribed fixed density, 3 728 additional datasets were created by using the P&R generator. Four groups of problems are considered:

- *Small* (see Table 3.3) – This family contains 240 QUBO minimization problems, whose number of variables ranges from 25 to 100 (in steps of 25). The expected densities vary from 20% to 100% (in steps of 20%), and the expected values of  $\rho$  vary from 0.40 to 0.85 (in steps of 0.15). There are three instances ( $k = 1, 2, 3$ ) for each set of parameters. The starting seed for the random number generator is the value of the expression  $k + 2^{10} \times \lfloor 100\bar{d} \rfloor + 2^{17}n$ . An optimal solution to the

minimum of the associated quadratic pseudo-Boolean function is known for 213 “small” datasets, whose values can be seen in Table A.6 of the Appendix.

Table 3.3: Characteristics of QUBO problems in the *Small* family.

Sub-Family	Variables ( $n$ )	Number Problems	Density ( $\bar{d}$ %)	Linear Coef.		Quadr. Coef.		$\bar{\rho}$
				( $c^-$ )	( $c^+$ )	( $q^-$ )	( $q^+$ )	
S-25-0.40	25	15	20 to 100	-50	50	-61	50	0.40
S-50-0.40	50	15	20 to 100	-50	50	-61	50	0.40
S-75-0.40	75	15	20 to 100	-50	50	-61	50	0.40
S-100-0.40	100	15	20 to 100	-50	50	-61	50	0.40
S-25-0.55	25	15	20 to 100	-50	50	-45	50	0.55
S-50-0.55	50	15	20 to 100	-50	50	-45	50	0.55
S-75-0.55	75	15	20 to 100	-50	50	-45	50	0.55
S-100-0.55	100	15	20 to 100	-50	50	-45	50	0.55
S-25-0.70	25	15	20 to 100	-50	50	-32	50	0.70
S-50-0.70	50	15	20 to 100	-50	50	-32	50	0.70
S-75-0.70	75	15	20 to 100	-50	50	-32	50	0.70
S-100-0.70	100	15	20 to 100	-50	50	-32	50	0.70
S-25-0.85	25	15	20 to 100	-50	50	-20	50	0.85
S-50-0.85	50	15	20 to 100	-50	50	-20	50	0.85
S-75-0.85	75	15	20 to 100	-50	50	-20	50	0.85
S-100-0.85	100	15	20 to 100	-50	50	-20	50	0.85

- *Medium* (see Table 3.4) – This family of QUBO maximization problems was introduced in Boros et al. [62]. The number of variables in these problems ranges from 500 to 2000 (in steps of 500). Expected density values range from 20% to 100% (in steps of 20%). The lower and upper bounds of the coefficients of linear terms were fixed to  $-50$  and  $+50$  respectively. The upper bound  $q^+$  of the coefficients of quadratic terms was fixed to  $+50$ , while the lower bound  $q^-$  of these coefficients was left to be determined for each dataset by the values of the parameter  $\bar{\rho}$ . In fact, a *continuous* uniform distribution was assumed, and therefore the expression (3.2) of Lemma 3.2 becomes

$$\bar{\rho} \equiv \frac{1}{1 + \left(\frac{q^-}{q^+}\right)^2},$$

and thus

$$q^- = - \left[ 50 \sqrt{\frac{1 - \bar{\rho}}{\bar{\rho}}} \right]$$

was used to determine the value of  $q^-$ . The values of  $\bar{\rho}$  range from 0.02 to 0.58

in steps of 0.02. We have generated a total of 2 900 problems representing five instances ( $k = 1, \dots, 5$ ) for each combination of the four values of the parameter  $n$ , five values of the parameter  $\bar{d}$  and 29 values of the parameter  $\bar{\rho}$ . The starting seed for the random number generator is the value of the expression  $\lfloor \frac{49}{20}n \rfloor + \lfloor 1225\bar{d} \rfloor + \lfloor 250\bar{\rho} \rfloor + k - 1475$ .

Table 3.4: Characteristics of QUBO problems in the *Medium* family.

Sub-Family	Variables ( $n$ )	Number Problems	Density ( $\bar{d}$ %)	Linear Coef.		Quadr. Coef.		$\bar{\rho}$
				( $c^-$ )	( $c^+$ )	( $q^-$ )	( $q^+$ )	
M-0.02	500 to 2 000	100	20 to 100	-50	50	-350	50	0.02
M-0.04	500 to 2 000	100	20 to 100	-50	50	-244	50	0.04
M-0.06	500 to 2 000	100	20 to 100	-50	50	-197	50	0.06
M-0.08	500 to 2 000	100	20 to 100	-50	50	-169	50	0.08
M-0.10	500 to 2 000	100	20 to 100	-50	50	-150	50	0.10
M-0.12	500 to 2 000	100	20 to 100	-50	50	-135	50	0.12
M-0.14	500 to 2 000	100	20 to 100	-50	50	-123	50	0.14
M-0.16	500 to 2 000	100	20 to 100	-50	50	-114	50	0.16
M-0.18	500 to 2 000	100	20 to 100	-50	50	-106	50	0.18
M-0.20	500 to 2 000	100	20 to 100	-50	50	-100	50	0.20
M-0.22	500 to 2 000	100	20 to 100	-50	50	-94	50	0.22
M-0.24	500 to 2 000	100	20 to 100	-50	50	-88	50	0.24
M-0.26	500 to 2 000	100	20 to 100	-50	50	-84	50	0.26
M-0.28	500 to 2 000	100	20 to 100	-50	50	-80	50	0.28
M-0.30	500 to 2 000	100	20 to 100	-50	50	-76	50	0.30
M-0.32	500 to 2 000	100	20 to 100	-50	50	-72	50	0.32
M-0.34	500 to 2 000	100	20 to 100	-50	50	-69	50	0.34
M-0.36	500 to 2 000	100	20 to 100	-50	50	-66	50	0.36
M-0.38	500 to 2 000	100	20 to 100	-50	50	-63	50	0.38
M-0.40	500 to 2 000	100	20 to 100	-50	50	-61	50	0.40
M-0.42	500 to 2 000	100	20 to 100	-50	50	-58	50	0.42
M-0.44	500 to 2 000	100	20 to 100	-50	50	-56	50	0.44
M-0.46	500 to 2 000	100	20 to 100	-50	50	-54	50	0.46
M-0.48	500 to 2 000	100	20 to 100	-50	50	-52	50	0.48
M-0.50	500 to 2 000	100	20 to 100	-50	50	-50	50	0.50
M-0.52	500 to 2 000	100	20 to 100	-50	50	-48	50	0.52
M-0.54	500 to 2 000	100	20 to 100	-50	50	-46	50	0.54
M-0.56	500 to 2 000	100	20 to 100	-50	50	-44	50	0.56
M-0.58	500 to 2 000	100	20 to 100	-50	50	-42	50	0.58

- *Large* (see Table 3.5) – This family of 480 QUBO maximization problems was also introduced in Boros et al. [62]. The number of variables in these problems ranges from 500 to 5 000. Expected density values range from 25% to 100% (in steps of 25%). The lower and upper bounds of the coefficients of the terms are described in Table 3.5. This choice of bounds produces three types of problems:

160 submodular problems with  $\bar{\rho} = 0.0$ ; 160 problems with  $\bar{\rho} = 0.2$ ; and 160 problems with  $\bar{\rho} = 0.5$ . Ten instances ( $k = 1, \dots, 10$ ) for each combination of the four values of the parameter  $n$ , four values of the parameter  $\bar{d}$  and three values of the parameter  $\bar{\rho}$ . The starting seed for the random number generator is the value of the expression

$$\left\lfloor \frac{n}{5} \right\rfloor + \lfloor 40\bar{d} \rfloor + k - 110 + \begin{cases} 4000, & \bar{\rho} = 0.0 \\ 3000, & \bar{\rho} = 0.2 \\ 0, & \bar{\rho} = 0.5 \end{cases} .$$

The best known values of the problems in this particular group are given in Table A.7 of the Appendix.

Table 3.5: Characteristics of QUBO problems in the *Large* family.

Sub-Family	Variables ( $n$ )	Number Problems	Density ( $\bar{d}$ %)	Linear Coef.		Quadr. Coef.		$\bar{\rho}$
				( $c^-$ )	( $c^+$ )	( $q^-$ )	( $q^+$ )	
L-500-0.00	500	40	25 to 100	1	100	-100	-1	0.00
L-1000-0.00	1000	40	25 to 100	1	100	-100	-1	0.00
L-2500-0.00	2500	40	25 to 100	1	100	-100	-1	0.00
L-5000-0.00	5000	40	25 to 100	1	100	-100	-1	0.00
L-500-0.20	500	40	25 to 100	-50	100	-100	50	0.20
L-1000-0.20	1000	40	25 to 100	-50	100	-100	50	0.20
L-2500-0.20	2500	40	25 to 100	-50	100	-100	50	0.20
L-5000-0.20	5000	40	25 to 100	-50	100	-100	50	0.20
L-500-0.50	500	40	25 to 100	-100	100	-100	100	0.50
L-1000-0.50	1000	40	25 to 100	-100	100	-100	100	0.50
L-2500-0.50	2500	40	25 to 100	-100	100	-100	100	0.50
L-5000-0.50	5000	40	25 to 100	-100	100	-100	100	0.50

- *Massive* (see Table 3.6) – This family contains 108 QUBO maximization problems with the number of variables ranging from 15 000 to 30 000 (in steps of 5 000), expected densities varying from 30% to 90% (in steps of 30%), expected values 0.0, 0.25 and 0.5, for the  $\rho$  parameter, and three instances ( $k = 1, 2, 3$ ) for each set of parameters. The starting seed for the random number generator is the value of the expression  $k + 2^3 q^+ + 2^{10} \lfloor 100\bar{d} \rfloor + 2^{17} \lfloor n/10 \rfloor$ .

Table 3.6: Characteristics of QUBO problems in the *Massive* family.

<i>Sub-Family</i>	<i>Variables</i> ( <i>n</i> )	<i>Number</i> <i>Problems</i>	<i>Density</i> ( $\bar{d}$ %)	<i>Linear Coef.</i>		<i>Quadr. Coef.</i>		$\bar{\rho}$
				( $c^-$ )	( $c^+$ )	( $q^-$ )	( $q^+$ )	
H-15000-0.00	15 000	9	30 to 90	1	50	-100	0	0.00
H-20000-0.00	20 000	9	30 to 90	1	50	-100	0	0.00
H-25000-0.00	25 000	9	30 to 90	1	50	-100	0	0.00
H-30000-0.00	30 000	9	30 to 90	1	50	-100	0	0.00
H-15000-0.25	15 000	9	30 to 90	1	50	-100	57	0.25
H-20000-0.25	20 000	9	30 to 90	1	50	-100	57	0.25
H-25000-0.25	25 000	9	30 to 90	1	50	-100	57	0.25
H-30000-0.25	30 000	9	30 to 90	1	50	-100	57	0.25
H-15000-0.50	15 000	9	30 to 90	1	50	-100	100	0.50
H-20000-0.50	20 000	9	30 to 90	1	50	-100	100	0.50
H-25000-0.50	25 000	9	30 to 90	1	50	-100	100	0.50
H-30000-0.50	30 000	9	30 to 90	1	50	-100	100	0.50

### 3.2 Graphs for maximum clique

A set of 138 benchmark graphs related to the maximum clique problem, introduced earlier in Section 2.2.1, is described in this section.

In order to facilitate comparisons among different methods related to clique problems, a set of benchmark graphs arising from different fields of application has been constructed in conjunction with the 1993 DIMACS challenge on maximum cliques, coloring and satisfiability [151]. These data are publicly available at a DIMACS FTP site<sup>2</sup>, along with other useful information.

Tables 3.7 and 3.8 describe the list of 85 DIMACS graphs, including the size of the largest clique found for each instance. The largest clique is known to be maximum in most graphs. The information on the largest and optimal cliques was taken mostly from the DIMACS FTP site described above. Other sources used were [75, 186].

The DIMACS graphs are categorized in 11 subfamilies:

- The *Brockington* graphs ([71]) are constructed to deliberately “hide” the optimal clique in relatively unattractive regions of the solutions space. This property makes this class of problems difficult to be solved by algorithms that use local information (e.g., vertex degree), which is generally used to guide the search

---

<sup>2</sup>DIMACS. (10/29/2004). The Second DIMACS Implementation Challenge: 1992-1993. <ftp://dimacs.rutgers.edu/pub/challenge/>.

Table 3.7: DIMACS graphs for the maximum clique problem (Part I).

<i>Sub-Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	$\rho$ %	$p$ %	<i>Maximum Clique</i>
Brockington	brock200.1	200	14 834	25.46	3.80	3.95	21
	brock200.2	200	9 876	50.37	1.96	2.00	12
	brock200.3	200	12 048	39.46	2.48	2.55	15
	brock200.4	200	13 089	34.23	2.85	2.94	17
	brock400.1	400	59 723	25.16	1.95	1.99	27
	brock400.2	400	59 786	25.08	1.96	2.00	29
	brock400.3	400	59 681	25.21	1.95	1.99	31
	brock400.4	400	59 765	25.11	1.96	2.00	33
	brock800.1	800	207 505	35.07	0.71	0.71	23
	brock800.2	800	208 166	34.87	0.71	0.72	24
	brock800.3	800	207 333	35.13	0.71	0.71	25
	brock800.4	800	207 643	35.03	0.71	0.71	26
C-FAT	c-fat200-1	200	1 534	92.29	1.08	1.09	12
	c-fat200-2	200	3 235	83.74	1.19	1.20	24
	c-fat200-5	200	8 473	57.42	1.72	1.75	58
	c-fat500-1	500	4 459	96.43	0.41	0.42	14
	c-fat500-2	500	9 139	92.67	0.43	0.43	26
	c-fat500-5	500	23 191	81.41	0.49	0.49	64
	c-fat500-10	500	46 627	62.62	0.64	0.64	126
C	C125.9	125	6 963	10.15	13.71	15.88	34
	C250.9	250	27 984	10.09	7.37	7.96	44
	C500.9	500	112 332	9.95	3.87	4.03	$\geq 57$
	C1000.9	1 000	450 079	9.89	1.98	2.02	$\geq 68$
	C2000.9	2 000	1 799 532	9.98	0.99	1.00	$\geq 78$
	C2000.5	2 000	999 836	49.98	0.20	0.20	$\geq 16$
	C4000.5	4 000	4 000 268	49.98	0.10	0.10	$\geq 18$
DSJC	DSJC500.5	500	125 248	49.80	0.80	0.80	14
	DSJC1000.5	1000	499 652	49.98	0.40	0.40	15
Hamming	hamming6-2	64	1 824	9.52	25.00	33.33	32
	hamming6-4	64	704	65.08	4.65	4.88	4
	hamming8-2	256	31 616	3.14	20.00	25.00	128
	hamming8-4	256	20 864	36.08	2.13	2.17	16
	hamming10-2	1 024	518 656	0.98	16.67	20.00	512
hamming10-4	1 024	434 176	17.11	1.13	1.14	$\geq 40$	
Johnson	johnson8-2-4	28	210	44.44	14.29	16.67	4
	johnson8-4-4	70	1 855	23.19	11.11	12.50	14
	johnson16-2-4	120	5 460	23.53	6.67	7.14	8
	johnson32-2-4	496	107 880	12.12	3.23	3.33	16
Keller	keller4	171	9 435	35.09	3.24	3.35	11
	keller5	776	225 990	24.85	1.03	1.04	27
	keller6	3 361	4 619 898	18.18	0.33	0.33	$\geq 59$

Table 3.8: DIMACS graphs for the maximum clique problem (Part II).

<i>Sub-Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	$\rho$ %	$p$ %	<i>Maximum Clique</i>
Mannino	MANN_a9	45	918	7.27	38.46	62.50	16
	MANN_a27	378	70 551	0.99	35.00	53.85	126
	MANN_a45	1 035	533 115	0.37	34.33	52.27	345
	MANN_a81	3 321	5 506 380	0.12	33.88	51.25	$\geq 1100$
P-HAT	p_hat300-1	300	10 933	75.62	0.88	0.88	8
	p_hat300-2	300	21 928	51.11	1.29	1.31	25
	p_hat300-3	300	33 390	25.55	2.55	2.62	36
	p_hat500-1	500	31 569	74.69	0.53	0.54	9
	p_hat500-2	500	62 946	49.54	0.80	0.81	36
	p_hat500-3	500	93 800	24.81	1.59	1.62	$\geq 50$
	p_hat700-1	700	60 999	75.07	0.38	0.38	11
	p_hat700-2	700	121 728	50.24	0.57	0.57	44
	p_hat700-3	700	183 010	25.20	1.12	1.14	$\geq 62$
	p_hat1000-1	1 000	122 253	75.52	0.26	0.27	10
	p_hat1000-2	1 000	244 799	50.99	0.39	0.39	$\geq 46$
	p_hat1000-3	1 000	371 746	25.58	0.78	0.78	$\geq 68$
	p_hat1500-1	1 500	284 923	74.66	0.18	0.18	12
	p_hat1500-2	1 500	568 960	49.39	0.27	0.27	$\geq 65$
p_hat1500-3	1 500	847 244	24.64	0.54	0.54	$\geq 94$	
R.5	r100.5	100	5 016	49.33	3.93	4.10	9
	r200.5	200	20 072	49.57	1.99	2.03	11
	r300.5	300	44 722	50.14	1.32	1.33	12
	r400.5	400	80 122	49.80	1.00	1.01	13
	r500.5	500	124 322	50.17	0.79	0.80	13
Sanchis	san200_0.7_1	200	13930	30.00	3.24	3.35	30
	san200_0.7_2	200	13930	30.00	3.24	3.35	18
	san200_0.9_1	200	17 910	10.00	9.13	10.05	70
	san200_0.9_2	200	17 910	10.00	9.13	10.05	60
	san200_0.9_3	200	17 910	10.00	9.13	10.05	44
	san400_0.5_1	400	39 900	50.00	0.99	1.00	13
	san400_0.7_1	400	55 860	30.00	1.64	1.67	40
	san400_0.7_2	400	55 860	30.00	1.64	1.67	30
	san400_0.7_3	400	55 860	30.00	1.64	1.67	22
	san400_0.9_1	400	71 820	10.00	4.77	5.01	100
	san1000	1000	250 500	49.85	0.40	0.40	15
	sanr200_0.7	200	13 868	30.31	3.21	3.32	18
	sanr200_0.9	200	17 863	10.24	8.94	9.82	42
	sanr400_0.5	400	39 984	49.89	0.99	1.00	13
	sanr400_0.7	400	55 869	29.99	1.64	1.67	$\geq 21$
	gen200_p0.9_44	200	17 910	10.00	9.13	10.05	44
	gen200_p0.9_55	200	17 910	10.00	9.13	10.05	55
	gen400_p0.9_55	400	71 820	10.00	4.77	5.01	55
	gen400_p0.9_65	400	71 820	10.00	4.77	5.01	65
gen400_p0.9_75	400	71 820	10.00	4.77	5.01	75	

through the solution space.

- A major step in the algorithm of the fault diagnosis problem proposed by Berman and Pelc [40] is to find the maximum clique of a special class of graphs, called  $c$ -fat rings. In order to define a  $c$ -fat graph  $G = (V, E)$ , let us consider an arbitrary finite set of vertices  $V$ . Let  $c$  be a real parameter,  $k = \left\lfloor \frac{|V|}{c \log |V|} \right\rfloor$ , and let us consider a partition  $W_0, \dots, W_{k-1}$  of  $V$ , such that  $c \log |V| \leq |W_i| \leq \lceil c \log |V| \rceil + 1$  for all  $i = 0, \dots, k - 1$ . The edge set  $E$  is defined as the set of those edges  $(u, v)$  which link distinct pairs of vertices  $u \in W_i$  and  $v \in W_j$ , such that  $|i - j| \in \{0, 1, k - 1\}$ . The DIMACS  $c$ -fat rings were created by Panos Pardalos using the  $c$ -fat rings generator of Hasselberg, Pardalos and Vairaktarakis [136].
- The  $C$  graphs  $G_{n,p}$  were randomly generated by Michael Trick using *ggen*, a program by Craig Morgenstern. The parameters used to create these graphs are the number of vertices  $n$ , and the probability  $p$  of an edge to exist between any two vertices.
- The *DSJC* graphs were randomly generated by Johnson et al. [150], all having an expected density of 50%.
- The *Hamming* graphs arise from coding theory problems ([224]). The Hamming distance between the binary vectors  $\mathbf{u} = (u_1, \dots, u_n)$  and  $\mathbf{v} = (v_1, \dots, v_n)$  is the number of indices  $i = 1, \dots, n$  where  $u_i \neq v_i$ . The Hamming graph  $H(n, h)$  of size  $n$  and distance  $h$  is the graph whose vertex set is the set of all binary  $n$ -vectors, and whose edges link any two  $n$ -vectors at distance  $h$  or larger. Clearly, the graph  $H(n, h)$  has  $2^n$  vertices,  $2^{n-1} \sum_{i=h}^n \binom{n}{i}$  edges, and the degree of each vertex is  $\sum_{i=h}^n \binom{n}{i}$ . A binary code consisting of a set of binary vectors, any two of which have Hamming distance greater or equal to  $h$ , can correct  $\lfloor \frac{h-1}{2} \rfloor$  errors. Thus, a coding theorist (see [174]) would like to find the maximum number of binary vectors of size  $n$  with Hamming distance  $h$ , i.e. the maximum clique of  $H(n, h)$ . The DIMACS Hamming graphs were created by Panos Pardalos (for details see [136]).

- The *Johnson* graphs also arise from coding theory problems. The Johnson graph  $Jpcn, w, h$ , with parameters  $n, w$  and  $h$ , is the graph with vertex set of binary vectors of size  $n$  and weight  $w$ , where two vertices are adjacent if their Hamming distance is at least  $h$ . The graph  $J(n, w, h)$  has  $\binom{n}{w}$  vertices,  $\frac{1}{2} \binom{n}{w} \sum_{k=\lceil \frac{h}{2} \rceil}^w \binom{w}{k} \binom{n-w}{k}$  edges and the degree of each vertex is  $\sum_{k=\lceil \frac{h}{2} \rceil}^w \binom{w}{k} \binom{n-w}{k}$  ([136]). A binary code consisting of vectors of size  $n$ , weight  $w$  and distance  $h$ , can correct  $w - \frac{h}{2}$  errors. In this case, a coding theorist ([174]) would like to find a weighted binary code, defined by the maximum number of binary vectors of size  $n$  that have precisely  $w$  indices with value 1, and for which the Hamming distance of any two of these vectors is  $h$ . This number is precisely the maximum clique of  $J(n, w, h)$ . The DIMACS Hamming graphs were created by Panos Pardalos (for details see [136]).
- The *Keller* graphs are graphs for which a maximum clique can be used to prove or disprove the Keller's conjecture on tilings hypercubes (see [164, 136] for more details). The Keller graph  $\Gamma_k$  is a graph with vertex set

$$V_k = \{(d_1, \dots, d_k) : d_i \in \{0, 1, 2, 3\}, i = 1, \dots, k\}$$

where two vertices  $u = (d_1^u, \dots, d_k^u)$  and  $v = (d_1^v, \dots, d_k^v)$  in  $V_k$  are adjacent if and only if

$$\exists i, i = 1, \dots, k : d_i^u - d_i^v \equiv 2 \pmod{4}$$

and

$$\exists j \neq i, j = 1, \dots, k : d_j^u \neq d_j^v.$$

Corrádi and Szabó [88] show that there is a counterexample to Keller's conjecture if and only if there is a positive integer  $k$ , such that  $\Gamma_k$  has a clique of size  $2^k$ .

$\Gamma_k$  has  $4^k$  vertices,  $\frac{1}{2}4^k(4^k - 3^k - k)$  edges and the degree of each node is  $4^k - 3^k - k$ .  $\Gamma_k$  is very dense, and it has at least  $8^k k!$  different cliques.

- The *Mannino* graphs are a consequence of a clique formulation of the Steiner triple problem, translated from the set covering formulation. It should be noted

that these graphs are extremely dense in which only a few from all possible edges are missing.

- The *P-fat* graphs ([106]) are created from a generalization of the classical uniform random graph generator. These graphs have a wider node degree spread and larger cliques than uniform random graphs.
- The five graphs in the *R.5* family were proposed in the DIMACS challenge to serve as benchmarks for defining ratios of computing times between different computer machines when the same source code of program *dfmax* is used. *dfmax* is a simple-minded branch-and-bound program very similar to that of Carraghan and Pardalos [79]. The source code of program *dfmax*, written by David Applegate and David Johnson, is available at the DIMACS FTP site. In practice it can find a maximum clique for graph with 500 vertices, and 50% density, in a few minutes. The *R.5* graphs are random graphs, 50% dense, with the number of vertices varying from 100 to 500.
- Sanchis [214, 215] proposed three sub-families of test problems: *san*, *sanr* and *gen*. The *san* graphs are randomly generated problems from the complement graph of instances of the vertex covering problem (see Section 2.2.1). The generation parameters include the number of vertices, the number of edges, and the maximum clique size. The *sanr* graphs are of similar size to the *san* graphs, but with different clique characteristics. The *gen* graphs are artificially generated instances with large, known embedded clique. Regarding the difficulty of the problems generated, the reader is referred to [214].

Pardalos with Carraghan [79] and Rodgers [196] proposed a routine to generate graphs. Using this routine Pardalos et al. [79, 196] proposed some benchmarks for which the corresponding maximum cliques were found. We shall call this subfamily of test problems has *CPR*. The list of problems and the corresponding sizes of the maximum cliques can be seen in Table 3.9.

The *FRB* maximum clique benchmarks presented in Table 3.9 are directly transformed from forced satisfiable *Constraint Satisfaction Problems* ([237]), with the set of vertices and the set of edges respectively corresponding to the set of variables, and to the set of binary clauses in the satisfiability instances. Based on this model (called model *RB*) and transformation, the *FRB* graphs are obtained as follows:

1. Generate  $k$  disjoint cliques, each of which has  $k^a$  vertices (where  $a > 0$  is a constant);
2. Randomly select two different cliques and then generate without repetitions  $pn^{2a}$  random edges between these two cliques (where  $0 < p < 1$  is a constant);
3. Run step 2 (with repetitions) for another  $rn \log n - 1$  times (where  $r > 0$  is a constant).

The graph obtained with the previous procedure generates a graph with a maximum independent set of size at most  $k$ . Determining if such an upper bound can be reached is equivalent to determining the satisfiability of the corresponding constraint satisfaction problem. Furthermore, there is a one-to-one correspondence between the solutions of these two problems. To hide an independent set of size  $k$  in these graph instances, a vertex is selected at random from each disjoint clique to form an independent set of size  $k$ . Then, in the step of generating random edges, no edge is allowed to violate this maximum independent set. The graphs of Table 3.9 are the complements of graph instances generated in this way.

In this study, we solve the maximum clique problem by associating to it a quadratic pseudo-Boolean function (see (2.1) in Section 2.2.1), for which the maximum value is the maximum clique size. Otherwise stated, for every edge  $(i, j) \in E$ ,  $\epsilon_{(i,j)} = 0$  is assumed in (2.1) and (2.3). The parameters  $d$ ,  $\rho$  and  $p$  shown in Tables 3.7–3.9 were computed using this assumption.

A group of 8 graphs related to the weighted maximum clique problem (see Section 2.2.1) is listed in Table 3.10. These test problems are obtained from complemented

Table 3.9: Additional graphs ([79, 196, 237]) for the maximum clique problem.

<i>Sub-Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	$\rho$ %	$p$ %	<i>Maximum Clique</i>
CPR	1000A	1 000	50 000	89.99	0.22	0.22	6
	1000B	1 000	100 000	79.98	0.25	0.25	7
	1000C	1 000	150 000	69.97	0.29	0.29	10
	2000B	2 000	400 347	79.97	0.12	0.13	8
	3000B	3 000	899 647	80.00	0.08	0.08	9
FRB	frb30-15-1	450	83 198	17.65	2.46	2.52	30
	frb30-15-2	450	83 151	17.69	2.46	2.52	30
	frb30-15-3	450	83 216	17.63	2.46	2.53	30
	frb30-15-4	450	83 194	17.65	2.46	2.52	30
	frb30-15-5	450	83 231	17.61	2.47	2.53	30
	frb35-17-1	595	148 859	15.76	2.09	2.14	35
	frb35-17-2	595	148 868	15.76	2.09	2.14	35
	frb35-17-3	595	148 784	15.81	2.09	2.13	35
	frb35-17-4	595	148 873	15.76	2.09	2.14	35
	frb35-17-5	595	148 572	15.93	2.07	2.11	35
	frb40-19-1	760	247 106	14.32	1.81	1.84	40
	frb40-19-2	760	247 157	14.31	1.81	1.84	40
	frb40-19-3	760	247 325	14.25	1.82	1.85	40
	frb40-19-4	760	246 815	14.43	1.79	1.83	40
	frb40-19-5	760	246 801	14.43	1.79	1.83	40
	frb45-21-1	945	386 854	13.27	1.57	1.60	45
	frb45-21-2	945	387 416	13.14	1.59	1.61	45
	frb45-21-3	945	387 795	13.06	1.60	1.62	45
	frb45-21-4	945	387 491	13.13	1.59	1.61	45
	frb45-21-5	945	387 461	13.13	1.59	1.61	45
	frb50-23-1	1 150	580 603	12.12	1.42	1.44	50
	frb50-23-2	1 150	579 824	12.24	1.40	1.42	50
	frb50-23-3	1 150	579 607	12.27	1.40	1.42	50
	frb50-23-4	1 150	580 417	12.15	1.41	1.43	50
	frb50-23-5	1 150	580 640	12.11	1.42	1.44	50
	frb53-24-1	1 272	714 129	11.66	1.33	1.35	53
	frb53-24-2	1 272	714 067	11.66	1.33	1.35	53
	frb53-24-3	1 272	714 229	11.64	1.33	1.35	53
	frb53-24-4	1 272	714 048	11.67	1.33	1.35	53
	frb53-24-5	1 272	714 130	11.66	1.33	1.35	53
	frb56-25-1	1 400	869 624	11.20	1.26	1.28	56
	frb56-25-2	1 400	869 899	11.17	1.26	1.28	56
	frb56-25-3	1 400	869 921	11.17	1.26	1.28	56
	frb56-25-4	1 400	869 262	11.24	1.26	1.27	56
	frb56-25-5	1 400	869 699	11.19	1.26	1.28	56
frb59-26-1	1 534	1 049 256	10.76	1.20	1.21	59	
frb59-26-2	1 534	1 049 648	10.73	1.20	1.22	59	
frb59-26-3	1 534	1 049 729	10.72	1.20	1.22	59	
frb59-26-4	1 534	1 048 800	10.80	1.19	1.21	59	
frb59-26-5	1 534	1 049 829	10.71	1.20	1.22	59	

graphs, randomly generated by Pardalos and Desai ([192]), for which a maximum weighted independent set was found. The weights of the vertices are random integers between 1 and 10. Table 3.10 list the characteristics of the complements of the original graphs, including the weight of the optimal clique. Note that [192] only reports the size of the largest independent set, and not the corresponding weight.

Table 3.10: Graphs of Pardalos and Desai [192] for the weighted maximum clique problem.

<i>Sub-Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	$\rho$ %	$p$ %	<i>Maximum Clique Weight</i>
PD-500	PD-500.1	500	112 176	10.08	1.96	1.99	$\geq 380$
	PD-500.2	500	99 832	19.97	0.99	1.00	$\geq 222$
	PD-500.3	500	87 445	29.90	0.67	0.67	$\geq 163$
	PD-500.4	500	74 925	39.94	0.50	0.50	125
	PD-500.5	500	62 422	49.96	0.40	0.40	96
	PD-500.6	500	49 749	60.12	0.33	0.33	78
	PD-500.7	500	37 183	70.19	0.28	0.29	63
	PD-500.8	500	25 083	79.89	0.25	0.25	51

In this study, we solve the weighted maximum clique problem by associating to it a quadratic pseudo-Boolean function (see (2.3) in Section 2.2.1), for which the maximum value is the maximum weight of a clique. We used  $\epsilon_{(i,j)} = c_i + c_j$  in (2.3), for every edge  $(i, j) \in E$ . The values of  $d$ ,  $\rho$  and  $p$  in Table 3.10 are a consequence of this option.

### 3.3 Planar graphs for minimum vertex cover

A set of planar graphs randomly generated by the LEDA software package ([176]) is considered in this study. It is important to note that even for planar graphs it is known that solving the minimum vertex cover problem is NP-hard ([105]).

Using the LEDA generator we tried to replicate the experiment reported in Alber, Dorn and Niedermeier [13], although it should be noted that not having access to the seeds used in [13], the graphs generated by us are not exactly identical to the ones used by Alber, Dorn and Niedermeier [13]. In order to distinguish between the two planar vertex cover benchmarks, we shall call those of [13] ADN benchmark graphs and the new ones *PVC LEDA benchmark*.

The total number of planar graphs that we have generated with LEDA is 400, partitioned into 4 sets of 100 graphs, each subset having a specific number of vertices: 1000, 2000, 3000 and 4000. The *planar density* of each graph  $G(V, E)$  was randomly determined, i.e.  $|E| \sim \text{discrete uniform}(|V| - 1, 3|V| - 6)$ . Some comparative statistical numbers about these two benchmarks are displayed in Table 3.11.

Table 3.11: Comparative statistical numbers about the LEDA benchmarks.

Benchmark	Vertices	Number of Graphs	Average Number of Edges	Average Maximum Degree	Average Degree	Average Minimum Vertex Cover
PVC LEDA	1000	100	2037.9	73.0	4.08	460.6
	2000	100	4068.6	106.7	4.07	921.1
	3000	100	6204.3	132.4	4.14	1391.2
	4000	100	8207.1	149.2	4.10	1848.2
ADN ([13])	1000	100	1978.9	73.3	3.96	453.9
	2000	100	3960.8	104.9	3.96	917.3
	3000	100	6070.6	129.6	4.05	1373.8
	4000	100	8264.5	146.6	4.13	1856.8

In addition to the PVC LEDA planar graphs we have also generated a dataset containing larger graphs with up to 500 000 vertices. These graphs were generated in order to analyze the scalability of the routine PREPRO. Because of size limitations associated to our trial license on LEDA, we used for this experiment Rinaldi's ([211]) generator called RUDY. With the RUDY program, we generated a total of 36 graphs whose sizes are of 50 000, 100 000, 250 000 and 500 000 vertices; for each of these graph sizes, we generated nine graphs: three instances with density of 10%, three with density of 50% and three with density of 90%. This set of benchmark graphs is called PVC RUDY.

### 3.4 Graphs for MAX-CUT

A collection of (weighted) graphs related to the (weighted) MAX-CUT problem, introduced earlier in Section 2.2.3, is described in this section. A set of 135 (weighted) graphs used previously in other studies, is described next. In addition to the public benchmarks, we randomly generated a set of 240 graphs with  $m$ -Hamiltonian cycles, where  $m$  is a specified parameter. The generation details of the Hamiltonian graphs are presented at the end of this section.

### 3.4.1 Benchmark families

The MAX-CUT problem arises from different applications, such as when one needs to find the minimum energy and particle states of a Ising model, or when the minimum number of layers/vias has to be computed during the design process for VLSI chips or printed circuit boards.

Most of the benchmark problems for MAX-CUT are randomly generated with different algorithms or settings, making it possible to analyze this problem on variety of classes of graphs. The benchmarks for MAX-CUT include 14 2D-toroidal graphs and 34 3D-toroidal graphs. Ten graphs derived from VLSI problems are also part of this group of problems.

Helmberg and Rendl [140] used the graph generator *rudi*, written by Rinaldy, to create the G graphs. The graphs of the G family have been frequently cited in several publications related to the MAX-CUT problem (e.g., [74, 97, 188]). It contains a group of random graphs with no weights associated to the edges (listed in Table 3.12), and a group of graphs with a  $\pm 1$  weights associated to the edges (listed in Table 3.13). The probability of an edge to have a negative weight is in this case 50%.

For each group of problems, Helmberg and Rendl [140] considers three classes of graphs: *random* graphs with a prescribed edge density; graphs resulting from the union of two random *planar* graphs; and *2D-toroidal* graphs. Tables 3.12 and 3.13 contain the graph characteristics, and also includes information about the largest known cut.

Burer et al. [74] proposed the graph instances in Table 3.14. These graphs consist of thirty cubic lattices having randomly generated  $\pm 1$  interaction magnitudes. Each graph has a side length  $L$ , has  $n = L^3$  vertices and  $3n$  edges. There are ten graphs for each value of the side length  $L$ , which are the values 5, 10 and 14. [74] tested a rank-2 relaxation heuristic (called *circut*) for MAX-CUT on these cubic lattice graphs. Subsequently, Festa et al. [97] and Palubeckis et al. [188] respectively used these graphs for testing GRASP and tabu search as heuristic techniques for MAX-CUT.

The *torus* graphs are 3D-toroidal graphs, originated from the Ising model of spin

Table 3.12: G-graphs of Helmberg and Rendl [140] for MAX-CUT.

<i>Family</i>	<i>Sub-Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	<i>MAX-CUT</i> $ (\mathcal{S}, \bar{\mathcal{S}}) $	
Random	GR-800	G1	800	19 176	6.00	$\geq 11\,624^\dagger$	
		G2	800	19 176	6.00	$\geq 11\,620^\ddagger$	
		G3	800	19 176	6.00	$\geq 11\,622^\dagger$	
		G4	800	19 176	6.00	$\geq 11\,646$	
		G5	800	19 176	6.00	$\geq 11\,631$	
	GR-1000	G43	1 000	9 990	2.00	$\geq 6\,660^\ddagger$	
		G44	1 000	9 990	2.00	$\geq 6\,650^\ddagger$	
		G45	1 000	9 990	2.00	$\geq 6\,654^\ddagger$	
		G46	1 000	9 990	2.00	$\geq 6\,649$	
		G47	1 000	9 990	2.00	$\geq 6\,657$	
	GR-2000	G22	2 000	19 990	1.00	$\geq 13\,358^\ddagger$	
		G23	2 000	19 990	1.00	$\geq 13\,354^\ddagger$	
		G24	2 000	19 990	1.00	$\geq 13\,335$	
		G25	2 000	19 990	1.00	$\geq 13\,339$	
		G26	2 000	19 990	1.00	$\geq 13\,317$	
	GR-5000	G55	5 000	12 498	0.10	$\geq 10\,264$	
	GR-7000	G60	7 000	17 148	0.07	$\geq 14\,149$	
	$2 \times$ Planar	GP-800	G14	800	4 694	1.47	$\geq 3\,064$
			G15	800	4 661	1.46	$\geq 3\,050^\ddagger$
			G16	800	4 672	1.46	$\geq 3\,052^\ddagger$
G17			800	4 667	1.46	$\geq 3\,044$	
GP-1000		G51	1 000	5 909	1.18	$\geq 3\,848$	
		G52	1 000	5 916	1.18	$\geq 3\,849$	
		G53	1 000	5 914	1.18	$\geq 3\,848$	
		G54	1 000	5 916	1.18	$\geq 3\,848$	
GP-2000		G35	2 000	11 778	0.59	$\geq 7\,683$	
		G36	2 000	11 766	0.59	$\geq 7\,674$	
		G37	2 000	11 785	0.59	$\geq 7\,681^\ddagger$	
		G38	2 000	11 779	0.59	$\geq 7\,672$	
GP-5000		G58	5 000	29 570	0.24	$\geq 19\,246$	
GP-7000		G63	7 000	41 459	0.17	$\geq 26\,959$	
Toroidal	GT-50 $\times$ 60	G48	3 000	6 000	0.13	6 000*	
	GT-30 $\times$ 100	G49	3 000	6 000	0.13	6 000*	
	GT-25 $\times$ 120	G50	3 000	6 000	0.13	5 880*	

$^\dagger$ Solution reported first by Festa et al. [97].

$^\ddagger$ Solution reported first by Palubeckis and Krivickiene [188].

\*Solution reported first by Burer et al. [74].

Table 3.13:  $G_{\pm 1}$ -graphs of Helmberg and Rendl [140] for MAX-CUT.

<i>Family</i>	<i>Sub-Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	<i>MAX-CUT</i> $ W(S, \bar{S}) $	
Random	GR-800	G6	800	19 176	6.00	$\geq 2178$	
		G7	800	19 176	6.00	$\geq 2006$	
		G8	800	19 176	6.00	$\geq 2005$	
		G9	800	19 176	6.00	$\geq 2054$	
		G10	800	19 176	6.00	$\geq 2000$	
	GR-2000	G27	2000	19 990	1.00	$\geq 3325$	
		G28	2000	19 990	1.00	$\geq 3296$	
		G29	2000	19 990	1.00	$\geq 3391$	
		G30	2000	19 990	1.00	$\geq 3408$	
		G31	2000	19 990	1.00	$\geq 3294$	
	GR-5000	G56	5000	12 498	0.10	$\geq 3994$	
	GR-7000	G61	7000	17 148	0.07	$\geq 5741$	
	$2 \times$ Planar	GP-800	G18	800	4 694	1.47	$\geq 988$
			G19	800	4 661	1.46	$\geq 906$
G20			800	4 672	1.46	$\geq 941$	
G21			800	4 667	1.46	$\geq 930$	
GP-2000		G39	2000	11 778	0.59	$\geq 2375$	
		G40	2000	11 766	0.59	$\geq 2384$	
		G41	2000	11 785	0.59	$\geq 2380$	
		G42	2000	11 779	0.59	$\geq 2465$	
GP-5000		G59	5000	29 570	0.24	$\geq 5971$	
GP-7000		G64	7000	41 459	0.17	$\geq 8575^*$	
Toroidal		GT-100 $\times$ 8	G11	800	1 600	0.50	564 <sup>†</sup>
	GT-50 $\times$ 16	G12	800	1 600	0.50	556 <sup>†</sup>	
	GT-25 $\times$ 32	G13	800	1 600	0.50	582 <sup>‡</sup>	
	GT-100 $\times$ 20	G32	2000	4 000	0.20	$\geq 1410$	
	GT-80 $\times$ 25	G33	2000	4 000	0.20	$\geq 1382$	
	GT-50 $\times$ 40	G34	2000	4 000	0.20	$\geq 1384$	
	GT-100 $\times$ 50	G57	5000	10 000	0.08	$\geq 3492$	
	GT-100 $\times$ 70	G62	7000	14 000	0.06	$\geq 4862$	
	GT-100 $\times$ 80	G65	8000	16 000	0.05	$\geq 5550$	
	GT-90 $\times$ 100	G66	9000	18 000	0.04	$\geq 6352$	
	GT-100 $\times$ 100	G67	10 000	20 000	0.04	$\geq 6932$	

<sup>†</sup>Solution reported first by Festa et al. [97].

<sup>‡</sup>Solution reported first by Palubeckis and Krivickiene [188].

\*Solution reported first by Burer et al. [74].

Table 3.14: Cubic lattice graphs of Burer et al. [74] for MAX-CUT.

<i>Family</i>	<i>Sub-Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	<i>MAX-CUT</i> $ W(S, \bar{S}) $
sg3dl	sg3dl05	sg3dl051000	125	375	4.84	110
		sg3dl052000	125	375	4.84	112
		sg3dl053000	125	375	4.84	106
		sg3dl054000	125	375	4.84	114
		sg3dl055000	125	375	4.84	112
		sg3dl056000	125	375	4.84	110
		sg3dl057000	125	375	4.84	112
		sg3dl058000	125	375	4.84	108
		sg3dl059000	125	375	4.84	110
		sg3dl0510000	125	375	4.84	112
	sg3dl10	sg3dl101000	1 000	3 000	0.60	$\geq 896$
		sg3dl102000	1 000	3 000	0.60	$\geq 900$
		sg3dl103000	1 000	3 000	0.60	$\geq 892$
		sg3dl104000	1 000	3 000	0.60	$\geq 898$
		sg3dl105000	1 000	3 000	0.60	$\geq 886$
		sg3dl106000	1 000	3 000	0.60	$\geq 888$
		sg3dl107000	1 000	3 000	0.60	$\geq 900$
		sg3dl108000	1 000	3 000	0.60	$\geq 882$
		sg3dl109000	1 000	3 000	0.60	$\geq 902$
		sg3dl1010000	1 000	3 000	0.60	$\geq 894$
	sg3dl14	sg3dl141000	2 744	8 232	0.22	$\geq 2 446$
		sg3dl142000	2 744	8 232	0.22	$\geq 2 458$
		sg3dl143000	2 744	8 232	0.22	$\geq 2 442$
		sg3dl144000	2 744	8 232	0.22	$\geq 2 450$
		sg3dl145000	2 744	8 232	0.22	$\geq 2 446$
		sg3dl146000	2 744	8 232	0.22	$\geq 2 450$
		sg3dl147000	2 744	8 232	0.22	$\geq 2 444$
		sg3dl148000	2 744	8 232	0.22	$\geq 2 446$
		sg3dl149000	2 744	8 232	0.22	$\geq 2 424$
		sg3dl1410000	2 744	8 232	0.22	$\geq 2 458$

glasses in physics. They were taken from the DIMACS library of mixed semidefinite-quadratic-linear programs [1] (see also [179]). Two graphs have  $\pm 1$  interaction magnitudes, whereas the other two graphs have interactions determined by a Gaussian distribution. The general characteristics of these graphs, and the largest cut information can be seen in Table 3.15.

Table 3.15: DIMACS torus graphs for MAX-CUT.

<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	<i>MAX-CUT</i> $ W(S, \bar{S}) $
pm3-8-50	512	1 536	1.17	458
pm3-15-50	3 375	10 125	0.18	$\geq 3\,016^\dagger$
g3-8	512	1 536	1.17	41 684 814 $^\ddagger$
g3-15	3 375	10 125	0.18	$\geq 285\,790\,637^\ddagger$

$^\dagger$ Solution reported first by Palubeckis and Krivickiene [188].

$^\ddagger$ Solution reported first by Burer et al. [74].

Homer and Peinado ([145]) tested several approximation algorithms for MAX-CUT on sparse random graphs and on graphs derived from circuit design problems:

- *Sparse random graphs* – These eight graphs constitute the family  $R$  of Homer and Peinado ([145]). Each graph has an edge probability of  $10/n$ , and the number of vertices  $n$  varies from 1 000 to 8 000. These graphs belong to the random graph class  $C$  in Goemans and Williamson [112].
- *Via graphs* – Graphs provided by Homer and Peinado [145], derived from layer assignment problems in the design process for VLSI chips. Each edge has a coefficient associated to it, some of them being negative.

The characteristics of these graphs, and the largest cut information can be seen in Table 3.16.

Kim et al. [157] tested a hybrid genetic algorithm on both the  $R$  and the *via* families of graphs. Kim et al. [157] also includes the following classes of graphs in their experiments:

- *Gn.p graphs*: Each graph has  $n$  vertices ( $n$  being 500 or 1000), and an edge is placed between two vertices with probability  $p$ , independently of other edges. The probability  $p$  is chosen so that the expected vertex degree is  $d = p(n - 1)$ .

Table 3.16: Graphs of Homer and Peinado [145] for MAX-CUT.

<i>Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	<i>MAX-CUT</i> $ W(S\bar{S}) $
random	R1000	1 000	5 033	1.01	$\geq 3687^\dagger$
	R2000	2 000	9 943	0.50	$\geq 7308^\dagger$
	R3000	3 000	14 965	0.33	$\geq 10997$
	R4000	4 000	19 939	0.25	$\geq 14684$
	R5000	5 000	24 794	0.20	$\geq 18225$
	R6000	6 000	29 862	0.17	$\geq 21937$
	R7000	7 000	35 110	0.14	$\geq 25763$
	R8000	8 000	39 642	0.12	$\geq 29140^\dagger$
via	via.c1n	828	1 389	0.41	6 150
	via.c2n	980	1 712	0.36	7 098
	via.c3n	1 327	2 393	0.27	6 898
	via.c4n	1 366	2 539	0.27	10 098
	via.c5n	1 202	2 129	0.29	7 956
	via.c1y	829	1 693	0.49	7 746
	via.c2y	981	2 039	0.42	8 226
	via.c3y	1 328	2 757	0.31	9 502
	via.c4y	1 367	2 848	0.31	12 516
	via.c5y	1 203	2 452	0.34	10 248

<sup>†</sup>Solution reported first by Kim et al. [157].

- *Un.p graphs*: Each graph has  $n$  vertices ( $n$  being 500 or 1000) that lie in the unit square and whose coordinates are chosen uniformly from the unit interval. There is an edge between two vertices if their Euclidean distance is  $t$ , which results in an expected vertex degree of  $d = n\pi t^2$ .

The characteristics of these graphs, and the largest cut information can be seen in Table 3.17.

### 3.4.2 Graphs with $m$ -Hamiltonian random cycles

We have randomly generated 240 graphs for MAX-CUT. Each graph has a user-specified number of Hamiltonian cycles randomly generated. This family was named as the *Hamilton* family, and has the following characteristics (see Table 3.18): number of vertices is 250, 500, 1 000 or 2000, plus one additional vertex that represents an exterior field; number of Hamiltonian cycles varies from 2 to 8 (in steps of 2); weights of edges are discrete uniformly distributed as  $[-50, 100]$ ,  $[-50, 50]$ ,  $[-50, -1]$ , or are fixed to 1; and three instances ( $k = 1, 2, 3$ ) were generated for each set of parameters.

The best known solutions of the problems in the Hamilton family are given in Table

Table 3.17: Graphs of Kim, Kim and Moon [157] for MAX-CUT.

<i>Family</i>	<i>Problem Name</i>	<i>Vertices</i> ( $ V $ )	<i>Edges</i> ( $ E $ )	<i>Density</i> ( $d$ %)	<i>MAX-CUT</i> $ W(S\bar{S}) $
random	g500.2.5	500	625	0.50	574
	g500.05	500	1 223	0.98	$\geq 1 008$
	g500.10	500	2 355	1.89	$\geq 1 735$
	g500.20	500	5 120	4.10	$\geq 3 390$
	g1000.2.5	1 000	1 272	0.25	$\geq 1 173$
	g1000.05	1 000	2 496	0.50	$\geq 2 053$
	g1000.10	1 000	5 064	1.01	$\geq 3 705$
	g1000.20	1 000	10 107	2.02	$\geq 6 729$
geometric	U500.05	500	1 282	1.03	900
	U500.10	500	2 355	1.89	$\geq 1 546$
	U500.20	500	4 549	3.65	$\geq 2 783$
	U500.40	500	8 793	7.05	$\geq 5 181$
	U1000.05	1 000	2 394	0.48	$\geq 1 711$
	U1000.10	1 000	4 696	0.94	$\geq 3 073$
	U1000.20	1 000	9 339	1.87	$\geq 5 737$
	U1000.40	1 000	18 015	3.61	$\geq 10 560$

Table 3.18: Graphs with  $m$ -Hamiltonian randomly generated cycles for MAX-CUT.

<i>Family</i>	<i>Sub-Family</i>	<i>Vertices</i> ( $ V $ )	<i>Number Problems</i>	<i>Number Cycles</i> ( $m$ )	<i>Exterior Field</i> ( $h$ )	<i>Edge's Weights</i>	
						( $w^-$ )	( $w^+$ )
Hamilton	HAM-2-1	250 to 2 000	12	2	-75	50	100
	HAM-2-2	250 to 2 000	12	2	75	50	100
	HAM-2-3	250 to 2 000	12	2	0	-50	50
	HAM-2-4	250 to 2 000	12	2	25	-50	50
	HAM-2-5	250 to 2 000	12	2	0	1	1
	HAM-4-1	250 to 2 000	12	4	-75	50	100
	HAM-4-2	250 to 2 000	12	4	75	50	100
	HAM-4-3	250 to 2 000	12	4	0	-50	50
	HAM-4-4	250 to 2 000	12	4	25	-50	50
	HAM-4-5	250 to 2 000	12	4	0	1	1
	HAM-6-1	250 to 2 000	12	6	-75	50	100
	HAM-6-2	250 to 2 000	12	6	75	50	100
	HAM-6-3	250 to 2 000	12	6	0	-50	50
	HAM-6-4	250 to 2 000	12	6	25	-50	50
	HAM-6-5	250 to 2 000	12	6	0	1	1
	HAM-8-1	250 to 2 000	12	8	-75	50	100
	HAM-8-2	250 to 2 000	12	8	75	50	100
	HAM-8-3	250 to 2 000	12	8	0	-50	50
	HAM-8-4	250 to 2 000	12	8	25	-50	50
	HAM-8-5	250 to 2 000	12	8	0	1	1

A.8 of the Appendix.

### 3.5 Maximum 2-satisfiability test problems

A set of (weighted) satisfiability formulas related to the (weighted) MAX-2-SAT problem, introduced earlier in Section 2.3, is described in this section. A set of 34 benchmark (weighted) formulas is described in the following subsection. In addition to the benchmarks, a set of 640 satisfiability formulas were randomly generated by using probabilistic parameters over the set of all possible clauses. The MAX-2-SAT generator and the details of the parameters of these formulas are presented in the end of this section.

#### 3.5.1 Benchmark families

Borchers and Furman [47] proposed an exact algorithm for (weighted) MAX-SAT, and tested this solver in a set of random (weighted) MAX-2-SAT problems. Since then, several other researchers ([16, 17, 113, 147, 219, 220, 221, 236, 241]) used this algorithm and test problems, for comparison with their proposed algorithmic approaches. The source code and the MAX-2-SAT instances are publicly available on the Internet ([3]).

The list of problems contains 17 standard formulas and 17 formulas with weights (ranging from one to ten) associated to the clauses. The number of variables in the formulas is 50, 100 and 150. The number of clauses varies from 100 to 600, depending on the number of variables.

The details of the non-weighted formulas can be seen in Table 3.19, and the details of the weighted formulas can be seen in Table 3.20. The optimal MAX-SAT solution is known for all instances. In this study, we solved the (weighted) MAX-2-SAT problem by associating a quadratic posiform  $\phi$  (see (2.7) in Section 2.3) to it, for which the minimum value  $\nu(\phi)$  is the minimum weighted set of unsatisfied clauses. The values of the parameters  $d$ ,  $\rho$  and  $p$  are relative to the (unique) quadratic pseudo-Boolean polynomial 1.5 associated with the posiform  $\phi$ .

Table 3.19: MAX-2-SAT instances of Borchers and Furman [47].

<i>Sub-Family</i>	<i>Problem (<math>\phi</math>) Name</i>	<i>Variables (<math>n</math>)</i>	<i>Clauses (<math>A(\phi)</math>)</i>	<i>Density (<math>d</math> %)</i>	$\rho$ %	$p$ %	<i>False Clauses (<math>\nu(\phi)</math>)</i>
BF-50	BF-50-100	50	100	7.59	49.32	57.45	4
	BF-50-150	50	150	10.86	49.76	50.74	8
	BF-50-200	50	200	14.37	46.56	43.96	16
	BF-50-250	50	250	17.22	50.79	40.27	22
	BF-50-300	50	300	20.90	48.78	34.67	32
	BF-50-350	50	350	23.10	49.16	34.74	41
	BF-50-400	50	400	25.22	48.50	37.06	45
	BF-50-450	50	450	29.88	49.60	28.65	63
	BF-50-500	50	500	30.69	49.71	23.80	66
BF-100	BF-100-200	100	200	3.88	49.84	61.73	5
	BF-100-300	100	300	5.88	50.69	45.95	15
	BF-100-400	100	400	7.47	50.49	36.97	29
	BF-100-500	100	500	9.56	48.34	37.34	44
	BF-100-600	100	600	10.85	49.11	31.12	65
BF-150	BF-150-300	150	300	2.67	50.65	55.70	4
	BF-150-450	150	450	3.94	49.69	44.14	22
	BF-150-600	150	600	5.12	49.94	41.70	38

Table 3.20: Weighted MAX-2-SAT instances of Borchers and Furman [47].

<i>Sub-Family</i>	<i>Problem (<math>\phi</math>) Name</i>	<i>Variables (<math>n</math>)</i>	<i>Clauses Number</i>	<i>Weight (<math>A(\phi)</math>)</i>	<i>Density (<math>d</math> %)</i>	$\rho$ %	$p$ %	<i>False Clauses Weight (<math>\nu(\phi)</math>)</i>
BFW-50	BFW-50-100	50	100	554	7.76	51.31	57.68	16
	BFW-50-150	50	150	800	11.18	49.91	50.66	34
	BFW-50-200	50	200	1 103	15.18	51.11	44.03	69
	BFW-50-250	50	250	1 361	18.94	50.68	45.52	96
	BFW-50-300	50	300	1 634	21.06	49.98	36.85	132
	BFW-50-350	50	350	1 936	24.57	48.80	38.95	211
	BFW-50-400	50	400	2 204	27.51	53.01	33.96	211
	BFW-50-450	50	450	2 519	30.53	52.09	36.18	257
	BFW-50-500	50	500	2 820	33.88	48.74	29.58	318
BFW-100	BFW-100-200	100	200	1 103	3.94	48.71	65.70	7
	BFW-100-300	100	300	1 634	5.92	51.43	49.81	67
	BFW-100-400	100	400	2 204	7.90	52.17	43.89	119
	BFW-100-500	100	500	2 820	9.62	51.63	37.75	241
	BFW-100-600	100	600	3 369	11.47	49.17	39.59	266
BFW-150	BFW-150-300	150	300	1 634	2.65	50.80	57.71	24
	BFW-150-450	150	450	2 519	3.92	50.88	51.77	79
	BFW-150-600	150	600	3 369	5.23	50.43	44.44	189

### 3.5.2 Randomly generated MAX-2-SAT formulas

To increase the number of MAX-2-SAT problems, we have randomly generated 640 satisfiability formulas. The random generator of MAX-2-SAT problems can create formulas with distinct characteristics, such as: high or low frequency of unit clauses (i.e., clauses with one literal); high or low density (i.e., the probability of any two literals to belong to a quadratic clause); high or low impurity (i.e., the ratio of the number of quadratic clauses with exactly one complemented literal and all quadratic clauses); high or low frequency of biterms (i.e., sets of two distinct quadratic clauses involving the same two variables).

As we have mentioned in the previous subsection, a MAX-2-SAT problem can be solved by optimizing a quadratic posiform  $\phi$  (see (2.7) in Section 2.3), for which the minimum value  $\nu(\phi)$  is the minimum (weighted) size of a set with false clauses. Computationally, it is simple to obtain the *unique* multilinear quadratic pseudo-Boolean function  $f$  associated to a *quadratic* posiform  $\phi$ . A quadratic term  $x_i x_j$  has a nonzero coefficient in  $f$  if and only if there is a term in  $\phi$  containing literals involving the same variables  $x_i$  and  $x_j$ , and consequently if there is a clause with literals of these two variables. Therefore, a quadratic clause involving variables  $x_i$  and  $x_j$ , results in one out of the following six cases, in a nonzero term of  $\phi$ :  $x_i x_j$ ,  $\bar{x}_i x_j$ ,  $x_i \bar{x}_j$ ,  $\bar{x}_i \bar{x}_j$ ,  $x_i x_j + \bar{x}_i \bar{x}_j$  and  $\bar{x}_i x_j + x_i \bar{x}_j$ . Unit clauses involving variable  $x_i$  are result of terms involving a single literal of this variable, i.e.  $x_i$  and  $\bar{x}_j$ .

The input parameters of the MAX-2-SAT generator are: the number of variables  $n$ ; the cumulative distribution of the linear terms (including a probability of nonexistence) for all variables  $x_i, i = 1, \dots, n$ ; the cumulative distribution of the possible cases of quadratic terms (including a probability of nonexistence) for all pairs of variables with indices  $1 \leq i < j \leq n$ , the lower and upper bounds of the clause weights, and a seed to initiate the generator of random numbers.

Table 3.21 list the eight profiles of probability parameters that were used in this study. For instance, profiles 2, 4 and 6 generate *dense* formulas, whereas 7 and 8 generate sparser formulas. The number of negated literals in a formula generated with

profiles 5-to-8 is approximately the same number of nonnegated literals in the same formula. Profiles 1 and 2 generate formulas with considerably more negated literals than nonnegated ones, whereas 3 and 4 generate formulas with the reverse role.

Table 3.21: Profiles of probabilities for a clause to belong to a (weighted) MAX-2-SAT formula.

Profile ID	Unit clauses, $1 \leq i \leq n$			Quadratic clauses, $1 \leq i < j \leq n$						
	Inex.	$x_i$	$\bar{x}_i$	Inex.	$x_i x_j$	$\bar{x}_i x_j$	$x_i \bar{x}_j$	$\bar{x}_i \bar{x}_j$	$x_i x_j + \bar{x}_i \bar{x}_j$	$\bar{x}_i x_j + x_i \bar{x}_j$
1	0.50	0.25	0.25	0.60	0.05	0.10	0.10	0.05	0.02	0.08
2	0.50	0.25	0.25	0.40	0.05	0.10	0.10	0.05	0.10	0.20
3	0.50	0.25	0.25	0.60	0.10	0.05	0.05	0.10	0.08	0.02
4	0.50	0.25	0.25	0.40	0.10	0.05	0.05	0.10	0.10	0.20
5	0.50	0.25	0.25	0.60	0.05	0.05	0.05	0.05	0.10	0.10
6	0.50	0.25	0.25	0.30	0.10	0.10	0.10	0.10	0.15	0.15
7	0.50	0.25	0.25	0.80	0.02	0.02	0.02	0.02	0.06	0.06
8	0.50	0.25	0.25	0.80	0.04	0.04	0.04	0.04	0.02	0.02

The list of MAX-2-SAT problems that we have randomly generated include formulas with 50, 100, 200 and 400 variables. Each set of formulas with the same number of variables has five instances for each one of the eight profiles of probability distributions shown in Table 3.21. Table 3.22 displays the different sub-families that we have randomly generated, and it includes some statistics about the number of clauses generated in the different categories.

The best known solutions found by the proposed methods are given in Table A.9 of the Appendix. Interesting to be noted that the class of harder MAX-2-SAT instances for our methods belong to profiles 3 and 7, followed closely by profiles 4 and 8.

Recently this generator of MAX-2-SAT formulas has been used to create “very” hard small instances, which were considerably more difficult to be solved than other instances, similar in size, that were created by other random generators ([148]).

Table 3.22: Randomly generated (weighted) MAX-2-SAT formulas.

<i>Family</i>	<i>Sub-Family</i>	<i>Variables (n)</i>	<i>Number Problems</i>	<i>Clause Weights</i>	<i>Clauses number</i>		
					<i>min</i>	<i>avg</i>	<i>max</i>
SAT	SAT-50	50	40	1	242	557.4	892
	SAT-100	100	40	1	991	2 214.0	3 554
	SAT-200	200	40	1	3 986	8 808.4	14 136
	SAT-400	400	40	1	16 020	35 142.8	56 103
WSAT-[1,10]	WSAT-50-[1,10]	50	40	[1,10]	251	558.4	906
	WSAT-100-[1,10]	100	40	[1,10]	997	2 214.5	3 530
	WSAT-200-[1,10]	200	40	[1,10]	3 954	8 800.6	14 115
	WSAT-400-[1,10]	400	40	[1,10]	16 016	35 118.8	56 194
WSAT-[1,100]	WSAT-50-[1,100]	50	40	[1,100]	254	558.7	891
	WSAT-100-[1,100]	100	40	[1,100]	988	2 218.5	3 531
	WSAT-200-[1,100]	200	40	[1,100]	3 960	8 810.2	14 085
	WSAT-400-[1,100]	400	40	[1,100]	15 819	35 085.9	56 237
WSAT-[90,100]	WSAT-50-[90,100]	50	40	[90,100]	235	560.8	891
	WSAT-100-[90,100]	100	40	[90,100]	990	2 220.2	3 572
	WSAT-200-[90,100]	200	40	[90,100]	4 023	8 798.0	14 117
	WSAT-400-[90,100]	400	40	[90,100]	15 966	35 097.6	56 224

## Chapter 4

### Basic Tools and Concepts

This chapter describes a set of concepts, definitions and tools related to pseudo-Boolean optimization, which will be used throughout this dissertation.

The first section introduces the definition of strong and weak persistency following the same approach of Boros and Hammer [54]. Persistency is a property inherent to certain variables which can be removed from the function by fixing them at a known value, without changing the optimal value of the resulting function.

Section 4.2 introduces the first order partial derivatives of the functions and its inherent properties, like persistency, decomposition, local optimization and minimum and maximum values.

Section 4.3 presents the second order derivatives of Hammer and Hansen [121] and generalizes this concept further. This new type of derivatives is able to determine certain persistency property for a relation between two binary variables.

Section 4.4 introduces the concept of locotope, which is a polytope characterized by first order derivatives information. This polytope is able to enforce local optimality conditions and is useful if used in combination with linear programming techniques to solve QUBO.

Section 5.3 introduces the implication graph, an important concept that is able to represent order logical relations between the binary variables. This concept is extended in Chapter 5 to the implication network model to represent quadratic pseudo-Boolean functions.

Section 4.6 introduces some basic concepts about posiform minimization, including how to define a canonical representation of it, called the standard form.

The last section covers certain continuous extensions and related properties for

pseudo-Boolean optimization.

## 4.1 Persistency

The concept of *persistency* is needed in the discussion that follows. Before defining it, let us start by calling a *partial assignment*, to a binary vector  $\mathbf{y} \in \mathbb{B}^S$  corresponding to a subset  $S \subseteq \mathbf{V}$ . Further, for a subset  $S \subseteq \mathbf{V}$  of indices and a vector  $\mathbf{x} \in \mathbb{B}^n$ ,  $x[S] \in \mathbb{B}^S$  denotes the subvector corresponding to indices in  $S$ , i.e.  $\mathbf{x}[S] = (x_i | i \in S)$ . For a partial assignment  $\mathbf{y} \in \mathbb{B}^S$  and a vector  $\mathbf{x} \in \mathbb{B}^n$ , let the *switch* of  $\mathbf{x}$  by  $\mathbf{y}$  be the binary vector  $\mathbf{z}$  defined by

$$z_j = \begin{cases} x_j & \text{if } j \notin S \\ y_j & \text{if } j \in S, \end{cases}$$

and let us denote it by  $\mathbf{z} = \mathbf{x}[S \leftarrow \mathbf{y}]$ .

**Definition 4.1** ([54]). *Given a pseudo-Boolean function  $f$  and a partial assignment  $\mathbf{y} \in \mathbb{B}^S$ , we say that:*

- i) Strong persistency holds for  $f$  at  $\mathbf{y}$ , if for all  $\mathbf{x} \in \text{Argmin}_{\mathbb{B}^n}(f)$  we have  $x[S] = \mathbf{y}$ , i.e. if the restriction of all minimizing points of  $f$  to  $S$  coincide with the partial assignment  $\mathbf{y}$ .*
- ii) Weak persistency holds for  $f$  at  $\mathbf{y}$ , if  $x[S \leftarrow \mathbf{y}] \in \text{Argmin}_{\mathbb{B}^n}(f)$ , i.e. if a switch of a minimizing point of  $f$  by the partial assignment  $\mathbf{y}$  is an optimal point too.*

## 4.2 First order partial derivatives

For all indices  $i \in \mathbf{V}$ , the  $i$ th partial derivative by variable  $x_i$  of a pseudo-Boolean function  $f$  is given as

$$\begin{aligned} \Delta_i(\mathbf{x}) &\stackrel{\text{def}}{=} \frac{\partial f}{\partial x_i}(\mathbf{x}) \\ &= f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) - f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \end{aligned} \tag{4.1}$$

and its  $i$ th *residual* is given as

$$\Theta_i(\mathbf{x}) = f(\mathbf{x}) - x_i \Delta_i(\mathbf{x}). \quad (4.2)$$

The functions  $\Delta_i$  and  $\Theta_i$  are themselves pseudo-Boolean functions, which depend on all variables, but  $x_i$ . From (4.2), a pseudo-Boolean function  $f$  can be expressed as  $f(\mathbf{x}) = x_i \Delta_i(\mathbf{x}) + \Theta_i(\mathbf{x})$  for any given variable  $x_i$ ,  $i \in \mathbf{V}$ . The following proposition uses this expression to prove some necessary conditions of optimality for the minimizing points of  $f$ .

**Proposition 4.1** ([121, 130]). *Let  $f$  be a pseudo-Boolean function. Every minimizing point  $\mathbf{x} \in \text{Argmin}_{\mathbb{B}^n}(f)$  satisfies*

$$(2x_i - 1) \Delta_i(\mathbf{x}) \leq 0, \quad (4.3)$$

for all indices  $i \in \mathbf{V}$ .

*Proof.* Let us consider an arbitrary minimizer  $\mathbf{x} \in \text{Argmin}_{\mathbb{B}^n}(f)$  of function  $f$ , and consider the partial assignments  $(b) \in \mathbb{B}^{\{i\}}$  and  $(1 - b) \in \mathbb{B}^{\{i\}}$ , where  $b$  is a binary value. First let us remark, that either  $\mathbf{x}[\{i\} \leftarrow (b)] \in \text{Argmin}_{\mathbb{B}^n}(f)$  or  $\mathbf{x}[\{i\} \leftarrow (1 - b)] \in \text{Argmin}_{\mathbb{B}^n}(f)$ . If only one of these two cases is verified then strong persistency holds at the corresponding partial assignment. If both cases are verified then weak persistency holds at these two partial assignments. Using (4.2) one easily derives

$$\begin{aligned} f(\mathbf{x}[\{i\} \leftarrow (b)]) - f(\mathbf{x}[\{i\} \leftarrow (1 - b)]) &= (b \Delta_i(\mathbf{x}) - (1 - b) \Delta_i(\mathbf{x})) \\ &= (2b - 1) \Delta_i(\mathbf{x}). \end{aligned}$$

Without loss of generality  $x_i = b$  can be assumed. This fact implies that the assertion is true. Let us also note that if strong persistency holds at one of the partial assignments, then the previous relation (assuming  $x_i = b$ ) is strictly negative. If instead weak persistency holds, then the previous relation is zero.  $\square$

Trivial consequences of the last proposition are the following two corollaries.

**Corollary 4.1.** *Let  $f$  be a pseudo-Boolean function, and let  $S \supseteq \text{Argmin}_{\mathbb{B}^n}(f)$ . Then, if for some  $i \in \mathbf{V}$ ,  $\Delta_i(\mathbf{y}) > 0$  ( $< 0$ ) for all  $\mathbf{y} \in S$ , then all minimizing points  $\mathbf{x} \in \text{Argmin}_{\mathbb{B}^n}(f)$  must have  $x_i = 0$  ( $= 1$ ).*

**Corollary 4.2.** *Let  $f$  be a pseudo-Boolean function, and let  $S \subseteq \mathbb{B}^n$  such that  $\text{Argmin}_S(f) \cap \text{Argmin}_{\mathbb{B}^n}(f) \neq \emptyset$ . Then, if for some  $i \in \mathbf{V}$ ,  $\Delta_i(\mathbf{y}) \geq 0$  ( $\leq 0$ ) for all  $\mathbf{y} \in S$ , then there exists at least a minimizing point  $\mathbf{x} \in \text{Argmin}_{\mathbb{B}^n}(f)$  with  $x_i = 0$  ( $= 1$ ).*

The usefulness of the previous corollaries depends on the set  $S$ . Typically  $S = \mathbb{B}^n$  is used, but the finding of more persistencies usually depends on the size of  $S$  being as small as possible.

Specializing the notations given earlier (4.1) to the general case, the  $i$ th partial derivative ( $i \in \mathbf{V}$ ) of a quadratic pseudo-Boolean function is given by

$$\Delta_i(x_1, \dots, x_n) = c_i + \sum_{j=1}^{i-1} c_{ij}x_j + \sum_{j=i+1}^n c_{ij}x_j. \quad (4.4)$$

The derivative functions  $\Delta_i$  given in (4.4) are *linear* pseudo-Boolean functions, whose minimum and maximum values are denoted as

$$\begin{aligned} L_i &\stackrel{\text{def}}{=} v(\Delta_i) = c_i + \sum_{\substack{j=1 \\ c_{ji} < 0}}^{i-1} c_{ji} + \sum_{\substack{j=i+1 \\ c_{ij} < 0}}^n c_{ij}, \text{ and} \\ U_i &\stackrel{\text{def}}{=} \tau(\Delta_i) = c_i + \sum_{\substack{j=1 \\ c_{ji} > 0}}^{i-1} c_{ji} + \sum_{\substack{j=i+1 \\ c_{ij} > 0}}^n c_{ij}, \end{aligned} \quad (4.5)$$

for all  $i \in \mathbf{V}$ .

It is simple to note that

$$\Delta_i\left(\frac{1}{2}, \dots, \frac{1}{2}\right) = \frac{U_i + L_i}{2},$$

for all  $i \in \mathbf{V}$ .

### 4.3 Second order derivatives

Similarly to the single variable case (4.2), one can express a quadratic pseudo-Boolean function as

$$f(\mathbf{x}) = x_i \Delta_i(\mathbf{x}) + x_j \Delta_j(\mathbf{x}) - x_i x_j c_{ij} + \varphi_{ij}(\mathbf{x}), \quad (4.6)$$

for all pairs of variables  $(i, j)$  ( $1 \leq i < j \leq n$ ), where  $\varphi_{ij}$  is the residual part of  $f$  not containing terms involving both the  $i^{\text{th}}$  and  $j^{\text{th}}$  variables.

Let  $b \in \mathbb{B}$ , then the following relations

$$\begin{aligned} f(\mathbf{x}[\{i, j\} \leftarrow (b, \bar{b})]) &= b \Delta_i(\mathbf{x}[\{j\} \leftarrow (\bar{b})]) + \bar{b} \Delta_j(\mathbf{x}[\{i\} \leftarrow (b)]) + \varphi_{ij}(\mathbf{x}) \\ f(\mathbf{x}[\{i, j\} \leftarrow (b, b)]) &= b \Delta_i(\mathbf{x}[\{j\} \leftarrow (b)]) + b \Delta_j(\mathbf{x}[\{i\} \leftarrow (b)]) - b c_{ij} + \varphi_{ij}(\mathbf{x}) \end{aligned} \quad (4.7)$$

can easily be derived from (4.6), for all pairs of indices of variables  $(i, j)$  ( $1 \leq i < j \leq n$ ).

The following theorem provides some necessary conditions based on quadratic relations between variables, for quadratic pseudo-Boolean minimization problems.

**Theorem 4.1.** *Let  $f$  be a quadratic pseudo-Boolean function represented as expression (1.5). For all  $i$  and  $j$  ( $1 \leq i < j \leq n$ ):*

- i) If  $\Delta_j(\mathbf{x}) - c_{ij} x_i < 0$  or  $\Delta_i(\mathbf{x}) - c_{ij} x_j < 0$  or  $\Delta_i(\mathbf{x}) + \Delta_j(\mathbf{x}) + c_{ij}(1 - (x_i + x_j)) < 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then  $\bar{x}_i^* \bar{x}_j^* = 0$  for all  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$ ;*
- ii) If  $\Delta_j(\mathbf{x}) + c_{ij} \bar{x}_i < 0$  or  $\Delta_i(\mathbf{x}) - c_{ij} x_j > 0$  or  $\Delta_i(\mathbf{x}) - \Delta_j(\mathbf{x}) + (x_i - x_j) c_{ij} > 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then  $x_i^* \bar{x}_j^* = 0$  for all  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$ ;*
- iii) If  $\Delta_j(\mathbf{x}) - c_{ij} x_i > 0$  or  $\Delta_i(\mathbf{x}) + c_{ij} \bar{x}_j < 0$  or  $\Delta_i(\mathbf{x}) - \Delta_j(\mathbf{x}) + (x_i - x_j) c_{ij} < 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then  $\bar{x}_i^* x_j^* = 0$  for all  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$ ;*
- iv) If  $\Delta_j(\mathbf{x}) + c_{ij} \bar{x}_i > 0$  or  $\Delta_i(\mathbf{x}) + c_{ij} \bar{x}_j > 0$  or  $\Delta_i(\mathbf{x}) + \Delta_j(\mathbf{x}) + c_{ij}(1 - (x_i + x_j)) > 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then  $x_i^* x_j^* = 0$  for all  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$ .*

*Proof.* We shall prove the third case of (i) by using a contradiction. The other cases can be proved in a similar way. Suppose that there is a point  $\mathbf{x} \in \text{Argmin}_{\mathbb{B}^n}(f)$  satisfying  $\bar{x}_i \bar{x}_j = 1$ , i.e.  $x_i = x_j = 0$ , and that  $\Delta_i(\mathbf{x}) + \Delta_j(\mathbf{x}) + c_{ij}(1 - (x_i + x_j)) < 0$ . Since

$\mathbf{x}$  is a minimizer of  $f$  with  $x_i = 0$  and  $x_j = 0$ , then  $f(\mathbf{x}) = f(\mathbf{x}[\{i, j\} \leftarrow (0, 0)]) \leq f(\mathbf{x}[\{i, j\} \leftarrow (1, 1)])$ . Using the relations (4.7), a contradiction to our assumption is obtained as follows:

$$\begin{aligned} & f(\mathbf{x}[\{i, j\} \leftarrow (0, 0)]) \leq f(\mathbf{x}[\{i, j\} \leftarrow (1, 1)]) \\ \Rightarrow & f(\mathbf{x}[\{i, j\} \leftarrow (1, 1)]) - f(\mathbf{x}[\{i, j\} \leftarrow (0, 0)]) \geq 0 \\ \Rightarrow & \Delta_i(\mathbf{x}[\{j\} \leftarrow (1)]) + \Delta_j(\mathbf{x}[\{i\} \leftarrow (1)]) - c_{ij} \geq 0 \\ \Rightarrow & \Delta_i(\mathbf{x}) + \Delta_j(\mathbf{x}) + c_{ij}(1 - (x_i + x_j)) \geq 0. \end{aligned}$$

□

Hammer and Hansen [121] called to the linear function  $\Delta_i(\mathbf{x}) - \Delta_j(\mathbf{x}) + (x_i - x_j)c_{ij}$  the  $(i, j)$ th second order derivative of  $f$  and denote it by  $\Delta_{ij}$ . Theorem 4.1 shows that the linear function  $\Delta_i(\mathbf{x}) + \Delta_j(\mathbf{x}) + c_{ij}(1 - (x_i + x_j))$  has a similarly important role as  $\Delta_{ij}$ ; it will be called  $(i, j)$ th second order co-derivative and will be denoted by  $\nabla_{ij}$ .

**Example 4.1.** Consider the quadratic pseudo-Boolean function  $f_6$ . Since

$$\Delta_5(x_1, x_2, x_3, x_4, x_5, x_6) = 1 + 2x_1 - x_2 - 2x_3 + 2x_4 + 2x_6$$

$$\Delta_6(x_1, x_2, x_3, x_4, x_5, x_6) = -1 - x_1 + x_2 + x_3 - x_4 + 2x_5,$$

then

$$\nabla_{56}(x_1, x_2, x_3, x_4, x_5, x_6) = 2 + x_1 - x_3 + x_4.$$

From Theorem 4.1.(iv), because  $\nabla_{56}(\mathbf{x}) > 0$  for all  $\mathbf{x} \in \mathbb{B}^n$ , then relation  $x_5x_6 = 0$  follows for all minimizers of  $f_6$ .

The other conclusions of Theorem 4.1, which are not related to the  $(i, j)$ th second order derivatives, can be derived from a two-stage process involving an analysis of the first derivatives. In the first stage a variable  $i$  is assumed to have a binary value  $v$ . In practice, this step results in a new function  $f'$  with one less variable, so that in the second stage a first derivative analysis can be made in  $f'$ . Let us assume that variable  $j$  in this second stage is strongly persistent with value  $u$ . Then, a quadratic

relation satisfied by the minimizers of  $f$  is of the form  $x_i = v \Rightarrow x_j = v$ . Let us remark the implications in Theorem 4.1 involving  $\Delta_{ij}$  and  $\nabla_{ij}$  cannot be obtained simply by looking at first derivative conclusions, as is done in the two-stage process explained before.

Trivial consequence of Theorem 4.1 is the following corollary presenting weaker conditions for the existence of persistencies.

**Corollary 4.3.** *Let  $f$  be a quadratic pseudo-Boolean function represented as expression (1.5). For all  $i$  and  $j$  ( $1 \leq i < j \leq n$ ):*

- i) If  $\Delta_j(\mathbf{x}) - c_{ij}x_i \leq 0$  or  $\Delta_i(\mathbf{x}) - c_{ij}x_j \leq 0$  or  $\Delta_i(\mathbf{x}) + \Delta_j(\mathbf{x}) + c_{ij}(1 - (x_i + x_j)) \leq 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then there exists a  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$  such that  $\bar{x}_i^* \bar{x}_j^* = 0$  also holds;*
- ii) If  $\Delta_j(\mathbf{x}) + c_{ij}\bar{x}_i \leq 0$  or  $\Delta_i(\mathbf{x}) - c_{ij}x_j \geq 0$  or  $\Delta_i(\mathbf{x}) - \Delta_j(\mathbf{x}) + (x_i - x_j)c_{ij} \geq 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then there exists a  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$  such that  $x_i^* \bar{x}_j^* = 0$  also holds;*
- iii) If  $\Delta_j(\mathbf{x}) - c_{ij}x_i \geq 0$  or  $\Delta_i(\mathbf{x}) + c_{ij}\bar{x}_j \leq 0$  or  $\Delta_i(\mathbf{x}) - \Delta_j(\mathbf{x}) + (x_i - x_j)c_{ij} \leq 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then there exists a  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$  such that  $\bar{x}_i^* x_j^* = 0$  also holds;*
- iv) If  $\Delta_j(\mathbf{x}) + c_{ij}\bar{x}_i \geq 0$  or  $\Delta_i(\mathbf{x}) + c_{ij}\bar{x}_j \geq 0$  or  $\Delta_i(\mathbf{x}) + \Delta_j(\mathbf{x}) + c_{ij}(1 - (x_i + x_j)) \geq 0$  holds for all  $\mathbf{x} \in \mathbb{B}^n$ , then there exists a  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$  such that  $x_i^* x_j^* = 0$  also holds.*

**Example 4.2.** *Consider the quadratic pseudo-Boolean function  $f_6$ . If  $x_4 = 0$  holds in a minimizer of  $f_6$ , then because*

$$\Delta_1(x_1, x_2, x_3, 0, x_5, x_6) = 2 - x_2 + 2x_3 + 2x_5 - x_6 \geq 0, \text{ for all } \mathbf{x} \in \mathbb{B}^6,$$

*the quadratic relation  $x_1 \bar{x}_4 = 0$  must hold in at least one minimizer of  $f_6$ .*

#### 4.4 Locotope

Binary vectors, no single component of which can be changed so as to decrease the value of a pseudo-Boolean function  $f$ , are called *local minima* of  $f$ . It should be noted that the number of local minima can be exponentially large ([191]) and that the computational complexity of finding a local minimum of a quadratic pseudo-Boolean function is open (see e.g. [194]).

**Proposition 4.2.** *Given a quadratic pseudo-Boolean function  $f$ , a binary vector  $\mathbf{x} \in \mathbb{B}^n$  is a local minimum of  $f$  if and only if*

$$(2x_i - 1) \Delta_i(\mathbf{x}) \leq 0,$$

for all indices  $i \in \mathbf{V}$ .

*Proof.* Expressing  $f$  as (4.2) and noticing that neither  $\Delta_i$  nor  $\Theta_i$  depend on  $x_i$ , the statement follows readily.  $\square$

Together, Propositions 4.1 and 4.2 show that a minimizer  $\mathbf{x} \in \text{Argmin}_{\mathbb{B}^n}(f)$  is also a local minimum of  $f$ . But, the reverse is not true in general. Let us denote by  $\mathbb{M}$  the set of local minima of  $f$ .

**Proposition 4.3.** *For every index  $i$ ,  $i = 1, \dots, n$ , let  $U_i$  and  $L_i$  be respectively the minimum and the maximum values of the first derivative  $\Delta_i$  function, as were defined in (4.5). Then, the set  $\mathbb{M}$  of local minima of a pseudo-Boolean function  $f$  is given by*

$$\mathbb{M} = \{\mathbf{x} \in \mathbb{B}^n : L_i x_i \leq \Delta_i(\mathbf{x}) \leq U_i \bar{x}_i, i = 1, \dots, n\}.$$

*Proof.* If  $x_i = 0$  then  $0 \leq \Delta_i(\mathbf{x}) \leq U_i$  and hence  $(2x_i - 1) \Delta_i(\mathbf{x}) \leq 0$  holds. If  $x_i = 1$  then  $L_i \leq \Delta_i(\mathbf{x}) \leq 0$  and  $(2x_i - 1) \Delta_i(\mathbf{x}) \leq 0$  also holds.  $\square$

The usefulness of the last proposition comes from the fact that it characterizes all local minima of a pseudo-Boolean function by using  $2n$  linear inequalities. It is also simple to verify that whenever  $U_i < 0$  ( $L_i > 0$ ), then  $x_i$  must be one (zero) in every

minimizing point of  $f$ . Further, if  $U_i \leq 0$  ( $L_i \geq 0$ ), then there is a minimizing point of  $f$  for which  $x_i$  is one (zero).

We denote by  $\mathbb{L} (\supseteq \mathbb{M})$  the polytope defined by relaxing the integrality property of the components of vectors in  $\mathbb{M}$ , i.e.

$$\mathbb{L} \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbb{U}^n : L_i x_i \leq \Delta_i(\mathbf{x}) \leq U_i \bar{x}_i, i = 1, \dots, n\}.$$

We shall name this very special polytope as the *standard locotope*. By employing linear programming techniques, it is possible to improve the lower and upper bounds of the first derivatives, while maintaining the property that it also contains the set of local minima  $\mathbb{M}$ . Let  $\mathbf{l}$  and  $\mathbf{u}$  be real  $n$ -vectors. Then

$$\mathbb{L}(\mathbf{l}, \mathbf{u}) \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbb{U}^n : l_i x_i \leq \Delta_i(\mathbf{x}) \leq u_i \bar{x}_i, i = 1, \dots, n\}$$

satisfies  $\mathbb{L} \subseteq \mathbb{L}(\mathbf{l}, \mathbf{u}) \subseteq \mathbb{M}$ .

Let us remark that getting “better” lower and upper bounds for the first derivative functions, may lead to the finding of new logical relations, which would not be found if these bounds were not improved.

Next, an algorithm that improves the minimum and maximum values of the first derivative functions over the locotope, is given. We named it as the *Locotope Tightening Algorithm* (or LTA in short). The LTA description is given in Figure 4.1. In this algorithm, the following linear program is called every time an individual bound is improved.

$$\text{opt } \Delta_k(\mathbf{x})$$

subject to

$$\mathbf{x} \in \mathbb{L}(\mathbf{l}, \mathbf{u}) \quad (\text{LP}(\text{opt}, k, \mathbf{l}, \mathbf{u}))$$

$$x_i = 0, \quad i \in \{j \in \mathbf{V} \mid l_j > 0\} \cup \{j \in \mathbf{V} \mid l_j = 0, u_j > 0\}$$

$$x_i = 1, \quad i \in \{j \in \mathbf{V} \mid u_j \leq 0\}$$

The objective function of this problem is the first derivative function  $\Delta_k$  associated

to variable  $x_k$ . To get the appropriate bound, this function is either maximized to get the upper bound (i.e.  $opt = max$ ), or minimized to get the lower bound (i.e.  $opt = min$ .) The decision space is the intersection of the polytope  $\mathbb{L}(\mathbf{l}, \mathbf{u})$  with the current bounds  $\mathbf{l}$  and  $\mathbf{u}$ , and a set of equations (possibly empty) for each variable that has a value fixed to 0 or 1, according to the current bounds  $\mathbf{l}$  and  $\mathbf{u}$ .

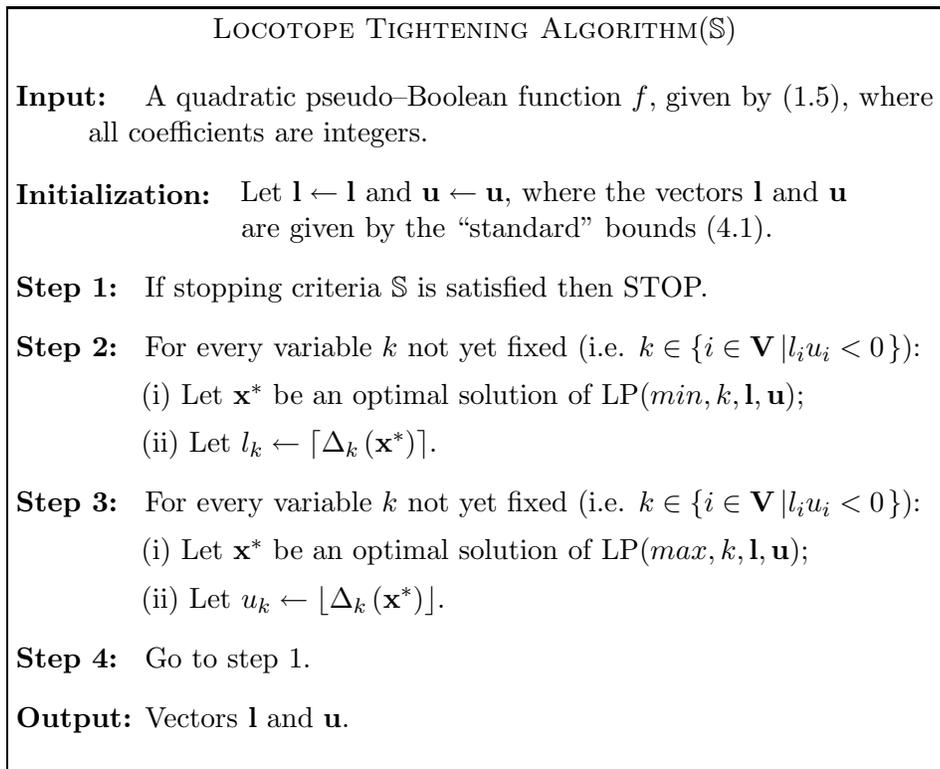


Figure 4.1: Locotope Tightening Algorithm (LTA).

Note that due to the way  $LP(min, k, \mathbf{l}, \mathbf{u})$  is formulated, the locotope obtained in this way may exclude some optimal solutions of the original function, but at least one minimizer is guaranteed to belong to it.

As soon as a linear program is solved, the corresponding bound is compared with the old value, and updated if it is better. This last step also assumes that a quadratic pseudo-Boolean function has integer coefficients, implying that the bounds must be integral as well. This is not a restriction in practice since a QUBO problem can be brought into this condition by scaling the function by a enough large factor.

The sequence of linear programs to be solved may be different than the one presented

in Figure 4.1. Let us note that the improvement in the LTA is first given to the lower bounds, and only after to the upper bounds. However, an alternating sequence between maximization and minimization programs is also possible. Let us also note that consecutive programs have constraints almost identical, typically only different in one coefficient corresponding to the most recent improvement found for a bound. This characteristic may be exploited to obtain a more efficient algorithm. For instance, if a simplex method is adopted to solve the linear program, then a basic feasible solution in one improvement step may be used in the following simplex iteration. Another possibility is to keep the constraints unchanged (with the new bounds found) until a persistency is found, or until a prescribed number of simplex calls is made.

The stopping criterion  $\mathbb{S}$  in the LTA routine, can be a condition to test if no improvement is possible for each bound. To satisfy this condition, the number of times that steps 2 and 3 are executed may be very large. Thus, a stopping criterion based on the maximum number of calls to steps 2 and 3 can be used in addition to the improvement condition. This maximum number of calls may also depend on the fact that a variable was fixed to 1 or 0 in the last execution of step 2 or step 3.

**Example 4.3.** *Consider the quadratic pseudo-Boolean function  $f_{10}$ . A call to the LTA routine using  $f_{10}$  as input, produced no persistencies. All possible improvements of the bounds were found in one execution of steps 2 and 3. The “standard bounds” of the first derivatives of  $f_{10}$  are*

$$\begin{aligned} \mathbf{l} &= (-5, -3, -5, -8, -5, -6, -5, -5, -8, -5) \quad \text{and} \\ \mathbf{u} &= (7, 8, 8, 6, 10, 6, 6, 6, 6, 12), \end{aligned}$$

*and the improved bounds returned by the output of LTA are*

$$\begin{aligned} \mathbf{l}' &= (-4, -2, -5, -7, -3, -6, -4, -4, -5, -3) \quad \text{and} \\ \mathbf{u}' &= (7, 6, 6, 6, 8, 4, 5, 4, 5, 8). \end{aligned}$$

## 4.5 Implication graph

Given a set of logical relations, it is desirable to derive all logical consequences of it. In particular, we are interested in deriving logical conclusions from a set of quadratic relations between two binary (Boolean) variables. In the present study, the relevance of analyzing such a set of quadratic relations is that they may imply persistent values for the individual variables, or to other logical quadratic relations. In practice, if all minimizers of function  $f$  satisfy the original set of relations, then they must also satisfy the implied conditions.

We represent a quadratic relation between two literals  $u, v \in \mathbf{L}$  by *elementary Boolean equations* of the type  $uv = 0$ , which is an implication meaning that both literals must have complemented values (e.g., if  $u = 1$  then  $v = 0$ ) if they are distinct (quadratic), and they must have value zero if  $u$  and  $v$  refer to the same literal (linear).

**Example 4.4.** *Knowing that the quadratic relations  $x_1^* \bar{x}_2^* = 0$ ,  $x_2^* \bar{x}_3^* = 0$  and  $x_3^* \bar{x}_1^* = 0$  hold for all minimizers  $\mathbf{x}^* \in \text{Argmin}(f)$ , then  $\mathbf{x}^*$  must satisfy the conditions  $x_1^* = x_2^* = x_3^*$ . From these equality relations between variables, a new (quadratic) function  $f' : \mathbb{B}^{V \setminus \{x_1, x_2\}} \leftarrow \mathbb{R}$  with two less variables, can easily be obtained from  $f$ , such that  $f'(x_3, x_4, \dots, x_n) = f(x_3, x_3, x_3, x_4, \dots, x_n)$ , thus implying that a weak persistency holds for  $f$  at  $\mathbf{y}$ , where  $\mathbf{y} \in \text{Argmin}_{\mathbb{B}^{V \setminus \{x_1, x_2\}}}(f')$ .*

A *quadratic Boolean equation* is a system of elementary (linear and quadratic) equations. The name “equation” for this system is related to the fact that it can be represented as the disjunction (operator  $\vee$ , where  $u \vee v = \max(u, v)$ ) of a subset  $\mathcal{Q} \subseteq \mathbf{L} \times \mathbf{L}$  of pairs of literals and a subset  $\mathcal{L} \subseteq \mathbf{L}$  of literals, i.e.

$$\bigvee_{u \in \mathcal{L}} u \vee \bigvee_{u, v \in \mathcal{Q}} uv = 0. \quad (4.8)$$

**Definition 4.2.** *A Boolean equation is consistent (or satisfiable) if and only if there is a (partial) assignment  $\mathbf{y} \in \mathbb{B}^S$ ,  $S \subseteq V$  (or solution) that satisfies all elementary equations on it.*

The consistency of a quadratic Boolean equation, and a solution to it (if any), can be

carried out by using polynomial time algorithms (see e.g. [133]). Notice that the total number of solutions of a consistent quadratic Boolean equation may be exponentially large. Therefore, in some situations it is useful to produce a parametric solution of it (see e.g. [90]).

In this study we give preference to the Strong Components Algorithm (or SCA in short) of Aspvall, Plass and Tarjan [22]. The SCA algorithm exploits a digraph model called the implication graph. In the exposition that follows, a quadratic Boolean equation  $\Phi = 0$  expressed as (4.8) is considered, where without loss of generality, it can be assumed that  $\mathcal{L} = \emptyset$ .

The *implication graph* associated with  $\Phi$  is the digraph  $D = (\mathbf{L}, A)$ , where

$$A = \{(u, \bar{v}), (\bar{u}, v) \mid (u, v) \in \mathcal{Q}\}.$$

The digraph  $D$  is isomorphic to the digraph  $\tilde{D}$  obtained from  $D$  by reversing the orientation of every arc and complementing every literal.

The SCA is based on the following key result.

**Proposition 4.4** ([22]). *The equation  $\Phi = 0$  is consistent if and only if in the implication graph  $D$  no literal  $u \in \mathbf{L}$  is in the same strong component as its complement  $\bar{u}$ .*

The algorithm works on  $D$  and finds the strong components of  $d$  in reverse topological order. The isomorphism between  $D$  and  $\tilde{D}$  implies that for every strong component  $C$  of  $D$  there exists a “mirror” component  $\tilde{C}$  of  $D$ , called the *dual* of  $C$ , induced by the complements of the literals in  $C$ . Hence, Proposition 4.4 implies that  $\Phi = 0$  is satisfiable if and only if  $C \neq \tilde{C}$  for all  $C$  of  $D$ .

The implication nature of this graph comes from the fact that for any two literals  $u$  and  $v$ ,  $uv = 0$  if and only if  $u \leq \bar{v}$  (or equivalently  $v \leq \bar{u}$ .) A vertex  $v \in \mathbf{L}$  is said to be a *predecessor* (*sucessor*) of  $u \in \mathbf{L}$  if there is a path in  $D$  from  $v$  ( $u$ ) to  $u$  ( $v$ ). Having this in mind, the following facts are immediate.

**Lemma 4.1.** *Let  $D = (\mathbf{L}, A)$  be the implication graph of a quadratic Boolean equation*

$\Phi = 0$ . Then,

- i) If for all literals  $u \in \mathbf{L}$ ,  $u$  is not both a predecessor and a successor of its complement  $\bar{u}$ , then  $\Phi = 0$  is consistent;
- ii) If  $\Phi = 0$  is consistent, and  $u \in \mathbf{L}$  is a predecessor of its complement  $\bar{u}$ , then  $u = 0$ ;
- iii) If  $\Phi = 0$  is consistent, and there is a strong component  $C \in D$  involving more than one literal, then all literals  $u \in C$  must have the same value  $b$ , i.e.  $u = v = b$  for all  $u, v \in C$ ;
- iv) If  $\Phi = 0$  is consistent, and there is an arc going from the strong component  $C_u$  to the strong component  $C_v$ , then  $u\bar{v} = 0$  for all  $u \in C_u$  and  $v \in C_v$ ;
- v) If  $\Phi = 0$  is consistent in a solution  $\mathbf{x}$  with a literal  $u \in \mathbf{L}$  having value one ( $u(\mathbf{x}) = 1$ ), then all the successors  $v$  of  $u$  must also have value one ( $v(\mathbf{x}) = 1$ ).
- vi) If  $\Phi = 0$  is consistent in a solution  $\mathbf{x}$  with a literal  $u \in \mathbf{L}$  having value zero ( $u(\mathbf{x}) = 0$ ), then all the predecessors  $v$  of  $u$  must also have value zero ( $v(\mathbf{x}) = 0$ ).

Lemma 4.1 implies that  $\Phi = 0$  is inconsistent if and only if there is a cycle in  $D$  containing both a literal  $u$  and its complement  $\bar{u}$ . Parts (ii–iii) of Lemma 4.1 are rules that can be used to define persistencies on single variables, either by the existence of strong components with several literals, or simply because there is a path in  $D$  between a literal  $u$  and its complement  $\bar{u}$ .

We say that a strong component  $C$  in a digraph  $D$  is *condensed* if all the vertices in  $C$  are lumped together in a new vertex  $u_C$ , such that every outgoing arc  $(u, v)$ ,  $u \in C, v \in \mathbf{L} \setminus C$  is removed and replaced by an arc  $(u_C, v)$ , and every incoming arc  $(v, u)$ ,  $u \in C, v \in \mathbf{L} \setminus C$  is removed and replaced by an arc  $(v, u_C)$ . If  $|C| > 1$ , then this basic operation allows us to get an implication graph with fewer vertices, whose consistency and solutions (if any) also satisfy the initial equation  $\Phi = 0$ .

Let us call *condensation* to the digraph  $D^*$  obtained from digraph  $D$  by condensing all of its strong components. Notice that  $D^*$  is an acyclic digraph, and consequently it contains a vertex with no incoming arcs, and it contains a vertex with no outgoing arcs.

We say that an implication graph  $D$  is in *normal form* if it is its condensation (i.e. if  $D = D^*$ ), and if there is no path between a literal  $u \in \mathbf{L}$  and its complement  $\bar{u}$ . Let us also denote by  $\phi_D$  to the left hand side of the quadratic Boolean equation associated to the implication graph  $D$ .

#### 4.6 Posiform minimization

Posiforms have interesting structural properties that can be explored towards finding logical relations between the literals, which imply a set of simple conditions that a minimizer of the associated function has to satisfy.

The most trivial result on posiforms is perhaps the fact that the minimum of a posiform  $\phi$  is bounded from below by its constant term  $C(\phi)$  (or  $a_\emptyset$ ), i.e.  $\nu(f_\phi) \geq C(\phi)$ . In fact, every pseudo-Boolean function  $f : \mathbb{B}^n \rightarrow \mathbb{R}$  can be represented by a posiform  $\phi$  for which  $C(\phi) = \min_{\mathbf{x} \in \mathbb{B}^n} f_\phi(\mathbf{x})$  (see e.g. [54]).

Another simple result is related to the concept of *pure* literals (sometimes also called *monotone* literals). A literal  $u$  is called pure in the posiform  $\phi$ , if it appears on it only in the positive form, or only in the complemented form. Pure literals define weak persistencies as follows.

**Lemma 4.2.** *If  $u$  is a pure literal in the posiform  $\phi$ , then there is a minimizer of  $f_\phi$  satisfying  $u = 0$ .*

Suppose that an upper bound  $z_\phi$  to the minimum of a posiform  $\phi$  is known. Then the following results follow trivially.

**Lemma 4.3.** *Let  $\phi$  be a posiform represented as (1.2). If  $\nu(f_\phi) \leq z_\phi < C(\phi) + a_T$  for a given non-empty subset of literals  $T \subseteq \mathbf{L}$ , then the condition  $\prod_{u \in T} u = 0$  must be satisfied by all minima of  $f_\phi$ .*

*Proof.* Let  $\mathbf{x}^*$  be a minimizer of  $f_\phi$ . Since  $z_\phi \geq \nu(f_\phi)$ , then  $z_\phi \geq f_\phi(\mathbf{x}^*)$ , or

$$z_\phi \geq \sum_{S \subseteq \mathbf{L}} a_S \prod_{u \in S} u^* \Rightarrow z_\phi \geq C(\phi) + a_T \prod_{u \in T} u^*.$$

To avoid a contradiction with the assumption  $z_\phi < C(\phi) + a_T$ , the minimizer indeed has to satisfy  $\prod_{u \in T} u^* = 0$ .  $\square$

**Example 4.5.** A quadratic posiform that represents the quadratic pseudo-Boolean function  $f_6$  is

$$\begin{aligned} \phi_{f_6} = & -5 + \bar{x}_6 + 2x_1x_3 + 2x_1\bar{x}_4 + 2x_1x_5 + \bar{x}_1x_2 + \bar{x}_1x_6 \\ & + x_2x_3 + x_2x_6 + \bar{x}_2x_4 + \bar{x}_2x_5 \\ & + x_3x_6 + 2\bar{x}_3\bar{x}_4 + 2\bar{x}_3x_5 \\ & + 2x_4x_5 + \bar{x}_4x_6 \\ & + 2\bar{x}_5\bar{x}_6. \end{aligned}$$

From this posiform, one gets  $\nu(f_6) \geq -5$ . Since  $f_6(1, 0, 0, 1, 0, 1) = -4$ , then we can set  $z_\phi = -4$  and thus by Lemma 4.3 we can conclude that the following quadratic Boolean equation must be satisfied by any minimizer of  $f_6$ :

$$x_1x_3 \vee x_1\bar{x}_4 \vee x_1x_5 \vee \bar{x}_3\bar{x}_4 \vee \bar{x}_3x_5 \vee x_4x_5 \vee \bar{x}_5\bar{x}_6 = 0.$$

**Lemma 4.4.** Let  $\phi$  be a quadratic posiform represented as (1.6). If  $\nu(f_\phi) \leq z_\phi < C(\phi) + \min(a_{uv}, a_{\bar{u}\bar{v}})$  for any two distinct literals  $u, v \in \mathbf{L}$ , then the condition  $u = \bar{v}$  must be satisfied by all minima of  $f_\phi$ .

*Proof.* By Lemma 4.3  $uv = 0 \wedge \bar{u}\bar{v} = 0$ , or similarly  $uv \vee \bar{u}\bar{v} = 0$ .  $\square$

#### 4.6.1 Standard quadratic posiforms

Through simple algebraic manipulations, any given quadratic posiform can be transformed to an equivalent quadratic posiform in *standard* form, whose nonzero terms satisfy some conditions as follows.

**Definition 4.3.** A quadratic posiform  $\phi$  of form (1.6) is in standard form if and only if  $a_{uv}a_{\bar{u}\bar{v}} = 0$ ,  $a_{uv}a_{u\bar{v}} = 0$  and  $a_u a_{\bar{u}} = 0$  for all literals  $u, v \in \mathbf{L}$ .

Next, we show that any posiform  $\phi \in \mathcal{P}_2(f)$  can be efficiently transformed into a standard posiform  $\phi' \in \mathcal{P}_2(f)$ . In Figure 4.2 the Standard Posiform Algorithm (or SPA

in short) to obtain a standard posiform is described. The SPA can be implemented to run in polynomial time in the size of  $\phi$ , and produces a posiform  $\phi'$  in the same variables, the size of which is not larger than  $\text{size}(\phi)$ .

In the sequel, we shall assume that every posiform associated to a network model is in standard form. It should be remarked that it is trivial to provide a standard posiform corresponding to any quadratic pseudo-Boolean function expressed as the multilinear polynomial (1.1). Furthermore, several combinatorial problems (e.g., MAX-CUT, MAX-2-SAT) have a natural representation as an optimization problem of a standard posiform.

#### 4.7 Rounding procedures and derandomization

The multi-linear expression (1.1) of a pseudo-Boolean function  $f : \mathbb{B}^n \mapsto \mathbb{R}$  can be used as well to characterize a function  $g : \mathbb{D}^n \mapsto \mathbb{R}$ , whose domain  $\mathbb{D}$  is not restricted to the case where all variables are binary (i.e. where  $\mathbb{D} = \mathbb{B}$ ). Obviously if  $\mathbb{B}$  is a subset of  $\mathbb{D}$  then the minimum of function  $f$  can not be smaller than that of  $g$  in the respective domains.

**Lemma 4.5.** *Let  $f : \mathbb{B}^n \mapsto \mathbb{R}$  and  $g : \mathbb{D}^n \mapsto \mathbb{R}$  be two functions. If  $\mathbb{B} \subseteq \mathbb{D}$ , then  $\nu(f) \geq \min \{g(\mathbf{x}) \mid \mathbf{x} \in \mathbb{D}\}$ .*

In particular, we shall see later on this section that if  $\mathbb{D} = \mathbb{U}$  then the previous result is gap free. In order to make it clear that two functions (maybe having different domains) have a common expression (1.1) we shall represent it with the same name, providing when necessary the domain to make this fact clear.

**Lemma 4.6.** *Let  $f$  be a pseudo-Boolean function given by (1.1), and let  $\mathbf{r} \in \mathbb{U}^n$ . Then,*

$$f(r_1, \dots, r_{i-1}, r_i, r_{i+1}, \dots, r_n) = r_i f(r_1, \dots, r_{i-1}, 1, r_{i+1}, \dots, r_n) + (1 - r_i) f(r_1, \dots, r_{i-1}, 0, r_{i+1}, \dots, r_n),$$

for every  $i = 1, \dots, n$ .

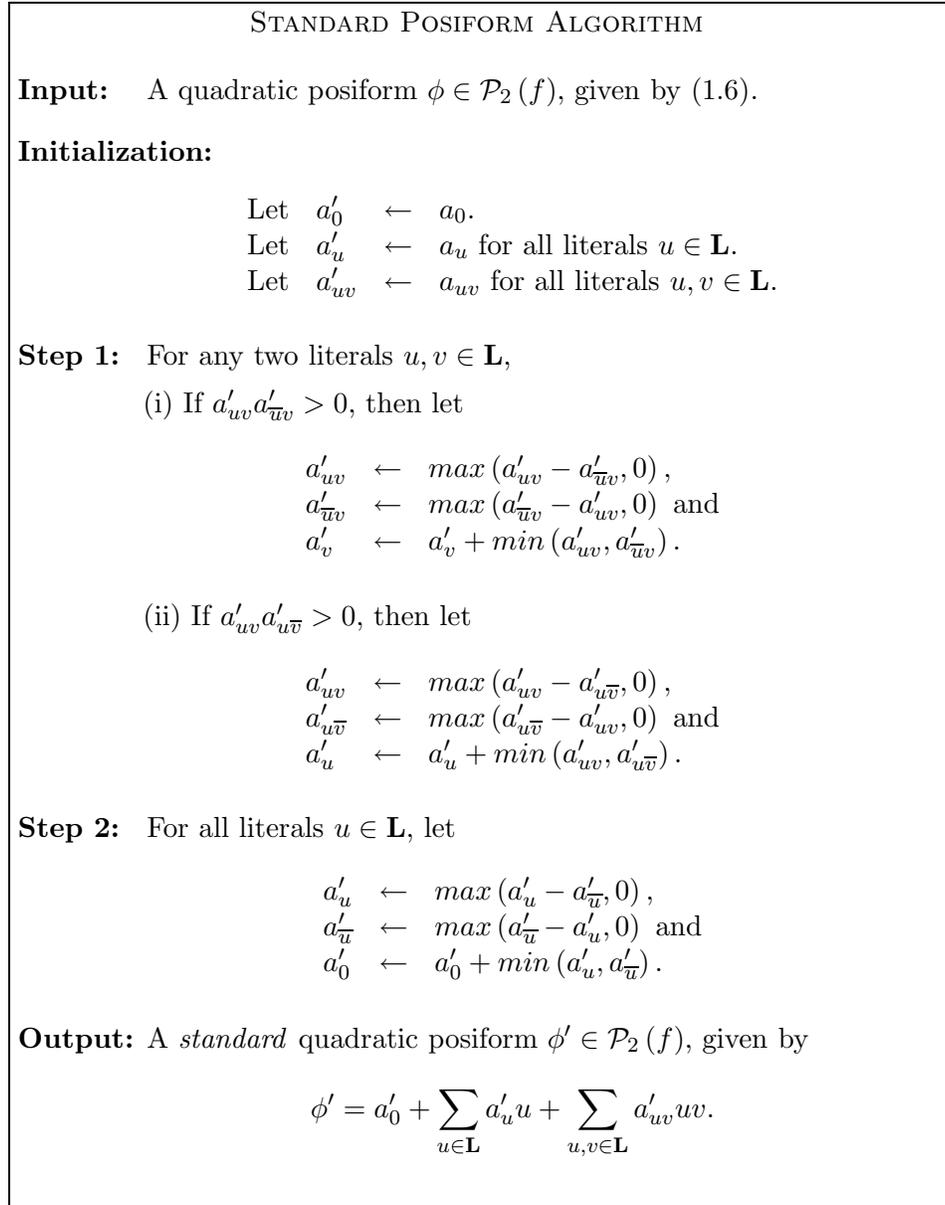


Figure 4.2: Standard posiform algorithm.

*Proof.* From (4.2),

$$f(\mathbf{r}) = r_i \Delta_i(\mathbf{r}) + \Theta_i(\mathbf{r}). \quad (4.9)$$

Substituting

$$\begin{aligned} \Delta_i(\mathbf{r}) &= f(r_1, \dots, r_{i-1}, 1, r_{i+1}, \dots, r_n) - f(r_1, \dots, r_{i-1}, 0, r_{i+1}, \dots, r_n) \text{ and} \\ \Theta_i(\mathbf{r}) &= f(r_1, \dots, r_{i-1}, 0, r_{i+1}, \dots, r_n) \end{aligned}$$

in (4.9), then the claimed result follows immediately.  $\square$

Given  $r \in \mathbb{U}$  and  $x \in \mathbb{B}$ , then

$$r^x (1-r)^{(1-x)} = \begin{cases} r, & \text{if } x = 1, \\ 1-r, & \text{if } x = 0. \end{cases}$$

Assuming that  $0^0 = 1$ , then the above expression can be determined in the standard algebraic way. This assumption is used in the following results.

**Theorem 4.2.** *Let  $f$  be a pseudo-Boolean function given by (1.1), and let  $\mathbf{r} \in \mathbb{U}^n$ .*

*Then,*

$$f(r_1, \dots, r_n) = \sum_{\mathbf{x} \in \mathbb{B}^n} \left( \prod_{i=1}^n r_i^{x_i} (1-r_i)^{(1-x_i)} \right) f(x_1, \dots, x_n).$$

*Proof.* Let us prove this fact by induction on the number of variables  $k$ . If  $k = 1$  the claimed result follows immediately by Lemma 4.6. Let us now assume that the claim is valid for  $1 \leq k < n$ , i.e.

$$f(r_1, \dots, r_k, r_{k+1}, \dots, r_n) = \sum_{\mathbf{x} \in \mathbb{B}^k} \left( \prod_{i=1}^k r_i^{x_i} (1-r_i)^{(1-x_i)} \right) f(x_1, \dots, x_k, r_{k+1}, \dots, r_n).$$

From Lemma 4.6

$$\begin{aligned} f(x_1, \dots, x_k, r_{k+1}, \dots, r_n) &= r_{k+1} f(x_1, \dots, x_k, 1, r_{k+2}, \dots, r_n) \\ &+ (1-r_{k+1}) f(x_1, \dots, x_k, 0, r_{k+2}, \dots, r_n). \end{aligned}$$

If we substitute this expression above then we get

$$f(r_1, \dots, r_k, r_{k+1}, \dots, r_n) = \sum_{\mathbf{x} \in \mathbb{B}^{k+1}} \left( \prod_{i=1}^{k+1} r_i^{x_i} (1-r_i)^{(1-x_i)} \right) f(x_1, \dots, x_{k+1}, r_{k+2}, \dots, r_n).$$

and therefore the claim is also valid for  $k+1$  variables.  $\square$

Let us note that

$$\sum_{\mathbf{x} \in \mathbb{B}^n} \left( \prod_{i=1}^n r_i^{x_i} (1-r_i)^{(1-x_i)} \right) = 1,$$

for any  $\mathbf{r} \in \mathbb{U}^n$ , and that  $0 \leq \prod_{i=1}^n r_i^{x_i} (1-r_i)^{(1-x_i)} \leq 1$  for every  $\mathbf{r} \in \mathbb{U}^n$  and  $\mathbf{x} \in \mathbb{B}^n$ .

This means that  $f(\mathbf{r})$  can be seen as a convex combination of the complete set of  $n$ -binary vectors, whose weights are defined by  $\mathbf{r}$ .

**Proposition 4.5** ([54]). *Let  $f$  be a pseudo-Boolean function given by (1.1), and let  $\mathbf{r} \in \mathbb{U}^n$ . There are binary vectors  $\mathbf{x}, \mathbf{y} \in \mathbb{B}^n$  for which*

$$f(\mathbf{x}) \leq f(\mathbf{r}) \leq f(\mathbf{y}).$$

*Proof.* Existence of vectors  $\mathbf{x}$  and  $\mathbf{y}$  is an immediate consequence of Theorem 4.2.

Consider any  $\mathbf{x}^* \in \text{Argmin}_{\mathbb{B}^n}(f)$ . Then,

$$\begin{aligned} f(r_1, \dots, r_n) &= \sum_{\mathbf{x} \in \mathbb{B}^n} \left( \prod_{i=1}^n r_i^{x_i} (1-r_i)^{(1-x_i)} \right) f(x_1, \dots, x_n) \\ &\geq \sum_{\mathbf{x} \in \mathbb{B}^n} \left( \prod_{i=1}^n r_i^{x_i} (1-r_i)^{(1-x_i)} \right) f(\mathbf{x}^*) \\ &\geq f(\mathbf{x}^*). \end{aligned}$$

So,  $\mathbf{x} = \mathbf{x}^*$  holds for the claimed inequality. Obviously, any  $\mathbf{y} \in \text{Argmax}_{\mathbb{B}^n}(f)$  will satisfy the other inequality as well.  $\square$

Immediate consequences of the above proposition are the following two facts:

**Corollary 4.4.**

$$\min_{\mathbf{r} \in \mathbb{U}^n} f(\mathbf{r}) = \min_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x})$$

and

$$\max_{\mathbf{r} \in \mathbb{U}^n} f(\mathbf{r}) = \max_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x}).$$

Let us recall next from [54, 64, 212] a few properties of continuous extensions of pseudo-Boolean functions.

**Proposition 4.6.** *Let us consider an arbitrary real vector  $\mathbf{p} \in \mathbb{U}^n$ , and assume that the variables  $x_i$ ,  $i = 1, \dots, n$  are pairwise independent random variables for which  $p_i = \text{Prob}[x_i = 1] = 1 - \text{Prob}[x_i = 0]$  for  $i = 1, \dots, n$ . Then,*

$$\text{Exp}[f(\mathbf{x})] = f(\mathbf{p}).$$

*Proof.* By definition we have  $\text{Exp}[x_i] = p_i$  for  $i = 1, \dots, n$ , and  $\text{Exp}\left[\prod_{j \in S} x_j\right] = \prod_{j \in S} p_j$  for  $S \subseteq \mathbf{V}$  by the pairwise independence assumption of  $x_i$  and  $x_j$  for every  $i \neq j$ . In view of the additivity of expectation, we obtain the stated equality by using (1.5) as follows:

$$\begin{aligned} \text{Exp}[f(\mathbf{x})] &= \text{Exp}\left[\sum_{S \subseteq \mathbf{V}} c_S \prod_{j \in S} x_j\right] \\ &= \sum_{S \subseteq \mathbf{V}} c_S \prod_{j \in S} \text{Exp}[x_j] \\ &= \sum_{S \subseteq \mathbf{V}} c_S \prod_{j \in S} p_j \\ &= f(\mathbf{p}). \end{aligned}$$

□

## 4.8 Best linear Euclidean approximations

Any *linear* pseudo-Boolean function in  $n$  variables can be optimized efficiently; if the coefficient associated to a variable is positive then the variable has a persistent value 0, if the coefficient is negative then the corresponding variable has a persistent value 1, and if the coefficient is zero then the optimal value of the variable can be either 0 or 1. It is natural therefore to find a “best” linear approximation of a nonlinear pseudo-Boolean function, and use it to get a quick upper bound to the nonlinear minimization problem.

Hammer and Holzman [125] studied the  $L_2$ -approximation of pseudo-Boolean functions by linear functions, enhancing some important properties preserved in the approximation, and its close relationship with the well know power indices of Banzhaf ([149])

and Shapley ([218]) in the presence of a simple game.

In this section, the results of [125] are extended by restricting the linear Euclidean approximation to the *homogeneous* case. This leads to an explicit formula for computing the approximation directly from (1.1).

**Definition 4.4.** *Let  $\mathcal{F}$  be a subfamily of pseudo-Boolean functions, and let  $f : \mathbb{B}^n \mapsto \mathbb{R}$  be a pseudo-Boolean function. The best Euclidean (or  $L_2$ -norm) approximation of  $f$  in  $\mathcal{F}$  is the function  $g \in \mathcal{F}$  that minimizes  $\sum_{\mathbf{x} \in \mathbb{B}^n} (g(\mathbf{x}) - f(\mathbf{x}))^2$ . We write  $g = A_{\mathcal{F}}(f)$ .*

In what follows, pseudo-Boolean functions are considered to be represented by their table form, or equivalently, as vectors in  $\mathbb{R}^{2^n}$ . The vector space associated to a subfamily  $\mathcal{F}$  of pseudo-Boolean Functions is defined as

$$W_{\mathcal{F}} \stackrel{\text{def}}{=} \{(g : \mathbb{B}^n \mapsto \mathbb{R}) \mid g \in \mathcal{F}\}.$$

**Lemma 4.7.** *If  $W_{\mathcal{F}}$  is a linear subspace, then the best  $L_2$ -approximation in  $\mathcal{F}$  exists and is unique.*

*Proof.* Since  $W_{\mathcal{F}}$  is a linear subspace, then the existence and uniqueness of the best  $L_2$ -approximation in  $\mathcal{F}$  follow from the theory of orthogonal projections in Euclidean spaces.  $\square$

**Lemma 4.8.** *If  $W_{\mathcal{F}}$  is a linear subspace, then  $A_{\mathcal{F}}$  is a linear operator, i.e.*

$$A_{\mathcal{F}} \left( \sum_{i=1}^m \alpha_i f_i \right) = \sum_{i=1}^m \alpha_i A_{\mathcal{F}}(f_i), \quad (4.10)$$

for all pseudo-Boolean functions  $f_i, i = 1, \dots, m$ , and all real numbers  $\alpha_i, i = 1, \dots, m$ .

*Proof.* By Lemma 4.7,  $A$  is the orthogonal projection onto  $W_{\mathcal{F}}$ . It follows that  $A_{\mathcal{F}}$  is a linear operator.  $\square$

The family of linear pseudo-Boolean functions is denoted as

$$\mathcal{L} \stackrel{\text{def}}{=} \{(l : \mathbb{B}^n \mapsto \mathbb{R}) \mid \deg(l) \leq 1\}$$

and the family of *homogeneous* linear pseudo-Boolean functions is denoted as

$$\mathcal{H} \stackrel{\text{def}}{=} \{l \in \mathcal{L} \mid c_\emptyset = 0\}.$$

$W_{\mathcal{L}}$  and  $W_{\mathcal{H}}$  are linear subspaces in  $\mathbb{R}^{n^2}$ . Indeed, a linear combination of (homogeneous) linear functions, it is also a (homogeneous) linear function.

**Proposition 4.7** ([125]). *The best linear approximation of a monomial  $\prod_{i \in S} x_i$  is*

$$A_{\mathcal{L}} \left( \prod_{i \in S} x_i \right) \equiv -\frac{|S|-1}{2^{|S|}} + \frac{1}{2^{|S|-1}} \sum_{j \in S} x_j.$$

**Proposition 4.8** ([125]). *The best linear approximation of a pseudo-Boolean function  $f$  given as in (1.1) is*

$$\begin{aligned} A_{\mathcal{L}}(f) &\equiv -\sum_{S \subseteq \mathbf{V}} \frac{c_S (|S|-1)}{2^{|S|}} + \sum_{j \in \mathbf{V}} \left( \sum_{S \subseteq \mathbf{V}: j \in S} \frac{c_S}{2^{|S|-1}} \right) x_j \\ &\equiv -\sum_{S \subseteq \mathbf{V}} \frac{c_S (|S|-1)}{2^{|S|}} + \sum_{j \in \mathbf{V}} \Delta_j \left( \frac{1}{2}, \dots, \frac{1}{2} \right) x_j. \end{aligned}$$

*Proof.* Use Proposition 4.7, Lemma 4.8, and apply the linearity property (4.10) to expression (1.1).  $\square$

Next, we show some lemmas that later on this section will be used to define the counterparts of Propositions 4.7 and 4.8 to the homogeneous case.

Let  $S \subseteq \mathbf{V}$ . The best homogeneous linear  $l_2$ -approximation of  $\prod_{i \in S} x_i$  is determined by solving the following nonlinear optimization problem:

$$\min_{(a_1, \dots, a_n) \in \mathbb{R}^n} \left( h(a_1, \dots, a_n) = \sum_{\mathbf{x} \in \mathbb{B}^n} \left( \sum_{i=1}^n a_i x_i - \prod_{i \in S} x_i \right)^2 \right) \quad (4.11)$$

**Lemma 4.9.**  *$h$  is a strictly convex function.*

*Proof.* The first partial derivatives of  $h$  are defined as

$$\frac{\partial h(a_1, \dots, a_n)}{\partial a_j} = 2 \sum_{\mathbf{x} \in \mathbb{B}^n} x_j \left( \sum_{i=1}^n a_i x_i - \prod_{i \in S} x_i \right),$$

for  $j = 1, \dots, n$ .

The second order partial derivatives of  $h$  are defined as

$$\begin{aligned} \frac{\partial^2 h(a_1, \dots, a_n)}{\partial a_j \partial a_j} &= 2^n & \text{for } j = 1, \dots, n \text{ and} \\ \frac{\partial^2 h(a_1, \dots, a_n)}{\partial a_j \partial a_k} &= 2^{n-1} & \text{for } j, k = 1, \dots, n, j \neq k. \end{aligned}$$

To show that  $h$  is a strictly convex function, we next show that the corresponding Hessian

$$\nabla^2 h(a_1, \dots, a_n) = 2^{n-1} \begin{bmatrix} 2 & 1 & \cdots & 1 & 1 \\ 1 & 2 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 2 & 1 \\ 1 & 1 & \cdots & 1 & 2 \end{bmatrix},$$

is positive definite.

The eigenvalues of  $\nabla^2 h(a_1, \dots, a_n)$  are positive and given by

$$\begin{cases} \lambda_j = 2^{n-1}, & j = 1, \dots, n-1 \\ \lambda_n = (n+1)2^{n-1}. \end{cases} \quad (4.12)$$

This result can be found by computing the eigenvalues of the matrix of dimension  $n \times n$  with all elements equal to 1 (we call it  $\mathbf{B}$ ). Clearly, the determinant of  $\mathbf{B}$  is 0,  $n-1$  of the  $\mathbf{B}$  eigenvalues are zero and the remaining eigenvalue is  $n$  (because the sum of the eigenvalues is equal to the sum of the elements of the main diagonal). Since the eigenvalues of  $\mathbf{B} + \mathbf{I}$  can be obtained by the sum of the eigenvalues of  $\mathbf{B}$  plus 1 (i.e. the eigenvalues of  $\mathbf{I}$ ), then (4.12) follows.  $\square$

$h(a_1, \dots, a_n) \geq 0$  and Lemma 4.9 imply that the solution of (4.11) exists, is unique

and it is determined by the first order stationary conditions:

$$\frac{\partial h(a_1, \dots, a_n)}{\partial a_j} = 2 \sum_{\mathbf{x} \in \mathbb{B}^n} x_j \left( \sum_{i=1}^n a_i x_i - \prod_{i \in S} x_i \right) = 0 \quad \text{for all } j = 1, \dots, n. \quad (4.13)$$

Note that the existence and uniqueness properties of this solution, can also be derived from Lemma 4.7.

**Lemma 4.10.** *The unique solution of (4.13) is given by*

$$a_j = \begin{cases} \frac{n-|S|+2}{2^{|S|-1}(n+1)}, & j \in S \\ -\frac{|S|-1}{2^{|S|-1}(n+1)}, & j \in \mathbf{V} \setminus S. \end{cases} \quad (4.14)$$

*Proof.* Since

$$2 \sum_{\mathbf{x} \in \mathbb{B}^n} x_i x_j = \begin{cases} 2^{n-1}, & i \neq j \\ 2^n, & i = j \end{cases} \quad \text{and} \quad 2 \sum_{\mathbf{x} \in \mathbb{B}^n} x_j \prod_{i \in S} x_i = \begin{cases} 2^{n-|S|}, & j \notin S \\ 2^{n-|S|+1}, & j \in S \end{cases}$$

then (4.13) can be formulated in matrix terms as

$$2^{n-1} (\mathbf{B} + \mathbf{I}) \mathbf{a} = \frac{2^n}{2^{|S|}} (\mathbf{e} + \chi_S),$$

where  $\mathbf{B}$  is defined as before,  $\mathbf{e} = (1, \dots, 1)$  and  $\chi_S$  is the characteristic vector representing  $S$  in  $\{1, \dots, n\}$ . So,

$$\begin{aligned} (\mathbf{B} + \mathbf{I}) \mathbf{a} &= \frac{1}{2^{|S|-1}} (\mathbf{e} + \chi_S) \\ \iff \mathbf{a} &= \frac{1}{2^{|S|-1}} (\mathbf{B} + \mathbf{I})^{-1} (\mathbf{e} + \chi_S) \\ \iff \mathbf{a} &= \frac{1}{2^{|S|-1}} \left( \mathbf{I} - \frac{1}{n+1} \mathbf{B} \right) (\mathbf{e} + \chi_S) \\ \iff \mathbf{a} &= \frac{((n+1) \mathbf{I} - \mathbf{B}) (\mathbf{e} + \chi_S)}{2^{|S|-1} (n+1)} \\ \iff \mathbf{a} &= \frac{\mathbf{e} + ((n+1) \mathbf{I} - \mathbf{B}) \chi_S}{2^{|S|-1} (n+1)}. \end{aligned}$$

□

**Theorem 4.3.** *The best homogeneous linear approximation of a monomial  $\prod_{i \in S} x_i$  is*

$$A_{\mathcal{H}} \left( \prod_{i \in S} x_i \right) \equiv \frac{1}{2^{|S|-1} (n+1)} \left( (n - |S| + 2) \sum_{j \in S} x_j - (|S| - 1) \sum_{j \in \mathbf{V} \setminus S} x_j \right).$$

*Proof.* Since the unique minimizer of problem (4.11) is determined by (4.10), the result follows from Lemma 4.10.  $\square$

There is a notorious difference in the  $L_2$ -approximation of a monomial  $\prod_{j \in S} x_j$  between the general and the homogeneous cases, and it is the fact that the coefficient of a variable not involved in the monomial is always zero for the first case and is possibly nonzero, i.e.

$$\begin{cases} \frac{2}{n+1}, & S = \emptyset \\ 0, & |S| = 1 \\ \frac{1-|S|}{2^{|S|-1}(n+1)} & |S| > 1, \end{cases} \quad (4.15)$$

for the homogeneous case. Furthermore, the coefficient does not depend on  $n$  for the general case, but the coefficients (4.15) of the homogeneous case depend on the dimension of the function to be approximated.

**Theorem 4.4.** *The best homogeneous linear approximation of a pseudo-Boolean function  $f$  given as in (1.1) is*

$$\begin{aligned} A_{\mathcal{H}}(f) &\equiv \sum_{j \in \mathbf{V}} \left( \sum_{S \subseteq \mathbf{V}: j \in S} \frac{c_S (n - |S| + 2)}{2^{|S|-1} (n+1)} - \sum_{S \subseteq \mathbf{V}: j \in \mathbf{V} \setminus S} \frac{c_S (|S| - 1)}{2^{|S|-1} (n+1)} \right) x_j \\ &\equiv \sum_{j \in \mathbf{V}} \left( \sum_{S \subseteq \mathbf{V}: j \in S} \frac{c_S}{2^{|S|-1}} - \frac{1}{n+1} \sum_{S \subseteq \mathbf{V}} \frac{c_S (|S| - 1)}{2^{|S|-1}} \right) x_j \\ &\equiv \sum_{j \in \mathbf{V}} \left( \Delta_j \left( \frac{1}{2}, \dots, \frac{1}{2} \right) - \frac{1}{n+1} \sum_{S \subseteq \mathbf{V}} \frac{c_S (|S| - 1)}{2^{|S|-1}} \right) x_j. \end{aligned}$$

*Proof.* Use Theorem 4.3, Lemma 4.8, and apply the linearity property (4.10) to expression (1.1).  $\square$

A consequence of the last result is the fact that the best homogeneous linear approximation has coefficients equal to the corresponding non-homogeneous case, minus

a constant

$$\frac{1}{n+1} \sum_{S \subseteq \mathbf{V}} \frac{c_S (|S| - 1)}{2^{|S|-1}}.$$

Because of this fact, the following relationship between the two linear operators  $A_{\mathcal{H}}$  and  $A_{\mathcal{L}}$  holds for any pseudo-Boolean function  $f$ .

**Corollary 4.5.**

$$A_{\mathcal{H}}(f) \equiv (1 - A_{\mathcal{H}}(1)) \sum_{S \subseteq \mathbf{V}} \frac{c_S (|S| - 1)}{2^{|S|}} + A_{\mathcal{L}}(f).$$

The best homogeneous linear  $L_2$ -approximation is exemplified next.

**Example 4.6.** Let  $g : \mathbb{B}^5 \mapsto \mathbb{R}$  be defined as

$$g(x_1, x_2, x_3, x_4, x_5) = 8 - x_1 + 5x_2 - x_1x_5 + 4x_3x_5 - 6x_2x_4x_5 + 2x_1x_2x_3x_4.$$

This example appears in [125] and the corresponding best linear approximation is

$$A_{\mathcal{L}}(g) \equiv \frac{67}{8} - \frac{1}{8}(-10x_1 + 30x_2 + 18x_3 - 10x_4 + 0x_5).$$

Using (4.10), we get a term-by-term homogeneous linear approximation of  $g$  as follows:

1	by	$A_{\mathcal{H}}(1)$	$\equiv$	$\frac{1}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 + \frac{1}{3}x_4 + \frac{1}{3}x_5,$
$x_1$	by	$A_{\mathcal{H}}(x_1)$	$\equiv$	$x_1,$
$x_2$	by	$A_{\mathcal{H}}(x_2)$	$\equiv$	$x_2,$
$x_1x_5$	by	$A_{\mathcal{H}}(x_1x_5)$	$\equiv$	$\frac{5}{12}x_1 - \frac{1}{12}x_2 - \frac{1}{12}x_3 - \frac{1}{12}x_4 + \frac{5}{12}x_5,$
$x_3x_5$	by	$A_{\mathcal{H}}(x_3x_5)$	$\equiv$	$-\frac{1}{12}x_1 - \frac{1}{12}x_2 + \frac{5}{12}x_3 - \frac{1}{12}x_4 + \frac{5}{12}x_5,$
$x_2x_4x_5$	by	$A_{\mathcal{H}}(x_2x_4x_5)$	$\equiv$	$-\frac{1}{12}x_1 + \frac{1}{6}x_2 - \frac{1}{12}x_3 + \frac{1}{6}x_4 + \frac{1}{6}x_5$ and
$x_1x_2x_3x_4$	by	$A_{\mathcal{H}}(x_1x_2x_3x_4)$	$\equiv$	$\frac{1}{16}x_1 + \frac{1}{16}x_2 + \frac{1}{16}x_3 + \frac{1}{16}x_4 - \frac{1}{16}x_5.$

As expected, the linear part of  $g$  is its own approximation. Putting things together

and using linearity, we get the best homogeneous  $L_2$ -norm linear approximation of  $g$ :

$$\begin{aligned}
A_{\mathcal{H}}(g) &\equiv 8 \left( \frac{1}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 + \frac{1}{3}x_4 + \frac{1}{3}x_5 \right) - x_1 + 5x_2 \\
&\quad - \left( \frac{5}{12}x_1 - \frac{1}{12}x_2 - \frac{1}{12}x_3 - \frac{1}{12}x_4 + \frac{5}{12}x_5 \right) \\
&\quad + 4 \left( -\frac{1}{12}x_1 - \frac{1}{12}x_2 + \frac{5}{12}x_3 - \frac{1}{12}x_4 + \frac{5}{12}x_5 \right) \\
&\quad - 6 \left( -\frac{1}{12}x_1 + \frac{1}{6}x_2 - \frac{1}{12}x_3 + \frac{1}{6}x_4 + \frac{1}{6}x_5 \right) \\
&\quad + 2 \left( \frac{1}{16}x_1 + \frac{1}{16}x_2 + \frac{1}{16}x_3 + \frac{1}{16}x_4 - \frac{1}{16}x_5 \right) \\
&\equiv \frac{1}{24} (37x_1 + 157x_2 + 121x_3 + 37x_4 + 67x_5).
\end{aligned}$$

The following example will provide explicit formulas of approximations of polynomial expressions involving complemented variables, i.e. literals.

**Example 4.7.** Let  $P, N \subseteq \mathbf{V}$ ,  $P \cap N = \emptyset$ , and  $g : \mathbb{B}^n \mapsto \mathbb{R}$  be defined as

$$g(\mathbf{x}) = \prod_{i \in P} x_i \prod_{j \in N} \bar{x}_j. \quad (4.16)$$

The approximation coefficient  $\Gamma_1$  corresponding to a variable  $x_i$  not appearing in product (4.16) (i.e., variables  $x_i \in \mathbf{V} \setminus (P \cup N)$ ) is defined first. If  $\bar{x}_j = 1 - x_j$ ,  $j \in N$  is used to find the associated multilinear expression, then a polynomial with  $2^P$  monomials (without complemented variables) is obtained. According to the number of variables on it, the coefficient of each of the resulting monomials is either  $+1$  or  $-1$ . So,

$$\Gamma_1 = \sum_{j=0}^{|N|} \binom{|N|}{j} (-1)^{|N|+j} \bar{A}(n, |P| + |N| - j), \quad (4.17)$$

where  $\bar{A}(n, |P| + |N| - j) = \frac{-|P| - |N| + j + 1}{2^{|P| + |N| - j - 1} (n + 1)}$ . Simplifying the right hand side of (4.17),

then we get

$$\begin{aligned}
\Gamma_1 &= \sum_{j=0}^p \binom{|N|}{j} (-1)^{|N|+j} \frac{-|P| - |N| + j + 1}{2^{|P|+|N|-j-1} (n+1)} \\
&= \frac{(-1)^{|N|} \left[ (-|P| - |N| + 1) \sum_{j=0}^{|N|} \binom{|N|}{j} (-2)^j + \sum_{j=1}^{|N|} \binom{|N|}{j} (-2)^j \right]}{2^{|P|+|N|-1} (n+1)} \\
&= \frac{(-1)^{|N|} \left[ (-|P| - |N| + 1) (-1)^{|N|} + |N| \sum_{k=0}^{|N|-1} \binom{|N|-1}{k} (-2)^{k+1} \right]}{2^{|P|+|N|-1} (n+1)} \\
&= \frac{-|P| - |N| + 1 - 2|N|p (-1)^{|N|} (-1)^{|N|-1}}{2^{|P|+|N|-1} (n+1)} \\
&= \frac{|N| - |P| + 1}{2^{|P|+|N|-1} (n+1)}.
\end{aligned}$$

The approximation coefficient  $\Gamma_2$  corresponding to a non-negated variable  $x_i$  appearing in product (4.16) (i.e., variables  $x_i, i \in P$ ) is given as

$$\Gamma_2 = \sum_{j=0}^{|N|} \binom{|N|}{j} (-1)^{|N|+j} A(n, |P| + |N| - j), \quad (4.18)$$

where  $A(n, |P| + |N| - j) = \frac{n - |P| - |N| + j + 2}{2^{|P|+|N|-j-1} (n+1)}$ . Simplifying (4.18), then the following formula to get  $\Gamma_2$  is derived:

$$\begin{aligned}
\Gamma_2 &= \sum_{j=0}^{|P|+|N|} \binom{|N|}{j} (-1)^{|N|+j} \frac{n - |P| - |N| + j + 2}{2^{|P|+|N|-j-1} (n+1)} \\
&= \frac{|N| - |P| + n + 2}{2^{|P|+|N|-1} (n+1)} \\
&= \Gamma_1 + 2^{1-|P|-|N|}.
\end{aligned}$$

The approximation coefficient  $\Gamma_3$  corresponding to a negated variable appearing in product (4.16) (i.e., variables  $x_i \in N$ ) is obtained through the use of the previous approximations,  $\Gamma_1$  and  $\Gamma_2$ .  $\Gamma_1$  is used for a term with less one variable and coefficient

+1.  $\Gamma_2$  is used for a term with the same number of variables but with coefficient  $-1$ :

$$\begin{aligned}
\Gamma_3 &= \Gamma_1(|P| + |N| - 1, |N| - 1) - \Gamma_2(|N| + |N|, |N| - 1) \\
&= \frac{- (|P| + |N| - 1) + 1 + 2(|N| - 1)}{2^{(|P|+|N|-1)-1} (n+1)} - \frac{n - |P| - |N| + 2 + 2(|N| - 1)}{2^{|P|+|N|-1} (n+1)} \\
&= \frac{-2(|P| + |N|) + 4|N| - n + |P| + |N| - 2|N|}{2^{|P|+|N|-1} (n+1)} \\
&= \frac{|N| - |P| - n}{2^{|P|+|N|-1} (n+1)} \\
&= \Gamma_1 - 2^{1-|P|-|N|}.
\end{aligned}$$

**Example 4.8.** Let  $g : \mathbb{B}^5 \mapsto \mathbb{R}$  be defined as

$$g(x_1, x_2, x_3, x_4, x_5) = 8 - \bar{x}_1 + 5x_2 - x_1\bar{x}_5 + 4\bar{x}_3\bar{x}_5 - 6\bar{x}_2x_4x_5 + 2x_1\bar{x}_2x_3\bar{x}_4.$$

This example appears in [125] and the corresponding best linear approximation is

$$A_{\mathcal{L}}(g) \equiv \frac{79}{8} + \frac{1}{8}(6x_1 + 50x_2 - 14x_3 - 14x_4 - 24x_5)$$

Using the results of Example 4.7, we get a term-by-term homogeneous linear approximation of  $g$  as follows:

$$\begin{array}{llll}
\bar{x}_1 & \text{by} & A_{\mathcal{H}}(\bar{x}_1) & \equiv -\frac{2}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 + \frac{1}{3}x_4 + \frac{1}{3}x_5, \\
x_1\bar{x}_5 & \text{by} & A_{\mathcal{H}}(x_1\bar{x}_5) & \equiv \frac{7}{12}x_1 + \frac{1}{12}x_2 + \frac{1}{12}x_3 + \frac{1}{12}x_4 - \frac{5}{12}x_5, \\
\bar{x}_3\bar{x}_5 & \text{by} & A_{\mathcal{H}}(\bar{x}_3\bar{x}_5) & \equiv \frac{1}{4}x_1 + \frac{1}{4}x_2 - \frac{1}{4}x_3 + \frac{1}{4}x_4 - \frac{1}{4}x_5, \\
\bar{x}_2x_4x_5 & \text{by} & A_{\mathcal{H}}(\bar{x}_2x_4x_5) & \equiv 0x_1 - \frac{1}{4}x_2 + 0x_3 + \frac{1}{4}x_4 + \frac{1}{4}x_5 \text{ and} \\
x_1\bar{x}_2x_3\bar{x}_4 & \text{by} & A_{\mathcal{H}}(x_1\bar{x}_2x_3\bar{x}_4) & \equiv \frac{7}{48}x_1 - \frac{5}{48}x_2 + \frac{7}{48}x_3 - \frac{5}{48}x_4 + \frac{1}{48}x_5.
\end{array}$$

Putting things together, we get

$$\begin{aligned}
A_{\mathcal{H}}(g) &\equiv 8 \left( \frac{1}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 + \frac{1}{3}x_4 + \frac{1}{3}x_5 \right) \\
&\quad - \left( -\frac{2}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 + \frac{1}{3}x_4 + \frac{1}{3}x_5 \right) \\
&\quad + 5x_2 \\
&\quad - \left( \frac{7}{12}x_1 + \frac{1}{12}x_2 + \frac{1}{12}x_3 + \frac{1}{12}x_4 - \frac{5}{12}x_5 \right) \\
&\quad + 4 \left( \frac{1}{4}x_1 + \frac{1}{4}x_2 - \frac{1}{4}x_3 + \frac{1}{4}x_4 - \frac{1}{4}x_5 \right) \\
&\quad - 6 \left( 0x_1 - \frac{1}{4}x_2 + 0x_3 + \frac{1}{4}x_4 + \frac{1}{4}x_5 \right) \\
&\quad + 2 \left( \frac{7}{48}x_1 - \frac{5}{48}x_2 + \frac{7}{48}x_3 - \frac{5}{48}x_4 + \frac{1}{48}x_5 \right) \\
&= \frac{1}{24} (97x_1 + 229x_2 + 37x_3 + 37x_4 + 7x_5)
\end{aligned}$$

**Corollary 4.6.** *The best homogeneous linear approximation of a quadratic pseudo-Boolean function  $f$  given as in (1.5) is*

$$A_{\mathcal{H}}(f) \equiv \sum_{j \in \mathbf{V}} \left( \frac{2c_0}{n+1} + c_j + \frac{n \left( \sum_{r=1}^{j-1} c_{rj} + \sum_{r=j+1}^n c_{jr} \right) - \sum_{1 \leq r < s \leq n: r \neq j, s \neq j} c_{rs}}{2(n+1)} \right) x_j.$$

## Chapter 5

### Roof–Duality and New Persistency Results

Hammer, Hansen and Simeone [123] have shown how three different approaches yielded the same upper bound to the maximum of a quadratic pseudo–Boolean function  $f \in \mathcal{F}_2$ . This bound was called the *roof dual* of  $f$ . Roof–duality appeared in many other studies since [123], and its strong relation with many other basic methods was demonstrated in several publications (e.g., [12, 123, 56]), together with numerous generalizations (e.g., [49, 50, 54]) and algorithmic improvements (e.g., [51, 59]).

In this study, the roof dual results of Hammer et al. [123] are translated into the lower bound case, for which the term *floor dual* would probably be more appropriate. However, we kept the name roof dual in order to emphasize that all results are perfectly analogous for the case of upper and lower bounds.

Boros et al. [51, 59] proposed a network flow model which is able to represent a quadratic posiform. Based on this network model, Boros et al. [51, 59] proposed a maximum flow algorithm in a capacitated network with  $2n + 2$  vertices, which provides an efficient way to compute the roof dual. A special implementation of the maximum flow algorithm of [51, 59] to find the roof dual is described in this chapter. The practical computational efficiency of the implemented algorithm is then demonstrated and compared with that of other alternative methods.

The quality of the roof dual bound is highly dependent on the type of problem. For a general QUBO, the quality of this bound is not very good. However, we will see in this and in the subsequent chapters that roof dual delivers near-optimal solutions for QUBOs derived from various families of problems.

## 5.1 Minorization

**Definition 5.1.** A linear minorant (or lower plane) of a pseudo-Boolean function  $f$  in  $S$  is any linear function  $l(\mathbf{x}) = a_0 + \sum_{i \in \mathbf{V}} a_i x_i$  having real coefficients  $a_0, a_1, \dots, a_n$ , which satisfy  $l(\mathbf{x}) \leq f(\mathbf{x})$  for all  $\mathbf{x} \in S$ .

If a pseudo-Boolean function  $f$  is replaced with a linear minorant  $l$  in (1.3) then a linear relaxation

$$\min_{\mathbf{x} \in S} l(\mathbf{x}) \tag{5.1}$$

of the minimum of the function is obtained. Clearly, the optimal solution of (5.1) is a lower bound on  $\nu_S(f)$ , the optimum value of (1.3).

**Definition 5.2.** Let  $\mathcal{L}$  be a family of linear minorants of  $f$  in  $S$ . The set of linear minorants  $\mathcal{L}$  is said to be complete in  $S$  if  $f$  is the pointwise maximum of these linear functions, i.e. if

$$f(\mathbf{x}) = \max_{l \in \mathcal{L}} l(\mathbf{x}),$$

for all  $\mathbf{x} \in S$ .

For a complete family of minorants  $\mathcal{L}$  in  $S$ , the equality

$$\min_{\mathbf{x} \in S} f(\mathbf{x}) = \min_{\mathbf{x} \in S} \max_{l \in \mathcal{L}} l(\mathbf{x})$$

holds.

It is desirable to find a linear minorant in  $\mathcal{L}$  for which the optimal value of (5.1) is as close as possible to  $\nu_S(f)$ . This result can be obtained by solving the problem

$$M_S(f, \mathcal{L}) \stackrel{\text{def}}{=} \max_{l \in \mathcal{L}} \min_{\mathbf{x} \in S} l(\mathbf{x}) \leq \min_{\mathbf{x} \in S} \max_{l \in \mathcal{L}} l(\mathbf{x}) = \nu_S(f).$$

A linear minorant  $l^*$  such that  $\max_{\mathbf{x} \in S} l^*(\mathbf{x}) = M_S(f, \mathcal{L})$  is called *best linear minorant* (or *best lower plane*) in  $\mathcal{L}$ . One always has  $M_S(f, \mathcal{L}) \leq \nu_S(f)$ , and the difference  $\nu_S(f) - M_S(f, \mathcal{L})$  is called the *duality gap* with respect to  $\mathcal{L}$ . Note that the same reasoning can be used to define *linear majorants* (or *upper planes*) of  $f$  in  $S$ , i.e. a

linear function in the  $\mathbf{V}$  variables, such that  $l(\mathbf{x}) \geq f(\mathbf{x})$  for all  $\mathbf{x} \in S$ .

A term-by-term majorization procedure was shown by Hammer, Hansen and Simeone [123], which provides a well known upper bound for  $f \in \mathcal{F}_2$  in  $\mathbb{B}^n$ , called *roof dual* of  $f$ . The method of finding a “good” linear majorant (or minorant) to a pseudo-Boolean function appeared much earlier in the literature (e.g., [122, 127]).

The roof dual results of Hammer et al. [123] are translated next into the lower bound case. This method considers a complete family of linear minorants, formed by combining best  $L_1$ -norm linear minorants of the quadratic terms of the polynomial representation (1.5) of the function.

**Lemma 5.1** ([123]). *All linear minorants  $\alpha_0 + \alpha_x x + \alpha_y y$  of the product  $xy$  in  $L_1$  are of the form  $-\alpha_0 = \alpha_x = \alpha_y = \lambda$  with  $\lambda \in [0, 1]$ .*

**Lemma 5.2** ([123]). *All linear minorants  $\beta_0 + \beta_x x + \beta_y y$  of the product  $-xy$  in  $L_1$  are of the form  $\beta_0 = 0$  and  $\beta_x = 1 - \beta_y = \lambda$  with  $\lambda \in [0, 1]$ .*

At this point, it is natural to ask about the possible existence of other linear minorants found by using other norms. It turns out however that the solution sets presented in Lemmas 5.1 and 5.2 for the  $L_1$ -norm case, contain the solutions of the  $L_2$  and  $L_\infty$  cases. As a consequence the  $L_1$ -norm provides best linear minorants which are not “worse” than the ones corresponding to the  $L_2$  and  $L_\infty$  cases ([123]).

Given a quadratic pseudo-Boolean function  $f \in \mathcal{F}_2$  as in (1.1), a family  $\mathcal{R}$  of linear minorants of it is defined by taking the weighted sum of the best  $L_1$ -norm linear minorants of its terms, and using as weights the coefficients of the terms, i.e.

$$\begin{aligned}
\mathcal{R}(f) &\stackrel{\text{def}}{=} \left\{ \begin{array}{l} c_0 + \sum_{i=1}^n c_i x_i + \sum_{\substack{1 \leq i < j \leq n \\ c_{ij} < 0}} c_{ij} (\lambda_{ij} x_i + (1 - \lambda_{ij}) x_j) \\ \quad + \sum_{\substack{1 \leq i < j \leq n \\ c_{ij} > 0}} c_{ij} \lambda_{ij} (-1 + x_i + x_j) \end{array} \middle| \begin{array}{l} 0 \leq \lambda_{ij} \leq 1, \\ 1 \leq i < j \leq n \end{array} \right\} \\
&= \left\{ c_0 + \sum_{i=1}^n \left[ \begin{array}{l} - \sum_{\substack{j=i+1 \\ c_{ij} > 0}}^n c_{ij} \lambda_{ij} + \\ \left( c_i + \sum_{j=i+1}^n c_{ij} \lambda_{ij} \right. \\ \quad \left. + \sum_{\substack{j=1 \\ c_{ji} > 0}}^{i-1} c_{ji} \lambda_{ji} + \sum_{\substack{j=1 \\ c_{ji} < 0}}^{i-1} c_{ji} (1 - \lambda_{ji}) \right) x_i \end{array} \right] \middle| \begin{array}{l} 0 \leq \lambda_{ij} \leq 1, \\ 1 \leq i < j \leq n \end{array} \right\}
\end{aligned}$$

$\mathcal{R}(f)$  is a complete family, and therefore the roof dual value of  $f$  follows ([123]):

$$M_{\mathcal{R}}(f) \stackrel{\text{def}}{=} M(f, \mathcal{R}(f)) = \min_{l \in \mathcal{R}(f)} \max_{\mathbf{x} \in \mathbb{B}^n} l(\mathbf{x}).$$

Let us see that  $M_{\mathcal{R}}(f)$  can be found by solving a linear program. A linear pseudo-Boolean function of family  $\mathcal{R}(f)$  has the form  $l(\lambda) = g_0(\lambda) + \sum_{i \in \mathbf{V}} g_i(\lambda) x_i$ . For a fixed  $\lambda = \lambda^*$  the minimum value of  $l(\lambda)^*$  is simple determined by the sign of the linear coefficients  $g_i(\lambda^*)$  for all indices  $i \in \mathbf{V}$ . If  $g_i(\lambda^*) > 0$  then  $x_i^* = 0$ , if  $g_i(\lambda^*) < 0$  then  $x_i^* = 1$ , and if  $g_i(\lambda^*) = 0$  then  $x_i^*$  can be any binary value. Putting these facts together, and introducing a variable  $t_i$  for every negative linear coefficient  $g_i(\lambda)$ ,  $i \in \mathbf{V}$ , it is not difficult to show ([123]) that  $M_{\mathcal{R}}(f)$  is the optimal value of the linear program

$$\begin{aligned}
&\max && c_0 - \sum_{\substack{1 \leq i < j \leq n \\ c_{ji} > 0}} c_{ij} \lambda_{ij} + \sum_{i=1}^n t_i \\
&\text{subject to} && \\
&&& c_i + \sum_{\substack{j=1 \\ c_{ji} > 0}}^{i-1} c_{ji} \lambda_{ji} + \sum_{\substack{j=1 \\ c_{ji} < 0}}^{i-1} c_{ji} (1 - \lambda_{ji}) + \sum_{j=i+1}^n c_{ij} \lambda_{ij} \geq t_i, \quad i \in \mathbf{V} \\
&&& 0 \leq \lambda_{ij} \leq 1, \quad 1 \leq i < j \leq n \\
&&& t_i \leq 0, \quad i \in \mathbf{V},
\end{aligned} \tag{5.2}$$

which is characterized by  $n$  constraints and  $\binom{n}{2} + n$  variables.

## 5.2 Linearization

It is very common to use linear integer programming to optimize a polynomial in 0–1 variables. The first publication of this nature is perhaps due to Fortet [100], and there were many others to follow ([27, 110, 111, 128, 210, 233]).

The basic idea of this approach is to replace a term of the polynomial by a new variable, and use linear inequalities to enforce the new variable to take the value of the term for all possible values of the 0–1 variables involved in the term.

**Example 5.1.** *Let  $f$  be a pseudo-Boolean function given as a multilinear polynomial (1.1), and let us construct a linear integer program, whose optimal value is equivalent to the minimum value of  $f$ . Consider a term  $\prod_{i \in S} x_i$  of  $f$  with a nonzero coefficient  $c_S$ . The linearization procedure replaces the product  $\prod_{i \in S} x_i$  by a new variable  $z_S$ , such that  $z_S = \prod_{i \in S} x_i$  for all binary assignments of the variables in  $S$ . The inequalities*

$$\begin{aligned} z_S &\leq x_i, & i \in S \\ z_S &\geq \sum_{i \in S} x_i - |S| + 1 \\ z_S &\geq 0 \end{aligned}$$

are used to enforce this relation. It is easy to see that  $z_S$  is bounded from above by the first set of constraints, and is bounded from below by the last two constraints. One of these two options is redundant in a minimum of  $f$ . Namely, if  $c_S$  is negative (respectively positive) then the last two constraints are redundant, whereas if  $c_S$  is positive (respectively negative) then the first  $|S|$  constraints are redundant in any minimum (respectively maximum) of  $f$ .

The standard linearization procedure for computing the minimum value of a quadratic

pseudo-Boolean function given as (1.5) is

$$\begin{aligned}
& \min \left( c_0 + \sum_{i=1}^n c_i x_i + \sum_{1 \leq i < j \leq n} c_{ij} y_{ij} \right) \\
& \text{subject to} \\
& \quad y_{ij} \leq x_i, \quad 1 \leq i < j \leq n, \quad c_{ij} < 0, \\
& \quad y_{ij} \leq x_j, \quad 1 \leq i < j \leq n, \quad c_{ij} < 0, \\
& \quad y_{ij} \geq x_i + x_j - 1, \quad 1 \leq i < j \leq n, \quad c_{ij} > 0, \\
& \quad y_{ij} \geq 0, \quad 1 \leq i < j \leq n, \\
& \quad x_j \in \mathbb{B}, \quad j \in \mathbf{V},
\end{aligned} \tag{5.3}$$

whose optimal solutions  $\mathbf{x}^* \in \mathbb{B}^n$  are minimizers of  $f$ .

Replacing in the above formulation the integrality conditions on  $\mathbf{x}$ , by the conditions  $\mathbf{x} \in \mathbb{U}$ , a linear programming relaxation is obtained, whose optimum value  $L_{\mathcal{R}}(f)$  is a lower bound on the minimum of  $f$ .

**Proposition 5.1** ([123]).

$$L_{\mathcal{R}}(f) = M_{\mathcal{R}}(f).$$

*Proof.* The somewhat technical proof can be found in Hammer et al. [123], whose basic step is to show that the linear program (5.3) is the dual of problem (5.2).  $\square$

### 5.2.1 Persistency of linearizations

**Proposition 5.2** ([48]). *Let  $U_k, Z_k$  ( $k = 1, \dots, m$ ) be subsets of the set of variables  $\mathbf{V}$ , such that  $U_k \cap Z_k = \emptyset$  ( $k = 1, \dots, m$ ). If the polyhedron  $\mathbf{P}$  defined by the set of inequalities*

$$\begin{aligned}
& \sum_{i \in U_k} x_i + \sum_{i \in Z_k} (1 - x_i) \leq 1 \quad (k = 1, \dots, m), \\
& 0 \leq x_i \leq 1 \quad i \in \mathbf{V}.
\end{aligned} \tag{5.4}$$

*is not empty, then it has an element with values  $x_i \in \{0, \frac{1}{2}, 1\}$  for all  $i \in \mathbf{V}$ .*

*Proof.* To a real vector  $\alpha \in \mathbb{R}^n$  we shall associate another vector  $\hat{\alpha}$  defined by

$$\hat{\alpha}_i = \begin{cases} 0 & \text{if } \alpha_i < \frac{1}{2}, \\ \frac{1}{2} & \text{if } \alpha_i = \frac{1}{2}, \\ 1 & \text{if } \alpha_i > \frac{1}{2}, \end{cases} \quad (i = 1, \dots, n).$$

Let us prove the claim by showing that if  $\alpha$  is an element of the polyhedron  $\mathbf{P}$ , then so is  $\hat{\alpha}$ . Clearly, there is at most an index for which either  $\alpha_i < \frac{1}{2}$  and  $i \in Z_k$  or  $\alpha_i > \frac{1}{2}$  and  $i \in U_k$  for a given  $k = 1, \dots, m$ . If such index  $i$  exists, then for all the remaining indices  $j \neq i$  one must have  $\alpha_j < \frac{1}{2}$  for  $j \in U_k \setminus \{i\}$  and  $\alpha_j > \frac{1}{2}$  for  $j \in Z_k \setminus \{i\}$ , implying thus  $\hat{\alpha}_j = 0$ , and hence  $\sum_{j \in U_k} \hat{\alpha}_j + \sum_{j \in Z_k} (1 - \hat{\alpha}_j) \leq 1$ . If there is no such index  $i$ , then  $\hat{\alpha}_j \leq \alpha_j$  for  $j \in U_k$  and  $\hat{\alpha}_j \geq \alpha_j$  for  $j \in Z_k$ , implying that  $\sum_{j \in U_k} \hat{\alpha}_j + \sum_{j \in Z_k} (1 - \hat{\alpha}_j) \leq \sum_{j \in U_k} \alpha_j + \sum_{j \in Z_k} (1 - \alpha_j) \leq 1$ .  $\square$

**Definition 5.3** ([60]). *Given a vector  $\alpha \in \mathbf{P}$ , the set of indices  $i = 1, \dots, n$  having  $\alpha_i = \frac{1}{2}$  is called the curse of  $\alpha$ , and is denoted by  $\mathcal{C}(\alpha)$ .*

**Proposition 5.3** ([60]). *If  $\alpha \in \mathbf{P}$  and  $\beta \in \mathbf{P}$ , then there is a  $\gamma \in \mathbf{P}$  such that  $\mathcal{C}(\gamma) = \mathcal{C}(\alpha) \cap \mathcal{C}(\beta)$ .*

*Proof.* It is easy to check that  $\gamma = \frac{1}{4}\hat{\alpha} + \frac{3}{4}\hat{\beta}$  satisfies the above conditions.  $\square$

Since there is only a finite number of possible curses, it follows from Proposition 5.3 that

**Corollary 5.1.** *There is a unique maximal subset  $\mathcal{C}_{\mathbf{P}} \subseteq \{1, \dots, n\}$  such that  $\mathcal{C}_{\mathbf{P}} \subseteq \mathcal{C}(\alpha)$  for every  $\alpha \in \mathbf{P}$ .*

The next question that we address is how to find out an element of  $\mathbf{P}$  that has a maximal set of integral values.

**Lemma 5.3.** *Given a polyhedron  $\mathbf{P}$  defined as (5.4) and an index  $i \in \{1, \dots, n\}$ , then the lower dimension polyhedron  $\mathbf{P}' \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbf{P} \mid x_i = b\}$  ( $b = 0, 1$ ), when it is non-empty, has a set of inequalities representation equivalent to the structure of those “packing” inequalities of (5.4).*

*Proof.* Let us assume that there is a point  $\alpha \in \mathbf{P}$  having  $\alpha_i = b$ . Then this assignment in (5.4) results in packing inequalities having the form  $\sum_{j \in U} x_j + \sum_{j \in Z} (1 - x_j) \leq 1 - b$ . There are two cases to consider:

- (i) If the right hand side  $1 - b$  is 1 then the resulting inequality has the required form;
- (ii) If the right hand side  $1 - b$  is 0 then all the points must have value 0 for all indices in  $U$  and must have value 1 for all indices in  $Z$ .

All the transitive assignments (ii) can be re-applied to the remaining packing inequalities, either resulting again in either case (i) or (ii). Thus, in the end of the this recursive procedure either we get an assignment of values to some indices or we get inequalities of the type  $\sum_{j \in U'} x_j + \sum_{j \in Z'} (1 - x_j) \leq 1$ .  $\square$

**Theorem 5.1.** *If  $\mathbf{P}$  is non-empty, then there is a vector  $\gamma \in \mathbf{P}$  that has integral values for all indices except for indices belonging to  $C_{\mathbf{P}}$ .*

*Proof.* Since  $C_{\mathbf{P}}$  is the intersection of all courses of points in  $\mathbf{P}$ , then for the remaining indices  $j \in \{1, \dots, n\} \setminus C_{\mathbf{P}}$  there is a point  $\alpha$  with integral value for  $\alpha_j$ . Using this property and the result of Corollary 5.1 we shall construct a point  $\gamma$  that has a maximal set of integral values. Given an index  $j \in \{1, \dots, n\} \setminus C_{\mathbf{P}}$  find a point  $\alpha$  with integral value for  $\alpha_j$ . Fix  $\gamma_j = \alpha_j$  and define a new polyhedron  $\mathbf{P}' = \mathbf{P} \cap \{\mathbf{x} \in \mathbf{P} \mid x_j = \alpha_j\}$ . From Lemma 5.3,  $\mathbf{P}'$  also has half integral points, and Corollary 5.1 also applies to it. Also, note that  $C_{\mathbf{P}'} = C_{\mathbf{P}}$ . Thus, if there are indices not in  $C_{\mathbf{P}}$  not yet fixed in  $\gamma$  we can apply the above procedure to fix those indices to an integral value, thus proving the claim.  $\square$

**Lemma 5.4.** *Let  $f$  be a pseudo-Boolean function  $f$  and let  $\phi_f = \sum_{T \subseteq \mathbf{L}} a_T \prod_{u \in T} u$  be a posiform representation of it. Then, the maximum of  $f$  is the optimum value of the*

0–1 linear integer program

$$\begin{aligned}
& \max && a_\emptyset + \sum_{T \subseteq \mathbf{L} | a_T > 0} a_T y_T \\
& \text{subject to} && \\
& && y_T + (1 - u) \leq 1, \quad T \subseteq \mathbf{L}, a_T > 0, u \in T, \\
& && \bar{u} = 1 - u, \quad u \in \mathbf{L}, \\
& && \mathbf{u} \in \mathbb{B}^{\mathbf{L}}, \\
& && \mathbf{y} \in \mathbb{B}^{|\{T \subseteq \mathbf{L} | a_T > 0\}|}.
\end{aligned} \tag{5.5}$$

*Proof.* The result follows directly after applying the linearization procedure to each non-trivial term of the posiform.  $\square$

If the integrality constraint of problem (5.5) is relaxed and if all the complemented literals  $\bar{u}$  are replaced by  $1 - u$ , then the corresponding set of feasible solutions is defined by packing inequalities. Let us call this polyhedron as  $\mathbf{P}_S$ . It is trivial to see that  $\mathbf{P}_S$  is non-empty, since point  $(\frac{1}{2}, \dots, \frac{1}{2})$  belongs to it. Corollary 5.1 implies that there is a unique maximal subset  $\mathcal{C}_{\mathbf{P}_S}$  of indices of points of  $\mathbf{P}_S$  having value  $\frac{1}{2}$ .

The question addressed next is to show that the integral values of an optimal solution of the relaxation are persistent in (5.5). This result allows us to simplify the problem at hand by fixing the corresponding variables with those values.

**Theorem 5.2.** *Let  $\alpha^* \in \mathbf{P}_S$  be an optimal solution to the relaxation of (5.5). Then, if  $\alpha_i^* = 1$  for  $i \in U$ , and  $\alpha_i^* = 0$  for  $i \in Z$ , then there is an integral optimal solution  $\beta^*$  of (5.5) having  $\beta_i^* = \alpha_i^*$  for  $i \in U \cup Z$ .*

*Proof.* To prove this result we first reduce problem (5.5) to a vertex packing problem on a weighted graph  $G = (V, E)$ . The vertex set  $V$  is defined by the set of literals plus the set of variables used to linearize high degree terms, i.e.  $V = V_1 \cup V_{\geq 2}$  where

$$\begin{aligned}
V_1 &= \{\{u\} | u \in \mathbf{L}\}, \text{ and} \\
V_{\geq 2} &= \{T | T \subseteq \mathbf{L}, a_T > 0, |T| > 1\}.
\end{aligned}$$

The edge set  $E$  is defined as follows:

- For every non-complemented vertex  $\{u\} \in V_1$  we define an edge  $(\{u\}, \{\bar{u}\})$ ;
- For every vertex  $T \in V_{\geq 2}$  we define an edge  $(T, \{\bar{u}\})$  for every  $u \in T$ .

The weight of a vertex  $\{u\} \in V_1$  is  $a_{\{u\}} + M$ , where  $M$  is a sufficient large number (e.g.  $M = 1 + 2 \sum_{T \subseteq \mathbf{L} | T \neq \emptyset} a_T$ ). The weight of a vertex  $T \in V_{\geq 2}$  is simply  $a_T$ . Next, we present the standard 0-1 linear program to find a maximum vertex packing (or similarly, stable set, independent set) to graph  $G$  (see also Section 10.1):

$$\begin{aligned}
\max \quad & a_\emptyset + \sum_{i=1}^n (M + a_{\{x_i\}}) x_i + \sum_{i=1}^n (M + a_{\{\bar{x}_i\}}) x_i^c + \sum_{T \subseteq \mathbf{L} | a_T > 0, |T| \geq 2} a_T y_T \\
\text{subject to} \quad & \\
& y_T + x_i^c \leq 1, \quad T \subseteq \mathbf{L}, a_T > 0, |T| \geq 2, x_i \in T, \\
& y_T + x_i \leq 1, \quad T \subseteq \mathbf{L}, a_T > 0, |T| \geq 2, \bar{x}_i \in T, \\
& x_i + x_i^c \leq 1, \quad i = 1, \dots, n, \\
& \mathbf{x}, \mathbf{x}^c \in \mathbb{B}^n, \\
& \mathbf{y} \in \mathbb{B}^{| \{T \subseteq \mathbf{L} | a_T > 0, |T| \geq 2 \} |}.
\end{aligned} \tag{5.6}$$

Let us call to the polyhedron defined by the continuous relaxation of problem (5.6) as  $\mathbf{P}'_S$ . It is well known that every extreme point of  $\mathbf{P}'_S$  is half-integral ([30]), and that if an optimal solution of the relaxation of problem (5.6) has integral values, then there is a persistency for each one of those indices with discrete values in a optimum of problem (5.6) ([182]; see Section 10.1). Every optimal solution  $(\mathbf{x}^*, \mathbf{x}^{c*}, \mathbf{y}^*)$  of problem (5.6) must have  $\mathbf{x}^* = (1, \dots, 1) - \mathbf{x}^{c*}$ . Also due to our choice of value  $M$ , any such optimal solution originates an optimal solution  $(\mathbf{x}^*, \mathbf{y}^*)$  of problem (5.5) with the same value shifted below by  $M$ .  $\square$

We summarize next the main results presented so far in this section. First, we have seen that posiform maximization (5.5) is equivalent to a weighted graph stability problem (5.6). From graph stability theory, we have seen that the relaxed linearized version of the problem has half-integral values ([30]), and that the variables of the relaxation with value 1 belong to an optimal weighted stable set ([182]). Second, Corollary 5.1 shows that the relaxation of the linearization problem associated to the maximization

of a posiform has a maximal set of variables with value  $\frac{1}{2}$  in all feasible solutions. This last result gives a limit on the maximum number of variables that can be simplified by their persistent values in the relaxed version of the linearization models.

The important question to address next is therefore how to find efficiently this maximal set of persistencies. A possible procedure is to use linear programming to solve the relaxation of the stability problem (5.6) of a graph  $G$ . If there are any variables with integral values, then add all associated vertices with value 1 to a set  $S$  and remove from  $G$  all vertices in  $S$  and all their neighbors. After this stage then a probing procedure can be applied. For each vertex  $i$  left in  $G$ , we would force vertex  $i$  to belong to a maximum stable set, and apply any persistent conclusions on the reduced graph to infer if this vertex is on a maximum stable set of the original graph. For instance, if the optimum of problem (5.6) for the reduced graph is equivalent to the optimum of that problem associated to the original graph, then vertex  $i$  must belong to a maximum stable set of the original graph. This procedure has been initially proposed by Nemhauser and Trotter [182].

In Section 10.1 we propose an alternative algorithm that is based on the network flow model presented in the following section. A maximal set of persistencies for graph stability (and consequently for posiform maximization as well) is derived through the application of a strong components algorithm to the resulting residual network (see Chapter 7 and Section 10.1).

It should be remarked that it is possible to translate the previous results to posiform *minimization*, since given any posiform in  $n$  variables it is possible to reproduce a negaform of it without exponentially increasing the size of the original posiform. For instance, the term  $u\bar{v}\bar{w}z$  could be replaced by  $u\bar{v}\bar{w}z = 1 - u\bar{v}\bar{w}\bar{z} - u\bar{v}w - uv - \bar{u}$ .

Clearly, a QUBO problem can be represented in either forms, and from its multilinear form (1.5), a quadratic pseudo-Boolean function can be brought into any quadratic posiform or negaform. Thus, all the results described in this section apply to the QUBO linearization problem (5.3).

### 5.3 Implication networks

In this section, we introduce the network flow model for a given *quadratic* posiform  $\phi$  in *standard form* (see Section 4.6.1). It will be seen that this network model can be used to decompose the original problem in “simpler” subproblems, and to derive lower bounds, (weak and strong) persistencies and logical relations of the minimum of the associated posiform.

Let us associate to a standard quadratic posiform  $\phi$ , a capacitated directed network  $G_\phi = (N, A)$ , where the node set is defined as  $N = \mathbf{L} \cup \{x_0, \bar{x}_0\}$ , with  $x_0$  being an additional symbol representing the constant  $x_0 = 1$ . To every quadratic term  $a_{uv}uv$  of  $\phi$  we associate two arcs  $(u, \bar{v})$  and  $(v, \bar{u})$ , and let the capacity of both arcs be  $\frac{1}{2}a_{uv}$ . By writing the linear terms as  $a_u u = a_u u x_0$ , similarly we associate two arcs  $(u, \bar{x}_0)$  and  $(x_0, \bar{u})$ , and let the capacity of both arcs be  $\frac{1}{2}a_u$ . Let us note that the constant term  $C(\phi)$  was disregarded from this construction.

Conversely, given a directed network  $G_\phi = (N, A)$  with  $N = \mathbf{L} \cup \{x_0, \bar{x}_0\}$ , and with nonnegative capacities  $c_{uv}$  assigned to the arcs  $(u, v)$ , we can associate to it a quadratic posiform:

$$\phi_G = \sum_{(u,v) \in A} c_{uv} u \bar{v}.$$

It is easy to see that the above definitions imply that

**Proposition 5.4** ([54]). *There is a one-to-one correspondence between quadratic posiforms for which  $C(\phi) = 0$ , and capacitated directed networks  $G = (N, A)$  with node set  $N = \mathbf{L} \cup \{x_0, \bar{x}_0\}$ . Furthermore, the involution  $\phi_{G_\phi} = \phi$  holds for such corresponding pairs.*

**Example 5.2.** *Consider the quadratic pseudo-Boolean function  $g = f_6(x_1, x_2, x_3, x_4, x_5, 0)$ .*

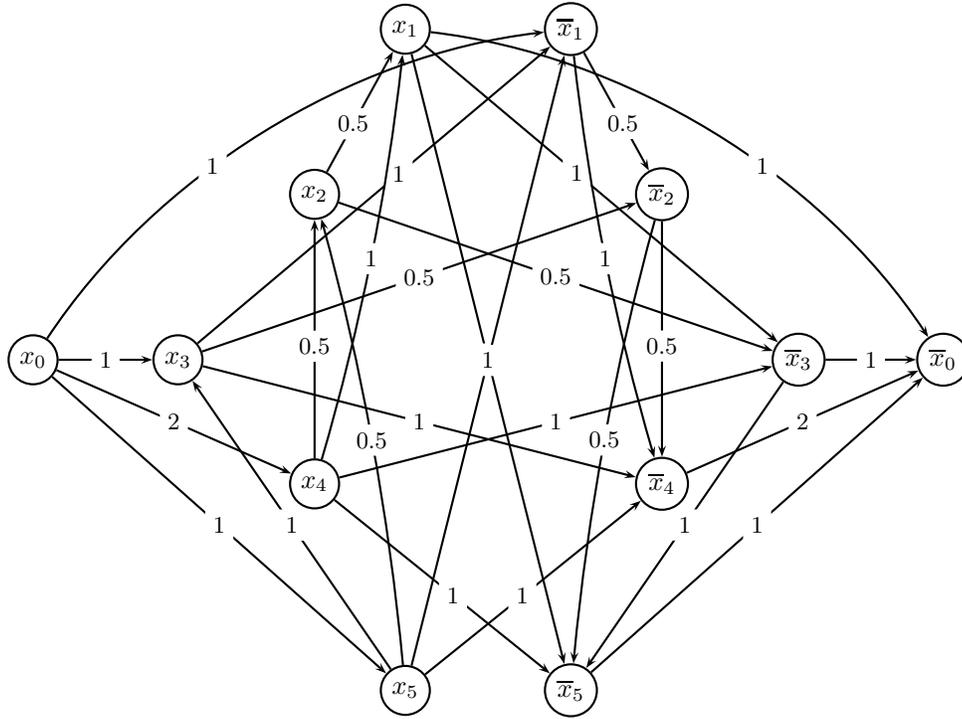


Figure 5.1: The network  $G_{\phi_g}$  corresponding to the posiform  $\phi_g$  of Example 5.2.

A quadratic posiform that represents  $g$  is

$$\begin{aligned}
 \phi_g &= -8 + 2x_1 + 2\bar{x}_3 + 4\bar{x}_4 + 2\bar{x}_5 \\
 &\quad + 2x_1x_3 + 2x_1x_5 + \bar{x}_1x_2 + 2\bar{x}_1x_4 \\
 &\quad + x_2x_3 + \bar{x}_2x_4 + \bar{x}_2x_5 \\
 &\quad + 2x_3x_4 + 2\bar{x}_3x_5 \\
 &\quad + 2x_4x_5.
 \end{aligned}$$

The corresponding capacitated network  $G_{\phi_g}$  can be seen in Figure 5.1.

**Definition 5.4.** A posiform  $\phi \in \mathcal{P}_2(f)$  given as expression (1.6) is purely quadratic if it does not contain linear terms, i.e.  $a_u = 0$  for all  $u \in \mathbf{L}$ .

Purely quadratic posiforms can be used to detect components of the original network that can be optimized separately, thus providing a convenient way to find the optimum of the original problem by solving several simpler sub-problems.

**Proposition 5.5** ([42]). *Let  $\phi \in \mathcal{P}_2(f)$  be a purely quadratic posiform, and let  $G_\phi = (N, A)$  be the corresponding capacitated network. Let  $C_1, C_2, \dots, C_k \subseteq N$  be the strongly connected components of  $G_\phi$ , which contains both a variable and its complement. Let  $G_i = (C_i, A_i), i = 1, \dots, k$ , be the subnetworks of  $G_\phi$  induced by  $C_i, i = 1, \dots, k$ . Then,*

$$\nu(f) = C(\phi) + \sum_{i=1}^k \nu(f_{\phi_{G_i}}).$$

In Proposition 5.5, the capacities of the network are not involved. In fact, if one disregards the capacities and the linear terms, the network is exactly the implication graph of Aspvall et al. [22] (see Section 4.5.) The strong components of the network  $G_\phi$  can be found in linear time by depth first search ([227]), or by a specialized version that uses the symmetry property of the network construction (see e.g. [134]).

Proposition 5.5 is a consequence of the symmetry of the network construction. This property implies that whenever two literals  $u$  and  $v$  belong to the same strong component  $C_i$ , then the complements  $\bar{u}$  and  $\bar{v}$  also belong to the same strong component  $C_j$ . Furthermore, if a literal  $u$  appears in a component  $C_i$ , and if its complement  $\bar{u}$  appears in a dual component  $C_j$ , then every other literal  $v \in C_i$  has its complement also in  $C_j$ . Let us note that  $C_i$  and  $C_j$  could or could not refer to the same component. If the components are distinct, then they are called *dual* components of each other. In this case, there is no cycle between any literal  $v \in C_i$  and its complement  $\bar{v}$  in  $G_\phi$ .

Let  $\psi$  be the purely quadratic posiform containing terms of  $\phi$ , which involve variables appearing in dual components of  $G_\phi$ . By Proposition 5.5, terms having variables in dual components can be disregarded from the optimization process. The reason for this being possible is the fact that there is a solution to the expression  $\psi = 0$ , for any given minimizer of  $f_{\phi-\psi}$ , i.e. for any minimizer of the posiform containing all terms of  $\phi$  with literals belonging to non-dual components.

**Example 5.3.** *Consider the quadratic pseudo-Boolean function  $g = f_6(x_1, x_2, x_3, x_4, x_5, 0)$*

of Example 5.2. A purely quadratic posiform that represents  $g$  is

$$\begin{aligned}\psi_g &= -3 + x_1\bar{x}_2 + 2x_1x_3 + 2x_1\bar{x}_4 + x_1x_5 + \bar{x}_1\bar{x}_5 \\ &\quad + x_2x_3 + x_2\bar{x}_5 + \bar{x}_2x_4 \\ &\quad + 2\bar{x}_3\bar{x}_4 + 2\bar{x}_3x_5 \\ &\quad + 2x_4x_5.\end{aligned}$$

The capacitated network  $G_{\psi_g}$  has two strong components:  $C_1 = \{\bar{x}_1, \bar{x}_2, x_3, \bar{x}_4, x_5\}$  and  $C_2 = \{x_1, x_2, \bar{x}_3, x_4, \bar{x}_5\}$  (see Figure 5.2).  $C_1$  and  $C_2$  are dual components. Then, by Proposition 5.5 the minimum value of  $g$  in  $\mathbb{B}^5$  is  $C(\psi_g)$  ( $= -3$ ), and all optimal solutions must satisfy the quadratic Boolean equation

$$x_1\bar{x}_2 \vee x_1x_3 \vee x_1\bar{x}_4 \vee x_1x_5 \vee \bar{x}_1\bar{x}_5 \vee x_2x_3 \vee x_2\bar{x}_5 \vee \bar{x}_2x_4 \vee \bar{x}_3\bar{x}_4 \vee \bar{x}_3x_5 \vee x_4x_5 = 0.$$

If the capacities of  $G_{\psi_g}$  are disregarded, then the network given in Figure 5.2 also represents the implication graph. From this implication graph, it is simple to verify that every literal  $u \in C_1$  must have an optimal value not smaller than the value of any literal in  $v \in C_2$ , i.e.  $u \geq v$ . Thus, the solution of the previous quadratic Boolean equation is unique, and equal to  $(0, 0, 1, 0, 1)$ .

Example 5.3 suggests the following result.

**Proposition 5.6** ([22]). *Let  $\phi \in \mathcal{P}_2(f)$  be a purely quadratic posiform, for a quadratic pseudo-Boolean function  $f \in \mathcal{F}_2$ . The minimum of a  $f$  coincides with the constant part of the posiform  $\phi$  (i.e.  $\nu(f) = C(\phi)$ ) if and only if all the strong components in  $G_\phi$  have dual components.*

*Proof.* Sufficiency is an immediate result of Proposition 5.5. To prove necessity, we assume that there is a strong component  $C$  in  $G_\phi$ , without a dual component. By the symmetry property of  $G_\phi$  and the non-duality property of  $C$ , every literal  $u$  in the component  $C$  has its complement  $\bar{u}$  also in  $C$ .

Consider now the implication graph associated to the literals in  $C$ . From the theory of quadratic Boolean equations, since the implication graph is strongly connected (i.e.

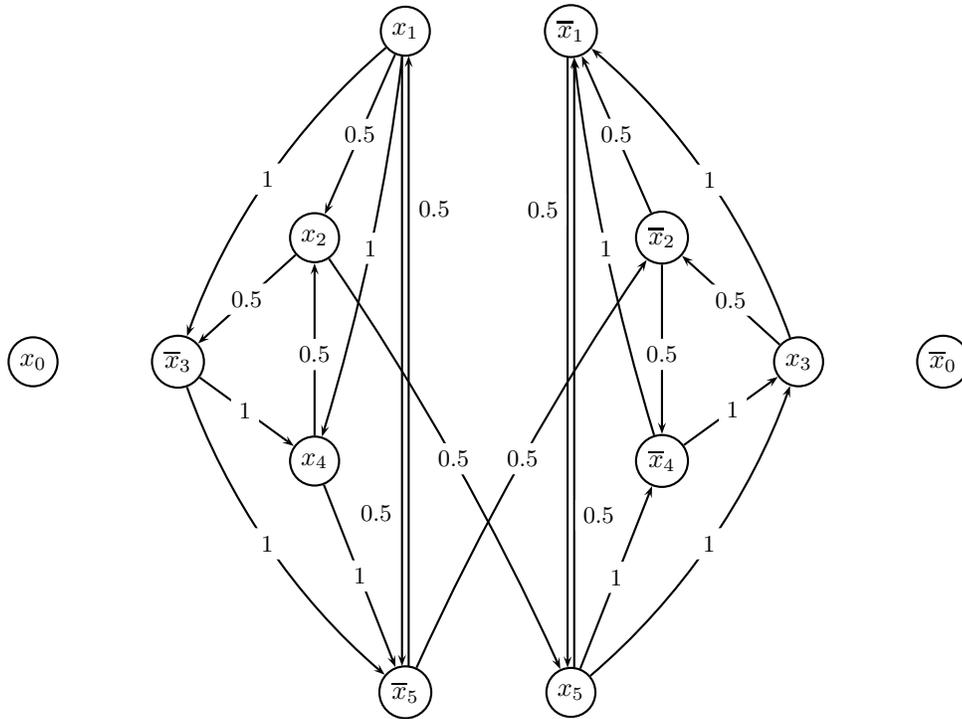


Figure 5.2: The network  $G_{\psi_g}$  corresponding to the posiform  $\psi_g$  of Example 5.3.

there is a path from any literal  $u \in C$  to its complement), then the associated quadratic Boolean equation is not consistent. This implies that the posiform  $\phi_C$  associated to the subnetwork of  $G_\phi$  induced by the nodes in  $C$  is positive for any assignment of the variables in  $C$ . Consequently, we get a contradiction  $\nu(f) = C(\phi) + \nu(f_{\phi_C}) > C(\phi)$  of our assumption.  $\square$

For any given quadratic pseudo-Boolean function  $f$ , an efficient way to obtain a purely quadratic posiform  $\psi$  is given next. This method uses flow techniques in a capacitated network  $G_\phi$ , where  $\phi$  is any posiform representing  $f$ .

Let  $G = (\mathbf{L} \cup \{x_0, \bar{x}_0\}, A)$  be a capacitated network with *source*  $x_0$  and *sink*  $\bar{x}_0$ , and having positive capacities  $c_{uv}$  for all arcs  $(u, v) \in A$ . A *feasible flow* in  $G = (N, A)$  is a mapping  $\varphi : A \leftarrow \mathbb{R}_+$  satisfying the constraints

$$0 \leq \varphi(u, v) \leq c_{uv} \text{ for all arcs } (u, v) \in A,$$

$$\sum_{(x_0, v) \in A} \varphi(x_0, v) + \sum_{(v, \bar{x}_0) \in A} \varphi(v, \bar{x}_0) = 0,$$

and

$$\sum_{(u, v) \in A} \varphi(u, v) = \sum_{(v, w) \in A} \varphi(v, w) \text{ for all nodes } v \in \mathbf{L}.$$

Given a capacitated network  $G = (N, A)$  with positive capacities  $c_{uv}$  for all arcs  $(u, v) \in A$ , and a feasible flow  $\varphi$  in it, the *residual network*  $G[\varphi] = (N, A^\varphi)$  is a capacitated network with the same node set  $N$ , arcs set given by

$$A^\varphi = \{(u, v) \in A \mid c_{uv} > \varphi(u, v)\} \cup \{(v, u) \mid (u, v) \in A, \varphi(u, v) > 0\},$$

and *residual* capacities

$$c_{uv}^\varphi = \begin{cases} c_{uv} - \varphi(u, v), & (u, v) \in A, \\ \varphi(v, u), & (v, u) \in A, \end{cases}$$

for all arcs  $(u, v) \in A^\varphi$ .

An *augmenting path* of capacity  $w$  in the residual network  $G[\varphi]$  is a directed path (i.e., a sequence of nodes:  $x_0 \equiv u_0, u_1, \dots, u_k, u_{k+1} \equiv \bar{x}_0$ ) from the source node  $x_0$  to the sink  $\bar{x}_0$ , where  $w$  is the minimum *residual capacity* of any arc in the path, i.e.  $w = \min_{j=0, \dots, k+1} (c_{u_j u_{j+1}}^\varphi)$ .

Due to the special structure of the network  $G_\phi$ , a feasible flow can always be assumed to be symmetric, i.e. such that  $\varphi(u, v) = \varphi(\bar{v}, \bar{u})$  holds for every arc  $(u, v)$  in  $G_\phi$ . Thus, for symmetric feasible flows  $\varphi$ , if  $x_0, u_1, \dots, u_k, \bar{x}_0$  represents an augmenting path with capacity  $w$  in  $G_\phi[\varphi]$ , then the path  $\bar{x}_0, \bar{u}_k, \dots, \bar{u}_1, x_0$  is also augmenting and has the same capacity  $w$  in  $G_\phi[\varphi]$ . This path is called the *twin* path. It should be remarked that twin paths can actually share arcs.

An *alternating sum* is an expression of the form

$$u_1 + \bar{u}_1 u_2 + \bar{u}_2 u_3 + \dots + \bar{u}_{k-1} u_k + \bar{u}_k \tag{5.7}$$

involving the literals  $u_1, \dots, u_n \in \mathbf{L}$ . A quadratic posiform  $\phi$  contains the alternating

sum (5.7) with maximum weight  $w$ , if for the corresponding coefficients of  $\phi$  in (1.6) we have  $w = \min(a_{u_1}, a_{\bar{u}_1 u_2}, a_{\bar{u}_2 u_3}, \dots, a_{\bar{u}_{k-1} u_k}, a_{\bar{u}_k})$ . It is well known that the following identity holds for alternating sums:

$$u_1 + \bar{u}_1 u_2 + \bar{u}_2 u_3 + \dots + \bar{u}_{k-1} u_k + \bar{u}_k = 1 + u_1 \bar{u}_2 + u_2 \bar{u}_3 + \dots + u_{k-1} \bar{u}_k.$$

If a quadratic posiform  $\phi$  contains an alternating sum (5.7) with maximum weight  $w$ , then  $\phi$  can be transformed into a posiform  $\phi'$  representing the same function, but having a larger constant term  $C(\phi')$  ( $= C(\phi) + w$ ), as follows:

$$\begin{aligned} \phi' &= \phi \\ &= C(\phi) + a_{u_1} u_1 + \sum_{i=1}^{k-1} a_{\bar{u}_i u_{i+1}} \bar{u}_i u_{i+1} + a_{\bar{u}_k} \bar{u}_k + \psi \\ &= C(\phi) + w + w \sum_{i=1}^k u_i \bar{u}_{i+1} + (a_{u_1} - w) u_1 \\ &\quad + \sum_{i=1}^{k-1} (a_{\bar{u}_i u_{i+1}} - w) \bar{u}_i u_{i+1} + (a_{\bar{u}_k} - w) \bar{u}_k + \psi, \end{aligned}$$

where  $\psi$  is the remaining part of  $\phi$ , not containing the alternating sum and the constant part  $a_0$ .

From the previous observations, it is clear that there is a one-to-one correspondence between alternating sums contained in a posiform  $\phi$ , and augmenting paths in the corresponding network  $G_\phi$ . Thus, we have:

**Proposition 5.7** ([54, 59]). *Let  $\phi$  be a quadratic posiform, and let  $\varphi$  be a feasible flow in the corresponding capacitated network  $G_\phi$ . Then,  $x_0, u_1, \dots, u_k, \bar{x}_0$  is an augmenting path with capacity  $w$  in the residual network  $G_\phi[\varphi]$  if and only if  $u_1 + \bar{u}_1 u_2 + \dots + \bar{u}_{k-1} u_k + \bar{u}_k$  is an alternating sum of maximum weight  $w$  in the corresponding posiform  $\phi_{G_\phi[\varphi]}$ .*

Also a consequence of the above is the following result.

**Proposition 5.8** ([54, 59]). *Let  $\phi \in \mathcal{P}_2(f)$  for a quadratic pseudo-Boolean function  $f \in \mathcal{F}_2$ , and let  $\varphi$  be a feasible flow in the corresponding network  $G_\phi$ . Let us denote by  $v(\varphi)$  the value of the flow (i.e. the total flow leaving the source, or the total flow arriving to the sink), and let  $\psi = \phi_{G_\phi[\varphi]}$  denote the posiform corresponding to the*

residual network. Then,

$$C(\phi) + v(\varphi) + \psi \in \mathcal{P}_2(f).$$

All feasible flows can be obtained from the constant zero flow by iteratively increasing the flow along augmenting paths. A flow  $\varphi$  is a *maximum flow* if and only if the residual network  $G_\phi[\varphi]$  contains no augmenting path.

**Example 5.4.** Consider the quadratic pseudo-Boolean function  $h = f_6(x_1, x_2, x_3, x_4, x_5, 1)$ .

A quadratic posiform that represents  $h$  is

$$\begin{aligned} \phi_h &= -7 + x_1 + x_2 + \bar{x}_3 + 5\bar{x}_4 \\ &\quad + 2x_1x_3 + 2x_1x_5 + \bar{x}_1x_2 + 2\bar{x}_1x_4 \\ &\quad + x_2x_3 + \bar{x}_2x_4 + \bar{x}_2x_5 \\ &\quad + 2x_3x_4 + 2\bar{x}_3x_5 \\ &\quad + 2x_4x_5. \end{aligned} \tag{5.8}$$

The corresponding network  $G_{\phi_h}$  can be seen in Figure 5.3.a). Checking in Figure 5.3, we can see that 0.5 units of flow can be pushed sequentially through each of the following augmenting paths:

$$\begin{aligned} x_0 \rightarrow \bar{x}_2 \rightarrow \bar{x}_4 \rightarrow \bar{x}_0 \quad \text{and its twin} \quad x_0 \rightarrow x_4 \rightarrow x_2 \rightarrow \bar{x}_0, \\ x_0 \rightarrow x_4 \rightarrow \bar{x}_3 \rightarrow \bar{x}_0 \quad \text{and its twin} \quad x_0 \rightarrow x_3 \rightarrow \bar{x}_4 \rightarrow \bar{x}_0, \\ x_0 \rightarrow \bar{x}_1 \rightarrow \bar{x}_4 \rightarrow \bar{x}_0 \quad \text{and its twin} \quad x_0 \rightarrow x_4 \rightarrow x_1 \rightarrow \bar{x}_0. \end{aligned}$$

Since there is no augmenting path in the residual network  $G_{\phi_h}[\varphi^*]$  displayed in Figure 5.3.b), we have arrived to a maximum flow  $\varphi^*$  of value 3.

The corresponding quadratic posiform of the network  $G_{\phi_h}[\varphi^*]$  is

$$\begin{aligned} \psi_h &= 2\bar{x}_4 \\ &\quad + 2x_1x_3 + x_1\bar{x}_4 + 2x_1x_5 + \bar{x}_1x_2 + \bar{x}_1x_4 \\ &\quad + x_2x_3 + x_2\bar{x}_4 + \bar{x}_2x_5 \\ &\quad + x_3x_4 + \bar{x}_3\bar{x}_4 + 2\bar{x}_3x_5 \\ &\quad + 2x_4x_5. \end{aligned}$$

Since  $C(\phi_h) = -7$ , and that  $v(\varphi^*) = 3$ , then  $\phi = C(\phi_h) + v(\varphi^*) + \psi_h = -4 + \psi_h$ , and hence  $-4 + \psi_h \in \mathcal{P}_2(h)$ .

Note that the dotted arcs appearing in Figure 5.3.b), i.e. those arcs which enter the source or leave the sink, have positive capacity but play no role in the analysis.

A  $x_0\text{-}\bar{x}_0$   $[S, \bar{S}]$ -cut in the residual network  $G_\phi[\varphi]$  is a partition of the node set  $N = \mathbf{L} \cup \{x_0, \bar{x}_0\}$  into two subsets  $S$  and  $\bar{S} = N - S$ , such that  $x_0 \in S$  and  $\bar{x}_0 \in \bar{S}$ . From the theory of network flows, it is well known that finding a maximum flow in  $G_\phi$  produces at least a minimum cut  $x_0\text{-}\bar{x}_0$   $[S, \bar{S}]$ -cut in the resulting residual network. Note that this cut does not have forward arcs in  $G_\phi[\varphi]$ . As a consequence of this operation, the number of strong components may increase. If this is the case (i.e. if  $S$  contains other nodes than  $x_0$ ), then the original posiform can be decomposed, and the posiforms associated to the strong components can be optimized separately. In fact, Boros and Hammer [54] proved the following result about the existence of persistent values in every minima of  $f_\phi$  for the literals of  $\phi$  (if any) belonging to the source side of the minimum cut.

**Proposition 5.9** (strong persistency ([54])). *Let  $\phi \in \mathcal{P}_2(f)$  for a quadratic pseudo-Boolean function  $f \in \mathcal{F}_2$ . Let  $\varphi^*$  denote a symmetric maximum flow in  $G_\phi$ , and let  $S \subseteq \mathbf{L}$  denote the set of nodes of  $G_\phi[\varphi^*]$  that are reachable from  $x_0$  via a path with positive residual capacities. Then,  $u(\mathbf{x}^*) = 1$  must be satisfied for all  $u \in S$  in every minimizer  $\mathbf{x}^* \in \text{Argmin}(f)$ .*

*Proof.* By symmetry  $\bar{S}$  contains a set of dual components (call it  $D$ ), corresponding to the strong components of  $S$ . Let  $G_S = (S, A_S)$ , be the subnetwork of  $G_\phi$  induced by  $S$ , and let  $G_D = (D, A_D)$  be the subnetwork of  $G_\phi$  induced by  $D$ . By duality,  $\phi_{G_S} = \phi_{G_D}$ . Next, we prove that there is a unique minimizer of  $\phi_{G_S}$  (or  $\phi_{G_D}$ ) with value equal to zero. Let us consider every arc with positive residual capacity, that was used to find the set  $S$ . Two types of arcs were used for this purpose:

1. If the arc (with positive capacity  $c_u$ ) is of type  $(x_0, u)$ , where  $u \in S$ , then there exists a term with positive coefficient  $c_u \bar{u}$  in  $\phi_{G_S}$ . In order for the term to vanish in any solution of  $\phi_{G_S} = 0$ , then  $u = 1$ .

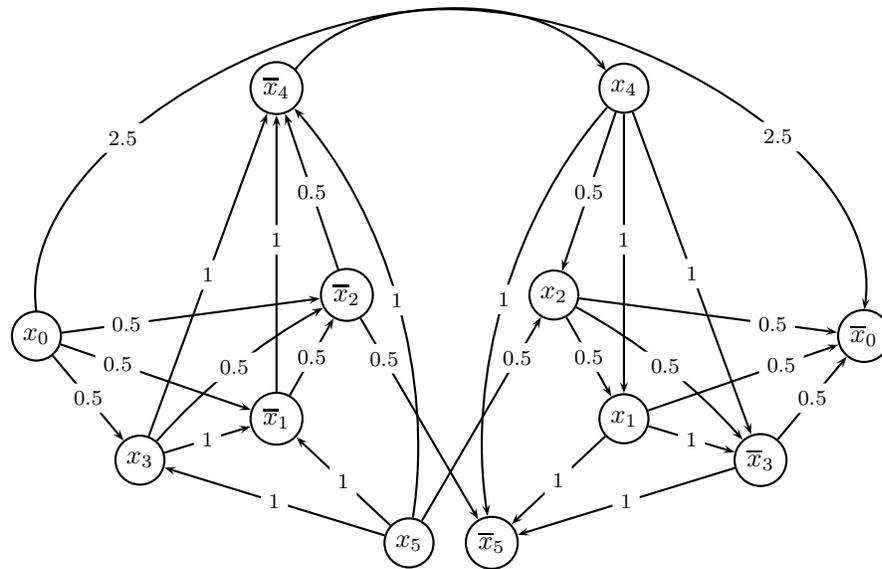
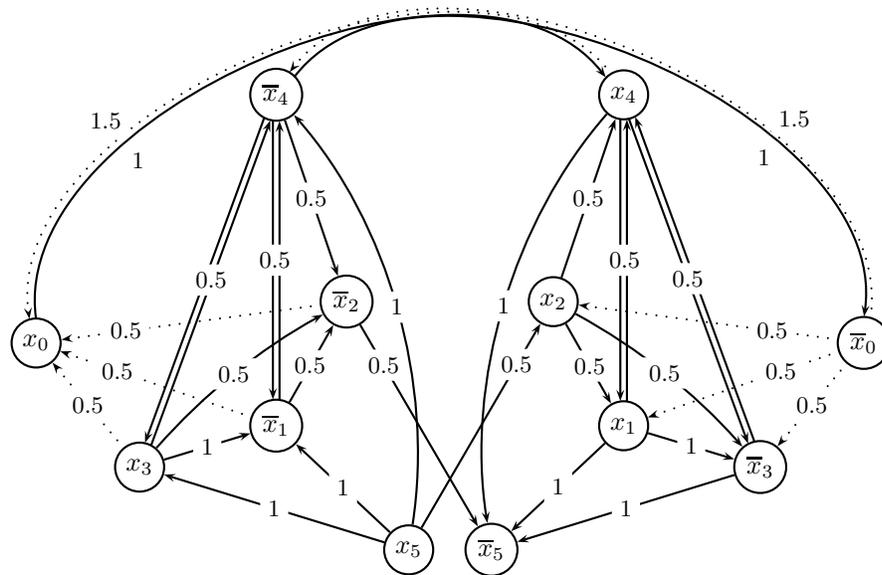
(a) The capacitated network  $G_{\phi_h}$ .(b) The residual network  $G_{\phi_h} [\varphi^*]$  with a maximum flow  $\varphi^*$ .

Figure 5.3: Capacitated networks of Example 5.4.

2. If the arc (with positive capacity  $c_{uv}$ ) is of type  $(u, v)$ , where  $u, v \in S$ , then there exists a term with positive coefficient  $c_{uv}u\bar{v}$  in  $\phi_{G_S}$ . Let us notice that this arc is followed by a previous arc  $(w, u)$ , where either  $w = x_0$  or  $w \in S$ . If  $w = x_0$  then apply 1) to obtain  $u = 1$ . With the additional constraint  $u\bar{v} = 0$ , then  $v = 1$ . If  $w \in S$  then recursively apply 2) to the arc  $(w, u)$ , which ultimately will imply that  $u = v = 1$ .

Thus, every literal in  $S$  must have a value equal to one so that  $\phi_{G_S} = 0$ . To prove that this is also a necessary condition for every minimizer of  $f$ , we have to check if all terms involved in the  $x_0\text{-}\bar{x}_0$   $[S, \bar{S}]$ -cut also have value zero in this solution. Indeed, every arc in the cut is an arc from a literal  $v \in \bar{S} \setminus D$  and a literal  $u \in S$ . The twin of arc  $(v, u)$  is the arc  $(\bar{u}, \bar{v})$  from  $\bar{u} \in D$  to  $\bar{v} \in \bar{S} \setminus D$ . Assuming that the capacity of each of these two arcs is  $\frac{c_{uv}}{2}$  ( $> 0$ ), then the corresponding term is  $c_{uv}v\bar{u}$ , and because  $u = 1$  the value of the term is also zero.  $\square$

**Example 5.5.** Consider the quadratic pseudo-Boolean function  $h = f_6(x_1, x_2, x_3, x_4, x_5, 1)$  of Example 5.4. From the residual network of Figure 5.3.b), the set  $S = \{x_4, x_1, \bar{x}_3, \bar{x}_5\}$  of literals that are reachable from the source  $x_0$  can be easily obtained. Since there is no augmenting path in the residual network, then every minimizer  $\mathbf{x} \in \text{Argmin}(h)$  must satisfy  $x_1 = x_4 = 1$  and  $x_3 = x_5 = 0$  for all literals in  $S$ . This partial assignment makes  $\phi_h = -4$ , where  $\phi_h$  is expression (5.8). Thus, the minimum of  $h$  is  $-4$ , and the corresponding minimizers are:  $(1, 0, 0, 1, 0)$  and  $(1, 1, 0, 1, 0)$ .

Notice that  $x_5$  is a pure literal in  $\phi_h$ . The pure literal rule of Lemma 4.2 could be used to prove that there is an optimal solution with  $x_5 = 0$ , but it can not be used to assure that all solutions have this value for  $x_5$ .

Let us remark that the minimum cut in  $G_\phi$  is not necessarily unique, and therefore the original problem can be further decomposed into several sub-problems, that are easier to be optimized.

**Theorem 5.3** (weak persistency). Let  $\phi \in \mathcal{P}_2(f)$  be a purely quadratic posiform, for a quadratic pseudo-Boolean function  $f \in \mathcal{F}_2$ . Let  $u \in L$  be a literal for which there is no path with positive capacities from  $u$  to  $\bar{u}$  in  $G_\phi$ . Let  $S \subseteq \mathbf{L}$  denote the set of nodes of

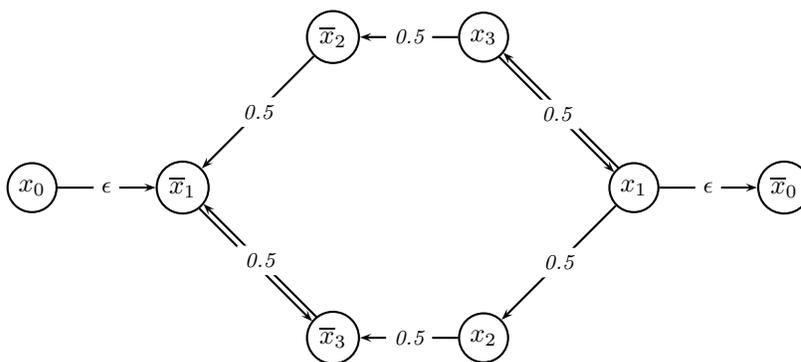
$G_\phi$  that are reachable from  $u$  via a path with positive residual capacities. Then, there is a minimizer  $\mathbf{x}^* \in \text{Argmin}(f)$  that satisfies  $u(\mathbf{x}^*) = 1$  and  $v(\mathbf{x}^*) = 1$  for all  $v \in S$ .

*Proof.* Since  $\phi$  does not contain linear terms, then the corresponding network  $G_\phi$  does not contain outgoing arcs from  $x_0$ , and by symmetry does not contain incoming arcs into  $\bar{x}_0$ . Let  $\varepsilon$  be a positive number. Let us add a linear term  $2\varepsilon\bar{u}$  to  $\phi$ , and call the resulting posiform as  $\psi (= 2\varepsilon\bar{u} + \phi)$ . Next, we find a symmetric maximum flow in the network  $G_\psi$ . If the maximum flow is positive, then there is a (augmenting) path with positive residual capacities from  $u$  to  $\bar{u}$ . So, the maximum flow must be zero to be in accordance with the theorem assumption. We now use the fact that  $\varepsilon$  can be very small, and therefore in the limit when  $\varepsilon$  approaches zero, Proposition 5.9 can be applied to the network  $G_\psi$ , and the corresponding (maximum) flow with value zero. The claim follows immediately.  $\square$

**Example 5.6.** Consider the quadratic pseudo-Boolean function

$$f(x_1, x_2, x_3) = 1 + (2 + \epsilon)x_1 + x_3 - x_1x_2 - 2x_1x_3 + x_2x_3.$$

A posiform that represents  $f$  is  $\phi = 1 + \epsilon x_1 + x_1\bar{x}_2 + \bar{x}_1x_3 + x_1\bar{x}_3 + x_2x_3$ . Consider the corresponding network  $G_\phi$ :



Since there is no augmenting path in  $G_\phi$ , then by Proposition 5.9 all the literals reachable from  $x_0$  must have value one, i.e.  $\bar{x}_1 = \bar{x}_3 = 1$  or  $x_1 = x_3 = 0$ , in every minima of  $f$ .

The values of the first derivatives of variables  $x_1$  and  $x_3$  are characterized as  $-1+\epsilon \leq \Delta_1 \leq 2+\epsilon$  and  $-1+\epsilon \leq \Delta_3 \leq 2+\epsilon$ . If  $\epsilon$  is chosen to be in  $[0,1[$  then the simple knowledge of the ranges of the first derivatives cannot be used to derive the persistencies found by roof-duality for  $x_1$  or  $x_3$ .

Through the assignment of any implied persistent results, Proposition 5.9 can be used to obtain a purely quadratic posiform with the same minimum value of the original posiform.

**Corollary 5.2.** *Let  $\phi \in \mathcal{P}_2(f)$  be a quadratic posiform, for a quadratic pseudo-Boolean function  $f \in \mathcal{F}_2$ . Let  $\varphi^*$  denote a symmetric maximum flow in  $G_\phi$ , and let  $S \subseteq \mathbf{L}$  denote the set of nodes of  $G_\phi[\varphi^*]$  that are reachable from  $x_0$  via a path with positive residual capacities. Let  $D = \{\bar{u} \mid u \in S\}$ . Let  $G = (N \setminus (S \cup D), A_S)$  be a subnetwork of the residual network  $G_\phi[\varphi^*]$  induced by the nodes in  $N$ , not belonging to  $S$  or  $D$ . Then,  $\phi_G$  is a purely quadratic posiform for which  $\nu(f) = \nu(f_{\phi_G})$ . Further, if  $V_S$  is the set of variables appearing in  $S$ , then a weak persistency holds for  $f$  at  $\mathbf{y} \in \text{Argmin}_{\mathbb{B}^{V \setminus V_S}}(f_{\phi_G})$ .*

The final comment of this section is to remark once again, the importance that the employment of max-flow techniques have in the optimization of quadratic pseudo-Boolean functions. Several persistent results can be asserted just by finding the source side of a minimum cut. It also provides an equivalent purely quadratic posiform  $\phi$  with a larger constant term, and therefore, it gives better lower bounds to the minimum of the corresponding quadratic pseudo-Boolean function  $f_\phi$ . Interestingly, this bound called *roof-dual* is equal to the bound returned by other alternative (linear programming) techniques ([54, 123]), computationally more demanding than the network flow model approach presented here.

## 5.4 Computational results

The network flow model presented in the previous section, which represents a quadratic pseudo-Boolean function  $f$ , has been implemented as a computer program. It consists of  $2n + 2$  nodes, one for each literal, and  $2m$  arcs, two for each nonzero term of the multilinear representation of  $f$  (see Section 5.3).

The data structure adopted is a special network, where each node contains of a list of incoming arcs and a “map” of outgoing arcs.

A *map* is a dictionary collection that maps unique keys (i.e. literals) to values (i.e. nodes). Once a key-value pair has been inserted into the map, then the pair can efficiently<sup>1</sup> be retrieved or deleted using the key to access it. One can also iterate over all the elements in the map in constant time per element.

With this structure is possible to get the list of all terms involving any literal in  $O(\log(n) + n)$  time, and is possible to create, to delete or to get the coefficient of any quadratic term in  $O(\log(n))$ .

A maximum flow algorithm applied over the network just described, has been implemented to compute the roof-dual value of  $f$ . The maximum flow implementation that has been considered is based on the shortest augmenting path algorithm, yielding a worst case time of  $O(n^3)$ , and is especially designed to deal with the existence of the flow symmetry conditions ([134]).

All algorithms based on the network flow model were implemented in C++, compiled using the Microsoft Windows 32-bit C/C++ Optimizing Compiler (version 12) for 80x86, and linked with the Microsoft Incremental Linker (version 6).

In our computational experiments of this section we present two types of results. In subsection 5.4.1 we demonstrate both the effectiveness and efficiency of our implementation to derive roof-duality consequences, by comparing it to the one achieved through the use of linear programming (see 5.4.1). In subsection 5.4.2 we first show that roof-duality can deal with very large problems, which have some special characteristics, by producing both a large number of persistencies, and near-optimal bounds.

Roof-duality is a key tool used frequently throughout this dissertation. Many other computational results based on roof-duality algorithms, which are associated to various types of applications, are presented in the chapters that follow.

---

<sup>1</sup>If the map has  $r$  entries then an element can be either found or proved not to exist in the map in  $O(\log_2(r))$

### 5.4.1 Network flow model versus linear programming

An important aspect about roof-duality is the fact that the Linear Programming (LP) solution of problem (5.3) is half-integral ([30]). Moreover, Balinski [30] has also shown that all basic feasible solutions are also  $(0, \frac{1}{2}, 1)$ -valued. This information could be used by special designed linear programming algorithms, so that the solve times of problem (5.3) could be made more efficient. This can be particularly important for solvers based on interior point methods, for which a reasonable good starting point can provide faster solve times. This possibility was not tried here, since the available linear programming solvers that we had at hand did not include this possibility.

All the experiments of this section were found using the same computer system, which is based on a Xeon 3.06 GHz, 3.5 GB RAM and Windos XP.

We have considered two methods to solve the LPs. One is the Newton-barrier algorithm and the other is the simplex dual algorithm. Both algorithms are part of the mathematical programming software package Xpress-MP. The presolve and the crossover algorithm of the Newton-barrier solver (version 2006B) were turned off in all runs, since we noticed that they would slow down the computing times for the problems that we have tested in this section.

For testing our implementation of the network flow model against the LP solvers, we have considered two families of problems:

- The  $G_1$  group consists of maximization QUBOs with 1 000 variables and densities varying from 10% to 100% (in steps of 10%). The best known solutions are given in Table A.3 of the Appendix. This group of problems is used to compare the effect played by the density ( $d$ ) parameter on the outcome of the algorithms.
- The 60 Beasley maximization QUBOs, which have a 10% density each. The number of variables  $n$  varies from 50 to 2 500; 10 problems were created for each value of  $n$ . This group of problems is used to compare the effect played by the number of variables ( $n$ ) parameter on the outcome of the algorithms.

Table 5.1: Network flow model versus LP to find the roof-duals of the  $G_1$  problems proposed by Glover et al. [109].

Problem Number	Density (d%)	Roof-Dual Value	Roof-Dual Computing Time using		
			LP Newton-barrier*	LP Dual*	Network Flows
1	10	587 424.0	5.3 s	70.7 s	0.2 s
2	20	1 186 105.0	8.1 s	290.9 s	0.4 s
3	30	1 772 322.5	10.8 s	694.2 s	0.7 s
4	40	2 360 450.0	13.1 s	1 309.3 s	1.0 s
5	49	2 957 813.0	14.4 s	2 222.2 s	1.5 s
6	60	3 565 800.0	16.8 s	2 963.6 s	2.0 s
7	69	4 159 309.0	19.3 s	4 037.4 s	2.7 s
8	79	4 743 848.5	22.4 s	5 388.2 s	3.6 s
9	89	5 330 495.0	26.3 s	7 271.1 s	4.6 s
10	99	5 933 962.5	27.6 s	9 218.0 s	5.6 s

Table 5.1 gives the  $G_1$  roof-dual computing times of the 3 algorithms that we have considered to find the roof-dual bound. It can be seen in this table that our implementation of the network flow model is faster than the LP Newton-barrier solver. Namely, it is

- 26 times faster for the instance with 10% density;
- 10 times faster for the instance with 50% density; and
- 5 times faster for the full dense instance.

This trend of results indicates that the network flow model is considerably faster than LP Newton-barrier in computing the roof-dual bound. These facts also show that the efficiency of the network model is more noticeable for sparser QUBOs, which appear frequently in real world applications (see Chapter 10).

The LP simplex dual solver is somewhat slower in finding the roof-dual bound for the  $G_1$  instances. It should be remarked the fact that we have also tried the crossover to an optimal basic feasible solution of the LP Newton-barrier solution. The conclusion about using this option is that it was even slower than the option of using the dual algorithm (which always produces an optimal basis).

Table 5.2 provides the average computing times to find the roof-duals of the problems belonging to the Beasley group of problems. The results show that the network flow model implementation is 10 to 30 times faster than the LP Newton-barrier solver, and

Table 5.2: Average relative gap ( $g$ ) to the best known lower bound ( $z$ ) and average computing times of the roof-duals of 10% dense QUBO problems (Beasley [37]).

Family	Variables ( $n$ )	Roof-Dual Gap ( $g = \frac{p-z}{z}$ )	Roof-Dual Computing Time using		
			LP Newton-barrier	LP Dual	Network Flows
ORL-50	50	0.1%	<0.05 s	<0.05 s	<0.05 s
ORL-100	100	15.3%	<0.05 s	<0.05 s	<0.05 s
ORL-250	250	78.1%	0.1 s	0.2 s	<0.05 s
ORL-500	500	150.6%	0.6 s	4.2 s	0.05 s
ORL-1000	1 000	248.8%	5.3 s	68.4 s	0.2 s
ORL-2500	2 500	430.4%	43.8 s	1 889.0 s	1.5 s

that the speedup of the network model increases with an increase on the number of variables. For this group of problems the LP dual solver is still the slowest option, but the results show that the simplex algorithm is somewhat better for sparser problems.

The roof-dual values (and gaps) presented in the previous tables are also clear indication that roof-duality is mostly effective for QUBOs with a sparse structure. We remark once again that many QUBOs derived from practical applications are sparse.

#### 5.4.2 Application of roof-duality to VLSI design

The purpose of this section is twofold. The first objective is to stress out the importance of QUBO in VLSI design. The second objective is to illustrate that roof-duality can be a powerful tool for certain structured combinatorial optimization problems.

Quadratic optimization has been used for a long time in the field of LSI and PBCs (Printed Circuit Boards) (e.g. [32, 52, 55, 85, 114, 152]). In the sections that follow we shall consider two problems in VLSI design. First, we investigate the impact of roof-duality in some MAX-CUT problems derived from via minimization problems, which are part of the layer assignment phase of channel routing. Second, we look at the so called module flipping problem considered in the placement phase of the layout design of circuits. We propose a random generator of a particular family of these problems, and analyze the impact of roof-duality in various instances.

### 5.4.2.1 Via minimization

The physical layout design of integrated circuits is usually split up into the placement, routing, layer assignment and compaction phases. We assume that modules placement and routing has been already completed. We concentrate on QUBO problems derived from the layer assignment phase, whose objective is to assign wire segments to layers such that intersection segments belonging to different nets are assigned to different layers. Wires of a net on different layers are connected by *vias*. Vias need additional space and they create difficulties during the compaction phase. The *via minimization* problem consists in finding a layer assignment such that the number of vias is as small as possible.

We assume that the *transient* routing has been found, i.e. that all cells are placed on the chip and that all nets have been routed, but the assignment of wire segments to layers has not been performed yet. A net may connect two or more pins. In the later case, the net may contain 3-way junctions and more rarely it may also contain 4-way junctions.

From the transient routing, the via minimization problem (for two-layers) is transformed into an equivalent MAX-CUT problem of the so called *layout graph* (see e.g. [83, 114, 202]).

If the transient routing contains no  $k$ -junctions for  $k \geq 4$ , then in the two-layers case the layout graph is planar. Consequently the MAX-CUT of this class of graphs can be found in polynomial time ([117]), thus implying that the via minimization problem can be efficiently found for these cases ([83, 202]).

Certain side constraints are required in practice. Frequently, one of the two layers is preferred and pins are preassigned to a specific layer ([32, 114]). The previous MAX-CUT reduction can be generalized to the via minimization problem subject to layer preference and pin preassignments ([32]). In this case however the MAX-CUT that results from this reduction is NP-hard ([32]; see also [180]).

In Table 5.3 we considered MAX-CUT instances derived from layout graphs provided by Homer and Peinado [145]. There are two groups of five graphs each:

- All five problems of the group *via.cy* problems are solved optimally by using the strong persistency property of roof-duality, i.e. for each problem the bound coincides with the optimum and the residual problem is a small satisfiable quadratic Boolean expression. The largest residual problem in this group has 36 binary variables only.
- Strong persistency is not so effective on the group *via.cn* set of problems, since only a few variables are fixed by roof-duality in the associated QUBO. The average gap to the optimum from the roof dual bound is relatively small since it varies between 3% and 6%.

Due to the previous results, it is not surprising that all VIA problems considered here, could be solved efficiently by using state-of-the-art solvers to solve the standard (roof-duality) mixed-integer program (5.3). In Chapter 7 we will demonstrate that all of these ten problems can be entirely solved very quickly just by using preprocessing techniques for QUBO, without the support of any type of branching or enumeration procedure. This is particularly important for VIA minimization problems of large dimensions.

Table 5.3: Via minimization problems of Homer and Peinado [145].

<i>Problem</i>	<i>Vertices</i> ( <i>n</i> )	<i>Edges</i>	<i>Maximum</i> ( <i>z</i> )	<i>Roof-Dual</i>			
				<i>Persist.</i>	<i>Time</i>	<i>Value</i> ( $\rho$ )	<i>Gap</i> ( $g = \frac{\rho-z}{z}$ )
via.c1n	828	1389	6150	96	0.06 s	6339	3.1%
via.c2n	980	1712	7098	7	0.08 s	7473	5.3%
via.c3n	1327	2393	6898	13	0.12 s	7282	5.6%
via.c4n	1366	2539	10098	10	0.09 s	10437	3.4%
via.c5n	1202	2129	7956	6	0.06 s	8427	5.9%
via.c1y	829	1693	7746	814	0.05 s	7746	0.0%
via.c2y	981	2039	8226	957	0.06 s	8226	0.0%
via.c3y	1328	2757	9502	1315	0.09 s	9502	0.0%
via.c4y	1367	2848	12516	1341	0.09 s	12516	0.0%
via.c5y	1203	2452	10248	1167	0.09 s	10248	0.0%

#### 5.4.2.2 Cell flipping in standard cell technology

In the layout stage of VLSI and printed circuit board (PCB) design, after all circuit modules (rectangular) are placed, it is possible to flip the modules so as to reduce the

total net length([55, 85]). Cheng et al. [85] formulate the orientation of modules as a graph problem and prove it to be NP-complete. The orientation problem is shown to be equivalent to MAX-CUT of a graph ([85]; see also [52]). Experiments with real cases show that module orientation reduces the total net length and improves the routability ([85]).

After the routing phase in the layout design of VLSI and PBCs, the optimal wiring has to be decided. Direct connections have to be established between certain *pairs* of pins belonging to different modules. The pins have fixed locations on the rectangular perimeter of the module. Since each module can be placed on the base plate in four different positions occupying the same rectangular area, then the total length of the wiring depends substantially on their positioning. The different placements of a module can be achieved by flipping it either vertically, horizontally, or in both directions. The problem of finding the flipping positions of the modules which minimize the total net length is the *flipping problem*.

A random generator of 2-pin cell flipping problems has been created. For simplicity, we only consider a simplified version of the rectangular modules, by allowing only vertical flippings and assuming that the modules are simply segments of a line. The generator has four parameters that have to be specified:

$n$  – Number of 2-pin cells;

$m$  – Number of nets, which connect two 2-pin cells;

$H$  – Horizontal size of the chip;

$k$  – A seed to feed to the pseudo-random numbers generator.

The starting horizontal location ( $L_i$ ) and size ( $S_i$ ) of the  $i$ -th cell is uniformly determined from the interval  $[0, H]$ , i.e.

$$\begin{aligned} A_i &= \text{Uniform}(0, H), \\ B_i &= \text{Uniform}(0, H), \\ L_i &= \min(A_i, B_i), \\ S_i &= \max(A_i, B_i) - L_i, \end{aligned}$$

for every cell  $i = 1, \dots, n$ .

Each net  $(i, j)$  has an origin cell  $i$  and a destination cell  $j$  ( $\neq i$ ) randomly chosen. The extremity of the cell (i.e., left or right) which is used by the net in both endpoints is also randomly determined. The “origin” extremity is denoted with the binary indicator  $o_{i,j}$  (is equal to 0 for left extremity, and 1 otherwise), and similarly the “destination” extremity is denoted with the binary indicator  $d_{i,j}$ .

Let us further define  $I(i, 0) = A_i$  and  $I(i, 1) = B_i$  for every cell  $i = 1, \dots, n$ .

The objective is to minimize the total horizontal wiring needed to connect all the 2-pin cells by doing cell flipping, i.e.

$$\min_{\mathbf{x} \in \mathbb{B}^n} \sum_{(i,j) \in \mathcal{N}} \left( |I(i, o_{i,j}) - I(j, d_{i,j})| x_i x_j + |I(i, \bar{o}_{i,j}) - I(j, d_{i,j})| \bar{x}_i x_j + |I(i, o_{i,j}) - I(j, \bar{d}_{i,j})| x_i \bar{x}_j + |I(i, \bar{o}_{i,j}) - I(j, \bar{d}_{i,j})| \bar{x}_i \bar{x}_j \right),$$

where  $\mathcal{N}$  is the set of  $m$  nets, and the binary decision  $x_i$  is 0 if cell  $i$  has to be flipped or 1 otherwise.

The first set of experiments is based on problems randomly created with the previous generator. It considers instances with an horizontal size  $H$  of 1 000 having 1 000 000 nets each, and whose number of cells  $n$  is either 250 000, 500 000 or 750 000.

From the results of Table 5.4 it can be seen that the roof duals of the 1 million nets problems can be computed in about 2 minutes.

The number of strong persistencies of these QUBO problems is around 675 and is somewhat independent of the number of cells  $n$ .

Weak persistencies were found by applying the pure literal rule (see Lemma 4.2) to the residual posiform of the QUBO problem obtained after applying strong persistency. Interestingly, the number of weak persistencies that were found in this way is very large. Namely, it is over 87% of the original cells for problems with 250 000 cells, and it is 96% of the original cells for problems with 750 000 cells. It should be remarked that the computing times could be improved if the pure literal rule is applied both initially and at the end of the method. In fact, most of the pure literals can be detected initially in this particular family of QUBO problems.

Table 5.4: Impact of roof-duality on large 2-pin cell flipping randomly generated problems having 1 000 000 nets each.

<i>Problem</i>	<i>Cells</i> ( <i>n</i> )	<i>Nets/Cells</i> ( $\frac{m}{n}$ )	<i>Roof-Dual</i>				<i>Residual QUBO</i>		
			<i>Strong Pers. (s)</i>	<i>Pure Lit. (p)</i>	<i>Time*</i>	<i>Value (ρ)</i>	( <i>n'</i> = <i>n</i> - <i>s</i> - <i>p</i> )	( $\frac{n'}{n}$ )	<i>Quad. Terms</i>
fliflop-250K-1M-1	250 000	4	691	217 259	126 s	266 282 720.0	32 050	12.8%	655 312
fliflop-250K-1M-2			692	217 270	130 s	266 148 770.5	32 038		655 978
fliflop-250K-1M-3			614	217 262	127 s	264 853 326.0	32 124		662 568
fliflop-250K-1M-4			665	217 264	128 s	265 823 910.5	32 071		656 688
fliflop-250K-1M-5			679	217 274	128 s	266 445 939.0	32 047		654 725
fliflop-500K-1M-1	500 000	2	688	467 262	128 s	266 350 120.5	32 050	6.4%	654 964
fliflop-500K-1M-2			658	467 265	128 s	266 093 176.5	32 077		656 528
fliflop-500K-1M-3			628	467 262	130 s	264 931 318.0	32 110		662 248
fliflop-500K-1M-4			660	467 262	128 s	265 835 648.0	32 078		656 960
fliflop-500K-1M-5			686	467 274	131 s	266 600 282.0	32 040		654 333
fliflop-750K-1M-1	750 000	1.33	684	717 259	129 s	266 337 871.5	32 057	4.3%	655 260
fliflop-750K-1M-2			687	717 269	132 s	266 081 472.0	32 044		656 003
fliflop-750K-1M-3			649	717 263	126 s	264 954 628.5	32 088		662 022
fliflop-750K-1M-4			675	717 261	128 s	265 808 816.0	32 064		656 604
fliflop-750K-1M-5			710	717 274	131 s	266 646 530.5	32 016		653 744

\*Obtained on an Intel Xeon 3.06GHz.

Also interesting, is the fact that the residual size of the QUBO problems, after applying both weak and strong persistency, is somewhat constant, consisting of about 32 000 variables and about 655 000 nonzero quadratic terms.

To measure the impact of roof-duality in solving 2-pin cell problems of this nature, a set of additional instances has been generated, each one having 10 000 nets and an horizontal size  $H$  of 1 000 as before. The number of cells varies between 1 000 and 7 500 in steps of 500. Five distinct instances were created for each combination of  $n$ ,  $m$  and  $H$ . When not solving completely a given 2-pin cell problem, the quality of the roof-dual bound has been compared with that one provided by a one-pass heuristic (see Chapter 6).

Table 5.5 shows that roof-duality delivers optimal solutions for all problems having more than 4 000 nets. As soon as the density ( $d$ ) starts increasing, the relative gap ( $g$ ) associated to the roof-dual increases rapidly from 0.01% to 6.5%, respectively for the 3 500 and 1 000 cells cases.

Table 5.5: Average relative gap of roof-duality on randomly generated 2-pin cell flipping problems having 10 000 nets.

<i>Cells</i> ( $n$ )	<i>Nets/Cells</i> ( $\frac{m}{n}$ )	<i>Density</i> ( $d$ )	<i>Roof Dual</i> ( $\rho$ )	<i>Upper bound</i> ( $z$ )	<i>Rel. Gap</i> ( $\frac{z-\rho}{z}$ )
1,000	10.0	2.00%	2,661,363	2,845,028	6.46%
1,500	6.7	0.89%	2,623,640	2,745,590	4.44%
2,000	5.0	0.50%	2,599,938	2,661,745	2.32%
2,500	4.0	0.32%	2,573,110	2,607,212	1.31%
3,000	3.3	0.22%	2,502,887	2,511,469	0.34%
3,500	2.9	0.16%	2,445,970	2,446,239	0.01%
4,000	2.5	0.13%	2,398,005	2,398,005	0.00%
4,500	2.2	0.10%	2,368,370	2,368,370	0.00%
5,000	2.0	0.08%	2,299,954	2,299,954	0.00%
5,500	1.8	0.07%	2,259,256	2,259,256	0.00%
6,000	1.7	0.06%	2,217,748	2,217,748	0.00%
6,500	1.5	0.05%	2,192,810	2,192,810	0.00%
7,000	1.4	0.04%	2,151,678	2,151,678	0.00%
7,500	1.3	0.04%	2,115,016	2,115,016	0.00%

This trend of results linked to QUBOs of low or high density, is typically the one that determines if roof-duality is or not is a good tool to solve a given QUBO problem.

This particular family of QUBO problems is also interesting by the fact that roof-duality impacts differently for flipping problems of 10 000 nets and for flipping problems of 1 000 000 nets. In the smaller instances, after applying roof-duality, the residual problems are minuscule, whereas for the large instances the number of quadratic terms in the residual problems is about  $\frac{2}{3}$  of the number of nets.

## Chapter 6

### Heuristics

Several exact methods have been developed and tested for QUBO in the literature (see Chapter 9). Since however QUBO is known to be NP-hard (see [104]) many of the large problems arising from practical applications proved to be not tractable for these exact approaches. Several heuristic algorithms, based on different ideas, were proposed recently in the literature to find acceptable solutions for such large problems.

The heuristics proposed in the past for QUBO can be broadly classified in three groups:

- The *one-pass* heuristics (see e.g. [58, 107, 178]) are based on polynomial time algorithms, which assure solutions with “reasonable” quality in “very good” computing times, for “very large” problems (up to tens of thousands of variables).
- The *local-search* heuristics (see e.g. [62, 178]; sometimes called 1-opt heuristics) are based on the exploration of directions of improvement, within a simple and well defined neighborhood of solutions. This class of heuristics provides a solution with “good” quality in a typically “good” computing time for “large” problems (up to several thousands of variables), but it does not provide a provably polynomial running time.
- The *meta-heuristics* are based on the search of a well defined neighborhood of other solutions, providing a solution with “very good” quality in a usual “reasonable” amount of computing time for “reasonable” sized problems (up to a few thousands of variables). In many cases, the local-search methods are used as subroutines in this class.

In this chapter we consider the pseudo-Boolean minimization problem (1.4) in  $\mathbb{B}^n$ ,

that is the problem of minimizing a pseudo-Boolean function  $f$  over the set  $\mathbb{B}^n$  of binary  $n$ -vectors.

## 6.1 One-pass heuristics

One drawback of local search based approaches (see Section 6.3) is that they do not provide guarantees – neither for running times, nor for solution quality. While we can have little hopes for the second type of guarantees with a quick, local search type approach, running time guarantees can easily be achieved by simply terminating earlier the search process. Such approaches appear in the literature and are called sometimes *one-pass* or *greedy* procedures (see e.g. [107, 178]).

### 6.1.1 DDT methods

We start by presenting the family of DDT heuristics introduced by Boros, Hammer and Sun [58], which is one of the earliest one-pass approaches presented for QUBO. DDT consists of three stages: Devour, Digest and Tidy-up. In the “devour” stage, a linear or quadratic logical relation is produced. In the “digest” stage all the logical consequences of the previous step are derived. In the “tidy-up” step the new logical conclusions are enforced by transforming the function being optimized accordingly.

The DDT framework can be described in a more general way, which includes not only the original methods, but it also covers a family of recently proposed one-pass procedures (see Section 6.1.2). The general idea of the DDT algorithm is described in Figure 6.1.

At each iteration of the DDT algorithm, a 0–1 value is assigned to a certain “elementar” quadratic expression  $\Psi$  defined by method  $\mathbb{X}$ .  $\Psi$  has to be chosen in such a way that the resulting set of logical relations is *satisfiable* (i.e.  $\Phi \vee \Psi = 0$  is consistent), and such that the size of the set of solutions of the equation of  $\Phi \vee \Psi = 0$  is *strictly* smaller than that of the equation  $\Phi = 0$ , i.e.

$$\left\{ \mathbf{x} \in \mathbb{B}^{\mathbf{V} \setminus (U \cup Z)} \mid (\Phi + \Psi)(\mathbf{x}) = 0 \right\} \subset \left\{ \mathbf{x} \in \mathbb{B}^{\mathbf{V} \setminus (U \cup Z)} \mid \Phi(\mathbf{x}) = 0 \right\}.$$

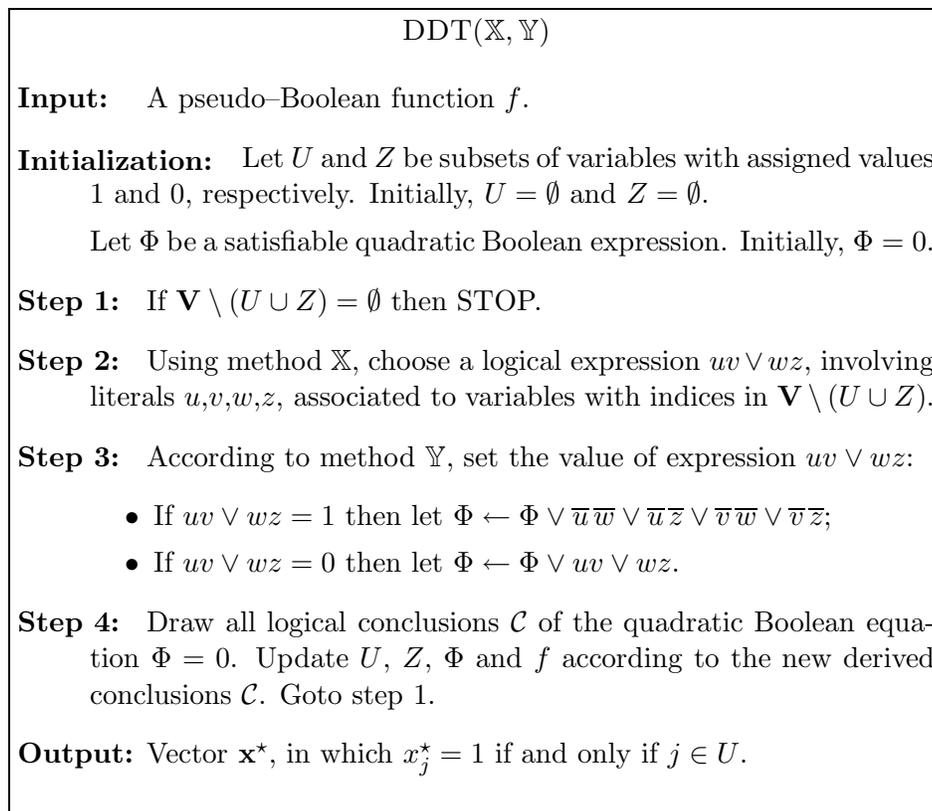


Figure 6.1: The DDT heuristics.

Boros et al. [58] applied the DDT method to quadratic pseudo-Boolean functions represented by a posiform (1.6). They proposed two variants of method  $\mathbb{X}$ :

- (i) Select a term  $ab$  with the largest weight of a given posiform  $\phi_f$ . In this case  $uv = ab$  and  $wz = ab$  in step 2 of DDT.
- (ii) Select a bi-term  $\bar{a}\bar{b} + \bar{a}b$  with largest weight in the bi-form of  $f$  (see Section 8.1). In this case  $uv = \bar{a}\bar{b}$  and  $wz = \bar{a}b$  in step 2 of DDT.

The value assigned to expression  $\Psi$ , which was determined by method  $\mathbb{X}$ , is defined by method  $\mathbb{Y}$ . For the Boros et al. [58] DDT methods,  $\Psi$  is always assumed to be equal to zero. Let us note that in case (ii) above, this choice leads to an assignment of the type  $a = b$  for literals  $a$  and  $b$  (because  $a = b \Leftrightarrow \bar{a}\bar{b} + \bar{a}b = 0$ ). This choice implies in this case that the next iteration of DDT has one less variable, and thus that DDT ends after  $n$  iterations.

Step 4 of DDT correspond to the “digest” and “tidy-up” phases. The complexity of doing these operations is basically associated to the complexity of finding all strong persistencies for quadratic Boolean equations. There are several ways to accomplish this task efficiently (see e.g. [133]). We have adopted in this study the implication graph algorithm of Aspvall et al. [22] (see Section 4.5 for details) in our implementation of case (i) above of the DDT method.

**Proposition 6.1** ([58]). *The DDT method (i) of Boros, Hammer and Sun [58] generates a heuristic solution to the minimum of a quadratic pseudo-Boolean function  $f$  given as quadratic posiform  $\phi_f$  (1.6) in  $O(m^2)$  time, where  $m$  is the number of (nonzero) terms of  $\phi_f$ .*

*Proof.* Since the total number of terms of  $\phi_f$  is  $m$ , then the DDT method runs in  $m$  iterations. Let us now see how much time DDT takes in each iteration. In step 2 the search for the largest coefficient of a nonzero term of  $\phi_f$  takes at most  $O(m)$  steps. By using the set of rules of Lemma 4.1, Step 4 uses at most  $O(n)$  steps to derive the consequences of assigning a term to zero. To update both  $\phi_f$  and the implication graph data structures we need at most  $O(n)$  steps for each variable fixed. Putting these results together we obtain the claimed  $O(m^2)$  time.  $\square$

**Proposition 6.2** ([58]). *The DDT method (ii) of Boros, Hammer and Sun [58] generates a heuristic solution to the minimum of a quadratic pseudo-Boolean function  $f$  given as a bi-form  $\beta_f$  in  $O(nm)$  time, where  $m$  is the number of (nonzero) terms of  $\beta_f$ .*

*Proof.* We have already seen that this method takes at most  $O(n)$  iterations. As in the previous case the amount of time needed for each iteration is at most  $O(m)$ , thus generating the claimed  $O(nm)$  time.  $\square$

The DDT methods can be implemented in such a way that the *average* computing time is reduced considerably. The bottleneck of DDT occurs during the search of a large coefficient term in step 2. To improve the time of this operation we used a data structure which includes for each non-fixed variable  $x_j$ , the largest coefficient of a term

where  $x_j$  appears. Clearly, the search for a large coefficient term is now  $O(n)$  instead of  $O(m)$ . The time needed to update the structure for each variable fixed is  $O(n)$  on average, and for each quadratic term fixed is  $O(1)$ . Adding these times, it can be seen that the DDT method could be implemented to run in  $O(nm)$  average time for case (i), and in  $O(n^2)$  average time for case (ii).

As a final remark of this section, it is important to note that the heuristic value of case (i) of Boros et al. [58] is highly dependent on the posiform selected as input to the algorithm, whereas case (ii) is invariant with respect to the input, since the bi-form is uniquely defined for each quadratic pseudo-Boolean function (see Section 8.1).

### 6.1.2 Greedy heuristics

In the family of one-pass algorithms proposed in this section, the variables get binary assignments one by one, until a binary value is assigned to every variable, at which time the procedure stops. The algorithm invokes two methods: method  $\mathbb{X}$  is used to choose the index of the next variable to which a binary value will be assigned, while method  $\mathbb{Y}$  decides which of the two possible assignments is more advantageous.

This type of algorithms can be seen as special cases of the DDT method introduced in the previous section, but since the assignment of 0–1 values to *quadratic* expressions is not considered in this family of heuristics, we provide the general approach of the considered algorithms in Figure 6.2.

At a given iteration of ONE-PASS, with given methods  $\mathbb{X}$  and  $\mathbb{Y}$ , we shall frequently use the partial assignment  $\mathbf{y}^{U,Z}$  defined by

$$y_j^{U,Z} = \begin{cases} 1 & \text{if } j \in U \\ 0 & \text{if } j \in Z, \end{cases}$$

which denotes a binary vector containing the partial assignment induced by the sets  $U$  and  $Z$ .

In the following sections we consider three particular methods  $\mathbb{X}$  and  $\mathbb{Y}$ . The first choice is based on best linear approximation methods. The second choice is based on

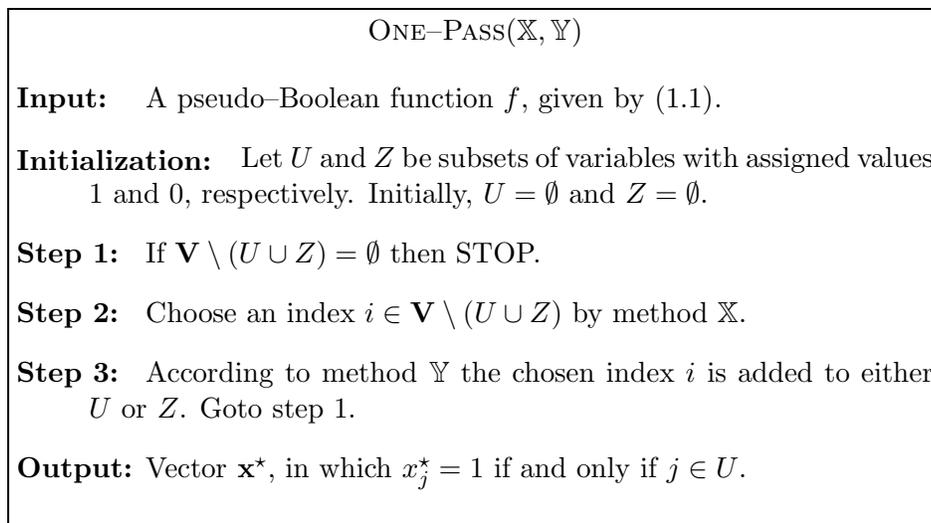


Figure 6.2: Algorithm description of one-pass heuristics.

probabilistic assumptions on the partial derivatives of the function. The last choice is based on rounding methods.

### 6.1.2.1 Best linear approximation methods

The first greedy heuristic that we present is based on finding a “best” linear approximation (see Section 4.8) to a pseudo-Boolean function  $f$ , and using then the coefficients of the linear terms of this function to infer a measure of the variables’ contribution to the optimal value of the original function  $f$ . The basic idea is that a variable whose linear coefficient is large in *absolute* value has a potentially larger impact on the value of  $f$ , and therefore fixing this variable (to 1 if the coefficient is positive, and to 0 if it is negative) may have a high impact on the value to the function.

For the purpose of describing the one-pass methods  $\mathbb{X}$  and  $\mathbb{Y}$  of this subsection, we shall assume that the best linear approximation is given as

$$A(f(\mathbf{x}[U \cup Z \leftarrow \mathbf{y}^{U,Z}]))) = a_0^{U,Z} + \sum_{j \in \mathbf{V} \setminus (U \cup Z)} a_j^{U,Z} x_j.$$

Let

$$S^{U,Z} \stackrel{\text{def}}{=} \arg \max_{j \in \mathbf{V} \setminus (U \cup Z)} \left\{ |a_j^{U,Z}| \right\}.$$

denote the set of variables whose coefficients are largest in absolute value.

The ONE-PASS( $\mathbb{X}, \mathbb{Y}$ ) heuristic method of this subsection can be described as follows:

$$\begin{array}{ll} \mathbb{X} \equiv & i \leftarrow \min \{S^{U,Z}\}, \\ \mathbb{Y} \equiv & \text{if } a_j^{U,Z} \leq 0 \text{ then } U \leftarrow U \cup \{i\} \text{ else } Z \leftarrow Z \cup \{i\}. \end{array}$$

Method  $\mathbb{X}$  selects from the set of variables in  $\{\mathbf{V} \setminus (U \cup Z)\}$  which have the largest coefficients in absolute value  $|a_j^{U,Z}|$ , the one which has the smallest index  $j$ . Since we are considering minimization problems, the value of  $x_j$  determined by subroutine  $\mathbb{Y}$  is 1 if  $a_j^{U,Z} \leq 0$ , and 0 otherwise.

### Best linear $L_2$ -approximation

The best Euclidean linear approximation of a pseudo-Boolean function was derived by Hammer and Holzman [125] (see Proposition 4.8 of Section 4.8). The linear coefficients of the best Euclidean linear approximation are simply the values associated to the corresponding first derivatives in the point  $(\frac{1}{2})^{\mathbf{V} \setminus (U \cup Z)}$ , associated to the variables which were not yet fixed by ONE-PASS, i.e.

$$a_j^{U,Z} = \Delta_j \left( \left( \frac{1}{2}, \dots, \frac{1}{2} \right) [U \cup Z \leftarrow \mathbf{y}^{U,Z}] \right), \quad (6.1)$$

for all  $j \in \mathbf{V} \setminus (U \cup Z)$ .

**Example 6.1.** *Let us consider the application of ONE-PASS based on the best Euclidean linear approximation to the hitting set problem (2.4), and demonstrate that the classical greedy algorithm for this problem is equivalent to this one-pass heuristic. Let us denote by  $d_{\mathcal{H}}(v)$  the degree of vertex  $v$  in the hypergraph  $\mathcal{H}$ , i.e.  $d_{\mathcal{H}}(v) = |\{H \in \mathcal{H} \mid H \ni v\}|$ , and let  $D_{\mathcal{H}} = \max_{v \in \mathbf{V}} d_{\mathcal{H}}(v)$ . In the pseudo-Boolean formulation (2.4) we shall consider  $\epsilon_H = -1 + 2^{|H|-1}$  for every subset  $H \in \mathcal{H}$ . In this case*

$$a_j^{U,Z} = 1 - \sum_{H \in \mathcal{H}_j^{U,Z}} 1 = 1 - d_{H_j^{U,Z}}(j),$$

where  $\mathcal{H}_j^{U,Z}$  is the subset of hyperedges of  $\mathcal{H}$  containing vertex  $j$ , but which do not contain any vertex in  $U \cup V$ . Recalling that the greedy algorithm for the hitting set problem selects at every iteration the vertex with largest degree of the remaining hyperedges, one trivially can see that both the greedy procedure and this ONE-PASS heuristic will return the same solution.

Two interesting facts arise from Example 6.1. First, it can be seen that ONE-PASS depends on the way the hitting set problem was formulated – since  $\epsilon_H$  ( $H \in \mathcal{H}$ ) can be any nonnegative number. Second, due to the equivalence shown before, Example 6.1 demonstrates that the value returned by ONE-PASS based on the best Euclidean linear approximation is a polynomial time  $(1 + \log(D_{\mathcal{H}}))$ -factor approximation algorithm ([173]) for the hitting set (and thus for the set covering) problem.

Formula (6.1) can be specialized for the quadratic case using (4.4) to get

$$a_j^{U,Z} = c_j + \frac{1}{2} \left( \sum_{k \in \mathbf{V} \setminus (U \cup Z): k < j} c_{kj} + \sum_{k \in \mathbf{V} \setminus (U \cup Z): k > j} c_{jk} \right) + \left( \sum_{k \in U: k < j} c_{kj} + \sum_{k \in U: k > j} c_{jk} \right), \quad (6.2)$$

for all  $j \in \mathbf{V} \setminus (U \cup Z)$ .

It is interesting to notice that the best linear coefficients (6.1) coincide in absolute value with the coefficients of the linear terms of the bi-form associated to the quadratic function (see Section 8.1). This situation implies that there is a close relationship between the DDT method (ii) introduced in the previous subsection and the one-pass approach based on best linear approximations. The big difference between these two methods is that DDT also considers equality relations between literals at each iteration of the method, and the one-pass method of this section does not.

From formula (6.2), it is simple to verify that

$$\begin{aligned} a_j^{U \cup \{i\}, Z} &= a_j^{U,Z} + \frac{1}{2} c_{ij}, & \text{if } j > i, \\ a_j^{U \cup \{i\}, Z} &= a_j^{U,Z} + \frac{1}{2} c_{ji}, & \text{if } j < i, \\ a_j^{U, Z \cup \{i\}} &= a_j^{U,Z} - \frac{1}{2} c_{ij}, & \text{if } j > i, \\ a_j^{U, Z \cup \{i\}} &= a_j^{U,Z} - \frac{1}{2} c_{ji}, & \text{if } j < i, \end{aligned} \quad (6.3)$$

for all  $j$  and  $i \in \mathbf{V} \setminus (U \cup Z)$ ,  $j \neq i$ .

**Theorem 6.1.** *Using best linear  $L_2$ -approximations, a heuristic solution to the minimization of a quadratic pseudo-Boolean function  $f$  is provided by ONE-PASS in  $O(n^2)$  time.*

*Proof.* To get the claimed complexity, the computations need to be organized carefully. First, we can build a variable-term data structure, and pre-compute the starting  $a_j^{\emptyset, \emptyset}$  values in  $O(\text{size}(f))$  time. After this, each of the  $n$  iterations can be executed in at most  $O(n)$  steps (to select the appropriate coefficient), and using (6.3) the evaluation of each  $a_j^{U, Z}$ ,  $j \in \mathbf{V} \setminus (U \cup Z)$  can be executed in time proportional to the number of occurrences of  $x_j$  in (1.5). Hence the total time of the algorithm after the pre-computations is  $O(n^2)$ , thus proving our claim.  $\square$

### Best homogeneous linear $L_2$ -approximation

When optimizing a pseudo-Boolean function given as multi-linear polynomial (1.1), the constant  $c_0$  value does not affect the optimal 0–1 vector solution. For the same reason, the constant part of the best linear approximation does not have an impact in its optimization. Therefore, we considered next the use of the best homogeneous linear approximation (see Theorem 4.4 in Section 4.8) in the framework of ONE-PASS.

In what follows we consider the application of this heuristic to quadratic pseudo-Boolean functions only, in spite of the fact that these ideas would also work in the general case. We recall Corollary 4.5 that basically states that the linear coefficients between the homogeneous and the non-homogeneous case differ by a constant  $Q$ . In the quadratic pseudo-Boolean case this constant is

$$Q = -\frac{2c_0}{n+1} + \frac{\sum_{1 \leq r < s \leq n} c_{rs}}{2(n+1)} \quad (6.4)$$

We shall disregard the constant part ( $c_0$ ) of the function at every iteration of ONE-PASS. In this case,  $Q$  is equal to the sum of all quadratic coefficients divided by  $2(n+1)$ .

The linear coefficients of the best homogeneous Euclidean linear approximation are therefore

$$a_j^{U,Z} = \Delta_j \left( \left( \frac{1}{2}, \dots, \frac{1}{2} \right) [U \cup Z \leftarrow \mathbf{y}^{U,Z}] \right) - Q^{U,Z},$$

for all  $j \in \mathbf{V} \setminus (U \cup Z)$ , where

$$Q^{U,Z} = \frac{\sum_{r,s \in \mathbf{V} \setminus (U \cup Z): r < s} c_{rs}}{2(|\mathbf{V} \setminus (U \cup Z)| + 1)}.$$

It is simple to verify that

$$\begin{aligned} a_j^{U \cup \{i\}, Z} &= a_j^{U,Z} + \frac{1}{2}c_{ij} + Q^{U,Z} - Q^{U \cup \{i\}, Z}, & \text{if } j > i, \\ a_j^{U \cup \{i\}, Z} &= a_j^{U,Z} + \frac{1}{2}c_{ji} + Q^{U,Z} - Q^{U \cup \{i\}, Z}, & \text{if } j < i, \\ a_j^{U, Z \cup \{i\}} &= a_j^{U,Z} - \frac{1}{2}c_{ij} + Q^{U,Z} - Q^{U, Z \cup \{i\}}, & \text{if } j > i, \\ a_j^{U, Z \cup \{i\}} &= a_j^{U,Z} - \frac{1}{2}c_{ji} + Q^{U,Z} - Q^{U, Z \cup \{i\}}, & \text{if } j < i, \end{aligned} \tag{6.5}$$

for all  $j$  and  $i \in \mathbf{V} \setminus (U \cup Z)$ ,  $j \neq i$ , and where

$$\begin{aligned} Q^{U,Z} - Q^{U \cup \{i\}, Z} &= Q^{U,Z} - Q^{U, Z \cup \{i\}} \\ &= \frac{\sum_{r,s \in \mathbf{V} \setminus (U \cup Z): r < s} c_{rs}}{2(|\mathbf{V} \setminus (U \cup Z)| + 1)} - \frac{\sum_{r,s \in \mathbf{V} \setminus (U \cup Z \cup \{i\}): r < s} c_{rs}}{2|\mathbf{V} \setminus (U \cup Z)|} \\ &= \frac{-2Q^{U,Z} + \left( \sum_{k \in \mathbf{V} \setminus (U \cup Z): k < i} c_{ki} + \sum_{k \in \mathbf{V} \setminus (U \cup Z): k > i} c_{ik} \right)}{2|\mathbf{V} \setminus (U \cup Z)|} \end{aligned}$$

**Theorem 6.2.** *Using best homogeneous linear  $L_2$ -approximations, a heuristic solution to the minimization of a quadratic pseudo-Boolean function  $f$  is provided by ONE-PASS in  $O(n^2)$  time.*

*Proof.* Clearly,  $Q^{\emptyset, \emptyset}$  and the initial  $a_j^{\emptyset, \emptyset}$  values can be computed in  $O(\text{size}(f))$  time. If a variable-term data structure is used to handle the coefficients in (1.5), then the intermediate  $a_j^{U,Z}$  and  $Q^{U,Z}$  values can be computed in at most  $2n$  steps from (6.5). Since a single variable is fixed at each iteration, then the algorithm has  $n$  iterations which with the pre-computations will take a total time of  $O(n^2)$ , thus proving our claim.  $\square$

### 6.1.2.2 Probabilistic methods

Let  $f$  be a quadratic pseudo-Boolean function represented by a multi-linear polynomial (1.5). For every index  $j \in \mathbf{V}$ , let us associate to the partial derivative  $\Delta_j$  of  $f$ , a stochastic function

$$\zeta_j(\xi_1, \dots, \xi_n) \stackrel{\text{def}}{=} c_j + \sum_{k=1}^{j-1} c_{kj} \xi_k + \sum_{k=j+1}^n c_{jk} \xi_k,$$

where  $\xi_k, k \in \mathbf{V}$  are random variables.  $\zeta_j$  can be seen as a random variable that simulates the distribution of the values associated with the  $j$ th partial derivative of  $f$ . Consider now some results that will be helpful in justifying the proposed heuristic methods of this subsection.

**Lemma 6.1.**

$$\text{Exp}[\zeta_j(\xi_1, \dots, \xi_n)] = \Delta_j(\text{Exp}[\xi_1], \dots, \text{Exp}[\xi_n]), \quad j \in \mathbf{V}.$$

*Proof.* Using (4.4), the equation stated above follows immediately from the additivity of the expected value.  $\square$

**Lemma 6.2.** *If  $\xi_1, \dots, \xi_n$  are independent random variables then*

$$\text{Var}[\zeta_j(\xi_1, \dots, \xi_n)] = \sum_{k=1}^{j-1} c_{kj}^2 \text{Var}[\xi_k] + \sum_{k=j+1}^n c_{jk}^2 \text{Var}[\xi_k], \quad j \in \mathbf{V}.$$

*Proof.* The statement follow by using (4.4) and the independence property combined with the fact that if  $\alpha \in \mathbb{R}$ , then  $\text{Var}[\alpha \xi_k] = \alpha^2 \text{Var}[\xi_k]$ .  $\square$

**Lemma 6.3.** *If  $\xi_1, \dots, \xi_n$  is a sequence of independent random variables defined on the same probability space, having finite expected values  $\mu_1, \dots, \mu_n$ , variances  $\sigma_1^2, \dots, \sigma_n^2$  and third central moments  $r_j^3 = \text{Exp}[|\xi_j - \mu_j|^3]$ ,  $j \in \mathbf{V}$ , and if*

$$\lim_{n \rightarrow \infty} \frac{r_n}{\sqrt{\sum_{j=1}^n \sigma_j^2}} = 0,$$

then

$$N_j \equiv \frac{\zeta_j(\xi_1, \dots, \xi_n) - \Delta_j(\mu_1, \dots, \mu_n)}{\sqrt{\sum_{k=1}^{j-1} c_{kj}^2 \sigma_k^2 + \sum_{k=j+1}^n c_{jk}^2 \sigma_k^2}}, \quad j \in \mathbf{V},$$

converges to the standard normal distribution.

*Proof.* This is a restatement of a generalization of the central limit theorem under the Lyapunov conditions.  $\square$

**Remark 6.1.** *The practical implication of Lemma 6.3 is that a large linear combination of independent random variables has values normally distributed. Therefore, if a traditional (e.g., uniform, Bernoulli, etc.) probability distribution is considered for  $\xi_k$ ,  $k \in \mathbf{V}$ , then*

$$\text{Normal} \left( \Delta_j(\mu_1, \dots, \mu_n), \sum_{k=1}^{j-1} c_{kj}^2 \sigma_j^2 + \sum_{k=j+1}^n c_{jk}^2 \sigma_j^2 \right) \quad (6.6)$$

is a good approximation of  $\zeta_j(\xi_1, \dots, \xi_n)$ ,  $j \in \mathbf{V}$ .

According to the necessary conditions of optimality stated in Proposition 4.1, every minimizing point  $(x_1^*, \dots, x_n^*)$  of  $f$  must satisfy the following two conditions for every index  $j \in \mathbf{V}$ :

1. If  $x_j^* = 1$  then  $\Delta_j(\mathbf{x}^*) \leq 0$ ;
2. If  $x_j^* = 0$  then  $\Delta_j(\mathbf{x}^*) \geq 0$ .

Proposition 4.1 suggests that a potentially good strategy for selecting the next variable to be fixed by subroutine  $\mathbb{X}$  in ONE-PASS, is to give priority to those variables whose partial derivatives have *constant signs with large probability*.

This selection process and subsequent fixation at every iteration of ONE-PASS create a new function, where the new derivative functions of the remaining variables (depending on the selected variable) also change.

The ONE-PASS procedure creates a sequence of actions that depend on the decisions made previously, each one being based on how the values of the resulting derivative functions are distributed. A key issue is therefore to know what probability distribution to assume for the partial derivative values. We consider two basic approaches:

- **Uniform** – The range of values of the first derivative function  $\Delta_j$  ( $j \in \mathbf{V}$ ) is uniformly distributed between its minimum and maximum values (i.e.,  $\text{Uniform}(L_j, U_j)$ );
- **Normal** – The range of values of the first derivative function  $\Delta_j$  ( $j \in \mathbf{V}$ ) is normally distributed, as defined in (6.6).

Let us introduce further notation to indicate the fact that some of the variables have already been assigned a 0–1 value. The function  $f$  under the partial assignment  $\mathbf{y}^{U,Z}$  will be denoted as  $f^{\mathbf{U},\mathbf{Z}}$ , and similarly the first derivatives, the associated stochastic functions, and the corresponding minimum and maximum values will be denoted respectively as  $\Delta_j^{\mathbf{U},\mathbf{Z}}$ ,  $\zeta_j^{\mathbf{U},\mathbf{Z}}$ ,  $U_j^{\mathbf{U},\mathbf{Z}}$  and  $L_j^{\mathbf{U},\mathbf{Z}}$ , for all  $j \in \mathbf{V} \setminus (U \cup Z)$ .

Let us denote the subset of (random) variables for which the probabilities of the corresponding partial derivatives to have constant signs are highest by

$$S^{U,Z} \stackrel{\text{def}}{=} \arg \max_{j \in \mathbf{V} \setminus (U \cup Z)} \left( \text{Prob} \left[ \zeta_j^{U,Z} \geq 0 \right], \text{Prob} \left[ \zeta_j^{U,Z} < 0 \right] \right).$$

The ONE-PASS( $\mathbb{X}, \mathbb{Y}$ ) sub-family of heuristic methods that we shall study in this subsection is characterized as follows:

$\begin{aligned} \mathbb{X} &\equiv i = \min \left\{ k \in S^{U,Z} \mid \left  \text{Exp} \left[ \zeta_k^{U,Z} \right] \right  = \max_{j \in S^{U,Z}} \left( \left  \text{Exp} \left[ \zeta_j^{U,Z} \right] \right  \right) \right\}, \\ \mathbb{Y} &\equiv \text{if } \left( \text{Exp} \left[ \zeta_i^{U,Z} \right] \leq 0 \right) \text{ then } U \leftarrow U \cup \{i\} \text{ else } Z \leftarrow Z \cup \{i\}. \end{aligned}$
--

This algorithm depends on the way in which the probability distributions of the first derivative values are defined. In our computational experiments we only considered cases which use the same probability distribution for all non-fixed variables (i.e. for  $x_j$ ,  $j \in \mathbf{V} \setminus (U \cup Z)$ ). In addition, we assumed that the expected value of each non-fixed variable is 0.5. These choices were made with the intuitive idea of giving equal chances to every non-fixed variable to be selected.

The stochastic variants of the partial derivative functions that we have tested are described in Table 6.1. For simplicity, this table only considers the cases analyzed in the first iteration of ONE-PASS (i.e. if  $U = Z = \emptyset$ ). It should be noted here that a partial

Table 6.1: Probability distributions of random variables used to characterize the probability distribution of partial derivatives of quadratic pseudo-Boolean functions.

$\zeta_j, j \in \mathbf{V}$	$\xi_j, j \in \mathbf{V}$
discrete Uniform ( $L_j, U_j$ )	–
Normal $\left( \frac{L_j+U_j}{2}, \frac{1}{4} \left( \sum_{k=1}^{j-1} c_{kj}^2 + \sum_{k=j+1}^n c_{jk}^2 \right) \right)$	Bernoulli $\left( \frac{1}{2} \right)$
Normal $\left( \frac{L_j+U_j}{2}, \frac{1}{12} \left( \sum_{k=1}^{j-1} c_{kj}^2 + \sum_{k=j+1}^n c_{jk}^2 \right) \right)$	Uniform (0, 1)
Normal $\left( \frac{L_j+U_j}{2}, \frac{1}{48} \left( \sum_{k=1}^{j-1} c_{kj}^2 + \sum_{k=j+1}^n c_{jk}^2 \right) \right)$	Uniform (0.25, 0.75)
Normal $\left( \frac{L_j+U_j}{2}, \frac{1}{300} \left( \sum_{k=1}^{j-1} c_{kj}^2 + \sum_{k=j+1}^n c_{jk}^2 \right) \right)$	Uniform (0.40, 0.60)
Normal $\left( \frac{L_j+U_j}{2}, \frac{1}{1200} \left( \sum_{k=1}^{j-1} c_{kj}^2 + \sum_{k=j+1}^n c_{jk}^2 \right) \right)$	Uniform (0.45, 0.55)
Normal $\left( \frac{L_j+U_j}{2}, \frac{1}{30000} \left( \sum_{k=1}^{j-1} c_{kj}^2 + \sum_{k=j+1}^n c_{jk}^2 \right) \right)$	Uniform (0.49, 0.51)

assignment essentially results in a new quadratic pseudo-Boolean function to which the options listed in this table are applied again. We also remark that variables fixed during the process can be seen as constants, i.e. random variables with no variance.

The first option in Table 6.1 considers that the values of the partial derivatives are integers between their minimum and maximum values. This choice was motivated by the fact that the test problems have integer coefficients. The second and following variants consider the variate (6.6), and assume that all variables have an independent and identical distribution. All options in Table 6.1 have the same expectation. In these cases, the expected value of the  $j$ -th partial derivative in a given iteration of ONE-PASS is

$$\text{Exp} \left[ \zeta_j^{U,Z} \right] = \frac{L_j^{U,Z} + U_j^{U,Z}}{2} = \Delta_j \left( \left( \frac{1}{2}, \dots, \frac{1}{2} \right) [U \cup Z \leftarrow \mathbf{y}^{U,Z}] \right),$$

where

$$\begin{aligned}
L_j^{U,Z} &= \min_{\mathbf{x} \in \mathbf{V} \setminus (U \cup Z)} \Delta_j(\mathbf{x} [U \cup Z \leftarrow \mathbf{y}^{U,Z}]) \\
&= L_j - \sum_{\substack{k \in Z: k < j \\ c_{kj} < 0}} c_{kj} - \sum_{\substack{k \in Z: k > j \\ c_{jk} < 0}} c_{jk} + \sum_{\substack{k \in U: k < j \\ c_{kj} > 0}} c_{kj} + \sum_{\substack{k \in U: k > j \\ c_{jk} > 0}} c_{jk} \\
&= c_j + \sum_{\substack{k \in \mathbf{V} \setminus (U \cup Z): k < j \\ c_{kj} < 0}} c_{kj} + \sum_{\substack{k \in \mathbf{V} \setminus (U \cup Z): k > j \\ c_{jk} < 0}} c_{jk} + \sum_{k \in U: k < j} c_{kj} + \sum_{k \in U: k > j} c_{jk} \text{ and} \\
U_j^{U,Z} &= \max_{\mathbf{x} \in \mathbf{V} \setminus (U \cup Z)} \Delta_j(\mathbf{x} [U \cup Z \leftarrow \mathbf{y}^{U,Z}]) \\
&= U_j - \sum_{\substack{k \in Z: k < j \\ c_{kj} > 0}} c_{kj} - \sum_{\substack{k \in Z: k > j \\ c_{jk} > 0}} c_{jk} + \sum_{\substack{k \in U: k < j \\ c_{kj} < 0}} c_{kj} + \sum_{\substack{k \in U: k > j \\ c_{jk} < 0}} c_{jk} \\
&= c_j + \sum_{\substack{k \in \mathbf{V} \setminus (U \cup Z): k < j \\ c_{kj} > 0}} c_{kj} + \sum_{\substack{k \in \mathbf{V} \setminus (U \cup Z): k > j \\ c_{jk} > 0}} c_{jk} + \sum_{k \in U: k < j} c_{kj} + \sum_{k \in U: k > j} c_{jk},
\end{aligned}$$

for all  $j \in \mathbf{V} \setminus (U \cup Z)$ .

The expected values  $\text{Exp} [\zeta_j^{U,Z}]$ ,  $j \in \mathbf{V} \setminus (U \cup Z)$ , are coincident with the coefficients (6.1) of the best linear approximation of the function, which is defined by a partial assignment corresponding to the subset of variables  $(U \cup Z)$  with value fixed. As a consequence of this relation,  $\text{Exp} [\zeta_j^{U,Z}]$  can be computed efficiently between two consecutive iterations of ONE-PASS, as it was shown in (6.3) for the ONE-PASS case based on the best linear approximation.

In the normal distribution cases, it can be seen that the impact of the distribution assumed for the set of non-fixed variables is basically in the ‘‘multiplier’’ term associated with the variance component. Therefore, the key player to be analyzed in the computational analysis is in fact which multiplier is more suitable to obtain better quality solutions for QUBO.

For the options listed in Table 6.1, the variance of the partial derivatives  $j$ ,  $j \in \mathbf{V} \setminus (U \cup Z)$  in a given iteration of ONE-PASS is

$$\text{Var} [\zeta_j^{U,Z}] = \alpha \left( \sum_{k \in \mathbf{V} \setminus (U \cup Z): k < j} c_{kj}^2 + \sum_{k \in \mathbf{V} \setminus (U \cup Z): k > j} c_{jk}^2 \right), \quad (6.7)$$

where  $\alpha$  is the variance of the individual variables. In the cases listed in Table 6.1 the

variances of the variables were chosen to be all equal.

When using

$$\begin{aligned}
\text{Var} \left[ \zeta_j^{U \cup \{i\}, Z} \right] &= \text{Var} \left[ \zeta_j^{U, Z} \right] - \alpha c_{ij}^2, & \text{if } j > i, \\
\text{Var} \left[ \zeta_j^{U \cup \{i\}, Z} \right] &= \text{Var} \left[ \zeta_j^{U, Z} \right] - \alpha c_{ji}^2, & \text{if } j < i, \\
\text{Var} \left[ \zeta_j^{U, Z \cup \{i\}} \right] &= \text{Var} \left[ \zeta_j^{U, Z} \right] - \alpha c_{ij}^2, & \text{if } j > i, \\
\text{Var} \left[ \zeta_j^{U, Z \cup \{i\}} \right] &= \text{Var} \left[ \zeta_j^{U, Z} \right] - \alpha c_{ji}^2, & \text{if } j < i,
\end{aligned} \tag{6.8}$$

for all  $j$  and  $i \in \mathbf{V} \setminus (U \cup Z)$ ,  $j \neq i$ , then the variance of the partial derivatives (6.7) can be computed efficiently between two consecutive iterations of ONE-PASS. These values also show that the variance of the partial derivatives are non-increasing functions at every iteration of One-Pass, which at the end of the procedure will have value zero.

**Theorem 6.3.** *Using the probability distributions of partial derivatives, a heuristic solution to the minimization of a quadratic pseudo-Boolean function  $f$  is provided by ONE-PASS in  $O(n^2)$  time.*

*Proof.* Using the relations (6.8) and using the fact mentioned above about calculating the expected values of the partial derivatives between two consecutive iterations, we can use the same arguments used in Theorem 6.1 to prove our claim.  $\square$

### 6.1.2.3 Rounding methods

The minimum of a polynomial function  $f$  in 0–1 variables coincides with the minimum of the same polynomial defined when the variables are continuous, taking values in the interval  $[0, 1]$  (see Section 4.7). Furthermore, the value of  $f$  at any point  $\mathbf{p} \in \mathbb{U}^n$  can be seen as being the expected value of the pseudo-Boolean function  $f$ , whose 0–1 variables have independent probabilities given as  $\text{Prob}[x_i = 1] = p_i$  for all  $i = 1, \dots, n$  (see Proposition 4.6), i.e.  $f(\mathbf{p}) = \text{Exp}[f(\mathbf{x})]$ .

The heuristic method presented in this subsection finds a finite sequence of points  $\mathbf{p}^{(0)}, \dots, \mathbf{p}^{(n)}$  belonging to  $\mathbb{U}^n$  such that the following four conditions are met:

i) The expected value of  $f$  does not increase along the sequence, i.e.

$$f(\mathbf{p}^{(k)}) \geq f(\mathbf{p}^{(k+1)}) \text{ for all } k = 0, \dots, n-1;$$

ii) Two consecutive points differ in at most a single coordinate, i.e.

$$p_i^{(k)} = p_i^{(k+1)} \text{ for all } k = 0, \dots, n-1, \text{ and } i \in \mathbf{V} \setminus \{i_{k+1}\};$$

iii) The set of indices  $\{i_k | k = 1, \dots, n\}$  selected along the sequence, defines the complete set of variables, i.e.

$$\mathbf{V} = \{i_k | k = 1, \dots, n\};$$

iv) The last point of the sequence is a binary vector, i.e.  $\mathbf{p}^{(n)} \in \mathbb{B}^n$ .

These conditions together imply that the value of each variable is rounded (if fractional) or switched (if binary) to a 0–1 value, thus justifying the name given to the “rounding” heuristics considered in this section.

The value decrease of  $f$  between two consecutive points of the sequence is

$$\begin{aligned} & f(\mathbf{p}^{(k)}) - f(\mathbf{p}^{(k+1)}) \\ &= p_{i_{k+1}}^{(k)} \Delta_{i_{k+1}}(\mathbf{p}^{(k)}) + \Theta_{i_{k+1}}(\mathbf{p}^{(k)}) - p_{i_{k+1}}^{(k+1)} \Delta_{i_{k+1}}(\mathbf{p}^{(k+1)}) - \Theta_{i_{k+1}}(\mathbf{p}^{(k+1)}) \quad (6.9) \\ &= \left( p_{i_{k+1}}^{(k)} - p_{i_{k+1}}^{(k+1)} \right) \Delta_{i_{k+1}}(\mathbf{p}^{(k)}), \end{aligned}$$

for all  $k = 0, \dots, n-1$ . Thus, from our choices (*i-iv*) given above we get

$$\begin{aligned} \Delta_{i_{k+1}}(\mathbf{p}^{(k)}) > 0 & \Rightarrow p_{i_{k+1}}^{(k+1)} = 0 \text{ and} \\ \Delta_{i_{k+1}}(\mathbf{p}^{(k)}) < 0 & \Rightarrow p_{i_{k+1}}^{(k+1)} = 1. \end{aligned}$$

The heuristic value returned by these rounding procedures is therefore

$$f(\mathbf{p}^{(n)}) = f(\mathbf{p}^{(0)}) - \sum_{k=0}^{n-1} \left( \left( p_{i_{k+1}}^{(k)} - p_{i_{k+1}}^{(k+1)} \right) \Delta_{i_{k+1}}(\mathbf{p}^{(k)}) \right).$$

Because both the speed and the quality of solutions are important factors to consider in the design of algorithms for one-pass heuristics, we shall adopt a *greedy* variant of each step of the method. Therefore:

1. The initial value of  $f(\mathbf{p}^{(0)})$  should be chosen as small as possible;
2. The “transition index”  $i_k$  defining point  $\mathbf{p}^{(k)}$  from point  $\mathbf{p}^{(k-1)}$  should be selected so as to minimize the value of  $f(p^{(k)})$ .

In order to make the choice of the initial point both effective and efficient, we have studied several alternative ways of computing a low valued point in  $O(\text{size}(f))$  time. The various starting points considered in this study are listed in Table 6.2.

The description of the starting points considered in this study is listed in Table 6.2.

The selection method  $I_1$  is self-explanatory. Methods  $I_2$  and  $I_3$  are two slightly different attempts to decrease  $\text{Exp}[f]$  (see Proposition 4.6 and Corollary 4.4). Finally, methods  $I_4$  and  $I_5$  try to estimate the probability of each variable  $x_i$  to take the value 0 at a local minimum. They both use the characterization of local minima (see Proposition 4.1).  $I_4$  uses the available bounds for the derivatives, and assumes for simplicity that the (integer) values of the derivatives are uniformly distributed between their upper and lower bounds.  $I_5$  assumes that the values of the derivatives are normally distributed, having means and standard deviations defined as in Lemma 6.3.

We have successfully experimented with a more sophisticated type procedure for identifying the starting point  $\mathbf{p}^{(0)}$ . The procedure consists in choosing one coordinate at a time, identifying an “ideal” value of it, substituting this value into the function, repeating the above steps on the above function, etc., until  $\mathbf{p}^{(0)}$  is found.

Using the previous idea, we have considered four additional starting points based on the same basic constructs used by methods  $I_s$ ,  $s = 2, \dots, 5$ . The idea is to reapply the principle associated to starting point  $I_s$ ,  $s = 2, \dots, 5$ , to every component  $j = 1, \dots, n$ . For instance, if the optimal choice of method  $I_3$  is used for the first component, then  $x_1 = \alpha$  (see definition of  $\alpha$  in Table 6.2),  $x_2$  would get the optimal choice  $\alpha_2$  applied to the “new” function  $f(\mathbf{x}[\{1\} \leftarrow (\alpha)])$ , and in general  $x_j$ ,  $j = 3, \dots, n$ , would get

Table 6.2: Starting points considered in the computational experiments.

**I<sub>1</sub>** (Center):  $\mathbf{p}^{(0)} = (\frac{1}{2}, \dots, \frac{1}{2})$

**I<sub>2</sub>** (Pos/Neg):  $\mathbf{p}^{(0)} = (1 - \rho, \dots, 1 - \rho)$ , where  $\rho$  is the proportion of terms having positive coefficients, i.e.

$$\rho = \frac{\sum_{i \in \mathbf{V}: c_i > 0} c_i + \sum_{1 \leq i < j \leq n: c_{ij} > 0} c_{ij}}{\sum_{i=1}^n |c_i| + \sum_{1 \leq i < j \leq n} |c_{ij}|}.$$

**I<sub>3</sub>** (Best):  $\mathbf{p}^{(0)} = (\alpha, \dots, \alpha)$  assumes that all variables take the same value, and  $\alpha$  is the optimal choice for that value, i.e.

$$\alpha = \arg \min_{0 \leq \lambda \leq 1} \lambda \left( \sum_{i=1}^n c_i + \lambda \sum_{1 \leq i < j \leq n} c_{ij} \right).$$

**I<sub>4</sub>** (Delta):  $\mathbf{p}^{(0)} = (\gamma_1, \dots, \gamma_n)$ , where for  $i = 1, \dots, n$  we have

$$\gamma_i = \begin{cases} 0, & L_i \geq 0, \\ \frac{0.5 - L_i}{1 + U_i - L_i}, & L_i < 0 \text{ and } U_i > 0, \\ 1, & \text{otherwise.} \end{cases}$$

**I<sub>5</sub>** (Normal):  $\mathbf{p}^{(0)} = (\nu_1, \dots, \nu_n)$ , where for  $i = 1, \dots, n$  we have

$$\nu_i = \begin{cases} 1, & L_i \geq 0, \\ 1 - \Phi \left( \frac{L_j + U_j}{\sqrt{(\sum_{k=1}^{j-1} c_{kj}^2 + \sum_{k=j+1}^n c_{jk}^2)}} \right), & L_i < 0 \text{ and } U_i > 0, \\ 0, & \text{otherwise.} \end{cases}$$

the optimal choice  $\alpha_j$  applied to the function  $f(\mathbf{x}[\{1, \dots, j-1\} \leftarrow (\alpha, \alpha_2, \dots, \alpha_{j-1})])$ . This method results in the starting point  $(\alpha, \alpha_2, \dots, \alpha_n)$ .

These improved starting point methods will be denoted  $I_s^*$ ,  $s = 2, \dots, 5$ ; each of them is based on the underlying ideas associated to the corresponding methods  $I_s$ ,  $s = 2, \dots, 5$ , described in Table 6.2, enhanced by their iterative application illustrated above.

It should be remarked that all methods  $I_s^*$ ,  $s = 2, \dots, 5$ , run also in  $O(\text{size}(f))$ , but do obviously require somewhat larger computing times than the corresponding methods  $I_s$ ,  $s = 2, \dots, 5$ .

In order to decrease the value of the function  $f$  by changing the value of a variable, the selection of that variable has to balance computational time and loss in function value. To be able to describe the proposed method, we need a few more notations.

Given a quadratic pseudo-Boolean function  $f$  and a vector  $\mathbf{p}^{(k)} \in \mathbb{U}^n$ , let us introduce for every  $j \in \mathbf{V} \setminus (U \cup Z)$  the quantities

$$d_j = d_j(\mathbf{p}^{(k)}) = f(\mathbf{p}^{(k)}) - f(\mathbf{p}^{(k+1)}), \quad (6.10)$$

which measure the size of local improvement when changing only the component  $j$  (optimally).

**Lemma 6.4.** *For any vector  $\mathbf{p}^{(k)} \in \mathbb{U}^n$  and index  $j \in \mathbf{V} \setminus (U \cup Z)$  we have*

$$d_j(\mathbf{p}^{(k)}) = \max \left\{ p_j^{(k)} \Delta_j(\mathbf{p}^{(k)}), (p_j^{(k)} - 1) \Delta_j(\mathbf{p}^{(k)}) \right\}.$$

*If the values of the derivatives  $\Delta_j(\mathbf{p}^{(k)})$  ( $j \in \mathbf{V} \setminus (U \cup Z)$ ), are available, then the quantities  $d_j$  ( $j \in \mathbf{V} \setminus (U \cup Z)$ ), can be computed in  $O(n)$  extra time.*

*Proof.* Immediate by (6.9) and the definitions. □

A ONE-PASS( $\mathbb{X}, \mathbb{Y}$ ) family of rounding heuristics that we shall study is characterized as follows:

$$\begin{aligned}
\mathbb{X} &\equiv i = i_k = \min \left\{ r \mid d_r = \max_{j \in \mathbf{V} \setminus (U \cup Z)} (d_j) \right\}, \\
\mathbb{Y} &\equiv \text{if } \left( \Delta_i^{U,Z} \leq 0 \right) \text{ then } U \leftarrow U \cup \{i\} \text{ else } Z \leftarrow Z \cup \{i\}.
\end{aligned} \tag{6.11}$$

**Theorem 6.4.** *Using rounding procedures, a heuristic solution to the minimization of a quadratic pseudo-Boolean function  $f$  is provided by ONE-PASS in  $O(n^2)$  time.*

*Proof.* The value of the partial derivatives  $\Delta_j$ ,  $j \in \mathbf{V} \setminus (U \cup Z)$ , is kept updated in the point  $\mathbf{p}^{(k)}$ ,  $k = 1, \dots, n$ , associated to each of the  $n$  rounding steps. The initial calculation of these values can be carried out in  $O(\text{size}(f))$  time, and the subsequent updates can be done in  $O(n)$  time, by using the relations

$$\begin{aligned}
\Delta_j^{U \cup \{i\}, Z} &= \Delta_j^{U, Z} + \left(1 - p_i^{(k-1)}\right) c_{ij}, & \text{if } j > i, \\
\Delta_j^{U \cup \{i\}, Z} &= \Delta_j^{U, Z} + \left(1 - p_i^{(k-1)}\right) c_{ji}, & \text{if } j < i, \\
\Delta_j^{U, Z \cup \{i\}} &= \Delta_j^{U, Z} - p_i^{(k-1)} c_{ij}, & \text{if } j > i, \\
\Delta_j^{U, Z \cup \{i\}} &= \Delta_j^{U, Z} - p_i^{(k-1)} c_{ji}, & \text{if } j < i,
\end{aligned}$$

for all  $j \in \mathbf{V} \setminus (U \cup Z)$ . With the  $\Delta_j$  ( $j \in \mathbf{V} \setminus (U \cup Z)$ ) values available, the component selection step given in Lemma 6.4 can also be done in  $o(n)$  time. Adding these times for  $n$  rounding steps, the claimed time complexity of  $O(n^2)$  follows readily.  $\square$

**Theorem 6.5.** *One-pass rounding heuristic using the center as starting point produces a solution for QUBO equivalent to the solution returned by the best linear approximation one-pass heuristic. (see Section 6.1.2.1)*

*Proof.* This result follows trivially since the coefficients of the best linear  $l_2$ -approximation of a pseudo-Boolean function  $f$  are the values of the partial derivatives of  $f$  in the center (see Proposition 4.8 and (6.1)).  $\square$

At every iteration of the ONE-PASS method, several variables  $x_j$  not yet fixed (i.e.  $j \in \mathbf{V} \setminus \{U \cup Z\}$ ) may have a partial derivative  $\Delta_j^{U, Z}$  with constant sign regardless of the values of the other variables  $x_i$  not yet fixed (i.e.  $i \neq j, i \in \mathbf{V} \setminus \{U \cup Z\}$ ) by the procedure.

The variables in those circumstances will be rounded to the corresponding value determined by the local minimum conditions (i.e., if  $\Delta_j^{U,Z} > 0$  then  $x_j = 0$ , otherwise  $x_j = 1$ ). The point that can make a difference in the one-pass rounding algorithms is to when an immediate rounding (or switching) operation is applied or not.

In the study of the one-pass rounding heuristics we consider a version of ONE-PASS( $\mathbb{X}, \mathbb{Y}$ ) that immediately rounds all variables (not yet fixed) which satisfy the local optimality conditions. Before presenting this variant, let us introduce  $L_j^{U,Z}$  (respectively  $U_j^{U,Z}$ ) as being the minimum (respectively maximum) of the pseudo-Boolean linear function  $\Delta_j^{U,Z}$ . Let

$$S_+^{U,Z} = \left\{ j \in \mathbf{V} \setminus \{U \cup Z\} \mid L_j^{U,Z} U_j^{U,Z} \geq 0 \right\}.$$

be the set containing the variables (not yet fixed) which satisfy the local optimality conditions.

An additional family of one-pass rounding heuristics that we consider is characterized as follows:

$\begin{aligned} \mathbb{X} &\equiv \begin{cases} i = i_k = \min S_+^{U,Z}, & S_+^{U,Z} \neq \emptyset \\ \mathbb{X} \text{ from (6.11)}, & S_+^{U,Z} = \emptyset, \end{cases} \\ \mathbb{Y} &\equiv \mathbb{Y} \text{ from (6.11)}. \end{aligned}$
---

**Theorem 6.6.** *Using rounding procedures that enforce local optimality as soon as possible, a heuristic solution to the minimization of a quadratic pseudo-Boolean function  $f$  is provided by ONE-PASS in  $O(n^2)$  time.*

*Proof.* The claimed complexity follows from the fact that the minimum  $L_j^{U,Z}$  and the maximum  $U_j^{U,Z}$  of the partial derivative  $\Delta_j^{U,Z}$  can be obtained directly from separate data structures which contain the values updated according to the sets  $U$  and  $Z$ . The

initial calculation of these values can be carried out in  $O(\text{size}(f))$  time, and the subsequent updates can be done in  $O(n)$  time, by using the relations

$$\begin{aligned}
U_j^{U \cup \{i\}, Z} &= U_j^{U, Z} + \min(0, c_{ij}), & \text{if } j > i, \\
U_j^{U \cup \{i\}, Z} &= U_j^{U, Z} + \min(0, c_{ji}), & \text{if } j < i, \\
U_j^{U, Z \cup \{i\}} &= U_j^{U, Z} - \max(0, c_{ij}), & \text{if } j > i, \\
U_j^{U, Z \cup \{i\}} &= U_j^{U, Z} - \max(0, c_{ji}), & \text{if } j < i, \\
L_j^{U \cup \{i\}, Z} &= L_j^{U, Z} + \max(0, c_{ij}), & \text{if } j > i, \\
L_j^{U \cup \{i\}, Z} &= L_j^{U, Z} + \max(0, c_{ji}), & \text{if } j < i, \\
L_j^{U, Z \cup \{i\}} &= L_j^{U, Z} - \min(0, c_{ij}), & \text{if } j > i, \\
L_j^{U, Z \cup \{i\}} &= L_j^{U, Z} - \min(0, c_{ji}), & \text{if } j < i,
\end{aligned}$$

for all  $j \in \mathbf{V} \setminus (U \cup Z)$ . With the previous relations, it is simple to see that the same arguments used in the proof of Theorem 6.4 imply the claimed time complexity.  $\square$

We end this section by presenting an example that illustrates that more complex expressions can be used in one-pass procedures for QUBO.

**Example 6.2.** *Let us consider an arbitrary quadratic pseudo-Boolean function  $f$  given as (1.5). Let  $\mathbf{p} \in \mathbb{U}^n$  be an arbitrary real vector, and assume that the variables  $x_i$ ,  $i = 1, \dots, n$  are pairwise independent random variables for which  $p_i = \text{Prob}[x_i = 1] = 1 - \text{Prob}[x_i = 0]$  for  $i = 1, \dots, n$ . We start by finding the expected value decrease of  $f$  when the quadratic relation  $x_1 x_2 = 0$  is assumed:*

$$\begin{aligned}
& \text{Exp}[f(\mathbf{x})] - \text{Exp}[f(\mathbf{x}) | x_1 x_2 = 0] \\
&= \text{Exp}[f(\mathbf{x})] - \frac{\text{Exp}[f(\mathbf{x})] - \text{Prob}[x_1 x_2 = 1] \text{Exp}[f(\mathbf{x}) | x_1 x_2 = 1]}{1 - \text{Prob}[x_1 x_2 = 1]} \\
&= f(p_1, \dots, p_n) - \frac{f(p_1, \dots, p_n) - p_1 p_2 f(1, 1, p_3, \dots, p_n)}{1 - p_1 p_2} \\
&= \frac{p_1 p_2}{1 - p_1 p_2} (f(1, 1, p_3, \dots, p_n) - f(p_1, \dots, p_n)) \\
&= \frac{p_1 p_2}{1 - p_1 p_2} ((1 - p_1) \Delta_1(p_1, \dots, p_n) + (1 - p_2) \Delta_2(p_1, \dots, p_n) + (1 - p_1 - p_2 + p_1 p_2) c_{12}) \\
&= \frac{p_1 p_2}{1 - p_1 p_2} (\bar{p}_1 \Delta_1(p_1, \dots, p_n) + \bar{p}_2 \Delta_2(p_1, \dots, p_n) + \bar{p}_1 \bar{p}_2 c_{12}).
\end{aligned} \tag{6.12}$$

In particular if  $p_i = \frac{1}{2}$ ,  $i = 1, \dots, n$ , then (6.12) becomes

$$\begin{aligned} & \text{Exp}[f(\mathbf{x})] - \text{Exp}[f(\mathbf{x}) | x_1 x_2 = 0] \\ &= \frac{1}{3} \left( \frac{1}{2} \Delta_1 \left( \frac{1}{2}, \dots, \frac{1}{2} \right) + \frac{1}{2} \Delta_2 \left( \frac{1}{2}, \dots, \frac{1}{2} \right) + \frac{1}{4} c_{12} \right) \\ &= \frac{L_1 + U_1 + L_2 + U_2 + c_{12}}{12}. \end{aligned}$$

In the general case of assigning a quadratic term  $x_1^{(\alpha)} x_2^{(\beta)} = 0$  under the previous conditions, the expected value decrease of this assignment is given by the following formula:

$$\begin{aligned} & \text{Exp}[f(\mathbf{x})] - \text{Exp} \left[ f(\mathbf{x}) \mid x_1^{(\alpha)} x_2^{(\beta)} = 0 \right] \\ &= \frac{p_1 p_2}{1 - p_1 p_2} \left( (\alpha - p_1) \Delta_1(p_1, \dots, p_n) + (\beta - p_2) \Delta_2(p_1, \dots, p_n) + (\alpha\beta - \beta p_1 - \alpha p_2 + p_1 p_2) c_{12} \right). \end{aligned}$$

A possible good strategy for selecting a term in the DDT *devour* stage (presented in Section 6.1.1) is to consider either a quadratic term or a linear term, which provides the expected largest decrease in function value, as was illustrated in the previous example for the quadratic term and by method  $\mathbb{X}$  of (6.11) for the linear case.

### 6.1.3 Measuring heuristics performance

Let  $f$  be a pseudo-Boolean function given as a multilinear polynomial (1.1), whose minimum value is  $\nu(f)$ . Let us denote the solution returned by a given heuristic  $H$  as  $\mathbf{x}^H$ .

The traditional way to measure the performance of heuristics on QUBO problems is to use the *relative error*

$$R(H; f) \stackrel{\text{def}}{=} \frac{f(\mathbf{x}^H) - \nu(f)}{\nu(f)}.$$

In spite of being widely used, the relative error  $R(H; f)$  does not satisfy some important properties that these performance indicators should have. Namely, the relative

error depends on the constant of the function,

$$\lim_{c \rightarrow \infty} R(H; f + c) = 0, \text{ and}$$

the relative error depends on complementation of variables. For instance,  $R(H; f)$  would report different results for

$$\begin{aligned} f(x, y) &= -2x - 3y + 3xy, \quad \text{and} \\ 3+ \quad f(x, \bar{y}) &= x + 3y - 3xy, \quad \text{where } \bar{y} = 1 - y. \end{aligned}$$

Let us note that a minimizer of  $f$  can easily be derived from the optimum of the functions obtained with the previous transformations. Therefore, the measure of quality of heuristics should be independent of such variations.

An idea to solve this issue is to adopt a normalization of the relative error by using a constant  $c$  which satisfies

$$\text{Exp}[f + c] = [f + c] \left( \frac{1}{2}, \dots, \frac{1}{2} \right) = 0.$$

As a consequence of the previous relation, the *normalized relative error*

$$N(H; f) \stackrel{\text{def}}{=} R \left( H; f - f \left( \frac{1}{2}, \dots, \frac{1}{2} \right) \right) = \frac{f(\mathbf{x}^H) - \nu(f)}{\nu(f) - f \left( \frac{1}{2}, \dots, \frac{1}{2} \right)}.$$

is obtained. It turns out that  $N$  was proposed by Zemel [240] (see also [184]), which concludes that  $N$  is a “proper” measure to evaluate the quality of approximate solutions to 0–1 programming problems.

Since the optimum is not known for many of the test problems, we also will use the *approximative relative error*

$$G(H; f) \stackrel{\text{def}}{=} \frac{f(\mathbf{x}^H) - f(\mathbf{x}^{\text{best}})}{f(\mathbf{x}^{\text{best}})},$$

and the *approximative normalized error*

$$K(H; f) \stackrel{\text{def}}{=} \frac{f(\mathbf{x}^H) - f(\mathbf{x}^{\text{best}})}{f(\mathbf{x}^{\text{best}}) - f(\frac{1}{2}, \dots, \frac{1}{2})},$$

for evaluation purposes, where  $\mathbf{x}^{\text{best}}$  stands for the best known solution to the minimum value of  $f$ .

Given a computer system  $\mathcal{S}$ , the computing time of heuristic  $H$  applied to problem  $f$  in  $\mathcal{S}$  is denoted by  $T(\mathcal{S}; H; f)$ .

To analyze the performance of the proposed algorithms in a particular family  $\mathcal{F}$  of QUBO problems, we shall frequently use the average value  $\bar{W}$ , associated to a heuristic  $H$  in a set of problems  $\mathcal{F}$ , i.e.

$$\bar{W}(H; \mathcal{F}) \stackrel{\text{def}}{=} \text{Exp}[W(H; f) | f \in \mathcal{F}],$$

where  $W$  stands for any of the performance indicators previously defined:  $N$ ,  $K$ ,  $R$  or  $G$ . Similarly, the average computing time of a heuristic  $H$  in a set of problems  $\mathcal{F}$  using a computer system  $\mathcal{S}$  is denoted as

$$\bar{T}(\mathcal{S}; H; \mathcal{F}) \stackrel{\text{def}}{=} \text{Exp}[T(\mathcal{S}; H; f) | f \in \mathcal{F}].$$

The variance of the results of  $W$  in a set of problems  $\mathcal{F}$  is an important performance measure that we also consider:

$$\sigma_W^2(H; \mathcal{F}) \stackrel{\text{def}}{=} \text{Var}[W(H; f) | f \in \mathcal{F}].$$

#### 6.1.4 Computational results

Five class families of one-pass heuristics were described in Section 6.1. In Table 6.3 a list of 49 one-pass algorithms for QUBO is presented. Each method listed in the table has a name and is briefly distinguished from the others, so that they can be referenced in the text that follows. The analysis includes only 48 variants of the proposed 49 algorithms

since the solution returned by ONE-PASS-BLA is equal to the solution returned by ONE-PASS-R( $I_1$ ) (see Theorem 6.5).

Except for the ONE-PASS-DDT-L heuristic, which was implemented using the implication network structure (see definition in Section 5.3), all the heuristics were implemented using an upper triangular (dense) matrix structure. This choice was made with the intuitive idea of allowing all heuristics to compete equally in terms of using the same data structure.

We remark the fact that we did not study several other data structures for the DDT heuristic of Boros et al. [58], which uses a signed graph as input to the algorithm. If an adjacency list is adopted to represent the signed graph, then the DDT heuristic would outperform the corresponding heuristic based on the matrix representation in problems with density smaller than 19%. However, for the other cases, the usage of the matrix structure would clearly provide faster runs of the DDT algorithm.

The number of test problems considered in the subsequent analysis is 5 458. Except for the 36 massive planar graphs of the RUDY benchmark, for which the adopted matrix structure would be prohibitive in terms of the available capacity of computer memory, we considered here all the problems described in Chapter 3. The characteristics of the data set can be seen in Table 3.1. A summary of the classes of QUBO problems considered is presented in Table 6.4. The total number of experiments considered in this section is 258 404.

The ONE-PASS-DDT-L heuristic was only considered in 1 942 datasets. We shall disregard this heuristic from the analysis that follows, since its performance was clearly inferior to the other heuristics in all aspects (e.g., the average relative gap ( $G$ ) value is over 200%).

The ONE-PASS-BLA heuristic was tested in 5 394 problems. The 62 problems that were not tried for this heuristic have a sum of coefficients which is larger than the largest valid (integer) number of the 32 bits computer used for testing (i.e.  $2^{31}$ ); recall that this particular heuristic uses this parameter at every iteration of the algorithm (see (6.4)).

Table 6.3: One-pass heuristics for QUBO considered in the computational experiments.

<i>Class</i>	<i>Name</i>	<i>Options</i>
DDT	ONE-PASS-DDT-L	largest term from the roof-dual posiform
	ONE-PASS-DDT-B	largest bi-term of the bi-form
Best Linear Approximation	ONE-PASS-BLA	best linear approximation
	ONE-PASS-BHLA	best homogeneous linear approximation
Probabilistic	ONE-PASS-P-U	$\Delta_j \sim \text{Uniform}(L_j, U_j)$
	ONE-PASS-P-N( $\sigma^2 = \frac{1}{4}$ )	$\Delta_j \sim \text{Normal}; x_j \sim \text{Bernoulli}(\frac{1}{2})$
	ONE-PASS-P-N( $\sigma^2 = \frac{1}{12}$ )	$\Delta_j \sim \text{Normal}; x_j \sim \text{Uniform}(0, 1)$
	ONE-PASS-P-N( $\sigma^2 = \frac{1}{48}$ )	$\Delta_j \sim \text{Normal}; x_j \sim \text{Uniform}(0.25, 0.75)$
	ONE-PASS-P-N( $\sigma^2 = \frac{1}{300}$ )	$\Delta_j \sim \text{Normal}; x_j \sim \text{Uniform}(0.40, 0.60)$
	ONE-PASS-P-N( $\sigma^2 = \frac{1}{1200}$ )	$\Delta_j \sim \text{Normal}; x_j \sim \text{Uniform}(0.45, 0.55)$
	ONE-PASS-P-N( $\sigma^2 = \frac{1}{30000}$ )	$\Delta_j \sim \text{Normal}; x_j \sim \text{Uniform}(0.49, 0.51)$
Rounding	ONE-PASS-R( $I_1$ )	starting point is $I_1 \equiv (\frac{1}{2}, \dots, \frac{1}{2})$
	ONE-PASS-R( $I_2$ )	starting point is $I_2$
	ONE-PASS-R( $I_3$ )	starting point is $I_3$
	ONE-PASS-R( $I_4$ )	starting point is $I_4$
	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ )	s.p. is $I_5; x_j \sim \text{Bernoulli}(\frac{1}{2})$
	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{12}$ )	s.p. is $I_5; x_j \sim \text{Uniform}(0, 1)$
	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{48}$ )	s.p. is $I_5; x_j \sim \text{Uniform}(0.25, 0.75)$
	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{300}$ )	s.p. is $I_5; x_j \sim \text{Uniform}(0.40, 0.60)$
	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{1200}$ )	s.p. is $I_5; x_j \sim \text{Uniform}(0.45, 0.55)$
	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{30000}$ )	s.p. is $I_5; x_j \sim \text{Uniform}(0.49, 0.51)$
	ONE-PASS-R( $I_2^*$ )	starting point is $I_2^*$
	ONE-PASS-R( $I_3^*$ )	starting point is $I_3^*$
	ONE-PASS-R( $I_4^*$ )	starting point is $I_4^*$
	ONE-PASS-R( $I_5^*, \sigma^2 = \frac{1}{4}$ )	s.p. is $I_5^*; x_j \sim \text{Bernoulli}(\frac{1}{2})$
	ONE-PASS-R( $I_5^*, \sigma^2 = \frac{1}{12}$ )	s.p. is $I_5^*; x_j \sim \text{Uniform}(0, 1)$
	ONE-PASS-R( $I_5^*, \sigma^2 = \frac{1}{48}$ )	s.p. is $I_5^*; x_j \sim \text{Uniform}(0.25, 0.75)$
	ONE-PASS-R( $I_5^*, \sigma^2 = \frac{1}{300}$ )	s.p. is $I_5^*; x_j \sim \text{Uniform}(0.40, 0.60)$
	ONE-PASS-R( $I_5^*, \sigma^2 = \frac{1}{1200}$ )	s.p. is $I_5^*; x_j \sim \text{Uniform}(0.45, 0.55)$
	ONE-PASS-R( $I_5^*, \sigma^2 = \frac{1}{30000}$ )	s.p. is $I_5^*; x_j \sim \text{Uniform}(0.49, 0.51)$
	Rounding with local optimality conditions	ONE-PASS-R <sup>+</sup> ( $I_1$ )
ONE-PASS-R <sup>+</sup> ( $I_2$ )		starting point is $I_2$
ONE-PASS-R <sup>+</sup> ( $I_3$ )		starting point is $I_3$
ONE-PASS-R <sup>+</sup> ( $I_4$ )		starting point is $I_4$
ONE-PASS-R <sup>+</sup> ( $I_5, \sigma^2 = \frac{1}{4}$ )		s.p. is $I_5; x_j \sim \text{Bernoulli}(\frac{1}{2})$
ONE-PASS-R <sup>+</sup> ( $I_5, \sigma^2 = \frac{1}{12}$ )		s.p. is $I_5; x_j \sim \text{Uniform}(0, 1)$
ONE-PASS-R <sup>+</sup> ( $I_5, \sigma^2 = \frac{1}{48}$ )		s.p. is $I_5; x_j \sim \text{Uniform}(0.25, 0.75)$
ONE-PASS-R <sup>+</sup> ( $I_5, \sigma^2 = \frac{1}{300}$ )		s.p. is $I_5; x_j \sim \text{Uniform}(0.40, 0.60)$
ONE-PASS-R <sup>+</sup> ( $I_5, \sigma^2 = \frac{1}{1200}$ )		s.p. is $I_5; x_j \sim \text{Uniform}(0.45, 0.55)$
ONE-PASS-R <sup>+</sup> ( $I_5, \sigma^2 = \frac{1}{30000}$ )		s.p. is $I_5; x_j \sim \text{Uniform}(0.49, 0.51)$
ONE-PASS-R <sup>+</sup> ( $I_2^*$ )		starting point is $I_2^*$
ONE-PASS-R <sup>+</sup> ( $I_3^*$ )		starting point is $I_3^*$
ONE-PASS-R <sup>+</sup> ( $I_4^*$ )		starting point is $I_4^*$
ONE-PASS-R <sup>+</sup> ( $I_5^*, \sigma^2 = \frac{1}{4}$ )		s.p. is $I_5^*; x_j \sim \text{Bernoulli}(\frac{1}{2})$
ONE-PASS-R <sup>+</sup> ( $I_5^*, \sigma^2 = \frac{1}{12}$ )		s.p. is $I_5^*; x_j \sim \text{Uniform}(0, 1)$
ONE-PASS-R <sup>+</sup> ( $I_5^*, \sigma^2 = \frac{1}{48}$ )		s.p. is $I_5^*; x_j \sim \text{Uniform}(0.25, 0.75)$
ONE-PASS-R <sup>+</sup> ( $I_5^*, \sigma^2 = \frac{1}{300}$ )		s.p. is $I_5^*; x_j \sim \text{Uniform}(0.40, 0.60)$
ONE-PASS-R <sup>+</sup> ( $I_5^*, \sigma^2 = \frac{1}{1200}$ )		s.p. is $I_5^*; x_j \sim \text{Uniform}(0.45, 0.55)$
ONE-PASS-R <sup>+</sup> ( $I_5^*, \sigma^2 = \frac{1}{30000}$ )		s.p. is $I_5^*; x_j \sim \text{Uniform}(0.49, 0.51)$

Table 6.4: Families of QUBO problems used to evaluate the proposed one-pass heuristics.

<i>Families of QUBO Problems</i>	<i>Number of Problems</i>	<i>Optimum Known</i>	<i>Variables (n)</i>
Benchmarks	143	69	20 to 6 000
Randomly generated	3 728	285	25 to 30 000
MAX-Clique	138	119	28 to 4 000
MIN-VC (planar)	400	400	1 000 to 4 000
MAX-CUT	375	36	125 to 10 000
MAX-2-SAT	674	360	50 to 400
All Problems	5 458	1 269	20 to 30 000

The application of an immediate rounding step to the variables, which were not yet fixed by the one-pass procedure, that have partial derivatives with constant sign (thus, satisfying the local optimality conditions for any 0-1 value of the remaining variables), proved to be useful in 16.3% of the cases, while it returned worse results in 5.4% of the cases. Formally, these results can be described as

$$\begin{aligned} \frac{|\{f \in \mathcal{C}, I \in \mathbb{I} \mid f(\text{ONE-PASS-R}^+(I)) > f(\text{ONE-PASS-R}(I))\}|}{|\mathcal{C}| |\mathbb{I}|} &= 16.3\%, \\ \frac{|\{f \in \mathcal{C}, I \in \mathbb{I} \mid f(\text{ONE-PASS-R}^+(I)) = f(\text{ONE-PASS-R}(I))\}|}{|\mathcal{C}| |\mathbb{I}|} &= 78.3\%, \\ \frac{|\{f \in \mathcal{C}, I \in \mathbb{I} \mid f(\text{ONE-PASS-R}^+(I)) < f(\text{ONE-PASS-R}(I))\}|}{|\mathcal{C}| |\mathbb{I}|} &= 5.4\%, \end{aligned}$$

where  $\mathcal{C}$  corresponds to the complete set of 5 458 test problems, and  $I$  corresponds to the set of 19 initialization procedures considered.

In the current implementation, the average computing time of the heuristics, which consider the local optimality conditions, is 39.5% higher than the corresponding heuristic versions that do not consider the optimality conditions. For this reason, and because the solutions returned are typically similar, the computational experiments and analysis that follow do not include these (ONE-PASS-R<sup>+</sup>) heuristics.

#### 6.1.4.1 Computing time

The worst complexity time of all one-pass heuristics implemented is  $O(n^2)$ . All heuristics were implemented using the same data structure (i.e. a triangular dense matrix). Seven heuristics use the probabilistic approach presented in Section 6.1.2.2, whereas the remaining nineteen heuristics use the rounding procedures presented in Section 6.1.2.3.

The algorithms were implemented in C++, compiled using Microsoft Windows 32-bit C/C++ Optimizing Compiler (version 12) for 80x86, and linked with the Microsoft Incremental Linker (version 6) using the single-threaded run-time library.

All the experiments were found using the same computer system, which is based on a Xeon 3.06 GHz, 3.5 GB RAM and Windows XP. The previous choices allowed us to evaluate the heuristics computing times under the same conditions.

Table 6.5 displays average computing times of the one-pass heuristics on several families of QUBO problems. From the 26 one-pass heuristics tested, Table 6.5 lists the fastest (average) computing times, the overall average computing times and the slowest (average) computing times for the various groups of problems.

Table 6.5: Computing time of the one-pass heuristics across several families of QUBO problems.

<i>Families of QUBO Problems</i>	<i>Computing Time of the One-Pass Heuristics having</i>		
	<i>Fastest Time</i>	<i>Average Time</i>	<i>Slowest Time</i>
Benchmarks	0.2 s	0.4 s	0.6 s
Randomly generated	1.9 s	2.9 s	3.8 s
MAX-Clique	<0.1 s	<0.1 s	<0.1s
MIN-VC (planar)	0.3 s	0.5 s	0.6 s
MAX-CUT	0.2 s	0.4 s	0.5 s
MAX-2-SAT	<0.1 s	<0.1 s	<0.1 s
All Problems	1.5 s	2.4 s	3.1 s

If all problems are considered, it can be seen that the average running time of the complete set of heuristics varies between 1.5 and 3.1 seconds. The more time consuming tests occurred in the large randomly generated cases, clearly indicating that the number of variables ( $n$ ) is highly correlated with the heuristics computing time.

Figure 6.3 displays the average computing time of the fastest, slowest and average case one-pass heuristics according to the number of variables. The following average computing times are displayed:

- For QUBOs having 1 000 variables, the heuristics time is in the interval  $]0.0s, 0.2s]$ ;
- For QUBOs having 5 000 variables, the heuristics time is in the interval  $[1.2s, 4.7s]$ ;
- For QUBOs having 10 000 variables, the heuristics time is in the interval  $[8s, 17s]$ ;

- For QUBOs having 30 000 variables, the heuristics time is in the interval  $[96s, 272s]$ .

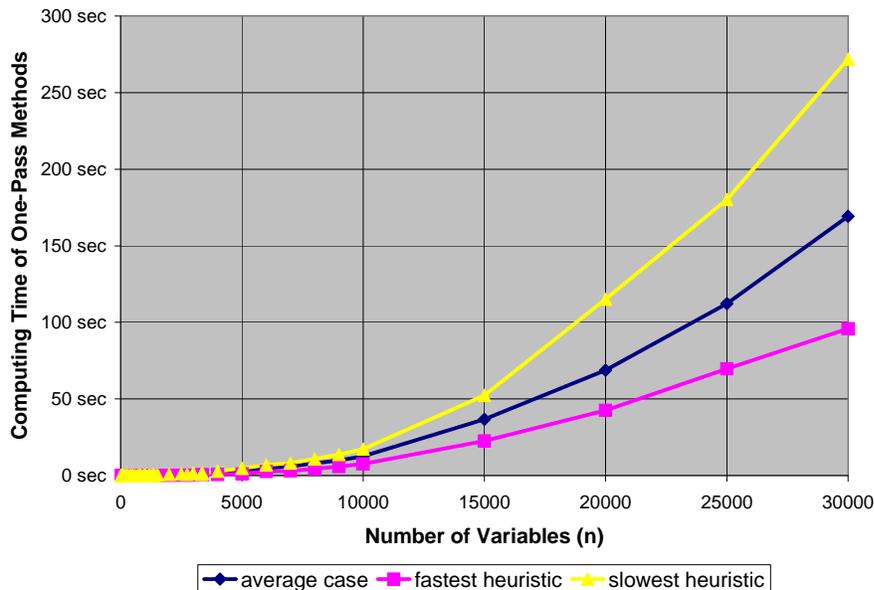


Figure 6.3: Average computing time of the (fastest, slowest, average case) one-pass QUBO heuristics according to the number of variables ( $n$ ).

It should be remarked the fact that the test problems include instances with very large density. For instance some of the 30 000 variable's problems have 90% density, thus implying that the associated QUBO has almost 405 million nonzero quadratic terms.

#### 6.1.4.2 Quality of solutions

The quality of the proposed one-pass heuristics can be analyzed through the *normalized* or the *relative* error, earlier introduced in Section 6.1.3.

Table 6.6 lists the *approximate* expected relative and normalized errors of the studied set of one-pass heuristics, for several families of QUBO problems. It is interesting to note that the performance of the heuristics varies considerably using these two criteria. For instance, the most “difficult” class of problems studied for the proposed one-pass heuristics is MAX-Clique if the relative error is considered, and is MAX-CUT if the normalized error is used instead.

Optimality is known for 23% of the problems that were used for testing. The best

Table 6.6: Quality of the one-pass heuristics across several families of QUBO problems.

(a) Approximate relative error.

<i>Families of QUBO Problems</i>	<i>Relative Error <math>G</math> of One-Pass Heuristics having</i>					
	<i>Larger Error <math>\pm</math> St.Dev.</i>		<i>Average Error <math>\pm</math> St.Dev.</i>		<i>Smaller Error <math>\pm</math> St.Dev.</i>	
Benchmarks	5%	$\pm$ 12%	4%	$\pm$ 7%	2.3%	$\pm$ 4.5%
Randomly generated	29%	$\pm$ 24%	25%	$\pm$ 18%	21%	$\pm$ 14%
MAX-Clique	32%	$\pm$ 20%	28%	$\pm$ 15%	25%	$\pm$ 12%
MIN-VC (planar)	30%	$\pm$ 9%	6%	$\pm$ 1.6%	0.1%	$\pm$ 0.1%
MAX-CUT	8%	$\pm$ 7%	6%	$\pm$ 3%	4.4%	$\pm$ 2.7%
MAX-2-SAT	5%	$\pm$ 17%	4%	$\pm$ 8%	2.7%	$\pm$ 3.7%
All Problems	22%	$\pm$ 23%	18%	$\pm$ 18%	15.7%	$\pm$ 15.2%

(b) Approximate normalized error.

<i>Families of QUBO Problems</i>	<i>Normalized Error <math>W</math> of One-Pass Heuristics having</i>					
	<i>Larger Error <math>\pm</math> St.Dev.</i>		<i>Average Error <math>\pm</math> St.Dev.</i>		<i>Smaller Error <math>\pm</math> St.Dev.</i>	
Benchmarks	3%	$\pm$ 3%	1%	$\pm$ 1%	0.7%	$\pm$ 0.6%
Randomly generated	1.4%	$\pm$ 2.6%	0.7%	$\pm$ 1.2%	0.5%	$\pm$ 0.8%
MAX-Clique	3%	$\pm$ 14%	0.8%	$\pm$ 3%	0.4%	$\pm$ 0.9%
MIN-VC (planar)	25%	$\pm$ 8%	5%	$\pm$ 1.3%	0.0%	$\pm$ 0.1%
MAX-CUT	13%	$\pm$ 5%	10%	$\pm$ 3%	7.9%	$\pm$ 2.6%
MAX-2-SAT	8%	$\pm$ 6%	6%	$\pm$ 5%	4.7%	$\pm$ 4.0%
All Problems	4%	$\pm$ 7%	2.4%	$\pm$ 4.1%	1.7%	$\pm$ 3.0%

known solutions of the open problems were found by meta-heuristic approaches, either obtained from the literature or by using our own implementations.

The expected approximate *relative* error of the one-pass heuristics studied is particularly high for the groups of randomly generated problems and for MAX-Clique, having respective expected errors in  $[21\%, 29\%]$  and  $[25\%, 32\%]$ . However, for the remaining classes of QUBOs the expected relative error is inferior to 10%, and if the best one-pass heuristic is considered the error is smaller than 5%.

The expected approximate *normalized* error of the one-pass heuristics studied is smaller than 3% for the randomly generated problems and for the MAX-Clique problems. The “best” one-pass heuristic studied provides an expected error smaller than 0.7%, and variance smaller than 0.9%, for the groups of randomly generated problems, benchmark problems, MAX-Clique and MIN-VC of planar graphs. The expected normalized error for MAX-CUT problems is in the interval  $[7.9\%, 13\%]$ , and for MAX-2-SAT problems is in the interval  $[4.7\%, 8\%]$ .

Table 6.7 gives the one-pass heuristics that minimize the approximate relative and normalized errors. There is no clear winner for each class of QUBO problems analyzed. The probabilistic based methods are somewhat superior for the standard benchmarks and for MIN-VC of planar graphs. However, in general the rounding methods provide better quality solutions.

Table 6.7: One-pass heuristics that minimize the approximate errors of several families of QUBO problems.

<i>Families of QUBO Problems</i>	<i>Heuristic that Minimizes the Expected</i>	
	<i>Relative Error (G)</i>	<i>Normalized Error (K)</i>
Benchmarks	ONE-PASS-P-N( $\sigma^2 = \frac{1}{48}$ )	ONE-PASS-P-N( $\sigma^2 = \frac{1}{48}$ )
Randomly generated	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ )	ONE-PASS-R( $I_4$ )
MAX-Clique	ONE-PASS-R( $I_2$ )	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ )
MIN-VC (planar)	any probabilistic	any probabilistic
MAX-CUT	ONE-PASS-R( $I_3$ )	ONE-PASS-R( $I_4$ )
MAX-2-SAT	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ )	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ )
All Problems	ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ )	ONE-PASS-R( $I_4$ )

The rounding heuristics ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ ) and ONE-PASS-R( $I_4$ ) frequently provide the best quality solutions of the one-pass methods that we have studied. It should be remarked the fact that the later heuristic does require the use and maintenance of simpler data structures, and therefore it produces heuristic solutions a little faster than the former heuristic. For instance, the average computing time of ONE-PASS-R( $I_4$ ), on the larger QUBOs having 30 000 variables and 90% density, is 164 seconds, whereas for the heuristic ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ ) the average computing time on the same large problems is 174 seconds.

### 6.1.5 Comparing proposed methods to other results from the literature

Several one-pass heuristics for QUBO have been proposed in the past. Boros et al. [58] proposed the DDT heuristics (see Section 6.1.1). Merz and Freisleben [178] proposed ONE-PASS-R( $I_2$ ), i.e. a steepest descent (greedy) heuristic having as starting point the center. Glover et al. [107] proposed a series of one-pass heuristics based on the posiform representation of the problem.

Since a quadratic pseudo-Boolean function has possibly many posiform representations of it, then the performance of the proposed heuristics, based on this structure, usually differs for different posiforms representing the same function. For this reason, we have benchmarked the proposed one-pass heuristics only with those test problems previously studied by Glover et al. [107].

It should be remarked that the DDT heuristics also assume a posiform representation for the function. In this section we have only investigate the special DDT heuristic based on the bi-form (see Section 8.1) representation (i.e. ONE-PASS-DDT-B), which is uniquely defined for every quadratic pseudo-Boolean function.

Based on the full range of computational testing conducted during the heuristics selection phase, Glover et al. [107] conclude that their proposed methods  $A2$  ( $A2n$  and  $A2t$ ) and  $V3$  ( $V3n$  and  $V3t$ ) are effective methods for the problems that they have tested. Considering both solution quality and computing time,  $A2n$  gave overall the best performance, followed closely by  $A2t$ ,  $V3n$  and  $V3t$ , in this order. For comparison with our proposed one-pass methods, we shall use a heuristic based on the maximum of the four best one-pass heuristics proposed by Glover et al. [107]; we call it *best* ( $A2, V3$ ).

The solution quality of the four one-pass heuristics  $A2n$ ,  $A2t$ ,  $V3n$  and  $V3t$  has been compared with the best known solutions on standard publicly available benchmarks. All the test problems considered are *maximization* QUBO problems. These benchmarks include the following groups of problems (see Table 3.2 in Section 3.1.1):

- Beasley [37] QUBO problems – Consists of a set of 60 randomly generated test problems, where the number of variables  $n$  varies from 50 to 2 500, and having 10% density; the coefficients of the multilinear representation of the functions range between -100 and 100;
- $F_1$  from Glover et al. [108] – Consists of 5 problems having 500 variables with densities ranging from 10% to 100%; the linear coefficients are uniformly distributed in  $[-75, 75]$  and the quadratic coefficients are uniformly distributed in  $[-50, 50]$ .
- $G_1$  from Glover et al. [109] – Consists of 10 problems having 1 000 variables with densities ranging from 10% to 100%, but only 5 instances are considered in [107];

the coefficients have the same distributions as those of problems belonging to group  $F_1$ .

- $B$  from Glover et al. [108] – Consists of 10 problems with all nonzero quadratic coefficients being negative, 100% dense problems with the number of variables ranging from 20 to 125.
- $F_2$  and  $G_2$  from Kochenberger et al. [158] – Consists of 10 problems with all nonzero quadratic coefficients being negative, respectively having 500 and 1000 variables, and densities varying from 10% to 100%.

Best known values of the previous problems are listed in Table A.1, Table A.2, Table A.3 and Table A.4 of the Appendix.

We have noticed that the quality of solutions of the one-pass heuristics depends heavily on the distribution of the coefficients sign of the quadratic terms of the multilinear polynomials. Problems in the Beasley family,  $F_1$  and  $G_1$  have approximately 50% positive nonzero quadratic terms (i.e.  $\bar{\rho} = 0.5$ ), whereas problems in the groups  $B$ ,  $F_2$  and  $G_2$  have no positive quadratic terms (i.e.  $\bar{\rho} = 0.0$ ).

Table 6.8: Quality of solutions comparison between the proposed methods and the one-pass heuristics from the literature.

One-Pass Heuristic	Exp. Relative Error ( $G$ ) for		Exp. Normalized Error ( $K$ ) for	
	$\bar{\rho} = 0.5$	$\bar{\rho} = 0.0$	$\bar{\rho} = 0.5$	$\bar{\rho} = 0.0$
best( $A2, V3$ ) ([107])	8.5%	12.4%	8.27%	0.049%
ONE-PASS-DDT-B ([58])	1.1%	31.2%	1.01%	0.146%
ONE-PASS-R( $I_1$ ) ([178])	1.2%	32.1%	1.04%	0.149%
ONE-PASS-R( $I_3$ )	1.0%	11.7%	0.90%	<b>0.041%</b>
ONE-PASS-R( $I_4$ )	0.9%	13.1%	0.87%	0.050%
ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ )	1.1%	12.7%	1.01%	0.048%
ONE-PASS-P-N( $\sigma^2 = \frac{1}{48}$ )	<b>0.8%</b>	<b>10.8%</b>	<b>0.79%</b>	0.043%

Table 6.8 displays the approximate relative and normalized errors of several one-pass methods for the groups of problems listed above. It can be seen there that, for the analyzed groups of problems, the probabilistic based one-pass heuristic ONE-PASS-P-N( $\sigma^2 = \frac{1}{48}$ ) has the smallest expected relative errors for both groups of problems having  $\bar{\rho} = 0.5$  and for problems having  $\bar{\rho} = 0.0$ . This heuristic has also the smallest expected normalized error for problems having  $\bar{\rho} = 0.5$ .

ONE-PASS-R( $I_3$ ), followed very closely by the probabilistic heuristic, has the smallest expected normalized error for problems having  $\bar{\rho} = 0.0$ .

It can also be seen that the probabilistic heuristic and the rounding heuristics ONE-PASS-R( $I_k$ ) ( $k = 3, 4$ ) and ONE-PASS-P-N( $\sigma^2 = \frac{1}{48}$ ) have similar performance.

It is interesting to note that  $I_3$  is a starting point which is determined by parameter  $\rho$  (see Table 6.2), and therefore it is somewhat expected that the quality of solutions provided by this heuristic is improved according to the value of the parameter  $\bar{\rho}$  considered.

The one-pass methods from the literature are only good choices for one of the groups, e.g.  $\text{best}(A2, V3)$  is good for problems having  $\bar{\rho} = 0.0$ , and both ONE-PASS-DDT-B and ONE-PASS-R( $I_1$ ) are good options for problems having  $\bar{\rho} = 0.5$ .

## 6.2 Local-search heuristics

In this section, a large family of monotone heuristics is considered, in which the value of  $f$  is increased iteratively, by changing the value of only one of the variables (procedures of this type are sometimes called *local search* or *1-opt* in the literature). We focus on variants of local search in which we stop only if no further improvement can be achieved by changing the value of a single variable. The main aim of this study is to evaluate a new family of starting point and variable selection techniques, and to demonstrate that they can substantially improve the effectiveness of local search methods without diminishing their efficiency.

Let us remark that the considered local search procedures have no theoretical guarantees to terminate in polynomial time (in terms of the size of the input). To achieve potentially faster, polynomial time termination, one may change the stopping criterion, and have the procedure stop after a certain (polynomial number) of iterations.

In a variant of this type (so called *one-pass* procedures) once a variable received a binary assignment, it is not changed subsequently (see e.g. [58, 107, 178]). Hence, such one-pass algorithms terminate in at most  $n$  iterations. We have examined a large family of one-pass heuristics in Section 6.1.

In the next subsection we describe the proposed family of heuristics, and in the subsequent subsections we thoroughly analyze them by experimenting on large number of benchmark problems from the literature as well as on randomly generated problems sets. As our analysis shows, there are substantial differences depending on how we choose the starting point for our procedure, as well as on the strategy applied to choose a next variable and its value. Finally, we compare the results of the best heuristics we found to the best results and methods from the literature, and demonstrate that despite the simplicity of this procedure, it still performs quite competitively, even with more sophisticated (and hence more time consuming) approaches.

### 6.2.1 Methods

In this section we describe a parametric family of local search heuristics for QUBO, and the motivating mathematical theory behind the choices we propose for the parameters.

#### 6.2.1.1 Basic concepts and notations

To measure computational complexity, let us denote by  $\omega_i(f)$  the number of occurrences of variable  $x_i$  in the polynomial expression (1.5) of  $f$ , i.e.,

$$\omega_i(f) = \begin{cases} 1 + |\{(i, j) \mid 1 \leq i < j \leq n, c_{ij} \neq 0\}| & \text{if } c_i \neq 0, \text{ and} \\ |\{(i, j) \mid 1 \leq i < j \leq n, c_{ij} \neq 0\}| & \text{otherwise.} \end{cases}$$

Let us denote by  $\text{size}(f) = \sum_{i=1}^n \omega_i(f)$  the “length” of the polynomial expression of  $f$ .

Binary vectors, no single component of which can be changed so as to decrease the value of a pseudo-Boolean function  $f$ , are called *local minima* of  $f$ . It should be noted that the number of local minima can be exponentially large (see [191]) and that the computational complexity of finding a local minimum of a quadratic pseudo-Boolean function is open (see e.g. [194]). It will be seen in the sequel that the large volume of computational experience carried out for finding a local optimum of a quadratic pseudo-Boolean function, indicates clearly that in most cases such a local optimum can be obtained efficiently. Moreover, by choosing carefully the starting point and

the sequence of local improvement steps, it will be seen that usually the value of the function in the local optimum obtained in this way is low.

The mathematical property, which makes it possible that the above described simple monotone procedure works successfully, is the multi-linearity of pseudo-Boolean functions.

Let us consider the partial derivatives  $\Delta_i$  ( $i = 1, \dots, n$ ), given by (4.4), of a quadratic pseudo-Boolean function  $f$ . Let us note that the *residual* function,  $\theta_i = f - x_i \Delta_i$  does not depend on variable  $x_i$ , for  $i = 1, \dots, n$ , and hence the characterization of local minimum of  $f$  follows immediately from the necessary conditions of optimality established by Proposition 4.1 (see Section 4.2).

The above simple characterization serves as a basis for a number of heuristic algorithms in the literature. The core of such procedures is the following *local improvement* along a component for which the corresponding necessary condition (4.3) of optimality is violated:

$\mathbf{x}' = \text{IMPROVE}(f, \mathbf{x}, i)$	
<b>Input:</b>	A quadratic pseudo-Boolean function $f$ , given by (1.5), and a vector $\mathbf{x}$ .
<b>Main Step:</b>	Obtain $\mathbf{x}'$ from $\mathbf{x}$ by switching the value of its $i$ th component to 0 if $\Delta_i(\mathbf{x}) \geq 0$ and to 1 if $\Delta_i(\mathbf{x}) < 0$ .
<b>Output:</b>	Vector $\mathbf{x}'$ .

In the family of heuristics considered, we shall produce a finite sequence  $\mathbf{x}^{(k)}$  of vectors for which  $\mathbf{x}^{(k+1)} = \text{IMPROVE}(f, \mathbf{x}^{(k)}, i_{(k)})$ , and consequently  $f(\mathbf{x}^{(k)}) \geq f(\mathbf{x}^{(k+1)})$  hold for every  $k$ . We can obtain a uniquely described heuristic procedure by further specifying

- (i) how to choose  $\mathbf{x}^{(0)}$ ;
- (ii) how to choose the index  $i_k$ ; and
- (iii) when to stop.

Before going into these details, let us note that the above discussion did not really utilize the fact that variables take only binary values. In fact, (1.5) is a real-valued expression, and can be evaluated for an arbitrary real vector  $\mathbf{x} \in \mathbb{U}^n$ , just like the expressions of  $\Delta_i$ ,  $\theta_i$ , the local optimality conditions, and  $\text{IMPROVE}(f, \mathbf{x}, i)$ .

The continuous extensions properties of pseudo-Boolean functions (see Section 4.7) suggest that we could also start from a fractional vector  $\mathbf{p} \in \mathbb{U}^n$  and use  $\text{IMPROVE}$  to obtain a “better” binary vector.

**Proposition 6.3.** *Given the expression (1.5) of a quadratic pseudo-Boolean function  $f$  and a vector  $\mathbf{p} \in \mathbb{U}^n$ , a binary vector  $\mathbf{x} \in \mathbb{B}^n$  for which  $f(\mathbf{x}) \leq f(\mathbf{p})$  can be obtained in  $O(\text{size}(f))$  time, by applying  $\text{IMPROVE}$  at most  $n$  times.*

*Proof.* Let us start with  $\mathbf{x}^{(0)} = \mathbf{p}$ , and let  $\pi = (i_0, \dots, i_t)$  be a permutation of the indices of the fractional components of  $\mathbf{p}$ . Let us apply then  $\mathbf{x}^{(j+1)} = \text{IMPROVE}(f, \mathbf{x}^{(j)}, i_j)$  for  $j = 0, \dots, t$  (we have  $t < n$ ), and let  $\mathbf{x}^\pi = \mathbf{x}^{(t+1)}$ . Clearly, for any permutation  $\pi$  of the fractional components of  $\mathbf{p}$  we obtain a binary vector  $\mathbf{x}^\pi$  satisfying the claim in the statement. Furthermore, in  $O(\text{size}(f))$  time we can build a data structure associating variables with their occurrences, and holding the values of  $f$  and  $\Delta_i$  for  $i = 1, \dots, n$  at vector  $\mathbf{p}$ . In the subsequent calls of  $\text{IMPROVE}$  we update these values, using the previously built data structure. Clearly, in the  $j$ th call of  $\text{IMPROVE}$  computations depend only on the occurrences of variable  $x_{i_j}$ , and thus this step can be executed in  $O(\omega_{i_j}(f))$  time. Consequently, the total computational time is limited by  $O(\text{size}(f))$ . □

The procedure described in the above proof is in fact a simple variant of the heuristic algorithms we consider in this paper. It is known also as *pseudo-Boolean rounding* (see Section 4.7). An immediate corollary of the above is the fact that optimizing a pseudo-Boolean function over the unit cube or over its extreme points results in the same optimum.

Let us remark that while a violation of the local optimality conditions (4.3) at a binary vector implies  $\Delta_i \neq 0$  for some index  $i = 1, \dots, n$ , the same does not necessarily hold true at a fractional vector. For instance, if  $f(x_1, x_2) = x_1 + x_2 - 4x_1x_2$  then we

have  $\Delta_1(\frac{1}{4}, \frac{1}{4}) = \Delta_2(\frac{1}{4}, \frac{1}{4}) = 0$ , even though the fractional point  $\mathbf{p} = (\frac{1}{4}, \frac{1}{4})$  is not a local maximum. This implies that special care has to be taken when selecting index  $i$  for IMPROVE, whenever we start with a fractional vector.

### 6.2.1.2 Algorithms

We shall describe below a general outline of the proposed family of algorithms. These procedures are based on iteratively calling IMPROVE. There are however a great number of possibilities on the way to initiate the algorithm, and to choose an index for IMPROVE. The algorithms will be described using three independent parameters. First, a method will be specified by the way it chooses the initial vector  $\mathbf{x}^{(0)} \in \mathbb{U}^n$ ; we shall consider a set  $\mathbb{I}$  of various alternatives for this choice. Let us remark that in view of Proposition 4.6, the initial point  $\mathbf{x}^{(0)}$  does not have to be necessarily binary, but can be an arbitrary point of the unit cube. Second, the preference method  $\mathbb{P}$  will specify the subset  $\mathbf{S} \subseteq \mathbf{V}$  of preferred indices, from which the particular candidate selection method  $\mathbb{C}$  (our third parameter) chooses an index for IMPROVE.

#### ALGORITHM( $\mathbb{I}, \mathbb{P}, \mathbb{C}$ )

**Input:** A quadratic pseudo-Boolean function  $f$ , given by (1.5).

**Initialization:** Choose an initial vector  $\mathbf{x}^{(0)} \in \mathbb{U}^n$  by method  $\mathbb{I}$ , and set  $k = 0$  and  $\mathbf{H} = \emptyset$ .

**Step 1:** Choose a subset  $\mathbf{S} \subseteq \mathbf{V}$  by preference method  $\mathbb{P}$ . If  $\mathbf{S} = \emptyset$ , then STOP, and finish with **Output**.

**Step 2:** Otherwise, choose an index  $i_k \in \mathbf{S}$  by method  $\mathbb{C}$ .

**Step 3:** Set  $\mathbf{x}^{(k+1)} = \text{IMPROVE}(f, \mathbf{x}^{(k)}, i_k)$ , and  $\mathbf{H} = \mathbf{H} \cup \{i_k\}$ .

**Step 4:** Set  $k = k + 1$ , and return to **Step 1**.

**Output:** Vector  $\mathbf{x}^{(k)}$ .

The detailed description of the parameters selection methods will be given in the next subsections.

### Initialization

The description of four starting points ( $I_1$ ,  $I_2$ ,  $I_3$  and  $I_4$ ) considered in the local search heuristics studied is given in Table 6.2. We also consider the starting point  $I_6$  (Random), i.e.  $\mathbf{x}^{(0)} = (\xi_1, \dots, \xi_n)$ , where  $\xi_i$ ,  $i = 1, \dots, n$  are independent, random variables, uniformly distributed in  $\mathbb{U}$ .

### Candidate Selection

In order to increase the value of the function  $f$ , by changing the value of a variable, the selection of that variable has to balance computational time and gain in function value.

In what follows, we shall use the quantities  $d_i(\mathbf{p})$  previously defined in (6.10), which measure the size of local improvement when changing only one component (optimally) of point  $(\mathbf{p})$ .

Let us associate to a given function  $f$  and vector  $\mathbf{p}$  the set

$$\mathbf{I}(f, \mathbf{p}) = \{i \mid d_i(\mathbf{p}) > 0\}$$

containing all indices at which we could decrease the value of the function by a local improvement step. When determining the “pool” of candidate variables to which the IMPROVE routine will be applied we shall consider two basic alternatives (see Table 6.9). In the first one, the pool will consist simply of  $\mathbf{I}(f, \mathbf{p})$ . In the second one, the pool will be restricted to those elements of  $\mathbf{I}(f, \mathbf{p})$  to which the IMPROVE step has not yet been applied. In Table 6.9,  $\mathbf{H}$  stands for the set of those variables on which transformations were carried out in previous steps.

Once the pool  $\mathbf{S}$  is defined, we have to define the criterion which will determine the choice of the variable to serve as a “pivot” to which the IMPROVE routine is applied. This decision is based on the one hand on the expected improvement in function value,

Table 6.9: List of target sets considered in the computational experiments.

Pool of Candidate Variables $\mathbb{P}$
<b>P<sub>1</sub></b> : $\mathbf{S} = \mathbf{I}(f, \mathbf{p})$ .
<b>P<sub>2</sub></b> : $\mathbf{S} = \mathbf{I}(f, \mathbf{p}) \cap (\mathbf{V} \setminus \mathbf{H})$ if $\mathbf{H} \neq \mathbf{V}$ , and $\mathbf{S} = \mathbf{I}(f, \mathbf{p})$ otherwise.

and on the other hand on the computational effort of applying this selection step. The four criteria examined are shown in Table 6.10.

Table 6.10: Criteria list of pivot selection considered in the computational experiments.

Alternatives $\mathbb{C}$ for Pivot Selection
<b>C<sub>1</sub></b> : Choose the smallest index $i$ such that $d_i(\mathbf{p}) = \max_{j \in \mathbf{S}} d_j(\mathbf{p})$ .
<b>C<sub>2</sub></b> : Choose the smallest index in $\mathbf{I}(f, \mathbf{p}) \cap \mathbf{S}$ .
<b>C<sub>3</sub></b> : Let $j$ be the index of the last pivot, and let $\mathbf{S}^{j+} = \{i \in \mathbf{S} \mid i > j\}$ and $\mathbf{S}^{j-} = \{i \in \mathbf{S} \mid i < j\}$ . If $\mathbf{I}(f, \mathbf{p}) \cap \mathbf{S}^{j+} \neq \emptyset$ choose the smallest index in this set. Otherwise, choose the smallest index in $\mathbf{I}(f, \mathbf{p}) \cap \mathbf{S}^{j-}$ .
<b>C<sub>4</sub></b> : Choose randomly an index in $\mathbf{I}(f, \mathbf{p}) \cap \mathbf{S}$ .

### Algorithm Specification

The combinations of the five initialization alternatives  $\mathbb{I}$ , the two variable pool selection alternatives  $\mathbb{P}$  (see Table 6.9), and the four variable selection alternatives  $\mathbb{C}$  (see Table 6.10), define 40 different variants of  $\text{ALGORITHM}(\mathbb{I}, \mathbb{P}, \mathbb{C})$ . We shall label these variants by  $A_{i,p,c}$ , where  $i \in \{1, 2, 3, 4, 6\}$ ,  $p = 1, 2$  and  $c \in \{1, \dots, 4\}$ .

#### 6.2.1.3 Implementation details

In each iteration of a given heuristic, a variable has to be selected from the pool of candidates, and then assigned the value 0 or 1. To make this procedure efficient, we used a *list* structure to associate variables with their occurrences. The adoption of this

data structure implies that the list of quadratic terms having both a nonzero coefficient and the corresponding variable, can be obtained in constant time. Further, we hold the values of the first derivative functions  $\Delta_i$  for  $i = 1, \dots, n$  at the current vector  $\mathbf{x}^t$ .

Let us analyze the complexity of the steps of  $\text{ALGORITHM}(\mathbb{I}, \mathbb{P}, \mathbb{C})$ :

**Initialization:** The initialization time is highly dependent on the method  $\mathbb{I}$  adopted; cases  $I_1$  and  $I_6$  are computed in  $O(n)$  time, and cases  $I_2, I_3$  and  $I_4$  are calculated in  $O(\text{size}(f))$ .

**Step 1:** The preference methods  $\mathbb{P}$  analyzed in this paper consider pools of variables that are subsets of  $\mathbf{I}(f, \mathbf{p})$ . Since we kept the values of  $\Delta_i$  for  $i = 1, \dots, n$  at the current vector  $\mathbf{x}^t$ , then by using Lemma 6.4, the set  $\mathbf{I}(f, \mathbf{p})$  can be defined in  $O(n)$  time.

**Step 2:** The computing time of the four criteria methods  $\mathbb{C}$  described in Table 6.10, depends on the size of  $\mathbf{I}(f, \mathbf{p})$ , implying that at most  $O(n)$  time is needed in this step.

**Step 3:** Because the first derivative values are held,  $\text{IMPROVE}$  takes  $O(1)$  time. The values of  $\Delta_j$  for  $j = 1, \dots, n$  are updated in this step as follows:

$$\Delta_j(\mathbf{x}^{(t+1)}) = \begin{cases} \Delta_j(\mathbf{x}^{(t)}) + (x_{i^t}^{(t+1)} - x_{i^t}^{(t)}) c_{i^t j}, & j > i^t \\ \Delta_j(\mathbf{x}^{(t)}), & j = i^t \\ \Delta_j(\mathbf{x}^{(t)}) + (x_{i^t}^{(t+1)} - x_{i^t}^{(t)}) c_{j i^t}, & j < i^t \end{cases}$$

Clearly, this operation takes at most  $O(\omega_{i^t})$  time per variable, implying that this step takes at most  $O(n)$  time.

As a consequence of the previous computing times, each iteration in the loop of  $\text{ALGORITHM}(\mathbb{I}, \mathbb{P}, \mathbb{C})$  takes at most  $O(n)$  time.

The heuristics were implemented in C++, compiled using the Microsoft Windows 32-bit C/C++ Optimizing Compiler (version 12) for 80x86, and linked with the Microsoft Incremental Linker (version 6). The computer used for testing has a Xeon(TM)

CPU 3.06 GHz, 3.5 GB of RAM and has installed the Windows XP Professional (version 2002) operating system.

### 6.2.2 Algorithm selection

The aim of this section is to analyze the relative efficiency of the 40 variants of the methods described in the previous section.

The different variants of the proposed algorithms have been tested on 125 publicly available benchmark problems, and on additional 4900 randomly generated test problems having prescribed parameters.

The 125 benchmarks are described in Section 3.1.1 and include the Glover et al. [108] and the Beasley [37] problems.

2900 of the 4900 randomly generated problems correspond to the *Medium* family described in Section 3.1.2 (see also Table 3.4). The remaining 2000 problems were generated using the same characteristics as those exhibited by the *Medium* family problems, however covering different values of  $\bar{\rho}$  ranging from 0.6 to 0.98. This will allow us to test the proposed heuristics on problems that cover the full spectrum of  $\rho$  values, which vary between 0 and 1.

Coincidentally all QUBOs above are maximization problems. Any QUBO maximization problem represented as a multilinear polynomial of a quadratic pseudo-Boolean function  $f$  can be brought to a QUBO minimization problem just by considering the symmetric function  $-f$ .

We shall present in the sequel the results of the computational testing of variants of initial point selection, alternatives for selecting the pool of candidate variables, and for choosing the pivot.

In order to evaluate the performance of the proposed algorithms, it will be useful to compare the value of the quadratic pseudo-Boolean function in the solution produced by one of these algorithms with the best value found by existing algorithms.

Since the performance of the *Multi-Start Tabu Search* (MSTS) routine of Palubeckis (downloaded from [183]) was found to deliver on the available benchmark problems at

least as good solutions as those reported in the literature (e.g., see [187]), we have used this value as standard for comparisons. For instance, this routine delivered for some of the sub-families of problems ( $F_2$  and  $G_2$ ) better values, previously not known (e.g., compare with the values in [107]).

Let us denote by  $z_f$  the *best known value* of the quadratic function  $f$ . The list of all best values and the corresponding 0–1 solutions for the test problems used in this study can be found in [229].

Let  $i$  denote one of the five possible initialization methods proposed in Table 6.2. Let  $p$  denote one of the two pools of candidates for pivots proposed in Table 6.9. Similarly, let  $c$  denote one of the four criteria proposed in Table 6.10 for choosing a pivot. The output of algorithm  $A_{i,p,c}$  when applied to the quadratic pseudo-Boolean function  $f$  will be denoted by  $x_{i,p,c,f}$ , and the value of the function  $f$  in this point will be denoted by  $v_{i,p,c,f}$ . We shall denote by  $r_{i,p,c,f}$  the *performance ratio* of the heuristic over the best known value

$$r_{i,p,c,f} \stackrel{\text{def}}{=} \frac{v_{i,p,c,f}}{z_f}.$$

We shall denote the computing time of applying  $A_{i,p,c}$  using a specific computer system  $T$  by  $t_{i,p,c,f;T}$ ; since in this work all the different algorithms will be run on the same computer system (see Subsection 6.2.1.3), for the sake of simplicity we shall omit to specify  $T$  for each test problem and shall denote the running times by  $t_{i,p,c,f}$ .

If  $\mathcal{F}$  is a family of quadratic pseudo-Boolean functions used for algorithm testing, we shall denote by  $V_{i,p,c;\mathcal{F}}$  the set of values obtained by applying the algorithm  $A_{i,p,c}$  to all the test problems  $f \in \mathcal{F}$ . Similarly,  $V_{I,p,c;\mathcal{F}}$  will represent the set of values obtained by applying the algorithms  $A_{I,p,c}$  for every  $i \in I$  to every  $f \in \mathcal{F}$ . The notations  $V_{I,p,C;\mathcal{F}}$ ,  $V_{i,P,c;\mathcal{F}}$ , etc. have similar interpretations. We shall also denote by  $T_{I,p,C;\mathcal{F}}$  and  $R_{I,p,C;\mathcal{F}}$ , etc. the sets of computing times, respectively heuristic over best known values, for the corresponding sets.

Several of the tables below present the statistics of the computational experiments for some outcome  $w_{i,p,c;\mathcal{F}}$ , where  $w$  can stand for value  $v$ , or the ratio  $r$  of heuristic over best known value, or computing time  $t$ . In these tables (see Figure 6.4) four data are

grouped together in a cell. The number in the top left corner represents  $\min W_{I,P,C;\mathcal{F}}$ . The number in the lower right corner represents  $\max W_{I,P,C;\mathcal{F}}$ . The center of the cell involves the expected values  $\text{Exp} [(W_{I,P,C;\mathcal{F}})]$  and standard deviations  $\sigma (W_{I,P,C;\mathcal{F}})$ .

$\min W_{I,P,C;\mathcal{F}}$	
$\text{Exp} (W_{I,P,C;\mathcal{F}}) \pm \sigma (W_{I,P,C;\mathcal{F}})$	
	$\max W_{I,P,C;\mathcal{F}}$

Figure 6.4: Description of the details shown in a cell of a cross-analysis table.

We have applied all the 40 alternatives of the proposed heuristics (corresponding to the five initialization procedures, two choices of the pool and four possible selection criteria of the pivot) to the solution of each one of the 5025 test problems in  $\mathcal{S}$ . The analysis and selection process described below is based on these 201 000 experiments.

### Solution Quality Analysis

Tables 6.11(a) and 6.11(b) below report statistics concerning the sets  $R_{i,p,c;\text{benchmarks}}$  and  $R_{i,p,c;\text{random tests}}$  of performance ratios respectively, for the different procedures  $i \in \{1, 2, 3, 4, 6\}$ ,  $p = 1, 2$ , and  $c \in \{1, \dots, 4\}$ .

It can be seen that the best average values both for benchmark and for random problems are obtained for  $c = 1$ . It can also be seen that the algorithm  $A_{2,1,1}$  is optimal for both families of problems, and that  $A_{2,2,1}$  and  $A_{4,1,1}$  are two other excellent candidates in both cases.

If the algorithm's selection criterion is not based on average performance but on high "worst case" performance (i.e., on assuring the highest minimum performance ratio) both for benchmark and random problems, then perhaps the best algorithm is  $A_{3,1,1}$ .

Finally if the algorithm's selection criterion aims at minimizing variance then again  $A_{2,1,1}$  is the best choice for both families of problems.

The minimum performance ratios for the algorithm  $A_{2,2,1}$  in the case of benchmark problems are comparable with those of  $A_{2,1,1}$ , but are of somewhat lower quality in the case of random problems. Therefore,  $A_{2,2,1}$  will not be included in our selection of best heuristics.

Table 6.11: Performance ratio sets  $R_{i,p,c;\mathcal{F}}$  for algorithms  $A_{I,P,C}$ .

(a)  $\mathcal{F}$ =benchmarks.

	$P_1$				$P_2$			
	$C_1$	$C_2$	$C_3$	$C_4$	$C_1$	$C_2$	$C_3$	$C_4$
$I_1$	47.5% 95.8 ± 9.8% 100.0%	44.1% 94.3 ± 11.3% 100.0%	33.8% 93.5 ± 13.0% 100.0%	25.5% 93.7 ± 12.6% 100.0%	47.5% 95.7 ± 9.8% 100.0%	33.8% 93.1 ± 13.7% 100.0%	33.8% 93.5 ± 13.0% 100.0%	33.8% 93.1 ± 13.5% 100.0%
$I_6$	58.9% 95.7 ± 7.6% 100.0%	58.6% 95.1 ± 9.2% 100.0%	39.7% 93.1 ± 13.0% 100.0%	44.9% 94.2 ± 10.2% 100.0%	58.9% 95.1 ± 8.9% 100.0%	39.7% 92.9 ± 13.5% 100.0%	39.7% 93.1 ± 13.0% 100.0%	41.3% 92.6 ± 13.6% 100.0%
$I_2$	76.6% 97.7 ± 5.0% 100.0%	70.5% 96.0 ± 6.9% 100.0%	46.6% 95.2 ± 8.5% 100.0%	66.9% 95.5 ± 7.5% 100.0%	76.6% 97.7 ± 5.0% 100.0%	46.6% 95.3 ± 8.6% 100.0%	46.6% 95.2 ± 8.5% 100.0%	67.4% 95.8 ± 7.0% 100.0%
$I_3$	77.3% 96.4 ± 5.3% 100.0%	61.6% 95.1 ± 8.2% 100.0%	61.6% 94.9 ± 8.3% 100.0%	46.3% 93.9 ± 9.9% 100.0%	77.3% 96.4 ± 5.3% 100.0%	61.6% 95.0 ± 8.3% 100.0%	61.6% 95.0 ± 8.3% 100.0%	43.3% 93.7 ± 10.9% 100.0%
$I_4$	77.3% 97.6 ± 5.3% 100.0%	61.6% 95.9 ± 8.4% 100.0%	61.6% 95.8 ± 7.7% 100.0%	51.9% 94.8 ± 10.2% 100.0%	77.3% 97.5 ± 5.3% 100.0%	61.6% 95.9 ± 7.8% 100.0%	61.6% 95.8 ± 7.7% 100.0%	45.5% 94.6 ± 10.9% 100.0%

(b)  $\mathcal{F}$ =random tests.

	$P_1$				$P_2$			
	$C_1$	$C_2$	$C_3$	$C_4$	$C_1$	$C_2$	$C_3$	$C_4$
$I_1$	27.4% 88.4 ± 16.0% 100.0%	24.6% 86.2 ± 18.1% 100.0%	10.3% 83.0 ± 21.2% 100.0%	20.2% 84.0 ± 20.1% 100.0%	26.8% 87.6 ± 16.8% 100.0%	10.3% 83.7 ± 20.8% 100.0%	10.3% 83.0 ± 21.2% 100.0%	12.4% 83.2 ± 21.0% 100.0%
$I_6$	36.3% 88.1 ± 14.9% 100.0%	24.1% 85.9 ± 18.3% 100.0%	10.3% 82.9 ± 21.3% 100.0%	16.6% 83.7 ± 20.3% 100.0%	25.3% 85.6 ± 18.3% 100.0%	10.3% 83.6 ± 20.9% 100.0%	10.3% 82.9 ± 21.3% 100.0%	15.3% 83.0 ± 21.3% 100.0%
$I_2$	38.3% 89.9 ± 13.5% 100.0%	28.3% 87.5 ± 15.9% 100.0%	15.5% 83.0 ± 21.2% 100.0%	20.4% 84.3 ± 19.3% 100.0%	33.1% 88.5 ± 15.3% 100.0%	15.5% 83.7 ± 20.8% 100.0%	15.5% 83.0 ± 21.2% 100.0%	11.0% 83.1 ± 21.1% 100.0%
$I_3$	43.1% 86.9 ± 15.1% 100.0%	16.4% 83.6 ± 20.9% 100.0%	13.9% 82.8 ± 21.3% 100.0%	18.1% 83.0 ± 21.0% 100.0%	43.1% 86.9 ± 15.1% 100.0%	16.4% 83.3 ± 21.2% 100.0%	13.9% 82.8 ± 21.3% 100.0%	12.0% 82.8 ± 21.3% 100.0%
$I_4$	38.4% 88.7 ± 15.3% 100.0%	28.3% 87.7 ± 15.8% 100.0%	15.5% 83.1 ± 21.2% 100.0%	22.9% 84.6 ± 19.0% 100.0%	24.3% 87.0 ± 17.9% 100.0%	15.5% 83.9 ± 20.7% 100.0%	15.5% 83.1 ± 21.2% 100.0%	17.1% 83.1 ± 21.2% 100.0%

### Computing Time Analysis

Tables 6.12(a) and 6.12(b) show average computing times in a manner similar to the way Tables 6.11(a) and 6.11(b) show average solution qualities. It can be seen that for the benchmark problems the average computing times range from 0.03 to 0.06 seconds, while for the random problems the times vary between 0.21 and 0.38 seconds. It seems to us that the differences between the average computing times given by the different heuristics considered are minimal. Therefore, the final conclusion will be essentially based on the quality of solutions.

### Selected Algorithms

Since the algorithms  $A_{I,1,1}$ , where  $I = \{2, 3, 4\}$  were seen to provide both outstanding solution quality and computing time, we shall restrict from here on our attention to these algorithms. In conclusion the selected algorithms

- (i) select as initial point, one identified by any of the three problem-dependent criteria 2, 3 or 4, and
- (ii) use as pivot the first (i.e., smallest index) variable – regardless of whether it has or has not been previously used as pivot – the switching (if it is binary) or rounding (if fractional) of which gives the maximum objective function increase.

When comparing the three selected algorithms from the point of view of solution quality,  $A_{2,1,1}$  is best, followed closely by  $A_{4,1,1}$ ; from the point of view of computing time  $A_{3,1,1}$  is the leader.

### Comparative Performance Analysis

In order to narrow down further the selection of “best” algorithms, we shall reexamine the quality of solutions and the computing times of the three selected algorithms, as functions of the number of variables  $n$ , the density  $d$ , diagonal dominance  $p$ , and parameter  $\rho$  defined in subsection 6.2.1.2. By examining the relationship between the

Table 6.12: Computing times  $T_{i,p,c;\mathcal{F}}$  for algorithms  $A_{I,P,C}$  ( $I = \{1, 2, 3, 4, 6\}$ ,  $P = \{1, 2\}$ , and  $C = \{1, \dots, 4\}$ ).

(a)  $\mathcal{F}$ =benchmarks.

	$P_1$				$P_2$			
	$C_1$	$C_2$	$C_3$	$C_4$	$C_1$	$C_2$	$C_3$	$C_4$
$I_1$	$0.00s$ $0.04 \pm 0.07s$ $0.23s$	$0.00s$ $0.04 \pm 0.09s$ $0.33s$	$0.00s$ $0.03 \pm 0.05s$ $0.20s$	$0.00s$ $0.05 \pm 0.10s$ $0.34s$	$0.00s$ $0.03 \pm 0.06s$ $0.22s$	$0.00s$ $0.03 \pm 0.06s$ $0.23s$	$0.00s$ $0.03 \pm 0.05s$ $0.20s$	$0.00s$ $0.05 \pm 0.09s$ $0.30s$
$I_6$	$0.00s$ $0.04 \pm 0.07s$ $0.23s$	$0.00s$ $0.05 \pm 0.09s$ $0.34s$	$0.00s$ $0.03 \pm 0.05s$ $0.20s$	$0.00s$ $0.05 \pm 0.10s$ $0.36s$	$0.00s$ $0.03 \pm 0.06s$ $0.22s$	$0.00s$ $0.03 \pm 0.06s$ $0.23s$	$0.00s$ $0.03 \pm 0.05s$ $0.20s$	$0.00s$ $0.05 \pm 0.09s$ $0.31s$
$I_2$	$0.00s$ $0.05 \pm 0.09s$ $0.33s$	$0.00s$ $0.05 \pm 0.11s$ $0.44s$	$0.00s$ $0.04 \pm 0.07s$ $0.28s$	$0.00s$ $0.06 \pm 0.12s$ $0.41s$	$0.00s$ $0.04 \pm 0.08s$ $0.28s$	$0.00s$ $0.04 \pm 0.08s$ $0.33s$	$0.00s$ $0.04 \pm 0.07s$ $0.27s$	$0.00s$ $0.06 \pm 0.11s$ $0.36s$
$I_3$	$0.00s$ $0.03 \pm 0.06s$ $0.28s$	$0.00s$ $0.05 \pm 0.09s$ $0.39s$	$0.00s$ $0.03 \pm 0.06s$ $0.23s$	$0.00s$ $0.04 \pm 0.08s$ $0.28s$	$0.00s$ $0.03 \pm 0.06s$ $0.27s$	$0.00s$ $0.03 \pm 0.07s$ $0.27s$	$0.00s$ $0.03 \pm 0.06s$ $0.24s$	$0.00s$ $0.04 \pm 0.07s$ $0.28s$
$I_4$	$0.00s$ $0.05 \pm 0.09s$ $0.31s$	$0.00s$ $0.05 \pm 0.10s$ $0.39s$	$0.00s$ $0.04 \pm 0.07s$ $0.30s$	$0.00s$ $0.06 \pm 0.12s$ $0.41s$	$0.00s$ $0.04 \pm 0.08s$ $0.30s$	$0.00s$ $0.04 \pm 0.07s$ $0.31s$	$0.00s$ $0.04 \pm 0.07s$ $0.28s$	$0.00s$ $0.06 \pm 0.11s$ $0.36s$

(b)  $\mathcal{F}$ =random tests.

	$P_1$				$P_2$			
	$C_1$	$C_2$	$C_3$	$C_4$	$C_1$	$C_2$	$C_3$	$C_4$
$I_1$	$0.00s$ $0.25 \pm 0.25s$ $1.06s$	$0.00s$ $0.24 \pm 0.26s$ $1.76s$	$0.00s$ $0.21 \pm 0.22s$ $1.00s$	$0.00s$ $0.28 \pm 0.28s$ $1.34s$	$0.00s$ $0.25 \pm 0.25s$ $1.03s$	$0.00s$ $0.22 \pm 0.23s$ $1.36s$	$0.00s$ $0.21 \pm 0.22s$ $1.00s$	$0.00s$ $0.28 \pm 0.28s$ $1.30s$
$I_6$	$0.00s$ $0.25 \pm 0.25s$ $1.05s$	$0.00s$ $0.25 \pm 0.26s$ $1.94s$	$0.00s$ $0.21 \pm 0.22s$ $0.99s$	$0.00s$ $0.29 \pm 0.28s$ $1.38s$	$0.00s$ $0.25 \pm 0.25s$ $1.05s$	$0.00s$ $0.22 \pm 0.23s$ $1.33s$	$0.00s$ $0.21 \pm 0.22s$ $1.02s$	$0.00s$ $0.28 \pm 0.28s$ $2.00s$
$I_2$	$0.00s$ $0.35 \pm 0.35s$ $1.47s$	$0.00s$ $0.33 \pm 0.35s$ $2.17s$	$0.00s$ $0.30 \pm 0.31s$ $1.36s$	$0.00s$ $0.38 \pm 0.37s$ $1.69s$	$0.00s$ $0.34 \pm 0.35s$ $1.48s$	$0.00s$ $0.31 \pm 0.32s$ $1.73s$	$0.00s$ $0.30 \pm 0.31s$ $1.36s$	$0.00s$ $0.38 \pm 0.37s$ $1.67s$
$I_3$	$0.00s$ $0.23 \pm 0.25s$ $1.38s$	$0.00s$ $0.24 \pm 0.28s$ $2.80s$	$0.00s$ $0.22 \pm 0.24s$ $1.38s$	$0.00s$ $0.24 \pm 0.27s$ $1.70s$	$0.00s$ $0.23 \pm 0.25s$ $1.39s$	$0.00s$ $0.23 \pm 0.25s$ $1.58s$	$0.00s$ $0.22 \pm 0.24s$ $1.39s$	$0.00s$ $0.24 \pm 0.27s$ $1.97s$
$I_4$	$0.00s$ $0.35 \pm 0.35s$ $1.45s$	$0.00s$ $0.34 \pm 0.35s$ $2.15s$	$0.00s$ $0.30 \pm 0.32s$ $1.55s$	$0.00s$ $0.38 \pm 0.37s$ $1.70s$	$0.00s$ $0.34 \pm 0.35s$ $1.50s$	$0.00s$ $0.31 \pm 0.33s$ $1.70s$	$0.00s$ $0.31 \pm 0.32s$ $1.51s$	$0.00s$ $0.38 \pm 0.38s$ $2.19s$

Table 6.13: Correlations between quality of solutions ( $r_{i,1,1}$ ,  $i = 2, 3, 4$ ), computing times ( $t_{i,1,1}$ ,  $i = 2, 3, 4$ ) and input parameters ( $n$ ,  $d$ ,  $\rho$  and  $p$ ) of the test problems in  $\mathcal{S}$ .

	$r_{2,1,1}$	$r_{3,1,1}$	$r_{4,1,1}$	$t_{2,1,1}$	$t_{3,1,1}$	$t_{4,1,1}$
$n$	-0.08	-0.04	-0.07	0.77	0.71	0.77
$d$	-0.10	-0.08	-0.09	0.47	0.44	0.47
$\rho$	0.84	0.86	0.84	0.00	-0.12	0.00
$p$	0.13	0.13	0.13	-0.18	-0.17	-0.18

variables  $r$  and  $t$ , and the variables  $n$ ,  $d$ ,  $\rho$  and  $p$ , on the dataset  $\mathcal{S}$ , we find the correlations of Table 6.13.

Clearly, the most influential factor for the solution quality  $r$  is the parameter  $\rho$ , while  $n$  and  $d$  are the most influential factors for the computing time  $t$ . It is also clear that diagonal dominance  $p$  is less influential (both on solution quality and time) than the other three parameters. Finally, the negative role of  $n$  and  $d$  on solution quality, which is to be expected, can be seen to be present in the table, but at a very low level. We can also remark that the influence of  $\rho$  on time is negligible.

Since  $\rho$  emerges as the critical parameter on which the solution quality parameter depends, we have analyzed the behavior of the 3 functions  $\text{Exp}[r_{i,1,1}]$  ( $i \in \{2, 3, 4\}$ ), as functions of  $\rho$ .

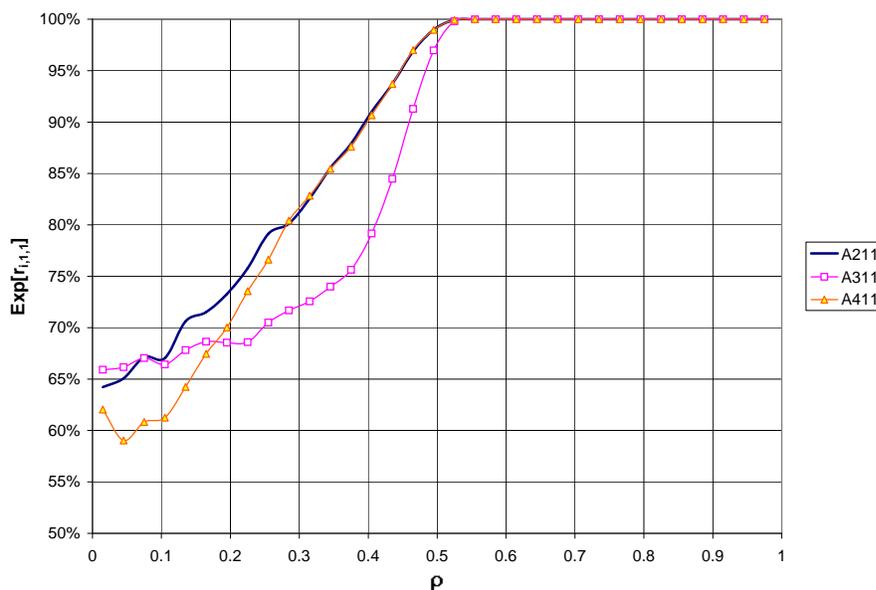


Figure 6.5:  $\text{Exp}[r_{i,1,1}]$ ,  $i \in \{2, 3, 4\}$ , values as a function of  $\rho$  in the random test problems.

It can be seen from the graphs of these three functions (Figure 6.5) that the algorithms giving on the average the highest quality solutions depend on the values of  $\rho$  as shown in Table 6.14. In particular it can be seen that algorithm  $A_{2,1,1}$  is an optimal one for any value of  $\rho \geq 0.075$ . Therefore, the algorithm starting from the point  $X = (\rho, \dots, \rho)$ , and performing a sequence of rounding or switching operations on a greedy selected pivot (regardless of whether it has or it has not been used as a pivot) provides a solution with the highest expected value.

Table 6.14: Algorithms returning on average the highest value for the test problems, according to the  $\rho$  parameter.

Values of $\rho$	"Best" Algorithms
$\rho < 0.075$	$A_{3,1,1}$
$0.075 \leq \rho < 0.275$	$A_{2,1,1}$
$0.275 \geq \rho < 0.525$	$A_{2,1,1}, A_{4,1,1}$
$0.525 \geq \rho$	$A_{2,1,1}, A_{3,1,1}, A_{4,1,1}$

The rest of this paper will deal with this algorithm. For a better identification and future reference, we named the algorithm  $A_{2,1,1}$  as the ACcelarated Sign Initiated Open Minded heuristic (or ACSIOM in short). In sections that follow, the performance ratio  $r$  and computing time  $t$  of heuristic ACSIOM is denoted as  $r_{ACSIOM}$  and  $t_{ACSIOM}$  respectively.

### 6.2.3 Parametric analysis of heuristic ACSIOM

The goal of this section is to analyze the computing time  $t$  of the proposed heuristic ACSIOM (i.e.  $A_{2,1,1}$ ) and the quality  $r$  of solutions provided by it, in terms of the input parameters  $n$ ,  $d$ ,  $\rho$  and  $p$  describing the datasets. The proposed heuristic was tested on a set  $\mathcal{T}$  of 8905 instances consisting of the following four groups of problems:

- (i) The 125 benchmark problems described in the previous section.
- (ii) The 2900 randomly generated problems corresponding to the *Medium* family, and described in Section 3.1.2 (see also Table 3.4). The reason for not including the random test problems with  $\rho > 0.6$  considered in Section 6.2.2 is that these cases have an excessively favorable behavior when ACSIOM is applied to them (their

heuristic values being the same as those given by the best known solutions in all but one of the 2 000 cases).

- (iii) The group of 5 400 randomly generated problems corresponding to the *Small* family, and described in Section 3.1.2 (see also Table 3.3). Because of the relatively small size of these problems it can be assumed that the best known solution values are actually the exact optimum of the problem.
- (iv) The group of 480 randomly generated problems corresponding to the *Large* family, and also described in Section 3.1.2 (see also Table 3.5). All the 160 test problems in the second subgroup turn out to be *submodular*; using a depth first search enumerative procedure, we were able to find the optimal solution of 60 test problems in this group. It is interesting to notice that the expected values of  $\rho$  in the three subgroups are respectively 0.5, 0.0 and 0.2.

### 6.2.3.1 Quality of solutions

The correlation analysis aimed at clarifying the influence of various input parameters of Section 6.2.2, included test problems with various values of  $\rho$ , in particular with values exceeding 0.6. In this subsection we shall analyze the role of the input parameters using the 8 905 test problems in  $\mathcal{T}$  described above.

If we compare the correlation results in Table 6.13 (referring to the test set  $\mathcal{S}$ ), with those in Table 6.15 (referring to the test set  $\mathcal{T}$ ), one can see that the values are similar, except for the correlation between the number  $n$  of variables and the performance ratio  $r_{ACSIOM}$ , which decreased from  $-0.08$  to  $-0.53$ . There are two explanations for this substantial change. On the one hand the datasets considered were restricted to problems with  $\rho \leq 0.6$ , while on the other the hand the variance of the number of variables was considerably higher than in the first one.

Figure 6.6 presents the number of problems for which the value provided by ACSIOM is within a given fraction of the best known solution. It can be seen that for 90% of the problems in  $\mathcal{S}$  the corresponding heuristic value is within 26% of the best known solution. More precisely:

Table 6.15: Correlations between quality of solutions ( $r_{ACSIOM}$ ), computing times ( $t_{ACSIOM}$ ) and input parameters ( $n$ ,  $d$ ,  $\rho$  and  $p$ ) of the test problems in  $\mathcal{T}$ .

	$r_{ACSIOM}$	$t_{ACSIOM}$
$n$	-0.53	0.73
$d$	-0.12	0.15
$\rho$	0.89	-0.25
$p$	0.20	-0.09

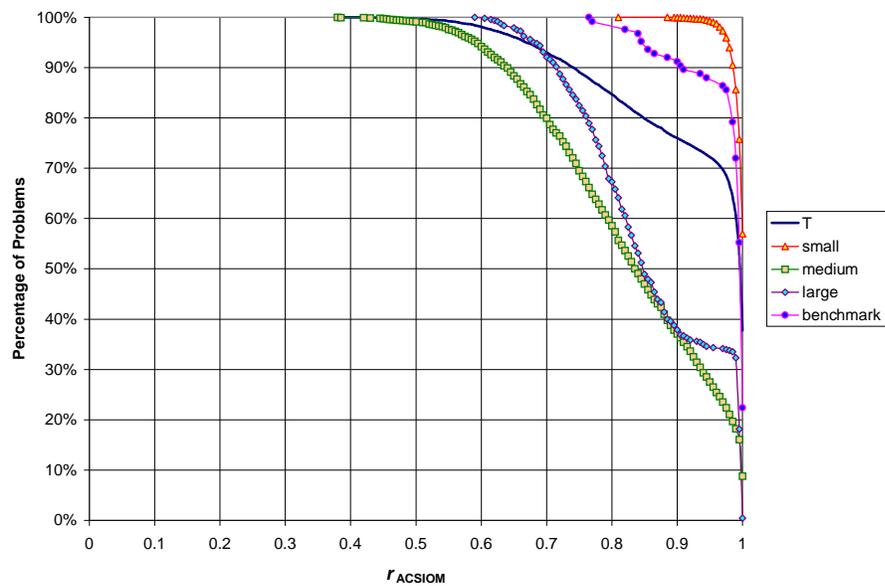


Figure 6.6: Distribution of problems according to  $r_{ACSIOM}$ .

- (i) For the *benchmark* problems (group (i)) the performance of ACSIOM is within 10% of the best known solution in 90% of the cases.
- (ii) For the *Medium* problems (group (ii)) the performance of ACSIOM is within 36% of the best known solution in 90% of the cases.
- (iii) For the *Small* problems (group (iii)) the performance of ACSIOM is within 3% of the best known (probably optimal) solution in 96% of the cases.
- (iv) For the *Large* problems (group (iv)) the performance of ACSIOM is within 28.5% of the best known solution in 90% of the cases.

It is interesting to remark that relatively speaking, the performance of ACSIOM in the class of *benchmark* problems was much better than in the class of randomly generated *Medium* and *Large* problems. These results are indicative of the fact that the study of heuristics of QUBO needs to include problems with different characteristics than the ones exhibited by the *benchmark* datasets. In the family of *Small* test problems, algorithm ACSIOM returns on average a value within 1% of the best known solution, for every combination of the number of variables (from 20 to 100 in steps of 10), densities (from 10% to 100% in steps of 10%), and magnitudes of quadratic coefficients (see Table 3.3).

The family of *Medium* size problems was partitioned into four subsets according to the values of  $\rho$ :

$$Medium = Medium_{\rho \leq 0.15} \cup Medium_{0.15 < \rho \leq 0.3} \cup Medium_{0.3 < \rho \leq 0.45} \cup Medium_{0.45 < \rho \leq 0.6}. \quad (6.13)$$

The analysis of the performance ratio  $r_{ACSIOM}$  as a function of the number of variables  $n$  and density  $d$  is reported in Table B.2 of the Appendix. Each cell of these tables refers to a group with between 35 and 40 test problems.

It can be seen in the tables that the performance ratio is highly dependent on the parameter  $\rho$ , confirming the conclusions of Subsection 6.2.2 of this paper.

Algorithm ACSIOM performs considerably better for low density problems ( $d = 20\%$ ) than for high density problems ( $d = 100\%$ ), this difference being substantially reduced when the parameter  $\rho$  increases. This is important since the type of problems in numerous applications (e.g., Ising model ([32]), maximum clique in sparse graphs) the quadratic model is of low density. In general, the ACSIOM heuristic decreases the performance ratios whenever the density increases.

The algorithm performs considerably better in problems with 500 variables than in problems with 2000 variables, this difference being substantially reduced as the parameter  $\rho$  increases. In general, the ACSIOM heuristic decreases the performance ratios whenever the number of variables is increased.

Figure 6.5 of Section 6.2.2 and the previous analysis suggest that the performance ratio of  $r_{ACSIOM}$  has a good chance to be well approximated by using some regression estimator, depending on the input parameters.

If  $\rho > 0.525$  then  $r_{ACSIOM}$  is very close to 100%, showing that the heuristic value is independent of the number of variables or density for this class of problems.

By using a least squares fitting, several linear regressions of  $r_{ACSIOM}$  were computed, the independent variables being:  $n$ ,  $d$ ,  $\rho$  and  $p$ . Given the regression population  $\mathcal{F}$ , we would like to approximate  $r_{ACSIOM}$  by using the linear estimator

$$\hat{r}_{ACSIOM} = a_0 + a_n n + a_d d + a_\rho \rho + a_p p,$$

where  $a_n$ ,  $a_d$ ,  $a_\rho$  and  $a_p$  are the coefficients of the regressors ( $n$ ,  $d$ ,  $\rho$  and  $p$ , resp.) and  $a_0$  is the intercept value of the estimator, when all the independent variables have a zero value.

The first regression was found for the family of *Medium* problems with  $\rho \leq 0.525$ . The ANOVA test shows that the model is significant at the 95% level, and the  $R^2$  value is high, but we noticed that the  $P$ -value associated with the coefficient of the diagonal dominance ( $a_p$ ) is 0.7081, this being a clear indication that  $p$  is not a reliable parameter for the model because it has “too much” dispersion/variance.

In view of the high  $P$ -value of the parameter  $p$ , it was decided to eliminate it from

the model. The new regression results are shown in Figure B.1 of the Appendix; the new model is of very high quality having a standard error of 0.0605, a  $R^2$  of 0.7880, and a correlation of 0.8877 between the estimator ( $\hat{r}_{ACSIOM}$ ) and the observed ( $r_{ACSIOM}$ ) values.

The final regression model corresponding to the heuristic ACSIOM is

$$\hat{r}_{ACSIOM} = \begin{cases} 0.68725 - 2.9132 \times 10^{-5}n - 0.079109d + 0.75261\rho, & \rho \leq 0.525 \\ 1, & \text{otherwise.} \end{cases}$$

As one can traditionally expect, both the number of variables ( $n$ ) and the density ( $d$ ) are negatively correlated with the estimator. It is to be remarked that the coefficient of  $\rho$  is positive and shows the important role played by this parameter.

### 6.2.3.2 Computational efficiency

We shall deal in this subsection with two important questions concerning the efficiency of the proposed local search algorithm ACSIOM. First, we shall analyze the computing time, and second we shall analyze the number of times the value of a variable was switched between 0 and 1, i.e. the number of times the routine IMPROVE was called for a given variable. The importance of the second question comes from the fact that the complexity of finding a local optimum of a QUBO is not known (see [194]).

#### Computing Time

We have analyzed the dependence of the computing time on three key parameters: the number of variables  $n$ , the density  $d$ , and  $\rho$  as defined in (3.1). Table 6.16 reports statistics on the computing times  $t_{ACSIOM}$  of problems in the *large* family, for various choices of  $n$  and  $d$ .

The reason for which  $\rho$  was not included in this analysis is that the difference in times for various values of  $\rho$  turned out to be minuscule (e.g., at most two tenths of a second in the problems with 5 000 variables).

It can be seen in Table 6.16 that computing times increase with increased values of  $n$  and  $d$ . As a matter of fact, it was seen that the correlation between the computing

Table 6.16: Number of variables ( $n$ ) versus density ( $d$ ) analysis on the heuristic computation times ( $t_{ACSIOM}$ ) for the *large* family of test problems.

	$d = 25\%$	$d = 50\%$	$d = 75\%$	$d = 100\%$
$n = 500$	0.01sec	0.03sec	0.05sec	0.06sec
	$0.02 \pm 0.01sec$	$0.03 \pm 0.00sec$	$0.05 \pm 0.01sec$	$0.07 \pm 0.01sec$
	0.03sec	0.05sec	0.06sec	0.08sec
$n = 1000$	0.06sec	0.14sec	0.20sec	0.26sec
	$0.08 \pm 0.01sec$	$0.14 \pm 0.01sec$	$0.22 \pm 0.04sec$	$0.28 \pm 0.01sec$
	0.09sec	0.16sec	0.42sec	0.30sec
$n = 2500$	0.50sec	1.00sec	1.59sec	2.28sec
	$0.53 \pm 0.03sec$	$1.04 \pm 0.03sec$	$1.63 \pm 0.03sec$	$2.33 \pm 0.04sec$
	0.58sec	1.13sec	1.67sec	2.44sec
$n = 5000$	2.47sec	5.24sec	8.16sec	10.98sec
	$2.58 \pm 0.12sec$	$5.40 \pm 0.11sec$	$8.33 \pm 0.20sec$	$11.18 \pm 0.17sec$
	2.83sec	5.61sec	9.09sec	11.73sec

times  $t_{ACSIOM}$  and the estimator

$$\hat{t}_{ACSIOM} = -0.044667 + 4.2410 \times 10^{-7} n^2 d,$$

is of 99.3%, while the standard error of the model is 0.1046, and its  $R^2$  is 0.9869.

The reason for selecting a model which includes a cubic term is that it provides an excellent fit, substantially better than that given by any regression involving functions of degree at most two.

### Roundings per Variable

In analyzing the number of roundings per variable in algorithm ACSIOM we have again this number to be a function of  $n$ ,  $d$  and  $\rho$ , and have carried out the computational experiments on the *Medium* family which includes a sufficient large number of examples to make its conclusions extremely relevant.

The set of test problems was partitioned in four subsets according to the parameter  $\rho$ , as shown in formula (6.13). Each of the four subsets it is analyzed according to  $n$  and  $d$ .

In Table B.3 of the Appendix, it can be seen that on the average approximately one rounding is needed for each variable to get a local optimum using algorithm ACSIOM.

It is interesting to note that for  $\rho \leq 0.3$  the number of roundings per variable

is monotonically nondecreasing on  $n$  and  $d$ . However, for  $\rho > 0.3$  this tendency is reversed. If  $0.45 < \rho \leq 0.6$  then the average number of roundings is almost constant (around 1.013).

Also interesting to notice is the fact that the set of problems with the highest average roundings per variable is  $Medium_{0.3 < \rho \leq 0.45}$  with an overall average value of 1.02849. This subset of problems is followed in this order by the subsets of problems  $Medium_{0.15 < \rho \leq 0.3}$ ,  $Medium_{0.45 < \rho \leq 0.6}$  and  $Medium_{\rho \leq 0.15}$  with respective average values of 1.01437, 1.01336 and 1.008367.

#### 6.2.4 Comparing ACSIOM to other results from the literature

In this section we compare ACSIOM to other 1-opt algorithms from the literature. The best result with this type of methods were reported recently in [178]. We ran ACSIOM on the previously set of benchmark problems, and reported the relative performance values in Table 6.17, along those from [178]. Note that this publication reported results with other, algorithmically more involved and hence more time consuming approaches ( $k$ -opt, and greedy- $k$ -opt), and we included those results as well in Table 6.17 for completeness. For details of those methods we refer the reader to [178].

Let us also remark that the computing environment used in [178] is quite different from ours. Discounting the speed difference between the computers, it seems that both implementations are similarly competitive, however a direct comparison of the running times would not be very meaningful. Therefore, we compared only the quality of the obtained solutions in Table 6.17.

The above results show that ACSIOM is producing uniformly better results than the 1-opt heuristic, and remains competitive even with the  $k$ -opt variants, which utilize the input in much more depth, and take more time.

#### 6.2.5 Conclusions

In summary, we can state that even very simple structured and time efficient local search based methods can be greatly improved by carefully analyzing and selecting the right parameters. We believe that the combination of the initial vector selection ( $I_2$ )

Table 6.17: Average relative error of local maximization heuristics within several *benchmark* sub-families.

Sub-Family ( $\mathcal{F}$ )	$1-opt^{(*)}$		$k-opt^{(*)}$		$greedy-k-opt^{(*)}$		ACSIOM		
	<i>avg</i>	<i>sdev</i>	<i>avg</i>	<i>sdev</i>	<i>avg</i>	<i>sdev</i>	$1 - E(R, \mathcal{F})$	$\sigma(R, \mathcal{F})$	$E(T, \mathcal{F})$
$GKA_a$	2.02%	0.83%	0.38%	0.30%	0.20%	0.27%	0.24%	0.41%	0.00 s
$GKA_b$	29.44%	5.31%	14.69%	4.54%	19.76%	6.03%	10.97%	8.92%	0.00 s
$GKA_c$	1.21%	0.78%	0.24%	0.24%	0.19%	0.14%	0.14%	0.18%	0.00 s
$GKA_d$	2.71%	0.73%	0.71%	0.35%	0.42%	0.27%	0.23%	0.45%	0.00 s
$GKA_e$	1.99%	0.96%	0.50%	0.31%	0.31%	0.15%	0.33%	0.28%	0.00 s
$F_1$	1.95%	0.36%	0.56%	0.09%	0.31%	0.10%	0.66%	0.48%	0.04 s
ORL 50	5.20%	3.57%	0.89%	0.82%	0.55%	0.50%	0.41%	0.70%	0.00 s
ORL 100	3.02%	1.54%	0.65%	0.46%	0.49%	0.56%	0.58%	0.94%	0.00 s
ORL 250	2.44%	1.12%	0.65%	0.45%	0.41%	0.24%	0.51%	0.45%	0.00 s
ORL 500	2.12%	0.48%	0.62%	0.23%	0.48%	0.18%	0.57%	0.30%	0.01 s
ORL 1000	1.71%	0.24%	0.54%	0.12%	0.39%	0.08%	0.41%	0.26%	0.05 s
ORL 2500	1.15%	0.13%	0.40%	0.07%	0.29%	0.07%	0.35%	0.13%	0.28 s

<sup>(\*)</sup>[178]

(which is a kind of surprise for us), and the greediness in choosing the next variable ( $P_1C_1$ ) together contributed to the good performance of ACSIOM. Let us remark that the choice of the initial point of course influences any local search method greatly (e.g., we could start from an optimal solution). However, it is a surprise for us that such a simple approach, derived from the input in linear time could make a characteristic difference, when compared to other linear time computable initial vectors.

The obtained results demonstrate that ACSIOM could effectively be used as a stand-alone solver for very large sized problems, or also in an ensemble of heuristics. We also plan to explore its use in an exact solver, where faster and better quality heuristic solutions can help to trim the size of the search tree.

### 6.3 One-pass heuristics enhancement by local-search

Given a  $n$ -binary vector  $\mathbf{x}$ , we denote its *neighborhood* as  $\mathcal{N}(\mathbf{x}) \subseteq \mathbb{B}^n$ .

A *local-search* procedure for a pseudo-Boolean function  $f$  produces a sequence of points  $(\mathbf{p}^{(0)}, \mathbf{p}^{(1)}, \dots, \mathbf{p}^{(t)})$  such that

$$\mathbf{p}^{(k)} = \arg \min \left\{ f(\mathbf{p}) \mid \mathbf{p} \in \mathcal{N}(\mathbf{p}^{(k-1)}) \right\},$$

and that  $f(\mathbf{p}^{(t)}) \leq f(\mathbf{y})$  for all  $\mathbf{y} \in \mathcal{N}(\mathbf{p}^{(t)})$ .

Clearly, this procedure will produce a finite sequence of *decreasing* values for function  $f$ . This is the reason why this procedure is sometimes called *steepest-descent* or a *greedy*

method.

Due to the termination condition of the method,  $\mathbf{p}^{(t)}$  is called a *local optimum* of  $f$  in the neighborhood  $\mathcal{N}$ .

The input parameters of this approach are the starting point  $\mathbf{p}^{(0)}$  and the characterization of the neighborhood set  $\mathcal{N}$ .

A *standard* special family of local-search procedures is defined by

$$\mathcal{N}_H = \{\mathbf{y} \in \mathbb{B}^n \mid \mathbf{x} \in \mathbb{B}^n, d_{Hamming}(\mathbf{x}, \mathbf{y}) \leq 1\},$$

where  $d_{Hamming}(\mathbf{x}, \mathbf{y})$  represents the Hamming distance<sup>1</sup> between the  $n$ -vectors  $\mathbf{x}$  and  $\mathbf{y}$ .

These standard methods have already been investigated in the previous section, along other several alternative local-search methods, which are not necessarily steepest approaches.

A heuristic of this nature, selected among 40 variants, was called ACSIOM and it so happens to be a steepest descent heuristic, whose starting point is defined by method  $I_3$  (see Table 6.2).

The local search procedure applied to QUBO, based on the neighborhood  $\mathcal{N}_H$ , is not known if it is polynomially solvable ([194]). However, this method works very efficiently in practice for QUBO problems.

In Section 6.1 we have introduced a class of one-pass heuristics based on a rounding procedure that fixes a single variable at every iteration of the method, therefore producing a solution in polynomial time. The method stops after  $n$  iterations and therefore the solution may or may not be a local optimum. Clearly, the 0–1 vector returned by a one-pass method can be used as a starting point of more sophisticated local searches.

In this section we investigate the impact – in the quality and computing time of solutions – of applying a *standard steepest-descent* procedure to a starting point defined by a one-pass heuristic. This enhanced heuristic was called as STEEPEST-DESCENT(I),

---

<sup>1</sup> $d_{Hamming}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n |x_i - y_i|$ .

where  $\mathbb{I}$  is the (one-pass) method that defines the starting point. Five one-pass heuristics, ONE-PASS-R( $I_k$ ) ( $k = 1, \dots, 4$ ) and ONE-PASS-R( $I_5, \sigma^2 = \frac{1}{4}$ ), are considered as starting points  $\mathbb{I}$  in the subsequent analysis.

### 6.3.1 Computing time

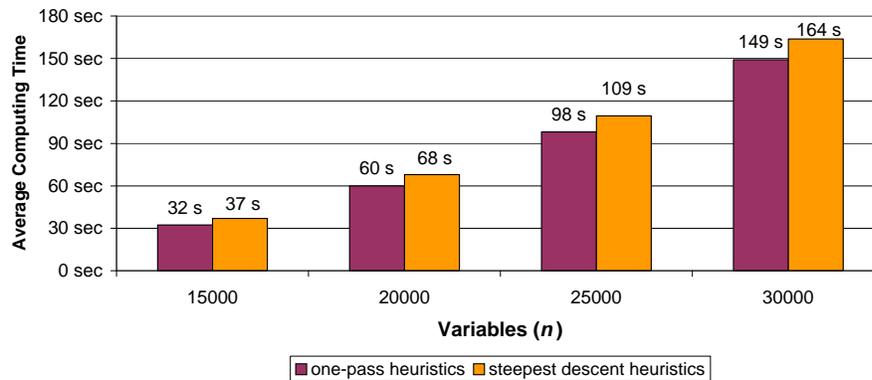


Figure 6.7: Average computing times of one-pass heuristics and of their local improvement by steepest ascent.

The average computing times of the five one-pass heuristics studied in this section, is displayed in Figure 6.7, for instances of the *Massive* family of QUBO problems. These instances have between 15 000 and 30 000 variables and expected densities of 30%, 60% and 90% (see Table 3.6). Since these instances are *maximization* QUBO problems, then we consider the greedy local search algorithm version to find a local *maximum*, and naturally called this procedure STEEPEST-ASCENT.

Figure 6.7 also displays the time needed to find a local optimum, for the same test problems, using the STEEPEST-ASCENT procedure from the one-pass solutions. The additional computing time needed to locally improve the one-pass heuristics is on an average case, 14% higher for QUBOs having 15 000 variables, and 10% higher for QUBOs having 30 000 variables. These numbers clearly indicate that a local optimum can be found efficiently in practice.

### 6.3.2 Quality of solutions

This section addresses the question of how much is the quality of the one-pass solutions improved if they are enhanced to a local optimum. Table 6.18 provides the average relative and normalized errors of the five ONE-PASS heuristics and corresponding STEEPEST-DESCENT versions, for various families of QUBO problems.

Table 6.18: Performance gain by locally improving the solutions of one-pass heuristics.

<i>Families of QUBO Problems</i>	<i>Expected Relative Error (G)</i>			<i>Expected Normalized Error (K)</i>		
	ONE PASS	STEEPEST DESCENT	<i>Average Gain</i>	ONE PASS	STEEPEST DESCENT	<i>Average Gain</i>
Benchmarks	3.61%	2.78%	<i>0.83%</i>	1.18%	0.67%	<i>0.51%</i>
Randomly generated	25.65%	19.36%	<i>6.29%</i>	0.76%	0.45%	<i>0.31%</i>
MAX-Clique	26.74%	24.21%	<i>2.53%</i>	0.53%	0.42%	<i>0.11%</i>
MIN-VC (planar)	0.28%	0.25%	<i>0.03%</i>	0.23%	0.21%	<i>0.03%</i>
MAX-CUT	5.10%	4.29%	<i>0.81%</i>	8.63%	7.18%	<i>1.45%</i>
MAX-2-SAT	3.35%	2.09%	<i>1.26%</i>	5.36%	3.10%	<i>2.26%</i>
All Problems	18.49%	14.04%	<i>4.45%</i>	1.96%	1.31%	<i>0.65%</i>

The average approximate relative error decrease of STEEPEST-DESCENT is 4.4%, and the associated average approximate normalized error is 0.65%.

The largest gain in relative error is in the randomly generated group of problems with an average value of 6.3%. The largest gain in normalized error occurs in MAX-2-SAT with an average value of 2.3%, followed by the MAX-CUT group of problems with an average error of 1.4%.

It is interesting to note that the efficacy of the STEEPEST-DESCENT heuristics studied, behaves quite differently depending on what performance measures are used.

### 6.3.3 Concluding remarks

Both the ONE-PASS and STEEPEST-DESCENT heuristics for QUBO produce reasonable quality solutions, very efficiently. These types of procedures are needed to handle problems having tens of thousands of variables and millions of quadratic terms.

The use of polynomial time algorithms embedded in more elaborated procedures is an important aspect, especially if one would like to guarantee that the more sophisticated algorithm has also a computing time bounded by a polynomial of its size.

The computational complexity of STEEPEST–DESCENT (or STEEPEST–ASCENT) methods for QUBO is not known. The use of ONE–PASS methods is therefore crucial to guarantee efficient performance. For instance, a probabilistic one-pass heuristic (based on a sparse network data structure) is used in the probing procedure of the QUBO preprocessing routine (see Figure 7.3).

One additional conclusion is that the one–pass and local–search heuristics for QUBO should use the distribution of the partial derivative (gradient) values to define starting points, as has been demonstrated by the computational experiments. This statement also suggests that a good starting point is typically an “interior” point of the cube  $\mathbb{U}^n$ . The literature on QUBO heuristics and meta-heuristics considers various alternative starting points, but they are traditionally defined by extreme points of  $\mathbb{B}^n$ .

To provide evidence to the previous statement, we did an experiment on the group  $G_1$  of QUBO problems proposed by Glover et al. [109].  $G_1$  includes 10 *maximization* problems having 1 000 variables and densities varying from 10% to 100% (see Tables 3.2 and A.3).

The experiment considers 1 000 randomly generated starting points used for the greedy local–search maximization heuristic (i.e. STEEPEST–ASCENT). Each starting point  $\mathbf{p}^{(k)} \in \mathbb{B}^n$  ( $k = 1, \dots, 1\,000$ ) is randomly generated from the Bernoulli ( $\frac{1}{2}$ ) distribution, i.e.  $p_i^{(k)} = 1$  with probability  $\frac{1}{2}$ .

Each of the heuristics STEEPEST–ASCENT( $p_1^{(k)}, \dots, p_n^{(k)}$ ),  $k = 1, \dots, 1\,000$ , is carried out on each of the  $G_1$  problems, and each one of their values is compared with the value returned by STEEPEST–ASCENT( $\frac{1}{2}, \dots, \frac{1}{2}$ ) on the same problem.

Under the circumstances that points  $\mathbf{p}^{(k)}$  have been generated, one would normally think that in about 50% of the cases the solution value of STEEPEST–ASCENT( $\mathbf{p}^{(k)}$ ) is not smaller than that returned by STEEPEST–ASCENT( $\frac{1}{2}, \dots, \frac{1}{2}$ ). The results of our experiment, displayed in Table 6.19, show however that this fact is not always true. Except for instance number five of the  $G_1$  family, STEEPEST–ASCENT( $\frac{1}{2}, \dots, \frac{1}{2}$ ) provided better values for all the remaining nine problems in more than 80% of the cases. Even in the case of instance five, the interior point method was superior in

63.9% of the cases.

Table 6.19: Percentage number of tests where the STEEPEST-ASCENT( $\frac{1}{2}, \dots, \frac{1}{2}$ ) heuristic gives better solution values than STEEPEST-ASCENT(Bernoulli( $\frac{1}{2}$ ),  $\dots$ , Bernoulli( $\frac{1}{2}$ )), for the  $G_1$  QUBO problems ([109]).

Problem Number	Density ( $\bar{d}$ %)	% of Cases where Interior Point Method STEEPEST-ASCENT( $\frac{1}{2}, \dots, \frac{1}{2}$ )		
		was Worse	was the Same	was Better
1	10	16.2%	0.1%	83.7%
2	20	19.7%	0.0%	80.3%
3	30	7.5%	0.0%	92.5%
4	40	0.7%	0.0%	99.3%
5	50	36.1%	0.0%	63.9%
6	60	1.8%	0.0%	98.2%
7	70	0.9%	0.0%	99.1%
8	80	2.3%	0.0%	97.7%
9	90	2.6%	0.0%	97.4%
10	100	9.3%	0.0%	90.7%

It is also interesting to note that out of the 10 000 experiments only one case has a value coincident with the value returned by STEEPEST-ASCENT( $\frac{1}{2}, \dots, \frac{1}{2}$ ). This fact provides some indication about the neighborhood space of solutions being potentially enlarged, if an interior point approach is also considered for QUBO.

## Chapter 7

### Preprocessing

A family of *preprocessing* techniques for QUBO is proposed in this chapter. It is based on several computationally efficient transformations of this problem, and aimed at simplifying it and possibly decomposing it into smaller problems of the same type. More precisely, the proposed preprocessing involves a series of transformations of the quadratic pseudo-Boolean objective function  $f$ , including:

- (i) The fixation of some of the variables at values which must hold at every optimum, and the enforcement of certain binary relations (e.g., equations, inequalities, or non-equalities) between the values of certain pairs of variables, which must hold in every optimum;
- (ii) The fixation of some of the variables and the enforcement of some binary relations between certain pairs of variables, which must hold in at least one optimal solution of the problem;
- (iii) The possible decomposition of the problem into several smaller QUBO problems involving pairwise disjoint subsets of the original variables.

As a result, we obtain a constant  $K$  and quadratic pseudo-Boolean functions  $f_r : \mathbb{B}^{S_r} \mapsto \mathbb{R}$ ,  $r = 1, \dots, c$ , where the sets of indices  $S_r \subseteq \{1, \dots, n\}$ ,  $r = 1, \dots, c$  are pairwise disjoint, such that

$$\min_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x}) = K + \sum_{r=1}^c \left( \min_{\mathbf{y} \in \mathbb{B}^{S_r}} f_r(\mathbf{y}) \right).$$

The proposed method has several key ingredients. The first ingredient is the *representation* of a quadratic pseudo-Boolean function by means of the implication network (see Section 5.3). This representation allows an efficient derivation via network flow

computations of a lower bound on the minimum of the function and allows the identification of variables whose values in the optimum can be easily predicted, as well as of other information which makes the simplification of the problem possible. The second ingredient is the *methodology* primarily based on the theory of *roof-duality* (see Chapter 5), combined with the use of first (see Section 4.2) and second order derivatives and co-derivatives (see Section 4.3). This methodology provides an effective lower bound for the minimum of a quadratic pseudo-Boolean function, as well as information about the values of a subset of the variables in the optimum (so called *linear persistencies*; see Section 4.1), and about binary relations which must hold between certain pairs of variables in the optimum (so called *quadratic persistencies*). An additional component of the methodology is the decomposition of a large problem into several smaller ones, which can be derived at a low extra cost by combining the conclusions of roof-duality with those offered by the network representation of the problem (see Section 7.2.3). The third ingredient of the proposed method consists of the *integration* of conclusions obtained from subproblems associated to the original problem. This integration is realized by combining the conclusions of probing (i.e. multiple application of roof-duality to naturally associated subproblems), and of Boolean consensus (i.e. the exhaustive expansion of the detected linear and quadratic persistencies).

We present an extensive computational evaluation of the proposed preprocessing method in Section 7.6, using various benchmark sets and randomly generated test problems of various types, involving thousands of variables, as described in Section 7.5. Our experience shows that for dense problems the proposed preprocessing technique is less effective as the size of the problems grows. It is demonstrated on numerous publicly available test problems that for relatively sparse problems, including in particular certain classes of structured problems, the proposed preprocessing method achieves substantial reductions in size at a very low computational cost. For instance, applying the method to QUBO problems corresponding to vertex cover problems in planar graphs involving up to 500 000 nodes, lead to the optimal fixation of 100% of the variables, i.e. to the exact solution of the problem, in every single instance considered (see Section

7.7).

## 7.1 Basic preprocessing tools

### 7.1.1 First order derivatives

The  $i$ -th first order partial derivative ( $i \in \mathbf{V}$ ) of a quadratic pseudo-Boolean function  $f$  is given by (4.4). Corollary 4.1 of Section 4.2 provides necessary conditions of optimality for QUBO by analyzing the sign of the derivative functions, which can automatically determine certain (strong or weak) persistencies. If  $\Delta_i(\mathbf{x})$  is strictly negative, respectively strictly positive, the above implications represent strong persistencies. These simple relations have been already noticed in Hammer and Rudeanu [131], and represent an essential component of even the most recent work on preprocessing (see [24]).

Hammer, Hansen and Simeone [123] have shown that those strong persistencies which can be obtained from the analysis of partial derivatives, can also be obtained by roof-duality (a tool described in Chapter 5). Moreover, roof-duality is a stronger preprocessing technique, as shown in [123], where an example is presented displaying persistencies found by roof-duality, but not following from the analysis of the signs of partial derivatives (see Example 5.6).

In view of these results, the preprocessing algorithm to be described in Section 7.4, which will exploit heavily roof-duality, will not explicitly include an analysis of the signs of partial derivatives, since the conclusions derivable from such an analysis will be automatically included among those provided by roof-duality.

### 7.1.2 Second order derivatives and co-derivatives

A natural generalization of the concept of the first order derivative (see Section 4.3) allows us to establish some persistent binary relations to hold among the values taken in the optimum by certain pairs of variables.

In particular, we shall consider quadratic persistencies established by Theorem 4.1 of Section 4.1, which uses the concept of  $(i, j)$ th *second order derivative* of  $f$  (denoted by  $\Delta_{ij}$ ), proposed by Hammer and Hansen [121]. The complement concept of  $(i, j)$ th

*second order co-derivative* (denoted by  $\nabla_{ij}$ ) was also introduced in Section 4.1. With these notations,

$$\Delta_{ij}(\mathbf{x}) = f(\mathbf{x}[\{i, j\} \leftarrow (1, 0)]) - f(\mathbf{x}[\{i, j\} \leftarrow (0, 1)])$$

and

$$\nabla_{ij}(\mathbf{x}) = f(\mathbf{x}[\{i, j\} \leftarrow (1, 1)]) - f(\mathbf{x}[\{i, j\} \leftarrow (0, 0)]),$$

i.e. evaluate the effect of simultaneously changing the values of  $x_i$  and  $x_j$ , while keeping the values of the other variables unchanged.

From Theorem 4.1, then if  $f(x_1, \dots, x_n)$  is a quadratic pseudo-Boolean function, and if  $x_i$  and  $x_j$  are two of its variables, then

- (i) If  $\nabla_{ij}(\mathbf{x}) \geq 0$  for every  $\mathbf{x} \in \mathbb{B}^{\mathbf{V}}$  then  $x_i x_j = 0$  is a weak persistency;
- (ii) If  $\nabla_{ij}(\mathbf{x}) \leq 0$  for every  $\mathbf{x} \in \mathbb{B}^{\mathbf{V}}$  then  $\bar{x}_i \bar{x}_j = 0$  is a weak persistency;
- (iii) If  $\Delta_{ij}(\mathbf{x}) \geq 0$  for every  $\mathbf{x} \in \mathbb{B}^{\mathbf{V}}$  then  $x_i \bar{x}_j = 0$  is a weak persistency;
- (iv) If  $\Delta_{ij}(\mathbf{x}) \leq 0$  for every  $\mathbf{x} \in \mathbb{B}^{\mathbf{V}}$  then  $\bar{x}_i x_j = 0$  is a weak persistency.

If in any of the implications above the left hand side inequality is strict, then the corresponding persistency is strong.

## 7.2 Roof-duality

Let us recall first that the *roof-dual bound* proposed in [123], can be determined efficiently by maximum flow computations in the implication network  $G_\phi$  ([51, 54, 56]; see Section 5.3).

From Proposition 5.8, for any quadratic posiform  $\phi$  given by (1.6) and any feasible flow  $\varphi$  in the corresponding implication network  $G_\phi$  the equality

$$\phi = a_0 + v(G_\phi[\varphi]) + \psi_{G_\phi[\varphi]}$$

holds, where the right hand side is a quadratic posiform. Therefore,

$$a_0 + v(G_\phi[\varphi]) \leq \min_{\mathbf{x} \in \mathbb{B}^V} \phi(\mathbf{x}). \quad (7.1)$$

Incidentally, if  $\varphi$  is a maximum flow for network  $G_\phi$  then we have  $v(G_\phi[\gamma]) \leq v(G_\phi[\varphi])$ , where  $\gamma$  is any flow of the network. Thus, the computation of the maximum flow determines the best lower bound in (7.1) exactly for the implication network form of a quadratic posiform. In fact the bound in (7.1) was shown in [51, 54] to be the same as the roof-dual value of  $\phi$ , introduced in [123].

### 7.2.1 Strong persistency

Let us observe next that if  $\varphi$  is a feasible flow in an implication network  $G_\phi[\varphi]$  then in view of the existence of a symmetric flow and of the equality

$$v(\tilde{\varphi}) = v(\varphi),$$

the symmetrized flow  $\tilde{\varphi}$  defined by

$$\tilde{\varphi}(u, v) = \tilde{\varphi}(\bar{v}, \bar{u}) = \frac{\varphi(u, v) + \varphi(\bar{v}, \bar{u})}{2} \text{ for all } u, v \in N, u \neq v$$

is also a feasible flow in  $G_\phi$ . This implies that in any implication network, among the maximum flows there always exists a *symmetric* one, for which  $\varphi = \tilde{\varphi}$  holds.

Let us consider therefore a symmetric maximum flow  $\varphi$  in the implication network  $G_\phi[\varphi]$ , and let  $\psi = \psi_{G_\phi[\varphi]}$ . As we observed above, the corresponding implication network  $G_\psi$  is exactly the residual network of  $G_\phi$  corresponding to flow  $\varphi$ . Let us then define  $S \subseteq N$  as the set of nodes  $v \in N$  to which there exists a directed  $x_0 \mapsto v$  path in  $G_\psi$ , each arc of which has a positive residual capacity (we assume that  $x_0 \in S$ ). Furthermore, let  $T = \{\bar{v} \mid v \in S\}$ . Since  $\varphi$  is a maximum flow in  $G_\phi$ , we cannot have  $\bar{x}_0 \in S$ , and consequently  $T \cap S = \emptyset$  follows by the assumed symmetry flow conditions of the implication network  $G_\psi$ .

The following fact is well-known in network optimization:

**Proposition 7.1.** *The set  $S$  is unique, and is independent of the choice of the maximum flow  $\varphi$ . It is in fact the intersection of the source sides of all minimum cuts of  $G_\phi$ .*

Let us note also that  $\{u, v\} \subseteq S$  cannot hold for any quadratic term  $a_{uv}uv$  of  $\psi$  with positive coefficient  $a_{uv}$ , since otherwise we would have a positive capacity arc  $(u, \bar{v})$  from  $u \in S$  to  $\bar{v} \in T$  by the definition of arc capacities in the implication network associated to  $\psi$ , leading to a positive capacity path from  $x_0$  to  $\bar{x}_0$ , in contradiction with the maximality of  $\varphi$ . Thus, it follows that the assignment which sets all literals in  $S$  to 1 (there exists such an assignment, since  $T \cap S = \emptyset$ ) makes all terms of  $\psi$  which involve literals from  $S$  or  $T$  vanish. Introduce

$$J = \{j \in \mathbf{V} \mid \{x_j, \bar{x}_j\} \cap S \neq \emptyset\}$$

and let  $\mathbf{y} \in \mathbb{B}^J$  be the partial assignment for which  $u(\mathbf{y}) = 1$  holds for all  $u \in S$  (and consequently,  $v(\mathbf{y}) = 0$  for all  $v \in T$ ).

From Proposition 5.9, the partial assignment  $\mathbf{y} \in \mathbb{B}^J$  is a strong autarky<sup>1</sup> for  $\psi$  (and hence for  $\phi$ ). Consequently, the assignments  $x_j = y_j$  for all  $j \in J$  are strongly persistent for QUBO.

In fact the set of variables  $x_j$ ,  $j \in J$  consists exactly of those involved in the so called *master roof* as defined in [123]. This discussion shows that as a byproduct of a maximum flow computation, the above approach determines  $J$  in additional computing time, which is linear in the number of nonzero terms of  $\psi$ , i.e. linear in the size of  $\phi$ .

### 7.2.2 Weak persistency

Let us consider now the directed graph  $\widehat{G}$  obtained from  $G_\psi$  by keeping only those arcs which have a positive residual capacity. Since we can assume that the maximum flow  $\varphi$  is symmetric, we shall have

$$(\bar{v}, \bar{u}) \in A(\widehat{G}) \quad \text{whenever} \quad (u, v) \in A(\widehat{G}). \quad (7.2)$$

---

<sup>1</sup>A partial assignment  $\mathbf{y} \in \mathbb{B}^S$ ,  $S \subseteq V$  is called an *autarky* for  $\phi$  if all terms of  $\phi$  involving variables with indices from  $S$  vanish when we substitute  $\mathbf{y}$ .

Let us compute the strong components of this directed graph (necessitating linear time in the number of arcs, i.e. linear time in the size of  $\psi$  [227]), and denote these strong components by  $K_i$ ,  $i = 1, \dots, c$ . It is easy to see that the symmetry (7.2) implies the following

**Proposition 7.2.** *For every strong component  $K_i$  of  $\widehat{G}$  we have either*

$$\{\bar{v} \mid v \in K_i\} = K_i \quad (7.3)$$

or

$$\{\bar{v} \mid v \in K_i\} = K_{i'} \quad (7.4)$$

for some  $i' \neq i$ .

*Proof.* Follows readily by (7.2). □

Let us label now as  $K_1, K_{1'}, K_2, K_{2'}, \dots, K_\ell, K_{\ell'}$ , those strong components which satisfy (7.4), in such a way that

(i) there is no directed path in  $\widehat{G}$  from  $K_i$  to  $K_{i'}$  for  $i = 1, \dots, \ell$ , and

(ii) there is no directed path in  $\widehat{G}$  from  $x_0$  to  $K_{i'}$  for  $i = 1, \dots, \ell$ .

Since  $\varphi$  is a maximum flow, we cannot have a directed path from  $x_0$  to  $\bar{x}_0$  in  $\widehat{G}$ , and hence the symmetry conditions (7.2) imply the existence of such a labeling. Let us note that condition (i) is equivalent to saying that the strong components  $K_i$  and  $K_{i'}$  are dual components of the associated implication graph (see Section 4.5).

Let  $J_i = \{j \in \mathbf{V} \mid \{x_j, \bar{x}_j\} \cap K_i \neq \emptyset\}$  and let  $\mathbf{y}_i \in \mathbb{B}^{J_i}$  be the partial assignment for which  $u(\mathbf{y}_i) = 1$  for all  $u \in K_i$ , for  $i = 1, \dots, \ell$ .

**Theorem 7.1.** *The partial assignment  $\mathbf{y}_i$  is an autarky for  $\psi$ , for  $i = 1, \dots, \ell$ . Moreover, it is a strong autarky if there is a directed path in  $\widehat{G}$  from  $x_0$ , or if there is a directed arc between  $K_i$  and  $K_{i'}$ . Consequently, the assignments  $x_j = y_{ij}$  for all  $j \in J_i$  and  $i = 1, \dots, \ell$  are all persistent assignments for QUBO.*

*Proof.* The claim is implied by the fact that the terms of  $\psi$  including variables in  $J_i, i = 1, \dots, \ell$  vanish.  $\square$

It should be remarked that the weak persistency results of the previous Theorem are “stronger” than those derived from Theorem 5.3, since the later persistencies are all derivable from those determined by Theorem 7.1.

Let us note that if there is a directed arc  $(u, v)$  between  $K_i$  and  $K_{i'}$  for some  $i = 1, \dots, \ell$ , then symmetry (7.2) implies that an arc  $(\bar{v}, \bar{u})$  must also exist between  $K_i$  and  $K_{i'}$ . It is this property that makes the persistency result of Proposition 7.1 to be strong, in this particular situation.

In general, deciding if a partial assignment  $\mathbf{y}_i$  is a strong autarky for  $\phi$ , for  $i = 1, \dots, \ell$  is a NP-hard decision problem. This result can easily be established, since the outcome of this decision depends on the optimization of NP-hard sub-problems, each one associated to a component of type (7.3).

**Example 7.1.** Consider the quadratic posiform  $\phi$  given by

$$\phi = 2x_1\bar{x}_2 + 4\bar{x}_1x_2 + 2x_2\bar{x}_3 + 2\bar{x}_2x_3 + 2x_1x_3 + 4\bar{x}_1\bar{x}_3 + 2x_4\bar{x}_5 + 2\bar{x}_4x_5 + 2x_5\bar{x}_1,$$

and the associated network  $G_\phi$  shown in Figure 7.1. The strong components of  $G_\phi$  are:

$$\begin{aligned} K_1 &= \{x_0\}, \\ K_{1'} &= \{\bar{x}_0\}, \\ K_2 &= \{\bar{x}_4, \bar{x}_5\}, \\ K_{2'} &= \{x_4, x_5\} \text{ and} \\ K_3 &= \{x_1, x_2, x_3, \bar{x}_1, \bar{x}_2, \bar{x}_3\}. \end{aligned}$$

Let us note that there is no directed path from  $K_2$  to  $K_{2'}$ , there is no directed path from  $x_0$  to  $K_{2'}$ , and that  $K_2$  and  $K_{2'}$  satisfy condition (7.4). Let  $J_2 = \{4, 5\}$  and let  $\mathbf{y}_2 \in \mathbb{B}^{J_2}$  be the partial assignment for which  $\bar{x}_4 = \bar{x}_5 = 1$ . By Proposition 7.1,  $x_4 = x_5 = 0$  must hold in a minimizer of  $\phi$ . Let us now show that these two assignments are not strongly persistent. In the first place one can verify that  $x_1 = 1$  must hold in all minimizers of

$\phi$ . Therefore, the only term connecting  $K_3$  to the other components vanishes. So, any solution satisfying the equation  $x_4\bar{x}_5 + \bar{x}_4x_5 = 0$ , including  $x_4 = x_5 = 1$ , must also hold in a minimizer of  $\phi$ .

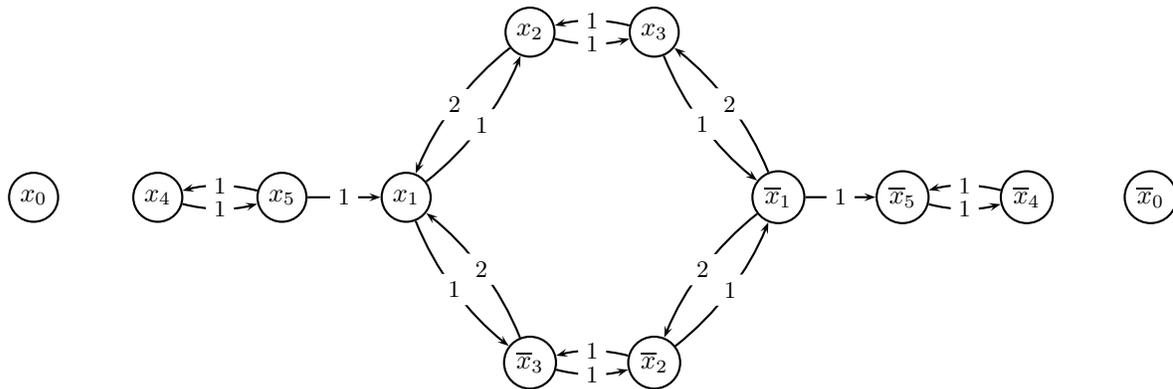


Figure 7.1: The network  $G_\phi$  corresponding to the posiform  $\phi$  of Example 7.1. We indicate only those arcs which have positive capacities.

### 7.2.3 Decomposition

Let us assume now that we have already fixed all weakly and strongly persistent variables (e.g., by Proposition 7.1), and that the strong components of the residual posiform  $\psi$ ,  $K_i$ ,  $i = 1, \dots, c$ , are all of type (7.3).

Clearly, in this case the symmetry conditions (7.2) imply that there are no arcs between different strong components, i.e.  $\psi$  does not involve a quadratic term  $a_{uv}uv$ ,  $a_{uv} > 0$  for which  $u \in K_i$ ,  $v \in K_j$  and  $i \neq j$ . Thus, denoting by  $\psi_i$  the posiform corresponding to the induced subnetwork  $K_i$ ,  $i = 1, \dots, c$ , we have

$$\psi = \sum_{i=1}^c \psi_i.$$

Furthermore, due to property (7.3), these posiforms involve disjoint sets of variables. Hence, we have

**Theorem 7.2.**

$$\min_{\mathbf{x} \in \mathbb{B}^V} \psi(\mathbf{x}) = \sum_{i=1}^c \left( \min_{\mathbf{x} \in \mathbb{B}^{J_i}} \psi_i(\mathbf{x}) \right). \quad (7.5)$$

A similar decomposition of quadratic posiforms which does not involve linear terms was already proposed in [42] (see Proposition 5.5). Let us note that after computing the maximum flow in the implication network  $G_\phi$ , both the persistent assignments, as well as the above decomposition can be determined in linear time of the size of  $\phi$ .

**Example 7.2.** Consider  $\phi$  to be a homogeneous quadratic posiform whose nonzero terms are:

$$\begin{aligned} & x_1\bar{x}_2, \bar{x}_1x_2, x_2\bar{x}_3, \bar{x}_2x_3, x_1x_3, \bar{x}_1\bar{x}_3, x_4\bar{x}_5, \bar{x}_4x_5, x_5\bar{x}_6, \bar{x}_5x_6, x_4x_6, \bar{x}_4\bar{x}_6, \\ & x_7\bar{x}_8, \bar{x}_7x_8, x_1\bar{x}_7, x_4\bar{x}_8, x_9\bar{x}_{10}, \bar{x}_9x_{10}, x_7\bar{x}_9, \bar{x}_{11}, x_{11}\bar{x}_{12}, \bar{x}_{11}x_{12}, x_{10}\bar{x}_{11}. \end{aligned}$$

The associated network  $G_\phi$ , shown in Figure 7.2, has the following strong components:

$$\begin{aligned} K_1 &= \{x_0\}, & K_{1'} &= \{\bar{x}_0\}, \\ K_2 &= \{x_{11}, x_{12}\}, & K_{2'} &= \{\bar{x}_{11}, \bar{x}_{12}\}, \\ K_3 &= \{x_9, x_{10}\}, & K_{3'} &= \{\bar{x}_9, \bar{x}_{12}\}, \\ K_4 &= \{x_7, x_8\}, & K_{4'} &= \{\bar{x}_7, \bar{x}_8\}, \\ K_5 &= \{x_1, x_2, x_3, \bar{x}_1, \bar{x}_2, \bar{x}_3\} \text{ and} \\ K_6 &= \{x_4, x_5, x_6, \bar{x}_4, \bar{x}_5, \bar{x}_6\}. \end{aligned}$$

Let us first note that there is no directed path from  $x_0$  to  $\bar{x}_0$ . Thus, by strong persistency (Proposition 5.9)  $x_{11} = x_{12} = 1$  must hold for all minimizers of  $\phi$ . Also, regardless of the values of the coefficients in the nontrivial terms of  $\phi$ , by weak persistency (Proposition 7.1) the partial assignment  $x_7 = x_8 = x_9 = x_{10} = x_{11} = x_{12} = 1$  must hold in a minimizer of  $\phi$ . This partial assignment automatically cancels all those terms of  $\phi$  which involve at least a variable from the set  $\{x_7, x_8, x_9, x_{10}, x_{11}, x_{12}\}$ . After eliminating these terms, the original problem is decomposed into two subproblems, involving disjoint sets of variables, coming respectively from  $K_5$  and  $K_6$ . Obviously, these two subproblems can be optimized separately, and the sum of their optimal values will coincide with the minimum of  $\phi$ .



improved lower bound of the minimum of a quadratic pseudo-Boolean function, and of enlarging at the same time the set of variables and binary relations for which persistency conclusions apply.

In view of the fact that finding the roof-dual of a quadratic pseudo-Boolean function can be achieved by simply solving a maximum flow problem in a network, the calculation of the roof-duals of  $2n$  quadratic pseudo-Boolean functions associated to the initially considered one is a feasible, easy-to-carry-out operation. Let us provide now some technical ideas on how to efficiently calculate the  $2n$  roof-duals required by probing.

Let us assume first that using a simple heuristic (e.g. as proposed in [58, 61]) we have found some upper bound  $U$  on the minimum of (1.5), say

$$\min_{\mathbf{x} \in \mathbb{B}^V} \phi(\mathbf{x}) \leq U. \quad (7.6)$$

Let us now consider the most typical branching procedure, in which we split the problem into two somewhat smaller ones by fixing variable  $x_j$  at the two possible binary values. Since  $\phi$  is a posiform, all of its terms – with the possible exception of the constant  $a_0$  – contribute nonnegative quantities to the objective. Therefore, if  $M > U - a_0$ , then

$$\min_{\mathbf{x} \in \mathbb{B}^V} \phi(\mathbf{x}) = \min \left\{ \min_{\mathbf{x} \in \mathbb{B}^V} \phi(\mathbf{x}) + Mx_j, \min_{\mathbf{x} \in \mathbb{B}^V} \phi(\mathbf{x}) + M\bar{x}_j \right\}, \quad (7.7)$$

where  $a_0$  is the constant in  $\phi$ , as given in (1.6).

The two subproblems in (7.7) have simple network representations.

In order to calculate the roof-duals of  $\min_{\mathbf{x} \in \mathbb{B}^V} \phi(\mathbf{x}) + Mx_j$  and of  $\min_{\mathbf{x} \in \mathbb{B}^V} \phi(\mathbf{x}) + M\bar{x}_j$ , and to derive persistency relations from these, we should add to the original network an arc,  $(x_0, \bar{x}_j)$  in the first case and  $(x_0, x_j)$  in the second case, and to assign to these the large capacity  $M$ .

Clearly, computationally it is simpler to increase the capacity of two arcs than to substitute  $x_j = 0$ , respectively  $x_j = 1$ , implying the deletion of nodes  $x_j$  and  $\bar{x}_j$ , and of all arcs incident to these nodes in the network. In addition, keeping the same network and updating the capacities of a few arcs at each branch evaluation, allows us to carry

out computations without increasing the amount of computer memory needed to keep the network data necessary to find a maximum flow for each subproblem. From an implementational point of view, this approach has the added advantage of allowing the easy restoration of the network corresponding to the original problem, by simply finding an additional maximum flow – an option which turned out to be on the average to be much better than creating a copy to be reused after branching. It should be remarked that without these simplifying steps, the large scale QUBOs (including for instance those coming from finding optimal vertex covers of planar graphs with half a million vertices; see Section 7.7) could not have been handled.

We can similarly administer more complicated branching policies, as well. For instance, if  $u, v \in \mathbf{L}$  are two literals,  $u \neq v$ , then branching on the binary relation  $u \leq v$  can be written as

$$\min_{\mathbf{x} \in \mathbb{B}^{\mathbf{V}}} \phi(\mathbf{x}) = \min \left\{ \min_{\mathbf{x} \in \mathbb{B}^{\mathbf{V}}} \phi(\mathbf{x}) + M\bar{u} + Mv, \min_{\mathbf{x} \in \mathbb{B}^{\mathbf{V}}} \phi(\mathbf{x}) + Mu\bar{v} \right\} \quad (7.8)$$

for some  $M > U - a_0$ , resulting in the modification of 4 arc capacities in the first branch corresponding to  $u = 1$  and  $v = 0$ , and of two arc capacities on the other branch corresponding to  $u \leq v$ .

We can also apply the above for handling persistencies. For instance, if we learn that  $u \leq v$  is a persistent binary relation, then we can rewrite (7.8) as

$$\min_{\mathbf{x} \in \mathbb{B}^{\mathbf{V}}} \phi(\mathbf{x}) = \min_{\mathbf{x} \in \mathbb{B}^{\mathbf{V}}} \phi(\mathbf{x}) + Mu\bar{v}. \quad (7.9)$$

Let us note that in all of the above cases, we had to increase the capacity of some of the arcs. Thus, as our procedure advances and we learn more and more persistencies, at the same time the maximum flow value is also increasing. Hence, according to (7.1), as an added value we get better and better lower bounds on the minimum of (1.5).

To describe probing and its benefits, let us consider an arbitrary quadratic posiform

$\phi$ , as given in (1.6). For a literal  $u \in \mathbf{L}$  let us consider the posiform

$$\psi_u = \phi + M\bar{u},$$

where  $M > U - a_0$  for an upper bound  $U$  satisfying (7.6). Let us further denote by  $S_u \subseteq \mathbf{L}$  ( $W_u \subseteq \mathbf{L}$ ) the set of strongly (weakly) persistent literals for  $\psi_u$ , as defined in Section 7.2.1 (7.2.2), and let  $L_u$  denote the roof–dual bound for  $\psi_u$ .

We can derive several consequences from the sets  $S_u$ ,  $W_u$  and lower bounds  $L_u$  when generating these for all literals  $u \in \mathbf{L}$ .

**Proposition 7.3.** *Let  $U$  be an upper bound of  $\min_{\mathbf{x} \in \mathbb{B}^{\mathbf{V}}} \phi(\mathbf{x})$ , and let  $u \in \mathbf{L}$  and  $j \in \mathbf{V}$ .*

*Then*

- *The value  $L^* = \max_{u \in \mathbf{L}} \min \{L_u, L_{\bar{u}}\}$  is a lower bound on the minimum of  $\phi$ .*
- *If  $L_u > U$  then  $u = 0$  is a strongly persistent assignment for  $\phi$ .*
- *If  $v \in S_{x_j} \cap S_{\bar{x}_j}$  ( $v \in W_{x_j} \cap W_{\bar{x}_j}$ ) then  $v = 1$  is a strongly (weakly) persistent assignment for  $\phi$ .*
- *If  $v \in S_{x_j}$  and  $\bar{v} \in S_{\bar{x}_j}$  ( $v \in W_{x_j}$  and  $\bar{v} \in W_{\bar{x}_j}$ ) then  $x_j = v$  is a strongly (weakly) persistent relation for  $\phi$ .*
- *For all  $v \in S_{x_j}$  and  $w \in S_{\bar{x}_j}$  ( $v \in W_{x_j}$  and  $w \in W_{\bar{x}_j}$ ) the quadratic relations  $x_j \leq v$ ,  $\bar{x}_j \leq w$  and  $\bar{w} \leq v$  are all strongly (weakly) persistent for  $\phi$ .*

All these follow from the above definitions, by which the assignment  $v = 1$  is strongly (weakly) persistent for  $\psi_u$ , for all  $v \in S_u$  ( $v \in W_u$ ). Let us note that by adding these new persistencies to  $\phi$ , as in (7.7) and (7.9), we may increase both the roof–dual value as well as the set of strongly and weakly persistent literals of  $\phi$ . Furthermore, the addition of these to  $\phi$  may also change the sets  $S_v$  or  $W_v$  for some other literals  $v \in \mathbf{L}$ ,  $v \neq u$ .

It is simple to verify that the quadratic persistencies determined through the previous proposition include a subset of those persistent binary relations derivable from

Theorem 4.1. However, probing may not be able to find quadratic persistencies which are determined by the analysis of the second order (co-)derivatives (see Section 7.1.2).

Let us remark that the lower bound derived from probing was also considered in [51, 56], and that analogous techniques were explored in the broader context of binary optimization in [23, 216].

### 7.3.2 Consensus

It has been remarked above that order relations between literals can be derived both from the signs of the second order derivatives and co-derivatives, as well as during the process of probing. The interactions of the various binary relations, and the conclusions obtained by combining them can be very easily derived by the introduction of a Boolean quadratic equation  $\Phi = 0$ , where the terms of  $\Phi$  are quadratic elementary conjunctions (representing the detected binary relations between literals) which must take the value zero in every minimum of (1.5). Moreover, the application of the consensus method to  $\Phi$  allows the polynomial detection of all of its prime implicants (see [89]). Taking into account that the prime implicants of this function represent either variables with fixed values in the optimum, or order relations which must hold between pairs of literals in every optimum, it is clear that the detection of all prime implicants of  $\Phi$  provides an enlarged set of strong persistencies.

Finally, we should remark that the conclusions that can be obtained from the knowledge of the prime implicants of  $\Phi$  can also be obtained directly (using Proposition 7.3) during our implementation of probing, by appropriately transforming the original functions via term additions (as explained in the previous section) corresponding to the persistent binary relations found by the preprocessing tools considered.

## 7.4 Algorithm and implementation

The proposed preprocessing algorithm is presented in this section, in which the tools described in the previous sections are used iteratively and recursively. The structure adopted is based on the network flow model of Section 5.3. Our goal is to find better

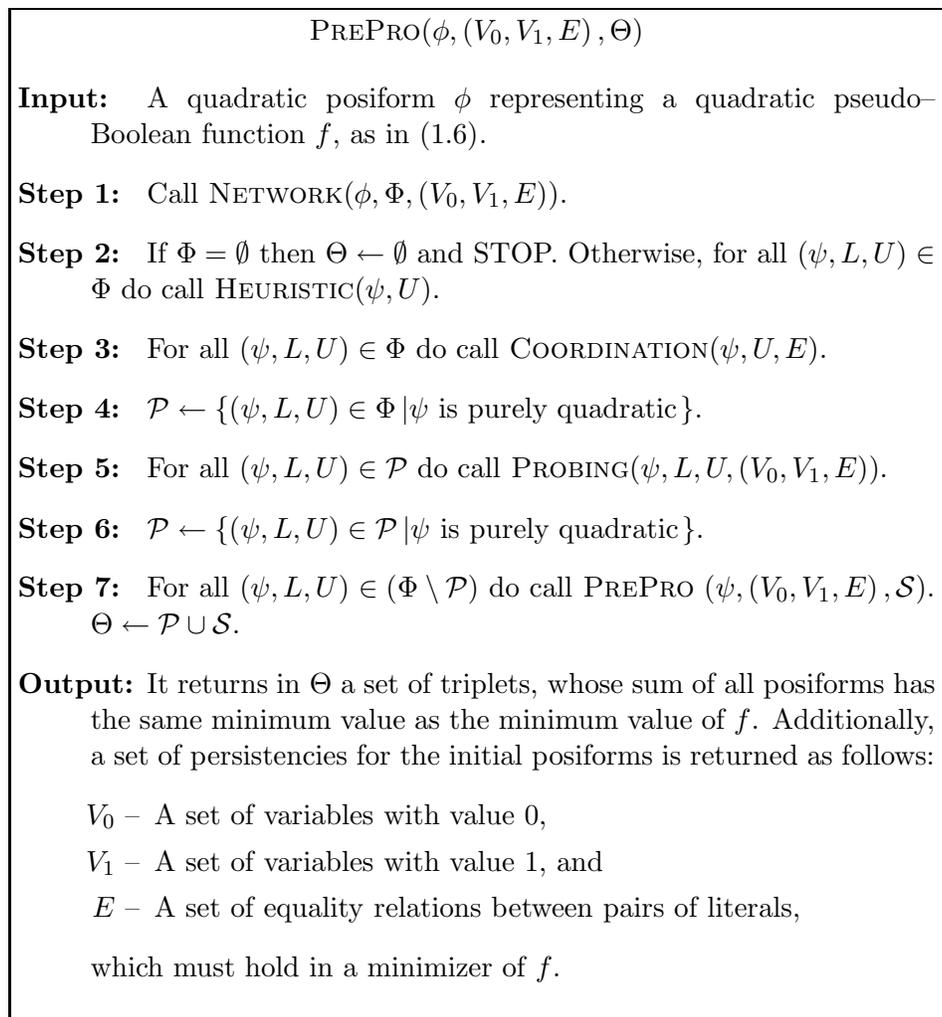


Figure 7.3: PREPRO algorithm.

and better lower bounds, weak (and strong) linear persistencies and weak (and strong) quadratic persistencies for  $\phi$ , as well as to decompose the problem into several smaller problems, as explained in Section 7.2.3.

The PREPRO algorithm is described in Figure 7.3. The input of the algorithm is a quadratic posiform  $\phi$  representing a quadratic pseudo-Boolean function  $f$ , as in (1.6). The output returned by PREPRO is a decomposed representation of the minimum of  $f$ , as in (7.5), where the subproblems on the right hand side of (7.5) involve pairwise disjoint sets of variables, together with a lower and an upper bound to the minimum of each subproblem, and with a partial (optimal) assignment to the derived persistent variables.

Four main components are part of PREPRO:

- NETWORK – This routine has a posiform  $\phi$  as input. It first finds a maximum flow in the network  $G_\phi$  as explained in Proposition 5.8 (see Section 7.2). The maximum flow implementation that we have adopted is based on the shortest augmenting path algorithm, yielding a worst case time of  $O(n^3)$ , and is especially designed to obtain a residual network satisfying the flow symmetry conditions. When a minimum cut is found, a set of strong persistencies is obtained directly from a non-empty source side of a minimum cut (see Propositions 5.8 and 7.1) and saved accordingly in  $(V_0, V_1)$ . The nodes of the residual network and corresponding arcs, which are associated to the set of strong persistencies are removed from the network. Using a linear time algorithm for identifying the strongly connected components of the residual network, a subset of weak persistencies is identified in every component of type (7.4) (see also Theorem 7.1), and saved accordingly in  $(V_0, V_1)$ . The nodes of the residual network and corresponding arcs, which are associated to the set of weak persistencies is removed from the network. What is left after applying NETWORK is a disjoint set of strong components, each corresponding to a subproblem (included in  $\Phi$ ) that can be optimized separately from the other subproblems.
- HEURISTIC – For each subproblem identified in NETWORK, an upper bound  $U$  is found, and later used in the COORDINATION and PROBING procedures. Any heuristic method valid for QUBO can be used. All of our experimental results include a fast one-pass heuristic based on a greedy approach, which ranks the set of non-fixed variables according to an approximated probability value of a variable to have a persistent value (see Chapter 6).
- COORDINATION – This procedure has as input a posiform  $\psi$ , and the upper bound  $U$  found by HEURISTIC. Let us note that the minimum of any posiform  $\psi$  called within PREPRO is *strictly* larger than its constant value  $a_0$ . This routine identifies binary persistent relations originated from the analysis of the second order derivative and co-derivative functions as explained in Section 4.3. The basic idea is to

compute over all possible pair of variables  $i < j$ , the minimum and the maximum of the linear pseudo-Boolean functions  $\Delta_{ij}$  and  $\nabla_{ij}$ . A key element to save computing time in this operation is to stop it as soon as one learns that the minimum is strictly negative and the maximum is strictly positive. If a quadratic persistency  $u \leq v$  is found, then  $\psi$  is updated by adding a term  $a_{u\bar{v}}u\bar{v}$  with a large enough coefficient (we use  $a_{u\bar{v}} = 2(U - a_0) + 1$ ). Since the implication network structure was adopted in all tools, this last operation can be efficiently implemented by updating the coefficient of the arcs  $(u, v)$  and  $(\bar{v}, \bar{u})$ . Our data structure is also able to quickly identifying if the reverse relation  $v \leq u$  is a quadratic persistency. In the affirmative case,  $E$  is updated to include the equality persistency  $u = v$ , and  $\psi$  is transformed by replacing  $v$  ( $\bar{v}$ ) by  $u$  ( $\bar{u}$ ). This routine stops as soon as a linear term or a equality persistency is found.

- **PROBING** – This procedure has as input a *purely* quadratic posiform  $\psi$  (i.e. a posiform that does not have linear terms) and the upper bound  $U$  found by **HEURISTIC**. The implication network structure plays a crucial role in this routine. Independently, each variable  $x_j$  is fixed to one of the possible binary values, and the resulting function is analyzed in terms of roof-duality and strong and weak persistencies. This operation can be accommodated easily in the network  $\psi$  as explained in Section 7.3.1. For a given assignment to  $x_j$ , a maximum flow algorithm is applied to the transformed network. All the strong and weak persistencies derived from the residual network are updated both in the network and in  $(V_0, V_1, E)$ , as explained in Proposition 7.3. To analyze the complement assignment of  $x_j$ , a maximum flow algorithm is applied to the residual network. All the strong and weak persistencies derived from the residual network are again updated as before. A third maximum flow algorithm is applied to obtain a network which represents the same function as  $\psi$ . The use of maximum flow algorithms to recuperate the original function being optimized is possible due to Proposition 5.8. A clear advantage of this approach is that the amount of memory needed for the data structures remains about the same through every step of the procedure.

Let us note that probing through the implication network is able to capture persistencies of transitive relations (see Section 7.3.2). For instance, suppose that  $u \leq v$  and  $v \leq w$  are quadratic persistencies, then if at some point  $w \leq u$  is also found to be persistent, then the network and the set  $E$  are immediately updated with the equality relation  $u = v = w$ . The routine stops as soon as a linear term or a linear/equality persistency is found.

Step 4 of PREPRO selects the subproblems for which probing is applied. Step 6 of PREPRO selects the subproblems for which PREPRO is recursively applied. Obviously, the rules that govern these choices may vary. In our implementation, the rule adopted is to apply NETWORK to a subproblem whenever a new linear persistency or a new linear term was found by COORDINATION or PROBING.

All the tools considered in PREPRO are polynomial time algorithms:

- NETWORK       –  $O(n^3)$ ;
- HEURISTIC     –  $O(n^2)$ ;
- COORDINATION –  $O(n^3 \log(n))$ ;
- PROBING       –  $O(n^4)$ .

As a consequence of the previous complexity times, each run from Step 1 to Step 5 of PREPRO takes at most  $O(n^4)$ .

## 7.5 Test problems

Most of the problem classes on which we have tested the proposed preprocessing procedures are benchmarks used in several other studies related to QUBO. These datasets include 85 problems with prescribed density, 13 graphs for MAX-Clique, 38 graphs for MAX-CUT, and 34 MAX-2-SAT formulas. Beside the above classes we have also carried out computational experiments on 436 randomly generated planar graphs for vertex cover optimization problems.

### 7.5.1 Benchmarks with prescribed density

The class of benchmarks with prescribed density that we have examined in this chapter includes the test problems of Glover, Kochenberger and Alidaee [108], and the problems of Beasley [37] with at most 500 variables. The basic generation parameters of the sub-families containing these problems can be seen in Table 3.2, while the individual characteristics of the problems appear in Tables A.1 and A.2 of the Appendix. Obviously, Glover’s and Beasley’s maximization problems have been first converted to minimization problems in order to make the direct application of our proposed procedures possible.

### 7.5.2 Maximum cliques of graphs

In order to facilitate comparisons among different exact and heuristic methods related to clique (see definition in Section 2.2.1) problems, a set of benchmark graphs has been constructed in conjunction with the 1993 DIMACS Challenge on maximum cliques, coloring and satisfiability ([151]). This chapter only reports preprocessing results for two families of this dataset, since for the other graphs PREPRO did not find any persistencies. Namely, we consider the benchmarks containing  $c$ -fat graphs or Hamming graphs. The graphs analyzed here are described in Section 3.2 and their main characteristics are listed in Table 3.7.

### 7.5.3 Minimum vertex cover problems of planar graphs

Motivated by a recent work by Alber, Dorn and Niedermeier [13] we have analyzed the performance of PREPRO for preprocessing minimum vertex cover problems in the class of planar graphs. A major difference between the two approaches is that the method of Alber, Dorn and Niedermeier considers “the influence of a clever, VERTEX COVER-specific data reduction” (see [13], page 220), whereas the results obtained by PREPRO are entirely due to its ways of simplifying QUBOs, since we have not introduced any specific adaptation of this method for the case of vertex cover problems.

In the computational experiments we considered 400 planar graphs randomly generated by the LEDA software package (see details in Section 3.3), whose general characteristics are similar to those graphs generated by [13] (see Table 3.11).

#### 7.5.4 Graphs for MAX-CUT

The MAX-CUT values of several classes of graphs is considered in the PREPRO experiments. Let us remark that any optimal solution  $\mathbf{x} = (x_1, \dots, x_n)$  of the MAX-CUT problem has a complementary solution  $(\bar{x}_1, \dots, \bar{x}_n)$ . Thus, before calling PREPRO for a MAX-CUT problem, we select a variable and assign to it a 0–1 value.

Let us note that these problems are maximization problems (see Section 2.2.3). Therefore, we transformed the MAX-CUT problems (2.5) into quadratic posiform minimization problems.

We tested the PREPRO algorithm on the following graphs, previously described in Section 3.4: torus graphs ([145]; see Table 3.15), via and sparse random graphs ([145]; see Table 3.16), and  $Gn.p$  and  $Un.p$  graphs ([157]; see Table 3.17).

#### 7.5.5 MAX-2-SAT formulas

Algorithm PREPRO was tested in the set of random weighted and non-weighted MAX-2-SAT formulas (see Section 2.3) proposed by Borchers and Furman [47]. The list [47] contains 17 standard formulas and 17 weighted formulas, which were described previously in Section 3.5 (see also Tables 3.19 and 3.20). We solve the (weighted) MAX-2-SAT problem by associating to it a quadratic posiform  $\phi$  (see (2.7)), for which the minimum value  $\nu(\phi)$  is the minimum weighted set of unsatisfied clauses.

### 7.6 Computational experiments

#### 7.6.1 Test environment

The algorithm PREPRO was implemented in C++, compiled using the Microsoft Windows 32-bit C/C++ Optimizing Compiler (version 12) for 80x86, and linked with the Microsoft Incremental Linker (version 6).

All the computational experiments of this chapter were carried out on a computer with a 2.80 GHz Pentium 4, 512 MB of RAM, hyper-threading technology, and has installed the Windows XP Professional (version 2002) operating system.

### 7.6.2 Results

Given a posiform  $\phi$ , its roof-dual bound, a set of strong/weak persistencies and a decomposition of type (7.5) of it, can easily be derived from the residual network resulting after applying a maximum flow algorithm to the implication network associated to  $\phi$ . We consider this preprocessing step (entirely based on the roof-duality theory) as a *standard* preprocessing tool in all experiments that we have carried out. We tested four possible preprocessing strategies:

- A** Only the standard tool is considered;
- B** Standard tool and coordination are considered;
- C** Standard tool and probing are considered; and
- D** All preprocessing tools are considered.

Since strategy **D** usually provides the best data reduction of the problems, we have included in Appendix C several statistical results about the preprocessing performance of this strategy in all benchmarks.

At first glance, we have tried to understand how any of the preprocessing techniques would impact in the reduction of the problem's size. Table 7.1 provides preprocessing results for the test beds, whose values are averages for groups of problems belonging to the same family. Strategy **D** provides 100% data reduction for the following benchmarks: MAX-Clique problems in all HAM-2 graphs and all  $c$ -FAT graphs; minimum vertex cover problems in all PVC LEDA planar graphs; MAX-CUT problems in all VIA graphs. These results clearly indicate that one can expect getting an outstanding data reduction level in these special well structured problems.

It should be remarked that the border separating successful from unsuccessful preprocessing cases is very thin. For instance, all the Hamming graphs in HAM-2 were

optimally solved with the standard preprocessing tool. However, in the closely related family HAM-4 there was no reduction found for any of the graphs, even when all the preprocessing tools were considered. We also remark the fact that strategy **C** provided optimality for all MAX-Clique problems and all MAX-CUT problem in the VIA graphs, using a substantial smaller computing time than the one corresponding to strategy **D**. Strategy **C** with an average value of 99.9%, provided also a very good data reduction on the minimum vertex cover problems in the PVC LEDA planar graphs.

Table 7.2 suggests the particular preprocessing techniques which can be recommended for each of the problem families considered, in order to achieve (on the average) as high a data reduction as possible within a limited amount of time. In view of the relatively uniform behavior of problems within the same family, the recommendation of a common strategy for problems within a family seems reasonable. Here are some remarks and some recommendations for the examined groups:

- *Problems with prescribed density* – Coordination does not have a practical influence in the preprocessing results. Probing should be used in the cases where density is low. In general, the probing tool should be used in this class, if the condition  $nd \leq 20$  is satisfied. Let us also note that family  $B$  consists of very dense submodular maximization problems, and for which the preprocessing outcome changed considerably, in comparison with the other problems. Six of the 10 problems in the  $B$  group were solved optimally, and for the unsolved cases, a large number of quadratic persistencies was found.
- *Minimum vertex cover of planar graphs* – Probing when used with the *coordination* method provides slightly better preprocessing data reduction, without degrading the computing times returned by probing only.
- *MAX-CUT in torus graphs* – The standard preprocessing tool should be used for the graphs with  $\pm 1$  weights in the edges (see also Table C.5). Probing should be used in the other weighted graphs.
- *MAX-CUT in VIA graphs* – All the problems in the VIA.CY family are solved optimally by the basic preprocessing tool (see also Table C.5). Every instance in

Table 7.1: Average QUBO simplifications and decomposition after preprocessing.

		Preprocessing Tools Used:													
Type of Problem	Family Name	Roof-Duality		Roof-Duality and Coordination				Roof-Duality and Probing				ALL Tools			
		Total Time	Variab. Reduc.	Total Time	Relat. Gap	Quad. Rel.	Variab. Reduc.	Total Time	Relat. Gap	Quad. Rel.	Variab. Reduc.	Total Time	Relat. Gap	Quad. Rel.	Variab. Reduc.
Fixed Degree	A	0.0	52.3%	0.0	8.2%	1	52.9%	0.0	5.1%	1	64.0%	0.0	5.1%	1	64.0%
	B	0.0	0.9%	0.1	85.3%	1762	0.9%	1.2	34.9%	785	72.2%	2.9	41.9%	801	68.2%
	C	0.0	18.1%	0.0	22.4%	0	18.5%	0.1	19.8%	1	30.2%	0.1	19.8%	1	30.2%
	D	0.0	0.6%	0.0	56.8%	0	0.6%	1.2	55.4%	3	1.6%	1.2	55.4%	3	1.6%
	E	0.0	0.0%	0.1	57.9%	0	0.0%	5.2	57.2%	6	0.0%	5.3	57.2%	6	0.0%
	F1	1.1	0.0%	1.6	78.7%	0	0.0%	149.8	78.6%	0	0.0%	150.1	78.6%	0	0.0%
	B-50	0.0	94.2%	0.0	0.2%	2	94.4%	0.0	0.0%	0	100.0%	0.0	0.0%	0	100.0%
	B-100	0.0	4.9%	0.0	13.3%	0	5.2%	0.3	8.8%	26	36.3%	0.4	8.8%	26	36.3%
B-250	0.0	0.0%	0.1	44.1%	0	0.0%	3.3	43.5%	1	0.0%	3.4	43.5%	1	0.0%	
B-500	0.1	0.0%	0.4	60.6%	0	0.0%	28.4	60.3%	0	0.0%	28.5	60.3%	0	0.0%	
MAX Clique	C-FAT-200	0.0	0.0%	1.3	68.7%	29	0.0%	4.8	0.0%	0	100.0%	14.1	0.0%	0	100.0%
	C-FAT-500	0.1	0.0%	21.9	77.0%	56	0.0%	80.8	0.0%	0	100.0%	327.4	0.0%	0	100.0%
	HAM-2	0.0	100.0%	0.0	0.0%	0	100.0%	0.0	0.0%	0	100.0%	0.0	0.0%	0	100.0%
	HAM-4	0.1	0.0%	12.8	89.6%	0	0.0%	79.0	88.3%	0	0.0%	91.5	88.3%	0	0.0%
MIN Vert. Cov.	LEDA-1000	0.0	75.3%	0.1	0.0%	0	99.9%	0.1	0.0%	0	99.9%	0.1	0.0%	0	100.0%
	LEDA-2000	0.1	74.5%	0.2	0.0%	0	99.9%	0.2	0.0%	0	99.8%	0.2	0.0%	0	100.0%
	LEDA-3000	0.2	75.0%	0.3	0.0%	0	99.9%	0.3	0.0%	0	99.8%	0.3	0.0%	0	100.0%
	LEDA-4000	0.4	75.1%	0.5	0.0%	0	99.8%	0.6	0.0%	0	99.9%	0.5	0.0%	0	100.0%
MAX Cut	Torus	0.1	0.2%	6.4	39.2%	0	0.2%	220.0	38.3%	2	3.1%	853.3	38.3%	2	3.1%
	R	0.4	0.0%	64.7	28.9%	0	0.0%	1818.1	28.7%	1	0.2%	3187.8	28.7%	1	0.2%
	VIA.CN	0.1	4.0%	1.1	4.6%	3	4.0%	13.2	0.0%	0	100.0%	100.5	0.0%	0	100.0%
	VIA.CY	0.1	100.0%	0.1	0.0%	0	100.0%	0.1	0.0%	0	100.0%	0.1	0.0%	0	100.0%
	G500	0.0	4.2%	0.5	23.1%	3	4.2%	6.2	22.3%	5	19.0%	11.8	22.3%	5	19.0%
	G1000	0.1	2.7%	2.2	23.5%	5	2.7%	36.1	23.0%	7	18.8%	100.8	23.0%	7	18.8%
	U500	0.0	1.0%	2.6	35.2%	31	2.6%	9.1	36.0%	0	5.7%	14.3	33.8%	32	11.5%
U1000	0.1	0.6%	14.7	35.2%	57	2.5%	39.0	35.9%	0	7.6%	76.2	34.2%	83	9.9%	
MAX 2-Sat	BF-50	0.0	9.3%	0.0	240.0%	1	11.3%	0.1	121.2%	8	25.8%	0.1	121.2%	8	25.8%
	BF-100	0.0	12.6%	0.0	691.1%	1	13.8%	0.1	271.9%	24	27.6%	0.2	271.9%	25	27.6%
	BF-150	0.0	17.1%	0.0	908.1%	0	17.1%	0.3	258.3%	55	42.2%	0.5	258.3%	55	42.2%
	BFW-50	0.0	6.2%	0.0	391.4%	1	6.2%	0.1	133.2%	8	24.2%	0.1	133.2%	8	24.2%
	BFW-100	0.0	14.2%	0.0	1731.4%	5	14.2%	0.2	335.5%	39	26.2%	0.3	335.5%	39	26.2%
	BFW-150	0.0	17.1%	0.0	3214.3%	4	17.1%	0.7	263.1%	68	45.6%	0.9	265.2%	67	45.8%

Table 7.2: Preprocessing strategies recommended for the benchmarks.

<i>Type of Problem</i>	<i>Family Name</i>	<i>Number of Instances</i>	<i>Density</i>	<i>Best Strategy</i>
Fixed Degree	A	8	18.78%	<b>C</b>
	B	10	98.83%	<b>C</b>
	C	7	36.01%	<b>C</b>
	D	10	54.31%	<b>A</b>
	E	5	29.62%	<b>A</b>
	F1	5	51.56%	<b>A</b>
	B-50	10	9.75%	<b>C</b>
	B-100	10	9.75%	<b>C</b>
MAX Clique	B-250	10	9.94%	<b>A</b>
	B-500	10	9.90%	<b>A</b>
	C-FAT-200	3	77.82%	<b>C</b>
	C-FAT-500	4	83.28%	<b>C</b>
MIN Vertex Cover	HAM-2	3	4.55%	<b>A</b>
	HAM-4	3	39.42%	<b>A</b>
	LEDA-1000	100	0.41%	<b>D</b>
	LEDA-2000	100	0.20%	<b>D</b>
MAX Cut	LEDA-3000	100	0.14%	<b>D</b>
	LEDA-4000	100	0.10%	<b>D</b>
	Torus	4	0.68%	<b>C</b>
	R	8	0.34%	<b>C</b>
	VIA.CN	5	0.32%	<b>C</b>
	VIA.CY	5	0.37%	<b>A</b>
	G500	4	1.87%	<b>C</b>
	G1000	4	0.95%	<b>C</b>
MAX 2-SAT	U500	4	3.40%	<b>D</b>
	U1000	4	1.72%	<b>D</b>
	BF-50	9	19.98%	<b>C</b>
	BF-100	5	7.53%	<b>C</b>
	BF-150	3	3.91%	<b>C</b>
	BFW-50	9	21.18%	<b>C</b>
BFW-100	5	7.77%	<b>C</b>	
BFW-150	3	3.93%	<b>C</b>	

this group of problems is solved in less than 0.2 seconds. The VIA.CN problems are all optimally solved with probing, taking an average computing time of 13.2 seconds. In two of the VIA.CN problems, the analysis of the starting implication network found 2 components which were preprocessed separately under the result of Theorem 7.5.

- *MAX-CUT in  $Gn.p$  graphs* – The preprocessing efficiency decreases with density for the graphs with the same number of vertices (see also Table C.5). Probing helped increasing data reduction for graphs with densities up to 5%, and attained “as expected” better performance for graphs with 500 vertices, than for those with 1000 vertices.
- *MAX-CUT in  $Un.p$  graphs* – The preprocessing efficiency decreases with density for the graphs with the same number of vertices (see also Table C.5). In this category, the standard preprocessing tool found some non trivial decomposition, and both coordination and probing helped improving the average data reduction rates from 2–3% to about 11%.
- *MAX-2-SAT* – The preprocessing efficiency decreases with the number of clauses when the number of variables is fixed (see also Table C.6). Both in the non-weighted and weighted formulas, the probing technique provided better reduction indicators.

In conclusion it can be seen that the choice “best strategy” is highly problem family dependent. It should also be remarked that only three out of the four examined strategies turn out to provide the “best” performance for some of the considered group of problems; strategy **B** (consisting of the application of the standard tool and coordination, but not of probing) did not give best results in any of the examined cases. Table 7.2 indicates the best recommended strategies for each of the examined families of problems.

## 7.7 Optimal vertex covers of planar graphs

In view of the outstanding results obtained by applying PREPRO (described in Figure 7.3) to the minimum vertex cover problem in random planar graphs, we have tried to refine this method to the point where it would not only preprocess the problem but actually produce an optimal solution of it. As it will be seen in this section, the resulting method allowed the efficient detection of minimum vertex covers in planar graphs of impressive dimensions, including some having 500 000 vertices.

Although the vertex cover problem is known to be NP-hard in the class of planar graphs ([105]), our computational experiments with a large collection of benchmark planar graphs indicate that, in all likelihood, finding vertex covers in planar graphs may be frequently tractable. This conclusion provides the motivation for the work reported in this section.

Before presenting the results of our computational experiments we would like to emphasize that PREPRO is not capable of solving the QUBO problems associated to *every* planar graph, and that it may encounter problems even in the case of very small graphs. For example there are no persistencies in the QUBO associated to the “toy box” graph of Figure 7.4.

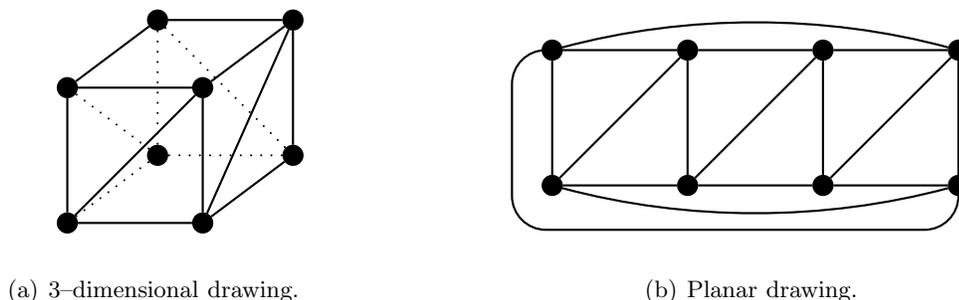


Figure 7.4: Planar graph for which PREPRO does not find any persistent result.

### 7.7.1 Deriving minimum vertex cover from QUBO’s optimum

We have seen in Section 2.2.2 that finding a minimum vertex cover of a graph  $G = (V, E)$  is a QUBO, and we have also noticed in Section 7.6.2 that out of the 400 QUBOs coming

from vertex cover problems in randomly generated *planar* graphs, every single QUBO was solved to optimality by PREPRO. The only matter which needs special attention is that – due to the fact that in (2.2) we have fixed to zero the values of  $\epsilon_{(i,j)}$  for every edge  $(i,j)$  – it may happen that the optimal 0–1 solution of a QUBO defines a vertex set which does not cover every edge. Let us see next that there is a simple polynomial time transformation, which associates to the optimal solution of QUBO an optimal vertex cover of  $G$ . This result follows directly from Theorem 2.1.

**Corollary 7.1.** *Let  $G = (V, E)$  be a graph, and let us associate to it the quadratic pseudo-Boolean function  $f(x_1, \dots, x_n) = \sum_{i \in V} x_i + \sum_{(i,j) \in E} \bar{x}_i \bar{x}_j$  and the QUBO (2.2). Let further  $f(x_1^*, \dots, x_n^*)$  be a minimum of  $f$ , and let  $S^*$  be the set of vertices  $j \in V$  for which  $x_j^* = 1$ . Then, the size of a minimum vertex cover is  $f(x_1^*, \dots, x_n^*)$ , and the set  $S^*$  can be enlarged to a minimum vertex cover  $\hat{S} \supseteq S^*$  in  $O(|E|)$  time.*

While the previous result holds in any graph, it is particularly useful in classes of graphs for which the associated QUBO (2.2) can be solved to optimality.

As a consequence of the discussion above, we have supplemented PREPRO with the simple procedure outlined in the proof of Theorem 2.1 to derive an optimal vertex cover from the optimal solution of the corresponding QUBO problem (2.2). This amended version of the proposed algorithm will be called PREPRO<sup>+</sup>.

In this section the preprocessing strategies **A**, **B** and **C** of the previous section will not be considered, i.e. all the experiments below were carried out by using strategy **D**, which involves all the preprocessing tools of Section 7.6.2.

The results obtained by PREPRO<sup>+</sup> for moderately sized graphs (i.e. having up to a few thousand vertices), have been compared with those of the recent paper of Alber, Dorn and Niedermeier (or ADN in short) reported in [13], which is essentially based on the data reduction results of Nemhauser and Trotter [182].

### 7.7.2 Preprocessing

Table 7.3 provides averages of results obtained by preprocessing minimum vertex cover problems on 400 random planar graphs generated by the LEDA package (see Section

7.5.3). Four groups of 100 graphs each have been considered, each graph in these sets containing respectively 1000, 2000, 3000 and 4000 vertices.

Table 7.3: Comparative preprocessing results for minimum vertex cover problems in planar graphs.

<i>Number of</i>		<i>Time</i>		<i>Variables Fixed</i>		<i>Size of Residual</i>	
<i>Graphs per Family</i>	<i>Vertices per Graph</i>	<i>(sec)</i>		<i>(%)</i>		<i>Problem</i>	
		<i>ADN ([13])</i>	<i>PREPRO</i>	<i>ADN ([13])</i>	<i>PREPRO</i>	<i>ADN ([13])</i>	<i>PREPRO</i>
100	1 000	4.06	0.05	68.4	100	315.8	0
100	2 000	12.24	0.16	67.4	100	652.9	0
100	3 000	30.90	0.27	65.5	100	1 036.1	0
100	4 000	60.45	0.53	62.7	100	1 492.9	0

Remarkably, PREPRO achieved 100% data reduction in all PVC LEDA graphs, whereas the ADN method obtained between 63% and 68% average data reduction on their LEDA benchmarks, which have similar characteristics to the PVC LEDA graphs (see Table 3.11). It can also be seen that the best performance of the ADN method (68.4% reduction of vertex set) occurs on the group of relatively smaller graphs, while the performance of PREPRO (100% reduction of vertex set) does not seem to be influenced by the size of the graphs considered.

### 7.7.3 Optimization

While the results reported in Table 7.3 refer to the preprocessing by PREPRO of the minimum vertex cover problem, we shall discuss below the results of applying PREPRO<sup>+</sup> for actually finding optimal solutions to this problem.

It is important to remark that PREPRO<sup>+</sup> assumes the knowledge of the exact optimum of the associated QUBO. If this optimum is not available PREPRO<sup>+</sup> is further enhanced to an algorithm PREPRO\*, by adding a branch-and-bound component to it, in order to handle minimum vertex cover problems even in this case. However, the use of PREPRO\* turned out not to be necessary in any of the 400 test problems randomly generated with the LEDA software package, which were all solved to optimality without the branch-and-bound component having been called.

Table 7.4 provides comparative results for finding optimal vertex covers for graphs

Table 7.4: Average computing times of optimal vertex covers for graphs belonging to the LEDA benchmarks.

Algorithm	ADN ([13])	PREPRO* in the PVC LEDA Benchmark	
Computer System (speed)	750 MHz Linux 720 MB RAM	500 MHz Pentium III Windows 98 96 MB RAM (slower)	2.8 GHz Pentium 4 Windows XP 512 MB RAM (faster)
1 000 vertices	5.75 s	0.24 s	0.06 s
2 000 vertices	19.93 s	0.64 s	0.18 s
3 000 vertices	51.54 s	1.07 s	0.31 s
4 000 vertices	109.84 s	1.71 s	0.56 s
<b>Average Speedup</b>		51 times	169 times

belonging to the LEDA benchmarks. It includes computing times for the exact algorithm of Alber, Dorn and Niedermeier [13] and solution times for PREPRO\* (which coincide with PREPRO<sup>+</sup> for all the 400 cases). We would like to recall the fact that – not having had access to the test problems of [13] – we have randomly generated our problems, but made sure (as explained in Section 7.5.3) that the parameters used in the random graph generation process were chosen so as to match exactly does of [13].

In order to be able to differentiate between the acceleration due to computer systems and those due to algorithms, all the experiments reported in Table 7.4 have been carried out twice, first on a somewhat slower computer system (500 MHz Pentium III, 98 MB RAM, Windows 98) than the one used by [13] (715 MHz, 720 MB RAM, Linux), and second on a faster system (2.8 GHz Pentium 4, 512 MB RAM, Windows XP).

The basic conclusion of this set of experiments is that using the slower computer system, PREPRO\* is about 50 times faster than that of [13], on average.

#### 7.7.4 Minimum vertex covers of very large planar graphs

Based on the high efficiency of PREPRO\* when applied to the optimization of vertex cover problems in planar graphs, we have investigated the possibility of using it on substantially larger planar graphs. The relevant experiments were carried out on the set of 36 benchmark problems contained in the RUDY list (described in Section 3.3; see Table C.4 of the Appendix), which contains graphs whose vertex sets contain 50 000, 100 000, 250 000 and 500 000 vertices, and have planar densities of 10%, 50% and 90%.

For each particular number of vertices and each density the list contains three graphs.

Table 7.5: Average computing times over 3 experiments of optimal vertex covers for graphs belonging to the PVC RUDY benchmark.

<i>Vertices</i>	<i>Planar Density</i>		
	10%	50%	90%
50 000	1.2 min	3.7 min	1.8 min
100 000	4.8 min	16.2 min	7.4 min
250 000	30.4 min	107.7 min	48.2 min
500 000	124.7 min	422.4 min	195.3 min

Table 7.5 presents the average computing times needed by PREPRO\* for finding minimum vertex cover sets in *all* the graphs contained in the RUDY list. Each of the computing times reported in the table represents the average needed for solving the three problems with a fixed number of vertices and a fixed planar density contained in the RUDY list. The average computing times range from 2.2 minutes for the graphs with 50 000 vertices up to 4.1 hours for the graphs with 500 000 vertices. Clearly, the computing times vary with the size of the vertex set. A similarly foreseeable phenomenon happens with the dependency of computing times and densities. Indeed, the average computing time for the low density graphs is 40 minutes, for medium density graphs this increases to 2.3 hours, and for high density graphs it drops to 1 hour.

More detailed information about the performance of PREPRO can be read from the statistics presented in Table C.4 in the Appendix, where specific data are given for each of the 36 problems of the RUDY list. First, it can be seen that almost all of the computing time (78.7%) is spent on calculating the roof duals; moreover, most of this time (99.9%) is spent on calculating the very first roof dual.

The large investment of computing time in the calculation of roof duals brings returns in the form of graph size reductions (which are due to strong and weak persistency) and in the form of decompositions.

- The detailed analysis of the problem size reductions occurring in PREPRO shows that roof-duality accounts for 99.8% of these reductions for planar graphs of density 10%, 93.2% for the 50% dense graphs, and 51.8% for the 90% dense graphs.

- It is interesting to note the extremely small size of the average components of the graphs left after applying decomposition and strong and weak persistency. Indeed, the average size of these components for graphs of 10%, 50% and 90% density is of 3.1, 4.4 and 14.4 vertices, respectively.

Beside roof–duality, important simplifications of the remaining QUBOs were obtained by the coordination method and by probing. It can be seen in column  $(n_e)$  of Table C.4 of the Appendix that the number of equality relations between pairs of variables or their complements, discovered by the coordination method is an increasing monotone function of planar density. Also, column  $(n_f)$  of Table C.4 shows that the number of variables whose values are fixed by probing reaches maximum values for the medium density graphs. In conclusion it can be seen that there is substantial complementarity in the effect of applying the basic preprocessing techniques considered in this paper. Indeed,

- 10% dense planar graphs derive almost the entire solution from the application of roof–duality;
- 50% dense planar graphs derive a considerable reduction through probing; and
- 90% dense planar graphs derive a considerable reduction through the coordination method.

However, the most important conclusion is that PREPRO<sup>+</sup> found optimal vertex covers for all the 36 benchmarks in the RUDY list.

## 7.8 Final remarks

This study is devoted to the systematic simplification of QUBOs. The proposed method uses enhanced versions of several basic techniques (e.g., extraction of conclusions from the analysis of first and second order derivatives [121], and from roof–duality [123]) and several integrative techniques (e.g., probing, consensus) for combining the conclusions provided by the basic techniques. The application of these techniques is implemented using the network flow model of [54, 56].

The use of the proposed preprocessing techniques provides:

- (i) A lower bound of the minimum of the objective function;
- (ii) The values of some of the variables in some or every optimum;
- (iii) Binary relations (equations, inequalities, or non-equalities) between the values of certain pairs of variables in some or every optimum;
- (iv) The decomposition (if possible) of the original problem into several smaller pairwise independent minimization problems.

The efficiency of the proposed techniques is demonstrated through numerous computational experiments carried both on benchmark problems and on randomly generated ones.

The simplifications obtained with the proposed methods exceed substantially those reported in the literature. An interesting example is the minimum vertex cover problem for which [13] reports a preprocessing stage reduction of dimensionality by 62.7%–68.4%, while the method proposed here achieves 100% reduction (i.e. exact solution) in each of the test problems. Moreover, while the computing times reported in [13] for finding optimal vertex covers for graphs from 1 000 to 4 000 vertices range from 5.75 to 109.84 seconds, those required by the proposed method range from 0.24 to 1.71 seconds using a somewhat slower computer, or from 0.06 to 0.56 seconds using a somewhat faster one.

The experiments show that the methods can be applied successfully to problems of unusually large size, for instance:

- MAX-Clique on graphs derived from fault diagnosis having up to 500 vertices;
- MAX-CUT problems on graphs derived from VLSI design having thousands of vertices;
- Minimum vertex cover problems on randomly generated planar graphs (an NP-hard problem [105]) having up to 500 000 vertices (reported in Section 7.7).

It should be added that the proposed preprocessing technique have not only simplified the above problems but have in fact produced their exact optimum solutions. As

far as we know there are no reports in the literature about methods capable of providing optimal solutions to vertex cover problems in planar graphs with the investigated sizes.

## Chapter 8

### Lower Bounds to the Minimum

Given any (heuristic) solution  $\mathbf{x}^+$  to the minimum of a quadratic pseudo-Boolean function  $f$ , it is desirable to know how far is  $f(\mathbf{x}^+)$  from the optimum  $\nu(f)$ . Since  $\nu(f)$  is most likely not known for most quadratic pseudo-Boolean functions  $f$ , then the quality of  $\mathbf{x}^+$  as a minimizer of  $f$  is typically analyzed by comparing how far is  $f(\mathbf{x}^+)$  from an “easy” computable *lower bound* to  $\nu(f)$ .

It is well known that contrary to several heuristics, which provide reasonably good solutions (i.e. upper bounds) in an “efficient” computing time to QUBO, the lower bound “closeness” to the optimum is traditionally associated to a much larger computer effort. Therefore, when selecting a lower bound technique, one has to trade-off between the desired quality and the time needed to compute the bound. This fact will become evident throughout the computational results shown subsequently in this and in the following chapters.

Lower and upper bounds are also crucial elements in the design framework of the state-of-the-art exact methods for QUBO (see Chapter 9). Typically, bounding is used by branch-and-bound methods to cutoff the solutions space as much as possible.

Probably, the best known lower bound to QUBO is the *roof dual* bound of Hammer et al. [123] (see Chapter 5). In addition to the value of the bound, this technique characterizes some strong persistent values for some variables, which hold in all optimal solutions ([123]), thus simplifying the problem at hand.

It turns out that the roof dual bound can be determined by using several alternative algorithm approaches. The most well known approach is based on solving the Linear Program (LP) (5.3), which is obtained by linearizing each quadratic term  $x_i x_j$  through the use of auxiliary variables  $y_{ij}$  for every  $\mathbf{x} \in \mathbb{B}^n$  ( $1 \leq i < j \leq n$ ). The roof dual can

also be (efficiently) computed by finding a symmetric maximum flow in the network model of ([51, 59, 226]) (see Section 5.3).

Let

$$L_f(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} c_0 + \sum_{i=1}^n c_i x_i + \sum_{1 \leq i < j \leq n} c_{ij} y_{ij}.$$

The 0-1 LP (5.3) can be rewritten as

$$\nu(f) = \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{S}^{[2]}, \mathbf{x} \in \mathbb{B}^n \right\}, \quad (8.1)$$

where

$$\mathbf{S}^{[2]} = \left\{ (\mathbf{x}, \mathbf{y}) \mid \begin{array}{rcl} & -y_{ij} & \leq 0, \\ -x_i & +y_{ij} & \leq 0, \\ & -x_j + y_{ij} & \leq 0, \\ x_i + x_j & -y_{ij} & \leq 1, \end{array} \quad (1 \leq i < j \leq n) \right\}.$$

The roof dual bound of quadratic pseudo-Boolean function  $f$  (denoted here as  $C_2(f)$ ) is obtained by relaxing the integrality constraints on  $\mathbf{x}$  in (8.1) to  $\mathbf{x} \in \mathbb{U}^n$ , i.e.

$$C_2(f) \stackrel{\text{def}}{=} \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{S}^{[2]}, \mathbf{x} \in \mathbb{U}^n \right\}. \quad (\text{ROOF DUAL})$$

Let us note that the (ROOF DUAL) LP is characterized by  $n + \binom{n}{2}$  nonnegative variables and by  $3\binom{n}{2}$  constraints.

Boros et al. [49] have presented a hierarchy of bounds  $C_2(f) \leq C_3(f) \leq C_4(f) \leq \dots \leq C_n(f) = \nu(f)$  for QUBO.  $C_2(f)$  corresponds to the roof dual value of  $f$ . The  $C_3$  bound, also known as the *cubic dual*, has been shown to be equal to the first Chvátal closure of  $\mathbf{S}^{[2]}$  ([50]). The present chapter is mostly devoted to studying alternative approaches that provide lower bounds to a quadratic pseudo-Boolean function  $f$  that have values between the roof dual and the cubic dual bounds.

$C_4(f)$  is characterized here for the first time, and we provide some combinatorial constructs to get improved bounds based on the so-called *arithmetic consensus* of two terms of a posiform.

Other than by solving an LP, it is not known if there is any combinatorial algorithm that could provide the cubic dual in polynomial time. Several combinatorial bounding approaches were proposed in the past to get better lower bounds than the roof dual bound. Any of these methods returns the roof dual bound when the function is gap-free, otherwise they return a strictly better bound than roof dual, with the first two cases being not better than the cubic dual of the function. A brief description and the main references of these bounding procedures are listed next:

- Boros and Hammer [52] introduced the iterated roof dual bound, which is based on solving a noose packing problem of a graph representation of the problem (called the *biform* graph). This bound can be efficiently computed by finding a sequence of maximum flows in a capacitated bi-form network.
- Using a different approach based on a consensus identity, Bourjolly et al. [65, 67] proposed a bound which partitions the function into two posiforms, the first one being an arbitrary quadratic posiform, and the second being a cubic positive posiform. There is not known reference about a comparative study between the bound of [65, 67] and the iterated roof dual bound.
- Billionnet and Sutter [44] proposed a method to find a *quartic* posiform of a given quadratic pseudo-Boolean function. Clearly the constant of this posiform is a lower bound to the minimum of the function. To do this operation, several minimum weight cycles are squeezed out of the residual positive posiform (similar to the ones proposed in [65, 67]). An additional improvement will produce the residual quartic posiform. There is not known relation between this bound and the hierarchy of bounds of Boros et al. [49].

It is well known that the MAX-CUT problem is equivalent to the maximization of a quadratic pseudo-Boolean function ([52]). Goemans and Williamson [112] proposed the *semidefinite* relaxation for MAX-CUT, and due to the 1-to-1 correspondence between these two problems, the semidefinite relaxation provides a lower bound to QUBO as well.

Recently, Rendl et al. [98] proposed to use Lagrangian duality theory, by using both the semidefinite relaxation and by dualizing a subset of the set of triangle inequalities, which are valid for the cut polytope, and whose complete set would provide the cubic dual of the function. The use of these additional inequalities provided a substantial reduction to the semidefinite relaxation gap of several MAX-CUT benchmarks ([206]).

Billionet and Elloumi [41] used semidefinite programming and used the fact that  $x_i^2 - x_i = 0$  (for all  $i = 1, \dots, n$ ) to produce an equivalent unconstrained quadratic optimization problem to QUBO, which is no longer multilinear, but whose relaxation is *convex*. The bound produced by solving this relaxation is the same as the semidefinite relaxation of [112]. The advantage of using this perturbed problem is the fact that the 0–1 version of this problem can be optimized by using other solvers, which have algorithms ready-to-solve these convex Integer Quadratic Problems (known as MIQP).

Another bounding approach to QUBO is based on the decomposition method of Chardaire and Sutter ([82]; see also [95]). This method partitions the original function into *bilinear* functions, for which the optimum can be obtained in reasonable time. The dual formulation of this decomposition provides at least the roof dual bound.

In order to solve QUBO, many researchers use an equivalent linear integer formulation. This formulation is obtained by doing *linearizations* of quadratic sub-expressions, through the use of additional variables and linear constraints (see e.g. [63]). Obviously, the relaxation of these integer programs produces a lower bound to the optimal value. It turns out that the bounds obtained from several linearization approaches are related to one of the bounds in the Boros et al. [49] hierarchy. In the related literature, the focus of these approaches has been given to find linearizations which require a small number of additional variables. A recent work by Gueye and Michelon [115] reviews and analyzes this aspect.

The operation that finds if the roof dual bound is gap free (i.e. if  $\nu(f) = C_2(f)$ ) can be done in polynomial time ([123]). This problem can be efficiently reduced (in size and time) to a 2-satisfiability problem, which is well known for having a polynomial running time (see e.g. [90, 133]).

If the roof dual bound is *not* gap free then  $\nu(f) > C_2(f)$ , and therefore in this case it is useful to study and analyze other approaches that can *strictly* reduce the roof dual gap  $\nu(f) - C_2(f)$ . In the hierarchy of bounds proposed by Boros et al. [49], the cubic dual bound (i.e.  $C_3(f)$ ) is such an example.

The cubic dual is well characterized by means of LP ([50]). It is simply defined by (ROOF DUAL) intersected with a family of valid cuts (called triangle inequalities) denoted by  $\mathbf{S}^{[3]}$ , which is produced by the first Chvátal closure of  $\mathbf{S}^{[2]}$ . Boros et al. [50] have shown that  $\mathbf{S}^{[3]}$  is characterized by  $3\binom{n}{2} + 4\binom{n}{3}$  inequalities as

$$\mathbf{S}^{[3]} = \mathbf{S}^{[2]} \cup \left\{ (\mathbf{x}, \mathbf{y}) \left| \begin{array}{cccc} x_i & +x_j & +x_k & -y_{i,j} - y_{i,k} - y_{j,k} \leq 1, \\ -x_i & & & +y_{i,j} + y_{i,k} - y_{j,k} \leq 0, \\ & -x_j & & +y_{i,j} - y_{i,k} + y_{j,k} \leq 0, \\ & & -x_k & -y_{i,j} + y_{i,k} + y_{j,k} \leq 0, \end{array} \right. \quad (1 \leq i < j < k \leq n) \right\}. \quad (8.2)$$

The cubic dual of a quadratic pseudo-Boolean function  $f$  can therefore be found by solving the LP

$$C_3(f) \stackrel{\text{def}}{=} \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{S}^{[3]}, \mathbf{x} \in \mathbb{U}^n \right\}, \quad (\text{CUBIC DUAL})$$

whose size is polynomially bounded by the size of  $f$ .

Other than using LP algorithms, there is not other known (combinatorial) method that can provide the cubic dual in efficient time, in contrast with the roof dual bound which is well characterized by using network flow procedures. In Section 8.5 we shall introduce some ideas that may lead to the discovery of such a method for the cubic dual. In the next section, we give particular emphasis to problem (ROOF DUAL) whose constraint set is augmented with various sub-families of inequalities from  $\mathbf{S}^{[3]}$ . We will demonstrate that a state-of-the-art solver for LP can solve relatively large sparse QUBOs efficiently, which appear frequently in practical applications.

This chapter is organized as follows. The next section will introduce bi-forms and the relation between graph balancing and QUBO. Section 8.2 will show how to relate the graph balancing graph with the network model introduced earlier in Section 5.3.

In Section 8.4, the iterated roof-dual bound is reviewed and an implementation of it based on the network model is given and demonstrated. The following two sections propose improved bounds over the iterated roof-dual one. These methods are based on combinatorial constructs and network flows. The last 2 sections of this chapter cover the usefulness of linear programming augmented with some families of cuts to solve sparse QUBOs.

## 8.1 Bi-forms and packing of cycles

A quadratic pseudo-Boolean function  $f \in \mathcal{F}_2$  may be represented by infinitely many posiforms. Among the posiforms representing  $f$  there may also exist some having degrees higher than two. For instance,  $1 - x - y - z + xy + xz + yz = xyz + \overline{x}\overline{y}\overline{z}$ .

A very special posiform is introduced in this section, which has the peculiar property of being *uniquely* defined for any quadratic pseudo-Boolean function.

**Definition 8.1** ([51]). *If  $x$  and  $y$  are binary variables, then the expression  $x_i\overline{x}_j + \overline{x}_i x_j$  is called a positive bi-term, while the expression  $x_i x_j + \overline{x}_i \overline{x}_j$  is called a negative bi-term.*

Bi-terms naturally express the equality or non-equality of the variables involved, i.e.

$$x_i\overline{x}_j + \overline{x}_i x_j = 0 \Leftrightarrow x = y \text{ and}$$

$$x_i x_j + \overline{x}_i \overline{x}_j = 0 \Leftrightarrow x \neq y.$$

**Definition 8.2** ([51]). *If  $E$  is a collection of bi-terms such that no pair of variables is involved in more than one element of  $E$ , and  $\alpha_e$  are positive numbers for all  $e \in E$ , then the quadratic pseudo-Boolean posiform  $\beta = \sum_{e \in E} \alpha_e e$  is called a bi-form.*

**Proposition 8.1** ([51]). *Any quadratic pseudo-Boolean function  $f$  in variables  $x_1, \dots, x_n$  has a unique constant  $c_f$ , and a unique bi-form  $\beta_f$  in the variables  $x_0, x_1, \dots, x_n$ , such that*

$$f(x_1, \dots, x_n) = c_f + \beta_f(1, x_1, \dots, x_n). \quad (8.3)$$

*Proof.* Let us assume that  $f$  is given as expression (1.5). To prove the claim, we use the fact that all coefficients in (1.5) are uniquely defined for  $f$ . In fact,  $c_f$  and  $\beta_f$  are

uniquely determined by the coefficients:

$$c_f = c_0 - \sum_{\substack{1 \leq i < j \leq n \\ c_{ij} > 0}} \frac{c_{ij}}{2} + \sum_{\substack{i \in \mathbf{V} \\ \Delta_i(\frac{1}{2}, \dots, \frac{1}{2}) < 0}} \Delta_i \left( \frac{1}{2}, \dots, \frac{1}{2} \right) \quad (8.4)$$

and

$$\begin{aligned} \beta_f = & \sum_{\substack{i \in \mathbf{V} \\ \Delta_i(\frac{1}{2}, \dots, \frac{1}{2}) > 0}} \Delta_i \left( \frac{1}{2}, \dots, \frac{1}{2} \right) (x_i x_0 + \bar{x}_i \bar{x}_0) \\ & + \sum_{\substack{i \in \mathbf{V} \\ \Delta_i(\frac{1}{2}, \dots, \frac{1}{2}) < 0}} \left| \Delta_i \left( \frac{1}{2}, \dots, \frac{1}{2} \right) \right| (\bar{x}_i x_0 + x_i \bar{x}_0) \\ & + \sum_{\substack{1 \leq i < j \leq n \\ c_{ij} > 0}} \frac{c_{ij}}{2} (x_i x_j + \bar{x}_i \bar{x}_j) + \sum_{\substack{1 \leq i < j \leq n \\ c_{ij} < 0}} \left| \frac{c_{ij}}{2} \right| (\bar{x}_i x_j + x_i \bar{x}_j). \end{aligned}$$

□

**Example 8.1.** Consider the quadratic pseudo-Boolean function  $g = -f_6$ . The unique bi-form of  $g$  is then

$$\begin{aligned} \beta_g = & 2(\bar{x}_1 x_0 + x_1 \bar{x}_0) + \frac{1}{2}(\bar{x}_2 x_0 + x_2 \bar{x}_0) \\ & + (x_4 x_0 + \bar{x}_4 \bar{x}_0) + \frac{5}{2}(\bar{x}_5 x_0 + x_5 \bar{x}_0) \\ & + \frac{1}{2}(x_1 x_2 + \bar{x}_1 \bar{x}_2) + (x_1 \bar{x}_3 + \bar{x}_1 x_3) + (x_1 x_4 + \bar{x}_1 \bar{x}_4) \\ & + (x_1 \bar{x}_5 + \bar{x}_1 x_5) + \frac{1}{2}(x_1 x_6 + \bar{x}_1 \bar{x}_6) + \frac{1}{2}(x_2 \bar{x}_3 + \bar{x}_2 x_3) \\ & + \frac{1}{2}(x_2 x_4 + \bar{x}_2 \bar{x}_4) + \frac{1}{2}(x_2 x_5 + \bar{x}_2 \bar{x}_5) + \frac{1}{2}(x_2 \bar{x}_6 + \bar{x}_2 x_6) \\ & + (x_3 \bar{x}_4 + \bar{x}_3 x_4) + (x_3 x_5 + \bar{x}_3 \bar{x}_5) + \frac{1}{2}(x_3 \bar{x}_6 + \bar{x}_3 x_6) \\ & + (x_4 \bar{x}_5 + \bar{x}_4 x_5) + \frac{1}{2}(x_4 x_6 + \bar{x}_4 \bar{x}_6) + (x_5 \bar{x}_6 + \bar{x}_5 x_6) \end{aligned}$$

satisfying the equation  $g(x_1, \dots, x_6) = \beta_g(1, x_1, \dots, x_6) - \frac{19}{2}$ .

**Proposition 8.2** ([51]). If  $\beta$  is a bi-form, then  $\beta(x_0, x_1, \dots, x_n) = \beta(\bar{x}_0, \bar{x}_1, \dots, \bar{x}_n)$  for every binary vector  $(x_0, x_1, \dots, x_n) \in \mathbb{B}^{n+1}$ .

*Proof.* Follows directly from the definitions, since the value of  $\beta$  depends only on equalities and non-equalities of the variables, that is on relations which do not change when simultaneously all the variables are complemented. □

**Proposition 8.3** ([51]). *If  $\beta_f$  is the unique bi-form of the quadratic pseudo-Boolean function  $f$ , then*

$$\min_{(x_1, \dots, x_n) \in \mathbb{B}^n} f(x_1, \dots, x_n) = c_f + \min_{(x_0, x_1, \dots, x_n) \in \mathbb{B}^{n+1}} \beta_f(x_0, x_1, \dots, x_n)$$

and

$$\max_{(x_1, \dots, x_n) \in \mathbb{B}^n} f(x_1, \dots, x_n) = c_f + \max_{(x_0, x_1, \dots, x_n) \in \mathbb{B}^{n+1}} \beta_f(x_0, x_1, \dots, x_n).$$

*Proof.* Follows readily by Proposition 8.2. □

The above remarks imply that instead of  $x_0$ , any of the  $n+1$  variables of the bi-form  $\beta_f$  could be fixed at 1, without changing the set of values associated to  $\beta_f$ .

**Corollary 8.1.** *If  $\beta_f$  is the unique bi-form of the quadratic pseudo-Boolean function  $f$ , given as in (8.3), and  $h(x_0, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = c_f + \beta_f(x_0, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$ , i.e. if we obtain  $g$  from  $\phi$  by fixing  $x_i = 1$ , then both  $f$  and  $h$  are quadratic pseudo-Boolean functions and have the same minimum and maximum values.*

**Example 8.2.** *Returning to the quadratic pseudo-Boolean function  $g = -f_6$  used in Example 8.1 and its unique bi-form  $\beta_g$ , then function*

$$\begin{aligned} h(x_0, x_2, x_3, x_4, x_5, x_6) &= -\frac{19}{2} + \beta_g(x_0, 1, x_2, x_3, x_4, x_5, x_6) \\ &= -2 + x_2 + x_4 + 2x_5 + 2x_6 \\ &\quad -x_0x_2 + 2x_0x_4 - 5x_0x_5 \\ &\quad -x_2x_3 + x_2x_4 + x_2x_5 - x_2x_6 \\ &\quad -2x_3x_4 + 2x_3x_5 - x_3x_6 \\ &\quad -2x_4x_5 + x_4x_6 - 2x_5x_6 \end{aligned}$$

has the same minimum (i.e.  $\nu(f) = -5$ ) and maximum values (i.e.  $\tau(g) = 4$ ) as  $g$ .

Noting that a quadratic pseudo-Boolean function, given as a multilinear polynomial (1.5), can be transformed to a bi-form in  $O(n^2)$  time, then Proposition 8.1 shows that the minimization of a quadratic Pseudo-Boolean function is equivalent to the

minimization of the corresponding unique bi-form.

It should be noted that certain combinatorial algorithms are naturally formulated as bi-forms, in particular MAX-CUT and graph balancing problems.

**Definition 8.3.** Given a bi-form,  $f = \sum_{e \in E} \alpha_e e$ , we associate to it a graph  $G_f$ , whose vertices correspond to the indices  $\{0, 1, \dots, n\}$  of the variables, and whose edges correspond to those pairs  $(i, j)$  for which there is a bi-term in  $f$  involving the variables  $x_i$  and  $x_j$ . The edge  $e = (i, j)$  will sometimes refer to the edge  $(i, j)$  of  $G_f$ , and some other times to the Boolean expression  $e(X) = (x_i \bar{x}_j + \bar{x}_i x_j)$  or  $= (x_i x_j + \bar{x}_i \bar{x}_j)$  associated to it in  $f$ . An edge will be called positive (negative) if the associated bi-term is positive (negative); the weight of an edge  $e$  is the corresponding positive coefficient  $\alpha_e$  in  $f$ . In other words,  $G_f$  is a weighted signed graph, associated to the bi-form of  $f$ .

**Example 8.3.** The weighted signed graph  $G_g$  of the bi-form  $\beta_g$  associated to the quadratic pseudo-Boolean function  $g = -f_6$  (see Example 8.1) is displayed in Figure 8.1.

**Definition 8.4.** If  $\mathbf{x}$  is a 0-1 vector of  $n+1$  components, then an edge  $e \in E$  is called conflicting with  $\mathbf{x}$  if  $e(\mathbf{x}) \neq 0$ , otherwise we say it agrees with  $\mathbf{x}$ .

**Remark 8.1.** For any 0-1 vector  $\mathbf{x} \in \mathbb{B}^{n+1}$ ,

$$f(\mathbf{x}) = \sum_{e \text{ is conflicting with } \mathbf{x}} \alpha_e.$$

**Definition 8.5.** Paths in  $G_f$  with a possible repetition of edges (such paths are called sometimes walks) are considered next. The number of times an edge  $e$  is used by a path  $P$  will be called the multiplicity of  $e$  with respect to  $P$ , and will be denoted by  $m_P(e)$ . A path is called closed, if its first and last vertices coincide.

**Definition 8.6.** A path is called negative if the sum of the multiplicities of the negative edges in it is odd. A closed negative path without repetition of edges is called a negative cycle, while a closed negative path, with possibly repeated edges, is called a noose. A noose is called rooted if it passes through the root of  $f$ . To a rooted noose  $N$  (which we

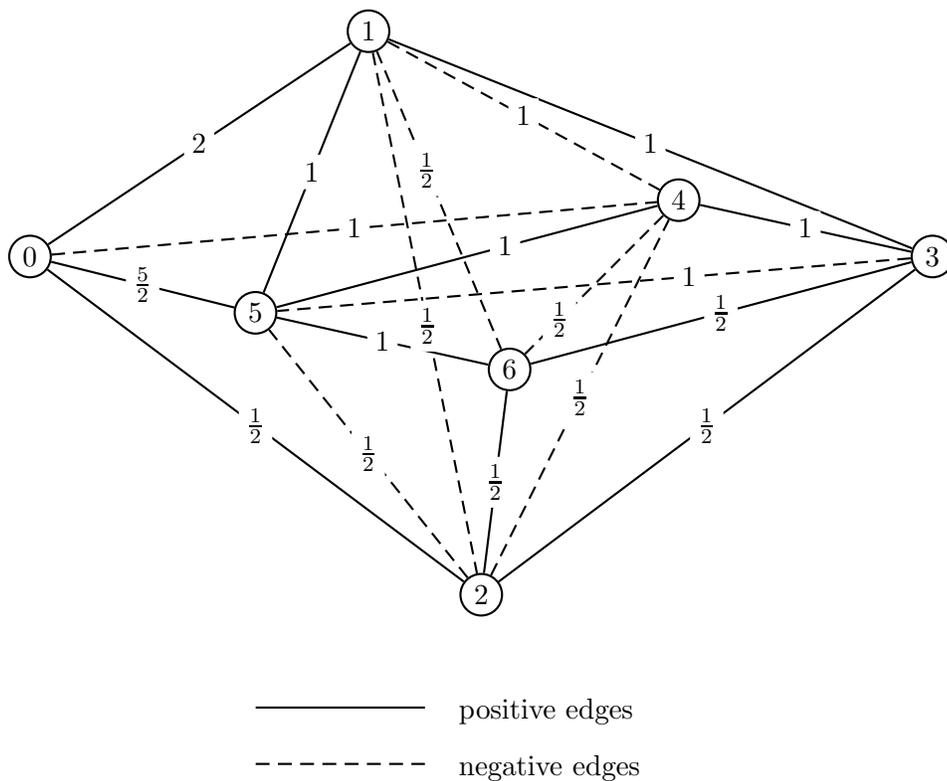


Figure 8.1: The graph  $G_g$  of the bi-form given in Example 8.1.

will consider as a subset  $N$  of the edges together with a multiplicity function  $m_N$ ) we shall also associate the QUBO  $N = \sum_{e \in N} m_N(e)e$ .

The following easy remarks (see e.g. [51, 57, 68]) will be useful later in this section.

**Remark 8.2.** *The equation  $f(X) = 0$  is consistent if and only if there is no negative cycle in  $G_f$ . Moreover, the equation  $f(X) = 0$  has a unique solution (assuming  $x_0 = 1$ ) if and only if  $G_f$  is connected and does not contain negative cycles.*

**Remark 8.3.** *If  $e$  and  $e'$  are bi-terms involving the pairs of variables  $x, y$  and  $y, z$ , respectively, then*

$$e + e' = e'' + c$$

*for some cubic posiform  $c$  and a bi-term  $e''$  involving  $x$  and  $z$ . Moreover the sign of  $e''$  is the product of the signs of  $e$  and  $e'$ .*

*Proof.*

$$\begin{aligned} (xy + \bar{x}\bar{y}) + (yz + \bar{y}\bar{z}) &= (x\bar{z} + \bar{x}z) + 2(xyz + \bar{x}\bar{y}\bar{z}), \\ (xy + \bar{x}\bar{y}) + (y\bar{z} + \bar{y}z) &= (xz + \bar{x}\bar{z}) + 2(xy\bar{z} + \bar{x}\bar{y}z), \\ (x\bar{y} + \bar{x}y) + (y\bar{z} + \bar{y}z) &= (x\bar{z} + \bar{x}z) + 2(x\bar{y}z + \bar{x}y\bar{z}). \end{aligned}$$

□

**Example 8.4.** Consider the rooted noose  $N = \{(0, 1), (1, 4), (4, 3), (3, 5), (5, 4), (4, 0)\}$  in the graph of Figure 8.1. Applying Remark 8.3, we have

$$\begin{aligned} N &= (x_0\bar{x}_1 + \bar{x}_0x_1) + (x_1x_4 + \bar{x}_1\bar{x}_4) + (x_3\bar{x}_4 + \bar{x}_3x_4) + \\ &\quad (x_3x_5 + \bar{x}_3\bar{x}_5) + (x_4\bar{x}_5 + \bar{x}_4x_5) + (x_0x_4 + \bar{x}_0\bar{x}_4) \\ &= 1 + 2[(x_0\bar{x}_1\bar{x}_4 + \bar{x}_0x_1x_4) + (x_0\bar{x}_3x_4 + \bar{x}_0x_3\bar{x}_4) + (x_0x_3x_5 + \bar{x}_0\bar{x}_3\bar{x}_5) + (x_0x_4\bar{x}_5 + \bar{x}_0\bar{x}_4x_5)]. \end{aligned}$$

Thus

$$N(1, x_1, x_2, x_3, x_4, x_5) = 1 + 2(\bar{x}_1\bar{x}_4 + \bar{x}_3x_4 + x_3x_5 + x_4\bar{x}_5).$$

More generally,

**Remark 8.4.** If  $N$  is a rooted noose in  $G_f$ , then

$$N(1, x_1, \dots, x_n) = 1 + q(x_1, \bar{x}_1, \dots, x_n, \bar{x}_n),$$

where  $q$  is a quadratic posiform.

Let  $f$  be a given bi-form,  $x_0$  its root, and let  $\mathcal{C}$ ,  $\mathcal{N}$  and  $\mathcal{N}_0$  denote the set of negative cycles, the set of nooses and the set of rooted nooses in  $G_f$ , respectively. If  $(P)$  is an optimization problem, its optimum value is denoted next by  $\omega(P)$ .

Given a bi-form  $f$ , Boros et al. [51, 57] associated to it the following mathematical programming problems:

- A “cycle covering” problem

$$\begin{aligned}
 \min \quad & v(\mathbf{y}) = \sum_{e \in E} \alpha_e y_e \\
 \text{s.t.} \quad & \sum_{e \in C} y_e \geq 1 \quad \forall C \in \mathcal{C}, \\
 & y_e \in \mathbb{B} \quad \forall e \in E,
 \end{aligned} \tag{CC}$$

- A “noose covering” problem

$$\begin{aligned}
 \min \quad & v(\mathbf{y}) = \sum_{e \in E} \alpha_e y_e \\
 \text{s.t.} \quad & \sum_{e \in N} m_N(e) y_e \geq 1 \quad \forall N \in \mathcal{N}, \\
 & y_e \in \mathbb{B} \quad \forall e \in E,
 \end{aligned} \tag{NC}$$

- The continuous relaxation ( $\mathbf{NC}^c$ ) of the noose covering problem, obtained from ( $\mathbf{NC}$ ) by replacing the conditions  $y_e \in \mathbb{B}$  by  $y_e \geq 0$  for all  $e \in E$ .
- A “noose packing” problem

$$\begin{aligned}
 \max \quad & w(\xi) = \sum_{N \in \mathcal{N}} \xi_N \\
 \text{s.t.} \quad & \sum_{N \ni e} m_N(e) \xi_N \leq \alpha_e \quad \forall e \in E, \\
 & \xi_N \geq 0 \quad \forall N \in \mathcal{N}.
 \end{aligned} \tag{NP}$$

- The “rooted noose packing” problem ( $\mathbf{RNP}$ ) which is obtained from ( $\mathbf{NP}$ ) by replacing  $\mathcal{N}$  by  $\mathcal{N}_0$ .

Boros et al. [51, 57] shown that problems ( $\mathbf{CC}$ ) and ( $\mathbf{NC}$ ) are integer programming problems which are equivalent with the minimization of  $f$ , while ( $\mathbf{NC}^c$ ), ( $\mathbf{NP}$ ) and ( $\mathbf{RNP}$ ) are weaker linear programming relaxations of the above integer programming problems, and the weakest one ( $\mathbf{RNP}$ ) turns out to be equivalent with roof-duality (see Proposition (8.4)).

**Example 8.5.** For bi-form  $\beta_g$  given in Example 8.1 the following is an optimal rooted

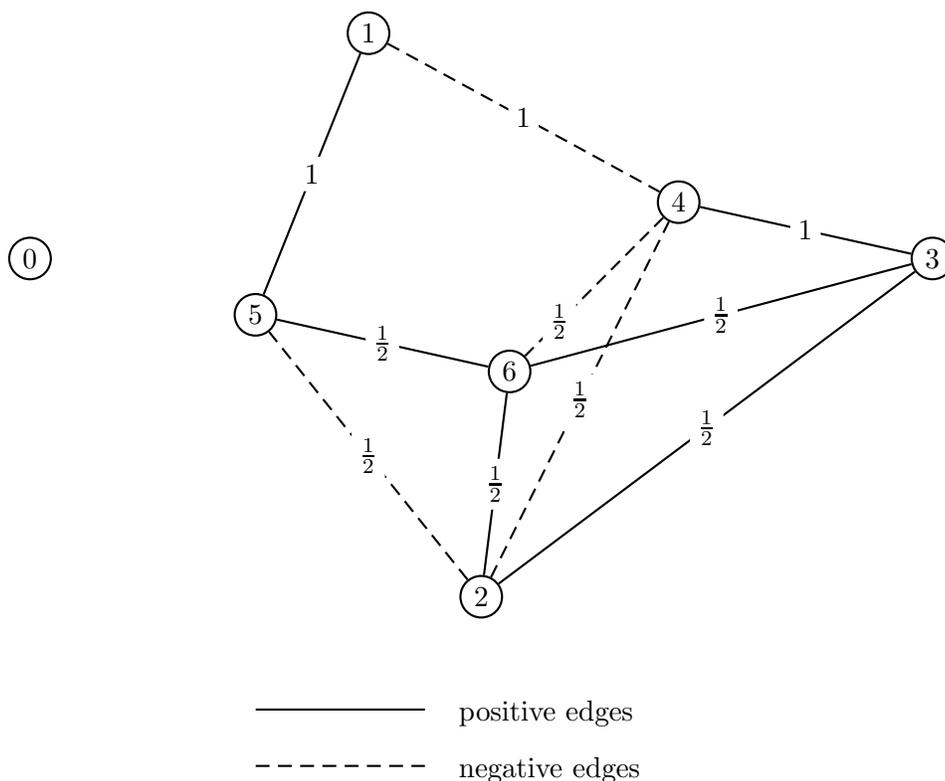


Figure 8.2: The residual graph  $G'_g$  of the bi-form given in Example 8.1.

noose packing for root  $x_0$  (i.e. to problem **(RNP)**):

$$\xi_N = \begin{cases} \frac{1}{2}, & \text{if } N = [(0, 1), (1, 2), (2, 0)], \\ 1, & \text{if } N = [(0, 4), (4, 5), (5, 0)], \\ 1, & \text{if } N = [(0, 1), (1, 3), (3, 5), (5, 0)], \\ \frac{1}{2}, & \text{if } N = [(0, 1), (1, 6), (6, 5), (5, 0)], \\ 0, & \text{otherwise,} \end{cases} \quad (8.5)$$

The residual graph  $G'_g$  is obtained after removing from graph  $G_g$  (see Figure 8.1) the strictly positive nooses from  $\xi_N$ .  $G'_g$  is displayed in Figure 8.2.

The sum of the rooted nooses values is 3, and consequently the roof-dual value of  $g$  is  $-\frac{19}{2} + 3 = -\frac{13}{2}$ .

The “roof duality” approach of [123] for the minimization of bi-forms is rephrased

next. For this, the bi-form of  $f$  is written as

$$f = \sum_{(i,j) \in E^+} \alpha_{ij}(x_i \bar{x}_j + \bar{x}_i x_j) + \sum_{(i,j) \in E^-} \alpha_{ij}(x_i x_j + \bar{x}_i \bar{x}_j),$$

where  $E^+$  and  $E^-$  denote the set of positive and negative edges in  $G_f$ , respectively.

For each quadratic term of  $f$ , its  $L_1$  optimal linear lower bounds are given by

$$x_i x_j \geq \lambda_{ij}(x_i + x_j - 1), \quad \text{for any } 0 \leq \lambda_{ij} \leq 1,$$

$$x_i \bar{x}_j \geq \lambda_{i\bar{j}}(x_i - x_j), \quad \text{for any } 0 \leq \lambda_{i\bar{j}} \leq 1,$$

$$\bar{x}_i x_j \geq \lambda_{\bar{i}j}(x_j - x_i), \quad \text{for any } 0 \leq \lambda_{\bar{i}j} \leq 1,$$

$$\bar{x}_i \bar{x}_j \geq \lambda_{\bar{i}\bar{j}}(1 - x_i - x_j), \quad \text{for any } 0 \leq \lambda_{\bar{i}\bar{j}} \leq 1,$$

for  $0 \leq i < j \leq n$ . For a fixed parameter vector  $\lambda$  let

$$L_\lambda(\mathbf{x}) = \sum_{(i,j) \in E^-} \alpha_{ij}(\lambda_{\bar{i}\bar{j}} - \lambda_{ij}) + \sum_{i=0}^n x_i \left[ \sum_{(i,j) \in E^-} \alpha_{ij}(\lambda_{ij} - \lambda_{\bar{i}\bar{j}}) + \sum_{(i,j) \in E^+} \alpha_{ij}(\lambda_{\bar{i}\bar{j}} - \lambda_{ij}) \right].$$

It can be seen (as in [123]) that the roof dual  $\rho(f)$  of  $f$  is given by

$$\rho(f) = \max_{\lambda} \min_{\mathbf{x}} L_\lambda(\mathbf{x}).$$

The following result relates the optimum of all the above (integer) linear programs. It is to be noted that recently the cycle packing problem has been rediscovered by Ibaraki et al. [148, 147], who have proposed various efficient heuristics to compute these improved bounds and when embedded within a branch-and-bound framework resulted in being one of the most efficient solvers for many classes of QUBO.

**Proposition 8.4** ([51, 57]).

$$\begin{aligned}
\min_{\mathbf{x} \in \mathbb{B}^{n+1}} f(\mathbf{x}) &= \omega(\mathbf{CC}) \\
&= \omega(\mathbf{NC}) \\
&\geq \omega(\mathbf{NC}^c) \\
&= \omega(\mathbf{NP}) \\
&\geq \omega(\mathbf{RNP}) = \rho(f).
\end{aligned}$$

By the symmetries observed in Corollary 8.1, it should be remarked that any of the variables could be used as roots, and in this way consider analogous rooted noose packing problems. Denoting by  $\mathcal{N}_i$  the set of nooses rooted at vertex  $x_i$ , for  $i = 0, 1, \dots, n$ , we consider the problems ([51, 52, 57])

$$\begin{aligned}
\max \quad & w(\xi) = \sum_{N \in \mathcal{N}_i} \xi_N \\
\text{s.t.} \quad & \sum_{N \ni e} m_N(e) \xi_N \leq \alpha_e \quad \forall e \in E, & (\mathbf{RNP}(\mathbf{i})) \\
& \xi_N \geq 0 \quad \forall N \in \mathcal{N}_i.
\end{aligned}$$

Clearly, problem  $(\mathbf{RNP}(\mathbf{0}))$  is the same as  $(\mathbf{RNP})$ . Furthermore, the optimum value of each of the problems  $(\mathbf{RNP}(\mathbf{i}))$  ( $i = 0, 1, \dots, n$ ) is a lower bound on the minimum of  $f$ . Thus, since  $\omega(\mathbf{RNP}) = \rho(f)$  then the following corollary applies readily.

**Corollary 8.2** ([51, 52, 57]).

$$\min_{\mathbf{x} \in \mathbb{B}^{n+1}} f(\mathbf{x}) \geq \max_{i=0, \dots, n} \omega(\mathbf{RNP}(\mathbf{i})) \geq \omega(\mathbf{RNP}) = \rho(f).$$

In concluding this section, let us make a few additional remarks. Let us note first that in the noose packing problem  $(\mathbf{NP})$  we could replace  $\mathcal{N}$  by  $\mathcal{C}$  without changing the optimum value. Furthermore, the resulting negative cycle packing problem can easily be shown to be equivalent with negative triangle packing, which we state here for completeness. For this, let us introduce a positive edge  $e$  for every pair of variables which are not connected by an edge in  $G_f$ , assume that  $\alpha_e = 0$  for these newly introduced edges, and denote by  $\tilde{E}$  this extended set of edges. Let us also denote by  $\bar{e}$  the sign

complement of edge  $e$  (i.e., if  $e$  is a negative edge between variables  $x_i$  and  $x_j$ , then  $\bar{e}$  denotes a positive edge between  $x_i$  and  $x_j$ , etc.), and note that if  $e \in \tilde{E}$ , then we have  $\bar{e} \notin \tilde{E}$ . Let us denote finally by  $\mathcal{T}$  the collection of all negative triangles (i.e., negative cycles consisting of three edges), and consider the problem

$$\begin{aligned} \max \quad & w(\xi) = \sum_{T \in \mathcal{T}} \xi_T - \sum_{e \in \tilde{E}} \left( \sum_{T \ni \bar{e}} \xi_T \right) \\ \text{s.t.} \quad & 0 \leq \sum_{T \ni e} \xi_T - \sum_{T \ni \bar{e}} \xi_T \leq \alpha_e \quad \forall e \in \tilde{E}, \\ & \xi_T \geq 0 \quad \forall T \in \mathcal{T}. \end{aligned} \tag{TP}$$

We can thus conclude that  $\omega(\mathbf{NP}) = \omega(\mathbf{TP})$ . Furthermore, comparing problem **(TP)** with the formulation of the *cubic dual* bound  $C_3$ , and in particular with the triangle inequalities based formulations of it (i.e. (CUBIC DUAL)), the following claim can be shown:

**Proposition 8.5.**  $\omega(\mathbf{NP}) = \omega(\mathbf{TP}) = C_3$ .

## 8.2 Optimal rooted noose packings relationship to the network model

Boros et al. [51, 57] has shown that the optimum value and an optimal rooted noose packing can be computed by solving a maximum-flow problem in an *undirected* network on  $2n+2$  vertices. Together with the result of the previous section this implies that the roof dual  $\rho(f)$  of a quadratic pseudo-Boolean function  $f$  in  $n$  variables, as well as any of the possibly improved lower bounds  $\omega(\mathbf{RNP}(\mathbf{i}))$  can be computed in  $O(n^3)$  time. We present this approach for the case of  $x_0$  as root, though it can be applied directly for any other choice of a root.

**Definition 8.7.** *If  $f$  is a bi-form rooted at  $x_0$ , then let  $U_f = (W, A)$  be the bi-form network, whose  $2n+2$  nodes correspond to the literals of the set  $W = \{x_0, \bar{x}_0, \dots, x_n, \bar{x}_n\}$ , and whose edges are associated to the edges of  $G_f$  in the following way. If  $e \in E$  is a positive edge between  $i$  and  $j$ , i.e.  $e = x_i \bar{x}_j + \bar{x}_i x_j$ , then there are two corresponding edges in  $A$ : an edge  $e'$  between  $x_i$  and  $x_j$  and another edge  $e''$  between  $\bar{x}_i$  and  $\bar{x}_j$ . If  $e \in E$  is a negative edge between  $i$  and  $j$ , i.e.  $e = x_i x_j + \bar{x}_i \bar{x}_j$ , then there are two*

corresponding edges in  $A$ : an edge  $e'$  between  $x_i$  and  $\bar{x}_j$  and another edge  $e''$  between  $\bar{x}_i$  and  $x_j$ . Let in both cases  $c(e') = c(e'') = \frac{1}{2}\alpha_e$  be the capacities of these edges in  $U_f$ .

**Definition 8.8** ([51, 57]). *If  $P$  is a path from  $x_0$  to  $\bar{x}_0$  in a bi-form network  $U_f$ , going through the vertices  $\{u_1, \dots, u_p\}$  (i.e.  $u_1 = x_0$ ,  $u_p = \bar{x}_0$ ), then the sequence  $\{\bar{u}_p, \dots, \bar{u}_1\}$  describes another path  $\bar{P}$  between  $x_0$  and  $\bar{x}_0$ . The pair  $(P, \bar{P})$  will be called a bi-path.*

The following Lemmas can be seen easily.

**Lemma 8.1** ([51, 57]). *There is a one-to-one correspondence between the rooted nooses in  $G_f$  and the bi-paths in the bi-form network  $U_f$ .*

*Proof.* A rooted noose provides a closed walk from  $x_0$  to  $x_0$  in  $G_f$ , in which we pass an odd number of times negative edges (some of them possibly twice). Thus, by the above definitions, the corresponding edges in  $U_f$  form a path  $P$  from  $x_0$  to  $\bar{x}_0$  and its twin  $\bar{P}$ , i.e., a bi-path. Conversely, a bi-path  $(P, \bar{P})$  in  $U_f$  corresponds to a closed walk  $W$  from  $x_0$  to  $x_0$  in  $G_f$ . Since along the path  $P$  (and  $\bar{P}$ ) we must move an odd number of times from an un-complemented variable to a complemented one, in  $W$  we must pass through an odd number of negative edges, i.e.,  $W$  is a rooted noose in  $G_f$ .  $\square$

It is well-known in the theory of network flows that a flow  $F$  from  $x_0$  to  $\bar{x}_0$  (in  $U_f$ ) can always be decomposed into the sum of a finite number of elementary flows  $F_1, \dots, F_t$ , going through the paths  $P_1, \dots, P_t$  from  $x_0$  to  $\bar{x}_0$ . Thus, due to the symmetric nature of  $U_f$ , the following claim follows readily from the definitions.

**Lemma 8.2** ([51, 57]). *Let  $F_i$   $i = 1, \dots, t$  be elementary flows from  $x_0$  to  $\bar{x}_0$  through the paths  $P_i$ , and having values  $f_i$ , respectively. Further, let  $\bar{F}_i$  be the elementary flow through the path  $\bar{P}_i$  having the value  $f_i$  for  $i = 1, \dots, t$ . If  $F = \sum F_i$  is a feasible flow in  $U_f$ , then  $\bar{F} = \sum \bar{F}_i$  is also a feasible flow in  $U_f$  (having the same value as  $F$ ).*

A flow  $F$  from  $x_0$  to  $\bar{x}_0$  in  $U_f$  with the property  $F = \bar{F}$  is called a bi-flow.

**Lemma 8.3** ([51, 57]). *To every feasible rooted noose packing  $\xi = \sum_{N \in \mathcal{N}_0} \xi_N N$  there is a corresponding bi-flow of  $U_f$  with  $\sum_{N \in \mathcal{N}_0} \xi_N$  as its flow value. Conversely, every feasible bi-flow in  $U_f$  corresponds in this way to a feasible solution of **(RNP)** (however, this correspondence may not be one-to-one, in general).*

*Proof.* Since a convex combination of feasible flows is again a feasible flow, Lemma 8.2 implies that from any feasible flow  $F$  of  $U_f$  we can obtain a feasible bi-flow with the same flow value, by considering simply  $\frac{1}{2}F + \frac{1}{2}\overline{F}$ . Therefore, Lemmas 8.1 and 8.2 imply readily the claim.  $\square$

**Example 8.6.** *To illustrate that rooted noose packings of  $G_f$  and bi-flows of  $U_f$  are not necessarily in a one-to-one correspondence, let us consider the bi-form  $f$  defined by*

$$\begin{aligned} f(x_0, x_1, x_2, x_3, x_4) &= 2(x_0x_1 + \overline{x}_0\overline{x}_1) + 2(x_0x_2 + \overline{x}_0\overline{x}_2) + 2(x_1x_2 + \overline{x}_1\overline{x}_2) \\ &\quad + 2(x_1x_3 + \overline{x}_1\overline{x}_3) + 2(x_3x_4 + \overline{x}_3\overline{x}_4) + 2(x_4x_1 + \overline{x}_4\overline{x}_1) \end{aligned}$$

and its graph  $G_f$ . The nooses

$$\begin{aligned} N_1 &= 2(x_0x_1 + \overline{x}_0\overline{x}_1) + (x_1x_3 + \overline{x}_1\overline{x}_3) + (x_3x_4 + \overline{x}_3\overline{x}_4) + (x_4x_1 + \overline{x}_4\overline{x}_1) \\ N_2 &= 2(x_0x_2 + \overline{x}_0\overline{x}_2) + 2(x_1x_2 + \overline{x}_1\overline{x}_2) + (x_1x_3 + \overline{x}_1\overline{x}_3) \\ &\quad + (x_3x_4 + \overline{x}_3\overline{x}_4) + (x_4x_1 + \overline{x}_4\overline{x}_1) \end{aligned}$$

with weights  $\xi_{N_1} = \xi_{N_2} = 1$  form a feasible rooted noose packing in  $G_f$ . In the corresponding bi-flow of  $U_f$  however the flows cancel out on some of the arcs (corresponding to a circulation), and the non-zero edges of the resulting bi-flow correspond to the rooted noose packing consisting of a single noose  $N_3$  with weight  $\xi_{N_3} = 2$ , where

$$N_3 = (x_0x_1 + \overline{x}_0\overline{x}_1) + (x_0x_2 + \overline{x}_0\overline{x}_2) + (x_1x_2 + \overline{x}_1\overline{x}_2).$$

Finally, Lemma 8.3 implies immediately the main statement of this section:

**Proposition 8.6** ([51, 57]).  $\omega(\mathbf{RNP})$  is equal to the value of the maximum flow from  $x_0$  to  $\overline{x}_0$  in the bi-form network  $U_f$ .

**Corollary 8.3.** Problem  $(\mathbf{RNP})$  can be solved in  $O(n^3)$  time.

Let us add that whenever the given bi-form  $f$  has integral coefficients, the corresponding network  $U_f$  has a half-integral maximum bi-flow, as the simple argument in the proof of Lemma 8.3 shows. Consequently, for integral bi-forms we have half-integral optimal noose packings.

The undirected network  $U_f$  could also be viewed, for algorithmic purpose, as the directed network  $N_f$  obtained from  $U_f$  by replacing every undirected edge  $e = (u, v)$  by two directed arcs  $e' = (u, v)$  and  $e'' = (v, u)$  between the same pair of literals, and assigning to both of them capacity  $C(e') = C(e'') = C(e)$ .  $N_f$  is precisely the capacitated directed network, in this case is associated to the bi-form  $f$ , as has been described in Section 5.3.

The directed network model can represent an arbitrary quadratic posiform of a quadratic PBF  $f$  (including the bi-form), and the corresponding maximum-flow makes also possible to derive the roof-dual of  $f$ . A natural question is therefore what network model version to consider for each situation.

On the one hand the symmetric bi-form based model ( $U_f$ ) is preferable, because that leads to the cycle and noose packing problems, and allows us to apply an iterated version of the rooted noose packing (see Sections 8.4 and 8.5). It also makes possible to use maximum flow algorithms especially designed for undirected networks. For instance, [154] shows that in an undirected network of  $n$  nodes and  $m$  edges a maximum flow of value  $v$  can be computed in  $O(nm^{2/3}v^{1/6})$  time. Thus, for bi-forms with "small" integer coefficients the roof dual value could be obtained more efficiently by using the algorithm of [154] in the above undirected network model than by standard network flow algorithms in the directed network model of [54, 226]

On the other hand the directed network model ( $N_f$ ) can be applied to an arbitrary quadratic posiform, makes possible to directly get weak and strong persistencies and may have certain algorithmic advantages depending on the input.

The next statement establishes for the first time the result that the conclusions (e.g. bounds and persistencies) derived by the three models:  $G_f$ ,  $U_f$  and  $N_f$ , can be explicitly related. In particular, we are interested in quickly determining how to find the residual graph of the noose packing problem (denoted as  $G'_f$ ) given that the residual network of the directed network model  $N_f$  is known. This result makes possible to use the directed network  $N_f$  (our model of choice in this dissertation) efficiently with the iterated rooted noose packing algorithms to be presented later (see Sections 8.4 and 8.5).

**Theorem 8.1.** *Let  $f$  be a quadratic pseudo-Boolean function. If*

- $G_f$  is the balancing graph of the bi-form of  $f$ ,
- $U_f$  is the undirected “network” associated to the bi-form of  $f$  and
- $N_f$  is the directed network associated to the bi-form of  $f$  (i.e. by duplicating an edge  $\{i, j\}$  of  $U_f$  into arcs  $(u, v)$  and  $(v, u)$  with the same capacity of edge  $\{u, v\}$ ),

then the residual graph  $G'_f$  that corresponds to the optimal noose cycle packing of  $G_f$  can be obtained directly from the residual network  $N_f[\lambda_0]$  corresponding to the maximum flow  $\lambda_0$ , by considering the residual capacities  $C(u, v)$  and  $C(v, u)$  of arcs  $(u, v)$  and  $(v, u)$  for all edges  $\{u, v\}$  of  $G_f$ , so that the capacity of edge  $\{u, v\}$  in  $G'_f$  is  $\alpha_{\{u, v\}} = \min(C(u, v), C(v, u))$  (if this value is 0 then it means that the edge does not exist in  $G'_f$ ).

*Proof.*  $\mathbf{x}$  is a feasible flow in  $N_f$  if and only if  $\mathbf{x}'$  is a feasible flow in the residual network  $N_f[\lambda_0]$  such that:

$$\begin{cases} \mathbf{x}' \geq 0 \\ \mathbf{x}'(u, v) - \mathbf{x}'(v, u) = \mathbf{x}(u, v) - \mathbf{x}_0(u, v) \rightarrow \mathbf{x}(u, v) = \mathbf{x}_0(u, v) + [\mathbf{x}'(u, v) - \mathbf{x}'(v, u)] \\ \mathbf{x}'(u, v) \mathbf{x}'(v, u) = 0 \end{cases} \quad (8.6)$$

$\mathbf{y}$  is a feasible flow in  $U_f$  if and only if for every edge  $\{u, v\}$

$$\begin{cases} \mathbf{y}(u, v) \geq 0 \\ \mathbf{y}(u, v) = \max(0, \mathbf{x}(u, v) - \mathbf{x}(v, u)) \\ \mathbf{y}(v, u) = \max(0, \mathbf{x}(v, u) - \mathbf{x}(u, v)) \\ \mathbf{y}(u, v) \mathbf{y}(v, u) = 0 \end{cases} \quad (8.7)$$

Using (8.6) and substituting in (8.7) then the following system of equations is obtained that relates flows  $\mathbf{y}$ ,  $\mathbf{x}_0$  and  $\mathbf{x}'$ :

$$\begin{cases} \mathbf{y}(u, v) \geq 0 \\ \mathbf{y}(u, v) = \max(0, \mathbf{x}_0(u, v) - \mathbf{x}_0(v, u) + 2[\mathbf{x}'(u, v) - \mathbf{x}'(v, u)]) \\ \mathbf{y}(v, u) = \max(0, \mathbf{x}_0(v, u) - \mathbf{x}_0(u, v) + 2[\mathbf{x}'(v, u) - \mathbf{x}'(u, v)]) \\ \mathbf{y}(u, v)\mathbf{y}(v, u) = 0 \end{cases}$$

From the above systems of equations, clearly there is a one-to-one correspondence between a feasible flow in  $U_f$  and a feasible flow in  $N_f$ . It is also clear that there is a one-to-one correspondence between a feasible flow in  $N_f$  and a feasible flow in the residual network  $N_f[\lambda_0]$ . From Lemma (8.3) to every feasible rooted noose packing there is a corresponding bi-flow in  $U_f$ . Thus, to every flow of the residual network  $N_f[\lambda_0]$  there is a corresponding feasible rooted noose packing. In particular from Proposition (8.6) there is an optimal rooted noose packing associated to a maximum flow of  $N_f[\lambda_0]$ .

Let us consider any arc  $(u, v)$  of  $N_f[\lambda_0]$  with capacity  $C(u, v)$  and an arc  $(v, u)$  with capacity  $C(v, u)$ . Clearly, if  $C(u, v) + C(v, u) > 0$  then there is an edge in  $U_f$  with strictly positive capacity and equal to  $\frac{1}{2}(C(u, v) + C(v, u))$ .

Without loss of generality, let us assume next that  $C(u, v) \geq C(v, u)$ . Then  $\mathbf{x}_0(u, v) = \frac{1}{2}(C(u, v) - C(v, u))$ . If  $\mathbf{x}' = 0$  and  $\mathbf{x}_0$  is a maximum flow of  $N_f$  (i.e.  $N_f[\lambda_0]$  is the corresponding residual network) then there is a maximum flow in  $U_f$  having a flow of  $\frac{1}{2}(C(u, v) - C(v, u))$  from  $u$  to  $v$  (i.e.  $\mathbf{y}(u, v) = \frac{1}{2}(C(u, v) - C(v, u))$ ). Since there exists a linear combination of rooted nooses packings in  $G_f$  associated to this flow, then the residual graph of  $G_f$  after removing these rooted nooses packings has an edge  $\{u, v\}$  with capacity  $\frac{1}{2}(C(u, v) + C(v, u)) - \frac{1}{2}(C(u, v) - C(v, u)) = C(v, u)$ , which is the smallest capacity of the two arcs in  $N_f[\lambda_0]$ .  $\square$

**Example 8.7.** *The quadratic pseudo-Boolean function  $g$  previously used in Example*

8.2 can be represented by the following standard posiform:

$$\begin{aligned}
\phi_g = & -\frac{13}{2} + \bar{x}_1\bar{x}_2 + 2\bar{x}_1x_3 + \bar{x}_1\bar{x}_6 + 2\bar{x}_3\bar{x}_5 + 2x_4\bar{x}_5 \\
& + (x_1x_4 + \bar{x}_1\bar{x}_4) + (x_1\bar{x}_5 + \bar{x}_1x_5) \\
& + \frac{1}{2}(x_2\bar{x}_3 + \bar{x}_2x_3) + \frac{1}{2}(x_2x_4 + \bar{x}_2\bar{x}_4) \\
& + \frac{1}{2}(x_2x_5 + \bar{x}_2\bar{x}_5) + \frac{1}{2}(x_2\bar{x}_6 + \bar{x}_2x_6) \\
& + (x_3\bar{x}_4 + \bar{x}_3x_4) + \frac{1}{2}(x_3\bar{x}_6 + \bar{x}_3x_6) \\
& + \frac{1}{2}(x_4x_6 + \bar{x}_4\bar{x}_6) + \left(\frac{3}{2}\bar{x}_5x_6 + \frac{1}{2}x_5\bar{x}_6\right)
\end{aligned} \tag{8.8}$$

Using Theorem 8.1 and the fact that  $\phi_g$  is in one-to-one correspondence with the residual network of  $N_g$ , then it is clear that

$$\psi_g = \bar{x}_1\bar{x}_2 + 2\bar{x}_1x_3 + \bar{x}_1\bar{x}_6 + 2\bar{x}_3\bar{x}_5 + 2x_4\bar{x}_5 + \bar{x}_5x_6$$

is a posiform that corresponds to a feasible rooted nooses packing. Since  $\phi_g$  corresponds to the residual network obtained after applying the maximum flow to the bi-form of  $g$ , then  $\psi_g$  it is also associated to the optimal rooted nooses packing (8.5).

### 8.3 Rooted noose packing structure and decomposition

From Theorem 8.1 (see also Example 8.7) it is clear that a bi-form  $f$  can be decomposed into two components, one that corresponds to the rooted nooses  $\xi_N$  and the other that corresponds to the residual balancing graph of  $G_f$ . This fact has already been introduced by Boros et al. [51] and was called the structure theorem. The following proposition presents this result on a different angle.

**Proposition 8.7.** *Let  $f$  be a quadratic pseudo-Boolean function whose unique bi-form in the variables  $x_0, x_1, \dots, x_n$  is  $\beta_f$ . Then bi-form  $\beta_f$  can be partitioned in 2 quadratic pseudo-Boolean functions  $\varphi_f^{(k)}$  and  $\phi_{N_f^{(k)}}$  satisfying the following conditions:*

$$(i) \quad \beta_f = \sum_{N \in \mathcal{R}_k} \xi_N + \varphi_f^{(k)} + \phi_{N_f^{(k)}}$$

$$(ii) \quad \varphi_f^{(k)} = x_k \psi_f^{(k)}(x_0, \dots, x_{k-1}, x_{k+1}, \dots, x_n) + \bar{x}_k \psi_f^{(k)}(\bar{x}_0, \dots, \bar{x}_{k-1}, \bar{x}_{k+1}, \dots, \bar{x}_n)$$

- (iii)  $\phi_{N_f^{(k)}}$  is a bi-form that corresponds to the residual balancing graph  $G_f^{(k)}$  after removing a set of nooses  $\mathcal{R}_k$  rooted at  $x_k$  with values  $\xi$ ;
- (iv)  $\psi_f^{(k)} = \phi_{N_f^{(k)}[\lambda_{\mathcal{R}}]} - \phi_{G_f^{(k)}}$ , i.e.  $\psi_f^{(k)}$  corresponds to the quadratic posiform associated to the nooses in set  $\mathcal{R}$  and can be determined by finding a flow  $\lambda_{\mathcal{R}_k}$  between nodes  $x_k$  and  $\bar{x}_k$  in  $N_f^{(k)}$ , as ascertained by Theorem 8.1.

We shall call the above decomposition as the  $\mathcal{R}_k$ -decomposition. If  $R_k$  is an optimal rooted noose packing then we sometimes represent this decomposition as

$$\beta_f = \xi_k + \varphi_k + \phi_k.$$

It should be noted that  $\varphi_f^{(k)}$  is a quadratic pseudo-Boolean function which has a particular cubic posiform representation. Namely, for every nonzero cubic term  $x_k uv$  of  $\varphi_f^{(k)}$  there is a cubic term on the same variables  $\bar{x}_k \bar{u} \bar{v}$  (see (ii) above). This fact can be shown by noticing that the identity

$$\begin{aligned} x_k uv + \bar{x}_k \bar{u} \bar{v} &= 1 - x_k - u - v + x_k u + x_k v + uv \\ &= -\frac{1}{2} + \frac{1}{2} (x_k u + \bar{x}_k \bar{u}) + \frac{1}{2} (x_k v + \bar{x}_k \bar{v}) + \frac{1}{2} (uv + \bar{u} \bar{v}) \end{aligned}$$

applies. This result will be explored further in the following sections.

**Example 8.8.** Let us consider the quadratic pseudo-Boolean function  $g$  of Example 8.2. Using  $x_3$  as root for the decomposition then the corresponding bi-form can be decomposed as follows:

$$\begin{aligned} \beta_g &= 2 + \varphi_g^{(3)} + \phi_{N_g^{(3)}}, \text{ where} \\ \varphi_g^{(3)} &= x_3 (\bar{x}_1 \bar{x}_2 + \bar{x}_1 \bar{x}_4 + \bar{x}_4 x_5 + x_5 \bar{x}_6) + \bar{x}_3 (x_1 x_2 + x_1 x_4 + x_4 \bar{x}_5 + \bar{x}_5 x_6) \text{ and} \\ \phi_{N_g^{(3)}} &= 2(x_0 \bar{x}_1 + \bar{x}_0 x_1) + \frac{1}{2}(x_0 \bar{x}_2 + \bar{x}_0 x_2) + \bar{x}_0 \bar{x}_4 + x_0 x_4 + \frac{5}{2}(x_0 \bar{x}_5 + \bar{x}_0 x_5) \\ &\quad + \frac{1}{2}(x_1 x_4 + \bar{x}_1 \bar{x}_4) + (x_1 \bar{x}_5 + \bar{x}_1 x_5) + \frac{1}{2}(x_1 x_6 + \bar{x}_1 \bar{x}_6) + \\ &\quad + \frac{1}{2}(x_2 x_4 + \bar{x}_2 \bar{x}_4) + \frac{1}{2}(x_2 x_5 + \bar{x}_2 \bar{x}_5) + \frac{1}{2}(x_2 \bar{x}_6 + \bar{x}_2 x_6) + \\ &\quad + \frac{1}{2}(x_4 \bar{x}_5 + \bar{x}_4 x_5) + \frac{1}{2}(x_4 x_6 + \bar{x}_4 \bar{x}_6) + \\ &\quad + \frac{1}{2}(x_5 \bar{x}_6 + \bar{x}_5 x_6). \end{aligned}$$

In this example the  $\mathcal{R}_3$  nooses correspond to an optimal noose packing rooted at  $x_3$ . Thus,  $\omega(\mathbf{RNP}(\mathbf{3})) = c_0 + \sum_{N \in \mathcal{R}_3} \xi_N = -9.5 + 2 = -7.5$ . Note that in this case  $\omega(\mathbf{RNP}(\mathbf{3})) < \rho(f) = -6.5$ .

Using  $x_5$  as root for the decomposition then the corresponding bi-form can be partitioned as follows:

$$\begin{aligned} \beta_g &= \frac{7}{2} + \varphi_g^{(5)} + \phi_{N_g^{(5)}}, \text{ where} \\ \varphi_g^{(5)} &= x_5(2x_1\bar{x}_0 + x_2\bar{x}_0 + 2\bar{x}_4\bar{x}_0 + \bar{x}_1\bar{x}_2 + 2\bar{x}_1x_3 + \bar{x}_1\bar{x}_6 + x_2\bar{x}_6) + \\ &\quad \bar{x}_5(2\bar{x}_1x_0 + \bar{x}_2x_0 + 2x_4x_0 + x_1x_2 + 2x_1\bar{x}_3 + x_1x_6 + \bar{x}_2x_6) \text{ and} \\ \phi_{N_g^{(5)}} &= (x_1\bar{x}_0 + \bar{x}_1x_0) + (x_1x_4 + \bar{x}_1\bar{x}_4) + \frac{1}{2}(x_2\bar{x}_3 + \bar{x}_2x_3) + \frac{1}{2}(x_2x_4 + \bar{x}_2\bar{x}_4) + \\ &\quad (x_3\bar{x}_4 + \bar{x}_3x_4) + \frac{1}{2}(\bar{x}_3x_6 + x_3\bar{x}_6) + \frac{1}{2}(x_4x_6 + \bar{x}_4\bar{x}_6). \end{aligned}$$

The nooses  $\mathcal{R}_5$  correspond to an optimal noose packing rooted at  $x_5$ . Thus,  $\omega(\mathbf{RNP}(\mathbf{5})) = -6$ . In this case  $\omega(\mathbf{RNP}(\mathbf{5})) > \rho(f) = -6.5 > \omega(\mathbf{RNP}(\mathbf{3}))$ .

#### 8.4 Iterated roof-duality

The decomposition result established by Proposition 8.7 can be rewritten as

$$f_{i_k} = \rho_{i_k} + \varphi_{f_{i_k}}^{(i_{k+1})} + f_{i_{k+1}},$$

where (i)  $\rho_{i_k}$  corresponds to the value of a feasible noose packing  $\mathcal{R}$  of function  $f_{i_k}$  rooted at  $x_{i_k}$  and (ii)  $\varphi_{f_{i_k}}^{(i_{k+1})}$  is the cubic posiform associated to  $\mathcal{R}$ , for all  $k = 0, \dots, n-1$ .

Thus,

$$\begin{aligned} f_{i_0} &= \sum_{j=0}^{(k-1)} \left( \rho_{i_j} + \varphi_{f_{i_j}}^{(i_{j+1})} \right) + f_{i_k} \\ &= \sum_{j=0}^{(k-1)} \rho_{i_j} + \sum_{j=0}^{(k-1)} \varphi_{f_{i_j}}^{(i_{j+1})} + f_{i_k} \end{aligned} \tag{8.9}$$

for all  $k = 1, \dots, n$ .

**Lemma 8.4.**

$$\beta_f = \sum_{j=0}^{(n-1)} \rho_{i_j} + \sum_{j=0}^{(n-1)} \varphi_{f_{i_j}}^{(i_{j+1})}$$

*Proof.* The claim follows directly from (8.9) since  $f_{i_0} = \beta_f$  and  $f_{i_n} = 0$ . To see that

$f_{i_n} = 0$  we use the fact that  $f_{i_n}$  is a bi-form that corresponds to the residual balancing graph with at most one vertex (i.e. with index  $i_n$ ).  $\square$

**Proposition 8.8.** *Given a quadratic pseudo-Boolean function  $f$  then*

$$\min_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x}) \geq C_3(f) \geq c_0 + \rho_0 + \sum_{j=1}^{(n-1)} \rho_{i_j} \geq \rho(f).$$

*Proof.* The second inequality follows due to Lemma 8.4 and because the set of rooted noose packing (with roots  $x_0, x_{i_1}, \dots, x_{i_{n-2}}$ ) is a solution to the noose packing problem (NP) and consequently Proposition 8.5 applies. The last inequality follows from the fact that  $\rho_0 = \rho(f) - c_f$  and because  $\rho_{i_j} \geq 0, j = 1, \dots, n - 1$ .  $\square$

**Definition 8.9.** *The lower bound*

$$\hat{\rho}(f; i_0, \dots, i_n) = c_f + \sum_{j=0}^{(n-1)} \rho_{i_j}$$

has been called by Boros et al. [51, 52] as the iterated roof-dual of quadratic pseudo-Boolean function  $f$ .

The iterated roof-dual value clearly depends on the sequence of roots  $(i_0, \dots, i_n)$  that is considered to define the above heuristic to the noose packing problem. In particular, if the first root selected is  $x_0$  then

$$\hat{\rho}(f; 0, i_1, \dots, i_n) = \rho(f) + \sum_{j=1}^{(n-1)} \rho_{i_j}.$$

Figure 8.3 describes algorithm ITERATED-ROOF-DUAL to compute the iterated roof-dual value of any quadratic pseudo-Boolean function represented as a bi-form network.

#### 8.4.1 Computational results

An extensive computational experimentation of algorithm ITERATED-ROOF-DUAL using several benchmark problems available in the literature has been carried out and are presented in this section. The results consider a particular implementation that considers a method  $\mathbb{X}$  to find a root  $x_k$  which is the vertex in the residual balancing

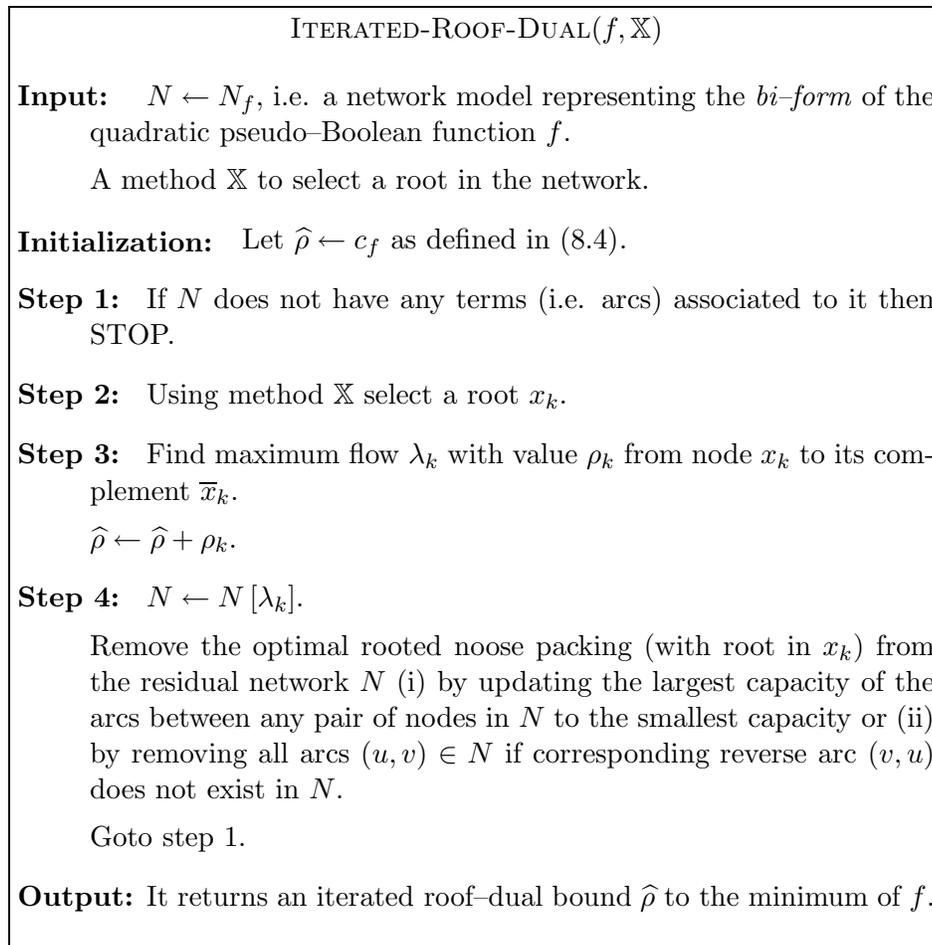


Figure 8.3: ITERATED-ROOF-DUAL algorithm.

graph having the largest total weight to its neighbors. Also the root considered in the first iteration of the method is  $x_0$ . This method was suggested originally by Boros et al. [51, 57]. We denote the resulting bound as  $\hat{\rho}$ . We have tested other methods to select roots, but in general the variability of the bound with different choices was not very significant.

Let us note that the maximization and the minimization of quadratic pseudo-Boolean functions are equivalent problems, since minimizing  $f$  is equivalent with maximizing  $-f$ . Thus, the presented results can be applied directly to maximization problems, as well. In this section we present our computational results in the context of maximization.

In order to compare the quality and the computing times obtained with our implementations of the roof-dual and the iterated roof dual algorithms, we used the *Semi-Definite Relaxation*, or SDR (see Goemans and Williamson [112]). There are many publicly available semidefinite solvers on the Internet. Each solver has strengths and weaknesses, which are very much dependent on the type and size of the problem to be solved. There is no solver which clearly dominates the others in all aspects, e.g. robustness, memory management, or solution speed (see [179]). We have used the following solvers that have been proposed to solve *SemiDefinite Programs*, or SDPs:

- SDPA** – SDPA is a software package for solving SDPs (see e.g. [102]). It is an implementation of a Mehrotra-type primal-dual predictor-corrector interior-point method. The Windows version 6.2.1 of SPDA was used in this study.
- DSDP** – DSDP is a software implementation of the dual interior-point method for SDP (see e.g. [39]). It provides primal and dual solutions, exploits low-rank structure and sparsity in the data, and has relatively low memory requirements for an interior-point method. The version 5.8 of DSDP was used in this study.
- SBM** – SBM is a software implementation of the spectral bundle method (see e.g. [138, 140]), for minimizing the maximum eigenvalue of an affine matrix function (real and symmetric). The code is suited for large scale problems. It allows to exploit structural properties of the matrices such as sparsity and low rank structure. The

version 1.1.3 of SBM was considered in this study.

We did not have a special preference for selecting any of the above methods. Our goal was to cover a variety of SDR solution methods, e.g. by using a robust method that handles small and medium sized problems (like SPDA), by using a method that handles larger problems with a sparse structure (like DSDP), and by adopting a method (like SBM) that is quicker than the others (although this speedup is achieved at the price of obtaining an approximate solution to SDR).

Finally, we should mention that for a restricted number of problems we have added computational experience also for the case when the  $C_3$  bound was used. Other than by using linear programming, we are unaware of any other approach that could provide  $C_3$  in polynomial time. Since solving these large LP problems is time and memory consuming, we computed this bound only for one family of problems.

The method used to solve the LPs is the Newton–Barrier algorithm that comes with Xpress–MP 2005B (release 16.10.02). Xpress–MP is a suite of mathematical modeling and optimization tools used to solve linear, integer, quadratic, non-linear, and stochastic programming problems (e.g. see [21]). The presolve and the crossover was turned off in all runs. In this section we use XPRESS to identify the results returned by this particular LP solver, with the options previously mentioned.

The Roof–Dual and the Iterated Roof–Dual Algorithms (respectively called RDA and IRDA hereafter) were implemented in C++, compiled using the Microsoft Windows 32-bit C/C++ Optimizing Compiler (version 12) for 80x86, and linked with the Microsoft Incremental Linker (version 6) using the single–threaded run–time library.

Three computer systems were used for testing. The decision to use this many computers and not one, is related to licensing requirements, operating system restrictions, and amount of physical memory available. Table 8.1 shows the main characteristics of each system, and also shows what algorithm(s) were tested in each one of them. The three platforms are comparable in terms of speed with a maximum speedup smaller than two, between the fastest (Computer III) and the slowest machine (Computer I).

Table 8.1: Characteristics of computer systems used for testing the algorithms.

<i>Computer Systems</i>	<i>I</i>	<i>II</i>	<i>III</i>
<i>CPU</i>	Intel Pentium 4	Xeon	Intel Pentium 4
<i>Clock Speed</i>	2.8 GHz	3.06 GHz	3.6 GHz
<i>Hyper-Threading?</i>	yes	no	yes
<i>RAM</i>	512 MB	3.5 GB	2 GB
<i>Cache</i>	$L_2$ 512 KB	$L_2$ 512 KB	$L_2$ 1MB
<i>Operating System</i>	Wind. XP <sup>†</sup>	Linux <sup>‡</sup>	Wind. XP <sup>†</sup>
<i>Algorithms Tested</i>	RDA, IRDA, SDPA	DSDP, SBM	XPRESS

<sup>†</sup>Microsoft Windows XP Professional version 5.1.2600 (service pack 2)

<sup>‡</sup>Fedora Core Linux 2.6.9-1.667smp i686

#### 8.4.1.1 MAX-CUT

In this study we included the analysis of two large groups of problems, one involving maximum cut problems (or MAX-CUT in short) on graphs (known to be equivalent to quadratic unconstrained binary maximization problems [52]), and another using randomly generated quadratic binary optimization problems proposed by Glover et al. [108] and Beasley [37].

The families of graphs used in the MAX-CUT experiments are listed below:

$G_{n,d}$  – Random graphs proposed by Kim and Moon [157];

$U_{n,d}$  – Random geometric graphs proposed by Kim and Moon [157];

$R_n$  – Sparse random graphs proposed by Homer and Peinado [145];

*via* – Graphs provided by Homer and Peinado [145], derived from layer assignment problems in the design process for VLSI chips;

$sg3dl_L$  – 3D-toroidal graphs proposed by Burer, Monteiro and Zhang [74];

*torus* – 3D-toroidal graphs taken from the DIMACS library of mixed semidefinite-quadratic-linear programs.

The characteristics of these graphs are briefly described in Section 3.4.

The first experiments concerned the finding of upper bounds for MAX-CUT in the 16 graphs of Kim et al. [157]. The results are shown in Tables D.1(a) and D.1(b) in

the Appendix. The tables show that in almost all these 16 cases the upper bound given by the semidefinite relaxation was slightly better (on the average 7.1% lower) than that given by the iterated roof-dual, and better than (on the average 27% lower) the roof-dual bound. The difference in the quality of the bounds was amply compensated by the computing times needed to find them. Indeed, on the average, the time needed by DSDP (the most efficient of the three implementations of semidefinite relaxations) was of 18.5 seconds, while that needed by the iterated roof-dual algorithm was of only 1.5 seconds, and that needed by the roof-dual algorithm was of 0.01 seconds.

Turning now to the MAX-CUT problem for the graphs of Homer and Peinado [145] (see Tables D.2(a) and D.2(b) in the Appendix) we notice that the comparative values of roof-dual-based versus semidefinite-relaxation-based upper bounds differ substantially between the group  $R$  of random graphs, and the group of *via* graphs coming from VLSI design. For the group  $R$ , the upper bounds of SDR are 21.1% better than those of roof-duality, and 8.1% better than those coming from iterated roof-duality. The situation of the *via* graphs is quite different, since the three upper bounds are quite comparable within this group. More precisely, the upper bounds of SDR are only 1.7% better than those of roof-duality, but the upper bounds of iterated roof-duality are 0.5% better than those of SDR. As far as computing times go the average time needed by SBM (the most efficient of the three implementations of SDR for the group of  $R$  graphs) was of 477.4 seconds, while for RDA the average time is 0.02 seconds and for IRDA it is of 35.8 seconds. For the group of *via* graphs, the average time needed by DSDP (the most efficient of the three implementations of SDR for this group) was of 42 seconds, while for RDA the average time is 0.03 seconds and for IRDA it is of 0.13 seconds.

The next group of MAX-CUT problems concerns cubic lattice graphs (similar in structure to the graphs appearing in Ising problems) of Burer et al. [74] (see Tables D.3(a) and D.3(b) in the Appendix). It can be seen that for these graphs, the upper bound given by the iterated roof-dual is 1.4% better than that given by SDR, while the SDR bound is 31.5% better than that given by the roof-dual bound. It is interesting to note that the computing time required by SBM (the most efficient of the three implementations of SDR for the cubic lattices) for finding the upper bound associated

to an average graph in the family was of 42.7 seconds, while that of IRDA was of 2.7 seconds, and that of RDA was less than 0.01 seconds.

The last group of MAX-CUT problems examined are associated to torus graphs (having also a similar structure to that of the graphs appearing in Ising problems) proposed at the 7th DIMACS Implementation Challenge on Semidefinite Programming, which are frequently used as benchmarks in computational studies concerning semidefinite programming (e.g. [73, 74, 137, 179]; see Tables D.4(a) and D.4(b) in the Appendix). For these graphs, the upper bound given by the iterated roof-dual bound is 1.4% better than that given by SDR, which in its turn is 26.9% better than that given by roof-duality. The average computing time required by SBM (the most efficient of the three implementations of SDR for the torus graphs) is 115.7 seconds, while that required by IRDA is of 4.5 seconds, and that required by RDA is of about 0.01 seconds.

In summary (see Tables 8.2(a) and 8.2(b)), the upper bounds for MAX-CUT examined in this study present the following characteristics:

- In the case of  $G$ ,  $U$  and  $R$  graphs the best bounds are obtained by SDR;
- In the case of  $via$ ,  $sg3dl$  and  $torus$  graphs the best bounds are obtained by IRDA;
- The shortest computing times are those of RDA, followed by those of IRDA. The average computing time per graph is of 92.3 seconds for SDP (the fastest implementation of the considered semidefinite programs), 6.0 seconds for IRDA, and about 0.014 seconds for RDA.

#### 8.4.1.2 Randomly generated quadratic binary optimization problems

The second group of problems includes standard randomly generated families of problems, having a constant density (i.e., proportion of coefficients with value zero) and having all nonzero coefficients from a closed interval. The following two families were considered:

- $D$  – A set of 10 QUBO problems proposed by Glover et al. [108] having 100 variables per problem, and densities varying from 10% to 100% in steps of 10%;

Table 8.2: Bounding MAX-CUT.

(a) Average relative gap ( $g$ ) to the largest known cut ( $z$ ).

<i>Family</i>	<i>Number of Problems</i>	<i>SDR Gap</i> ( $g = \varsigma/z - 1$ )	<i>Roof-Dual Gap</i> ( $g = \rho/z - 1$ )	<i>Iter. Roof-Dual Gap</i> ( $g = \hat{\rho}/z - 1$ )
<i>G</i> graphs	8	<b>5.68%</b>	28.79%	11.04%
<i>U</i> graphs	8	<b>2.64%</b>	56.28%	13.24%
<i>R</i> graphs	8	<b>7.31%</b>	36.01%	16.72%
<i>via</i> graphs	10	0.53%	2.32%	<b>0.02%</b>
<i>sg3dl</i> graphs	30	14.86%	67.65%	<b>13.27%</b>
<i>torus</i> graphs	4	12.47%	54.26%	<b>10.88%</b>

(b) Average computing times.

<i>Family</i>	<i>Best</i>	<i>Roof-Dual Algorithms</i>	
	<i>SDP**</i>	<i>RDA*</i>	<i>IRDA*</i>
<i>G</i> graphs	23.2 s	<0.01 s	0.9 s
<i>U</i> graphs	13.8 s	<0.01 s	2.2 s
<i>R</i> graphs	477.4 s	<0.02 s	35.8 s
<i>via</i> graphs	42.0 s	<0.03 s	0.1 s
<i>sg3dl</i> graphs	42.7 s	<0.01 s	2.7 s
<i>torus</i> graphs	115.7 s	<0.01 s	4.5 s

\*Computed on computer system I.

\*\*Computed on computer system II.

$B - n$  – A set of 60 QUBO test problems with 10% density proposed by Beasley [37] ( $n = 50, 100, 250, 500, 1000, 2500$ ; 10 problems for each value of  $n$ ). The problems with 50 variables turned to be all solved to optimality by applying iterated roof-duality (and in most cases even by applying roof-duality). Therefore, these problems have been eliminated from the study.

The previous families are briefly described in Section 3.1.1.

We present in Table 8.3(a) the maximum values of some randomly generated quadratic functions with binary variables, along with the values of four upper bounds to the maximum (SDR, RDA, IRDA and C3), expressed as percentages of the values of the corresponding exact maxima.

It can be seen that the best bounds for problems with densities of at most 40% were provided by the cubic-dual (averaging 2.2% over the maximum), while for problems having densities of 50% or higher the best upper bounds were given by SDR (averaging 7.6% above the maximum). On the other hand, the best computing times (see Table

Table 8.3: Iterated roof-duals of QUBO problems with 100 variables (Glover et al. [108]).

(a) Upper bounds.

Problem Name	Density ( $d$ )	Maximum	Upper Bounds to the Maximum			
			Semidefinite Relaxation	Roof Dual ( $\rho$ )	Iter. Roof Dual ( $\hat{\rho}$ )	Cubic Dual
1d	10%	6 333	6 592.77	7 063.50	6 424.50	<b>6333.00</b>
2d	20%	6 579	7 234.24	12 297.00	7 791.00	<b>6709.82</b>
3d	30%	9 261	9 962.97	18 053.50	10 875.25	<b>9374.79</b>
4d	40%	10 727	11 592.46	25 156.50	13 425.50	<b>11321.82</b>
5d	50%	11 626	<b>12632.10</b>	30 732.00	15 538.13	13 044.50
6d	60%	14 207	<b>15235.31</b>	37 334.50	18 041.50	15 664.33
7d	70%	14 476	<b>15671.97</b>	44 171.50	20 614.75	18 340.00
8d	80%	16 352	<b>17353.30</b>	50 239.50	22 723.50	20 625.67
9d	90%	15 656	<b>17010.86</b>	55 130.00	24 109.00	21 753.67
10d	100%	19 102	<b>20421.35</b>	63 830.50	28 370.50	25 951.67

(b) Computing times.

Problem	Semidefinite Programs			Roof-Dual Algorithm		Cubic Dual LP
	DSDP**	SBM**	SDPA*	RDA*	IRDA*	XPRESS***
1d	0.22 s	1.72 s	1.03 s	<0.005 s	<0.005 s	118 s
2d	0.26 s	2.68 s	1.06 s	<0.005 s	0.02 s	164 s
3d	0.23 s	4.91 s	1.08 s	<0.005 s	0.05 s	173 s
4d	0.22 s	3.16 s	1.11 s	<0.005 s	0.08 s	143 s
5d	0.22 s	4.95 s	1.17 s	0.02 s	0.11 s	122 s
6d	0.20 s	8.90 s	1.08 s	<0.005 s	0.14 s	74 s
7d	0.22 s	6.87 s	1.17 s	0.02 s	0.20 s	71 s
8d	0.23 s	11.19 s	1.19 s	0.02 s	0.25 s	72 s
9d	0.23 s	8.78 s	1.20 s	<0.005 s	0.27 s	68 s
10d	0.23 s	9.35 s	1.25 s	<0.005 s	0.33 s	69 s

\*Computed on computer system I.

\*\*Computed on computer system II.

\*\*\*Computed on computer system III.

8.3(b)) were achieved by RDA (averaging less than 0.01 seconds) and IRDA (averaging less than 0.2 seconds). It follows that for problems which have low densities, the most efficient methods may be those based on roof-duality. It is worth noting that numerous problem classes (e.g., minimum vertex covers of planar graphs or power-law graphs, MAX-CUT of Ising problems) belong to this category.

In Table D.5 in the Appendix we present three upper bounds obtained by SDR, roof-duality and iterated roof-duality for the 10% dense quadratic unconstrained binary optimization problems of Beasley [37] having up to 2500 variables. It can be seen that for the “small” problems, i.e. those with 100 variables, the bounds given by IRDA are the best among the three upper bounds considered; the average gap between IRDA and the true maximum of the function is of 3.3% (see Table 8.4(a)). However, for problems having 250 or more variables the best bounds are those given by SDR; the average gap between SDR and the best known solution (representing a lower bound to the maximum) is of 8.9%. As in the previous cases the computing times of the different methods follow a clear pattern. The average time needed by DSDP (the fastest of the three SRD procedures; see Table 8.4(b)) is of 1736.3 seconds. For the same problems, the average time required by RDA is of 0.4 seconds, and by IRDA is of 88.2 seconds.

The above results demonstrate that the iterated roof dual bound can be computed very efficiently with the proposed IRDA implementation. The computing time of this bound is much faster than the computation of semidefinite bounds or the cubic dual. We can also see that the quality of the iterated roof dual bound is highly competitive with other approaches. In particular, for sparse problems, which are quite frequent in applications, these bounds are superior to all other methods we tested, and can be computed on average 20-50 times faster than those. We can also see that the cubic dual bound is the best on a larger range of mostly sparser problems. However its time complexity makes its application impractical for larger problems.

In spite of the fact that the bounds given by semidefinite relaxation are of high quality, the time and memory requirements of the roof-duality based methods being substantially smaller, assure the practical applicability of this latter group of methods,

Table 8.4: Iterated roof-duals of 10% dense quadratic unconstrained binary optimization problems (Beasley [37]).

(a) Average relative gap ( $g$ ) to the best known lower bound ( $z$ ).

<i>Family</i>	<i>Variables</i> ( $n$ )	<i>SDR Gap</i> ( $g = \frac{s-z}{z}$ )	<i>Roof-Dual</i> <i>Gap</i> ( $g = \frac{\rho-z}{z}$ )	<i>Iter. Roof-Dual</i> <i>Gap</i> ( $g = \frac{\hat{\rho}-z}{z}$ )
ORL-100	100	6.3%	15.3%	<b>3.3%</b>
ORL-250	250	<b>7.6%</b>	78.1%	18.5%
ORL-500	500	<b>9.0%</b>	150.6%	41.6%
ORL-1000	1 000	<b>9.5%</b>	248.8%	73.0%
ORL-2500	2 500	<b>9.4%</b>	430.4%	129.1%

(b) Average computing times.

<i>Family</i>	<i>Semidefinite Programs</i>			<i>Roof-Dual Algorithms</i>	
	<i>DSDP**</i>	<i>SBM**</i>	<i>SDPA*</i>	<i>RDA*</i>	<i>IRDA*</i>
ORL-100	0.21 s	2.55 s	1.01 s	<0.005 s	0.01 s
ORL-250	2.20 s	28.88 s	14.98 s	0.02 s	0.24 s
ORL-500	24.97 s	115.56 s	131.21 s	0.05 s	2.38 s
ORL-1000	269.29 s	673.49 s	1 096.21 s	0.20 s	21.47 s
ORL-2500	8 385.05 s	107 623.67 <sup>‡</sup> s	n/a <sup>†</sup>	1.49 s	416.78 s

\*Computed on computer system I.

\*\*Computed on computer system II.

<sup>†</sup>Memory exceeded for all problems with 2 500 variables.

<sup>‡</sup>Computing time found for the first problem only.

and guarantee their high efficiency. Moreover, in view of the typical sparsity of real life quadratic unconstrained binary optimization problems, the use of roof-duality based methods is both effective and efficient.

## 8.5 Squeezed iterated roof-duality

The computational experiments that we have carried out (see e.g. Table 8.3(a)) clearly indicate that the iterated roof-duals of various benchmarks are still not nearly close to the corresponding  $C_3$  bounds. An algorithm is proposed here to improve the iterated roof-duality bounds by means of a combinatorial approach that we called as the *squeezed iterated roof-dual* bound.

Before getting into details, we recall the fact that any bi-form can be decomposed into a special cubic posiform and a residual bi-form, as explained in Section 8.3. The key ingredient of the enhanced iterated roof-duality method is to reuse part of the cubic

posiform of (8.9) on any given iteration of the standard iterated roof–dual algorithm. This in practice means that certain parts of the rooted noose packings that have already been applied are added to the residual bi–form to increase the chances of improving the bound on the next iteration of the method.

**Definition 8.10.** *Let us define the mate of a posiform  $\phi(x_1, \bar{x}_1, \dots, x_n, \bar{x}_n)$  as the posiform  $\phi(\bar{x}_1, x_1, \dots, \bar{x}_n, x_n)$ . If a posiform is equivalent to its mate then we say that the posiform is mated.*

In particular, the mate of the cubic term  $uvw$  is  $\bar{u}\bar{v}\bar{w}$ , for any literals  $u, v$  and  $w$ .

Let us consider the *cubic* posiforms  $\varphi_k, k = 0, \dots, n$  that (i) consists of terms that have mates on the same posiform with the same coefficients and (ii) has a literal  $x_k$  or its complement in every of its terms

**Definition 8.11.** *We shall call rotation to the operation of identifying a “maximal” set of terms derived from  $\sum_{k=0}^n \varphi_k$  that contains variable  $x_j, j \neq k$ . The rotation principle is based on the identity*

$$x_k(x_j\bar{u} + u\bar{v}) + \bar{x}_k(\bar{x}_j\bar{u} + uv) = x_j(x_k\bar{v} + \bar{u}v) + \bar{x}_j(\bar{x}_k v + u\bar{v}). \quad (8.10)$$

Clearly, the left hand side of (8.10) has two terms involving variable  $x_j$ , but the right hand side has four terms involving the same variable. The two extra terms per rotation increase the probability of improving the bound since the chance of finding additional nooses rooted at  $x_j$  (and with larger values) is also increased, when they are added to the residual balancing graph.

A very important aspect of the algorithm is how to save and handle the residual cubic posiform  $\Psi_f = \sum_{k=0}^n \varphi_k$ . The following characteristics are desirable:

- Given a variable  $x_j$  efficiently determine terms where it appears;
- Given a variable  $x_j$  efficiently remove terms where it appears;
- Given a cubic term  $x_j x_k \bar{u}$  efficiently identify all cubic terms  $v x_k u$  in the posiform;
- Apply quick rotation operations between any cubic terms  $x_j x_k \bar{u}$  and  $v x_k u$ ;

- Use memory as economically as possible.

It is not a trivial task to create a structure that satisfies all the above points. It is particular computationally difficult to identify a set of maximal rotations involving a certain variable.

**Lemma 8.5.** *For any variable  $x_j$  the procedure of applying all possible rotations of  $x_j$  in  $\Psi_f$  can be determined in  $O(n^3)$  time.*

*Proof.* The method is a finite procedure, since (i) there is a finite number of terms in  $\Psi_f$  (at most  $2\binom{n}{3}$ ) and (ii) the number of terms is monotonically increasing with respect to every rotation procedure.  $\square$

Let  $\text{ROTATION}(x_j, \Psi_f)$  be a procedure that computes a cubic posiform

$$\Psi_f^* = x_j \psi_f^{(j)}(x_0, \dots, x_{j-1}, x_j, \dots, x_n) + \bar{x}_j \psi_f^{(j)}(\bar{x}_0, \dots, \bar{x}_{j-1}, \bar{x}_j, \dots, \bar{x}_n) + \Psi'_f$$

that represents the same function  $f$  and for which there are no cubic terms involving a rotation of  $x_j$  that increases the number of terms involving this variable. Thus,  $\varphi_f^{(j)}$  is maximal with respect to possible additional  $x_j$  rotations.

**Remark 8.5.** *It is not known if applying  $x_k$  rotations ( $k \neq j$ ) would result in finding possible additional  $x_j$  rotations.*

**Conjecture 8.1.** *Let  $f$  be a quadratic pseudo-Boolean function  $f$  represented (in what follows) as  $f = c_f + \delta + \Psi + \theta$ , where  $\delta$  is a nonnegative number,  $\Psi$  is a mated cubic posiform and  $\theta$  is a bi-form. Then, the following combinatorial algorithm determines  $C_3(f)$ .*

*Initially,  $\Psi = \delta = 0$  and  $\theta = \beta_f$ . For every variable  $x_j, j = 0, \dots, n$ :*

- Find a maximal mated cubic posiform  $\Psi_j$  in terms of the number of terms that contain  $x_j$ , so that  $\Psi = \Psi^* + \Psi_j$ ;*
- The cubic component of  $f$  is updated  $\Psi \leftarrow \Psi^*$ ;*

(iii) Noting that  $\Psi_j$  represents a quadratic pseudo-Boolean function, then the bi-form component of  $f$  is updated as  $\theta \leftarrow \theta + \Psi_j$ ;

(iv) Find the optimal  $x_j$  rooted noose packing  $\mathcal{R}$  of  $\theta$ ;

(iv) Determine the  $\mathcal{R}$ -decomposition of the bi-form as  $\theta = \xi_j + \varphi_j + \phi_j$ . Update  $\theta \leftarrow \phi_j$ ,  $\Psi \leftarrow \Psi + \varphi_j$  and  $\delta \leftarrow \delta + \xi_j$ ;

At the end of the algorithm we conjecture that  $C_3(f) = c_f + \delta$ .

In practice the above algorithm will clearly determines a better bound than the iterated roof-dual. It will also determine a bound which is not better than the cubic-dual bound. We leave the above result as a conjecture to motivate other researchers to pursue the finding for the first time of an efficient combinatorial problem to compute  $C_3$ . Even if the above conjecture turns out not to be true, there are several related open questions:

- Is there a polynomial time algorithm to squeeze out a maximal set of  $x_j$ -terms ( $j = 0, \dots, n$ ) of the mated cubic posiform?
- Is the maximal set of the previous question also maximum possible?
- Is the maximum (or maximal) set sufficient to obtain the maximum bound possible from the residual bi-form?
- Does the order of the various rooted noose packings matter?
- Is only one rooted noose packing iteration per variable required or more?

In this dissertation, we developed a “squeezed” iterated roof-duality algorithm that achieves only partially the above results. The pseudo-code of the algorithm is given in Figure 8.4.

Firstly, the data structure used to maintain the mated cubic posiform  $\Psi$  is a a map of roots  $x_k$  ( $k = 0, \dots, n$ ) to the corresponding quadratic residuals  $\psi_f^{(k)}$  as described in Proposition 8.7. To be noted that the mate information is not saved directly on any structure; given any quadratic term  $a_{uv}uv$  of  $\psi_f^{(k)}$  then from the map we would get the

mated cubic terms  $a_{uv}(x_k uv + \bar{x}_k \bar{u} \bar{v})$ . The quadratic residuals structure is based on the network model of Section 5.3. This has certain advantages to efficiently find  $x_j$  rotations in  $\psi_f^{(k)}$ , since valid rotations correspond to paths of length 2 from  $x_j$  in the network.

Secondly, the rotation principle is only applied on a limited way since it only considers cubic mated terms within the scope of the  $\psi_f^{(k)}$  residuals.

Thirdly, we reapply the squeezing procedure a user specified number of rounds per root.

### 8.5.1 Computational results

The SQUEEZED-ITERATED-ROOF-DUAL algorithm (called SIRDA hereafter) has been implemented in C++ and linked as the IRDA.

Comparative results between RDA (Roof-dual), IRDA (Iterated Roof-Dual) and SIRDA are presented in this section. We start by looking again at the 10 QUBO problems created by Glover et al. [108] already considered in Section 8.4.1 (see Tables 8.3(a) and 8.3(b)).

Table 8.5(a) gives the squeezed iterated roof-duals of family  $D$  from Glover et al. [108]. The results include versions of the SQUEEZED-ITERATED-ROOF-DUAL algorithm with 1, 2 and 3 rounds. The quality of the “squeezed” bounds is clearly superior to those returned by iterated roof-duality. The improvement of the squeezed bounds is also noticeable between consecutive rounds, but with a smaller improvement percentage as the number of rounds increases. The results also shown that  $C_3$  is still superior to any of the squeezed iterated roof-duals.

Table 8.5(b) provides the computing times of our implementation of the squeezed iterated roof duals (we called it SIRDA), again considering 1, 2 and 3 rounds. As expected, the iterated roof-duality bounds returned by IRDA can be found considerably faster than those returned by SIRDA. This difference in computing times is more noticeable for sparser problems. For instance, for the 10% dense instance IRDA is over 50 times faster than any of the SIRDA runs, whereas for the 100% dense instance, IRDA is only about 5 times faster.

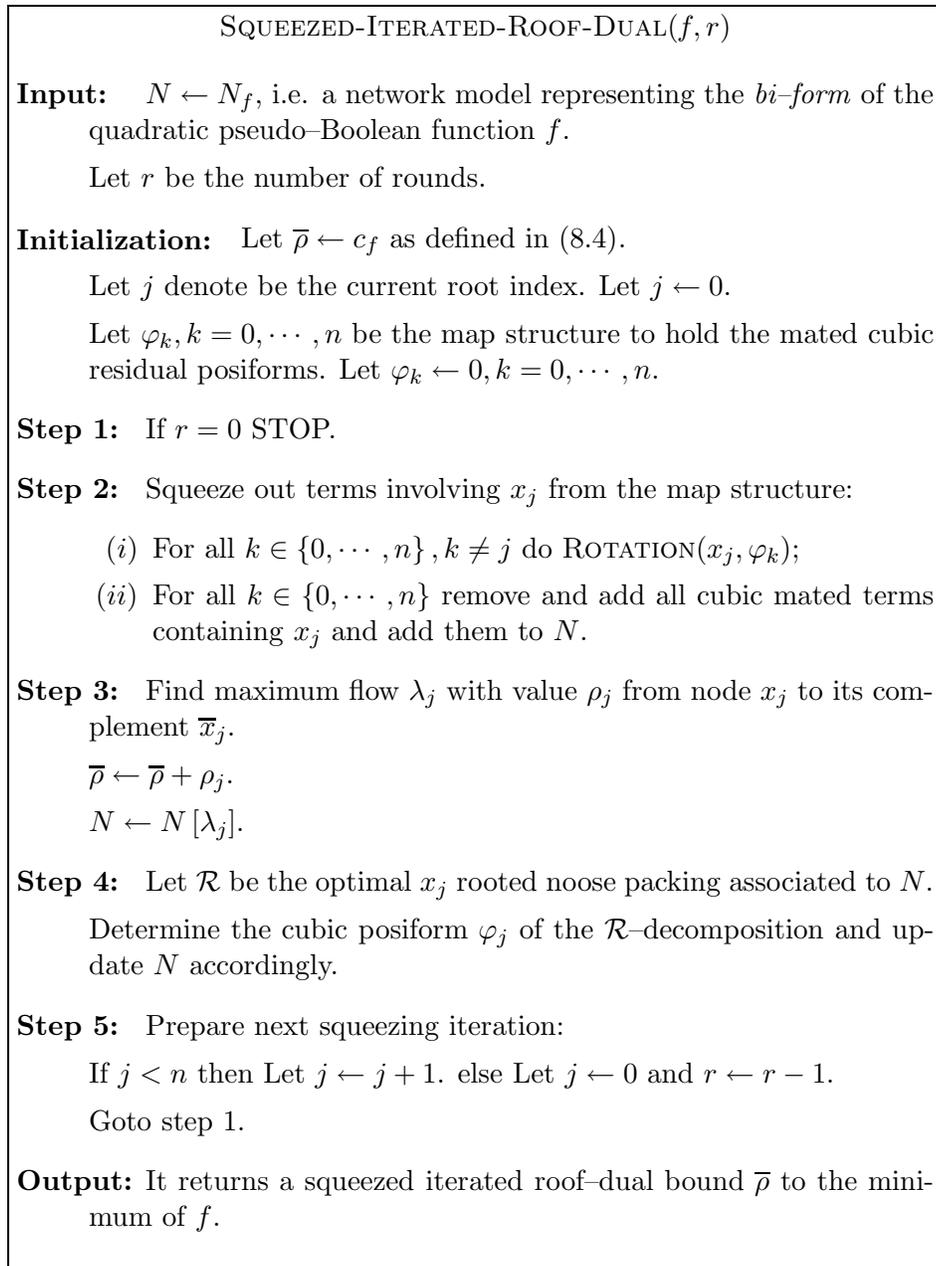


Figure 8.4: SQUEEZED-ITERATED-ROOF-DUAL algorithm.

Table 8.5: Squeezed iterated roof-duals of QUBO problems with 100 variables (Glover et al. [108]).

(a) Upper bounds.

Problem Name	Density (d)	Maximum	Upper Bounds to the Maximum					
			Roof Dual ( $\rho$ )	Iter. Roof Dual ( $\hat{\rho}$ )	Squeezed Iterated Roof-Dual			Cubic Dual ( $C_3$ )
					( $\bar{\rho}; r = 1$ )	( $\bar{\rho}; r = 2$ )	( $\bar{\rho}; r = 3$ )	
1d	10%	6 333	7 063.50	6 424.50	6 343.25	6 340.22	6 340.22	6 333.00
2d	20%	6 579	12 297.00	7 791.00	7 364.25	7 193.29	7 135.98	6 709.82
3d	30%	9 261	18 053.50	10 875.25	10 364.31	10 165.36	10 062.67	9 374.79
4d	40%	10 727	25 156.50	13 425.50	12 918.92	12 496.26	12 358.87	11 321.82
5d	50%	11 626	30 732.00	15 538.13	14 942.95	14 494.67	14 370.52	13 044.50
6d	60%	14 207	37 334.50	18 041.50	17 759.85	17 214.67	17 016.77	15 664.33
7d	70%	14 476	44 171.50	20 614.75	20 045.04	19 318.99	19 114.80	18 340.00
8d	80%	16 352	50 239.50	22 723.50	22 422.55	21 644.35	21 432.43	20 625.67
9d	90%	15 656	55 130.00	24 109.00	23 447.48	22 628.34	22 436.98	21 753.67
10d	100%	19 102	63 830.50	28 370.50	27 851.14	26 830.85	26 630.88	25 951.67

(b) Computing times.

Problem	Roof-Dual Algorithms					Cubic Dual LP
	RDA*	IRDA*	SIRDA* ( $r = 1$ )	SIRDA* ( $r = 2$ )	SIRDA* ( $r = 3$ )	XPRESS**
1d	<0.005 s	<0.005 s	0.44 s	0.52 s	0.61 s	118 s
2d	<0.005 s	0.02 s	0.72 s	2.25 s	3.86 s	164 s
3d	<0.005 s	0.05 s	0.72 s	2.14 s	3.91 s	173 s
4d	<0.005 s	0.08 s	0.78 s	2.56 s	4.83 s	143 s
5d	0.02 s	0.11 s	0.89 s	2.89 s	5.42 s	122 s
6d	<0.005 s	0.14 s	1.08 s	3.44 s	6.23 s	74 s
7d	0.02 s	0.20 s	1.16 s	3.53 s	6.38 s	71 s
8d	0.02 s	0.25 s	1.33 s	3.77 s	7.00 s	72 s
9d	<0.005 s	0.27 s	1.44 s	4.17 s	7.53 s	68 s
10d	<0.005 s	0.33 s	1.73 s	4.30 s	7.86 s	69 s

\*Computed on computer system I (see Table 8.1).

\*\*Computed on computer system III (see Table 8.1).

Table 8.6: Squeezed iterated roof-duals of 10% dense quadratic unconstrained binary optimization problems (Beasley [37]).

(a) Average relative gap ( $g$ ) to the best know lower bound ( $z$ ).

Family	Variables ( $n$ )	Roof-Dual	Iter. Roof-Dual	Squeezed Iter. Roof-Dual ( $g = \frac{\hat{p}^* - z}{z}$ )		
		( $g = \frac{p - z}{z}$ )	( $g = \frac{\hat{p} - z}{z}$ )	( $r = 1$ )	( $r = 2$ )	( $r = 3$ )
ORL-100	100	15.3%	3.3%	0.9%	0.4%	0.4%
ORL-250	250	78.1%	18.5%	12.9%	10.9%	9.1%
ORL-500	500	150.6%	41.6%	35.0%	32.4%	31.5%
ORL-1000	1000	248.8%	73.0%	64.9%	60.7%	59.7%

(b) Average computing times.

Family	RDA*	IRDA*	SIRDA*		
			( $r = 1$ )	( $r = 2$ )	( $r = 3$ )
ORL-100	<0.005 s	0.01 s	0.38 s	0.68 s	0.97 s
ORL-250	0.02 s	0.24 s	4.42 s	17.54 s	35.86 s
ORL-500	0.05 s	2.38 s	15.99 s	79.99 s	205.65 s
ORL-1000	0.20 s	21.47 s	96.41 s	338.39 s	923.13 s

\*Computed on computer system I (see Table 8.1).

To have an idea how the squeezed iterated roof-dual bounds scale with the number of variables, we apply SIRDA to the 10% dense QUBOs of Beasley [37] having up to 1000 variables. The average relative gaps of various sub-families (for which all problems have the same number of variables) to the best know solutions are given in Table 8.6(a). The corresponding average computing times are given in Table 8.6(b).

The average relative gap of the squeezed bounds of the 100 variable problems is less than 1.0% (for  $r \leq 2$  is 0.4%), instead of 3.3% for IRDA. In this case all the bounds could be obtained within one second, thus implying that most likely these problems could be solved to optimality by SIRDA without branching substantially. We will get back to this topic in the next chapter.

In spite of the fact that the relative gaps increase considerably with the number of variables, it should be noted that the improvement from the IRDA to the SIRDA results is larger for the later algorithm. In particular, the differences from IRDA to SIRDA ( $r = 2$ ) are: 2.9% for the 100 variable group, 7.6% for the 250 variable group, 9.2% for the 500 variable group and 12.3% for the 1000 variable group.

As far as computing time concerns, the slow down factor from IRDA to SIRDA

( $r = 2$ ) are: 68 times for the 100 variable group, 73 times for the 250 variable group, 33 times for the 500 variable group and 16 times for the 1 000 variable group.

More results about the squeezed iterated roof-duality are presented in the following section, which considers a “non-bi-form” squeezed iterated roof duality algorithm.

## 8.6 Project-and-lift iterated roof-duality

In the previous sections we have seen that the bi-form representation of a quadratic pseudo-Boolean function can provide substantially improved bounds over that one returned by roof-duality. In this section we consider general posiform representations to get improved bounds over roof-duality.

The following algebraic equation is due to Simeone [222].

**Lemma 8.6.** *The identity*

$$uv + \bar{v}\bar{w} = u\bar{w} + uvw + \bar{u}\bar{v}\bar{w},$$

is valid for any 3 literals  $u$ ,  $v$  and  $w$ .

**Definition 8.12.** *The operation that transforms a posiform  $\theta$  with positive terms  $\alpha_{uv}uv$  and  $\alpha_{\bar{v}\bar{w}}\bar{v}\bar{w}$  to a new posiform*

$$\theta [uv, \bar{v}\bar{w}] = \theta - \alpha (uv + \bar{v}\bar{w}) + \alpha (u\bar{w} + uvw + \bar{u}\bar{v}\bar{w}),$$

where  $\alpha = \min(\alpha_{uv}, \alpha_{\bar{v}\bar{w}})$ , will be called as the  $\alpha$ -arithmetic consensus of terms  $\alpha_{uv}uv$  and  $\alpha_{\bar{v}\bar{w}}\bar{v}\bar{w}$  in  $\theta$ .

Let us note that the resulting cubic posiform  $\theta [uv, \bar{v}\bar{w}]$  represents the same quadratic function of posiform  $\theta$ . It should be noted that the transformed posiform may have positive terms involving the quadratic terms  $uw$  or  $\bar{u}\bar{w}$ . In this case we find the standard posiform (see Figure 4.2) of the transformed posiform and denote it as  $\theta'$ . Clearly,  $\theta'$  contains a linear term which is the result of adding the terms  $uw$  or  $\bar{u}\bar{w}$  and  $u\bar{w}$ .

**Example 8.9.** *The quadratic pseudo-Boolean function  $g$  previously used in Examples 8.2 and 8.7 can be represented by the pure quadratic posiform (8.8).*

*$2\bar{x}_1x_3 + 2\bar{x}_3\bar{x}_5$  is a feasible consensus of  $\phi_g$ . We apply a 2-arithmetic consensus transformation of  $\phi_g$  using these two terms to get*

$$\begin{aligned}
\phi_g &= -\frac{13}{2} + 2(\bar{x}_1\bar{x}_5 + \bar{x}_1x_3x_5 + x_1\bar{x}_3\bar{x}_5) \\
&\quad + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_6 + 2x_4\bar{x}_5 \\
&\quad + (x_1x_4 + \bar{x}_1\bar{x}_4) + (x_1\bar{x}_5 + \bar{x}_1x_5) \\
&\quad + \frac{1}{2}(x_2\bar{x}_3 + \bar{x}_2x_3) + \frac{1}{2}(x_2x_4 + \bar{x}_2\bar{x}_4) \\
&\quad + \frac{1}{2}(x_2x_5 + \bar{x}_2\bar{x}_5) + \frac{1}{2}(x_2\bar{x}_6 + \bar{x}_2x_6) \\
&\quad + (x_3\bar{x}_4 + \bar{x}_3x_4) + \frac{1}{2}(x_3\bar{x}_6 + \bar{x}_3x_6) \\
&\quad + \frac{1}{2}(x_4x_6 + \bar{x}_4\bar{x}_6) + \left(\frac{3}{2}\bar{x}_5x_6 + \frac{1}{2}x_5\bar{x}_6\right) \\
&= -\frac{13}{2} + \bar{x}_1 + \bar{x}_5 + 2(\bar{x}_1x_3x_5 + x_1\bar{x}_3\bar{x}_5) \\
&\quad + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_6 + 2x_4\bar{x}_5 \\
&\quad + (x_1x_4 + \bar{x}_1\bar{x}_4) \\
&\quad + \frac{1}{2}(x_2\bar{x}_3 + \bar{x}_2x_3) + \frac{1}{2}(x_2x_4 + \bar{x}_2\bar{x}_4) \\
&\quad + \frac{1}{2}(x_2x_5 + \bar{x}_2\bar{x}_5) + \frac{1}{2}(x_2\bar{x}_6 + \bar{x}_2x_6) \\
&\quad + (x_3\bar{x}_4 + \bar{x}_3x_4) + \frac{1}{2}(x_3\bar{x}_6 + \bar{x}_3x_6) \\
&\quad + \frac{1}{2}(x_4x_6 + \bar{x}_4\bar{x}_6) + \left(\frac{3}{2}\bar{x}_5x_6 + \frac{1}{2}x_5\bar{x}_6\right) \\
&= -\frac{11}{2} + 2(\bar{x}_1x_3x_5 + x_1\bar{x}_3\bar{x}_5) \\
&\quad + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_6 + 2x_4\bar{x}_5 \\
&\quad + 2\bar{x}_1\bar{x}_4 \\
&\quad + \frac{1}{2}(x_2\bar{x}_3 + \bar{x}_2x_3) + x_2x_4 \\
&\quad + \bar{x}_2\bar{x}_5 + \frac{1}{2}(x_2\bar{x}_6 + \bar{x}_2x_6) \\
&\quad + \left(\frac{1}{2}x_3\bar{x}_4 + \frac{3}{2}\bar{x}_3x_4\right) + x_3\bar{x}_6 \\
&\quad + \frac{1}{2}(x_4x_6 + \bar{x}_4\bar{x}_6) + 2\bar{x}_5x_6
\end{aligned}$$

*The last equality follows since*

$$\begin{aligned}
&\bar{x}_1 + x_1x_4 + \frac{1}{2}(\bar{x}_2\bar{x}_4 + x_2x_5) + \frac{1}{2}(x_3\bar{x}_4 + \bar{x}_3x_6 + x_5\bar{x}_6) + \bar{x}_5 \\
&= 1 + \bar{x}_1\bar{x}_4 + \frac{1}{2}(x_2x_4 + \bar{x}_2\bar{x}_5) + \frac{1}{2}(\bar{x}_3x_4 + x_3\bar{x}_6 + \bar{x}_5x_6).
\end{aligned}$$

Let us denote by  $q(\psi)$  to the posiform consisting of terms from  $\psi$  of degree not larger than 2.

Given a quadratic posiform representing a quadratic pseudo-Boolean function  $f$ , let us consider the following algorithmic approach:

1. Find a roof-dual posiform of  $f$ , remove any strong and weak persistencies, and call to the resulting pure quadratic posiform as  $\psi$ .
2. If  $q(\psi)$  is gap-free (i.e.  $\min_{\mathbf{x} \in \mathbb{B}^n} q(\psi) = C(\psi)$ ) then STOP and return  $C(\psi)$  as a bound to the minimum of  $f$ .
3. Find 2 terms in  $\psi$  with a feasible consensus of weight  $\alpha$  and apply an  $\alpha$ -arithmetic consensus operation to it. Let us call to the resulting standard posiform  $\psi$ .
4. If there is a linear term in  $\psi$  then the roof-dual posiform of  $q(\psi)$  is found, any resulting strong and weak persistencies are removed, and the resulting posiform is called  $\psi$ . Go to step 2.
5. Go to step 3.

The above iterative method is clearly a finite procedure since at every iteration it either (i) improves the lower bound of the function or (ii) it reduces the number of nontrivial quadratic terms by one. This type of algorithm has been originally proposed by Bourjolly et al. [65, 66, 67, 68].

It is also clear that at the end of the method,  $\psi$  is cubic posiform with all cubic terms being mated (see Definition 8.10). The end result of this approach is therefore equivalent to the iterated roof-duality approach (see Proposition 8.7), since in both cases the original posiform is transformed into a cubic posiform with this characteristic. Also in this case we have no knowledge about any reference that considered improving further the bound by reusing the residual cubic terms. Obviously, the approach that has been already proposed for the “squeezed” iterated roof-duality algorithm can be adopted here as well (see Section 8.5).

Before explaining our final implementation of the algorithm we first investigate a related new topic that we call “project-and-lift”.

It is very well known that any quadratic pseudo-Boolean function can be expressed as

$$f(\mathbf{x}) = x_k f(\mathbf{x}[\{k\} \leftarrow \{1\}]) + \bar{x}_k f(\mathbf{x}[\{k\} \leftarrow \{0\}]) \quad (8.11)$$

for all binary vector  $\mathbf{x} \in \mathbb{B}^n$  and  $k = 1, \dots, n$ .

**Lemma 8.7.** *Let  $f$  be a pseudo-Boolean function. Then*

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{B}^V} f(\mathbf{x}) &= \min_{\mathbf{x} \in \mathbb{B}^V} (x_k f(\mathbf{x}[\{k\} \leftarrow \{1\}]) + \bar{x}_k f(\mathbf{x}[\{k\} \leftarrow \{0\}])), \\ &= \min_{\mathbf{x} \in \mathbb{B}^{V \setminus \{k\}}} \min (f(\mathbf{x}[\{k\} \leftarrow \{1\}]), f(\mathbf{x}[\{k\} \leftarrow \{0\}])), \\ &= \min_{\mathbf{x} \in \mathbb{B}^{V \setminus \{k\}}} \min_{\lambda \in \mathbb{U}} (\lambda f(\mathbf{x}[\{k\} \leftarrow \{1\}]) + (1 - \lambda) f(\mathbf{x}[\{k\} \leftarrow \{0\}])), \\ &= \min \left( \min_{\mathbf{x} \in \mathbb{B}^{V \setminus \{k\}}} f(\mathbf{x}[\{k\} \leftarrow \{1\}]), \min_{\mathbf{x} \in \mathbb{B}^{V \setminus \{k\}}} f(\mathbf{x}[\{k\} \leftarrow \{0\}]) \right), \end{aligned}$$

for all  $k = 1, \dots, n$ .

*Proof.* Follow directly from (8.11). □

The last equality of Lemma 8.7 is particularly interesting since the end result does not depend on  $x_k$ . This suggests that the function  $f$  can be *projected* into two lower dimension pseudo-Boolean functions having  $n - 1$  variables each. Clearly, the minimum of the two bounds associated to these lower dimension functions is a bound to  $f$ .

In what follows, we consider projections over posiforms (or equivalent over capacitated networks). Let  $\phi_f$  be a quadratic posiform representing  $f$ . For simplicity, we denote  $\phi_{f|x_k=b}$  to a posiform representing  $f(\mathbf{x}[\{k\} \leftarrow \{b\}])$ , which is obtained from  $\phi_f$  by fixing  $x_k = b$ , for  $b = 0, 1$ .

Without loss of generality, it is assumed that there are no strong or weak persistencies in  $f$ , since in this case the QUBO optimization problem can be simplified by fixing those optimal values, and then the method described below would apply as well. Further let us assume that  $\phi_f$  is a standard pure quadratic posiform. This representation can be obtained from the network model (see Section 5.3).

Let us also consider the roof-dual quadratic posiforms

$$C(\phi_{f|x_k=b}) + v(\varphi_{f|x_k=b}) + \psi_{f|x_k=b}$$

found by computing a maximum flow  $\varphi_{f|x_k=b}$  in the network model (see Proposition 5.8) associated to  $\phi_{f|x_k=b}$  ( $b = 0, 1$ ).

Clearly,

$$\begin{aligned} & \min_{\mathbf{x} \in \mathbb{B}^V} \phi_f \\ &= \min(\phi_{f|x_k=1}, \phi_{f|x_k=0}) \\ &= \min \left( \begin{array}{l} C(\phi_{f|x_k=1}) + v(\varphi_{f|x_k=1}) + \psi_{f|x_k=1}, \\ C(\phi_{f|x_k=0}) + v(\varphi_{f|x_k=0}) + \psi_{f|x_k=0}. \end{array} \right). \end{aligned}$$

Therefore,

**Corollary 8.4.**

$$\min_{\mathbf{x} \in \mathbb{B}^V} \phi_f \geq \max_{k \in V} \min(C(\phi_{f|x_k=1}) + v(\varphi_{f|x_k=1}), C(\phi_{f|x_k=0}) + v(\varphi_{f|x_k=0})).$$

What has been discussed so far is somewhat trivial and well known. What makes the decomposition above interesting is the *common* structure of the lower dimension residuals  $\psi_{f|x_k=v}$  ( $v = 0, 1$ ). This is possible since these posiforms were created by using the same starting initial posiform  $\phi_f$ , and hence the (unique) corresponding capacitated networks have the same arcs and capacities between all pairs of literals  $(u, v)$  such that  $u, v \neq x_k$  and  $u, v \neq \bar{x}_k$ . Furthermore, the residual networks obtained after finding the maximum flow on those networks either both have the same  $(u, v)$  arcs or one network has a  $(u, v)$  arc and the other one has a reverse arc  $(v, u)$ . In conclusion, if  $\alpha u \bar{v}$  is a term of  $\phi_f$  then there are nonnegative constants  $c_b$  ( $b = 0, 1$ ) such that  $[(\alpha - c_1) u \bar{v} + c_1 \bar{u} v] \in \varphi_{f|x_k=1}$  and  $[(\alpha - c_0) u \bar{v} + c_0 \bar{u} v] \in \varphi_{f|x_k=0}$ . Without losing generality, let us assume that  $c_1 \geq c_0$ . Then

$$\begin{aligned} & [(\alpha - c_1) u \bar{v} + c_1 \bar{u} v] x_k + [(\alpha - c_0) u \bar{v} + c_0 \bar{u} v] \bar{x}_k \\ &= (\alpha - c_1) u \bar{v} + c_0 \bar{u} v + (a - c_0 - (a - c_1)) u \bar{v} \bar{x}_k + (c_1 - c_0) \bar{u} v x_k \\ &= (\alpha - c_1) u \bar{v} + c_0 \bar{u} v + (c_1 - c_0) (u \bar{v} \bar{x}_k + \bar{u} v x_k). \end{aligned}$$

The above remark implies the following lemma.

**Lemma 8.8.** *The lower dimension posiforms  $\psi_{f|x_k=1}$  and  $\psi_{f|x_k=0}$  have nonzero quadratic*

terms that

(i) are either precisely the same; or

(ii) if one posiform has the term  $\beta\bar{u}v$  then the other one has a term  $\beta u\bar{v}$ .

Lemma 8.8 provides a way to *lift* the lower dimension posiforms to the original function by means of a cubic posiform

$$\Psi_f = x_k [C(\phi_{f|x_k=1}) + v(\varphi_{f|x_k=1}) + \psi_{f|x_k=1}] + \bar{x}_k [C(\phi_{f|x_k=0}) + v(\varphi_{f|x_k=0}) + \psi_{f|x_k=0}].$$

whose cubic terms are all mated.

The *Project-and-Lift* (P&L for short) idea described above is clearly related to the arithmetic consensus algorithm described earlier in this section. The important achievement of P&L is to be able to get all consensus related to a variable by computing two maximum flows. One can imagine that it is possible to develop an iterative procedure that applies P&L to all the variables until there are no quadratic or linear terms left in  $\Psi_f$ .

**Example 8.10.** *Let us consider again the quadratic pseudo-Boolean function  $g$  of Example 8.9. Let us project  $\phi_g$  down using variable  $x_3$ . Then,*

$$\begin{aligned} \phi_{g|x_3=1} &= -5 + \frac{1}{2}\bar{x}_1 + \frac{1}{2}\bar{x}_6 + \frac{1}{2}x_1\bar{x}_5 + \bar{x}_1\bar{x}_2 + 2\bar{x}_1\bar{x}_4 + \frac{3}{2}\bar{x}_1x_5 + \bar{x}_1\bar{x}_6 \\ &\quad + \frac{1}{2}x_2x_4 + \frac{1}{2}x_2\bar{x}_6 + \frac{1}{2}\bar{x}_2\bar{x}_4 + \bar{x}_2\bar{x}_5 + \frac{1}{2}\bar{x}_2x_6 \\ &\quad + 2x_4\bar{x}_5 + \frac{1}{2}x_4x_6 + \frac{1}{2}\bar{x}_4\bar{x}_6 + \frac{1}{2}x_5\bar{x}_6 + \frac{3}{2}\bar{x}_5x_6, \text{ and} \end{aligned}$$

$$\begin{aligned} \phi_{g|x_3=0} &= -5 + \bar{x}_5 + x_1x_4 + x_1\bar{x}_5 + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_4 + \bar{x}_1x_5 + \bar{x}_1\bar{x}_6 \\ &\quad + x_2x_4 + x_2\bar{x}_6 + \bar{x}_2\bar{x}_5 + 2x_4\bar{x}_5 + x_4x_6 + 2\bar{x}_5x_6. \end{aligned}$$

Lifting back  $x_3$  to the original function we get

$$\begin{aligned}
\phi_g &= x_3 \phi_{g|_{x_3=1}} + \bar{x}_3 \phi_{g|_{x_3=0}} \\
&= x_3 \left( \begin{aligned} &-5 + \frac{1}{2}\bar{x}_1 + \frac{1}{2}\bar{x}_6 + \frac{1}{2}x_1\bar{x}_5 + \bar{x}_1\bar{x}_2 + 2\bar{x}_1\bar{x}_4 + \frac{3}{2}\bar{x}_1x_5 + \bar{x}_1\bar{x}_6 + \frac{1}{2}x_2x_4 \\ &+ \frac{1}{2}x_2\bar{x}_6 + \frac{1}{2}\bar{x}_2\bar{x}_4 + \bar{x}_2\bar{x}_5 + \frac{1}{2}\bar{x}_2x_6 + 2x_4\bar{x}_5 + \frac{1}{2}x_4x_6 + \frac{1}{2}\bar{x}_4\bar{x}_6 + \frac{1}{2}x_5\bar{x}_6 + \frac{3}{2}\bar{x}_5x_6 \end{aligned} \right) \\
&\quad + \bar{x}_3 \left( \begin{aligned} &-5 + \bar{x}_5 + x_1x_4 + x_1\bar{x}_5 + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_4 + \bar{x}_1x_5 + \bar{x}_1\bar{x}_6 \\ &+ x_2x_4 + x_2\bar{x}_6 + \bar{x}_2\bar{x}_5 + 2x_4\bar{x}_5 + x_4x_6 + 2\bar{x}_5x_6 \end{aligned} \right) \\
&= -5 + \frac{1}{2}\bar{x}_1x_3 + \frac{1}{2}x_3\bar{x}_6 + \bar{x}_3\bar{x}_5 \\
&\quad + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_4 + \bar{x}_1x_5 + \frac{1}{2}x_1\bar{x}_5 + \bar{x}_1\bar{x}_6 + \frac{1}{2}x_2x_4 \\
&\quad + \bar{x}_2\bar{x}_5 + \frac{1}{2}x_2\bar{x}_6 + 2x_4\bar{x}_5 + \frac{1}{2}x_4x_6 + \frac{3}{2}\bar{x}_5x_6 \\
&\quad + \frac{1}{2}(\bar{x}_1x_3x_5 + x_1\bar{x}_3\bar{x}_5) + (\bar{x}_1x_3\bar{x}_4 + x_1\bar{x}_3x_4) \\
&\quad + \frac{1}{2}(\bar{x}_2x_3\bar{x}_4 + x_2\bar{x}_3x_4) + \frac{1}{2}(x_2x_3\bar{x}_6 + \bar{x}_2\bar{x}_3x_6) \\
&\quad + \frac{1}{2}(x_3\bar{x}_4\bar{x}_6 + \bar{x}_3x_4x_6) + \frac{1}{2}(x_3x_5\bar{x}_6 + \bar{x}_3\bar{x}_5x_6)
\end{aligned} \tag{8.12}$$

From the above results then we can conclude that  $\mathbf{x} \in \mathbb{B}^n g(\mathbf{x}) \geq -5$ . In fact this case it is known that the minimum of function  $g$  coincides with the bound and thus the above posiform is gap-free. Generally, the problem of certifying that the above cubic posiform is gap-free is NP-complete.

A relatively unexplored area is about solving QUBO by considering *surrogate* functions whose minimum (or maximum) is the same as that one of the function being optimized. For bi-forms we have already introduced this concept in Corollary 8.1 (see also Example 8.2). A similar phenomenon can be found within the P&L mechanism. The trick is to detect and fix strong and weak persistencies in the lower dimension posiforms of  $x_k$ . This is possible since both lower dimension posiforms are independent of  $x_k$  (see Lemma 8.7). The following example illustrates this possibility.

**Example 8.11.** *Let us consider function  $g$  considered previously. In Example 8.10 we have already computed the roof-duals of the lower dimension posiforms:  $\phi_{g|_{x_3=1}}$  has two linear terms  $\frac{1}{2}\bar{x}_1$  and  $\frac{1}{2}\bar{x}_6$  and  $\phi_{g|_{x_3=0}}$  has one linear term  $\bar{x}_5$ . By strong persistency all minimizers of function  $g(\mathbf{x}[\{k\} \leftarrow \{1\}])$  must have the assignment  $x_1 = x_6 = 1$ , whereas all minimizers of function  $g(\mathbf{x}[\{k\} \leftarrow \{0\}])$  must have the assignment  $x_5 = 1$ .*

Thus, the posiform

$$\phi_g - \left( \frac{1}{2} \bar{x}_1 x_3 + \frac{1}{2} x_3 \bar{x}_6 + \bar{x}_3 \bar{x}_5 \right)$$

has the same minimum value of function  $g$ , but represents a different quadratic pseudo-Boolean function. Looking further into the persistency results it can be seen that

- $x_1 = \bar{x}_4 = x_5 = 1$  is strongly persistent for  $\phi_{g|x_3=0}$ ;
- $x_1 = \bar{x}_4 = x_5 = x_2 = x_6 = 1$  is strongly persistent for  $\phi_{g|x_3=1}$ .

Substituting these assignments in (8.12) then we can easily conclude that the minimum of  $g$  is -5.

As in the previous sections, the big challenge that we have at hand is to squeeze additional quadratic terms from the current cubic expression of  $\Psi_f$ . We considered here the same approach used previously for the squeezed iterated roof-duality bound presented in Section 8.5.

We have implemented a P&L iterated roof-duality algorithm, whose pseudo-code can be seen in Figure 8.5.

In the next section we will see that the bounds obtained by this new algorithm are superior in practice (at the end of the same round number) to those returned by SIRDA. The difference lies in the fact that the residual quadratic problem is a bi-form in SIRDA, whereas it is a generic posiform for the P&L algorithm version.

Improved data structures that (i) can efficiently handle both projections and lifted mated cubic posiforms, and that (ii) can perform rotation transformations efficiently, are important computational research topics. In this work, the focus of our implementation was mostly to show the value of the method (using the tools that we already had at hand) and to provide a working basis for future comparative studies.

### 8.6.1 Computational results

The PROJECT&LIFT-ITERATED-ROOF-DUAL algorithm (called PLIRDA hereafter) has been implemented in C++.

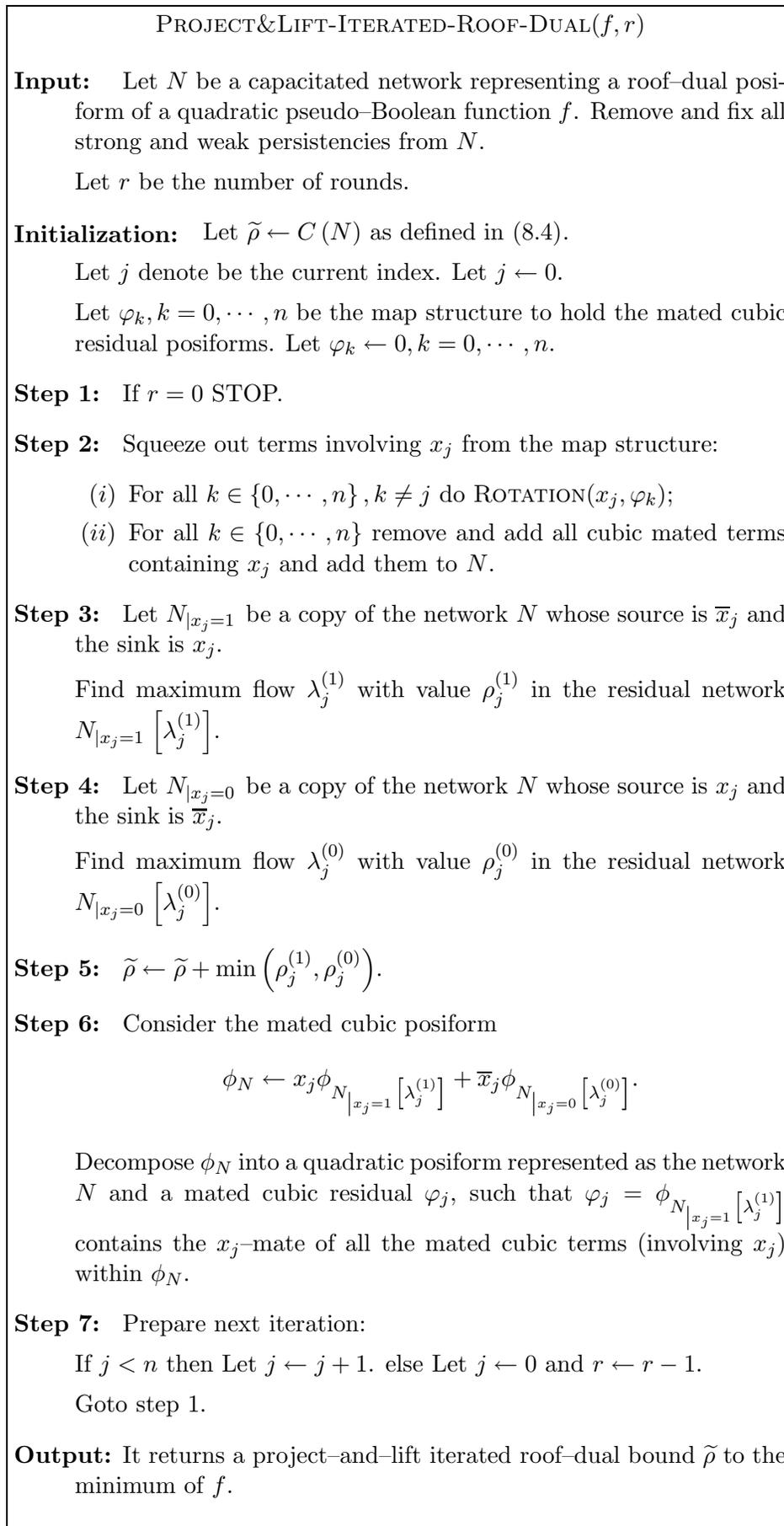


Figure 8.5: PROJECT&amp;LIFT-ITERATED-ROOF-DUAL algorithm.

Comparative results between SIRDA and PLIRDA are presented in this section. Table 8.7(a) indicates how far are these two algorithms from the cubic dual ( $C_3$ ), for the 10 QUBO problems created by Glover et al. [108]. The P&L iterated roof-dual bound is significantly closer to  $C_3$  than the squeezed version of the algorithm, for any number of rounds ( $r = 1, 2, 3$ ) considered. The proximity to  $C_3$  from the P&L algorithm is also more noticeable if the number of rounds  $r$  increases or if the density parameter  $d$  is either high or low.

On an average case, SIRDA is 4.3 (respectively 2.9 and 2.5) times faster than PLIRDA if one (respectively two and three) round is considered (see Table 8.7(b)).

In this section we investigate the quality of the P&L iterated roof dual bound in a group of (non-weighted) MAX-2-SAT formulas generated by Bonami and Minoux [46], who have proposed a new bound to solve these problems. This bound is based on the classical Lift-and-Project (L&P) cut generation of a standard mixed integer programming formulation of MAX-2-SAT. We have found recently that the L&P bounds for these problems nearly coincide with  $C_3$ .

As before we could see that P&L iterated roof-dual bounds are clearly superior to the squeezed ones. It also shows that the P&L bound is substantially better than the standard iterated roof-dual bound, and that for these datasets it is between 5% and 8% away from the L&P (or in this case  $C_3$ ) bound.

For the tested instances, the computing times of PLIRDA seem to grow quadratically with the number of variables. For instance when three rounds were considered, PLIRDA spent around 3 seconds for the 75 variable instances and spent 32 seconds for the 200 variable cases.

## 8.7 Linearization models for sparse QUBOs

Robust high performance implementations of the state-of-the-art LP solvers are up to 100 times faster than implementations of ten years ago on the same hardware ([21]). This fact used together with the hundred-fold or so increase in computer speed and memory capacity, implies that very large LPs can be solved optimally in a standard

Table 8.7: Project-and-lift and squeezed iterated roof-duals of QUBO problems with 100 variables (Glover et al. [108]).

(a) Relative gap to the  $C_3$  bound.

Problem Name	Density (d)	Relative gap to the $C_3$ bound (resp. $\frac{\hat{p}-C_3}{C_3}$ and $\frac{\tilde{p}-C_3}{C_3}$ ).					
		SIRDA ( $\bar{p}$ )	PLIRDA ( $\tilde{p}$ )	SIRDA ( $\bar{p}$ )	PLIRDA ( $\tilde{p}$ )	SIRDA ( $\bar{p}$ )	PLIRDA ( $\tilde{p}$ )
		(r = 1)		(r = 2)		(r = 3)	
1d	10%	0.2%	1.0%	0.1%	0.4%	0.1%	0.3%
2d	20%	9.8%	9.3%	7.2%	6.8%	6.4%	5.8%
3d	30%	10.6%	11.4%	8.4%	7.7%	7.3%	6.5%
4d	40%	14.1%	14.5%	10.4%	9.5%	9.2%	7.8%
5d	50%	14.6%	13.9%	11.1%	9.1%	10.2%	7.2%
6d	60%	13.4%	11.9%	9.9%	7.7%	8.6%	6.1%
7d	70%	9.3%	8.3%	5.3%	4.2%	4.2%	2.7%
8d	80%	8.7%	7.2%	4.9%	3.2%	3.9%	1.9%
9d	90%	7.8%	6.6%	4.0%	2.5%	3.1%	1.3%
10d	100%	7.3%	4.7%	3.4%	1.5%	2.6%	0.6%

(b) Computing time speed analysis between SIRDA and PLIRDA.

Problem	$\frac{\text{time of PLIRDA}}{\text{time of SIRDA}}$		
	(r = 1)	(r = 2)	(r = 3)
1d	2.5	3.8	4.6
2d	3.1	2.2	2.0
3d	3.9	2.7	2.3
4d	4.9	2.9	2.3
5d	4.9	2.8	2.2
6d	5.0	2.8	2.3
7d	5.1	2.9	2.4
8d	4.8	3.0	2.3
9d	4.9	2.8	2.2
10d	4.4	3.0	2.3

Table 8.8: Project-and-lift and squeezed iterated roof-duals of MAX-2-SAT (Bonami and Minoux [46]).

(a) Lower bounds.

<i>Average Lower Bounds to the Minimum</i>											
<i>Variables</i> ( <i>n</i> )	<i>Clauses</i> ( <i>m</i> )	Average Minimum	<i>Roof</i> <i>Dual</i> ( $\rho$ )	<i>Iter. Roof</i> <i>Dual</i> ( $\hat{\rho}$ )	<i>Squeezed Iterated Roof-Dual</i>			<i>Project-and-Lift Iterated Roof-Dual</i>			<i>Lift &amp; Project</i> ( <i>L&amp;P</i> from [46])
					$(\bar{\rho}; r = 1)$	$(\bar{\rho}; r = 2)$	$(\bar{\rho}; r = 3)$	$(\bar{\rho}; r = 1)$	$(\bar{\rho}; r = 2)$	$(\bar{\rho}; r = 3)$	
75	525	59.6	11.0	45.7	50.8	53.1	53.8	52.0	53.9	54.7	57.9
75	550	62.6	11.7	48.1	54.6	56.9	57.6	55.5	57.4	58.1	61.3
75	600	73.0	15.4	57.2	63.1	65.4	66.1	64.0	66.4	67.1	70.8
100	700	80.2	11.6	59.2	66.7	69.9	70.7	68.3	71.1	72.0	76.4
150	850	80.8	7.8	55.9	65.7	68.6	69.3	67.1	69.8	70.8	76.6
200	1000	89.8	6.0	57.5	67.8	71.2	71.8	71.2	74.2	75.1	81.7

(b) Computing times.

<i>Average Computing Time*</i>									
<i>Variables</i>	<i>Clauses</i>	<i>RDA</i>	<i>IRDA</i>	<i>SIRDA</i>			<i>PLIRDA</i>		
				( <i>r</i> = 1)	( <i>r</i> = 2)	( <i>r</i> = 3)	( <i>r</i> = 1)	( <i>r</i> = 2)	( <i>r</i> = 3)
75	525	<0.005 s	0.01 s	0.2 s	0.7 s	1.5 s	0.8 s	1.7 s	2.8 s
75	550	<0.005 s	0.02 s	0.3 s	0.9 s	1.6 s	0.8 s	1.8 s	2.8 s
75	600	<0.005 s	0.02 s	0.3 s	0.9 s	1.6 s	0.8 s	1.8 s	3.0 s
100	700	<0.005 s	0.02 s	0.5 s	1.6 s	3.3 s	1.4 s	3.4 s	5.6 s
150	850	<0.005 s	0.02 s	1.2 s	4.3 s	8.3 s	3.4 s	8.8 s	15.6 s
200	1000	<0.005 s	0.04 s	2.2 s	7.7 s	15.5 s	6.4 s	17.5 s	32.0 s

\*Computed on computer system I (see Table 8.1).

nowadays computer. There are several examples of LPs solved to optimality, involving tens of millions of nonzero elements, hundreds of thousands of constraints, and few millions of variables.

In Chapter 9 we shall demonstrate that LP in combination with branch-and-bound is an effective method to solve relatively sparse problems of large dimensions. The concept of sparsity that we use here is related to problems of low density and in particular to problems whose variables are typically not involved in too many (nonzero) quadratic terms. In this section, we study various families of well defined cuts and show that they can be characterized by a relatively small number of cuts for those sparse QUBO problems.

We start by providing a simple result that can potentially reduce considerably the size of certain (CUBIC DUAL) LPs.

Let

$$\mathbf{D}^{[2]} = \left\{ (\mathbf{x}, \mathbf{y}) \left| \begin{array}{ll} -y_{ij} \leq 0, & c_{ij} > 0, \\ -x_i + y_{ij} \leq 0, & c_{ij} < 0, \\ -x_j + y_{ij} \leq 0, & c_{ij} < 0, \\ x_i + x_j - y_{ij} \leq 1, & c_{ij} > 0, \end{array} \right. \right. \left. \left. (1 \leq i < j \leq n) \right\}.$$

Clearly,  $\mathbf{D}^{[2]} \subseteq \mathbf{S}^{[2]}$ .

**Lemma 8.9** ([54]).

$$C_2(f) = \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{D}^{[2]}, \mathbf{x} \in \mathbb{U}^n \right\}.$$

The above lemma raises the question about the possibility of ignoring certain sets of constraints, especially from  $\mathbf{S}^{[3]}$ , depending on the quadratic coefficients of the function being zero or not. If this is true then the size of the LP problems can be reduced substantially, and in particular if the problem is sparse. The following result gives a positive indication into this direction.

**Theorem 8.2.** Let  $f$  be quadratic pseudo-Boolean function given as (1.5) and

$$\mathbf{D}^{[3]} = \mathbf{S}^{[2]} \cup \left\{ (\mathbf{x}, \mathbf{y}) \left| \begin{array}{ll} x_i + x_j + x_k - y_{i,j} - y_{i,k} - y_{j,k} \leq 1, & \left( 1 \leq i < j < k \leq n \right) \\ -x_i & +y_{i,j} + y_{i,k} - y_{j,k} \leq 0, & c_{ij} \neq 0 \text{ or} \\ -x_j & +y_{i,j} - y_{i,k} + y_{j,k} \leq 0, & c_{ik} \neq 0 \text{ or} \\ -x_k & -y_{i,j} + y_{i,k} + y_{j,k} \leq 0, & c_{jk} \neq 0 \end{array} \right. \right\}.$$

Then,

$$C_3(f) = \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{D}^{[3]}, \mathbf{x} \in \mathbb{U}^n \right\}.$$

*Proof.* The result follows from the fact that  $C_3 = \omega(\mathbf{TP})$  (see Proposition 8.5).  $(\mathbf{TP})$  is a triangle packing problem that considers all negative triangles for packing. Linear combinations of negative triangles are feasible solutions within  $(\mathbf{TP})$ . Since every negative triangle is linked to at least one edge of  $G_f$  (the balancing graph of  $f$ ) then the result follows readily.  $\square$

We shall consider next other LP models that further reduce the size of the number of triangle cuts to be added to the roof dual LP formulation. The bound obtained in this way is between the roof-dual and the cubic-dual bounds.

First we consider

$$\mathbf{W}^{[2]} = \left\{ (\mathbf{x}, \mathbf{y}) \left| \begin{array}{ll} & -y_{ij} \leq 0, \\ -x_i & +y_{ij} \leq 0, & \left( 1 \leq i < j \leq n \right) \\ -x_j & +y_{ij} \leq 0, & c_{ij} \neq 0 \\ x_i + x_j & -y_{ij} \leq 1, \end{array} \right. \right\}$$

and

$$\mathbf{W}^{[3]}(\mathcal{S}) = \mathbf{W}^{[2]} \cup \left\{ (\mathbf{x}, \mathbf{y}) \left| \begin{array}{ll} x_i + x_j + x_k - y_{i,j} - y_{i,k} - y_{j,k} \leq 1, & \left( 1 \leq i < j < k \leq n \right) \\ -x_i & +y_{i,j} + y_{i,k} - y_{j,k} \leq 0, & (i, j, k) \in \mathcal{S} \\ -x_j & +y_{i,j} - y_{i,k} + y_{j,k} \leq 0, \\ -x_k & -y_{i,j} + y_{i,k} + y_{j,k} \leq 0, \end{array} \right. \right\}. \quad (8.13)$$

$\mathcal{S}$  in (8.13) represents the set of triplets  $(i, j, k)$  corresponding to the triangle inequalities involving variables  $x_i, x_j$  and  $x_k$ . In this study we consider three cases:

- $\mathcal{S}_0 = \{(i, j, k) \in V^3 \mid c_{ij}c_{ik}c_{jk} \neq 0\}$ ;
- $\mathcal{S}_1 = \{(i, j, k) \in V^3 \mid c_{ij} \neq 0 \text{ and } (c_{ik} \neq 0 \text{ or } c_{jk} \neq 0)\}$ ;
- $\mathcal{S}_2 = \{(i, j, k) \in V^3 \mid c_{ij} \neq 0\}$ .

Let us define

$$\kappa(f, \mathcal{S}) \stackrel{\text{def}}{=} \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{W}^{[3]}(\mathcal{S}), \mathbf{x} \in \mathbb{U}^n \right\}.$$

**Lemma 8.10.** *Let  $f$  be quadratic pseudo-Boolean function given as (1.5). Then,*

$$C_2(f) = \kappa(f, \emptyset) \leq \kappa(f, \mathcal{S}_0) \leq \kappa(f, \mathcal{S}_1) \leq \kappa(f, \mathcal{S}_2) \leq C_3(f).$$

*Proof.* The result follows trivially from the fact that  $\mathcal{S}_0 \subseteq \mathcal{S}_1 \subseteq \mathcal{S}_2$ . □

All the computational experiments that we have carried out suggest the following strong claim that we were unable to demonstrate.

**Conjecture 8.2.**

$$C_3(f) = \kappa(f, \mathcal{S}_2).$$

The above result is in accordance to the squeezing scheme presented previously in this chapter. We believe that there is a maximum possible iterated roof-dual improvement per variable, and that in subsequent iterations it is not necessary to squeeze further terms from variables whose terms were already squeezed out. Also we think that the variables sequencing order for the iterated procedure does not matter and that any sequence will return  $C_3$  after at most  $n$  iterations.

### 8.7.1 Computational results

The conclusions of the following computational experiments will clearly demonstrate that LP can “quickly” provide good results for many sparse QUBOs, which are typical

in many real problems.

The LP solver that we used is Xpress-Barrier from the 2007B release of Xpress-MP. In our tests we also consider the time of finding an optimal basic feasible solution by using the Xpress crossover algorithm. This step is important when the studied linear program bounds are used during the B&B search (see Chapter 9). Additionally, the finding of such basis allows us to determine what cuts are binding at the LP extreme point solution. This information can be used to reduce the size of the problem (by removing non-binding cuts) while keeping the same bound. Typically, the problem reduction makes the MIP solver to run faster and saves in memory consumption.

We start by analyzing the bounds in the test set of Bonami and Minoux [46] about MAX-2-SAT. It should be remarked that we considered the standard linearization model in what follows, but it should be mentioned that MAX-2-SAT can be solved by a more compacted linear program, which does not required additional variables (see [46]).

The results carried out on these problems are given on Tables 8.9(a) and 8.9(b), and cover the iterated roof duality, its P&L version, the L&P bound proposed in [46], and the linear programming bounds  $\kappa(\mathcal{S}_0)$ ,  $\kappa(\mathcal{S}_1)$  and  $\kappa(\mathcal{S}_2)$ .

Table 8.9(a) includes the average relative gap of the studied lower bounds to the minimum possible number of unsatisfied clauses. First, L&P and  $\kappa(\mathcal{S}_2)$  are remarkably close to each other. Second, these two bounds are clearly superior to the iterated roof-dual versions.

The computing times seem to favor the P&L iterated roof dual bound, especially when the number of variables increases. This result gives some indication to the fact that as the size of the problem increases the combinatorial approaches may have a better chance to succeed in solving them. We also remark that the solve times of PLIRDA can be somewhat improved simply by adopting better algorithm data structures and implementations.

Another interesting point to discuss is the fact that linear programming may take longer to compute the bound for those larger instances, but when the non-binding cuts are removed from the formulation, then this approach becomes attractive to be solved by the current MIP technology (see Chapter 9).

Table 8.9: Linear programming bounds of MAX-2-SAT (Bonami and Minoux [46]).

(a) Average relative gap to the minimum number of unsatisfied clauses.

		<i>Average Relative Gap of some Lower Bounds to the Minimum (<math>\nu</math>)</i>								
<i>Variables</i> ( $n$ )	<i>Clauses</i> ( $m$ )	<i>Iter. Roof</i> <i>Dual</i> ( $\frac{\nu-\hat{\rho}}{\nu}$ )	<i>Project-and-Lift Iterated Roof-Dual</i>			<i>Lift &amp; Project</i> ( <i>L&amp;P from</i> [46])	<i>Linear Programming</i>			
			( $\frac{\nu-\tilde{\rho}}{\nu}; r=1$ )	( $\frac{\nu-\tilde{\rho}}{\nu}; r=2$ )	( $\frac{\nu-\tilde{\rho}}{\nu}; r=3$ )		( $\frac{\nu-\kappa(\mathcal{S}_0)}{\nu}$ )	( $\frac{\nu-\kappa(\mathcal{S}_1)}{\nu}$ )	( $\frac{\nu-\kappa(\mathcal{S}_2)}{\nu}$ )	
75	525	23.3%	12.6%	9.4%	8.1%	<b>2.7%</b>	17.1%	3.4%	<b>2.7%</b>	
75	550	23.1%	11.3%	8.2%	7.1%	<b>2.0%</b>	14.6%	2.4%	<b>2.0%</b>	
75	600	21.7%	12.3%	9.1%	8.0%	<b>2.9%</b>	14.1%	3.4%	<b>2.9%</b>	
100	700	26.2%	14.8%	11.3%	10.2%	<b>4.6%</b>	24.6%	5.6%	<b>4.6%</b>	
150	850	30.7%	16.8%	13.6%	12.3%	<b>5.1%</b>	46.3%	10.0%	<b>5.1%</b>	
200	1000	35.9%	20.6%	17.3%	16.2%	8.9%	61.1%	17.2%	<b>8.5%</b>	

(b) Computing times.

		<i>Average Computing Time</i>						
<i>Variables</i>	<i>Clauses</i>	<i>IRDA*</i>	<i>PLIRDA*</i>			<i>XPRESS-Barrier**</i>		
			( $r=1$ )	( $r=2$ )	( $r=3$ )	( $\kappa(\mathcal{S}_0)$ )	( $\kappa(\mathcal{S}_1)$ )	( $\kappa(\mathcal{S}_2)$ )
75	525	0.01 s	0.8 s	1.7 s	2.8 s	0.1 s	1.0 s	5.9 s
75	550	0.02 s	0.8 s	1.8 s	2.8 s	0.2 s	1.3 s	6.6 s
75	600	0.02 s	0.8 s	1.8 s	3.0 s	0.1 s	1.3 s	5.9 s
100	700	0.02 s	1.4 s	3.4 s	5.6 s	0.1 s	1.9 s	13.7 s
150	850	0.02 s	3.4 s	8.8 s	15.6 s	0.1 s	2.8 s	47.0 s
200	1000	0.04 s	6.4 s	17.5 s	32.0 s	0.1 s	3.2 s	102.4 s

\*Computed on computer system I (see Table 8.1).

\*\*Computed on an AMD Athlon 64 X2 Dual Core 4800+, 2.41 GHz, 4GB RAM and runs XP.

The usefulness of the 3 linear programming models proposed in this section is mostly a property of sparse QUBOs. The concept of sparsity used here is different from the density property used along this dissertation. Here we consider that each variable is not involved in more than a constant  $k$  quadratic terms of the multilinear representation of the function. In a private conversation, Endre Boros called these functions as *ultra-sparse*, which have the interesting property that the density approaches zero (i.e.  $d \leftarrow 0$ ) as the number of variables increases (i.e as  $n \leftarrow \text{inf}$ ).

We have already seen that ultra-sparse functions are common in many real world applications (e.g. 2D/3D Ising models, planar vertex covers, via minimization, biomedical imaging).

Next we consider a ultra-sparse family that we have randomly generated for MAX-CUT. We called this family as the Hamilton family (see Section 3.4.2). It consists of weighted graphs with a  $m$  Hamiltonian cycles. Here we analyze the case where  $m$  is 2 and the graphs have 250 or 500 variables. Because  $m = 2$  then every vertex has at most four neighbors for any instance. In this case  $k = 4$  for the above sparsity definition.

Table 8.10(a) shows the average relative gaps to the best known weighted cuts of graphs having 250 and 500 vertices. The bounds  $(\kappa(\mathcal{S}_0))$  and  $(\kappa(\mathcal{S}_1))$  are very close to each other. They are clearly inferior to the bound  $(\kappa(\mathcal{S}_2))$ , whose relative gap varies from 0.0% to 3.5%, whereas the other bounds have gaps varying between 4.5% and 21%.

It is also interesting to see that the relative gap varies substantially for different weight schemes. The easiest group corresponds to graphs with weights  $[-50, 50]$ , for which XPRESS-Barrier with the  $\mathcal{S}_2$ -cuts returns near optimal solutions for the 250 vertices instances, and returns around 1%-to-2% relative gaps for the 500 vertices instances. The hardest group corresponds to the graphs having weights  $[50, 100]$  and negative exterior field.

The computing times are substantially larger for the case that uses the  $\mathcal{S}_2$ -cuts. The longest XPRESS-Barrier run time was 1 182 seconds for the 500 graphs instances having  $[-50, 50]$  weights.

Table 8.10: Upper bounds based on linear programming for MAX-CUT graphs from the Hamilton family.

(a) Average relative gap to the best known cuts with average weight ( $\bar{w}$ ).

<i>Exterior Field</i> ( $h$ )	<i>Edge's Weights</i> ( $[w^-, w^+]$ )	<i>Vertices</i> ( $ V $ )	<i>Linear Programming</i>		
			$\left(\frac{\kappa(\mathcal{S}_0) - \bar{w}}{\bar{w}}\right)$	$\left(\frac{\kappa(\mathcal{S}_1) - \bar{w}}{\bar{w}}\right)$	$\left(\frac{\kappa(\mathcal{S}_2) - \bar{w}}{\bar{w}}\right)$
0	[1, 1]	250	14.6%	14.6%	1.1%
		500	15.3%	15.3%	2.2%
0	[-50, 50]	250	21.0%	19.6%	0.0%
		500	20.8%	20.3%	1.0%
25	[-50, 50]	250	4.5%	4.0%	0.0%
		500	3.6%	3.4%	1.8%
-75	[50, 100]	250	16.2%	16.2%	1.5%
		500	17.5%	17.5%	3.5%
75	[50, 100]	250	9.3%	9.3%	1.1%
		500	9.6%	9.6%	1.8%

(b) Computing times obtained from XPRESS-Barrier.

<i>Exterior Field</i> ( $h$ )	<i>Edge's Weights</i> ( $[w^-, w^+]$ )	<i>Vertices</i> ( $ V $ )	<i>Linear Programming</i>		
			$\kappa(\mathcal{S}_0)$	$\kappa(\mathcal{S}_1)$	$\kappa(\mathcal{S}_2)$
0	[1, 1]	250	0.1 s	0.3 s	63.2 s
		500	0.3 s	0.9 s	914.5 s
0	[-50, 50]	250	0.1 s	0.3 s	108.6 s
		500	0.3 s	0.8 s	1039.4 s
25	[-50, 50]	250	0.1 s	0.3 s	91.8 s
		500	0.3 s	0.9 s	1182.1 s
-75	[50, 100]	250	0.1 s	0.3 s	50.1 s
		500	0.3 s	0.8 s	656.7 s
75	[50, 100]	250	0.1 s	0.3 s	54.0 s
		500	0.3 s	0.9 s	619.7 s

\*Computed on an AMD Athlon 64 X2 Dual Core 4800+,  
2.41 GHz, 4GB RAM and runs XP.

## 8.8 A closer look at $C_4$

$C_4(f)$  is a lower bound to the minimum of the quadratic pseudo-Boolean function  $f$ , and belongs to the fourth level of the hierarchy of bounds proposed by Boros et al. [49], following its immediate level  $C_3(f)$  bound. There is not much work in the literature about what properties characterize this bound, as well as its usefulness is not known in practical applications. In this section we provide a characterization of  $C_4$  in terms of cones of positive functions and in terms of LP. At the end, we will show an example derived from the minimum 3-partition minimization for which  $C_3$  provides weak bounds but  $C_4$  gives remarkably better results.

In what follows we shall consider quadratic pseudo-Boolean functions in  $n = |V|$  variables as vectors of the  $1 + n + \binom{n}{2}$  multilinear coefficients of their unique polynomial representation.

For a subset  $S \subseteq V$  of the variables, we denote by  $\mathcal{F}_S$  the family of quadratic pseudo-Boolean functions only depending on variables from  $S$ . Let  $\mathcal{F}_S^+ \subseteq \mathcal{F}_S$  the subfamily of nonnegative ones. It is simple to verify that  $\mathcal{F}_S$  is a subspace of dimension  $1 + |S| + \binom{|S|}{2}$  of  $\mathbb{R}^{1+|S|+\binom{|S|}{2}}$ , and that  $\mathcal{F}_S^+$  is a convex cone in this subset.  $\mathcal{F}_S^+$  is described by the  $2^{|S|}$  inequalities requiring the nonnegativity of its elements. Therefore it is a polyhedral cone and consequently it is finitely generated. Let

$$\mathcal{Q}_k = \{ \mathcal{F}_S^+ \mid S \subseteq V, |S| \leq k \}$$

be a convex cone in  $\mathbb{R}^{1+|S|+\binom{|S|}{2}}$  for  $2 \leq k \leq n$ , which is generated by the above cones that correspond to at most  $k$  variables.

**Proposition 8.9** ([49, 54]).

$$\mathcal{Q}_2 \subseteq \mathcal{Q}_3 \subseteq \mathcal{Q}_4 \subseteq \cdots \subseteq \mathcal{Q}_n,$$

*and all of these cones are finitely generated.*

We denote the generators of cone  $\mathcal{Q}_k$  ( $k = 2, \dots, n$ ) as the set  $\mathcal{B}(\mathcal{Q}_k)$ . The characterization of the extremal elements of the above cones is important but difficult to be

fully understood ([54]). It is well defined for cone  $\mathcal{Q}_3$  and we provide here for the first time a characterization of the generators of  $\mathcal{Q}_4$ .

Before showing this result, let us first consider a special family of functions proposed by Boros et al. [49]:

$$b_{U,\alpha} \stackrel{\text{def}}{=} \binom{\sum_{u \in U} u - \alpha}{2} \quad (8.14)$$

where  $U \subseteq \mathbf{L}$  is a subset of the literals with not complemented pairs, and where  $\alpha \in \mathbb{Z}$ .

It is clear that (8.14) defines a pseudo-Boolean function and that by using the identity  $u^2 = u$ , a quadratic polynomial representation of it can be computed.

**Example 8.12.**

$$\begin{aligned} b_{\{u,v,w,z\},1} &= \frac{(u+v+w+z-1)(u+v+w+z-2)}{2} \\ &= 1 - u - v - w - z + uv + uw + uz + vw + vz + wz. \end{aligned}$$

**Proposition 8.10** ([49]). *If  $U \subseteq \mathbf{L}$  is a subset of the literals containing no complemented pairs, and  $\alpha$  is an integer such that  $1 \leq \alpha \leq |U| - 2$  for  $|U| \geq 3$ , and  $\alpha = 1$  for  $|U| = 2$ , then  $b_{U,\alpha} \in \mathcal{B}(\mathcal{Q}_k)$  for  $k \geq |U|$ .*

**Proposition 8.11** ([50]). *Let*

$$\begin{aligned} \mathcal{B}_2 &= \{uv \mid u, v \in \mathbf{L}, u \neq v, u \neq \bar{v}\} \\ &= \{b_{U,1} \mid U \subseteq \mathbf{L} \text{ containing no complemented literals}, |U| = 2\}, \\ \mathcal{B}_3 &= \{uvw + \bar{u}\bar{v}\bar{w} \mid u, v, w \in \mathbf{L}, u \notin \{v, \bar{v}, w, \bar{w}\}, v \notin \{w, \bar{w}\}\} \\ &= \{b_{U,1} \mid U \subseteq \mathbf{L} \text{ containing no complemented literals}, |U| = 3\}, \end{aligned}$$

*Then, we have  $\mathcal{B}(\mathcal{Q}_2) = \mathcal{B}_2$  and  $\mathcal{B}(\mathcal{Q}_3) = \mathcal{B}_2 \cup \mathcal{B}_3$ .*

Ahead, we will characterize  $\mathcal{B}(\mathcal{Q}_4)$  which also has generators of the form (8.14). To be remarked that not every generator of the cones  $\mathcal{Q}_k$  is a function of the form (8.14). Boros and Hammer [53] provided several families of extremal elements of  $\mathcal{Q}_k$  that are not of this form.

From the comments of the previous sections it is also not a big surprise to see that the ‘‘cubic’’ generators of the cone  $\mathcal{Q}_3$  are precisely the mated cubic terms. So, one

immediately guesses that there is a strong connection between  $\mathcal{Q}_3$  and the cubic dual  $C_3$ . To see this, let us define

$$C_k(f) = \max \{C \in \mathbb{R} \mid f - C \in \mathcal{Q}_k\}$$

for all  $k = 2, \dots, n$ . Since  $f$  has finite possible values and since  $\mathcal{Q}_k$  is a closed convex cone in  $\mathbb{R}^{1+n+\binom{n}{2}}$  then the maximum of the above definition exists. From the definition of the cones  $\mathcal{Q}_k$  then it is clear that

$$C_2(f) \leq C_3(f) \leq C_4(f) \leq \dots \leq C_n(f) = \min_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x}).$$

Using the basis generators  $\mathcal{B}(\mathcal{Q}_k)$  of cone  $\mathcal{Q}_k$ , then the lower bound  $C_k(f)$  can be expressed as the optimum of the linear programming problem

$$C_k(f) = \max \left\{ C \in \mathbb{R} \mid f - C = \sum_{b \in \mathcal{B}(\mathcal{Q}_k)} \alpha_b b, \quad \alpha_b \in \mathcal{B}(\mathcal{Q}_k) \right\},$$

where the equations correspond to the  $1 + n + \binom{n}{2}$  coefficients of  $f$ .

From Proposition 8.11 it is easy to see that  $C_2(f)$  corresponds to the roof-dual and  $C_3(f)$  corresponds to the cubic-dual.

Let us show next some preliminary results that will lead to show how to find the basis generators of  $\mathcal{Q}_4$ .

**Lemma 8.11.** *Let  $\lambda_u, \lambda_v, \lambda_w, \lambda_z \in [0, 1]$  and  $\lambda_u + \lambda_v + \lambda_w + \lambda_z = 1$ . Then  $b_{\{u,v,w,z\},1}$  has the following quartic posiform representation:*

$$\begin{aligned} b_{\{u,v,w,z\},1} &= \bar{u}\bar{v}\bar{w}\bar{z} \\ &+ \lambda_u \bar{u}(v w z) + (1 - \lambda_u) v w z \\ &+ \lambda_v \bar{v}(u w z) + (1 - \lambda_v) u w z \\ &+ \lambda_w \bar{w}(u v z) + (1 - \lambda_w) u v z \\ &+ \lambda_z \bar{z}(v w z) + (1 - \lambda_z) v w z. \end{aligned}$$

*Proof.* To see this result we apply the following trivial steps:

$$\begin{aligned}
b_{U,1} &= \bar{u}\bar{v}\bar{w}\bar{z} - uvwz + uvw + uvz + uwz + vzw \\
&= \bar{u}\bar{v}\bar{w}\bar{z} - (\lambda_u + \lambda_v + \lambda_w + \lambda_z)uvwz + uvw + uvz + uwz + vzw \\
&= \bar{u}\bar{v}\bar{w}\bar{z} \\
&\quad + \lambda_u\bar{u}vwz - \lambda_uvwz + vzw \\
&\quad + \lambda_v\bar{u}\bar{v}wz - \lambda_uuwz + uwz \\
&\quad + \lambda_wuv\bar{w}z - \lambda_wuvz + uvz \\
&\quad + \lambda_zuvw\bar{z} - \lambda_zuvw + uvw.
\end{aligned}$$

□

**Lemma 8.12.** *Let  $\phi = \sum_{T \subseteq \{u,v,w,z,\bar{u},\bar{v},\bar{w},\bar{z}\}} \alpha_T \prod_{a \in T} a$  be a posiform representing a quadratic pseudo-Boolean function  $f : \mathbb{B}^{\{u,v,w,z\}} \rightarrow \mathbb{R}$ . Then, there is a posiform  $\theta = \sum_{T \subseteq \{u,v,w,z,\bar{u},\bar{v},\bar{w},\bar{z}\}} \beta_T \prod_{a \in T} a$  representing  $f$ , which satisfies the following conditions on its terms of degree 4:*

$$\beta_{\{u,v,w,z\}} = \alpha_{\{\bar{u},\bar{v},\bar{w},\bar{z}\}} + \alpha_{ch\bar{u},\bar{v},w,\bar{z}} + \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} + \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}}$$

$$\beta_{\{\bar{u},\bar{v},\bar{w},z\}} = \alpha_{\{\bar{u},\bar{v},\bar{w},z\}}$$

$$\beta_{\{\bar{u},\bar{v},w,\bar{z}\}} = \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}}$$

$$\beta_{\{\bar{u},v,\bar{w},\bar{z}\}} = \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}}$$

$$\beta_{\{u,\bar{v},\bar{w},\bar{z}\}} = \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}}$$

$$\beta_T = 0, \quad |T| = 4,$$

$$T \in A \stackrel{def}{=} \left\{ \begin{array}{l} \{\bar{u}, \bar{v}, \bar{w}, \bar{z}\}, \{u, v, w, \bar{z}\}, \{u, v, \bar{w}, z\}, \{u, \bar{v}, w, z\}, \{\bar{u}, v, w, z\}, \\ \{u, v, \bar{w}, \bar{z}\}, \{u, \bar{v}, w, \bar{z}\}, \{\bar{u}, v, w, \bar{z}\}, \{u, \bar{v}, \bar{w}, z\}, \{\bar{u}, v, \bar{w}, z\}, \{\bar{u}, \bar{v}, w, z\} \end{array} \right\}.$$

*Proof.* If the quartic term only differs in one literal (say  $\{\bar{u}, v, w, z\}$ ) from  $\{u, v, w, z\}$

then we apply the relation:

$$\begin{aligned} & \alpha_{\{u,v,w,z\}}uvwz + \alpha_{\{\bar{u},v,w,z\}}\bar{u}vwz \\ = & (\alpha_{\{u,v,w,z\}} - \alpha_{\{\bar{u},v,w,z\}})uvwz + \alpha_{\{\bar{u},v,w,z\}}vwz. \end{aligned}$$

If the quartic term differs in two literals (say  $\{\bar{u}, \bar{v}, w, z\}$ ) from  $\{u, v, w, z\}$  then we apply the relation:

$$\begin{aligned} & \alpha_{\{u,v,w,z\}}uvwz + \alpha_{\{\bar{u},\bar{v},w,z\}}\bar{u}\bar{v}wz \\ = & (\alpha_{\{u,v,w,z\}} + \alpha_{\{\bar{u},\bar{v},w,z\}})uvwz - \alpha_{\{\bar{u},\bar{v},w,z\}}(uwz + vwz) + \alpha_{\{\bar{u},\bar{v},w,z\}}wz. \end{aligned}$$

If the quartic term differs in all literals from  $\{u, v, w, z\}$  then we apply the relation:

$$\begin{aligned} & \alpha_{\{u,v,w,z\}}uvwz + \alpha_{\{\bar{u},v,w,z\}}\bar{u}v\bar{w}z \\ = & (\alpha_{\{u,v,w,z\}} + \alpha_{\{\bar{u},v,w,z\}})uvwz - \alpha_{\{\bar{u},v,w,z\}}(uvw + uvz + uwz + vwz) \\ & + \alpha_{\{\bar{u},v,w,z\}}(uv + uw + uz + vw + vz + wz) - \alpha_{\{\bar{u},v,w,z\}}. \end{aligned}$$

Obviously, it is always possible to find a posiform of the resulting expression. Also, it is clear that  $\beta_T = 0$ ,  $|T| = 4$ ,  $T \in A$  and the quartic terms differing in 3 literals from  $\{a, b, c, d\}$  have the same coefficient. Since  $\phi$  is a quadratic pseudo-Boolean function then all corresponding quartic terms must vanish when all terms of degree 4 are combined. Therefore,

$$\begin{aligned} 0 = & \alpha_{\{u,v,w,z\}} + \alpha_{\{\bar{u},v,w,z\}} + \alpha_{\{u,\bar{v},w,z\}} + \alpha_{\{u,v,\bar{w},z\}} + \alpha_{\{u,v,w,\bar{z}\}} \\ & - (\alpha_{\{\bar{u},\bar{v},w,z\}} + \alpha_{\{\bar{u},v,\bar{w},z\}} + \alpha_{\{\bar{u},v,w,\bar{z}\}} + \alpha_{\{u,\bar{v},\bar{w},z\}} + \alpha_{\{u,\bar{v},w,\bar{z}\}} + \alpha_{\{u,v,\bar{w},\bar{z}\}}) \\ & + (\alpha_{\{\bar{u},\bar{v},\bar{w},z\}} + \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}} + \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} + \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}}) - \alpha_{\{\bar{u},\bar{v},\bar{w},\bar{z}\}}. \end{aligned}$$

and using the coefficients obtained in the previous sequence of relations we get

$$0 = -\beta_{\{u,v,w,z\}} + (\alpha_{\{\bar{u},\bar{v},\bar{w},z\}} + \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}} + \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} + \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}}).$$

□

**Theorem 8.3.** *Let*

$$\mathcal{B}_4 = \left\{ \begin{array}{l} \lambda_u \bar{u} (v w z + \bar{v} \bar{w} \bar{z}) + (1 - \lambda_u) (v w z) + \\ \lambda_v \bar{v} (u w z + \bar{u} \bar{w} \bar{z}) + (1 - \lambda_v) (u w z) + \\ \lambda_w \bar{w} (u v z + \bar{u} \bar{v} \bar{z}) + (1 - \lambda_w) (u v z) + \\ \lambda_z \bar{z} (u v w + \bar{u} \bar{v} \bar{w}) + (1 - \lambda_z) (u v z) \\ |u, v, w, z \in \mathbf{L}, u \notin \{v, \bar{v}, w, \bar{w}, z, \bar{z}\}, v \notin \{w, \bar{w}, z, \bar{z}\}, w \notin \{z, \bar{z}\} \end{array} \right\},$$

$$\lambda_u, \lambda_v, \lambda_w, \lambda_z \in [0, 1], \lambda_u + \lambda_v + \lambda_w + \lambda_z = 1$$

$$= \{b_{U,1} | U \subseteq \mathbf{L} \text{ containing no complemented literals}, |U| = 4\}.$$

Then, we have  $\mathcal{B}(\mathcal{Q}_4) \subseteq \mathcal{B}_2 \cup \mathcal{B}_3 \cup \mathcal{B}_4$ .

*Proof.* Since  $b_{\{u,v,w,z\},1} = 1 - u - v - w - z + uv + uw + uz + vw + vz + wz$ , then  $\mathcal{B}_4$  is contained in  $\mathcal{Q}_4$ . Consider now a posiform  $\phi$  and assume that  $\phi \in \mathcal{Q}_4$ . We want to show that the posiform  $\theta \in \mathcal{Q}_4$  obtained from  $\phi$ , according to Lemma 8.12, can be written as a nonnegative combination of posiforms from  $\mathcal{B}_2 \cup \mathcal{B}_3 \cup \mathcal{B}_4$ . To check this, we express  $\theta$  in the form

$$\theta = \sum_{b \in \mathcal{B}_2 \cup \mathcal{B}_3 \cup \mathcal{B}_4} \lambda_b b + \sum_{T \in \Lambda} \alpha_T \prod_{u \in T} u,$$

where  $\lambda_b \geq 0$  ( $b \in \mathcal{B}_2 \cup \mathcal{B}_3 \cup \mathcal{B}_4$ ),  $\alpha_T > 0$ ,  $|T| \in \{3, 4\}$  ( $T \in \Lambda$ ), and  $\Lambda$  having the smallest number of quartic terms. Trivially,  $\theta$  can always be expressed in this form. Say  $|\Lambda| > 0$  and that at least a term of length 4 exists. Note that if there is no such term ( $\mathcal{B}_4 = \emptyset$ ) then according to Boros et al. [50]  $\mathcal{B}_2 \cup \mathcal{B}_3 \cup \mathcal{B}_4$  is a basis for  $\mathcal{Q}_3 \subseteq \mathcal{Q}_4$ . So, let us assume that a quartic term exists and let us call it  $T_n = \{u, v, w, z\} \in \Lambda$ . Since  $\theta$  is a quartic form of a quadratic pseudo-Boolean function, the quartic part of  $\alpha_{T_n} uvwz$  must be cancelled by some other quartic terms in  $\Lambda$ , which can only be of the form  $T_2 = \{\bar{u}, \bar{v}, \bar{w}, z\}$ . From lemma 8.12, we may assume that  $\alpha_{T_n} \geq \alpha_{T_2}$ . Then

$$\alpha_{T_n} uvwz + \alpha_{T_2} \bar{u} \bar{v} \bar{w} z = (\alpha_{T_n} - \alpha_{T_2}) uvwz + \alpha_{T_4} z (uv + uw + vw) - \alpha_{T_4} z (u + v + w) + \alpha_{T_4} z \quad (8.15)$$

These simple algebraic transformations contradict our initial assumption that  $\Lambda$  contains the smallest number of quartic terms. So, what it is left to prove is the case when  $\Lambda$  consists only of terms of length 3. So, let us assume that  $\Lambda$  consists of the smallest number of cubic terms and that at least one cubic term exists. We call this term as  $C_n = \{v, w, z\} \in \Lambda$ . Since  $\theta$  is a quadratic pseudo-Boolean function, the cubic part of  $\alpha_{C_n} v w z$  must be cancelled by some other cubic terms in  $\Lambda$ , which can only be of the form  $C_2 = \{\bar{v}, w, z\}$ . We may assume without loss of generality that  $\alpha_{C_n} \geq \alpha_{C_2}$ . Then,  $\alpha_{C_n} v w z + \alpha_{C_2} \bar{v} w z = (\alpha_{C_n} - \alpha_{C_2}) v w z + \alpha_{T_2} w z$ . If  $\alpha_{C_n} > \alpha_{C_2}$ , then the only possibility to cancel the  $(\alpha_{C_n} - \alpha_{C_2}) v w z$  term is by having in  $\Lambda$  the cubic term  $(\alpha_{C_n} - \alpha_{C_2}) \bar{v} \bar{w} \bar{z}$ . But, this situation would imply that  $(\alpha_{C_n} - \alpha_{C_2}) (v w z + \bar{v} \bar{w} \bar{z})$  belong to  $\Lambda$  and not  $\mathcal{B}_3$  as we have assumed. Therefore, the cubic terms originally existent in  $\Lambda$  must be cancelled by the new cubic terms defined in (8.15). Let  $c_T$  be the coefficient of the cubic term  $T(\{u, v, w\}, \{u, v, z\}, \{u, w, z\}, \{v, w, z\})$  generated by (8.15). According to these algebraic transformations, if there is a quartic term  $T_n$  in  $\Lambda$  with a positive coefficient, then we must have the following system of equations satisfied:

$$\begin{cases} c_{\{u,v,w\}} &= \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}} + \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} + \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}} \\ c_{\{u,v,z\}} &= \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}} + \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} + \alpha_{\{\bar{u},\bar{v},\bar{w},z\}} \\ c_{\{u,w,z\}} &= \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}} + \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}} + \alpha_{\{\bar{u},\bar{v},\bar{w},z\}} \\ c_{\{v,w,z\}} &= \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} + \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}} + \alpha_{\{\bar{u},\bar{v},\bar{w},z\}} \end{cases} \quad (8.16)$$

Now, recalling from lemma 8.12 that

$$\alpha_{T_n} = \alpha_{\{\bar{u},\bar{v},\bar{w},z\}} + \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}} + \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} + \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}},$$

and letting

$$(\lambda_u, \lambda_v, \lambda_w, \lambda_z) = \left( \frac{\alpha_{\{u,\bar{v},\bar{w},\bar{z}\}}}{\alpha_{T_n}}, \frac{\alpha_{\{\bar{u},v,\bar{w},\bar{z}\}}}{\alpha_{T_n}}, \frac{\alpha_{\{\bar{u},\bar{v},w,\bar{z}\}}}{\alpha_{T_n}}, \frac{\alpha_{\{\bar{u},\bar{v},\bar{w},z\}}}{\alpha_{T_n}} \right).$$

Then (8.15) and (8.16) imply that

$$\begin{aligned}
& \left( \alpha_{T_n} uvwz + \alpha_{\{u,\bar{v},\bar{w},\bar{z}\}} u\bar{v}\bar{w}\bar{z} + \alpha_{\{\bar{u},v,\bar{w},\bar{z}\}} \bar{u}v\bar{w}\bar{z} + \alpha_{\{\bar{u},\bar{v},w,\bar{z}\}} \bar{u}\bar{v}w\bar{z} + \alpha_{\{\bar{u},\bar{v},\bar{w},z\}} \bar{u}\bar{v}\bar{w}z \right) \\
& \quad + (c_{\{u,v,w\}} \bar{u}\bar{v}\bar{w} + c_{\{u,v,z\}} \bar{u}\bar{v}\bar{z} + c_{\{u,w,z\}} \bar{u}\bar{w}\bar{z} + c_{\{v,w,z\}} \bar{v}\bar{w}\bar{z}) \\
= & \alpha_{T_n} \left( \begin{aligned} & uvwz + \lambda_u u\bar{v}\bar{w}\bar{z} + \lambda_v \bar{u}v\bar{w}\bar{z} + \lambda_w \bar{u}\bar{v}w\bar{z} + \lambda_z \bar{u}\bar{v}\bar{w}z \\ & + (\lambda_u + \lambda_v + \lambda_w) \bar{u}\bar{v}\bar{w} + (\lambda_u + \lambda_v + \lambda_z) \bar{u}\bar{v}\bar{z} \\ & + (\lambda_u + \lambda_w + \lambda_z) \bar{u}\bar{w}\bar{z} + (\lambda_v + \lambda_w + \lambda_z) \bar{v}\bar{w}\bar{z} \end{aligned} \right) \\
= & \alpha_{T_n} \left( \begin{aligned} & uvwz + \lambda_u u\bar{v}\bar{w}\bar{z} + \lambda_v \bar{u}v\bar{w}\bar{z} + \lambda_u \bar{u}\bar{v}w\bar{z} + \lambda_z \bar{u}\bar{v}\bar{w}z \\ & + (1 - \lambda_z) \bar{u}\bar{v}\bar{w} + (1 - \lambda_w) \bar{u}\bar{v}\bar{z} + (1 - \lambda_v) \bar{u}\bar{w}\bar{z} + (1 - \lambda_u) \bar{v}\bar{w}\bar{z} \end{aligned} \right) \\
= & \alpha_{T_n} \left( \begin{aligned} & \lambda_u u (vwz + \bar{v}\bar{w}\bar{z}) + \lambda_v v (uwz + \bar{u}\bar{w}\bar{z}) + \lambda_u w (uvz + \bar{u}\bar{v}\bar{z}) + \lambda_z z (uvw + \bar{u}\bar{v}\bar{w}) \\ & (1 - \lambda_z) \bar{u}\bar{v}\bar{w} + (1 - \lambda_w) \bar{u}\bar{v}\bar{z} + (1 - \lambda_v) \bar{u}\bar{w}\bar{z} + (1 - \lambda_u) \bar{v}\bar{w}\bar{z} \end{aligned} \right)
\end{aligned}$$

must be a partial sum contained in the sum of  $\theta$  defined by  $\Lambda$ . Because there is a element of  $\mathcal{B}_4$  with terms in  $\Lambda$  with positive coefficients we got a contradiction as the size of  $\Lambda$  can be reduced.  $\square$

Note that  $b_{\{u,v,w\},1}$  has a unique cubic posiform representation in the literals  $u$ ,  $v$  and  $w$ . However,  $b_{\{u,v,w,z\},1}$  has several quartic posiforms representing it. One example is for instance

$$\begin{aligned}
b_{U\{u,v,w,z\},1} &= \bar{u}\bar{v}\bar{w}\bar{z} - uvwz + uvw + uvz + uwz + v wz \\
&= \bar{u}\bar{v}\bar{w}\bar{z} + \bar{u}vwz + uvw + uvz + uwz \\
&= \bar{u}(\bar{v}\bar{w}\bar{z} + vwz) + u(vw + vz + wz).
\end{aligned}$$

A consequence of the previous observation is that to compute  $C_4(f)$  by means of linear programming we need to consider 16 valid inequalities (out of possible 16) in the traditional linearization in order to assure that all the generators of  $\mathcal{B}_4$  are contemplated.

**Theorem 8.4.** *Let us define*

$$\mathbf{W}^{[4]} \stackrel{\text{def}}{=} \mathbf{W}^{[3]} \cup \left\{ (\mathbf{x}, \mathbf{y}) \left\{ \begin{array}{l} x_i + x_j + x_k + x_r - y_{i,j} - y_{i,k} - y_{i,r} - y_{j,k} - y_{j,r} - y_{k,r} \leq 1, \\ x_i + x_j + x_k - 2x_r - y_{i,j} - y_{i,k} + y_{i,r} - y_{j,k} + y_{j,r} + y_{k,r} \leq 1, \\ x_i + x_j - 2x_k + x_r - y_{i,j} + y_{i,k} - y_{i,r} + y_{j,k} - y_{j,r} + y_{k,r} \leq 1, \\ x_i - 2x_j + x_k + x_r + y_{i,j} - y_{i,k} - y_{i,r} + y_{j,k} + y_{j,r} - y_{k,r} \leq 1, \\ -2x_i + x_j + x_k + x_r + y_{i,j} + y_{i,k} + y_{i,r} - y_{j,k} - y_{j,r} - y_{k,r} \leq 1, \\ -x_i \qquad \qquad \qquad + y_{i,j} + y_{i,k} + y_{i,r} - y_{j,k} - y_{j,r} - y_{k,r} \leq 0, \\ \qquad \qquad \qquad -x_j \qquad \qquad \qquad + y_{i,j} - y_{i,k} - y_{i,r} + y_{j,k} + y_{j,r} - y_{k,r} \leq 0, \\ \qquad \qquad \qquad \qquad \qquad -x_k \qquad \qquad \qquad - y_{i,j} + y_{i,k} - y_{i,r} + y_{j,k} - y_{j,r} + y_{k,r} \leq 0, \\ \qquad \qquad \qquad \qquad \qquad \qquad \qquad -x_r \qquad \qquad \qquad - y_{i,j} - y_{i,k} + y_{i,r} - y_{j,k} + y_{j,r} + y_{k,r} \leq 0, \\ \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad -x_k \qquad -x_r \qquad - y_{i,j} + y_{i,k} + y_{i,r} + y_{j,k} + y_{j,r} - y_{k,r} \leq 0, \\ \qquad \qquad \qquad -x_j \qquad \qquad \qquad -x_r \qquad + y_{i,j} - y_{i,k} + y_{i,r} + y_{j,k} - y_{j,r} + y_{k,r} \leq 0, \\ \qquad \qquad \qquad -x_j \qquad -x_k \qquad \qquad \qquad + y_{i,j} + y_{i,k} - y_{i,r} - y_{j,k} + y_{j,r} + y_{k,r} \leq 0, \\ -x_i \qquad \qquad \qquad -x_r \qquad + y_{i,j} + y_{i,k} - y_{i,r} - y_{j,k} + y_{j,r} + y_{k,r} \leq 0, \\ -x_i \qquad \qquad \qquad -x_k \qquad \qquad \qquad + y_{i,j} - y_{i,k} + y_{i,r} + y_{j,k} - y_{j,r} + y_{k,r} \leq 0, \\ -x_i \qquad -x_j \qquad \qquad \qquad - y_{i,j} + y_{i,k} + y_{i,r} + y_{j,k} + y_{j,r} - y_{k,r} \leq 0, \\ 2x_i + 2x_j + 2x_k + 2x_r - y_{i,j} - y_{i,k} - y_{i,r} - y_{j,k} - y_{j,r} - y_{k,r} \leq 3, \\ (1 \leq i < j < k < r \leq n) \end{array} \right. \right\}. \quad (8.17)$$

Then

$$C_4(f) = \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{W}^{[4]}, \mathbf{x} \in \mathbb{U}^n \right\}.$$

A final idea that we would like to bring to this discussion is the possibility of improving the bounds by applying the arithmetic consensus repeatedly to a posiform, resulting possibly in an equivalent posiform with larger degree.

**Example 8.13.** *This example exemplifies how to apply the arithmetic consensus to transform a posiform of degree 4 into a posiform of degree 5:*

$$\begin{aligned} xyuv + xy\bar{v}z &= xy(uv + \bar{v}z) \\ &= xy(uz + uv\bar{z} + \bar{u}\bar{v}z) \\ &= xyuz + xyw\bar{z} + xy\bar{u}\bar{v}z. \end{aligned}$$

### 8.8.1 Computational results

The experiments shown in this section consider the Minimum  $k$ -Partition (MkP) problem. Given a weighted graph  $G = (V, E, \mathbf{w})$ , the MkP problem is the problem of

partitioning the set of vertices  $V$  into  $k$  disjoint subsets such that the total weight of the edges joining vertices of the same partition is minimum.

The  $MkP$  problem can be formulated as 0–1 LP (see [87]) or alternatively as a Semi-definite Program (see [20]). On a private communication Boros et al. [19] formulated the  $MkP$  problems as a QUBO as follows.

For each vertex  $i$  of  $G$  we associate  $k$  binary variables  $x_{ir}$  such that:

$$(i) \quad \sum_{r=1}^k x_{ir} = 1 \quad \text{and}$$

$$(ii) \quad \sum_{r=1}^k x_{ir}x_{jr} = 0 \quad \iff \quad i \text{ and } j \text{ are in different partitions.}$$

If vertices  $i$  and  $j$  are grouped together on the same partition then the objective is penalized by the weight  $w_{ij}$ . Given a feasible assignment according to (i) and (ii), then the quadratic pseudo-Boolean function

$$f_k(\mathbf{x}) = \sum_{i \in V} \left( \sum_{j \in V | i < j} w_{ij} \sum_{r=1}^k x_{ir}x_{jr} + M \left( \sum_{r=1}^k x_{ir} - 1 \right)^2 \right)$$

represents the total weight of a  $k$ -partition of  $G$ . If  $M$  is large enough ( $M = \sum_{i,j \in V | i < j} |w_{ij}|$  is enough), then the minimizers of  $f_k$  are characteristic vectors of weighted minimum  $k$ -partitions of graph  $G$ .

For  $k = 3$  this approach can be specialized further, since

$$\begin{cases} x_{i3} = 1 - x_{i1} - x_{i2}, \\ x_{i3} \in \mathbb{B} \end{cases} \iff x_{i1}x_{i2} = 0.$$

for any vertex  $i \in V$ . Thus, the minimizers of the quadratic pseudo-Boolean function

$$\begin{aligned} g(\mathbf{x}) &= \sum_{i \in V} \left( \sum_{j \in V | i < j} w_{ij} (x_{i1}x_{j1} + x_{i2}x_{j2} + (1 - x_{i1} - x_{i2})(1 - x_{j1} - x_{j2})) + Mx_{i1}x_{i2} \right) \\ &= \sum_{i \in V} \left( \sum_{j \in V | i < j} w_{ij} (2x_{i1}x_{j1} + 2x_{i2}x_{j2} + \bar{x}_{i1}\bar{x}_{j2} + \bar{x}_{i2}\bar{x}_{j1} - 1) + Mx_{i1}x_{i2} \right), \end{aligned}$$

are characteristic vectors of minimum weighted 3-partitions of  $G$ . To be noted that  $g$  is only defined by  $2|V|$  vertices.

Since the number of inequalities required to compute  $C_4$  is very large, in the next experiments we consider only a subset of the cuts. Instead of considering the 16 cuts of (8.17) for all cases in  $(1 \leq i < j < k < r \leq n)$  we only consider a subset of these cases  $\mathcal{Z}$  such that every element  $(i, j, k, r) \in \mathcal{Z}$  satisfies

$$c_{ik}c_{ij}c_{kr}c_{jr} \neq 0 \text{ or } c_{ij}c_{ir}c_{jk}c_{kr} \neq 0. \quad (8.18)$$

Condition (8.18) defines a tuple associated to 16 cuts not available in  $\mathbf{W}^{[3]}$  that due to its shape in the nonzero coefficients space, we named as the “pure square” inequalities. We denote this reduced subspace as  $\mathbf{W}^{[4]}(\mathcal{Z})$ .

In this section we investigate some M3P problems considered by Anjos et al. [20]. The graphs in question were generated by the software RUDY ([211]) and consist of 2-dimensional and 3-dimensional randomly generated Ising instances, some having Gaussian distributed weights and the others having +1 or -1 weights with 50% probability.

Using Xpress, under the same conditions of Section 8.7.1, we analyzed 4 bounds:  $\kappa(f, \mathcal{S}_1)$ ,  $\kappa(f, \mathcal{S}_2)$ ,  $z(f, \mathcal{S}_1)$  and  $z(f, \mathcal{S}_2)$ , where

$$z(f, \mathcal{S}) = \min \left\{ L_f(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{W}^{[3]}(\mathcal{S}) \cup \mathbf{W}^{[4]}(\mathcal{Z}), \mathbf{x} \in \mathbb{U}^n \right\}. \quad (8.19)$$

Table 8.11(a) includes the values of the four bounds, and Table 8.11(b) shows the corresponding computing times. There are two aspects to be emphasized:

- $z(f, \mathcal{S}_1)$  provides almost the same bounds as  $z(f, \mathcal{S}_2)$ , but it is much less computing demanding;
- The pure square cuts make the most difference with respect to how close the bound is from the minimum.

M3P is the (only) problem that we are aware of for which  $C_4$  is clearly superior than  $C_3$ , and therefore making possible to solve them in practice by using linear programming together with square inequalities (see Chapter 9).

Table 8.11: Lower bounds for M3P problems proposed by Anjos et al. [20].

(a) Lower bounds.

Instance	Weights	M3P	Without Pure Square Cuts		With Pure Square Cuts	
			$\kappa(\mathcal{S}_1)$	$\kappa(\mathcal{S}_2)$	$z(\mathcal{S}_1)$	$z(\mathcal{S}_2)$
4×4	Gaussian	-954 077	-1,222,806.7	-1,222,806.7	-954,077.0	-954,077.0
5×5		-1 484 348	-2,104,102.3	-2,078,937.0	-1,535,693.0	-1,496,165.6
6×6		-2 865 560	-3,724,596.0	-3,704,117.0	-2,952,370.3	-2,932,387.9
7×7		-3 282 435	-4,750,640.0	-4,750,640.0	-3,353,935.1	-3,350,514.6
8×8		-5 935 339	-7,186,373.7	-7,186,373.7	-6,004,188.1	-6,002,920.4
4×4	±1	-13	-18.0	-17.5	-13.8	-13.6
5×5		-20	-29.3	-29.3	-22.3	-22.1
6×6		-29	-42.3	-42.0	-31.9	-31.7
7×7		-40	-57.7	-57.7	-43.2	-43.1
8×8		-55	-77.7	-77.7	-58.4	-58.3
9×9	-65	-95.0	-95.0	-70.3	-70.3	
2×3×4	±1	-20	-32.5	-32.4	-23.1	-23.1
2×4×4		-28	-44.5	-44.2	-32.7	-32.7
3×3×3		-26	-42.3	-42.3	-30.0	-30.0
3×3×4		-36	-58.8	-58.8	-42.1	-41.9
3×4×4		-48	-79.2	-79.2	-56.7	-56.7
3×4×5		-65	-101.7	-101.7	-73.5	-73.4
4×4×4		-65	-108.7	-108.3	-79.1	-78.9

(b) XPRESS-Barrier computing times\*.

Instance	Weights	Without Pure Square Cuts		With Pure Square Cuts	
		$\kappa(\mathcal{S}_1)$	$\kappa(\mathcal{S}_2)$	$z(\mathcal{S}_1)$	$z(\mathcal{S}_2)$
4×4	Gaussian	0.1 s	0.3 s	0.9 s	1.1 s
5×5		0.2 s	0.9 s	1.3 s	2.3 s
6×6		0.3 s	2.5 s	1.9 s	5.7 s
7×7		0.5 s	6.7 s	2.9 s	11.1 s
8×8		0.7 s	14.9 s	3.9 s	29.2 s
4×4	±1	0.1 s	0.3 s	0.8 s	1.1 s
5×5		0.2 s	0.9 s	1.4 s	2.7 s
6×6		0.3 s	2.6 s	1.8 s	5.2 s
7×7		0.5 s	7.0 s	2.6 s	10.6 s
8×8		0.7 s	15.9 s	3.4 s	26.0 s
9×9	0.8 s	29.4 s	4.8 s	59.1 s	
2×3×4	±1	0.2 s	0.8 s	1.4 s	2.4 s
2×4×4		0.4 s	2.1 s	2.3 s	5.5 s
3×3×3		0.5 s	1.9 s	3.0 s	5.3 s
3×3×4		0.9 s	4.5 s	5.3 s	11.4 s
3×4×4		1.3 s	12.2 s	8.0 s	26.0 s
3×4×5		1.8 s	23.5 s	9.9 s	61.7 s
4×4×4		2.2 s	31.5 s	12.5 s	66.0 s

\*Computed on an AMD Athlon 64 X2 Dual Core 4800+,  
2.41 GHz, 4GB RAM and runs XP.

## Chapter 9

### Exact Methods

Let us start by considering the family  $\mathcal{N}$  of quadratic pseudo-Boolean functions

$$\mathcal{N} = \left\{ f : \mathbb{B}^n \mapsto \mathbb{R} \left| n \in \mathbb{Z}^+, f(x_1, \dots, x_n) = -n(n-1) \sum_{i=1}^n x_i - \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} x_i + 2n \sum_{1 \leq i < j \leq n} x_i x_j \right. \right\},$$

proposed by Pardalos [191].

**Proposition 9.1** ([146, 191, 195]). *A quadratic pseudo-Boolean function  $f \in \mathcal{N}$  having a number  $n \in \mathbb{Z}^+$  of even variables satisfies the following properties:*

(i)

$$\nu(f) = \min_{(x_1, \dots, x_n) \in \mathbb{B}^n} f(x_1, \dots, x_n) = -\frac{n}{2} \left( \frac{n^2}{2} + 1 \right)$$

(ii) *The unique global minimum  $\mathbf{x}^*$  of  $f$  in  $\mathbb{B}^n$  is  $\mathbf{x}^* = (1, \dots, 1, 0, \dots, 0)$ , having exactly  $\frac{n}{2}$  ones followed by  $\frac{n}{2}$  zeros;*

(iii)  *$f$  has an exponential number of local minima. More precisely, every point with  $\frac{n}{2}$  ones is a local minimum of  $f$ , and therefore there are  $\binom{n}{n/2}$  local minima.*

In the case where  $n$  is odd, it is also known ([146, 193]) that there is a global minimum  $\mathbf{x}^* = (1, \dots, 1, 0, \dots, 0)$ , having exactly  $\lfloor \frac{n}{2} \rfloor$  ones followed by  $\lfloor \frac{n+1}{2} \rfloor$  zeros.

Class  $\mathcal{N}$  demonstrates that there are functions with an exponential number of local minima ([191]) in the parameter  $n$ . Due to this property, this family of quadratic pseudo-Boolean functions has been mentioned to be a “difficult class of test problems” ([146, 195]) for finding their minimum using exact algorithmic approaches.

We shall present next, a different proof of the previous results that leads to a good algorithm to solve any function in  $\mathcal{N}$ .

**Lemma 9.1.** *Let  $f$  be a quadratic pseudo-Boolean function belonging to family  $\mathcal{N}$ , then  $\overline{x_i^* x_j^*} = 0$  is a strong quadratic persistency in every minimizer  $(x_1, \dots, x_i^*, \dots, x_j^*, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ , for all  $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$  and  $\lceil \frac{n+1}{2} \rceil \leq j \leq n$ .*

*Proof.* Let  $U = \{1, \dots, \lfloor \frac{n}{2} \rfloor\}$  and  $Z = \{\lceil \frac{n+1}{2} \rceil, \dots, n\}$  respectively denote the set of first  $\lfloor \frac{n}{2} \rfloor$  and last remaining indices from the  $n$ -vector. Then, the first order derivatives of  $f$  are

$$\Delta_i(x_1, \dots, x_n) = \begin{cases} -1 - n(n-1) + 2n \left( \sum_{j=1}^{i-1} x_j + \sum_{j=i+1}^n x_j \right), & i \in U, \\ -n(n-1) + 2n \left( \sum_{j=1}^{i-1} x_j + \sum_{j=i+1}^n x_j \right), & i \in Z. \end{cases}$$

Let us consider now the  $(i, j)$ th second order derivative of  $f$

$$\begin{aligned} \Delta_{ij} &= \Delta_i(\mathbf{x}) - \Delta_j(\mathbf{x}) + 2n(x_i^* - x_j^*) \\ &= \begin{cases} 0, & i \in U, j \in U, \\ -1, & i \in U, j \in Z, \\ 0, & i \in Z, j \in Z. \end{cases} \end{aligned}$$

The claimed result follows immediately from Theorem 4.1 since  $\Delta_{ij} < 0$  ( $i \in U, j \in Z$ ).  $\square$

**Theorem 9.1.** *Let  $f$  be a quadratic pseudo-Boolean function belonging to family  $\mathcal{N}$ , then*

- (i) *If  $i \in U$ , then  $x_i^* = 1$  is a strong persistency in every minimizer  $(x_1, \dots, x_i^*, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ ;*
- (ii) *If  $i \in Z$  and  $n$  is even, then  $x_i^* = 0$  is a strong persistency in every minimizer  $(x_1, \dots, x_i^*, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ ;*
- (iii) *If  $i \in Z$  and  $n$  is odd, then  $x_i^* = 0$  is a weak persistency in a minimizer  $(x_1, \dots, x_i^*, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ .*

*Proof.* Let us consider a positive constant  $M \geq 2n$  and define the quadratic pseudo-Boolean function

$$g(x_1, \dots, x_n) = f(x_1, \dots, x_n) + M \sum_{i \in U} \sum_{j \in Z} \bar{x}_i x_j.$$

First, it is trivial to see that  $g(x_1, \dots, x_n) \geq f(x_1, \dots, x_n)$  for any binary vector  $(x_1, \dots, x_n) \in \mathbb{B}^n$ . Second, Lemma 9.1 implies that  $g(x_1, \dots, x_n) = f(x_1, \dots, x_n)$  in every  $(x_1, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ . Thus,  $f$  and  $g$  have the same minima.

The partial derivatives of  $g$  are given by

$$\Delta_i^g(x_1, \dots, x_n) = \begin{cases} -1 - n(n-1) + 2n \sum_{j \in U \setminus \{i\}} x_j + (2n - M) \sum_{j \in Z} x_j, & i \in U, \\ M \lfloor \frac{n}{2} \rfloor - n(n-1) + 2n \sum_{j \in U} x_j + (2n - M) \sum_{j \in Z \setminus \{i\}} x_j, & i \in Z. \end{cases}$$

Since  $M \geq 2n$ , then the maximum of the linear pseudo-Boolean function  $\Delta_i^g(x_1, \dots, x_n)$  ( $i \in U$ ) is

$$\begin{aligned} U_i^g &= -1 - n(n-1) + 2n(|U \setminus \{i\}|) \\ &= -1 - n(n-1) + 2n(\lfloor \frac{n}{2} \rfloor - 1) \\ &= \begin{cases} -1 - n, & n \text{ even,} \\ -1 - 2n, & n \text{ odd.} \end{cases} \end{aligned}$$

Since  $U_i^g < 0$  ( $i \in U$ ), then the necessary conditions of optimality established by Corollary 4.1 imply that  $x_i^* = 1$  ( $i \in U$ ) in every minima of  $g$ , or equivalently in every minimizer  $(x_1, \dots, x_i^*, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ .

Since  $M \geq 2n$ , then the minimum of the linear pseudo-Boolean function  $\Delta_i^g(x_1, \dots, x_n)$  ( $i \in Z$ ) is

$$\begin{aligned} L_i^g &= M \lfloor \frac{n}{2} \rfloor - n(n-1) + (2n - M)(|Z \setminus \{i\}|) \\ &= \begin{cases} M \frac{n}{2} - n(n-1) + (2n - M)(n - \frac{n}{2} - 1), & n \text{ even,} \\ M \frac{n-1}{2} - n(n-1) + (2n - M)(n - \frac{n-1}{2} - 1), & n \text{ odd.} \end{cases} \\ &= \begin{cases} M - n, & n \text{ even,} \\ 0, & n \text{ odd.} \end{cases} \end{aligned}$$

Since for  $n$  even we have  $L_i^g > 0$  ( $i \in Z$ ), then the necessary conditions of optimality imply that  $x_i^* = 0$  ( $i \in Z$ ) in every minima of  $g$ , or equivalently in every minimizer  $(x_1, \dots, x_i^*, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ . Since for  $n$  odd we have  $L_i^g \geq 0$  ( $i \in Z$ ), then the necessary conditions of optimality established by Proposition 4.1 imply that there are minima of  $g$  having  $x_i^* = 0$  ( $i \in Z$ ), or equivalently there exists a minimizer  $(x_1, \dots, x_i^*, \dots, x_n) \in \text{Argmin}_{\mathbb{B}^n}(f)$ .  $\square$

**Corollary 9.1.** *A quadratic pseudo-Boolean function  $f \in \mathcal{N}$  having a number  $n \in \mathbb{Z}^+$  of odd variables satisfies the following properties:*

(i)

$$\nu(f) = \min_{(x_1, \dots, x_n) \in \mathbb{B}^n} f(x_1, \dots, x_n) = -\frac{n-1}{4}(n^2 + n + 2);$$

(ii)  $f$  has exactly  $\frac{n+1}{2}$  minima given by  $\text{Argmin}_{\mathbb{B}^n}(f) = \left\{ (x_1, \dots, x_n) \in \mathbb{B}^n \mid \sum_{i \in Z} x_i \leq 1 \right\}$ .

*Proof.* By Theorem 9.1 there exists a minimizer  $(x_1^*, \dots, x_n^*)$  of  $f$  satisfying  $x_i^* = 1$  for every  $i \in U$ , and  $x_j^* = 0$  for every  $j \in Z$ . Thus, part (i) follows from the evaluation of  $f(x_1^*, \dots, x_n^*)$ .

By the strong persistency result of Theorem 9.1 then one can simplify function  $f$  by fixing all the variables  $x_i$  with indices  $i \in U$ , and use the resulting function to find the optimal values of the remaining variables  $x_j$  with indices  $j \in Z$ . Since for  $n$  odd we derive

$$f(U \leftarrow \{1, \dots, 1\})(\mathbf{x}^Z) = -\frac{n-1}{4}(n^2 + n + 2) + 2n \sum_{i,j \in Z: i < j} x_i x_j,$$

then part (i) implies that all minimizers of  $f$  must satisfy the condition  $\sum_{i,j \in Z: i < j} x_i x_j = 0$ , which proves part (ii) of the claim.  $\square$

The previous results show clearly that optimizing quadratic pseudo-Boolean functions in class  $\mathcal{N}$  is an easy task. This family of problems has however been used ([146, 195]) to demonstrate that certain exact approaches for QUBO will remain difficult.

The point in presenting this example in the beginning of this chapter about *exact methods* is to remark the fact that even if certain approaches have some difficulties solving a specific class of problems, when coupled with additional tools, may result in the efficient computation of optimal solutions for those problems.

When we started investigating this class of problems using exact approaches, we have also encountered difficulties. However, as soon as the second order derivatives preprocessing tool (see Section 7.1.2) has been used during the presolve stage, then every problem in class  $\mathcal{N}$  could be solved to optimality during the preprocessing stage.

Table 9.1(a) shows the computing times of PREPRO to find the minimum of several functions from family  $\mathcal{N}$ , with the number of variables  $n$  varying between 25 and 250.

Table 9.1: Exact solutions of some quadratic pseudo-Boolean functions from family  $\mathcal{N}$ .

(a) Found by PREPRO .

$f \in \mathcal{N}$	Minimum	PREPRO
$(n)$	$(\nu(f))$	Computing Time*
25	-3912	<0.1 s
50	-31275	0.8 s
75	-105487	5.7 s
100	-250050	23.4 s
125	-488312	54.1 s
150	-843825	124.7 s
175	-1339887	217.8 s
200	-2000100	414.4 s
225	-2847712	595.6 s
250	-3906375	977.5 s

\*Obtained on a P4 2.8 GHz running XP.

(b) Found by BIQMAC.

$f \in \mathcal{N}$	Lower	Upper	Number of	BIQMAC
$(n)$	Bound		B&B Nodes	Computing Time*
25	3916	-3912	91519	>10800 s
50	-31275	-31275	1	0.1 s
75	-105506	-105487	23837	>10800 s
100	-250050	-250050	1	0.2 s

\*Obtained on a P4 3.6 GHz.

The Pardalos and Rodgers [195] depth first search method is only capable to handle problems in this class and in the period of 15-20 minutes having up to 25 variables. We have also tested a standard MIP linearization model for QUBO using a state-of-the-art MIP solver (in this case XPRESS-MP). This solver also struggled in proving optimality

for problems derived from  $\mathcal{N}$ , for both cases where the number of variables was either odd or even.

Table 9.1(b) presents the computing times of some problems in  $\mathcal{N}$  found by the solver BIQMAC, a binary quadratic and MAX-CUT solver developed by Rendl et al. [206]. The results indicate that BIQMAC can solve problems in  $\mathcal{N}$  having an even number of variables very efficiently, without requiring branching. However, BIQMAC could not solve to optimality the instances having 25 and 75 variables in 3 hours of computing time and after branching on several thousands of nodes. We recall here that BIQMAC considers a relaxation based on the intersection of the semidefinite relaxation with a subset of the triangle inequalities. Contrary to the conclusion presented in [206], these results indicate that there are QUBOs with less than 100 variables that BIQMAC can not handle efficiently.

The idea of applying very specific algorithms to better handle QUBO problems of specific classes is somehow illustrated throughout this dissertation. One such example is the problem of finding minimum vertex cover of planar graphs. This problem is covered in detail in Section 10.1.6.

In this chapter, we also provide practical evidence of this claim on MAX-CUT problems derived from the one-dimensional Ising chain problem (see Section 9.3.1.5). When comparing the results on some of these problems using one of the proposed algorithms against those published recently, we get solve times in the order of seconds, whereas the methods proposed in the literature solve the same problems running for several hours, and using similar computer technology.

The next section will introduce some background about the past research done on solving QUBOs to optimality. In this dissertation we propose three exact approaches for QUBO. We found for each type of solver at least an application that make it very competitive with other state-of-the-art solvers for QUBO. The first approach presented in Section 9.2 is based on a simple enumerative approach. The second approach described in Section 9.3 is based on a generic branch-and-bound code whose basic construct is roof-duality implemented in the network model. Several strengthened bounds can be used to cutoff the search tree as much as possible. These bounds can be based on

the semidefinite relaxation for MAX-CUT or on any of the improved iterated roof-duality bounds introduced in Chapter 8. The last method analyzed is based on linear programming and mixed 0-1 integer programming.

An extensive evaluation of the proposed methods is given across many different applications related to QUBO (e.g., Ising model, MAX-2-SAT, MAX-CUT).

## 9.1 Background

Most of the exact approaches that have been considered to solve *general* QUBOs, are of the divide-and-conquer type, where branch-and-bound is the predominant adopted method, with some earlier approaches proposing an enumerative depth first search scheme. The research effort is usually put in the analysis and definition of lower bounds for the sub-problems, giving less importance to other important questions, like for instance the definition of the branching strategy and the sub-problem selection. Some of these exact algorithms are briefly described next:

- Pardalos and Rodgers [195] proposed a depth first search for QUBO. The bounding method used in [195] is (at most) the sum of the negative coefficients. The variable select for branching in a sub-problem is the one having the largest range of values (in terms of both minimum and maximum) for the corresponding gradient. A variant of a depth-first branch and bound algorithm is described and its numerical performance is presented in a more recent work of Pardalos et al. [146].
- An improved enumerative approach to [195] was proposed by Hansen et al. [135]. The bound adopted at the root node is the roof dual. The associated roof dual posiform is used for variable fixation and to obtain lower bounds in the interior of the search tree.
- Williams [235] proposed and analyzed a branch-and-bound method for QUBO, using the LP formulation that leads to the roof dual bound.
- Billionnet and Sutter [44] presented a branch-and-bound algorithm for QUBO minimization. At each node of the search tree the lower bound is computed in

three phases and is equal to  $C_2 + C'_3 + C'_4$ .  $C_2$  corresponds to the roof-dual. The computation of  $C'_3$  uses the characterization of some positive quadratic posiforms associated with the directed cycles of the implication network.  $C'_4$  is computed by searching in a posiform of degree 4, which leads to an algorithm that derives some generators of  $\mathcal{B}(\mathcal{Q}_4)$  (see [44] and Theorem 8.3).

- Chardaire and Sutter [82] proposed a decomposition method to compute a lower bound QUBO minimization. First, they show that any quadratic function can be expressed as a sum of particular quadratic functions whose minima can be computed by a simple branch and bound algorithm. Then, they demonstrate that among all possible decompositions, the best one can be found by a Lagrangian decomposition method. The proposed decomposition gives at least the roof dual bound.
- Helmberg and Rendl [139] present an approach for QUBO that combines a semi-definite relaxation with a cutting plane technique, and is applied in a branch-and-bound setting.
- Billionnet and Elloumi [41] proposed a Mixed Integer Quadratic Programming solver (MIQP) for QUBO. The main idea is to disregard the multi-linearity property and second to convexify the quadratic function in 0–1 variables. To do this one could use a classical trick that simply raises up the entries of the  $x^2$  terms until the  $Q$  matrix associated to the quadratic function becomes positive semidefinite. Then using the fact that  $x(1-x) = 0$ , they obtain an equivalent convex objective function, which can then be handled by the MIQP solver. They propose two methods to convexify the quadratic function: one is based in the determination of the smallest eigenvalue of the  $Q$  matrix, and a stronger approach that leads to the semidefinite program associated to a MAX-CUT equivalent of the QUBO problem.
- Fischer et al. [98] proposed a dynamic version of the bundle method to get approximate solutions to semidefinite programs with a nearly arbitrary number of

linear inequalities. The suggested approach leads to function evaluations requiring to solve a relatively simple semidefinite program. This method provided exact solutions to semidefinite relaxations of the MAX-CUT problem, which was not achievable by approaches based only on interior-point methods.

- Rendl et al. [206] proposed a method for finding exact solutions to QUBO based on a semidefinite relaxation combined with a subset of triangle inequalities, which is solved using the bundle method. The expensive part of the bounding procedure is solving the basic semidefinite programming relaxation of the Max-Cut problem. As a result of this work, Rendl et al. [207] introduced the online solver called BIQMAC.
- Ibaraki et al. [147] proposed efficient heuristics to get solutions for the cycle packing problem (**NP**). They use two approaches: one is based on network algorithms and the other is based on column generation and linear programming. The implementation of [147] was developed to solve MAX-2-SAT problems, which is well known to be equivalent to QUBO.
- Barahona and L. Ladányi [34] present a branch-and-cut algorithm where the *volume algorithm* is applied to the linear programs arising at each node of the search tree. This fact results in the fast approximate solutions to these linear programs, making possible to explore many more search tree nodes and having larger LPs than if the standard dual simplex algorithm would be employed.

Other exact approaches for solving QUBO include ([33, 80, 116, 153, 185, 236]).

## 9.2 Enumerative approaches

Pardalos and Rodgers [195] Depth-First Search (DFS) approach was one of the earliest attempts at finding an optimal solution to QUBO. The fact that a simple bound has been considered by [195] as well as the adoption of DFS made possible to create a program that is able to process the various sub-problems very quickly.

Given a quadratic pseudo-Boolean function  $f$  represented as in (1.5), then the sum of the constant  $c_0$  with all the negative terms

$$N_f = c_0 + \sum_{i \in V} \min(0, c_i) + \sum_{1 \leq i < j \leq n} \min(0, c_{ij})$$

is an obvious lower bound to  $\nu(f)$ .

Given a partial assignment  $\mathbf{y} \in \mathbb{B}^S$  of  $f$ , Pardalos and Rodgers [195] bound for  $f(\mathbf{x}[S \leftarrow \mathbf{y}])$  in this assignment is

$$\begin{aligned} P_f(\mathbf{x}[S \leftarrow \mathbf{y}]) &= N_f + \sum_{i,j \in S | i < j} \max(0, c_{ij}) x_i x_j \\ &\quad + \sum_{i \in S} (\max(0, c_i) x_i - \min(0, c_i) \bar{x}_i) \\ &\quad - \sum_{1 \leq i < j \leq n | i \in S, j \notin S} \min(0, c_{ij}) \bar{x}_i \\ &\quad - \sum_{1 \leq i < j \leq n | i \notin S, j \in S} \min(0, c_{ij}) \bar{x}_j. \end{aligned} \tag{9.1}$$

When  $S = \emptyset$  then the bound of [195] coincides with  $N_f$ . However when  $S \neq \emptyset$  then  $P_f(\mathbf{x}[S \leftarrow \mathbf{y}])$  is typically strictly dominated by  $N_{f(\mathbf{x}[S \leftarrow \mathbf{y}])}$ . The reason for this fact is due to (9.1) not considering the “new” linear terms originated after fixing one variable to one on a nonzero quadratic term, as the following example illustrates.

**Example 9.1.** Consider the quadratic pseudo-Boolean function  $f(x_1, x_2) = -2x_1 + 3x_1x_2$ . It is trivial to verify that  $N_f = -2$ . If the partial assignment  $(1)^{\{2\}}$  (i.e.  $x_2 = 1$ ) is considered, then  $N_{f|_{x_2=1}} = 0$ , but  $P_f(\mathbf{x}[\{2\} \leftarrow (1)]) = -2$ .

To be able to calculate the  $N_{f(\mathbf{x}[S \leftarrow \mathbf{y}])}$  bound quickly, the DFS approach proposed here requires an additional vector structure (called  $\mathbf{v}$ ) to contain the linear coefficients of the variables not yet fixed by a partial assignment, i.e the coefficients of the linear terms that belong to the quadratic pseudo-Boolean function  $f(\mathbf{x}[S \leftarrow \mathbf{y}])$ .

Figure 9.1 describes our proposed DFS algorithm (called DEPTH-FIRST) whose bound is based on the sum of negative coefficients as has been explained previously. The process flow of the algorithm is identical to the approach proposed in [195].

For a given partial assignment the algorithm requires the knowledge of the minimum

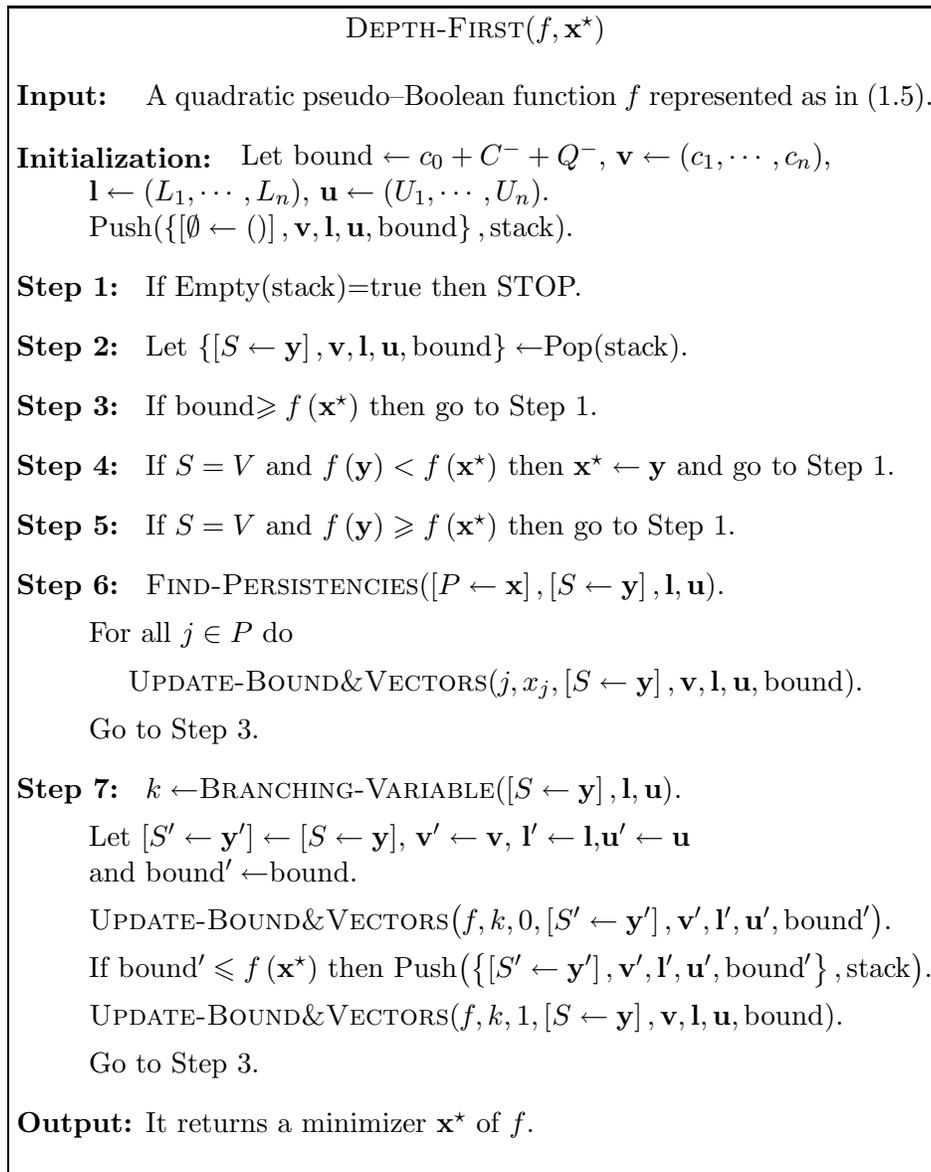


Figure 9.1: DEPTH-FIRST algorithm.

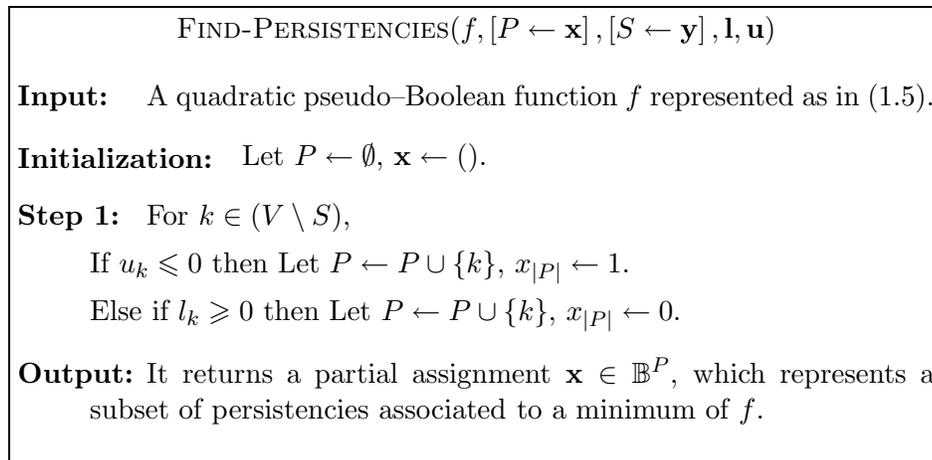


Figure 9.2: FIND-PERSISTENCIAS algorithm.

and maximum of the partial derivatives of the variables (respectively called  $\mathbf{l}$  and  $\mathbf{u}$ ) not yet fixed by the method. The  $\mathbf{l}$  and  $\mathbf{u}$  vectors are used to determine persistencies based on first derivative information (see Section 4.2). The pseudo-code of the algorithm used to find these persistencies is given in Figure 9.2.

Given an existent sub-problem (that corresponds to a partial assignment  $y_{|S|}$ ) then a branching variable  $x_k$  is selected using the same approach as that one considered by [195], i.e.  $k = \arg \max_{i \in (V \setminus S)} (-l_i, u_j)$ . We note the fact that  $l_i < 0$  and  $u_i > 0$  for all  $i \notin S$ , since otherwise they would be fixed first by persistency.

After selecting a variable  $x_k$  then 2 sub-problems are created, one corresponds to the assignment  $x_k = 0$  and the other to  $x_k = 1$ . At this point, the vectors  $\mathbf{v}$ ,  $\mathbf{l}$  and  $\mathbf{u}$ , and the lower bound associated to these two sub-problems are updated using  $n$  iterations. This procedure is specified by algorithm UPDATE-BOUND&VECTORS of Figure 9.3.

An enumerative DFS procedure based on the quadratic posiform representation has been analyzed and proposed by Hansen et al. [135]. The method starts by computing a roof-dual quadratic posiform representation of the function. Thus, at the top node the lower bound coincides with the roof-dual value. Strong persistency is also applied at the top node. Given a partial assignment, then the lower bound is updated by calculating the resulting standard posiform. The various sub-problems will have a lower bound that is typically inferior to the corresponding roof-dual. This bound is however superior to the sum of negative coefficients bound considered by the DEPTH-FIRST algorithm.

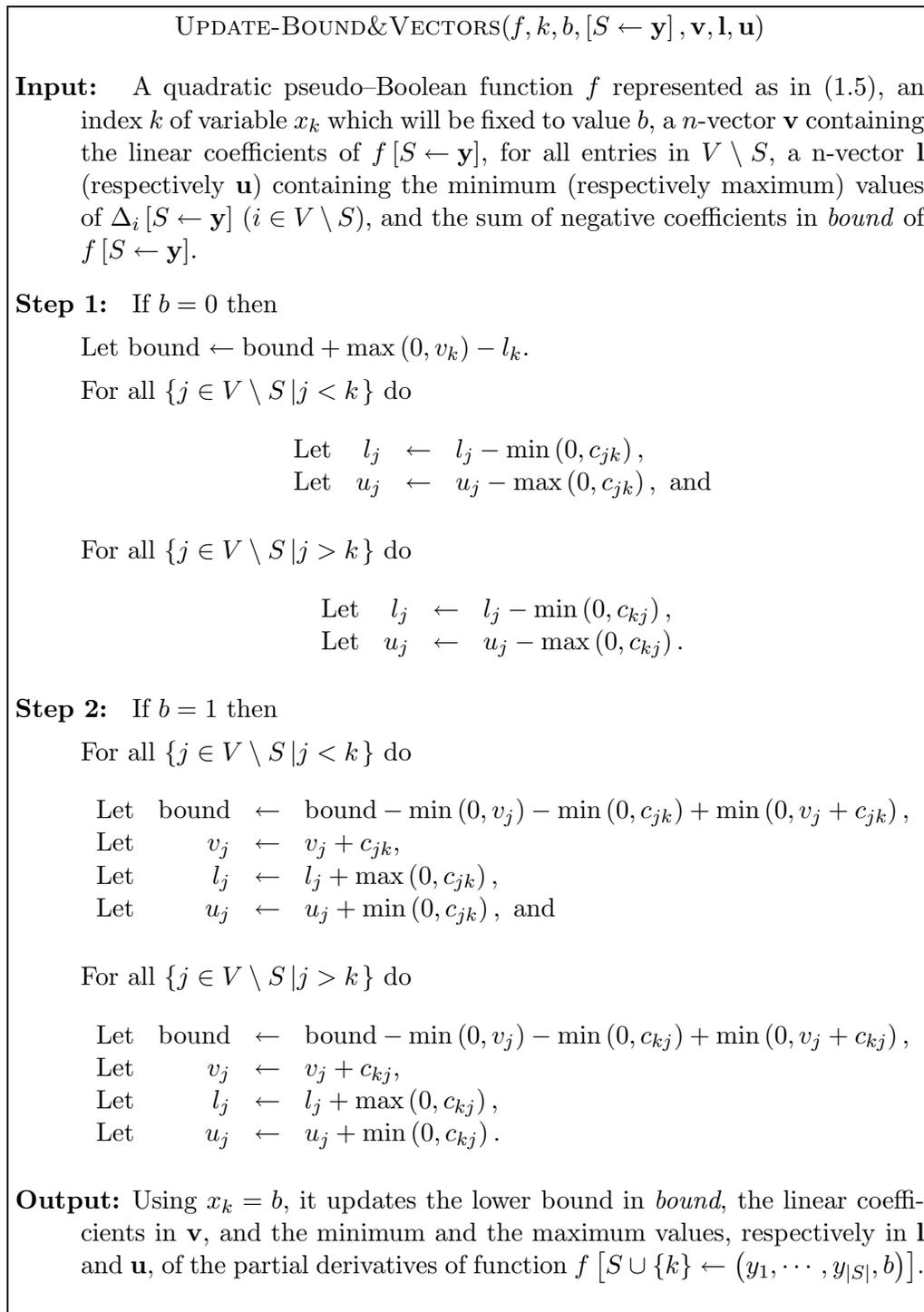


Figure 9.3: UPDATE-BOUND&amp;VECTORS algorithm.

### 9.2.1 Computational results

In the class of enumerative methods for QUBO we implemented the 3 approaches described previously. For their identification in the text, we shall call to the Pardalos and Rodgers [195] procedure as P&R–DFS, while Hansen et al. [135] method will be called as ROOF–DFS, and our proposed enumerative as DEPTH–FIRST (outlined in Figure 9.1).

Table 9.2: Computing times to find the optimum values of some  $F_2$  and  $G_2$  problems of Kochenberger et al. [158].

<i>Family</i>	<i>Problem Name</i>	<i>Density</i> ( $\bar{d}$ )	<i>Computing Time*</i>		
			P&R–DFS ([195])	ROOF–DFS ([135])	DEPTH–FIRST
F2 ( $n = 500$ )	F2a	10%	n/a	n/a	n/a
	F2b	25%	n/a	n/a	n/a
	F2c	50%	n/a	n/a	13 716.6 s
	F2d	75%	1042.8 s	554.8 s	65.8 s
	F2e	100%	19.5 s	32.8 s	2.8 s
G2 ( $n = 1000$ )	G2a	10%	n/a	n/a	n/a
	G2b	25%	n/a	n/a	n/a
	G2c	50%	n/a	n/a	n/a
	G2d	75%	n/a	n/a	16 332.3 s
	G2e	100%	766.3 s	1155.4 s	108.3 s

\*Found on a P4 2.8 GHz, 512 MB RAM and running Windows XP.

For the first time, Table 9.2 presents the optimal solutions of the denser graphs from the  $F_2$  and  $G_2$  sub–families. These families were randomly generated by Kochenberger et al. [158] and every instance is a sub–modular maximization function.

The DEPTH–FIRST enumerative approach outperforms the other two approaches in computing time. For instance, for the 500–variable instance  $F2d$  it is 8 times faster than the ROOF–DFS algorithm and 16 times faster than the original P R procedure.

Pardalos and Rodgers [195] have already claimed that P&R would exhibit in practice a time of  $O(n^3)$  for the 100% dense cases (sometimes these problems are called *Pos–Neg* instances). Here we have seen that DEPTH–FIRST is substantially faster than the P&R approach for these problems, and that it is able to solve relative large instances (up to 1 000 variables) with high density (from 75%).

To understand the power of the enumerative approaches, Table 9.3 shows some

Table 9.3: Statistics of DEPTH-FIRST to find optimal solutions of some  $F_2$  and  $G_2$  problems of Kochenberger et al. [158].

<i>Problem</i>	<i>Maximum</i>	<i>Persistencies</i>	<i>Backtrackings</i>	<i>Number of Sub-Problems To</i>	
				<i>Best Solution</i>	<i>Optimality</i>
F2c	1 094	5 223 119 119	1 268 246 250	178 793 210	242 255 974
F2d	685	33 723 443	5 693 103	880 248	894 967
F2e	418	1 429 788	160 553	16 698	17 334
G2d	866	5 772 194 364	917 769 049	109 484 508	125 967 506
G2e	452	28 443 883	2 622 488	222 229	249 794

numbers about the DEPTH-FIRST algorithm. It can be seen that the number of sub-problems created can be in the order of hundreds of millions and that all of them can be processed in a few hours. The number of persistencies and backtrackings that can be applied during this period of time is in the order of billions for these problems.

To further compare the three methods, we shall analyze the 3 enumerative approaches in the  $C$  family of QUBO problems (see Table 3.2). This family has been originally proposed and analyzed by Pardalos and Rodgers [195]. Subsequently, Hansen et al. [135] used this family to compare their approach with that one of [195].

Table 9.4(a) contains the computing times of the 3 studied approaches on these problems. The ROOF-DFS is the fastest approach, only slightly surpassed in the 4c and 5c instances. It is also evident that P&R-DFS is the slowest of the 3 methods.

Table 9.4(b) contains the number of iterations and the number of sub-problems generated during the enumerative stage. As expected, ROOF-DFS requires fewer branching steps as well as number of iterations (which includes all persistencies and backtracks), and is followed by the DEPTH-FIRST algorithm which requires substantially more nodes and iterations to solve these problems.

The results that were presented in this section illustrate that enumerative approaches can be used to effectively solve certain classes of QUBOS. In particular, (i) if the number of variables is small, (ii) if the problem is very sparse and (iii) the problem is a dense Pos-Neg instance, then these simple methods are good choices to solve these problems. These results also confirm that the fastest speed to prove optimality to QUBO (and many other optimization problems) is frequently determined by carefully

selecting what type of lower bounds to consider and the time required to computing them.

Table 9.4: Optimal solutions of the family  $C$  problems proposed by Pardalos and Rodgers [195].

(a) Computing Time.

<i>Problem Name</i>	<i>Variables (n)</i>	<i>Density (<math>\bar{d}</math>)</i>	<i>Computing Time*</i>		
			P&R-DFS ([195])	ROOF-DFS ([135])	DEPTH-FIRST
1c	40	80%	5.8 s	<b>0.2</b> s	1.3 s
2c	50	60%	95.8 s	<b>4.5</b> s	29.3 s
3c	60	40%	184.9 s	<b>6.0</b> s	46.6 s
4c	70	30%	53.0 s	42.5 s	<b>14.0</b> s
5c	80	20%	42.9 s	10.8 s	<b>10.4</b> s
6c	90	10%	0.4 s	< <b>0.05</b> s	0.06 s
7c	100	10%	0.8 s	< <b>0.05</b> s	0.06 s

\*Found on a P4 2.8 GHz, 512 MB RAM and running Windows XP.

(b) Number of iterations and sub-problems.

<i>Problem Name</i>	<i>Number of Iterations</i>			<i>Number of Sub-Problems</i>		
	P&R-DFS ([195])	ROOF-DFS ([135])	DEPTH-FIRST	P&R-DFS ([195])	ROOF-DFS ([135])	DEPTH-FIRST
1c	5 193 612	62 831	1 115 332	425 591	12 179	187 415
2c	76 952 702	1 221 654	21 762 024	4 471 714	197 103	2 845 176
3c	135 518 792	1 411 172	33 366 286	6 799 039	210 084	3 821 273
4c	36 930 288	9 001 633	9 944 560	1 744 814	995 040	846 691
5c	28 354 124	2 098 639	7 718 570	1 600 352	194 445	544 847
6c	275 918	5 060	67 822	32 527	284	5 068
7c	463 112	0	52 417	58 190	0	3 934

### 9.3 Branch-and-bound with network flows

A Branch-and-Bound (B&B) algorithm has been implemented to test the various bounds and persistent results derived in the previous chapters. Given a quadratic pseudo-Boolean function  $f$ , B&B attempts at finding the minimum value  $\nu(f)$  and a minimizer  $\mathbf{x}^*$ .

Along its execution, B&B maintains a list of active nodes  $\mathcal{A}$ . Each active node is a QUBO problem itself represented as a capacitated network (see Section 5.3), which is associated to a quadratic function derived from a partial assignment of  $f$ . Initially

$\mathcal{A} = \{f\}$ .

At any point in time, B&B records information about a lower ( $LB$ ) and an upper ( $UB$ ) bound to  $\nu(f)$ . Clearly, the minimum lower bound of any active node is a lower bound to  $\nu(f)$ .

B&B executes the following steps to process an active node:

- PREPROCESS – Apply roof-duality and remove weak and strong persistencies; Apply any other tools specified by the user (e.g. probing, one-pass heuristics, 2nd order derivatives); If there is a decomposition of the function (see Section 7.2.3), then B&B optimizes separately each component.
- CHECK-BOUNDS – If a node has a roof-dual bound worse than the upper bound  $UB$  then this node can be cutoff and  $B\&B$  proceeds to processing the next active node;
- UPDATE-LOWER-BOUND – An enhanced lower bound (e.g., SDP, IRDA, SIRDA, PLIRDA) specified by the user is computed. The bounds are checked again for possible pruning;
- UPDATE-UPPER-BOUND – If the problem of the active node is simply a constant then the  $UB$  and  $\mathbf{x}^*$  are updated accordingly;
- BRANCH – If the problem of the active node is still non-trivial then a variable  $y$  is selected for branching and the current active node is replaced by two new active nodes, one having an assignment  $y = 0$  and the other one having an assignment  $y = 1$ .

In this implementation we did not consider strong branching at its full extent, i.e. the possibility of testing a certain number of partial assignments to increase the chances of finding a branch that is more likely to produce fewer nodes in the future. We remark that our preprocessing code considers probing (optionally) and therefore we can easily get the 1-level branching information and incorporate it in the branching decisions.

We remark that after preprocessing every node (even at its basic usage) every remaining variable must have partial derivative that range between the negative and

positive values. Since, our representation of choice is the network model  $G_\phi$ , or equivalently a quadratic posiform  $\phi$ , then after preprocessing every literal  $u \in \mathbf{L}$  belongs to the same strong component as its complement  $\bar{u}$ . The *purely* quadratic posiform  $\phi$  can be decomposed into three parts

$$\phi = x_i(\alpha_1 u_1 + \cdots + \alpha_r u_r) + \bar{x}_i(\beta_1 v_1 + \cdots + \beta_s v_s) + \Psi_i,$$

for all  $i = 1, \dots, n$ .

To reduce the complexity of the branches as much as possible, B&B estimates the contribution of each branching  $\mathbf{b} = (b_1, \dots, b_m)$  by computing a positive real number  $\pi(\mathbf{b})$ , which is the real root  $p$  of the corresponding “tree polynomial”  $\sum_{i=1}^s p^{-b_i} = 1$ . B&B considers only 2-branches (i.e.  $m = 2$ ), one for the assignment 0 and the other for the assignment 1 of the variable selected. The branching estimates that we consider for  $x_i$  ( $i = 1, \dots, n$ ) is  $\mathbf{b} = \left( \sum_{j=1}^r \alpha_j, \sum_{j=1}^s \beta_j \right)$ . The intent of this branching strategy is to improve the lower bound as quickly as possible.

The estimation based on the tree polynomial roots for a branching purpose is not new and has been considered previously to solve satisfiability problems (see e.g. [163]). In these other studies, the branching estimates usually include information about the number of persistencies obtained on a given branching. We did not consider this additional piece of information, since we did not apply strong branching to be able to obtain a good measure about the number of persistencies that certain branches would infer. The root of the tree polynomial can be easily determined by using the Newton’s approximation method. In practice, only 5 iterations of the Newton method are required to compute the root with a good numerical precision.

To hold the active nodes, B&B uses a map structure whose keys are integers representing the closest integer from above to the lower bound of the optimization problems belonging to its elements. Every element of the map is a list of QUBOs. The first element of the map includes the problems having the smallest lower bounds, and the last element of the map contains the problems having the largest lower bounds. In this way, B&B can quickly select a node either with a small or a large lower bound to the

minimum.

Indeed, the code supports 3 possible node selections: (i) selects the sub-problem with the *smallest* lower bound, and if more than one is found then select the one with the *fewest* variables; (ii) select the sub-problem with the *largest* lower bound, and if more than one is found then select the node with the *fewest* variables; and (iii) uses (i) if the number of active nodes is below a certain limit (provided) by the user, otherwise uses (ii). Basically strategy (i) attempts at raising the bound quickly, (ii) is best at finding initial solutions and at not increasing the number of nodes substantially, and (iii) is a mix between the other 2 approaches that attempts at raising the bound quickly, while not using excessively the memory resource. In the computational experiments that follow we will always consider option (iii) with a “limit” number selected by the user.

In QUBO there is no reason to not apply any heuristic or meta-heuristic to provide a “good” incumbent to the exact approach. In the B&B implementation, the preprocessing PREPRO routine will compute one-pass heuristics (see Section 6.1) at certain stages and frequencies determined by the user.

### 9.3.1 Computational results

The analyzes of the experiments that we have carried out with the B&B exact solver will be focused on the impact of using the various lower bounds that we have proposed in Chapter 8. The roof-dual bound (and persistency) determined by the RDA algorithm is always applied by default. A strengthened bound can then be applied at certain or all nodes. The list of improved lower bounds include six algorithms: IRDA, SIRDA (we test it with 1 and 2 rounds), PLIRDA (with test it with 1 and 2 rounds) and SDP.

B&B has been implemented in C++ and linked using the Windows libraries. The following sub-sections illustrates the application of B&B to different types of QUBO functions and applications.

#### 9.3.1.1 MIN-VC of planar graphs

We start by providing practical evidence of two facts.

First, from the various tools and options that the user has at hand, the one choice selected on B&B can be determinant in solving the problem at hand in the following ways: very quickly, very slowly or even to not solving it at all. In this section we will see how the various tools available in PREPRO could affect the speed and quality of the end results.

Second, decomposition is something that occurs frequently on certain classes of problems (e.g., MIN-VC of planar graphs). B&B detects this decomposition of the problem and solves them independently using result (7.5).

Table 9.5: Using B&B to find minimum vertex covers of planar graphs.

		<i>B&amp;B Computing Time*</i> ( <i>B&amp;B Nodes</i> )		
<i>Strategy</i>	<i>Preprocessing Tools</i>	10 000 <i>vertices</i> 26 994 <i>edges</i>	20 000 <i>vertices</i> 53 994 <i>edges</i>	50 000 <i>vertices</i> 134 994 <i>edges</i>
<b>I</b>	roof-dual & decomposition	6.2 s (1 974 <i>nodes</i> )	18.0 s (3 986 <i>nodes</i> )	145.9 s (9 997 <i>nodes</i> )
<b>II</b>	I+probing (no weak persist.)	6.7 s (1 160 <i>nodes</i> )	19.2 s (2 232 <i>nodes</i> )	146.2 s (5 560 <i>nodes</i> )
<b>III</b>	I+probing (with weak persist.)	3.4 s (25 <i>nodes</i> )	12.4 s (29 <i>nodes</i> )	126.5 s (85 <i>nodes</i> )
<b>IV</b>	III & coordinance	3.2 s (1 <i>node</i> )	11.7 s (1 <i>node</i> )	125.3 s (1 <i>node</i> )

\*Computed on computer system I (see Table 8.1).

Table 9.5 presents results of B&B to find minimum vertex covers of planar graphs (randomly generated using RUDY). Alternative ways of preprocessing are considered by B&B. It is evident that the choice of the preprocessing strategy is determinant in terms of solving the problem faster and also in terms of the number of branching nodes required. It is interesting that strategy IV can solve the example graph problem without any branching.

### 9.3.1.2 Benchmarks with prescribed density

In this section we analyze the application of B&B to solve some QUBO instances randomly generated in such a way that all variables participate in about the same of number of quadratic terms. These instances were previously called as problems of fixed prescribed density (see Section 3.1.2).

We shall focus on the family  $D$  proposed by Glover et al. [108] whose instances have 100 variables per problem, and densities varying from 10% to 100%. These problems are maximization problems and were previously considered for testing various upper bounds (see Tables 8.3(a), 8.5(a) and 8.7(a)).

Table 9.6: Proving optimality to Beasley [37] QUBO problems with 250 variables.

(a) Relative gap to the maximum.

		<i>B&amp;B Relative Gap to the Maximum after Running B&amp;B for 1 Hour*</i>					
		<i>SDR</i>	<i>IRDA</i>	<i>SIRDA</i>		<i>PLIRDA</i>	
<i>Problem Name</i>	<i>Density (d)</i>	<i>Semidefinite Relaxation</i>	<i>Iter. Roof Dual (<math>\hat{\rho}</math>)</i>	<i>Squeezed Iterated Roof</i>		<i>P&amp;L Iterated Roof</i>	
				$(\tilde{\rho}; r = 1)$	$(\tilde{\rho}; r = 2)$	$(\tilde{\rho}; r = 1)$	$(\tilde{\rho}; r = 2)$
1d	10%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
2d	20%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
3d	30%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
4d	40%	0.0%	1.5%	2.1%	0.8%	3.9%	2.0%

\*Computed on computer system I (see Table 8.1).

(b) Computing time to optimality.

		<i>B&amp;B Computing Time* to Optimality</i>					
		<i>SDR</i>	<i>IRDA</i>	<i>SIRDA</i>		<i>PLIRDA</i>	
<i>Problem Name</i>	<i>Density (d)</i>	<i>Semidefinite Relaxation</i>	<i>Iter. Roof Dual (<math>\hat{\rho}</math>)</i>	<i>Squeezed Iterated Roof</i>		<i>P&amp;L Iterated Roof</i>	
				$(\tilde{\rho}; r = 1)$	$(\tilde{\rho}; r = 2)$	$(\tilde{\rho}; r = 1)$	$(\tilde{\rho}; r = 2)$
1d	10%	4.4 s	0.1 s	1.6 s	2.2 s	3.1 s	5.2 s
2d	20%	1 194.0 s	122.1 s	261.0 s	324.2 s	1 079.6 s	1 067.2 s
3d	30%	1 334.6 s	369.3 s	815.9 s	871.6 s	6 329.6 s	3 969.3 s
4d	40%	1 482.8 s	>7 200.0 s	>7 200.0 s	>7 200.0 s	>7 200.0 s	>7 200.0 s

\*Computed on computer system I (see Table 8.1).

(c) Branching nodes.

		<i>B&amp;B Nodes</i>					
		<i>SDR</i>	<i>IRDA</i>	<i>SIRDA</i>		<i>PLIRDA</i>	
<i>Problem Name</i>	<i>Density (d)</i>	<i>Semidefinite Relaxation</i>	<i>Iter. Roof Dual (<math>\hat{\rho}</math>)</i>	<i>Squeezed Iterated Roof</i>		<i>P&amp;L Iterated Roof</i>	
				$(\tilde{\rho}; r = 1)$	$(\tilde{\rho}; r = 2)$	$(\tilde{\rho}; r = 1)$	$(\tilde{\rho}; r = 2)$
1d	10%	24	10	10	10	10	9
2d	20%	2 607	1 666	247	127	333	166
3d	30%	2 695	2 940	700	320	1 342	481
4d	40%	2 711	16 566	3 455	1 757	1 618	821

Table 9.6(a) shows that B&B can solve problems in this family having up to 30% density. If the iterated roof–dual algorithm is used for bounding (using IRDA) then the 30% dense instance takes about 5 minutes to solve to optimality. The other bounds

seem to not be so effective, but all of them were able to solve these 3 instances.

The number of branching nodes gives an idea about the scalability of the exact approaches. As the size of the problem increases then the number of nodes can become exponentially large in the number of variables. A given bound can be computed very quickly, but the number of nodes required to achieve a certain value may be very large, and hence resulting in a total non-polynomial computing time.

It is interesting to note that the squeezed and project-and-lift iterated roof-duality versions are clearly superior to the simple iterated roof-duality and the semidefinite bounds (see Table 9.6(c)) for instances having up to 30% density.

The iterated roof-dual bounds are clearly superior to the semidefinite relaxation bound in speed performance for those problems that are “sparse”.

The results of Table 9.6(b) clearly indicate that B&B with the SDP bound is not highly dependent on the density parameter. Consequently, to be able to find the optimum of QUBOs of very large density and having up to 200 or so variables it is strongly recommended to use exact approaches based on semidefinite relaxations. This fact has been proposed and studied by Billionet and Elloumi [41, 43] for the type of QUBOs analyzed in this section. B&B with SDR was able to prove optimality for all the 10 instances from family  $D$  in a reasonable amount of time. B&B with IRDA was able to prove optimality for the instances having up to 40% density, the longest case requiring slightly over two hours of computing time.

Rendl et al. [208] proposed the use of a semidefinite relaxation combined with a subset of the triangle inequalities, which is then solved with the bundle method. An implementation of this algorithm is publicly accessible through the Internet ([207]) and is called the BIQMAC solver. Clearly this algorithm is trying to get the best out of the iterated roof-duality (which is indirectly associated to a subset of triangle inequalities) and the semidefinite approaches. The results of BIQMAC are very impressive for dense QUBOs of up to 200 or so variables, but the computation of the semidefinite bound makes it impracticable to solve QUBOs with many hundreds and especially thousands of variables (even if they are sparse).

The next family analyzed consists of 10 problems having 250 variables and 10%

densities. This family (called as B-250) was proposed by Beasley [37] (see Section 3.1.1). All the problems in family B-250 have a maximum value known (see Rendl et al. [208]).

The *relative gap to the maximum* value of these problems is the ratio between the *absolute gap* (equal to the difference between the B&B upper bound and the optimum) and the optimum. Table 9.7(a) lists the relative gaps of the 10 problems computed by B&B with various upper bounds (SDR, IRDA, SIRDA), after 1 hour of solve time. SDR provides the best relative gaps for this family varying between 3.6% and 9.8%. If IRDA is considered instead, then the relative gap varies between 6.2% and 21.2%. SIRDA with the 1 round option returns relative gaps varying between 3.6% and 15.8%. Except for instances 3 and 5, any of the implemented approaches analyzed here, requires a substantial large amount of computing time to prove optimality for problems belonging to this family (see also Section 9.4.1.1).

We close this section by suggesting to the interested researchers that they should focus in improving the existing implementations of the improved iterated roof-duality bounds. This work can be achieved by using improved algorithms (e.g. new theory and max-flow implementations) or by applying improved or customized data structures. These improvements will allow the exact approaches to process far more nodes than before, and hence, the chances of solving harder sparse QUBOs, previously unsolved, will increase substantially.

### 9.3.1.3 MAX-2-SAT

The application of B&B to solve (weighted) MAX-2-SAT problems is presented in this section. Two families of problems are considered in the experiments.

First we tested the code in the Borchers and Furman [47] instances. The results are shown in Tables 9.8(a) and 9.8(b). The B&B implementation is compared to the MaxSolver solver of Xing and Zhang [236], and to the solvers (BB\_C, BB\_P, BB\_HP) developed by Ibaraki et al. [148]. The solvers of [148] are methods based on packing cycles on the network (BB\_C) or LP (BB\_P and BB\_HP) models. Therefore the bounds considered by [148] are closely related to those considered by B&B.

Table 9.7: Proving optimality to Beasley [37] QUBO problems with 250 variables.

(a) Relative gap to the maximum.

		<i>B&amp;B Relative Gap to the Maximum after Running B&amp;B for 1 Hour*</i>			
		<i>SDR</i>	<i>IRDA</i>	<i>SIRDA</i>	
<i>Problem Name</i>	<i>Density (d)</i>	<i>Semidefinite Relaxation</i>	<i>Iter. Roof Dual (<math>\hat{\rho}</math>)</i>	<i>Squeezed Iterated Roof</i>	
				$(\bar{\rho}; r = 1)$	$(\bar{\rho}; r = 2)$
1	10%	4.9%	8.7%	5.8%	5.3%
2		5.0%	11.0%	7.6%	n/a
3		3.6%	6.2%	3.6%	3.1%
4		5.3%	11.7%	8.2%	7.5%
5		3.6%	6.2%	3.8%	3.2%
6		6.6%	15.5%	11.1%	10.8%
7		4.3%	9.1%	6.0%	5.2%
8		9.8%	21.2%	15.8%	n/a
9		4.7%	8.2%	6.0%	5.4%
10		6.6%	14.3%	10.1%	9.5%

\*Dual Xeon 3.0 GHz, 4GB of RAM and running XP.

(b) Branching nodes.

		<i>B&amp;B Nodes</i>			
		<i>SDR</i>	<i>IRDA</i>	<i>SIRDA</i>	
<i>Problem Name</i>	<i>Density (d)</i>	<i>Semidefinite Relaxation</i>	<i>Iter. Roof Dual (<math>\hat{\rho}</math>)</i>	<i>Squeezed Iterated Roof</i>	
				$(\bar{\rho}; r = 1)$	$(\bar{\rho}; r = 2)$
1	10%	423	5 825	599	189
2		435	5 945	621	n/a
3		417	5 846	655	231
4		419	5 624	591	175
5		411	5 646	607	205
6		433	5 482	615	171
7		415	5 799	599	189
8		451	6 087	631	n/a
9		417	5 991	621	203
10		447	5 711	611	173

For similar formulas, the solver that should be used to find the fastest proofs is algorithm BB\_C. B&B with the IRDA bound produces optimal solutions for this family in reasonable time.

The B&B with the squeezed or P&L iterated roof-duality based bounds is the method producing the fewer branching nodes to prove optimality, thus indicating that most likely these approaches are computing bounds closer to  $C_3$  than those proposed by Ibaraki et al. [148]. As in the previous section conclusions, these results indicate that the improvement of the SIRDA and PLIRDA algorithms will make B&B competitive (if not faster) than the other solvers for many “larger instances”.

B&B with the IRDA bound is comparable to the BB\_C method in the way the bounds are determined. A key difference is the fact that IRDA considers the bi-form representation whereas BB\_C does not. Maybe in part because of this reason (others may be preprocessing), it is interesting to note that the B&B version requires substantially fewer branching nodes (about half less nodes) to solve these MAX-2-SAT instances.

The second type of MAX-2-SAT formulas that we have analyzed are based on the break minimization problems created by Ibaraki et al. [148]. These problems arise from sports scheduling. Given a set of teams the problem is to determine a schedule that minimizes the total consecutive home or away games, both of which are called *breaks*. The MAX-2-SAT formulation requires  $\binom{n}{2}$  variables and  $n(n-2)$  clauses, where  $n$  is the number of teams. Every literal appears at most twice in the formulas.

The trend of results given in Tables 9.9(a) and 9.9(b) for certain break minimization instances is somewhat similar to the ones of the previous family. The linear programming based approaches BB\_P and BB\_HP are the fastest implementations in this case.

B&B with SIRDA with 2 rounds produces overall the least number of nodes (on average). The number of branching nodes processed by B&B with IRDA is many times smaller than those required by BB\_C. In this group of problems the difference is in the ten-fold order of magnitude.

Table 9.8: Proving optimality to the Borchers and Furman [47] MAX-2-SAT instances.

(a) Computing time.

Sub-Family	Variables ( $n$ )	Clauses $A(\phi)$	<i>B&amp;B with IRDA*</i>	<i>B&amp;B with SIRDA*</i>	<i>B&amp;B with PLIRDA*</i>		<i>Other Computing Times</i>				
			Iter. Roof Dual ( $\hat{\rho}$ )	Squeezed Iterated Roof		P&L Iterated Roof		MaxSolver**	BB_C†	BB_P†	BB_HP†
				( $\bar{\rho}; r = 1$ )	( $\bar{\rho}; r = 2$ )	( $\tilde{\rho}; r = 1$ )	( $\tilde{\rho}; r = 2$ )	([236])	Ibaraki et al. [148]		
BF-100	100	400	1.5 s	5.2 s	8.4 s	10.2 s	25.3 s	<b>0.3</b> s	0.4 s	1.4 s	0.6 s
		500	18.9 s	48.3 s	62.9 s	106.0 s	116.0 s	11.8 s	<b>1.1</b> s	6.0 s	3.5 s
		600	78.7 s	194.0 s	254.4 s	348.2 s	382.9 s	106.2 s	<b>3.4</b> s	36.8 s	9.0 s
BF-150	150	300	< <b>0.05</b> s	0.1 s	0.1 s	0.5 s	0.6 s	0.1 s	0.3 s	0.3 s	0.3 s
		450	3.1 s	9.8 s	12.7 s	12.9 s	22.1 s	1.9 s	<b>0.5</b> s	2.4 s	0.8 s
		600	14.6 s	41.5 s	54.1 s	72.9 s	135.4 s	10.4 s	<b>0.8</b> s	4.5 s	2.2 s
BFW-100	100	400	1.9 s	10.2 s	15.8 s	21.3 s	37.0 s	6.9 s	<b>0.5</b> s	1.8 s	1.0 s
		500	10.4 s	27.6 s	29.6 s	43.0 s	66.0 s	532.4 s	<b>0.8</b> s	7.1 s	1.4 s
		600	5.6 s	17.9 s	22.7 s	56.2 s	68.4 s	289.8 s	<b>0.9</b> s	7.9 s	4.9 s
BFW-150	150	300	<b>0.1</b> s	0.8 s	0.8 s	0.8 s	1.5 s	0.2 s	0.3 s	0.3 s	0.3 s
		450	1.0 s	5.0 s	10.3 s	13.5 s	25.0 s	53.5 s	<b>0.5</b> s	1.7 s	0.5 s
		600	17.2 s	103.0 s	129.0 s	186.2 s	276.3 s	3 527.5 s	<b>1.3</b> s	9.7 s	3.6 s

\*Computed on computer system I (see Table 8.1).

\*\*Computed on Pentium 2.4 GHz with 1 GB memory.

†Computed on Xeon (NetBurst) 3.06 GHz, 1 GB memory, 32-bit mode.

(b) Branching nodes.

Sub-Family	Variables ( $n$ )	Clauses $A(\phi)$	<i>Branching Nodes</i>							
			<i>B&amp;B with IRDA</i>	<i>B&amp;B with SIRDA</i>	<i>B&amp;B with PLIRDA</i>		<i>Other Solvers</i>			
			Iter. Roof Dual ( $\hat{\rho}$ )	Squeezed Iterated Roof		P&L Iterated Roof		BB_C	BB_P	BB_HP
( $\bar{\rho}; r = 1$ )	( $\bar{\rho}; r = 2$ )	( $\tilde{\rho}; r = 1$ )		( $\tilde{\rho}; r = 2$ )	Ibaraki et al. [148]					
BF-100	100	400	98	54	25	21	22	284	254	257
		500	1024	172	72	102	52	2 185	255	1 427
		600	2770	519	205	270	146	7 891	295	6 571
BF-150	150	300	6	6	5	5	5	11	11	5
		450	144	25	16	13	13	339	255	63
		600	545	69	40	37	28	813	255	25
BFW-100	100	400	108	29	23	34	25	333	255	57
		500	333	56	31	42	32	581	255	411
		600	179	35	25	42	35	479	255	51
BFW-150	150	300	12	7	6	6	6	47	47	11
		450	52	15	15	17	15	277	255	49
		600	582	105	51	75	53	857	255	673

Table 9.9: Proving optimality to the Ibaraki et al. [148] break minimization (MAX-2-SAT) problems.

(a) Computing time.

Variables ( $n$ )	Clauses $A(\phi)$	<i>B&amp;B with IRDA*</i>	<i>B&amp;B with SIRDA*</i>		<i>B&amp;B with PLIRDA*</i>		<i>Other Computing Times</i>		
		<i>Iter. Roof</i> <i>Dual</i> ( $\hat{\rho}$ )	<i>Squeezed Iterated Roof</i> ( $\tilde{\rho}; r = 1$ )   ( $\tilde{\rho}; r = 2$ )		<i>P&amp;L Iterated Roof</i> ( $\tilde{\rho}; r = 1$ )   ( $\tilde{\rho}; r = 2$ )		<i>BB_C</i> <sup>†</sup>	<i>BB_P</i> <sup>†</sup>	<i>BB_HP</i> <sup>†</sup>
							Ibaraki et al. [148]		
120	224	2.2 s	14.8 s	20.0 s	4.2 s	4.7 s	0.7 s	1.7 s	<b>0.7</b> s
153	288	14.1 s	55.3 s	78.5 s	36.4 s	24.6 s	4.7 s	3.3 s	<b>1.3</b> s
190	360	91.0 s	146.0 s	136.4 s	931.5 s	110.8 s	47.2 s	<b>11.0</b> s	14.4 s
231	440	4 057.3 s	930.2 s	885.9 s	2 836.6 s	337.3 s	452.0 s	25.7 s	<b>6.6</b> s

\*Computed on computer system I (see Table 8.1).

<sup>†</sup>Computed on Xeon (NetBurst) 3.06 GHz, 1 GB memory, 32-bit mode.

(b) Branching nodes.

		<i>Branching Nodes</i>							
Variables ( $n$ )	Clauses $A(\phi)$	<i>B&amp;B with IRDA</i>	<i>B&amp;B with SIRDA</i>		<i>B&amp;B with PLIRDA</i>		<i>Other Solvers</i>		
		<i>Iter. Roof</i> <i>Dual</i> ( $\hat{\rho}$ )	<i>Squeezed Iterated Roof</i> ( $\tilde{\rho}; r = 1$ )   ( $\tilde{\rho}; r = 2$ )		<i>P&amp;L Iterated Roof</i> ( $\tilde{\rho}; r = 1$ )   ( $\tilde{\rho}; r = 2$ )		<i>BB_C</i>	<i>BB_P</i>	<i>BB_HP</i>
							Ibaraki et al. [148]		
120	224	356	36	28	92	71	3 225	255	25
153	288	1 391	66	64	322	156	22 477	255	25
190	360	4 038	82	51	2 708	157	169 037	255	3 467
231	440	110 377	276	132	3 858	228	1 102 333	255	25

### 9.3.1.4 MAX-CUT

This section will show some results concerning the MAX-CUT problem. The graphs considered here were proposed by Resende [209], and are relative sparse instances.

Table 9.10: Proving optimality to some Resende [209] MAX-CUT problems.

Graphs			Upper Bounds				B&B with IRDA	
			Value		Computing Time		Nodes	Time*
Vertices	Edges	CUT	SDR	IRDA	SDR <sup>†</sup>	IRDA*		
10	26	17	18.17	20.00	0.01 s	0.00 s	5	0.03 s
20	47	37	38.48	39.00	0.04 s	0.00 s	5	0.01 s
25	51	42	43.04	43.00	0.05 s	0.00 s	3	0.02 s
30	77	61	63.22	63.00	0.06 s	0.00 s	8	0.05 s
50	131	105	109.99	109.75	0.21 s	0.02 s	18	0.11 s
100	269	214	226.16	227.50	1.00 s	0.02 s	2 006	24.25 s
150	355	294	308.63	309.50	3.38 s	0.05 s	2 017	51.96 s
200	495	405	427.30	436.32	7.95 s	0.06 s	114 402	5 161.98 s
250	331	305	317.26	310.25	17.81 s	0.03 s	165	3.62 s
500	625	574	598.15	590.50	276.48 s	0.11 s	39 794	2 946.03 s

\*Computed on computer system I (see Table 8.1).

<sup>†</sup>Computed on an Alpha CPU 21264 500MHz running Linux.

A comparative analysis between the IRDA and SDP bounds is given in Table 9.10. Traditionally the semidefinite bounds (due to its excellent performance guarantees) has been selected as the bound of choice to attach the optimal solution for MAX-CUT. The results (both in value and time) are indicative that the iterated roof-duality versions should be not disregarded especially if the graphs are very sparse or simply ultra-sparse (see Section 8.7.1), even in the case where the number of vertices is large. B&B with IRDA could handle the graph having 500 vertices in less than one hour of computing time.

### 9.3.1.5 One dimensional Ising chains

The *one dimensional Ising chain* consists of  $n$  spins lying equally spaced on a circle of perimeter  $n$ . Every pair of spins  $(i, j)$  is connected with each other by a coupling strength of

$$J_{i,j} = \frac{\epsilon_{i,j}}{r_{i,j}^\sigma},$$

where  $\epsilon_{i,j}$  is chosen according to a standard Gaussian distribution,  $r_{i,j}$  is the Euclidean distance between nodes  $i$  and  $j$  and  $\sigma$  is the power of strengthness.

The exact ground states of the spins  $S_i \in \{-1, 1\}$  ( $i = 1, \dots, n$ ) are found by minimizing the Hamiltonian

$$H = - \sum_{i=1}^{n-1} \sum_{j=i+1}^n J_{i,j} S_i S_j. \quad (9.2)$$

Making the substitutions  $S_i = 2x_i - 1$  ( $i = 1, \dots, n$ ) in (9.2), then the minimum energy state can be found by solving the QUBO problem

$$\max_{\mathbf{x} \in \mathbb{B}^n} (-H(x_1, \dots, x_n)) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n J_{i,j} S_i S_j + \max_{\mathbf{x} \in \mathbb{B}^n} \sum_{i=1}^{n-1} \sum_{j=i+1}^n J_{i,j} (-2J_{i,j}) (x_i \bar{x}_j + \bar{x}_i x_j). \quad (9.3)$$

From (9.3) it is simple to see that the energy minimization problem can be formulated as a MAX-CUT problem of a dense graph having  $L$  vertices and a weight  $-2J_{i,j}$  for every edge  $(i, j)$ .

Katzgraber and Young [156] indicate that the one-dimensional Ising chain model inherits several features that makes the study of its physical properties interesting (see also [172]).

Table 9.11 gives three methods and corresponding computing times necessary to find the minimum energy state of several one-dimensional Ising chains proposed by Rendl et al. [206, 208]. The 3 exact approaches analyzed here include a branch-and-cut model proposed by Liers [172], the BIQMAC solver developed by Rendl et al. [206, 208] and B&B. B&B was configured to use roof-duality as a bound and more important to consider the preprocessing routine PREPRO with the probing option. Branch-and-cut is traditionally one of the best approaches to solve the classical Ising models ([223]), however B&B is clearly faster over the other two methods. For example, for the hardest instance ( $\sigma = 2.5$  and  $n = 300$ ) branch-and-cut does not have a known solve time and the proposed method is about than 2 000 times faster than the BIQMAC solver.

The key to solve these Ising problems (for these  $\sigma$  values) is the application of the preprocessing method with the probing option (see Chapter 7). From the Ising

Table 9.11: Computing times to find the minimum energy state of the one-dimensional Ising chains proposed by Rendl et al. [206, 208].

$\sigma$	Number of Spins ( $n$ )	Problem Instance	MAX-CUT ( $W(S, \bar{S})$ )	Computing Time		
				Branch-Cut & Price* ([172])	BIQMAC** ([206, 208])	B&B PREPRO +probing**
2.5	100	5555	2 460 049	1 102 s	92 s	1.4 s
		6666	2 031 217	387 s	66 s	1.4 s
		7777	3 363 230	608 s	47 s	0.9 s
	150	5555	4 363 532	77 319 s	265 s	3.9 s
		6666	4 057 153	84 911 s	339 s	3.3 s
		7777	4 243 269	114 007 s	559 s	3.3 s
	200	5555	6 294 701	n/a	605 s	8.8 s
		6666	6 795 365	n/a	1 075 s	8.6 s
		7777	5 568 272	n/a	1 298 s	8.6 s
	250	5555	7 919 449	n/a	10 828 s	12.7 s
		6666	6 925 717	n/a	4 624 s	13.9 s
		7777	6 596 797	n/a	4 250 s	16.0 s
	300	5555	8 579 363	n/a	24 227 s	23.6 s
		6666	9 102 033	n/a	32 678 s	17.8 s
		7777	8 323 804	n/a	46 810 s	20.7 s
3.0	100	5555	2 448 189	292 s	96 s	0.8 s
		6666	1 984 099	24 s	34 s	0.6 s
		7777	3 335 814	451 s	48 s	0.5 s
	150	5555	4 279 261	9 406 s	278 s	1.7 s
		6666	3 949 317	17 345 s	235 s	1.8 s
		7777	4 211 158	13 721 s	366 s	1.7 s
	200	5555	6 215 531	33 723 s	607 s	3.1 s
		6666	6 756 263	118 083 s	1 133 s	2.9 s
		7777	5 560 824	32 006 s	1 362 s	3.0 s
	250	5555	7 823 791	76 627 s	6 389 s	3.8 s
		6666	6 903 351	27 745 s	949 s	3.7 s
		7777	6 418 276	63 013 s	3 444 s	4.7 s
	300	5555	8 493 173	62 454 s	8 414 s	4.3 s
		6666	8 915 110	37 300 s	5 542 s	4.8 s
		7777	8 242 904	66 829 s	11 533 s	5.1 s

\*1.8 GHz computer ([206])

\*\*Pentium IV, 3.6 GHz

instances listed in Table 9.11 only one instance ( $\sigma = 2.5$ ,  $n = 150$ , instance 7777) could not be entirely solved by PREPRO, our proposed preprocessing routine for QUBOs. The residual QUBO problem of this particular instance consists of 7 variables, thus even making possible to solve it by simply enumerating the  $2^7$  possible values of the 7 unknowns.

Table 9.12: Average computing times to find the minimum energy state of larger one-dimensional Ising chains.

$\sigma$	Number of Spins ( $n$ )	Average over 3 Graphs	
		Variables Left After PREPRO	B&B with PREPRO +probing*
2.5	500	5	13 s
	1 000	24	53 s
	1 500	32	124 s
	5 000	54	1 941 s
3.0	500	0	4 s
	1 000	0	23 s
	1 500	0	59 s
	5 000	0	1 248 s

\*Pentium M 1.6 GHz, 760MB RAM.

To investigate the scalability of our method we tested it on larger instances having the same  $\sigma$  values as before, but having up to 5 000 spins. PREPRO with probing could solve to optimality all cases having  $\sigma = 3.0$ . PREPRO with probing returns some residual QUBOs for the other case, the largest residual having 54 variables. These results show that most likely for smaller values of  $\sigma$  our approach will encounter difficulties. This simply means that as power-law component becomes less effective our method will take longer to find the true ground states. For example, some experiments on randomly generated instances with  $\sigma = 2.0$  indicate that PREPRO reduces the size of the problem to only about half of the number of spins.

#### 9.4 Linearization enhanced with logical cuts

The last exact approach for QUBO relies on the power of the LP and MIP technology. This power has origin in two key factors.

First, linear programming is able to determine the various bounds considered in this work, and especially the  $C_3$  bound. LP is the only known polynomial time algorithm

to compute  $C_3$  ([50]). The  $C_3$  bound is computed by using the standard linearization for QUBO (8.1) plus a set of triangle inequalities (see (8.2)). In Chapter 8 we have also analyzed several lower bounds which include certain subsets of the triangle inequalities (see Section 8.7). These bounds could be computed very efficiently and are of very good quality for many “sparse” benchmarks.

Second, the computer hardware and the LP and MIP technology (of the state-of-the-art solvers) has evolved to a state where previously unprovable problems can now be handled by the MIP technology (see e.g. [168]).

As we shall see in practice, these two factors can contribute to solving many of the “sparse” benchmarks and it especially scales well for certain families of QUBOs.

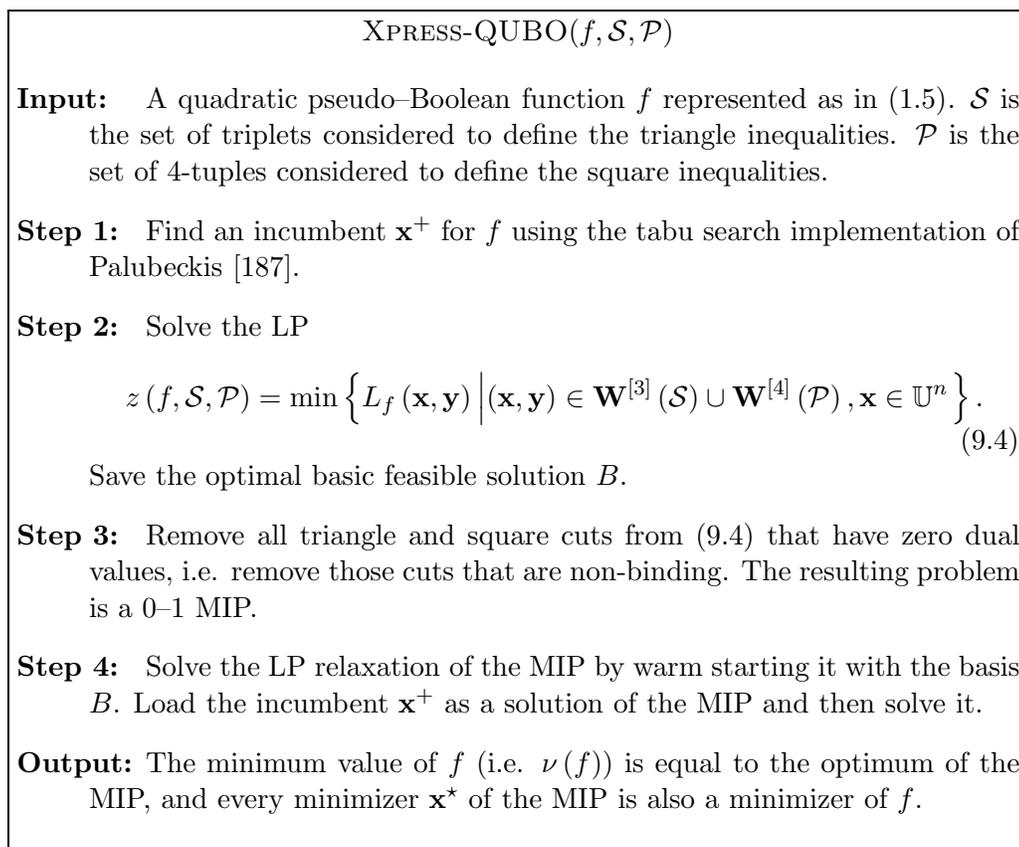


Figure 9.4: XPRESS-QUBO algorithm.

The MIP solver considered is part of Xpress-MP software (see e.g. [21]). The XPRESS-QUBO algorithm to solve QUBOs is described in Figure 9.4. It has been

implemented using the Xpress-Mosel modeling language, and contains 3 input parameters: the quadratic pseudo-Boolean function  $f$ , the set of triples  $\mathcal{S}$  that determines the subset of triangle cuts considered, and the set of tuples  $\mathcal{P}$  that determines the subset of square cuts to be added to the mathematical model. The MIP solver consists of four main steps:

1. Find an incumbent solution to QUBO by using a multi-start tabu search routine developed by Palubeckis [187].
2. Define the LP problem by adding the family of cuts specified by the user, and then solving the LP using the Xpress-Barrier algorithm. Save the optimal basic feasible solution for warm start in step 4.
3. Remove all non-binding cuts. Enforce integrality on the decisions, thus creating a 0–1 MIP.
4. Solve the MIP problem by using the incumbent found on step 1 and the optimal basic feasible solution found on step 2. The optimal solution  $\mathbf{x}$  of the problem is also a minimizer of the associated QUBO problem.

## 9.4.1 Computational results

### 9.4.1.1 Benchmarks with prescribed density

In this section we consider the application of the XPRESS-QUBO Mosel model to solve QUBOs belonging to the  $B$ -250 family. This test set has already been used to test the B&B code under the use of different bounds (see Table 9.7(a)). Table 9.13 lists the results obtained.

XPRESS-QUBO has been considered with two options: one where the cuts added to the model are a set of triangle inequalities defined by  $\mathcal{S}_1$ , and another where the cuts added to the model are a set of triangle inequalities defined by  $\mathcal{S}_2$ .

Table 9.13 also lists the computing times and branching nodes to optimality required by the BIQMAC solver (see [206, 208]). Using very little branching, this solver was able to prove optimality to all cases taking between 4 523 seconds (instance 4) for the easiest

Table 9.13: Maximum of QUBO problems with 250 variables and 10% density (Beasley [37]).

		BIQMAC ([207])	<i>XPRESS-QUBO with <math>\mathcal{S} = \mathcal{S}_1</math> and <math>\mathcal{P} = \emptyset</math></i>					<i>XPRESS-QUBO with <math>\mathcal{S} = \mathcal{S}_2</math> and <math>\mathcal{P} = \emptyset</math></i>				
<i>Instance</i>	<i>Nodes</i>	<i>Time to Maximum</i>	<i>Maximum</i>	<i>Nodes</i>	<i>Computing Time</i>			<i>Maximum</i>	<i>Nodes</i>	<i>Computing Time</i>		
					<i>Incumbent</i>	<i>Relaxation</i>	<i>MIP<sup>†</sup></i>			<i>Incumbent</i>	<i>Relaxation</i>	<i>MIP<sup>†</sup></i>
1	37	9 271 s	[45 607,46 064]	357	3.2 s	107.6 s	3 600.0 s	[45 607,45 992]	287	3.2 s	1 243.3 s	3 600.0 s
2	19	4 823 s	[44 810,46 312]	375	3.2 s	90.6 s	3 600.0 s	[44 810,46 275]	313	3.3 s	1 202.7 s	3 600.0 s
3	19	4 873 s	49 037	117	3.1 s	113.9 s	1 266.3 s	49 037	51	3.3 s	1 287.3 s	792.8 s
4	17	4 523 s	[41 274,42 696]	340	3.3 s	101.4 s	3 600.0 s	[41 274,42 579]	282	3.2 s	1 458.0 s	3 600.0 s
5	21	5 364 s	47 961	37	3.1 s	134.7 s	524.3 s	47 961	9	3.1 s	1 548.6 s	224.2 s
6	223	52 502 s	[41 014,43 629]	317	3.2 s	146.0 s	3 600.0 s	[41 014,43 568]	276	3.3 s	1 203.5 s	3 600.0 s
7	37	9 072 s	[46 757,47 510]	344	3.1 s	102.4 s	3 600.0 s	[46 757,47 426]	280	3.1 s	1 342.2 s	3 600.0 s
8	4 553	320 105 s	[35 726,39 594]	384	3.1 s	86.2 s	3 600.0 s	[35 726,39 517]	381	3.1 s	1 053.4 s	3 600.0 s
9	47	11 577 s	[48 916,49 738]	319	3.1 s	111.4 s	3 600.0 s	[48 916,49 674]	265	3.2 s	1 406.0 s	3 600.0 s
10	63	16 480 s	[40 442,42 597]	355	3.2 s	89.0 s	3 600.0 s	[40 442,42 442]	311	3.2 s	1 008.4 s	3 600.0 s

\*Computed on an AMD Athlon 64 X2 Dual Core 4800+, 2.41 GHz, 4GB RAM and runs XP.

<sup>†</sup>The MIP solver stage was set to run at most 3 600 sec.

case and 320 105 seconds for the hardest case (instance 8). Comparing these results with those of XPRESS-QUBO we can identify a larger number of branching nodes required for the later solver. XPRESS-QUBO can however process these nodes more quickly. Hence, our proposed method is faster for two instances (number 3 and 5) than BIQMAC.

As expected, the linear programming stage of solver XPRESS-QUBO( $\mathcal{S}_1$ ) is shorter than that of XPRESS-QUBO( $\mathcal{S}_2$ ), but for the MIP stage the later procedure is faster (at least for instances 3 and 5).

#### 9.4.1.2 MAX-2-SAT

In this section we compare the XPRESS-QUBO implementation with the lift-and-project decomposition approach developed by Bonami and Minoux [46].

In Section 8.7.1 we have already compared the iterated roof-duality bounds and its improved versions with that one returned by lift-and-project. We were intrigued then by the fact that the lift-and-project bound is near-identical to the bound  $\kappa(f, \mathcal{S}_2)$ .

Table 9.14 compares XPRESS-QUBO( $\mathcal{S}_1$ ) to the approach of [46]. The two methods have similar results, both in solution time and in the number of branching nodes. A one to one comparison between the two solvers may be not fair, since different computer systems and different MIP solvers were used by each one of the approaches.

#### 9.4.1.3 MAX-CUT

This section investigates the application of XPRESS-QUBO to solve the classical 2D and 3D Ising models. Traditionally, branch-and-cut have been very successful in computing the minimum state ground of 2D Ising models with Gaussian interactions ([34, 223]). It is well known that the standard Ising problem can be formulated as a MAX-CUT problem on a graph whose vertex set is the set of spins and whose edge set is determined by the set of non-empty interactions between any two spins. The interactions can be positive or negative.

In what follows, we will show that XPRESS-QUBO is a reasonable alternative to

Table 9.14: Optimal solutions for the Bonami and Minoux [46] MAX-2-SAT formulas.

Variables	Clauses	Instance	Bonami and Minoux [46]		XPRESS-QUBO		
			Nodes	Time*	Optimum	Nodes	Time**
75	525	1	1	36.2 s	57	1	7.2 s
		2	50	39.7 s	61	49	14.2 s
		3	14	44.2 s	59	6	11.8 s
		4	8	41.8 s	56	1	7.5 s
		5	276	57.0 s	65	241	28.0 s
75	550	1	11	46.4 s	62	1	8.5 s
		2	7	48.6 s	60	1	7.6 s
		3	1	21.1 s	59	1	10.3 s
		4	9	41.3 s	62	1	8.4 s
		5	231	62.1 s	70	398	44.5 s
75	600	1	55	66.2 s	77	35	16.9 s
		2	11	55.1 s	73	4	12.0 s
		3	19	57.8 s	71	8	14.7 s
		4	8	62.4 s	69	1	9.1 s
		5	246	63.5 s	75	274	31.6 s
100	700	1	344	128.0 s	83	399	80.8 s
		2	43	102.0 s	78	71	40.2 s
		3	17	90.3 s	74	6	25.7 s
		4	260	106.2 s	80	322	64.5 s
		5	764	155.4 s	86	1 130	150.5 s
150	850	1	41	177.0 s	75	14	77.4 s
		2	991	336.5 s	86	2 043	608.6 s
		3	21	180.5 s	75	6	72.5 s
		4	568	262.4 s	85	1 036	277.0 s
		5	307	220.9 s	83	403	151.7 s
200	1 000	1	13 786	3 030.0 s	[92, 94]	22 218	>7 200.0 s
		2	606	451.5 s	86	793	406.3 s
		3	18 726	4 189.0 s	[94, 96]	18 894	>7 200.0 s
		4	4 570	1 150.0 s	92	11 172	3 874.1 s
		5	713	445.7 s	81	502	367.3 s

\*Sun Sparc 1200 MHz.

\*\*Intel Core 2 CPU T7200, 2.0GHz, 2 GB RAM and running XP.

solve 2D and 3D Ising problems. Table 9.15 lists the results obtained by  $\text{XPRESS-QUBO}(\mathcal{S}_1)$  and  $\text{XPRESS-QUBO}(\mathcal{S}_2)$  to find the MAX-CUT of  $5 \times 5 \times 5$  spin glass Ising models proposed by Burer et al. [74].

The most important result of Table 9.15 is that five of the problems could be solved by  $\text{XPRESS-QUBO}(\mathcal{S}_2)$  without any branching.  $\text{XPRESS-QUBO}(\mathcal{S}_1)$  is able to solve all the instances in efficient time but it required more branching nodes to prove optimality. As in previous cases,  $\text{XPRESS-QUBO}(\mathcal{S}_1)$  takes longer time during the MIP stage, and  $\text{XPRESS-QUBO}(\mathcal{S}_2)$  takes longer time during the LP relaxation stage.

The next set of test problems includes the broadly investigated torus graphs considered in the DIMACS library of mixed semidefinite-quadratic-linear programs. The torus graphs are 3D-toroidal graphs, originated from the Ising model (see Table 3.15). Two graphs have  $\pm 1$  interactions and the other two have Gaussian interactions. For each of the above types of interactions there is a graph with 512 spins and a larger graph having 3375 spins. Next we will show that  $\text{XPRESS-QUBO}$  was able to find the MAX-CUT of graph pm3-8-50 for the first time. pm3-8-50 consists of 512 vertices and the edges interactions are  $\pm 1$ . The MAX-CUT of the smaller graph with Gaussian interactions (i.e. g3-8) was already known.

Table 9.16 provides the results obtained using  $\text{XPRESS-QUBO}(\mathcal{S}_1)$ . The instance having Gaussian interactions can be solved in about one hour and half. To best of our knowledge the instance with  $\pm 1$  interactions was solved for the first time by computing nearly 1.9 million nodes during a period of almost 22 days.

The final set of benchmarks considered in this section is derived from 2D toroidal graphs with  $\pm 1$  interactions. These problems were created by Helmberg and Rendl [140] and have been described in Table 3.13. The solver  $\text{XPRESS-QUBO}(\mathcal{S}_1)$  was also considered in this case. The MIP stage was set to run for a maximum computing time of 10 000 seconds. At this point,  $\text{XPRESS-QUBO}$  will return an upper bound and a lower bound for the MAX-CUT weight. The results can be seen in Table 9.17.

The largest relative gap returned by  $\text{XPRESS-QUBO}(\mathcal{S}_1)$  on these Ising instances is 0.7% and corresponds to the largest instance G67. More interesting than being able to

Table 9.15: MAX-CUT of  $5 \times 5 \times 5$  spin glass Ising models (Burer et al. [74]).

Instance	MAX-CUT	XPRESS-QUBO with $\mathcal{S} = \mathcal{S}_1$ and $\mathcal{P} = \emptyset$				XPRESS-QUBO with $\mathcal{S} = \mathcal{S}_2$ and $\mathcal{P} = \emptyset$			
		Nodes	Computing Time			Nodes	Computing Time		
			Incumbent	Relaxation	MIP		Incumbent	Relaxation	MIP
sg3dl051000	110	77	1.8 s	0.5 s	10.9 s	10	1.8 s	19.1 s	9.0 s
sg3dl052000	112	279	1.8 s	0.6 s	23.0 s	6	1.8 s	19.9 s	6.9 s
sg3dl053000	106	918	1.8 s	0.5 s	62.1 s	459	1.8 s	16.6 s	57.8 s
sg3dl054000	114	8	1.8 s	0.6 s	5.9 s	1	1.8 s	25.4 s	0.3 s
sg3dl055000	112	6	1.8 s	0.5 s	5.0 s	1	1.8 s	18.3 s	0.4 s
sg3dl056000	110	138	1.8 s	0.5 s	15.2 s	8	1.8 s	16.6 s	9.0 s
sg3dl057000	112	54	1.8 s	0.5 s	11.4 s	1	1.8 s	27.5 s	0.3 s
sg3dl058000	108	298	1.8 s	0.5 s	25.0 s	77	1.8 s	16.2 s	19.9 s
sg3dl059000	110	12	1.8 s	0.6 s	5.6 s	1	1.8 s	20.3 s	0.3 s
sg3dl0510000	112	20	1.8 s	0.6 s	7.2 s	1	1.8 s	24.6 s	0.3 s

\*Computed on an AMD Athlon 64 X2 Dual Core 4800+, 2.41 GHz, 4GB RAM and runs XP.

Table 9.16: Finding the MAX-CUT for the DIMACS torus graphs.

				<i>XPRESS-QUBO</i> ( $\mathcal{S}_1$ )	
<i>Graph</i>	<i>Spins</i>	<i>Interactions</i>	<i>MAX-CUT</i>	<i>Nodes</i>	<i>Time</i> *
g3-8	512	Gaussian	41 684 814	3 919	5 809 s
pm3-8-50		$\pm 1$	458	302 156	1 871 155 s

\*Computed on an AMD Athlon 64 X2 Dual Core 4800+,  
2.41 GHz, 4GB RAM and runs XP.

compute near optimal solutions is the fact that *XPRESS-QUBO*( $\mathcal{S}_1$ ) can find very large cuts quickly. We have improved in this way the best known solutions to these problems to all the unsolved instances. It is remarkable that this procedure is able to find much better solutions in about the same computing time than several meta-heuristic methods, which have also studied these problems (see e.g. [188]).

Table 9.17: MAX-CUT of the toroidal  $G_{\pm 1}$ -graphs of Helmberg and Rendl [140].

		<i>XPRESS-QUBO</i> with $\mathcal{S} = \mathcal{S}_1$ and $\mathcal{P} = \emptyset$				
<i>Instance</i>	<i>Vertices</i>	<i>MAX-CUT</i>	<i>Nodes</i>	<i>Computing Time</i> *		
				<i>Incumbent</i>	<i>Relaxation</i>	<i>MIP</i> †
G11	100×8	564	30	8.5 s	1.6 s	12.2 s
G12	50×16	556	39	8.4 s	1.8 s	17.7 s
G13	25×32	582	36	8.5 s	1.8 s	22.7 s
G32	100×20	[1 410,1 412]	83 837	35.2 s	5.3 s	10 000.0 s
G33	80×25	[1 382,1 383]	134 133	35.6 s	6.0 s	10 000.0 s
G34	50×40	[1 384,1 388]	66 149	35.2 s	5.9 s	10 000.0 s
G57	100×50	[3 492,3 505]	20 598	111.4 s	21.7 s	10 000.0 s
G62	100×70	[4 862,4 886]	10 109	178.7 s	36.9 s	10 000.0 s
G65	100×80	[5 550,5 581]	4 199	217.4 s	47.1 s	10 000.0 s
G66	90×100	[6 352,6 387]	5 065	258.8 s	159.7 s	10 000.0 s
G67	100×100	[6 932,6 981]	7 683	303.7 s	323.8 s	10 000.0 s

\*Computed on an AMD Athlon 64 X2 Dual Core 4800+, 2.41 GHz, 4GB RAM and runs XP.

†The MIP solver stage was set to run at most 10 000 sec.

#### 9.4.1.4 Minimum 3-partition

The Minimum 3-Partition (M3P) problem has been introduced in Section 8.8.1. M3P as been formulated as a QUBO problem by introducing a pair of 0–1 variables for every vertex.

In this section, we investigate how *XPRESS-QUBO* handles the M3P problem of

certain sparse graphs created by Anjos et al. [20]. Two versions of this approach are considered:  $XPRESS-QUBO(\mathcal{S}_1, \mathcal{Z})$  and  $XPRESS-QUBO(\mathcal{S}_2, \mathcal{Z})$ .  $\mathcal{Z}$  is a set of tuples that identifies the set of square inequalities to be added to the LP as cuts (see (8.19)).

The overall results are listed in Table 9.18. This table includes the computing times of the Anjos et al. [20] method (called SBC), which is a branch-and-cut algorithm based on semidefinite programming. Clearly, both  $XPRESS-QUBO$  methods require the computation of far more branching nodes, but the relaxations can be computed substantially faster than the computation of the SBC bounds. In conclusion, the QUBO approach seems to be an effective tool to solve the M3P of 2D and 3D toroidal graphs.

Table 9.18: Optimal solutions for the M3P problems proposed by Anjos et al. [20].

<i>Instance</i>	<i>Weights</i>	<i>M3P</i>	<i>SBC [20]</i>		<i>XPRESS-QUBO</i> ( $\mathcal{S}_1, \mathcal{Z}$ )		<i>XPRESS-QUBO</i> ( $\mathcal{S}_2, \mathcal{Z}$ )	
			<i>Nodes</i>	<i>Time*</i>	<i>Nodes</i>	<i>Time**</i>	<i>Nodes</i>	<i>Time**</i>
4×4	Gaussian	-954 077	1	16 s	1	1.7 s	1	2.1 s
5×5		-1 484 348	2	23 s	5	2.7 s	13	5.3 s
6×6		-2 865 560	1	312 s	1	4.4 s	9	10.4 s
7×7		-3 282 435	1	3 128 s	9	8.2 s	13	20.9 s
8×8		-5 935 339	1	8 503 s	27	12.7 s	45	43.9 s
4×4	±1	-13	1	< 0.005 s	1	1.8 s	1	2.4 s
5×5		-20	1	4 s	28	4.4 s	14	5.6 s
6×6		-29	1	22 s	107	7.5 s	68	10.8 s
7×7		-40	1	112 s	277	13.8 s	170	25.8 s
8×8		-55	1	1 598 s	243	22.6 s	330	50.1 s
9×9		-64	1	27 349 s	50 175	1 116.5 s	25 794	1 256.4 s
2 × 3 × 4	±1	-20	1	3 s	8	5.6 s	8	6.9 s
2 × 4 × 4		-28	4	234 s	522	19.1 s	592	25.4 s
3 × 3 × 3		-26	1	11 s	20	8.0 s	53	11.9 s
3 × 3 × 4		-36	1	50 s	453	30.0 s	1 222	60.5 s
3 × 4 × 4		-48	1	719 s	17 499	862.9 s	15 629	639.7 s
3 × 4 × 5		-63	16	32 133 s	13 123	1 126.5 s	32 709	2 657.1 s
4 × 4 × 4		-65	19	30 975 s	171 846	15 247.2 s	136 671	11 157.3 s

\*Sun Sparc 1200 MHz.

\*\*Computed on an AMD Athlon 64 X2 Dual Core 4800+, 2.41 GHz, 4GB RAM and runs XP.

## Chapter 10

### Applications

Three applications related to the optimization of quadratic pseudo-Boolean functions are given in this chapter.

The first section discusses how to solve the minimum vertex cover problem in more detail, both theoretically and in practice.

The next application is related to clustering. We propose two approaches: first we show how QUBO can determine clusters in the classical sense, by proposing a hierarchical clustering approach; and secondly we show how to find subsets of vertices “highly” connected on a given graph.

The last section presents a simple QUBO approach to the image binarization problem.

#### 10.1 Minimum vertex cover problem

Let  $G = (V, E)$  be a finite, undirected, simple and loopless graph, where  $V$  and  $E$  respectively denote the vertex and edge sets. Let us define a weight function  $w : V \mapsto \mathbb{R}^+$  on the set of vertices, and denote the *weight* of set  $S \subseteq V$  as  $\mathbf{w}[S] \stackrel{\text{def}}{=} \sum_{i \in S} w_i$ . For any  $S \subseteq V$  we define the *neighbors* of  $S$  to be the vertex set

$$N(S) \stackrel{\text{def}}{=} \{i \in V \setminus S \mid (i, j) \in E \text{ for some } j \in S\}.$$

To simplify the notation, the set of nodes adjacent to vertex  $i$  ( $i \in V$ ) will be sometimes denoted as  $N(i)$ , instead of  $N(\{i\})$ .

A *vertex cover*  $C \subseteq V$  of graph  $G$  is a set of vertices that has an endpoint in every edge of  $G$ . Naturally, the *minimum* vertex cover (or MIN-VC) problem seeks for a

vertex cover of minimum size.

MIN-VC is a classical NP-hard optimization problem ([104]) arising in many applications (see e.g. [11, 45, 169]).

The *minimum weight* vertex cover problem (or MIN-WVC) seeks for a vertex cover of minimum total weight  $w(C)$ . MIN-WVC can be related to many hard combinatorial optimization problems, and in particular with QUBO by solving the mathematical problem (5.6). Thus, in theory every QUBO can be transformed to a MIN-WVC in polynomial time and size, and find the optimum by determining the optimal vertex packing problem.

A *stable set*  $S \subseteq V$  of graph  $G$  is a set of pairwise nonadjacent vertices, i.e. such that  $(i, j) \notin E$ , for all  $i, j \in S$ . The *maximum stable set* is a stable set  $S$  with maximum cardinality  $|S|$ . The *maximum weighted stable set* is a stable set  $S$  with maximum weight  $\mathbf{w}[S]$ .

A stable set is also called as an *independent set* or as a *vertex packing*. The size of the maximum stable set of graph  $G$  is called the stability number of  $G$  and is denoted by  $\alpha(G)$ . If weights  $\mathbf{w}$  are assigned to the vertices, then the maximum weighted stable set of  $G$  is denoted by  $\alpha_{\mathbf{w}}(G)$ .

It is not difficult to verify that the complement  $C = V \setminus S$  of a stable set  $S$  is a vertex cover  $C$  of graph  $G$ . Thus, finding a minimum vertex cover of a graph  $G$  is as difficult as finding a maximum stable set on the same graph, and the knowledge of one solution to one problem directly implies a solution to the other problem.

It is well known that a maximum weighted stable set can be found by solving the 0–1 linear programming problem

$$\begin{aligned} \max \quad & \sum_{u \in V} w_u x_u \\ \text{subject to} \quad & \\ & x_u + x_v \leq 1, \quad (u, v) \in E, \\ & x_u \in \mathbb{B}, \quad u \in V. \end{aligned} \tag{SIP}$$

Each feasible solution  $\mathbf{x}$  of (SIP) correspond to the 0–1 characteristic vector associated to a stable set of  $G$ , i.e.  $x_u = 1$  if and only if  $u \in V$  belongs to a stable set of  $G$ .

The *fractional maximum stable set problem* is obtained from (SIP) by relaxing the integrality constraints to  $x_u \geq 0, \forall u \in V$ , and is denoted by (SLIP).

The following result is due to Balinski [30, 29] and arises from the fact that all basic feasible solutions of (SLIP) are half-integral.

**Proposition 10.1** ([30]). *Every optimal solution  $\mathbf{x}^*$  to (SLIP) is  $x_i^* = 0, \frac{1}{2}$ , or 1 for all  $i = 1, \dots, n$ .*

Nemhauser and Trotter [182] have shown that those variables assuming binary values in an optimal solution of (SLIP), retain the same values in an optimal solution of (SIP).

**Proposition 10.2** ([182]). *Suppose  $\mathbf{x}^*$  is an optimal  $(0, \frac{1}{2}, 1)$ -valued solution to (SLIP) and  $U = \{i \in V \mid x_i^* = 1\}$ . There exists an optimal stable set in  $G$  that contains  $U$ .*

The significance of the previous proposition results from the possibility of reducing the size of graph  $G$  to a graph  $G' = (V', E')$ , such that  $V' \subseteq V$ , and  $\alpha_{\mathbf{w}}(G) = \alpha_{\mathbf{w}}(G') + \mathbf{w}[U]$ , where  $U$  is a subset of vertices belonging to a stable set of  $G$ , and which is determined by the set of variables with value one in an optimal solution of (SLIP).

Nemhauser and Trotter [182] have also characterized the set of *irreducible* graphs for which there is no 0–1 value for any variable and any optimal solution to (SLIP).

**Proposition 10.3** ([182]). *The solution  $\mathbf{x}^* = (\frac{1}{2}, \dots, \frac{1}{2})$  is the unique optimal solution to (SLIP) if and only if  $\mathbf{w}[S] < \mathbf{w}[N(S)]$  for all non-empty stable sets  $S$  of  $G$ .*

Given a graph  $G = (V, E)$  we shall propose an efficient algorithm based on QUBO to find an irreducible graph  $G^*$ , such that  $\alpha_{\mathbf{w}}(G) = \alpha_{\mathbf{w}}(G^*) + \mathbf{w}[U]$ , where  $U$  is a set of vertices with value 1 in an optimal solution to (SLIP). Nemhauser and Trotter [182] have already proposed an algorithm for this procedure whose time complexity is  $O(|V| \text{max-flow}(|V|, |E|))$ .

The procedure proposed in this section consists of a 2-stage procedure. In the first stage a maximum flow is found in a network having  $2|V| + 2$  nodes and  $2(|E| + |V|)$  arcs. In the second stage the strong components of the residual network are found, and this information leads to finding an irreducible graph, which may be characterized by the union of disjoint and irreducible (smaller) graphs.

The combined complexity of the proposed method is therefore  $O(|E| + \text{max-flow}(|V|, |E|))$ .

**Definition 10.1.** For  $\mathbf{x} \in \mathbb{U}^n$ , let us define the integral set  $I(\mathbf{x})$  as the set of indices  $i$  ( $i = 1, \dots, n$ ) such that  $x_i$  has a binary value.

**Proposition 10.4** ([199]). If  $\mathbf{x}^*$  and  $\mathbf{x}^{**}$  are two minima to (SLIP), then there exists a minimizer  $\mathbf{x}^+$  to (SLIP) such that

$$I(\mathbf{x}^+) = I(\mathbf{x}^*) \cup I(\mathbf{x}^{**}).$$

The previous result due to Picard and Queyranne [199] implies that a solution to (SLIP) having a *maximum* number of variables with 0–1 values can be found simply by finding an irreducible graph of the original graph associated to (SLIP).

**Proposition 10.5** ([199]). There is a unique maximal set of variables which are integral in optimal (SLIP) solutions.

The proposed reduction algorithm will therefore find the *unique* irreducible graph  $G'$  of graph  $G$ . In practice, this reduction technique enhanced with some other preprocessing tools (e.g., probing and second order derivatives) can result in large simplifications for MIN-WVC (or similarly for weighted graph stability) problems of certain classes of graphs. We have tested the proposed algorithm in numerous classes of graphs. We shall give particular attention to finding the minimum vertex cover of power-law and certain classes of planar graphs.

### 10.1.1 Solving MIN-WVC using QUBO

The minimum weighted vertex cover of graph  $G = (V, E)$  can be found by solving the QUBO problem (see Section 2.2.2)

$$\min_{\mathbf{x} \in \mathbb{B}^V} \left( \sum_{i \in V} w_i x_i + \sum_{(i,j) \in E} \max(w_i, w_j) \bar{x}_i \bar{x}_j \right). \quad (10.1)$$

If an optimal solution  $\mathbf{x}^*$  to (10.1) is the characteristic vector of a vertex cover  $C^*$  of  $G$ , then its optimal weight is  $w(C^*)$ . If  $\mathbf{x}^*$  has however two entries (say  $x_i$  and  $x_j$ ,  $i \neq j$ ) with value 0, and  $G$  has an edge  $(i, j)$ , then clearly  $\mathbf{x}^*$  does not represent a vertex cover. In this case it is possible to find another optimal solution  $\mathbf{x}^{**}$  of problem (10.1) which covers all edges covered by  $C^*$  and edge  $(i, j)$ .

To verify this fact, let us define  $E^*$  as the set of edges not covered by the vertex set  $C^*$ , i.e.  $E^* = \{(i, j) \in E \mid x_i^* = x_j^* = 0\}$ . Thus, the optimum of problem (10.1) is  $\nu(G) = \sum_{i \in C^*} w_i + \sum_{(i,j) \in E^*} \max(w_i, w_j)$ .

Let us consider an edge  $(u, v) \in E^*$  (assuming that  $E^* \neq \emptyset$ ). We shall show next that we can add  $u$  (or  $v$ ) to  $C^*$  and get an objective value not larger than  $\nu(G)$ . Let

$$x_i^{**} = \begin{cases} x_i^*, & k \neq u, \\ 1, & k = u. \end{cases}$$

The total weight of  $C^{**} = C^* \cup \{u\}$ , provided by the value of (10.1) in  $\mathbf{x}^{**}$ , is

$$\begin{aligned} & \sum_{i \in C^* \cup \{u\}} w_i + \sum_{(i,j) \in E^*} \max(w_i, w_j) - \sum_{(u,j) \in E^*} \max(w_u, w_j) \\ = & \nu(G) + w_u - \sum_{(u,j) \in E^*} \max(w_u, w_j) \\ \leq & \nu(G) + w_u - w_u |E^*| \\ = & \nu(G) + w_u (1 - |E^*|) \\ \leq & \nu(G). \end{aligned}$$

Since  $C^{**}$  does not increase the total weight, then  $\mathbf{x}^{**}$  is also an optimal solution of problem (10.1). By iteratively applying the previous procedure (at most  $|V|$  times)

eventually one will obtain an optimal solution, which corresponds to a minimum vertex cover of  $G$ .

In what follows we consider the implication network model corresponding to the QUBO problem (10.1), described in Section 5.3. The network has a source  $x_0$ , a sink  $\bar{x}_0$ , a pair of nodes corresponding to each variable  $x_i$  and its complement  $\bar{x}_i$  ( $i \in V$ ), has an arc with capacity  $\frac{w_i}{2}$  from the source to every node of a complemented variable, has an arc with capacity  $\frac{w_i}{2}$  from every node of a simple variable to the sink, has two arcs with capacity  $\frac{\max(w_i, w_j)}{2}$  for every edge  $(i, j)$ , one from the  $\bar{x}_i$  node to the  $x_j$  node and another from the  $\bar{x}_j$  node to the  $x_i$  node.

**Example 10.1.** Consider the graph  $G$  of Figure 10.1(a), which has 6 vertices and 9 edges. There exists a unique minimum vertex cover  $C = \{v_1, v_3, v_5\}$ . It is also trivial to verify that  $S = \{v_2, v_4, v_6\}$  is the unique stable set of  $G$  of maximum cardinality. The optimum value of (SLIP) is 3 and it has an optimal solution given by  $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ . Figure 10.1(b) provides the implication network associated to problem (10.1).

The main results derived in Chapters 5 and 7 for QUBO are specialized in the following proposition for MIN-WVC.

**Proposition 10.6.**

- (i) The optimal value of (SLIP) coincides with the roof-dual of problem (10.1) (see Section 5.2).
- (ii) The roof-dual of problem (10.1) coincides with the value of maximum flow  $\varphi^*$  between the source  $x_0$  and the sink  $\bar{x}_0$  of the associated implication network  $N$  (see Proposition 5.8).
- (iii) The set  $P$  of nodes reachable from the source of the residual network  $N[\varphi^*]$  defines the set of strong persistencies, i.e. if  $x_i \in P$  then vertex  $i$  belongs to all minimum weighted vertex covers of graph  $G$ , and if  $\bar{x}_j \in P$  then vertex  $j$  does not belong to any minimum weighted vertex cover of  $G$  (or similarly, vertex  $j$  belongs to all stable sets of  $G$ ) (see Proposition 5.9).

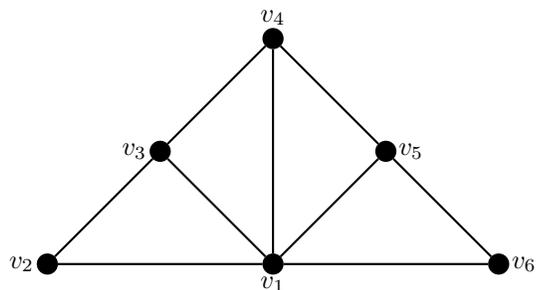
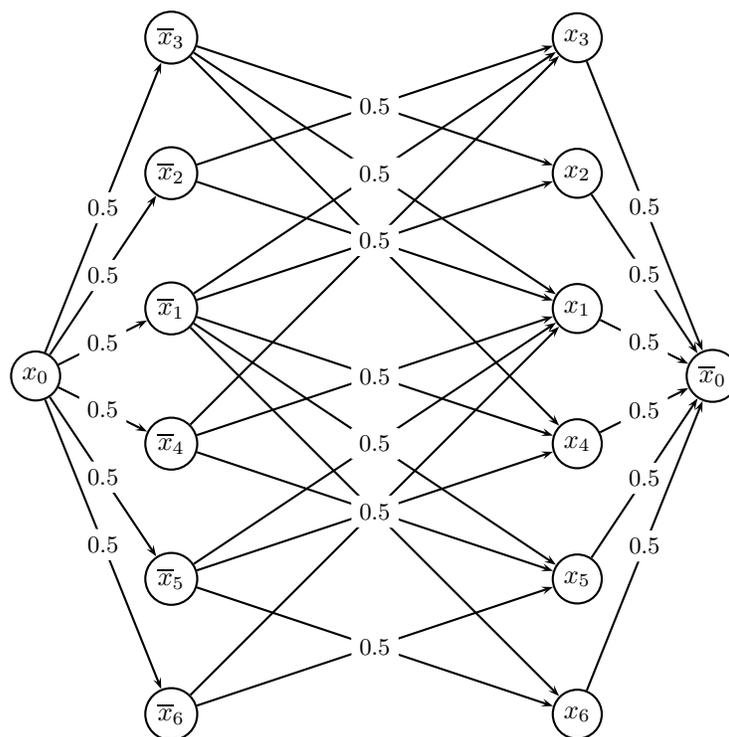
(a) Graph  $G$  with 6 vertices and 9 edges.(b) Implication network associated to problem (10.1) of graph  $G$ .

Figure 10.1: Representing a MIN-WVC problem using the implication network model.

- (iv) If a strong component  $K$  of the residual network  $N[\varphi^*]$  contains both a node  $x_i$  and its complement  $\bar{x}_i$ , then vertex  $i$  is irreducible, i.e.  $x_i^* = \frac{1}{2}$  for all optimal solutions  $\mathbf{x}^*$  of (SLIP). All other nodes  $x_j$  of  $K$  ( $i \neq j$ ) must have a complement node  $\bar{x}_j$  and consequently are all irreducible too (see Proposition 5.5).
- (v) If the residual network  $N[\varphi^*]$  contains dual components  $K$  and  $K'$ , then there exists a solution to (SLIP) with integral values for all vertices of  $G$  which appear in literals of  $K$ . Namely, if there exists a directed path from  $K'$  to  $K$  and there is no directed path from  $x_0$  to  $K'$ , then  $x_i \in K$  implies that there exists a minimum weighted vertex cover of  $G$  containing vertex  $i$ , and  $\bar{x}_i \in K$  implies that there exists a minimum weighted vertex cover which does not include vertex  $i$  (see Theorem 7.1).

*Proof.* Only point (iv) does not follow trivially from the results presented in the previous chapters. Let us show this result, which applies to any quadratic pseudo-Boolean function  $f$ , by using a network flows argument. Since  $K$  is a strong component, then there exists a path from  $x_i$  to its complement  $\bar{x}_i$ , and hence the roof dual of  $f(\mathbf{x}[\{i\} \leftarrow \{1\}])$  is strictly larger than the roof-dual of  $f$ . In addition, there exists a path from  $\bar{x}_i$  to its complement  $x_i$ , and hence the roof-dual of  $f(\mathbf{x}[\{i\} \leftarrow \{0\}])$  is strictly larger than the roof-dual of  $f$ . So, a 0-1 assignment to  $x_i$  for  $f$  would imply a larger roof-dual bound than the bound obtained by fixing  $x_i$  in  $f$  to the only possible value left by Proposition 10.1, i.e.  $x_i = \frac{1}{2}$ .  $\square$

**Example 10.2.** Consider again graph  $G$  of Figure 10.1(a). The maximum flow  $\varphi$  of the implication network  $N[\varphi]$  associated to this graph (see Figure 10.2) has value 3. The residual network (which is not uniquely defined) is given in Figure 10.2 and corresponds to the quadratic posiform

$$\phi_G = x_2x_3 + x_5x_6 + x_1x_4 + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_3 + \bar{x}_1\bar{x}_5 + \bar{x}_1\bar{x}_6 + \bar{x}_3\bar{x}_4 + \bar{x}_4\bar{x}_5.$$

The minimum of  $\phi_G$  in  $\mathbb{B}^6$  is 0, and corresponds to the solution  $\mathbf{x}^+ = (1, 0, 1, 0, 1, 0)$ , which is a characteristic vector associated to a minimum vertex cover  $C = \{1, 3, 5\}$  of

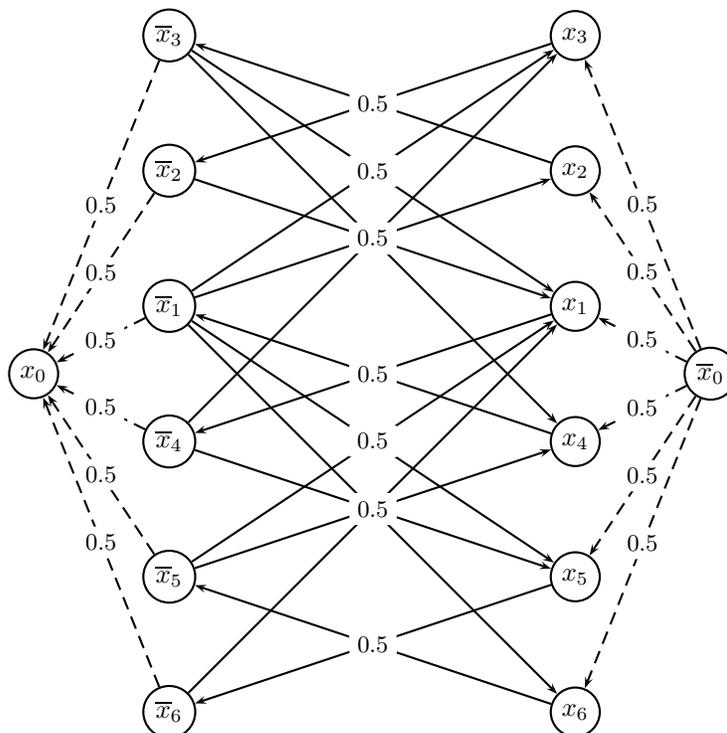


Figure 10.2: Residual network of the implication network model of a MIN-WVC problem.

*G.* This solution could be found using property (v) of Proposition 10.6. The strong components of  $N[\varphi]$  are  $K = \{x_1, \bar{x}_2, x_3, \bar{x}_4, x_5, \bar{x}_6\}$  and  $K' = \{\bar{x}_1, x_2, \bar{x}_3, x_4, \bar{x}_5, x_6\}$ . There exists a directed path (e.g.,  $\{(\bar{x}_5, x_1)\}$ ) from  $K'$  to  $K$  and there is no directed path from  $x_0$  to  $K'$ . Thus  $C = \{i \mid x_i \in K\}$  is a minimum vertex cover of  $G$ , and since there exists a directed arc from  $K'$  to  $K$  then  $C$  is also the unique minimum vertex cover of  $G$  by Theorem 7.1.

**Theorem 10.1.** Given a graph  $G(V, E)$  with vertex weights  $\mathbf{w}$ , a set of disjoint and irreducible graphs  $G^{(k)}(V^{(k)}, E^{(k)})$ ,  $k = 1, \dots, l$ , of  $G$  can be found in

$$O(\text{max-flow}(|V|, |E|) + \text{strong-components}(|V|, |E|)).$$

*Proof.* Clearly, the implication network  $N$  can be built in linear time of the size of the graph  $G$ . A maximal set of persistencies of (SLIP) can be derived from the residual network obtained by applying a max-flow algorithm to  $N[\varphi]$ . By (iii) of Proposition 10.6

a set of strong persistencies can be obtained from the source side of the residual network. After removing the nodes of  $N[\varphi]$  associated to the strong persistencies, we apply a strong components algorithm and use (v) of Proposition 10.6 to establish the remaining persistencies for (SLIP). After removing all nodes of the residual network  $N[\varphi]$  which have a persistency associated to it, the remaining network consists of a set of disjoint networks  $N_k$ , one for each strong component  $K_k$  left ( $k = 1, \dots, l$ ). Note that by (iv) of Proposition 10.6 all strong components  $K_k$  ( $k = 1, \dots, l$ ) must contain both a node associated to a variable  $x_i$  and a node associated to the complement  $\bar{x}_i$ . This property implies that the strong components must be disjoint. So,  $V^{(k)} = \{i \mid x_i, \bar{x}_i \in N_k\}$  and  $E^{(k)} = \{(i, j) \in E \mid i, j \in V^{(k)}\}$  ( $k = 1, \dots, l$ ).  $\square$

**Theorem 10.2.** *Given a graph  $G(V, E)$  with vertex weights  $\mathbf{w}$ , the set of disjoint and irreducible graphs  $G^{(k)}(V^{(k)}, E^{(k)})$ ,  $k = 1, \dots, l$ , of  $G$  is uniquely defined.*

*Proof.* This result is an immediate consequence of Proposition 10.4, or it can be obtained directly from the more general result established by Corollary 5.1 given in Section 5.2.1.  $\square$

In the literature there are two other approaches for identifying data reductions. One is based on the so called *critical independent sets* (see e.g. [77]) and the other one is based on the identification of *crown* structures followed by their removal from the graph (see e.g. [86]). Comparing these 2 approaches with roof-duality is an interesting topic of further research. We did not pursue a deep investigation of the other two approaches since we believe that roof-duality (including both strong and weak persistency) is not weaker in the data reduction sense than any of the other two approaches.

### 10.1.2 Struction

Hammer [120] has shown that posiform maximization is equivalent to the maximum weighted stable set problem on a graph. Using a different approach than that of [120], we have already seen in Section 5.2.1 that this claim is true using a certain type of

graph. Hammer [120] has shown this result differently through the concept of *conflict graph*.

Let us consider a pseudo-Boolean function  $f$  represented by posiform

$$\phi_f(x_1, \dots, c_n) = K + \sum_{i=1}^m w_i T_i,$$

where  $K$  is a constant,  $w_i$  is a positive coefficient ( $i = 1, \dots, m$ ), and  $T_i = \prod_{j \in A_i} x_j \prod_{j \in B_i} \bar{x}_j$ , with  $A_i, B_i \subseteq \{1, \dots, n\}$  and  $A_i \cap B_i = \emptyset$  ( $i = 1, \dots, m$ ).

To posiform  $\psi = \phi_f - K$  we associate a *conflict graph*  $G_f = (V, E)$  with the set of vertices  $V = \{1, \dots, m\}$  and the set of edges  $E = \{(i, j) \mid (A_i \cap B_j) \cup (A_j \cap B_i) \neq \emptyset\}$ . In other words, an edge  $(i, j)$  in  $G_f$  represents the fact the terms  $T_i$  and  $T_j$  are in conflict, i.e.  $T_i T_j = 0$ . To complete the construction, a positive weight  $w_i$  is associated to every vertex  $i \in V$ .

From the way  $G_f$  has been defined it is clear that

$$\max_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x}) = \alpha_{\mathbf{w}}(G_f),$$

i.e. the total weight of the maximum weighted stable set of  $G_f$  is equivalent to the maximum of the pseudo-Boolean function  $f$ .

The inverse reduction is also true ([120]), i.e. to every maximum weighted graph stability problem there is a pseudo-Boolean function whose maximum has the same value has the total weight of the optimal stable set.

The 1-to-1 correspondence between these two combinatorial optimization problems made possible the use of Boolean identities to derive useful graph transformations, which preserve the stability number or change it by a constant (see e.g. [124, 142, 143]).

This study will concentrate on such a method derived by Ebenegger et al. [94] and named by Hammer et al. [126] as the *struction* of a graph. This interesting approach for determining  $\alpha(G)$ , consists in transforming a graph  $G$  into a graph  $G'$  with  $\alpha(G') = \alpha(G) - 1$ . By repeated applications of such an operation, one eventually gets

a clique (whose stability number is 1) and the determination of  $\alpha(G)$  follows by counting the number of transformations applied during this process. By using the struction repeatedly, the size of the resulting graph may increase exponentially in the number of vertices (and edges). However, specialized versions of this algorithm have provided polynomial algorithms for the stability number of some classes of graphs (see e.g. [141]).

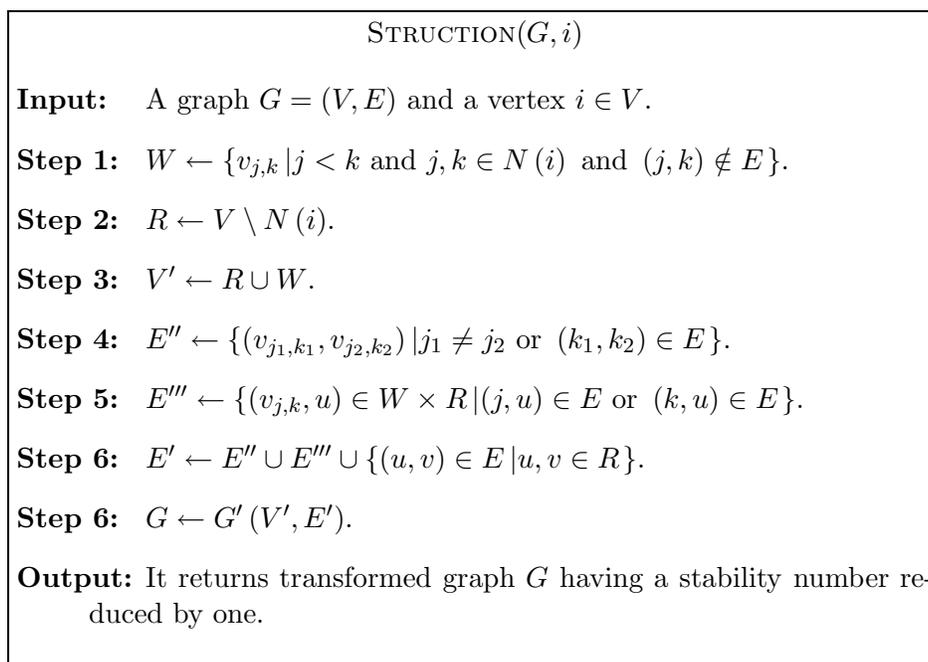


Figure 10.3: On iteration of struction of a graph.

Given a graph  $G = (V, E)$  (with all weights  $\mathbf{w} = (1, \dots, 1)$ ) and a vertex  $i$  (called the center of struction) the method proceeds by transforming  $G$  as indicated in Figure 10.3. The resulting graph has a stability number reduced by one when compared with the stability number of the original graph. This result can easily be established by using the pseudo-Boolean relation indicated above (see e.g. [14, 94, 126]).

### 10.1.3 Simplified reduction techniques for MIN-VC

In this subsection we shall provide simple rules to detect cases that can automatically simplify the MIN-VC problem.

**Rule 1.** *If vertex  $i$  of graph  $G = (V, E)$  belongs to a single edge  $(i, j) \in E$ , then there is a minimum vertex cover of  $G$  that includes vertex  $j$  (but that does not include vertex*

*i*).

**Rule 2.** *If vertex  $i$  of graph  $G = (V, E)$  is adjacent to exactly two vertices  $j$  and  $k$ , and*

*(i) if  $j$  and  $k$  are non-adjacent, then there is a minimum vertex cover of  $G$  that (1) either includes both vertices  $j$  and  $k$  or none and that (2) includes vertex  $i$  if and only if vertices  $j$  and  $k$  are not in a minimum vertex cover;*

*(ii) if  $j$  and  $k$  are adjacent, then there is a minimum vertex cover that includes both vertices  $j$  and  $k$  (but that does not include vertex  $i$ ).*

**Rule 3.** *If the induced subgraph defined by the set of adjacent vertices  $N(i)$  of vertex  $i$  of graph  $G = (V, E)$  is a clique  $C$ , then there is a minimum vertex cover that includes all vertices  $C$  (but that does not include vertex  $i$ ).*

A vertex in a graph is called *simplicial* if its neighborhood is a clique. Rule 3 applies therefore to simplicial vertices.

**Definition 10.2** ([124]). *A magnet in a weighted graph  $G = (V, E)$  is a pair  $[i, j]$  of adjacent vertices with the same weight and such that every vertex in  $N(i) \setminus (N(j) \cup \{j\})$  is linked to all vertices in  $N(j) \setminus (N(i) \cup \{i\})$ .*

Consider the following transformation which, given a graph  $G = (V, E)$  and a magnet  $[i, j]$  in  $G$ , builds a new graph  $G' = \mathcal{T}(G, [i, j])$  such that

1. remove all the edges between vertex  $i$  and the vertices in  $N(i) \setminus N(j)$ ; and
2. remove vertex  $j$  (together with all edges incident to it).

**Rule 4** ([124]). *If  $[i, j]$  is a magnet in a weighted graph  $G = (V, E)$ , then there is a minimum vertex cover of  $G$  that contains vertex  $j$ , i.e.*

$$\nu_{\mathbf{w}}(G) = 1 + \nu_{\mathbf{w}}(\mathcal{T}(G, [i, j])).$$

**Definition 10.3.** *A vertex  $i$  is dominated by vertex  $j$  in a graph  $G = (V, E)$  if  $N(i) \subseteq (N(j) \cup \{j\})$ .*

**Rule 5.** *If a graph  $G = (V, E)$  contains two adjacent vertices  $i$  and  $j$  such that  $i$  is dominated by  $j$ , then there is a minimum vertex cover of  $G$  that contains vertex  $i$ .*

**Definition 10.4.** *A magnet  $(C, S)$  in a graph  $G$  is a  $d$ -magnet if every vertex  $i \in C$  is dominated by some vertex  $j \in S$ .*

**Definition 10.5.** *A graph  $H$  is a demagnetization (respectively  $d$ -demagnetization) of a graph  $G$  if*

- (i)  *$H$  does not contain any magnet (respectively  $d$ -magnet); and*
- (ii) *there exists a sequence  $G = G_1, \dots, G_q = H$  of graphs such that  $G_{i+1} = T(G_i, [x_i, y_i])$  for a magnet (respectively  $d$ -magnet)  $[x_i, y_i]$  in  $G_i$  ( $i = 1, \dots, q$ ).*

The concept of removing vertices is a helpful tool for reducing the dimension of the minimum vertex cover problem, and clearly one would like to use it as often as possible. If none of the previous simplification rules applies, there is still a chance that inserting or deleting new edges might yield a graph which does allow some of the above rules to apply. Butz et al. [78] provided some conditions under which inserting (or deleting) a new edge does not change the size of the minimum vertex cover of the graph.

**Rule 6** ([78]). *If a graph  $G = (V, E)$  contains a vertex  $k$  which is adjacent to  $i$ , but not to  $j$ , such that the neighborhood of  $k$  is part of the union of the neighborhoods of  $i$  and  $j$  and possibly  $\{i\}$  (i.e.  $N(k) \subseteq \{i\} \cup N(i) \cup N(j)$ ), then the insertion of edge  $(i, j)$  (if is absent in  $G$ ), or the removal of this edge (if it is present in  $G$ ), does not alter the size of the minimum vertex cover of the graph  $G$ .*

An ordered triple  $(k, i, j)$ , whose vertices  $k$ ,  $i$  and  $j$  satisfy the conditions of Rule 6, has been called a *switching triple* ([14]). Clearly, Rule 6 allows us to perform edge removal and edge insertion to a switching triple as additional preserving graph stability transformations, which in its turn may make possible some of the other reduction rules to apply further simplifications.

### 10.1.4 Implementation algorithms

In this section we describe two basic approaches to find the size of the minimum vertex cover of graphs (with weights being disregarded). One approach uses roof-duality (see Section 10.1.1) and the other uses struction (see Section 10.1.2).

Both approaches use data reduction rules to simplify as much as possible the problem before attempting the more time consuming techniques (i.e. roof-dual and struction). Figure 10.4 describes the sequence of rules that we have considered to speedup the computing times. Each one of these rules was previously described in Section 10.1.3. This algorithm has been called `APPLY-RULES-VC` and it has 3 input parameters: a simple loopless graph  $G$ , the total number of attempts  $t$  to simplify along the rules sequence, and the maximum size clique  $c$  to be searched for in the neighborhood of each vertex.

The sequence of rules that is defined by `APPLY-RULES-VC` starts by applying computationally less time consuming rules, gradually applying more expensive rules along the sequence. `APPLY-RULES-VC` repeats this sequence whenever any simplification has been identified by the rules, up to a limit of  $t$  iterations.

`APPLY-RULES-VC` applies Rule 3 only to simplicial vertices with at most  $c$  neighbors. If  $c$  and  $t$  are large enough, then the transformed graph is a demagnetization at the end of the procedure.

Figure 10.5 describes the algorithm of the roof-dual based algorithm to find a minimum vertex cover  $C$  of a given graph  $G$ . This method has been called `MIN-VC-USING-ROOF-DUALITY`, and it contains three input parameters like routine `APPLY-RULES-VC`. It will apply any data reduction rules followed by roof-duality strong persistencies reduction as much as possible and as established by parameters  $t$  and  $c$ .

After this stage, if necessary `MIN-VC-USING-ROOF-DUALITY` applies the `PREPRO` routine to the transformed graph as explained before in Section 7.7.1. If the problem is not completely solved after applying all the simplification and preprocessing tools, then an enumerative exact procedure is applied to solve the residual problem. A minimum vertex cover is finally computed by extending the partial covering obtained from the

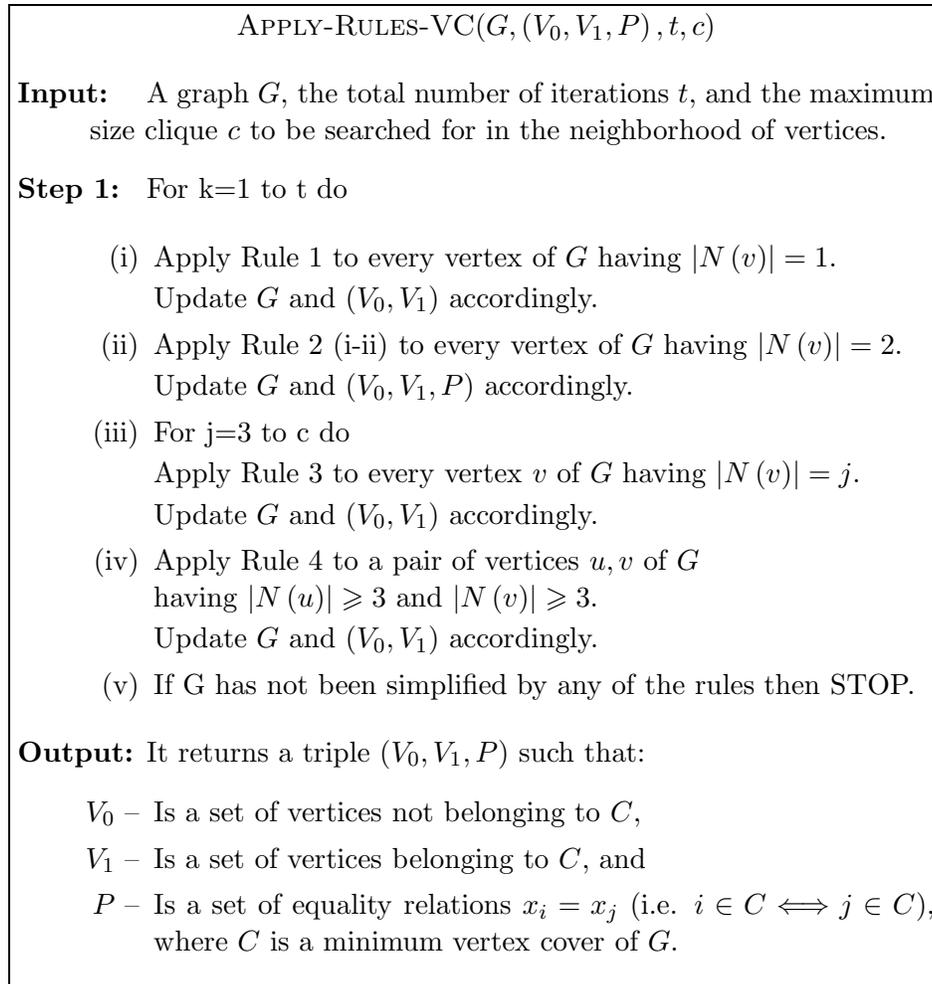


Figure 10.4: Data reduction algorithm for MIN-VC.

solution of the residual problem (see Corollary 7.1). The routine that includes PREPRO, enumeration and vertex cover expansion has been named in Section 7.7.3 as PREPRO\*.

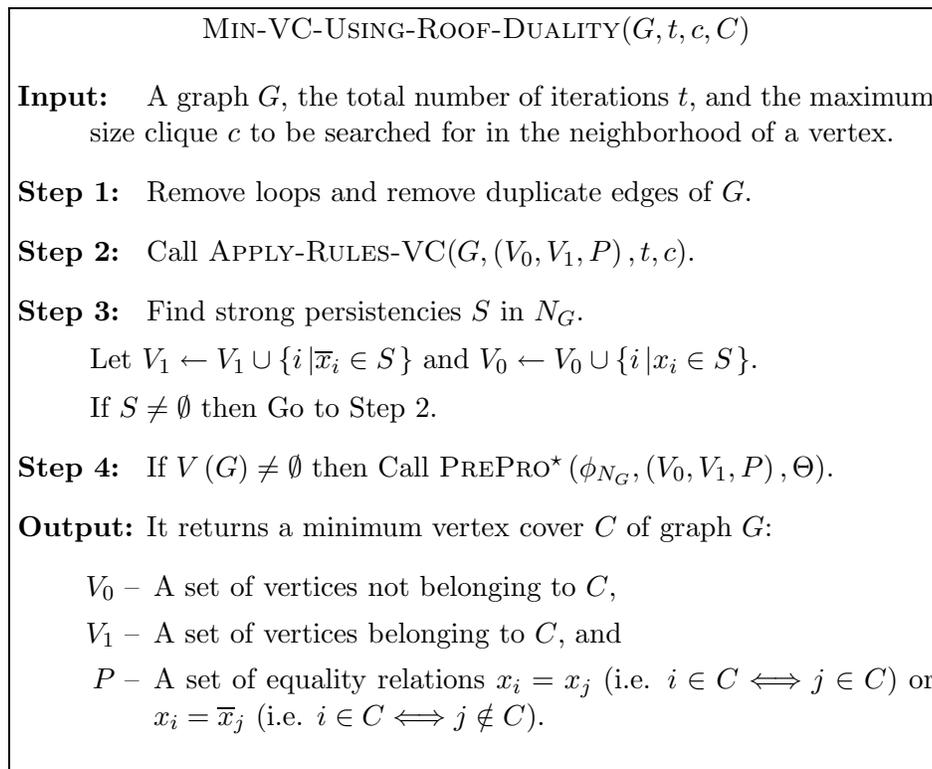


Figure 10.5: Exact method for MIN-VC using data reduction techniques and roof-duality.

Figure 10.6 describes the algorithm of the struction based algorithm to find the size  $\nu$  of the minimum vertex covers of graph  $G$ . At every step, this algorithm applies simplification rules as much as possible (by calling routine APPLY-RULES-VC), and (when required) is followed by an iteration of the struction algorithm (by calling routine STRUCTION). This sequence is repeated until the reduced graph has no vertices left. The size of the minimum vertex cover  $c$  is updated along this iterative process.

The implementation of each iteration of the struction algorithm is similar in nature to that algorithm proposed by Alexe et al. [14], and which was named the *compactification+guided struction* method. SELECT-CENTER( $G$ ) will return a vertex  $i$  of  $G$  for which the application of struction has  $i$  as the central vertex having either (i) any non-increasing transformation of the vertex set or (ii) a minimum increase of the vertex set size after the transformation.

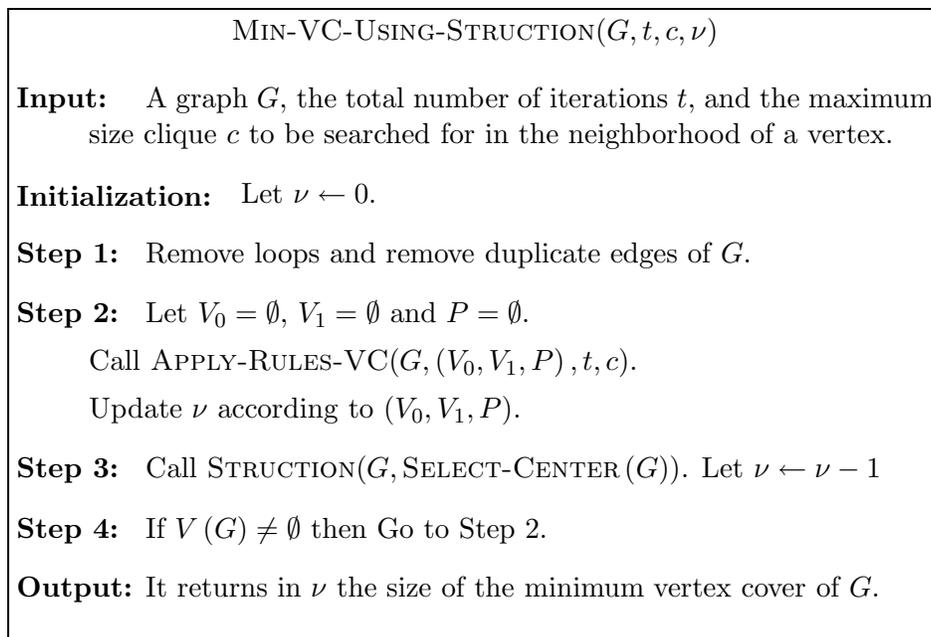


Figure 10.6: Exact method for MIN-VC using data reduction techniques and struction.

The data structure adopted in the implementation of both routines MIN-VC-USING-ROOF-DUALITY and MIN-VC-USING-STRUCTION is based on a ordered vector of vertices, where each entry  $i$  is a pointer to another ordered vector, which contains the neighbors  $N(i)$ .

The algorithms were implemented in C++, compiled using the Microsoft Windows 32-bit C/C++ Optimizing Compiler (version 12) for 80x86, and linked with the Microsoft Incremental Linker (version 6).

### 10.1.5 QUBO solvers considered for testing

In the sections that follow we shall analyze the computing times obtained with our algorithm implementations to find minimum vertex covers.

To compare different alternative approaches based on QUBO to solve MIN-VC we have considered several solvers, some of which are publicly available and have been considered top solvers for MAX-2-SAT and MAX-CUT.

The names of the solvers and their brief description are the following:

- MAXSATZ – A branch-and-bound solver for MAX-SAT developed by LI, Manyá and Planes (see [170]). At each node of the proof tree it transforms the formula

into an equivalent formula that preserves the number of unsatisfied clauses by applying some efficient refinements of unit resolution. It implements a lower bound computation method that consists of incrementing the lower bound by one for every disjoint inconsistent subset that can be detected by unit propagation. Moreover, the lower bound computation method is enhanced with failed literal detection. The variable selection heuristics takes into account the number of positive and negative occurrences in binary and ternary clauses. MAXSATZ and its variants are the best performing solvers in the un-weighted MAX-SAT category in the 2006 and 2007 MAX-SAT solvers evaluation.

- **TOOLBAR** – A branch-and-bound algorithm for Constraint Satisfaction Problems (CSP) developed by Givry, Heras, Larrosa and Schiex (see [92, 165]). The method maintains the state-of-the-art soft local consistency property EDAC\* during the search. The local consistency enforcement procedure is described in Larrosa and Schiex [165]. It uses specific data structures for efficient binary constraint updating. The min domain/max degree dynamic variable ordering heuristic is employed during the search. Domain values are dynamically ordered by increasing associated unary costs for value enumeration at each node of the search tree.
- **BIQMAC** – An SDP based branch-and-bound code that computes relaxations based on the intersection of the semidefinite relaxation with the set of triangle inequalities developed by Rendl, Rinaldi and Wiegele (see [206, 207]). It is publicly available online at [207].

Except for BIQMAC that runs on a Pentium 4, 3.6 GHz computer, all the other tests have run on the same computer system, which is based on an Intel Xeon 3.06 GHz, 3.5 GB RAM (operating system was Linux or Windows XP depending on the solver).

### 10.1.6 Minimum vertex cover of planar graphs

In view of the outstanding results obtained by applying QUBO preprocessing techniques to the minimum vertex cover problem in *random* planar graphs (see Section 7.7), we have tried to analyze the computational impact of using the new proposed methods

MIN-VC-USING-ROOF-DUALITY and MIN-VC-USING-STRUCTION, on this class and on various other classes of planar graphs.

Although the vertex cover problem is known to be NP-hard in the class of planar graphs ([105]), our computational experiments with a large collection of benchmark planar graphs indicate that finding minimum vertex covers in planar graphs is frequently tractable.

#### 10.1.6.1 Randomly generated planar graphs

In the computational experiments of this section we considered 36 random planar graphs generated by Rinaldi's ([211]) generator called RUDY. These planar graphs which have between 50 000 and 500 000 vertices, and planar densities varying between 10% and 90%. This group of problems has been called PVC RUDY (see Section 3.3).

The RUDY generator creates random planar graphs using three parameters: the number of vertices ( $n$ ), the planar density ( $d$ ), and a seed to initiate the pseudo-random generator. It is well known that any planar graph with  $n \geq 3$  must have  $m \leq 3(n - 2)$ , where  $m$  is the number of edges. Therefore, RUDY will create a planar graph having  $m = \lfloor 3(n - 2)d \rfloor$  edges.

In a first step, RUDY creates a random planar graph with maximum number of edges using an inductive procedure: for  $n = 3$  a triangle is created; for  $n > 3$  a random maximal planar graph of order  $n - 1$  is generated, and an additional vertex  $v$  is added to a random face  $f$ , and all edges from  $v$  to the extreme points of  $f$  are drawn. In a second step, all but  $m$  edges are removed randomly. An illustrative example of a planar graph having 50 vertices and 140 edges (97.2% planar density) generated in this way is given in Figure 10.7.

Table 10.1 presents the average computing times (over 3 experiments) for routine MIN-VC-USING-ROOF-DUALITY (with parameters set to  $i = 5$  and  $c = 4$ ) applied to the PVC RUDY benchmark. The largest planar graphs having 500 000 vertices and 90% planar density can be solved in 1.64 seconds on an average case. This result is significant faster when compared to the computing time needed by PREPRO\* to solve

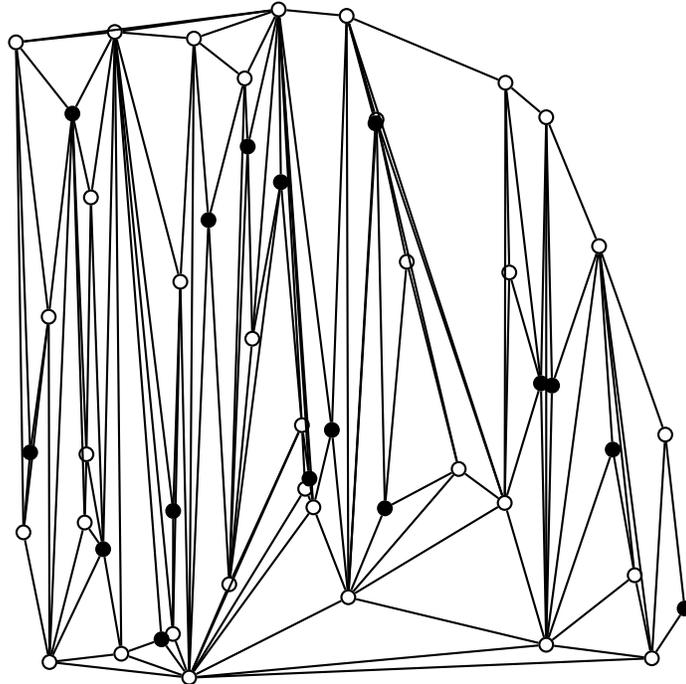


Figure 10.7: Planar graph with 50 vertices and 140 edges, generated by Gengraph using LEDA. The MIN-VC size is 34. A minimum vertex cover is indicated by the set of nodes with white color.

the same problems, without the simplification step (see Table 7.5). For instance, the largest planar graphs mentioned above are solved by PREPRO\* in 195.3 minutes on an average case. The differences observed in the computing times are explained by two factors:

- (i) The data structure used by MIN-VC-USING-ROOF-DUALITY is simple and can handle problems with sparse structure (i.e. having small density) much faster;
- (ii) The data reduction techniques employed in the routine APPLY-RULES-VC detect and apply problem simplifications at a much faster rate.

Table 10.1: Average computing times of MIN-VC-USING-ROOF-DUALITY applied to the PVC RUDY benchmark.

Vertices	Planar Density		
	10%	50%	90%
50 000	0.02 s	0.06 s	0.13 s
100 000	0.03 s	0.14 s	0.29 s
250 000	0.08 s	0.37 s	0.79 s
500 000	0.18 s	0.79 s	1.64 s

Table 10.2 demonstrates that the PVC RUDY problems are almost completely solved by the simplification routine APPLY-RULES-VC since the residual graphs are minuscule on an average case.

Table 10.2: Average number of variables of QUBOs solved by PREPRO\* within MIN-VC-USING-ROOF-DUALITY, when applied to the PVC RUDY benchmark.

<i>Vertices</i>	<i>Planar Density</i>		
	10%	50%	90%
50 000	0	3	39
100 000	0	0	30
250 000	0	57	0
500 000	0	54	4

The important conclusion of the previous experiments is the fact that the combination of a specialized data structure to a graph optimization problem, and subsequent application of fast data reduction techniques, can solve almost entirely the minimum vertex cover problem (and consequently the stability number) of the family of random planar graphs generated by RUDY.

It should be remarked the fact that the LEDA software package, introduced earlier in Section 3.3, also generates random planar graphs using the same approach to that one described above for the RUDY generator.

To further compare the computing times returned by the proposed algorithms (see also Section 7.7.2), we have used several state-of-the-art exact solvers for QUBO to verify how efficiently could they handle these particular problems. The results of Table 10.3 clearly indicate that the existent QUBO solvers can only efficiently solve minimum vertex cover problems of planar random graphs having up to a few hundred nodes. This fact is in clear contrast with the computing times of the proposed algorithms that can handle in few seconds problems of hundreds of thousands of vertices.

In an anonymous letter, it was called to our attention that the solver Cplex 10.01 ([9]) for Mixed Integer Programs (MIP) is also able to solve (SIP) efficiently for planar graphs generated by RUDY or LEDA. For transparency of results, we remark the following points included in the letter:

- Cplex was able to find the optimum for all the considered planar graphs at the root node.

Table 10.3: Minimum vertex covers of planar graphs randomly generated by LEDA.

(a) Graph details.

Graph $G = (V, E)$	Vertices ( $ V $ )	Edges ( $ E $ )	Planar Density	MIN-VC ( $\nu(G)$ )
planar-rnd-100	100	284	97.3%	67
planar-rnd-150	150	435	98.0%	95
planar-rnd-200	200	582	98.0%	132

(b) Computing Times<sup>†</sup>.

MIN-VC Solver		planar-rnd-100	planar-rnd-150	planar-rnd-200
MAXSATZ	Optimality?	yes	yes	no
	Comp. Time	2.3 s	94.2 s	>3 600 s
	Branching Nodes	444 974	12 131 399	n/a
TOOLBAR	Optimality?	yes	yes	no (best is 132)
	Comp. Time	5.9 s	72.4 s	1 802.6 s
	Branching Nodes	166 570	932 275	28 922 630
BIQMAC	Optimality?	no (best is 67)	n/a	n/a
	Comp. Time	10 800 s	n/a	n/a
	Branching Nodes	3 037	n/a	n/a
PREPRO*	Optimality?	yes	yes	yes
	Comp. Time	0.1 s	0.1 s	0.2 s
	Branching Nodes	0	0	0
ROOFDUALVC	Optimality?	yes	yes	yes
	Comp. Time	<0.01 s	<0.01 s	0.02 s
	Branching Nodes	0	0	0
STRUCTION	Optimality?	yes	yes	yes
	Comp. Time	0.02 s	0.02 s	<0.01 s
	Struction Calls	4	2	5

<sup>†</sup>Obtained on an Intel Xeon 3.06GHz.

- The planar graphs having 250 000 vertices and planar density of 90% (50%) could be solved by CPLEX to optimality in 38 (9) seconds on an average case using a AMD 3100+ computer system;
- The planar graphs having 500 000 vertices and planar density of 90% (50%) could be solved by CPLEX to optimality in 85 (20) seconds on an average case using a AMD 3100+ computer system.

There is a time speedup improvement clearly favorable to Cplex 10.01 for this class of problems when compared to the PREPRO\* approach (see Table 7.5). It should be noted that this speedup difference may be due to the fact that Cplex applies “simpler” reduction techniques before solving the residual problem using the simplex method.

The performance of ROOFDUALVC , as illustrated in Table 10.1, finds the minimum vertex covers of planar graphs having 500 000 vertices and planar density of 90% (50%) in 1.6 (0.8) seconds on average, representing therefore a clear speed improvement over that returned by Cplex 10.01. The set of simple data reduction rules was able to remarkably reduce the size of the planar graphs to minuscule graphs before PREPRO being called.

#### 10.1.6.2 Two dimensional grid graphs

In this subsection, we consider the family of two dimensional grid graphs  $G_{m,n} = (V_{m,n}, E_{m,n})$  ( $m, n \in \mathbb{Z}^+$ ), such that

$$\begin{aligned} V_{m,n} &= \{(i, j) \mid 1 \leq i \leq m, 1 \leq j \leq n\} \\ E_{m,n} &= \{(i, j, u, v) \mid (i, j) \in V_{m,n}, (u, v) \in V_{m,n}, |i - u| + |j - v| = 1\} \end{aligned}$$

The number of vertices in graph  $G_{m,n}$  is  $mn$  and the number of edges is  $2mn - (m + n)$  (see an example in Figure 10.8).

**Theorem 10.3.** *The minimum vertex cover size of graph  $G_{m,n}$  is  $\lfloor \frac{mn}{2} \rfloor$ .*

*Proof.* It is simple to verify that  $S_{m,n} = \{(i, j) \in V_{m,n} \mid (i \bmod 2) + (j \bmod 2) = 1\}$  is a vertex cover of  $G_{m,n}$  having  $|S_{m,n}| = \lfloor \frac{mn}{2} \rfloor$ . Let us show that  $S_{m,n}$  is also a

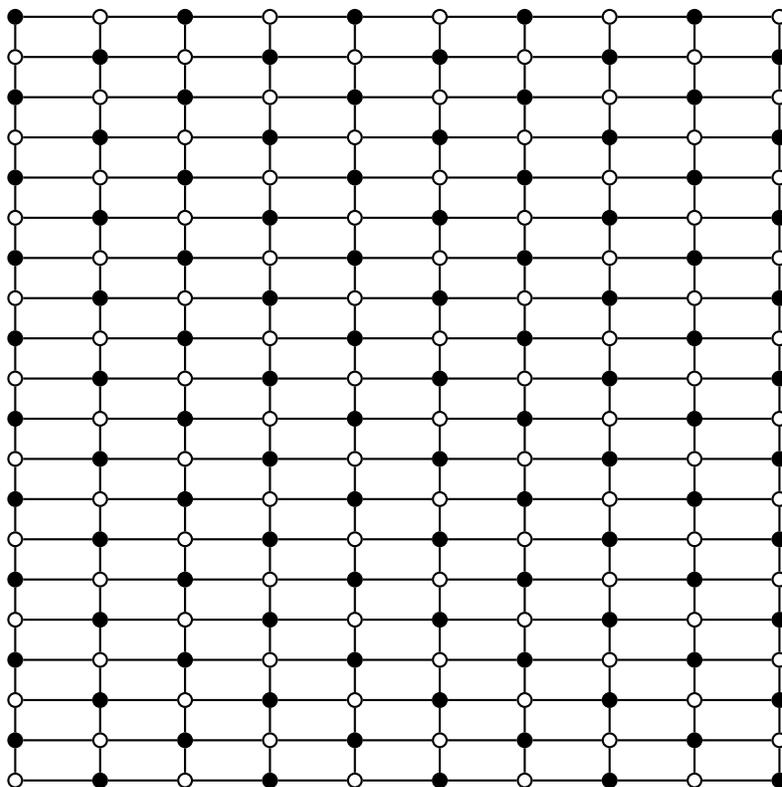


Figure 10.8: Graph  $G_{20,10}$ . The MIN-VC size is 100. A minimum vertex cover is indicated by the set of nodes with white (black) color.

minimum size vertex cover by using the fact that every vertex of graph  $G_{m,n}$  belongs to a cycle graph  $C_4$  (or square graph).  $G_{m,n}$  has  $mn - (m + n)$  such square graphs.  $S_{m,n}$  only requires that each square is covered exactly by 2 vertices. This result used together with the fact that each square graph needs at least 2 vertices to cover its four edges proves the claim.  $\square$

$V_{m,n} \setminus S_{m,n}$  is also a vertex cover of  $G_{m,n}$ , whose size is  $mn - \lfloor \frac{m}{2} \rfloor$ .

Whenever both  $m$  and  $n$  are odd then  $S_{m,n}$  is the unique minimum vertex cover of  $G_{m,n}$ . If either  $m$  or  $n$  is even then  $V_{m,n} \setminus S_{m,n}$  is also a minimum vertex cover of  $G_{m,n}$ .

Both  $S_{m,n}$  and  $V_{m,n} \setminus S_{m,n}$  are minimum and stable sets of graph  $G_{m,n}$ . Thus, any graph  $G_{m,n}$  is 2-colorable.

Table 10.4(b) contains the running times of the several solvers tested for finding the minimum vertex cover of graphs in  $G_{m,n}$  for small  $m$  and  $n$  values. Clearly, the roof-duality based algorithms are able to run much faster than the MAXSATZ or the BIQMAC solvers. TOOLBAR performs well in this family as well.

Table 10.4: Minimum vertex covers of 2 dimensional grid graphs.

(a) Graph details.

Graph			Vertices	Edges	Planar	MIN-VC
$G_{m,n}$	$m$	$n$	( $ V_{m,n} $ )	( $ E_{m,n} $ )	Density	( $\nu(G)$ )
2d-x10-y20	10	20	200	370	62.3%	100
2d-x10-y21	10	21	210	389	62.3%	105
2d-x11-y20	11	20	220	409	62.5%	110
2d-x11-y21	11	21	231	430	62.6%	115

(b) Computing Times<sup>†</sup>.

MIN-VC Solver		2d-x10-y20	2d-x10-y21	2d-x11-y20	2d-x11-y21
MAXSATZ	Optimality?	yes	yes	yes	yes
	Comp. Time	24.2 s	48.8 s	75.4 s	157.2 s
	Branching Nodes	1 867 089	3 492 333	5 065 394	10 128 770
TOOLBAR	Optimality?	yes	yes	yes	yes
	Comp. Time	0.3 s	0.3 s	0.4 s	0.4 s
	Branching Nodes	400	420	440	462
BIQMAC	Optimality?	yes	no (best is 105)	n/a	yes
	Comp. Time	8 990.0 s	10 810.3 s	n/a	6399.5 s
	Branching Nodes	527	519	n/a	463
PREPRO*	Optimality?	yes	yes	yes	yes
	Comp. Time	<0.01 s	<0.01 s	<0.01 s	<0.01 s
	Branching Nodes	0	0	0	0
ROOFDUALVC	Optimality?	yes	yes	yes	yes
	Comp. Time	<0.01 s	0.02 s	0.02 s	0.02 s
	Branching Nodes	0	0	0	0
STRUCTION	Optimality?	yes	yes	yes	yes
	Comp. Time	<0.01 s	<0.01 s	<0.01 s	<0.01 s
	Struction Calls	0	0	0	0

<sup>†</sup>Obtained on an Intel Xeon 3.06GHz.Table 10.5: MIN-VC-USING-ROOF-DUALITY computing times of minimum vertex covers of some  $G_{m,n}$  graphs.

Graph $G_{m,n}$		Vertices	Edges	MIN-VC	MIN-VC-USING-ROOF-DUALITY	
$m$	$n$	( $ V_{m,n} $ )	( $ E_{m,n} $ )	$\nu(G_{m,n})$	Rules+PREPRO Time	Total Time
333	333	110 889	221 112	55 444	2.6 s	2.6 s
333	666	221 778	442 557	110 889	22.1 s	709.5 s
333	1 000	333 000	664 667	166 500	35.2 s	1 575.3 s
666	666	443 556	885 780	221 778	51.2 s	2 782.3 s
666	1 000	666 000	1 330 334	333 000	85.6 s	6 185.3 s

In Table 10.5 we perform a scalability test for larger graphs in this family by using the algorithm MIN-VC-USING-ROOF-DUALITY. The results show that roof-duality is determinant in solving this type of problems efficiently.

### 10.1.6.3 Regular graphs consisting of hexagons

A family of regular graphs  $H_{m,n}$  ( $m, n \in \mathbb{Z}^+$ ) consisting of a grid of  $m \times n$  hexagons is considered in this subsection. The number of vertices in graph  $H_{m,n}$  is  $2(mn + m + n)$  and the number of edges is  $3mn + 2(m + n) - 1$  (see an example in Figure 10.8).

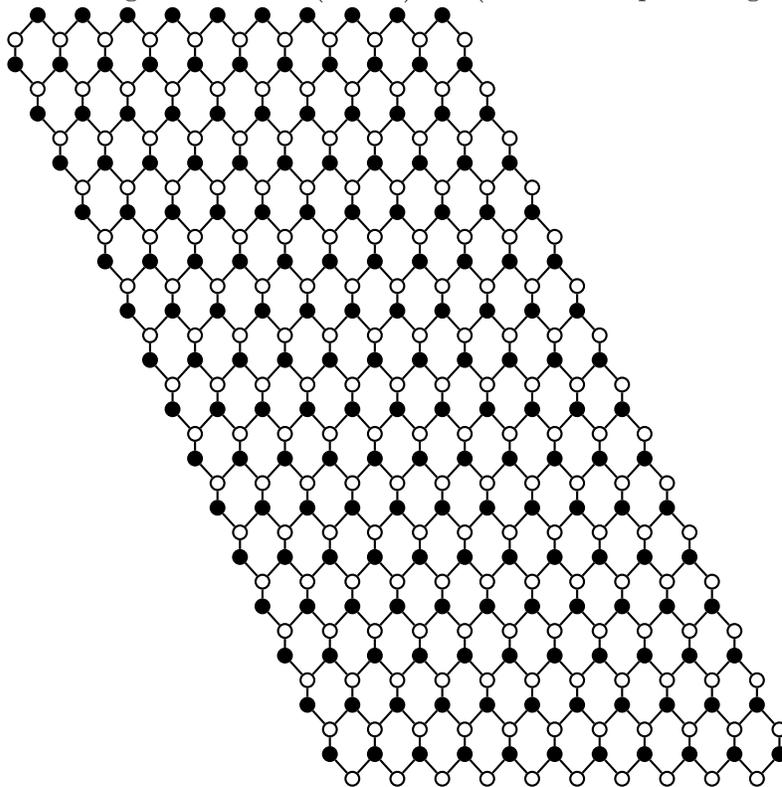


Figure 10.9: Graph  $H_{15,10}$ , generated by Gengraph. The MIN-VC size is 175. A minimum vertex cover is indicated by the set of nodes with white color.

**Theorem 10.4.** *The minimum vertex cover size of graph  $H_{m,n}$  is  $mn + m + n$ .*

*Proof.*  $H_{m,n}$  consists of contiguous hexagons. Every vertex cover of  $H_{m,n}$  must therefore include three vertices to cover all edges of each hexagon. Let us build a minimum vertex cover  $S_{m,n}$  by including on it all vertices in even positions of each hexagon, considering that the first position is the top node, while the other positions are determined clockwise

in increments of one (see MIN-VC of Figure 10.8). Since each hexagon is covered exactly by three vertices, then  $S_{m,n}$  is a the minimum size vertex cover of  $H_{m,n}$ .  $\square$

The complement set  $V(H_{m,n}) \setminus S_{m,n}$  is also a vertex cover of  $H_{m,n}$ , whose size is also  $mn + m + n$ . Therefore, the complement set of  $S_{m,n}$  is a minimum vertex cover of  $H_{m,n}$  as well. Both  $S_{m,n}$  and  $V(H_{m,n}) \setminus S_{m,n}$  are minimum vertex covers and maximum stable sets of graph  $H_{m,n}$ . Any graph  $H_{m,n}$  is therefore 2-colorable.

Table 10.6: Minimum vertex covers of regular planar graphs consisting of hexagons (generated by GenGraph).

(a) Graph details.

Graph			Vertices	Edges	Planar	MIN-VC
$H_{m,n}$	$m$	$n$	$( V(H_{m,n}) )$	$( E(H_{m,n}) )$	Density	$(\nu(G))$
c6-x10-y11	10	11	262	371	47.6%	131
c6-x10-y12	10	12	284	403	47.6%	142
c6-x11-y11	11	11	286	406	47.7%	143
c6-x11-y12	11	12	310	441	47.7%	155

(b) Computing Times<sup>†</sup>.

MIN-VC Solver		c6-x10-y11	c6-x10-y12	c6-x11-y11	c6-x11-y12
MAXSATZ	Optimality?	yes	yes	yes	no (best is 155)
	Comp. Time	436.1 s	1 962.7 s	2 416.4 s	10 456.4 s
	Branching Nodes	29 719 730	102 836 185	146 499 053	603 106 724
TOOLBAR	Optimality?	yes	yes	yes	yes
	Comp. Time	0.4 s	0.5 s	0.5 s	0.7 s
	Branching Nodes	524	568	572	620
BIQMAC	Optimality?	no (best is 131)	n/a	n/a	n/a
	Comp. Time	10 861.3 s	n/a	n/a	n/a
	Branching Nodes	65	n/a	n/a	n/a
PREPRO*	Optimality?	yes	yes	yes	yes
	Comp. Time	<0.01 s	0.02 s	0.02 s	<0.01 s
	Branching Nodes	0	0	0	0
ROOFDUALVC	Optimality?	yes	yes	yes	yes
	Comp. Time	<0.01	<0.01 s	0.02 s	<0.01 s
	Branching Nodes	0	0	0	0
STRUCTION	Optimality?	yes	yes	yes	yes
	Comp. Time	0.02 s	0.02 s	0.02 s	0.02 s
	Struction Calls	0	0	0	0

<sup>†</sup>Obtained on an Intel Xeon 3.06GHz.

The key to solve the minimum vertex cover of this type of graphs is to apply weak persistency from roof-duality. Table 10.7 shows that MIN-VC-USING-ROOF-DUALITY can compute MIN-VC for very large graphs. As in the previous example, MAXSATZ and BIQMAC struggle to find the minimum vertex cover in this family (see Table 10.6).

Table 10.7: MIN-VC-USING-ROOF-DUALITY computing times of minimum vertex covers of some  $H_{m,n}$  graphs.

Graph $G_{m,n}$		Vertices	Edges	MIN-VC	MIN-VC-USING-ROOF-DUALITY
$m$	$n$	$( V_{m,n} )$	$( E_{m,n} )$	$\nu(H_{m,n})$	Computing Time*
111	111	25 086	37 406	12 543	0.7 s
111	222	49 950	74 591	24 975	2.4 s
222	111	49 950	74 591	24 975	2.3 s
222	222	99 456	148 739	49 728	6.7 s

#### 10.1.6.4 Regular graphs consisting of triangles

A family of regular graphs  $T_s$  ( $s \in \mathbb{Z}^+$ ,  $s \geq 2$ ) consisting of a grid of triangles (see Figure 10.10). The overall shape of the planar straight-line embedding of graph  $T_s$  is close to an equilateral triangle whose base (line) is defined by  $s + 1$   $C_3$  cycles. The number of vertices in graph  $T_s$  is  $(s + 1) \frac{s}{2}$  and the number of edges is  $3(s - 1) \frac{s}{2}$ .

From the computational experiments that were carried out on this family, we conjecture that the minimum vertex cover size of graph  $T_s$  is

$$\nu(T_s) \cong \begin{cases} -1 + \lfloor (s + 1) \frac{s}{3} \rfloor, & s = 3, 5, \\ \lfloor (s + 1) \frac{s}{3} \rfloor, & s \in \{2, 4\} \cup \{i \in \mathbb{Z}^+ \mid i \geq 6\}, \end{cases}$$

which corresponds to about  $\frac{2}{3}$  of the number of nodes.

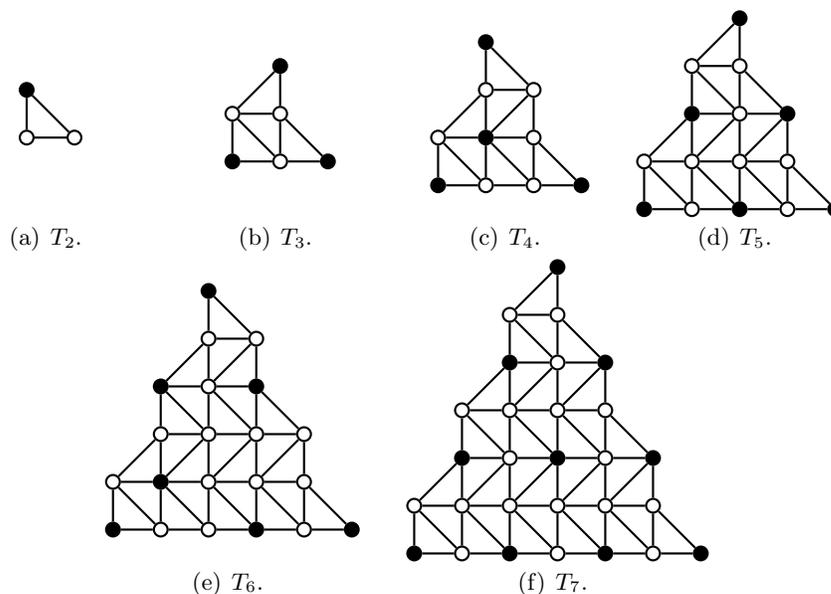


Figure 10.10: Family of regular graphs  $T_s$  ( $s = 2, \dots, 7$ ) consisting of triangles generated by Gengraph. A minimum vertex cover is indicated by the set of nodes with white color.

Table 10.8: Minimum vertex covers of regular planar graphs consisting of triangles (generated by GenGraph).

(a) Graph details.

Graph		Vertices ( $ V(T_s) $ )	Edges ( $ E(T_s) $ )	Planar Density	MIN-VC ( $\nu(G)$ )
$T_s$	$s$				
triangle-13	13	91	234	87.6%	60
triangle-14	14	105	273	88.3%	70
triangle-15	15	120	315	89.0%	80

(b) Computing Times<sup>†</sup>.

MIN-VC Solver		triangle-13	triangle-14	triangle-15
MAXSATZ	Optimality?	yes	yes	yes
	Comp. Time	9.4 s	88.0 s	867.8 s
	Branching Nodes	1 788 452	14 716 869	125 051 145
TOOLBAR	Optimality?	yes	yes	no (best is 80)
	Comp. Time	32.4 s	83.4 s	1 800 s
	Branching Nodes	1 059 826	2 984 400	41 578 397
PREPRO*	Optimality?	yes	yes	yes
	Comp. Time	1.1 s	4.9 s	24.7 s
	Branching Nodes	1 480	6 792	30 287
ROOFDUALVC	Optimality?	yes	yes	yes
	Comp. Time	0.8 s	5.5 s	21.2 s
	Branching Nodes	1 292	9 540	29 788
STRUCTION	Optimality?	yes	yes	yes
	Comp. Time	<b>0.1</b> s	<b>0.4</b> s	<b>0.6</b> s
	Struction Calls	4	4	6

<sup>†</sup>Obtained on an Intel Xeon 3.06GHz.

Table 10.8 indicates that STRUCTION is the fastest approach to solve problems in the triangles family. MAXSATZ and TOOLBAR clearly take considerably longer time to solve these problems than the roof-duality based solvers.

#### 10.1.6.5 A family of planar graphs with small diameter

The *diameter* of a graph is the length of the longest shortest path distance between any two vertices of the graph.

A family of regular planar graphs  $D_k$  ( $k \in \mathbb{Z}^+$ ) having low diameter and consisting of a “long” grid of triangles is considered in this subsection (see Figure 10.12; note that the straight-line embedding is not planar). The number of vertices in graph  $D_k$  is  $3k + 1$  and the number of edges is  $9k - 3$ .

**Theorem 10.5.** *The minimum vertex cover size of graph  $D_k$  is  $2k + 1$ .*

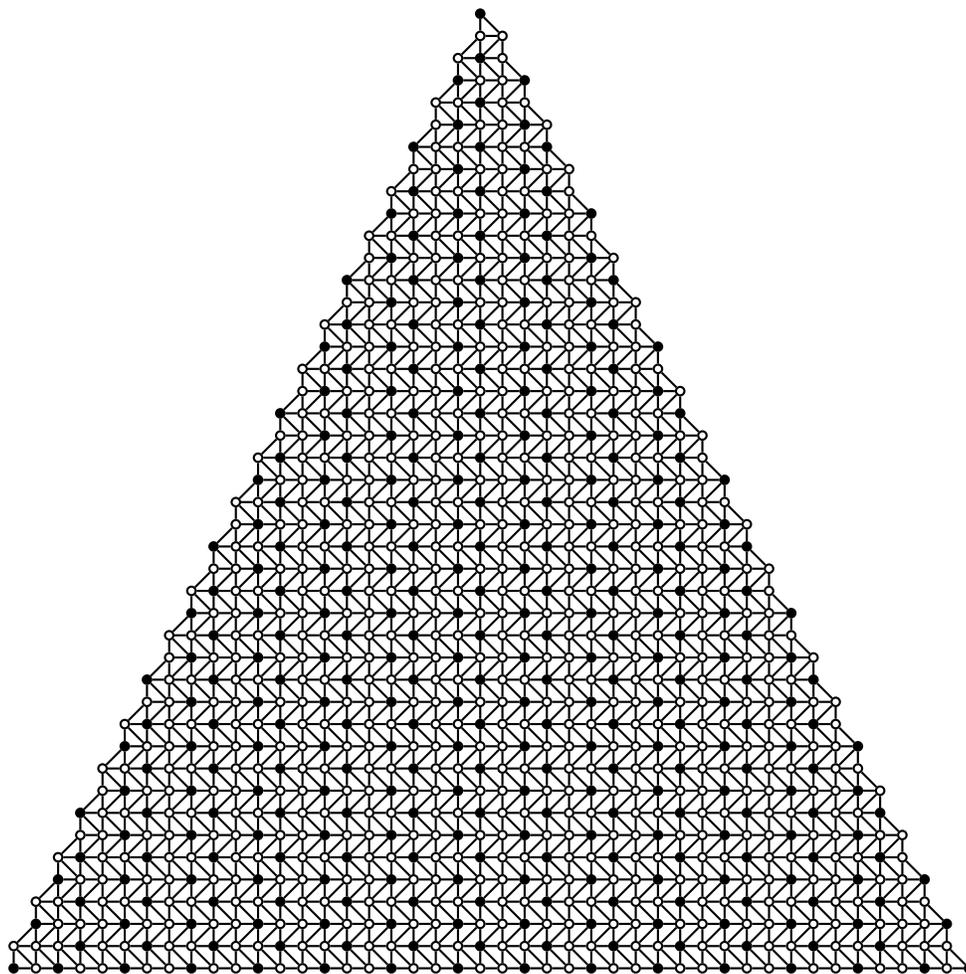


Figure 10.11: Regular graph  $T_{44}$  consisting of triangles with 990 vertices and 2838 edges, generated by Gengraph. The MIN-VC size is 660. A minimum vertex cover is indicated by the set of nodes with white color.

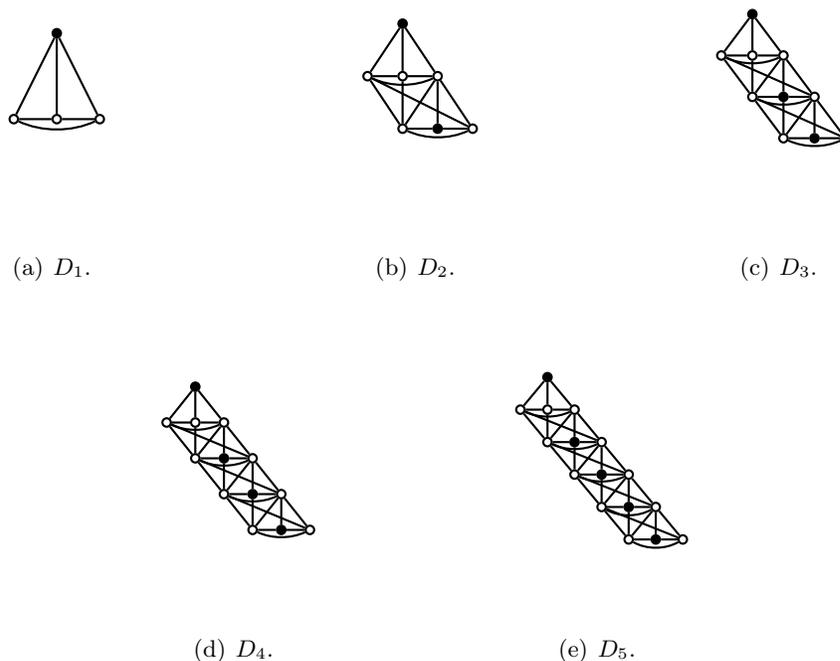


Figure 10.12: Family of regular graphs  $D_s$  ( $s = 1, \dots, 5$ ) with small diameter generated by Gengraph. A minimum vertex cover is indicated by the set of nodes with white color.

*Proof.* We shall prove this result by induction. Clearly,  $\nu(D_1) = 3$ . Also, from the construction procedure of this family of problems, the size of the maximum stable sets of  $D_k$  and  $D_{k+1}$  differ exactly by one. Assuming that  $\nu(D_k) = 2k + 1$ , or similarly that  $\alpha(D_k) = (3k + 1) - \nu(D_k) = k$ , then  $\alpha(D_{k+1}) = \alpha(D_k) + 1 = k + 1$ , thus proving that the induction step is also true.  $\square$

It is not difficult to show that there are at least two vertex covers of minimum size for each of the  $D_k$  ( $k \in \mathbb{Z}^+$ ) graphs.

STRUCTION and ROOFDUALVC have again better performance than the other contenders for this special family of MIN-VC. BIQMAC cannot handle the minimum vertex cover problem of the diameter graphs well.

#### 10.1.6.6 Planar graphs with Delaunay triangulations

Next we consider planar graphs with Delaunay triangulations. In this graph nodes are positioned at random in a unit square and the Delaunay triangulation is computed. To

Table 10.9: Minimum vertex covers of regular dense planar graphs with low diameter (generated by GenGraph).

(a) Graph details.

Graph		<i>Vertices</i> ( $ V(T_s) $ )	<i>Edges</i> ( $ E(T_s) $ )	<i>Planar</i> <i>Density</i>	<i>MIN-VC</i> ( $\nu(G)$ )
$T_s$	$s$				
diameter-25	25	76	222	100%	51
diameter-30	30	91	267	100%	61
diameter-35	35	106	312	100%	71
diameter-40	40	121	357	100%	81

(b) Computing Times<sup>†</sup>.

MIN-VC Solver		diameter-25	diameter-30	diameter-35	diameter-40
MAXSATZ	Optimality?	yes	yes	yes	yes
	Comp. Time	0.2 s	1.6 s	12.0 s	84.9 s
	Branching Nodes	40 002	255 699	1 633 003	10 446 094
TOOLBAR	Optimality?	yes	yes	yes	yes
	Comp. Time	1.1 s	6.9 s	76.8 s	337.6 s
	Branching Nodes	41 428	198 113	1 851 856	7 467 758
BIQMAC	Optimality?	yes	yes	no (best is 71)	n/a
	Comp. Time	6 374.2 s	10 131.7 s	10 818.5 s	n/a
	Branching Nodes	3 465	4 189	759	n/a
PREPRO*	Optimality?	yes	yes	yes	yes
	Comp. Time	0.02 s	0.03 s	0.03 s	0.05 s
	Branching Nodes	0	0	0	0
ROOFDUALVC	Optimality?	yes	yes	yes	yes
	Comp. Time	<0.01 s	<0.01 s	<0.01 s	<0.01 s
	Branching Nodes	0	0	0	0
STRUCTION	Optimality?	yes	yes	yes	yes
	Comp. Time	<0.01 s	<0.01 s	<0.01 s	<0.01 s
	Struction Calls	0	0	0	0

<sup>†</sup>Obtained on an Intel Xeon 3.06GHz.

obtain a smaller number of edges, edges are deleted at random. Examples of Delaunay graphs are given in Figures 10.13 and 10.14.

The minimum vertex cover of graphs in this family are traditionally difficult to be found by any known approach. To our surprise one method clearly stand out in solving problems in this class.

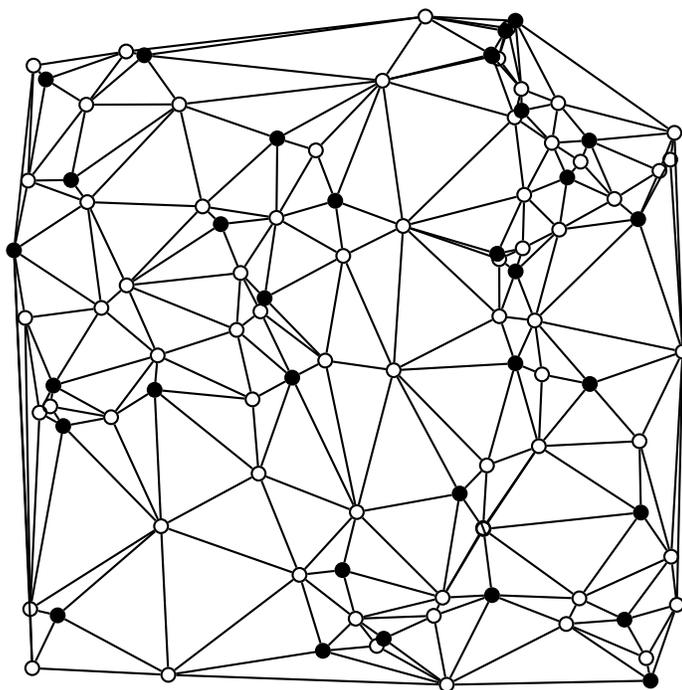


Figure 10.13: Planar graph with Delaunay triangulations having 100 vertices and 285 edges, generated by GenGraph. The MIN-VC size is 68. A minimum vertex cover is indicated by the set of nodes with white color.

The STRUCTION algorithm is by far the best approach to solve the MIN-VC problem of Delaunay graphs.

To see how STRUCTION scales well in solving graphs minimum vertex covers of planar graphs with Delaunay triangulations we have generated graphs (using program GenGraph) having up to 5 000 vertices. The results are listed in Table 10.11. For the larger graphs STRUCTION required 628 seconds and around 400 elementary struction calls to find an optimal minimum vertex cover.

To end this section we should emphasize the fact that all the other solvers including the commercial solvers for MIP would require hours to solve the 1 000 vertices problem.

Table 10.10: Minimum vertex covers of planar graphs with Delaunay triangulations (generated by GenGraph).

(a) Graph details.

Graph $G = (V, E)$	Vertices ( $ V $ )	Edges ( $ E $ )	Planar Density	MIN-VC ( $\nu(G)$ )
Delaunay-100	100	285	96.9%	68
Delaunay-110	110	313	96.6%	75
Delaunay-120	120	343	96.9%	82

(b) Computing Times<sup>†</sup>.

MIN-VC Solver		Delaunay-100	Delaunay-110	Delaunay-120
MAXSATZ	Optimality?	yes	yes	no
	Comp. Time	18.7 s	37.4 s	231.6 s
	Branching Nodes	3 100 866	6 157 790	33 639 395
TOOLBAR	Optimality?	yes	yes	yes
	Comp. Time	89.7 s	231.4 s	1 347.0 s
	Branching Nodes	3 552 148	8 763 289	42 847 188
BIQMAC	Optimality?	no (best is 68)	n/a	n/a
	Comp. Time	10 800 s	n/a	n/a
	Branching Nodes	1 663	n/a	n/a
PREPRO*	Optimality?	yes	yes	yes
	Comp. Time	31.3 s	3.9 s	1.5 s
	Branching Nodes	43 644	5 988	1 873
ROOFDUALVC	Optimality?	yes	yes	yes
	Comp. Time	31.7 s	0.9 s	1.2 s
	Branching Nodes	43 644	1 522	1 925
STRUCTION	Optimality?	yes	yes	yes
	Comp. Time	<b>0.03</b> s	<b>0.03</b> s	<b>0.05</b> s
	Struction Calls	15	9	12

<sup>†</sup>Obtained on an Intel Xeon 3.06GHz.

Table 10.11: Minimum vertex covers of planar graphs with Delaunay triangulations (generated by GenGraph).

Graph $G = (V, E)$	Vertices ( $ V $ )	Edges ( $ E $ )	Planar Density	MIN-VC ( $\nu(G)$ )	STRUCTION		
					Operations	Calls	Comp. Time
Delaunay-1000-1	1 000	2 979	99.5%	681	9 645	69	4.2 s
Delaunay-1000-2		2 978	99.5%	685	11 494	76	6.3 s
Delaunay-1000-3		2 978	99.5%	682	8 847	70	3.3 s
Delaunay-2000-1	2 000	5 976	99.7%	1 368	42 461	145	33.6 s
Delaunay-2000-2		5 975	99.7%	1 371	48 400	157	48.0 s
Delaunay-2000-3		5 979	99.7%	1 369	41 446	142	44.4 s
Delaunay-3000-1	3 000	8 976	99.8%	2 055	112 906	249	151.3 s
Delaunay-3000-2		8 975	99.8%	2 047	91 850	206	91.9 s
Delaunay-3000-3		8 975	99.8%	2 052	99 339	232	124.4 s
Delaunay-4000-1	4 000	11 976	99.8%	2 736	184 865	314	312.7 s
Delaunay-4000-2		11 975	99.8%	2 742	181 064	315	467.2 s
Delaunay-4000-3		11 972	99.8%	2 742	191 246	318	194.2 s
Delaunay-5000-1	5 000	14 976	99.9%	3 425	319 541	427	783.4 s
Delaunay-5000-2		14 981	99.9%	3 422	285 799	374	485.7 s
Delaunay-5000-3		14 972	99.9%	3 423	294 430	408	615.5 s

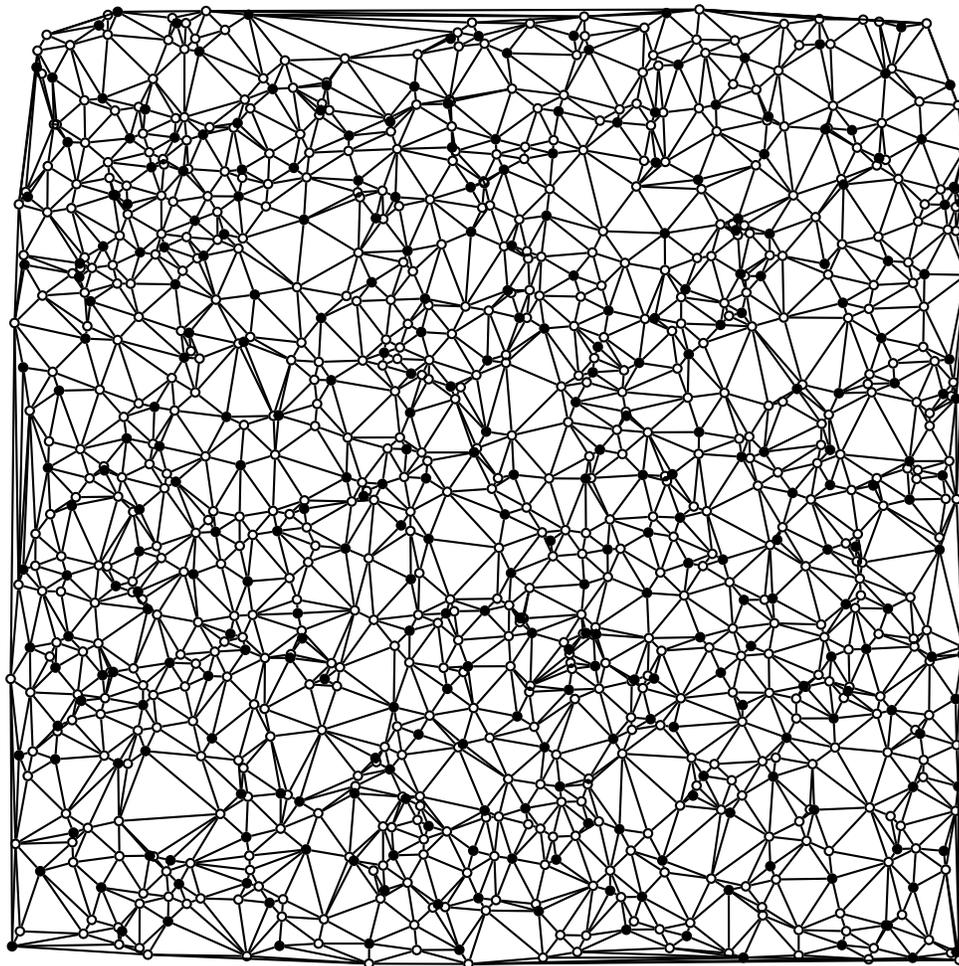


Figure 10.14: Planar graph with Delaunay triangulations having 1000 vertices and 2979 edges, generated by Gengraph. The MIN-VC size is 686. A minimum vertex cover is indicated by the set of nodes with white color.

### 10.1.7 Preventing Internet DDoS attacks

A *Denial of Service* (DoS) attack aims at disrupting services by consuming resources in networks, servers and hosts, with the malicious objective of preventing or degrading service. Resources that are typically depleted and clogged in such attacks include network bandwidth and computer CPU cycles.

The DoS attack is executed by sending bogus work in the form of junk traffic and service requests that tie up resources preventing a network system from operating in its normal mode. *Distributed* DoS (or DDoS in short) attacks, which forge the IP addresses source (called spoofing), are particularly severe, due to their concentrated force and difficulty to timely reestablish the normal operating status.

#### 10.1.7.1 Route-based distributed packet filtering

Route-based *Distributed Packet Filtering* (DPF) is a novel approach proposed by Lee and Park[169] which aims at preventing DDoS attacks with two goals. The first goal is to proactively preventing the spoofed IP packets from reaching their destination. The second goal is to reactively identifying the source of spoofed IP flows. The efficacy of their proposed method is evaluated in Internet *Autonomous Systems* (AS) topologies.

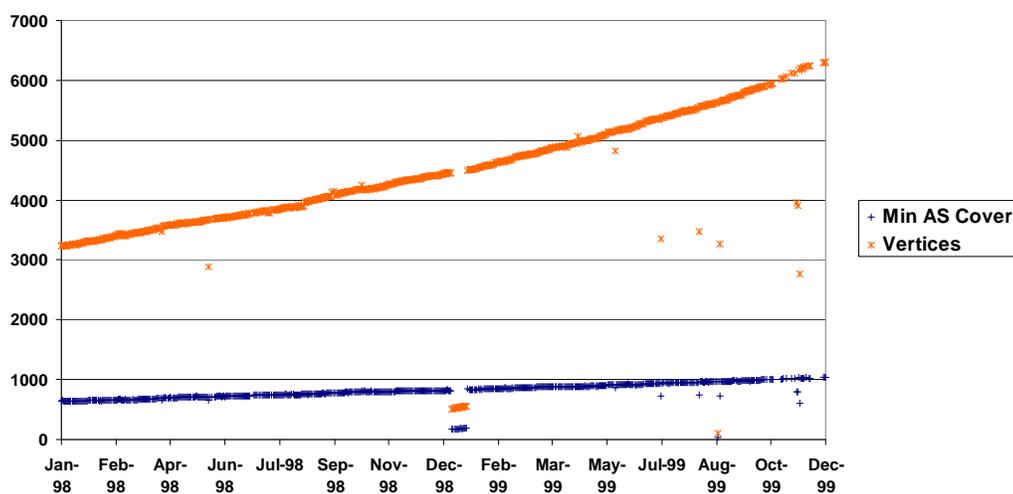
Route-based DPF uses routing information to determine if a packet arriving at a router is valid with respect to its inscribed source and destination addresses, given the constraints of reachability associated to the routing and network topology.

A single AS can only have a limited impact with respect to identifying and discarding forged IP flows. On the other extreme case, if all ASs implement route-based DPF then no spoofed IP flows can escape, but this case is not much different from that system which uses perfect ingress filtering.

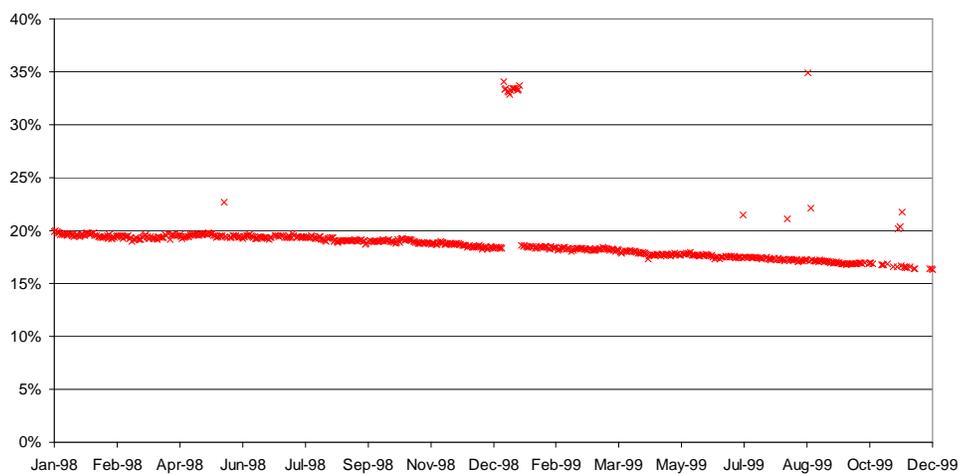
The main strength of route-based DPF lies in the fact that with partial coverage or deployment (about 18% of the Internet AS topologies according to [169]) a synergistic filtering effect is achieved whose collective filtering action proactively prevents spoofed IP flows from reaching other ASs.

Table 10.12: Minimum ASes covers of NLANR routing data ([10]).

<i>Graphs</i>	Number	735
	Avg. Vertices	4180.2
	Avg. Edges	7777.8
<i>Computing Time to Find the Minimum Vertex Covers</i>	Total	1920 s
	Average	2.6 s
	St. Dev.	1.9 s
	Minimum	0.0 s
	Maximum	10.2 s
	1st Quartile	0.8 s
	Median	2.3 s
3rd Quartile	3.7 s	



(a) Minimum ASes cover versus all ASes.



(b) ASes percentage in the minimum vertex cover.

Figure 10.15: Minimum ASes cover for 15 months of daily NLANR routing data.

The NLANR [10] goal was to characterize the behavior of High Performance Connection (HPC) networks. Network measurements are essential for assessing performance issues, identifying and locating problems (malfunctions, bottlenecks, inefficiencies, incompatibilities, etc.) in ultrafast research networks and in high-speed international links NLANR created the Network Analysis Infrastructure (NAI) that establish tools and methods for the collection of network measurement data and multiple analyzes.

NLANR gathers routing data that includes data from Border Gateway Protocol (BGP) routing tables, which reflect the transit relationships between individual Autonomous Systems (ASes) at a given point in time. We have obtained daily routing data from NLANR [10] to analyze the minimum ASes cover sizes of the routing network. The base routing data was retrieved from route-views.oregon-ix.net (Oregon Exchange BGP Route Viewer), and extends from 8 November 1997 (ASmap.19971108.879009857.gz) to 16 March 2001 (ASmap.20010316.984735601.gz), which corresponds to 735 days.

Table 10.12 lists the results. The ROOFDUALVC algorithm is able to find the optimum in every case taking a total time of 1920 seconds. The average running time was 2.6 seconds per network. The daily time evolution of the minimum ASes cover comparison with the total number of ASes available is given in Figure 10.15(a). Clearly the number of ASes is increasing exponentially, while the minimum ASes cover is increasing at a linear rate. Figure 10.15(b) gives the percentage of ASes required for the minimum vertex cover (which are those ASes where the filter will be assigned). Towards the end, the filtering coverage required for this dataset was of about 16%.

### 10.1.8 Maximum independent set of real world graphs

Network models are frequently used to study and describe many practical situations arising in the society or in the nature. There is a clear inter-relationship between the social and the infrastructure networks. Some examples of such problems include the analysis of social networks, the analysis of the Internet topology, the analysis of telecommunications traffic, the analysis of proteins interactions, etc.

An important result just recently discovered is the fact that many graph models arising from practical problems have the common property of following a power law

degree distribution.

Let us consider an undirected graph  $G = (V, E)$  with vertex set  $V$  and edge set  $E$ . The *vertex degree*  $d_G(v)$  of a given vertex  $v$  is the number of edges incident to it. The probability that a vertex  $v$  has degree  $k$  ( $k = 0, \dots, n$ ) is

$$\text{Prob}[d_G = k] = \frac{|\{v \in V \mid d_G(v) = k\}|}{n}.$$

Power law graphs have a probability degree distribution given as

$$\text{Prob}[d_G = k] \sim k^{-\lambda},$$

where  $\lambda$  is a positive parameter intrinsically associated to the graph.

Combinatorial optimization is an important field used to study the properties of power-law graphs. Some examples of this fact include the finding of minimum vertex covers to prevent internet worm attacks (see Section 10.1.7), and the finding of “large” clusters of proteins (which have a strong interaction between each other) to study which proteins are more relevant for the cells survival (see Section 10.2.2.1).

There is some practical evidence that combinatorial optimization in power law graphs originated from real-world applications (where typically  $1 < \lambda < 4$ ) is easier than in the case of general graphs. Ferrant et al. [96] shown however that the minimum vertex cover and the minimum dominating set problems are NP-hard for graphs having  $\lambda > 0$ , and MAX-Clique is NP-hard for graphs with  $\lambda > 1$ .

Table 10.13 lists a series of graphs that exhibit power-law degrees. The vertices and edges information is listed in this table, as well as a reference to the location where the graph was obtained from.

Table 10.14 gives the minimum vertex cover size and the maximum clique size for each one of the power-law graph examples. All the computing times of Table 10.14 were obtained on an Pentium M 1.6 GHz, 760 MB of RAM that runs Windows XP.

The minimum vertex cover has been determined by the BB solver (see Section 9.3) with the probing and coordinance option turned on for heavily presolving the problem

Table 10.13: Real world graphs.

<i>Graph</i>	<i>References</i>	<i>Vertices</i>		<i>Edges</i>	
		<i>Number</i>	<i>Designation</i>	<i>Number</i>	<i>Designation</i>
itdk0304_rlinks	[5]	192 244	Internet routers	609 066	links between routers
bgp_tables	[4, 93]	17 446	ASes from BGP tables	40 805	links between ASes
bgp_updates	[4, 93]	17 417	ASes from BGP updates	42 484	links between ASes
skitter	[4, 93]	9 204	ASes from skitter measurements	28 959	links between ASes
whois	[4, 93]	7 485	ASes from RIPE WHOIS	56 949	links between ASes
yeast	[6, 31]	1 870	proteins of yeast <i>Saccharomyces cerevisiae</i>	2 203	protein-protein interactions
hpy2000	[8]	1 570	proteins of pathogen <i>Helicobacter Pylori</i>	1 403	protein-protein interactions
geom	[181, 35]	7 343	co-authors in comp. geometry	11 898	author's collaborations
DEAauthors	[228]	1 853	co-authors in DEA papers	2 717	author's collaborations
Erdos1	[7]	509	co-authors of Paul Erdős	1 551	author's collaborations
UsPowerGrid	[225]	4 941	generators, transformers, etc.	6 594	power lines
USAir97	[35]	332	airports	2 126	air traffic links

Table 10.14: Combinatorics of real world graphs.

Graph	Minimum Vertex Cover		MAX-Clique	
	Size	B&B Comp. Time	Size	DEPTH-FIRST Comp. Time
itdk0304_rlinks	75 094	14 419.3 s	17	9 774.37 s
bgp_tables	2 648	114.7 s	17	0.6 s
bgp_updates	2 744	133.4 s	17	0.7 s
skitter	1 830	12.3 s	24	204.6 s
whois	2 289	6.7 s	58	244.8 s
yeast	626	0.1 s	6	2.4 s
hpy2000	204	<0.1 s	3	0.6 s
DEAauthors	927	0.1 s	10	1.6 s
Erdos1	237	<0.1 s	7	0.2 s
geom	3 083	0.8 s	22	175.6 s
UsPowerGrid	2 203	0.4 s	6	9.3 s
USAir97	149	0.1 s	22	<0.1 s

at the root node. The MIN-VC of the largest graph `itdk0304_rlinks` can be found much faster if the solver `ROOFDUALVC` is used instead. The initial data reduction by method `APPLY-RULES-VC` reduces the number of vertices (resp. edges) from 192 244 (resp. 609 066) to 2 074 vertices (resp. 5 392 edges) in 2.3 seconds. Then the `PREPRO*` method is applied to this reduced graph. The number of variables of this QUBO reduces further from 2 074 variables to 279 vertices in 3.0 seconds. This residual problem of 279 vertices is then solved in 1 617 seconds (using simply roof-duality as a bound for cutoff the search tree). The overall time of `ROOFDUALVC` was therefore considerably smaller than the 9 774 s required by BB.

The maximum clique size has been determined by the enumerative `DEPTH-FIRST` solver after presolving the problem (i.e. by removing first those vertices belonging to small cliques).

## 10.2 Clustering

The models considered in this section consider the problem of partitioning a given set of objects into several groups so that objects belonging to the same set have some affinity property. This process is frequently called *Clustering*, where the property used for grouping (or dividing) is traditionally based on a similarity (or dissimilarity) function between any two objects.

Clustering based on graph cuts has already been applied (see e.g. [99, 171]). A 2-parameter hierarchical clustering approach based on graph balancing is proposed and tested in Section 10.2.1.

Section 10.2.2 considers the use of QUBO to greedily partition a graph into either maximum independent sets (which leads to a graph coloring) or maximum cliques.

### 10.2.1 A QUBO model to 2-partitioning

A set of  $n$  objects is partitioned into 2 groups in such a way that penalties are imposed for 2 objects if they are:

- (i) Similar but assigned to different groups;
- (ii) Non-similar but assigned to the same group.

Let us define  $x_i = 1$  (respectively 0) if object  $i$  is assigned to partition 1 (respectively 0). Objects  $i$  and  $j$  are in the *same group* if and only if  $x_i\bar{x}_j + \bar{x}_i x_j = 0$  (equivalently  $x_i = x_j$ ). Objects  $i$  and  $j$  are in *different groups* if and only if  $x_i x_j + \bar{x}_i \bar{x}_j = 0$  (equivalently  $x_i \neq x_j$ ).

It is assumed that the “distance”  $d(i, j)$  between objects  $i$  and  $j$  is known, and that this closeness metric is used to measure “similarity” between pairs of objects.

The threshold parameter  $\delta^+$  characterizes the maximum distance between objects so that those objects are considered similar. The threshold parameter  $\delta^-$  ( $\geq \delta^+$ ) characterizes the minimum distance between objects so that those objects are considered different (i.e. non-similar).

On the one hand, if  $d(i, j) \leq \delta^+$  and objects  $i$  and  $j$  are assigned to different groups then a *penalty cost* term  $c^+(i, j)$  is associated to this assignment. On the other hand if  $d(i, j) \geq \delta^-$  and objects  $i$  and  $j$  are assigned to the same group then a *penalty cost* term  $c^-(i, j)$  is associated to this assignment.

There is however no penalizing term for any type of assignment of objects ( $i$  and  $j$ ) which are within distance  $\delta^+ < d(i, j) < \delta^-$ .

Given these parameters and penalty terms, a 2-partitioning *penalty cost function*

$$f(x_1, \dots, x_n) = \sum_{\substack{1 \leq i < j \leq n \\ d(i,j) \leq \delta^+}} c^+(i, j) (x_i \bar{x}_j + \bar{x}_i x_j) + \sum_{\substack{1 \leq i < j \leq n \\ d(i,j) \geq \delta^-}} c^-(i, j) (x_i x_j + \bar{x}_i \bar{x}_j)$$

is defined to quantify the sum of penalties (i) and (ii) associated to the partition of  $n$  objects into 2 groups.

The  $n$  objects are assigned into one of the two groups by *minimizing* the penalty cost function  $f(x_1, \dots, x_n)$ . This is a *graph balancing* QUBO problem. Cartwright and Harary [81] characterized a signed graph as *balanced* if and only if its vertices could be separated into two mutually exclusive subsets such that each positive edge joins to vertices of the same subset and each negative edge joins points from different subsets. This theorem shows that our idea of clustering is related to the structural balance problem in signed graphs, since positive links between two objects represent objects that are close to each other, and negative links between to objects represent objects that are distant from each other.

### 10.2.1.1 Hierarchical clustering

We have seen a graph balancing approach to partition the set of objects into two clusters. If we keep partitioning each one of these clusters using the same approach until some defined *stopping criteria* is met, then we have characterized a hierarchical clustering algorithm. Possible examples of stopping criteria for this method are:

- Minimum average distance between any two objects in the cluster;
- Minimum maximum distance between any two objects in the cluster;
- Maximum number of clusters;
- Minimum number of objects in the cluster.

An example with 6 000 objects has been randomly generated. Each object  $i$  has a know position  $(x_{i,1}, x_{i,2})$  in the Euclidean space (see Figure 10.16). The distance

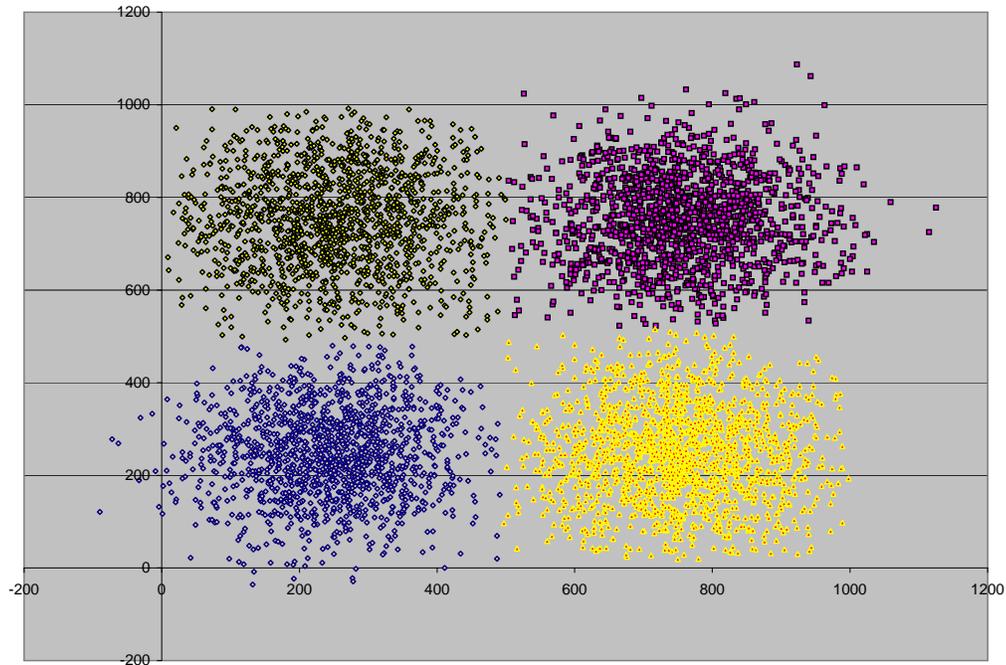


Figure 10.16: QUBO hierarchical clustering in the Euclidean space of 6 000 objects.

between any two objects  $i$  and  $j$  is simply the Euclidean distance, i.e.

$$d(i, j) = \sqrt{(x_{i,1} - x_{j,1})^2 + (x_{i,2} - x_{j,2})^2}.$$

The average distance between any two points is 488 and the corresponding standard deviation is 236.

The Hierarchical Clustering Algorithm (HCA) based on QUBO has been implemented and tested on this example. The stopping criterion that we use in this example is the minimum average distance requirement of 200 within the cluster. The parameters and the penalty coefficients of the cost function are

$$\begin{aligned} \delta^+ &= 100, \\ \delta^- &= 500, \\ c^+(i, j) &= \delta^+ - d(i, j) \quad \text{for all pairs } (i, j) \quad \text{and} \\ c^-(i, j) &= d(i, j) - \delta^- \quad \text{for all pairs } (i, j). \end{aligned}$$

HCA found four clusters displayed with different colors in Figure 10.16. The QUBO method used to 2-partitioning (i.e. to minimize the total penalty) was a steepest-descent heuristic introduced in Section 6.3. Using a Pentium M, 1.6 GHz, 760 MB of RAM and running Windows XP the total time to find the clusters was 12 seconds.

### 10.2.2 Greedy graph coloring and partitioning

In this section we consider the problem of clustering a set of objects  $V$  represented as a graph  $G = (V, E)$ . We consider two basic alternative algorithms:

- The *Greedy Graph Coloring Algorithm* (GGCA) recursively applies the following 2 steps:
  - 1 Find a *maximum stable set*  $S$  of graph  $G$  and assign a *color* to its elements;
  - 2 Remove the subgraph of  $G$  induced by  $S$ . Repeat 1 if  $G \neq \emptyset$ .
- The *Greedy Clique Partitioning Algorithm* (GCPA) recursively applies the following 2 steps:
  - 1 Find a *Maximum Clique*  $C$  of graph  $G$  and assign a *group* to its elements;
  - 2 Remove the subgraph of  $G$  induced by  $C$ . Repeat 1 if  $G \neq \emptyset$ .

Given graph  $G = (V, E)$  and its complement graph  $\overline{G}$ , it is well known that

$$\begin{aligned} \text{MAX-Clique}(G) &= \text{MAX-Stability}(\overline{G}) \text{ and} \\ \text{MAX-Stability}(G) &= |V| - \text{MIN-VC}(G). \end{aligned}$$

Thus, finding a *greedy graph coloring* in  $G$  is equivalent to find a *greedy clique partitioning* in  $\overline{G}$ .

We have implemented these two algorithms, each one adopting a different algorithmic approach:

- GGCA uses the MIN-VC-USING-ROOF-DUALITY algorithm described in Figure 10.5 as a basic solver to find a maximum stable set of the residual graph;

- GCPA uses the DEPTH-FIRST enumerative approach described in Figure 9.1 as a basic solver to find a maximum clique of the residual graph.

In spite of the fact that both algorithms are greedy, it should be noted that at every step either a *maximum* stable set (in GGCA is used) or a *maximum* clique (in GCPA is used) will be computed.

#### 10.2.2.1 The protein–protein interaction map of Helicobacter Pylori

According to Jeong et al. [31], the most *highly* connected proteins in the cell are the most important for its survival. Finding large groups of proteins with a large number of interacting between each other is therefore an important question to address.

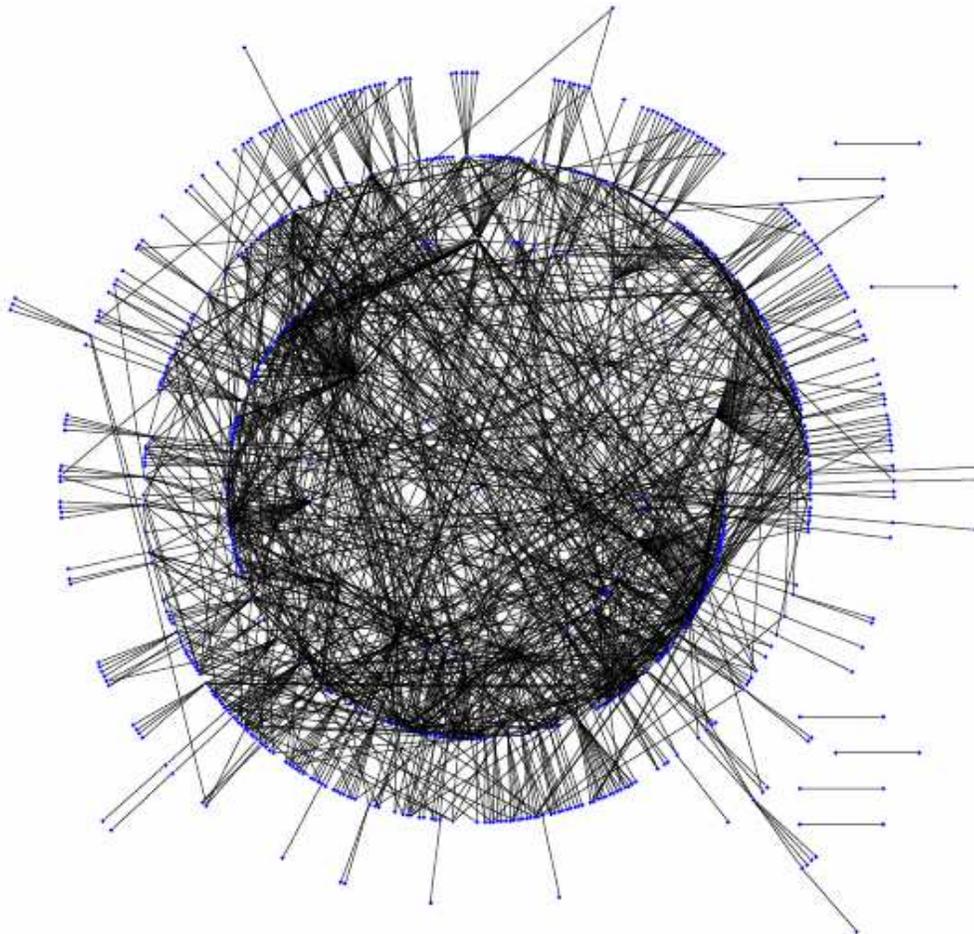


Figure 10.17: The protein–protein interaction map of the H. Pylori produced by Balasundaram et al. [28].

Table 10.15: Combinatorics of the H. Pylori map.

Vertices	859	
Edges	1403	
MIN-VC	204	< 1 sec
MAX-Clique	3	< 1 sec
MAX-2-distance-Clique	56	< 1 sec
MAX-3-distance-Clique	101	13 min
MAX-4-distance-Clique	290	2 min

Table 10.16: Large stable sets of the H. Pylori map.

<i>Color</i>	<i>Type</i>	<i>Stable Set</i>	<i>Vertices</i>	<i>Edges</i>	<i>Time</i>	<i>Cumulative Time</i>
1	Optimal	655	179	333	< 0.01 s	< 0.01 s
2	Optimal	130	65	112	< 0.01 s	< 0.01 s
3	Optimal	44	28	34	< 0.01 s	< 0.01 s
4	Optimal	18	7	5	< 0.01 s	< 0.01 s
5	Optimal	10	0	0	< 0.01 s	< 0.01 s

Figure 10.17 shows the protein–protein interaction map of the H. Pylori obtained from Balasundaram et al. [28]. How to find highly connected proteins in this map? One approach is to compute a  $k$ -distance clique, i.e. a set of vertices which are at most at distance  $k$  of each other. Obviously, a 1-distance clique is a clique.

Table 10.15 lists several combinatorial numbers for the H. Pylori map. Namely, MIN-VC is 204 and the MAX-Clique size is simply 3. The maximum 2-distance clique size is 56 and therefore far more interesting as far as protein connectivity concerns.

All the computing times given in this section were obtained by an Pentium M, 1.6 GHz, 760 MB of RAM and running Windows XP.

The maximum 3-distance clique (of size 101) has been found using the *enumerative* procedure for the first time (see [28]). It took about 13 minutes of computing time to find this set. The maximum 4-distance clique has been found for the 1st time as well, but by using the *B&B* solver (with roof-duality and heavy preprocessing) to find a maximum stable set in the complement graph. It took about 2 minutes to find this set.

The next question that we address is how to find large clusters of non-interacting proteins within the H. Pylori map. Table 10.16 lists the results of our implementation of GGCA, which gives a partition of the proteins into five stable sets. The largest stable set has 655 proteins. After removing this set, the largest stable set left contains 130

Table 10.17: Large 2–distance cliques of the H. Pylori map.

<i>Order</i>	<i>Type</i>	<i>2D-Clique</i>	<i>Vertices</i>	<i>Edges</i>	<i>Time</i>	<i>Cumulative Time</i>
1	Optimal	56	667	11 327	1.0 s	1.0 s
2	Optimal	51	616	7 851	0.5 s	1.5 s
3	Optimal	32	584	6 345	0.5 s	1.9 s
4	Optimal	25	559	5 560	0.4 s	2.3 s
5	Optimal	24	535	4 518	0.3 s	2.6 s
6	Optimal	24	511	3 930	0.2 s	2.9 s
7	Optimal	18	492	3 474	0.3 s	3.1 s
...	...	...	...	...	...	...
106	Optimal	2	0	0	0.0 s	5.3 s

Table 10.18: Large 3–distance cliques of the H. Pylori map.

<i>Order</i>	<i>Type</i>	<i>3D-Clique</i>	<i>Vertices</i>	<i>Edges</i>	<i>Time</i>	<i>Cumulative Time</i>
1	Optimal	101	622	30 800	803.2 s	803.4 s
2	Optimal	70	552	18 349	12.2 s	815.6 s
3	Optimal	41	511	13 314	1.4 s	817.0 s
4	Optimal	36	474	10 836	0.7 s	817.7 s
5	Optimal	34	440	7 944	0.5 s	818.2 s
6	Optimal	31	409	6 275	0.3 s	818.5 s
7	Optimal	26	383	4 609	0.2 s	818.7 s
...	...	...	...	...	...	...
80	Optimal	2	0	0	0.0 s	819.9 s

proteins and son on. The total time required for this calculation was minuscule.

The next point of discussion is how to find large clusters of interacting proteins for the H. Pylori map. Table 10.17 lists the outcome of GCPA to find 2–distance cliques. This method was able to decompose the initial set into 106 2–distance cliques of proteins. The first (and therefore largest) 2–distance clique has 56 elements. During the 2nd iteration, the largest possible 2–distance clique has 51 proteins. During the first 6 iterations, the method is able to identify 2–distance cliques of more than 20 elements. The total time required for GCPA was 5.3 seconds.

Table 10.18 lists the outcome of GCPA to find 3–distance cliques. During the first iteration the maximum 3–distance clique of size 101 is found after around 803 seconds. The second iteration finds a 3–distance clique of size 70 and it took approximately 12 seconds to show that this was the maximum possible. Overall, GCPA took 820 seconds to compute a proteins partition into 80 3–distance cliques.

Table 10.19 lists the outcome of GGCA to find 4–distance cliques for the H. Pylori map. Note that GGCA was run in the complement of the graph determined by the

Table 10.19: Large 4–distance cliques of the H. Pylori map.

<i>Order</i>	<i>Type</i>	<i>4D-Clique</i>	<i>Vertices</i>	<i>Edges</i>	<i>Time</i>	<i>CumulativeTime</i>
1	Optimal	290	434	62 185	125.9 s	126.4 s
2	Optimal	91	343	40 814	86.5 s	213.0 s
3	Optimal	57	286	29 618	34.2 s	247.3 s
4	Optimal	51	235	22 206	16.9 s	264.3 s
5	Optimal	35	200	16 842	8.0 s	272.5 s
6	Optimal	23	177	13 666	4.0 s	276.5 s
7	Optimal	18	159	11 229	2.1 s	278.8 s
...	...	...	...	...	...	...
51	Optimal	2	0	0	0.0 s	285.3 s

proteins within distance four of each other in the map. It found 51 4–distance cliques the largest having 290 members. The total time required by the method was 285 seconds.

### 10.3 A QUBO approach to image binarization

There is a large and growing interest in the field of document image analysis ([84, 230, 239, 238]), in order to acquire some relevant information for high level image processing.

One such method is the so-called *image binarization* or *thresholding*. This procedure segments an image into two classes: the foreground and the background. The foreground contains objects of interest, such as text, symbols, lines, or networks. Subsequently, human experts or symbol recognition and line vectorization programs are used to characterize those objects.

A possible application of this method occurs in the medical field, where a doctor with expertise in a given disease (e.g. lung cancer), could improve the accuracy of its diagnosis and prognosis by getting improved quality x-rays.

In this section a possible image binarization method is formulated as a QUBO, which provides an “optimal” binary image that minimizes the squared errors of every pixel with respect to the immediate neighbor pixels. Our method will only incorporate local information and therefore it only addresses part of the approach to solve these problems.

Before getting into further details on this specific problem, we should emphasize the fact that QUBO has been recently used as a very successful tool to solve certain

problems from vision (see e.g. [69, 70, 160, 203]). It is remarkable to see that roof-duality, persistency and probing together can provide in real time answers to many of these problems. Our preprocessing routine (i.e PREPRO) has been recently compared against another implementation, especially entailed for vision problems, developed by Kolmogorov et al. [159]. The preprocessing results on the report [159] were quite exciting for researchers interested in QUBO as well for researchers from the vision community.

Given a gray-scale image, represented as an  $m \times n$  array  $g$ , such that  $0 \leq g_{i,j} \leq 1$  ( $i = 1, \dots, m, j = 1, \dots, n$ ) we formulate a quadratic pseudo-Boolean function  $f$  by taking the  $L_2$ -norm of the difference between  $g_{i,j}$  ( $i = 1, \dots, m, j = 1, \dots, n$ ) from itself and the immediate eight neighbor weighted average assignment.

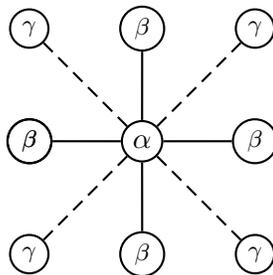


Figure 10.18: Image binarization weights used to define the neighborhood average assignment.

Figure 10.18 displays the nonnegative weights used to get the neighborhood average assignment for the pixel in the center. The pixel in the center has weight  $\alpha$ , its four immediate neighbors have the same weight  $\beta$ , and the four diagonal pixels have the same weight  $\gamma$ . If the center pixel corresponds to pixel  $(i, j)$ , then its neighborhood error is

$$e_{\alpha,\beta,\gamma}(i, j) = \left( \frac{\alpha x_{i,j} + \beta(x_{i-1,j} + x_{i,j-1} + x_{i+1,j} + x_{i,j+1}) + \gamma(x_{i-1,j-1} + x_{i+1,j-1} + x_{i-1,j+1} + x_{i+1,j+1})}{\alpha + 4\beta + 4\gamma} - g_{i,j} \right), \quad (10.2)$$

where  $x_{r,c} = 1$  if and only if the pixel  $(r, c)$  belongs to the foreground of the resulting binary image.

The sum of the squared-errors of all pixels in the image is provided by the quadratic function in binary variables

$$\begin{aligned}
f_{\alpha,\beta,\gamma}(\mathbf{x}) = & \\
& \sum_{i=2}^{m-1} \sum_{j=2}^{n-1} e_{\alpha,\beta,\gamma}^2(i, j) + \\
& \sum_{j=2}^{n-1} \left( \frac{\alpha x_{1,j} + \beta(x_{1,j-1} + x_{2,j} + x_{1,j+1}) + \gamma(x_{2,j-1} + x_{2,j+1})}{\alpha + 3\beta + 2\gamma} - g_{1,j} \right)^2 + \\
& \sum_{i=2}^{m-1} \left( \frac{\alpha x_{i,1} + \beta(x_{i-1,1} + x_{i,2} + x_{i+1,1}) + \gamma(x_{i-1,2} + x_{i+1,2})}{\alpha + 3\beta + 2\gamma} - g_{i,1} \right)^2 + \\
& \sum_{j=2}^{n-1} \left( \frac{\alpha x_{m-1,j} + \beta(x_{m-2,j} + x_{m-1,j-1} + x_{m-1,j+1}) + \gamma(x_{m-2,j-1} + x_{m-2,j+1})}{\alpha + 3\beta + 2\gamma} - g_{m,j} \right)^2 + \\
& \sum_{i=2}^{m-1} \left( \frac{\alpha x_{i,n-1} + \beta(x_{i,n-2} + x_{i-1,n-1} + x_{i+1,n-1}) + \gamma(x_{i-1,n-2} + x_{i+1,n-2})}{\alpha + 3\beta + 2\gamma} - g_{i,n} \right)^2 + \\
& \left( \frac{\alpha x_{1,1} + \beta(x_{1,2} + x_{2,1}) + \gamma x_{2,2}}{\alpha + 2\beta + \gamma} - g_{1,1} \right)^2 + \\
& \left( \frac{\alpha x_{m,1} + \beta(x_{m-1,1} + x_{m,2}) + \gamma x_{m-1,2}}{\alpha + 2\beta + \gamma} - g_{m,1} \right)^2 + \\
& \left( \frac{\alpha x_{1,n} + \beta(x_{1,n-1} + x_{2,n}) + \gamma x_{2,n-1}}{\alpha + 2\beta + \gamma} - g_{1,n} \right)^2 + \\
& \left( \frac{\alpha x_{m,n} + \beta(x_{m,n-1} + x_{m-1,n}) + \gamma x_{m-1,n-1}}{\alpha + 2\beta + \gamma} - g_{m,n} \right)^2.
\end{aligned} \tag{10.3}$$

Note that the four pixels in the four corners of the image, and the pixels appearing in the first and last rows and columns have different set sizes of neighbors, and hence their error function is somewhat different of the error (10.2), which corresponds to the pixels in the interior of the image.

Let us consider function

$$h_{i,j} \stackrel{\text{def}}{=} \begin{cases} \alpha + 4\beta + 4\gamma, & 2 \leq i \leq m-1, 2 \leq j \leq n-1, \\ \alpha + 2\beta + \gamma, & i = 1, j = 1 \vee i = m, j = 1 \vee i = 1, j = n \vee i = m, j = n, \\ \alpha + 3\beta + 2\gamma, & \text{otherwise,} \end{cases}$$

which returns the sum of the weights of the neighborhood of every pixel  $(i, j)$ .

Let us also also define the sets

$$N_{i,j}^{(v)} \stackrel{\text{def}}{=} \left\{ (r, c) \in \{i-1, \dots, i+1\} \times \{j-1, \dots, j+1\} \mid \begin{array}{l} 1 \leq r \leq m, 1 \leq c \leq n, \\ |i-r| + |j-c| = v, \end{array} \right\}$$

which contains the neighborhood pixels that are side-by-side to pixel  $(i, j)$  if  $v = 1$ , and that are contiguous in the same diagonal as pixel  $(i, j)$  if  $v = 2$ .

If the  $x^2(i, j)$  terms in (10.3) are linearized then we obtain the multilinear representation

$$\begin{aligned}
f_{\alpha, \beta, \gamma}(\mathbf{x}) = & \sum_{i=1}^m \sum_{j=1}^n g_{i,j}^2 \\
& \sum_{i=1}^m \sum_{j=1}^n \left( \frac{\alpha(\alpha - 2g_{i,j}h_{i,j})}{h_{i,j}^2} + \sum_{(r,c) \in N_{i,j}^{(1)}} \frac{\beta(\beta - 2g_{r,c}h_{r,c})}{h_{r,c}^2} + \sum_{(r,c) \in N_{i,j}^{(2)}} \frac{\gamma(\gamma - 2g_{r,c}h_{r,c})}{h_{r,c}^2} \right) x_{i,j} + \\
& \sum_{i=1}^m \sum_{j=1}^{n-1} 2 \left( \sum_{(r,c) \in N_{i,j}^{(1)}: c \geq j} \frac{\alpha\beta}{h_{r,c}^2} + \sum_{(r,c) \in N_{i,j+1}^{(1)}: c \leq j+1} \frac{\alpha\beta}{h_{r,c}^2} \right) x_{i,j} x_{i,j+1} + \\
& \sum_{i=1}^m \sum_{j=1}^{n-2} 2 \left( \frac{\beta^2}{h_{i,j+1}^2} + \sum_{(r,c) \in N_{i,j}^{(2)} \cap N_{i,j+2}^{(2)}} \frac{\gamma^2}{h_{r,c}^2} \right) x_{i,j} x_{i,j+2} + \\
& \sum_{i=1}^{m-1} \sum_{j=1}^n 2 \left( \sum_{(r,c) \in N_{i,j}^{(1)}: r \geq i} \frac{\alpha\beta}{h_{r,c}^2} + \sum_{(r,c) \in N_{i+1,j}^{(1)}: r \leq i+1} \frac{\alpha\beta}{h_{r,c}^2} \right) x_{i,j} x_{i+1,j} + \\
& \sum_{i=1}^{m-2} \sum_{j=1}^n 2 \left( \frac{\beta^2}{h_{i+1,j}^2} + \sum_{(r,c) \in N_{i,j}^{(2)} \cap N_{i+2,j}^{(2)}} \frac{\gamma^2}{h_{r,c}^2} \right) x_{i,j} x_{i+2,j} + \\
& \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} 2 \left( \sum_{(r,c) \in N_{i,j+1}^{(1)} \cap N_{i+1,j}^{(1)}} \frac{\alpha\gamma}{h_{r,c}^2} + \sum_{(r,c) \in N_{i,j}^{(1)} \cap N_{i+1,j+1}^{(1)}} \frac{\beta^2}{h_{r,c}^2} \right) x_{i,j} x_{i+1,j+1} + \\
& \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} 2 \left( \sum_{(r,c) \in N_{i,j+1}^{(1)} \cap N_{i+1,j}^{(1)}} \frac{\beta^2}{h_{r,c}^2} + \sum_{(r,c) \in N_{i,j}^{(1)} \cap N_{i+1,j+1}^{(1)}} \frac{\alpha\gamma}{h_{r,c}^2} \right) x_{i+1,j} x_{i,j+1} + \\
& \sum_{i=1}^{m-2} \sum_{j=1}^{n-2} 2 \frac{\gamma^2}{h_{i+1,j+1}^2} (x_{i,j} x_{i+2,j+2} + x_{i+2,j} x_{i,j+2}) + \\
& \sum_{i=1}^{m-1} \sum_{j=1}^{n-2} 2 \left( \frac{\beta\gamma}{h_{i,j+1}^2} + \frac{\beta\gamma}{h_{i+1,j+1}^2} \right) (x_{i,j} x_{i+1,j+2} + x_{i,j+2} x_{i+1,j}) + \\
& \sum_{i=1}^{m-2} \sum_{j=1}^{n-1} 2 \left( \frac{\beta\gamma}{h_{i+1,j}^2} + \frac{\beta\gamma}{h_{i+1,j+1}^2} \right) (x_{i,j} x_{i+2,j+1} + x_{i,j+1} x_{i+2,j})
\end{aligned} \tag{10.4}$$

corresponding to the quadratic pseudo-Boolean function  $f_{\alpha, \beta, \gamma}$ .

Since  $f_{\alpha, \beta, \gamma}(\mathbf{x})$  represents the sum of squared errors for a given binary “image”  $\mathbf{x}$  with respect to the original image  $g$ , then an optimal binarized image  $\mathbf{x}^*$  is one that

minimizes the total error, i.e.

$$f(\mathbf{x}^*) \equiv \min_{\mathbf{x} \in \mathbb{B}^{m \times n}} f_{\alpha, \beta, \gamma}(\mathbf{x}). \quad (10.5)$$

### 10.3.1 Computational results

In this section, we consider a one-pass heuristic approach to solve problem (10.5), i.e. the problem of finding a binary image  $\mathbf{x}$  that is reasonable close to a image  $\mathbf{x}^*$  that minimizes the total error to the original image defined by array  $g$ .

We have also tried some of the exact approaches presented in Chapter 9. The computing times obtained seemed however to not scale well for large size images in general.

The one-pass procedure adopted here is a probabilistic based method (see Section 6.1.2.2), i.e. a heuristic that at every iteration fixes the variable having the closest value in probability to a binary assignment of a local minimum of the function. In particular, the ONE-PASS-P-U algorithm was considered, which assumes that the partial derivatives  $\Delta_{i,j}$  of pixel  $(i, j)$  in  $f$  are uniformly distributed between  $L_{i,j}$  and  $U_{i,j}$ , which correspond respectively to the minimum and maximum of the 0–1 linear function  $\Delta_{i,j}$ .

A sparse data structure based on the network flow model has been adopted to provide the heuristic solution. It should be noted that  $f_{\alpha, \beta, \gamma}$  has  $mn$  binary variables and at most 24 nonzero quadratic terms per variable in its multilinear representation (10.4), thus  $f_{\alpha, \beta, \gamma}$  has at most  $24mn$  nonzero quadratic terms.

We considered 3 images for testing. The original images are shown respectively in option (a) of Figures 10.19, 10.20 and 10.21. The binarized image versions are shown on the same figures using different values for  $\alpha$ ,  $\beta$  and  $\gamma$ .

The tests were run on an Xeon 3.06 GHz, 3.5 GB RAM and Windows 32bit XP. The heuristic average computing times were 271 seconds, 41 085 seconds and 70 seconds, respectively for the image problems in Figures 10.19, 10.20 and Figure 10.21.

The computing times of the heuristics clear indicate that fast data structures specialized for this purpose are required. However, it is interesting to see visually the impact of the  $\alpha$ ,  $\beta$  and  $\gamma$  parameters in the binarized image.

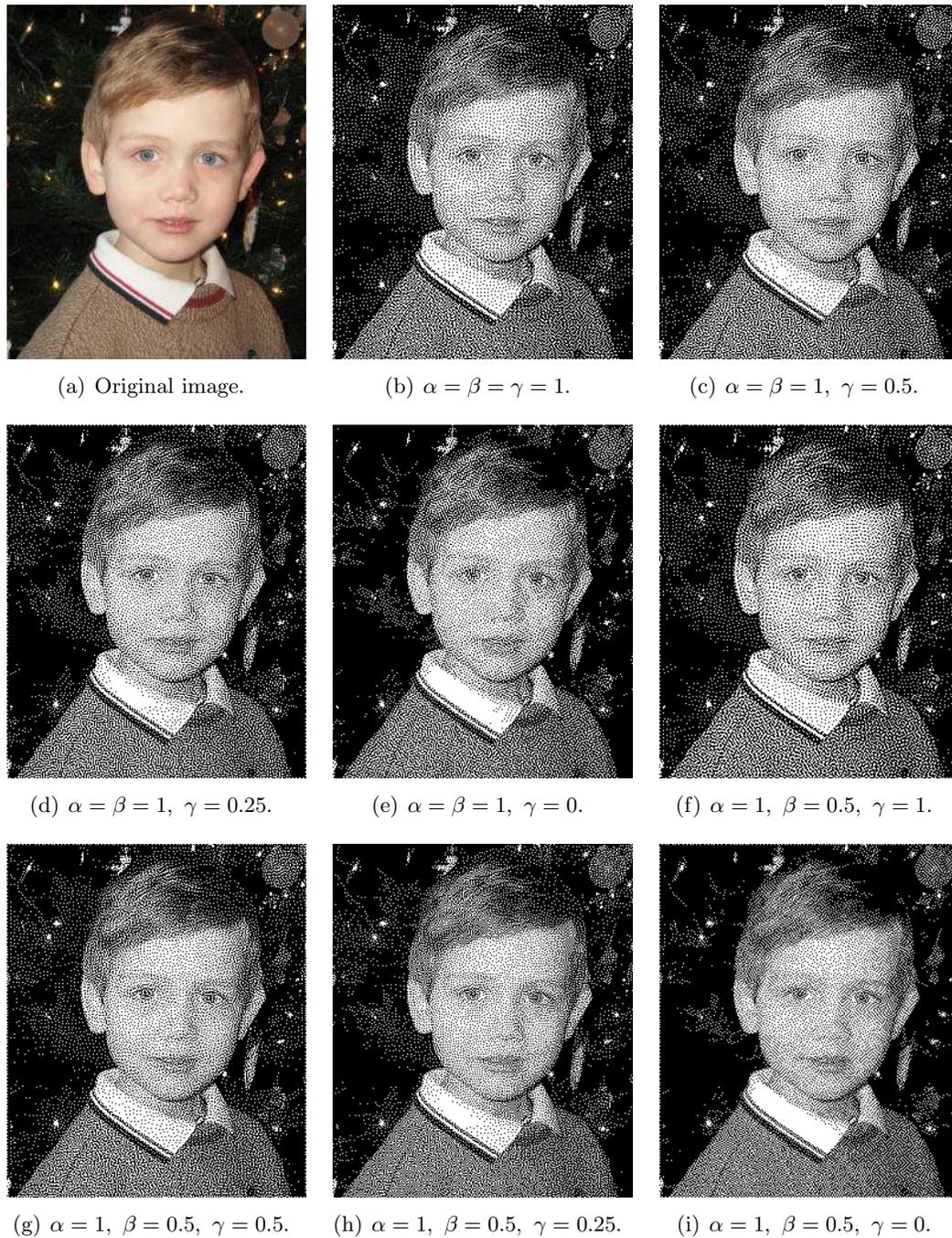


Figure 10.19: Image binarizations found by the one-pass heuristic applied to  $f_{\alpha,\beta,\gamma}$ . Original image is a 24-bit bitmap  $254 \times 300$  image of a child with a dark background.



Figure 10.20: Image binarizations found by the one-pass heuristic applied to  $f_{\alpha,\beta,\gamma}$ . Original image is a 24-bit  $1280 \times 755$  bitmap image of a project design of a house.

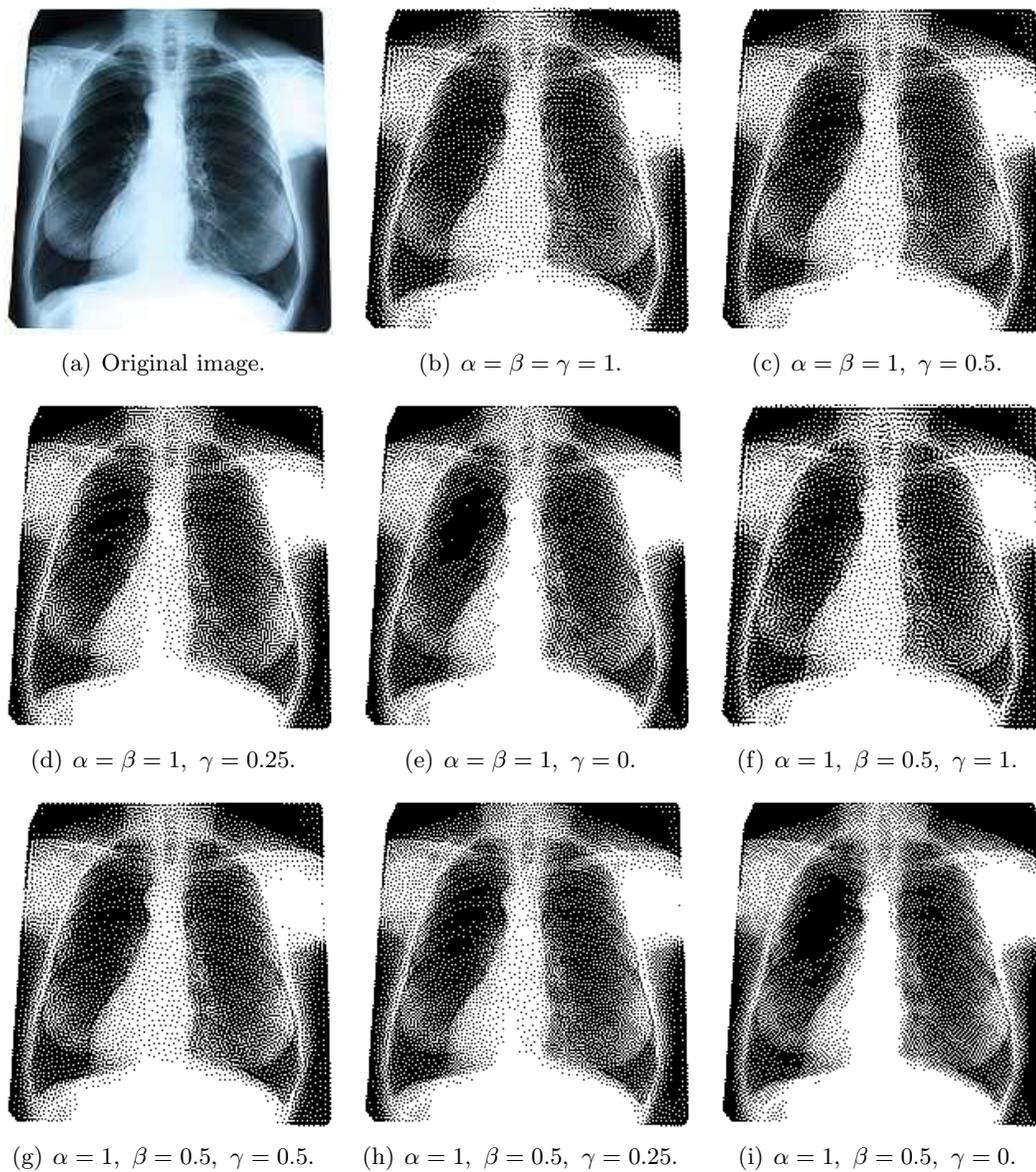


Figure 10.21: Image binarizations found by the one-pass heuristic applied to  $f_{\alpha,\beta,\gamma}$ . Original image is a 24-bit  $200 \times 199$  bitmap image of an x-ray.

Finally, we would like to mention that preprocessing (especially with the probing option) is in practice very effective in finding persistencies for these types of quadratic functions. In the future we plan to do a more detailed analysis of the various data reductions achieved by employing various preprocessing techniques (including PREPRO), as well as by looking at specialized data structures that enable the method to find solutions in real time.

## Chapter 11

### Conclusions

We have seen that QUBO is a suitable model to solve many different types of combinatorial optimization applications. In general, the current technology and algorithmic advances can routinely solve to optimality QUBO problems that have up to 100 variables. The proposed preprocessing techniques, together with the improved bounding methods can be determinant to solve to optimality special classes of optimization problems that have tens and hundreds of thousands of binary variables (e.g., via minimization from VLSI, MIN-VC of planar or power-law graphs, 2D Ising models, one-dimensional Ising chains, problems from vision).

The analysis of persistencies for pseudo-Boolean optimization in general is an interesting topic. We hope to investigate new algorithms to find further persistencies within higher degree representations (e.g. cubic posiforms) of the QUBO problem.

The focus of this dissertation was mostly driven to find the optimum of the QUBO problem. Heuristics should not be diminished since they are determinant to get good solutions for very large problems. The challenge in the future is (i) to define heuristics that have certain performance guarantees, and (ii) to propose very fast heuristics of good quality that can be used in other algorithms (e.g. for preprocessing). Soon enough, QUBOs of millions of variables will be created and the only possible way to handle those problems is by having efficient and effective heuristics at hand. These methods will require special fast data structures and probably even specific to the class of problem being solved (e.g. imaging problem).

In practice, we have demonstrated that bounding is still the most determining factor to solve QUBO efficiently. Improving the existent algorithms for bounding, and in particular the “squeezed” iterated roof-duality versions, can be crucial in solving a

wider range of problems.

The cubic–dual (i.e.  $C_3$ ) is an excellent bound for many classes of problems derived from real applications. In this dissertation we provided certain hints that were an attempt at computing  $C_3$  by means of combinatorial algorithms. We will continue to pursue the search of a combinatorial algorithm to compute  $C_3$ , which is going to be determinant to solve “super”–sparse QUBO (e.g. large 2D or 3D Ising models, problems from vision, combinatorics of large graphs).

$C_4$  gives a good bound for the minimum 3–partition problem. It is probably the first time where we have seen that  $C_4$  is clear advantageous over the  $C_3$  bound. This motivates us at studying further  $C_4$  and also to attempt at characterizing the next bound of the hierarchy, i.e.  $C_5$ .

Computationally, it is also a challenge to implement new network flow algorithms, and in particular if one would like to preprocess QUBO problems having hundreds of thousands of variables.

With this dissertation, we also hope that the researchers from other related fields (e.g. MAX–2–SAT, MAX–CUT, vision) realize that QUBO is a common framework that has been active for many years now. We will continue working with people with expertise in other fields to make sure that we both learn with each other, so that real problems can be solved in real time, if that can be possible.

Roof–duality is a key algorithm to solve many QUBOs derived from real world applications. Its success is due to several fronts. First it provides a bound computed by a maximum flow algorithm. Second it detects a unique maximal set of persistencies, making it possible to reduce the size of the problems. Third it is able to detect a well characterized decomposition, which can potentially result in the optimization of several smaller size independent QUBOs.

## Appendix A

### Test Problems Characteristics

Table A.1: QUBO problems of Beasley [37].

(a) Problems with 50, 100 and 250 variables.

<i>Sub-Family</i>	<i>Problem Number</i>	<i>Variables (n)</i>	<i>Density (d%)</i>	$\rho\%$	$p\%$	<i>Maximum (<math>\tau(f)</math>)</i>
B-50	1	50	8.82	35.72	1.86	2 098
	2	50	9.80	52.59	0.73	3 702
	3	50	10.78	57.11	1.44	4 626
	4	50	9.06	50.90	2.16	3 544
	5	50	10.69	51.46	1.78	4 012
	6	50	8.24	57.97	7.41	3 693
	7	50	10.12	57.75	1.83	4 520
	8	50	11.18	55.28	3.65	4 216
	9	50	9.96	53.44	3.63	3 780
	10	50	8.82	53.29	2.10	3 507
B-100	1	100	9.37	45.76	1.40	7 970
	2	100	9.74	51.51	1.17	11 036
	3	100	9.90	56.18	1.46	12 723
	4	100	9.60	49.77	1.49	10 368
	5	100	9.27	47.99	1.54	9 083
	6	100	10.30	50.74	1.89	10 210
	7	100	9.47	50.74	0.86	10 125
	8	100	9.94	49.51	0.60	11 435
	9	100	10.14	50.93	0.94	11 455
	10	100	9.80	53.72	1.54	12 565
B-250	1	250	9.92	49.81	0.47	45 607
	2	250	9.75	50.94	0.54	44 810
	3	250	9.84	52.70	0.49	49 037
	4	250	10.11	48.72	0.49	41 274
	5	250	10.00	50.74	0.32	47 961
	6	250	10.23	49.54	0.47	$\geq 41\,014$
	7	250	9.92	51.40	0.43	46 757
	8	250	9.69	47.81	0.39	$\geq 35\,726$
	9	250	10.10	51.91	0.34	48 916
	10	250	9.78	49.73	0.40	40 442

(b) Problems with with 500, 1000 and 2500 variables.

<i>Sub-Family</i>	<i>Problem Number</i>	<i>Variables (n)</i>	<i>Density (d%)</i>	$\rho\%$	$p\%$	<i>Maximum (<math>\tau(f)</math>)</i>
B-500	1	500	9.92	49.87	0.23	$\geq 116\,586$
	2	500	9.84	50.44	0.16	$\geq 128\,339^\ddagger$
	3	500	10.04	50.92	0.12	$\geq 130\,812$
	4	500	9.84	51.54	0.18	$\geq 130\,097$
	5	500	9.88	50.23	0.21	$\geq 125\,487$
	6	500	9.87	50.09	0.22	$\geq 121\,772^\ddagger$
	7	500	9.94	49.99	0.21	$\geq 122\,201$
	8	500	9.84	51.23	0.14	$\geq 123\,559$
	9	500	9.91	50.38	0.18	$\geq 120\,798$
	10	500	9.95	50.81	0.18	$\geq 130\,619$
B-1000	1	1000	9.90	50.76	0.21	$\geq 371\,438$
	2	1000	9.90	50.44	0.19	$\geq 354\,932$
	3	1000	9.95	50.55	0.21	$\geq 371\,236^\ddagger$
	4	1000	9.93	50.77	0.19	$\geq 370\,675^\ddagger$
	5	1000	9.91	50.53	0.20	$\geq 352\,760^*$
	6	1000	9.98	50.21	0.19	$\geq 359\,629^\ddagger$
	7	1000	9.89	50.58	0.19	$\geq 371\,193^\ddagger$
	8	1000	9.90	50.52	0.22	$\geq 351\,994^\ddagger$
	9	1000	9.91	50.29	0.17	$\geq 349\,337^\ddagger$
	10	1000	9.82	50.32	0.24	$\geq 351\,415$
B-2500	1	2500	9.94	50.57	0.04	$\geq 1\,515\,944^\ddagger$
	2	2500	9.90	50.53	0.04	$\geq 1\,471\,392^\ddagger$
	3	2500	9.89	50.24	0.04	$\geq 1\,414\,192^\ddagger$
	4	2500	9.90	50.55	0.04	$\geq 1\,507\,701^\ddagger$
	5	2500	9.90	50.53	0.04	$\geq 1\,491\,816^\ddagger$
	6	2500	9.88	50.46	0.04	$\geq 1\,469\,162^\ddagger$
	7	2500	9.92	50.48	0.04	$\geq 1\,479\,040^\ddagger$
	8	2500	9.89	50.43	0.04	$\geq 1\,484\,199$
	9	2500	9.92	50.45	0.04	$\geq 1\,482\,413^\ddagger$
	10	2500	9.90	50.51	0.04	$\geq 1\,483\,355^\ddagger$

<sup>†</sup>Solution reported first by Katayama and Narihisa [155].

<sup>‡</sup>Solution reported first by Merz and Freisleben [177].

\*Solution reported first by Palubeckis [187].

Table A.2: QUBO problems of Glover, Kochenberger and Alidaee [108].

<i>Sub-Family</i>	<i>Problem Number</i>	<i>Variables (n)</i>	<i>Density (d%)</i>	$\rho\%$	$p\%$	<i>Starting Seed</i>	<i>Maximum (<math>\tau(f)</math>)</i>
<i>A</i>	1	50	8.65	50.65	26.89	10	3 414
	2	60	9.21	56.02	20.25	10	6 063
	3	70	9.23	52.62	15.23	10	6 037
	4	80	9.62	54.51	13.12	10	8 598
	5	50	18.86	53.62	9.93	10	5 737
	6	30	40.00	53.48	8.10	10	3 980
	7	30	48.51	53.46	8.71	10	4 541
	8	100	6.14	53.23	16.53	10	11 109
<i>B</i>	1	20	98.42	4.09	4.26	10	133
	2	30	98.62	2.41	2.47	10	121
	3	40	98.97	1.62	1.65	10	118
	4	50	98.69	1.32	1.34	10	129
	5	60	98.93	1.06	1.07	10	150
	6	70	98.88	0.93	0.93	10	146
	7	80	98.89	0.79	0.80	10	160
	8	90	98.95	0.71	0.72	10	145
	9	100	99.05	0.66	0.67	10	137
	10	125	98.92	0.50	0.51	10	154
<i>C</i>	1	40	80.13	52.43	6.01	10	5 058
	2	50	62.29	52.84	7.00	70	6 213
	3	60	39.60	53.24	7.83	31	6 665
	4	70	29.81	52.67	9.48	34	7 398
	5	80	20.28	54.96	13.11	8	7 362
	6	90	9.99	53.92	21.65	80	5 824
	7	100	10.00	53.03	22.39	142	7 225
<i>D</i>	1	100	9.98	53.17	13.64	31	6 333
	2	100	20.53	47.87	6.84	37	6 579
	3	100	28.79	48.81	5.04	143	9 261
	4	100	40.40	49.32	3.82	47	10 727
	5	100	48.77	49.96	3.19	31	11 626
	6	100	59.56	50.41	2.21	47	14 207
	7	100	69.37	50.87	2.20	97	14 476
	8	100	78.93	50.12	1.88	133	16 352
	9	100	87.80	49.12	1.72	307	15 656
	10	100	98.93	51.05	1.56	1 311	19 102
<i>E</i>	1	200	9.67	49.42	11.06	51	16 464
	2	200	19.73	50.76	4.70	43	$\geq 23 395$
	3	200	29.40	49.28	3.40	34	$\geq 25 243$
	4	200	39.78	50.71	2.44	73	$\geq 35 594$
	5	200	49.53	50.67	1.96	89	$\geq 35 154$
<i>F<sub>1</sub></i>	1	500	9.96	49.76	2.99	137	$\geq 61 194$
	2	500	24.86	50.18	1.20	137	$\geq 100 161$
	3	500	49.62	50.17	0.57	137	$\geq 138 035$
	4	500	74.35	50.16	0.40	137	$\geq 172 771$
	5	500	99.01	50.10	0.31	137	$\geq 190 507$

Table A.3: QUBO problems of Glover, Kochenberger, Alidaee and Amini [109].

<i>Sub-Family</i>	<i>Problem Number</i>	<i>Variables (n)</i>	<i>Density (d%)</i>	$\rho\%$	$p\%$	<i>Maximum (<math>\tau(f)</math>)</i>
$G_1$	1	1000	9.89	47.69	1.50	$\geq 131\,456$
	2	1000	19.82	47.92	0.73	$\geq 172\,788$
	3	1000	29.64	47.85	0.51	$\geq 192\,565$
	4	1000	39.52	47.84	0.39	$\geq 215\,679$
	5	1000	49.49	47.91	0.30	$\geq 242\,367$
	6	1000	59.50	47.98	0.25	$\geq 243\,293$
	7	1000	69.38	47.97	0.22	$\geq 253\,590^\dagger$
	8	1000	79.23	47.93	0.19	$\geq 264\,268^\dagger$
	9	1000	89.10	47.92	0.17	$\geq 262\,658$
	10	1000	98.99	47.95	0.14	$\geq 274\,375^\dagger$

<sup>†</sup>Solution reported first by Amini et al. [18].

Table A.4: QUBO submodular problems of Glover, Alidaee, Rego and Kochenberger [107].

<i>Sub-Family</i>	<i>Problem Number</i>	<i>Variables (n)</i>	<i>Density (d%)</i>	$\rho\%$	$p\%$	<i>Maximum (<math>\tau(f)</math>)</i>
$F_2$	1	500	9.82	3.82	3.97	$\geq 4\,029$
	2	500	24.55	1.54	1.56	$\geq 2\,010$
	3	500	49.07	0.80	0.80	1 094
	4	500	73.72	0.52	0.52	685
	5	500	98.04	0.38	0.38	418
$G_2$	1	1000	9.79	2.01	2.05	$\geq 5\,216$
	2	1000	29.36	0.66	0.67	$\geq 2\,122$
	3	1000	49.04	0.39	0.40	$\geq 1\,272$
	4	1000	68.75	0.28	0.29	866
	5	1000	98.04	0.20	0.20	452

Table A.5: QUBO problems of Palubeckis and Tomkevičius [189].

<i>Sub-Family</i>	<i>Problem Number</i>	<i>Variables (n)</i>	<i>Density (d %)</i>	$\rho$ %	$p$ %	<i>Starting Seed</i>	<i>Maximum (<math>\tau(f)</math>)</i>
P-3000	1	3 000	49.76	50.01	0.07	31 000	$\geq 3\,931\,583^\dagger$
	2	3 000	79.58	50.03	0.04	32 000	$\geq 5\,193\,073^\dagger$
	3	3 000	79.60	50.01	0.04	33 000	$\geq 5\,111\,533^\dagger$
	4	3 000	99.50	50.02	0.03	34 000	$\geq 5\,761\,822^\dagger$
	5	3 000	99.50	49.99	0.03	35 000	$\geq 5\,675\,625^\ddagger$
P-4000	1	4 000	49.73	50.01	0.05	41 000	$\geq 6\,181\,830^\dagger$
	2	4 000	79.60	49.99	0.03	42 000	$\geq 7\,801\,355^\dagger$
	3	4 000	79.60	49.98	0.03	43 000	$\geq 7\,741\,685^\dagger$
	4	4 000	99.50	49.99	0.03	44 000	$\geq 8\,711\,822^\dagger$
	5	4 000	99.50	50.03	0.03	45 000	$\geq 8\,908\,979^\ddagger$
P-5000	1	5 000	49.74	50.01	0.04	51 000	$\geq 8\,559\,355$
	2	5 000	79.60	50.02	0.03	52 000	$\geq 10\,836\,019$
	3	5 000	79.61	49.96	0.02	53 000	$\geq 10\,489\,137$
	4	5 000	99.50	50.00	0.02	54 000	$\geq 12\,251\,874$
	5	5 000	99.50	50.03	0.02	55 000	$\geq 12\,731\,803$
P-6000	1	6 000	49.75	50.01	0.03	61 000	$\geq 11\,384\,976$
	2	6 000	79.59	49.99	0.02	62 000	$\geq 14\,333\,855$
	3	6 000	99.50	50.00	0.02	64 000	$\geq 16\,132\,915$

<sup>†</sup>Solution reported first by Palubeckis and Tomkevičius [189].

<sup>‡</sup>Solution reported first by Palubeckis [187].

Table A.6: Minimum values of the *Small* family QUBO problems.

Group ( $k$ )	$\bar{p}$	Variables ( $n$ )	Minimum Values ( $\nu(f)$ )				
			$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
1	0.40	25	-1 762	-1 567	-2 962	-6 314	-4 692
		50	-3 277	-10 118	-10 967	-17 383	-13 702
		75	-7 794	-16 368	-18 404	-27 004	-32 653
		100	-16 401	-24 763	-32 714	-47 654	-52 053
	0.55	25	-1 066	-748	-1 361	-2 800	-1 777
		50	-1 623	-3 920	-3 993	-5 459	-3 244
		75	-3 734	-5 842	-4 931	-6 626	-6 683
		100	-6 359	-6 615	-7 372	-10 296	$\leq -7 899$
	0.70	25	-674	-407	-711	-1 035	-686
		50	-917	-1 645	-1 444	-1 595	-1 083
		75	-1 682	-2 234	-1 678	-1 821	-1 764
		100	-2 640	$\leq -2 157$	$\leq -2 154$	-2 627	$\leq -1 979$
	0.85	25	-401	-241	-340	-358	-261
		50	-485	-660	-574	-548	-396
		75	-791	-901	-585	-553	-561
		100	-1 100	-815	-742	-847	-525
2	0.40	25	-998	-1 688	-3 224	-3 933	-3 556
		50	-4 634	-7 672	-13 405	-11 914	-13 234
		75	-10 944	-17 131	-19 358	-25 152	-33 169
		100	-13 277	-22 719	-37 422	-47 166	-63 481
	0.55	25	-616	-967	-1 478	-1 804	-1 019
		50	-2 294	-3 004	-5 092	-3 761	-3 637
		75	-5 025	-5 879	-5 686	-5 243	-6 422
		100	-5 028	-6 760	-8 243	$\leq -9 601$	$\leq -11 556$
	0.70	25	-384	-463	-657	-656	-363
		50	-1 198	-1 387	-1 834	-1 288	-1 256
		75	-2 347	-2 026	-1 692	-1 603	-1 616
		100	-1 948	$\leq -2 626$	$\leq -2 241$	-2 486	$\leq -2 312$
	0.85	25	-283	-299	-355	-279	-177
		50	-642	-629	-691	-448	-486
		75	-1 165	-732	-524	-527	-585
		100	-837	-997	-838	-710	-669
3	0.40	25	-589	-2 199	-3 683	-5 682	-5 177
		50	-3 690	-6 038	-9 230	-11 694	-16 734
		75	-7 874	-12 975	-22 579	-26 725	-32 181
		100	-14 731	-24 089	-36 633	-42 728	-61 757
	0.55	25	-368	-1 064	-1 743	-2 583	-2 068
		50	-1 944	-2 585	-3 369	-3 277	-3 870
		75	-3 349	-4 227	-6 082	-6 629	-6 122
		100	-5 566	-6 856	-8 997	$\leq -9 147$	$\leq -12 177$
	0.70	25	-230	-497	-746	-954	-830
		50	-1 139	-1 207	-1 092	-1 078	-1 089
		75	-1 635	-1 511	-1 915	-1 999	-1 672
		100	-2 231	$\leq -2 244$	$\leq -2 502$	$\leq -2 168$	$\leq -2 803$
	0.85	25	-155	-244	-376	-268	-259
		50	-709	-624	-461	-360	-341
		75	-878	-629	-808	-595	-499
		100	-848	-815	-802	-595	-717

Table A.7: Best known values of the *Large* family QUBO problems.(a) Problems with  $\bar{p} = 0.0$ .

Variables ( $n$ )	Instance ( $k$ )	Maximum Values ( $\tau(f)$ )			
		$d = 25\%$	$d = 50\%$	$d = 75\%$	$d = 100\%$
500	1	$\geq 1896$	1010	602	335
	2	$\geq 1975$	1026	657	322
	3	$\geq 1912$	1008	603	319
	4	$\geq 1977$	1041	631	309
	5	$\geq 1943$	988	636	331
	6	$\geq 1888$	998	648	359
	7	$\geq 1951$	1009	623	322
	8	$\geq 1899$	1056	601	321
	9	$\geq 1890$	1077	642	323
	10	$\geq 2005$	1022	641	324
1000	1	$\geq 2278$	$\geq 1211$	733	371
	2	$\geq 2320$	$\geq 1194$	703	369
	3	$\geq 2307$	$\geq 1238$	695	344
	4	$\geq 2321$	$\geq 1125$	702	349
	5	$\geq 2252$	$\geq 1149$	705	383
	6	$\geq 2301$	$\geq 1161$	686	341
	7	$\geq 2300$	$\geq 1113$	695	372
	8	$\geq 2359$	$\geq 1194$	714	341
	9	$\geq 2313$	$\geq 1201$	700	345
	10	$\geq 2341$	$\geq 1163$	666	342
2500	1	$\geq 2720$	$\geq 1337$	$\geq 786$	406
	2	$\geq 2626$	$\geq 1340$	$\geq 786$	395
	3	$\geq 2735$	$\geq 1353$	$\geq 828$	389
	4	$\geq 2757$	$\geq 1355$	$\geq 802$	394
	5	$\geq 2772$	$\geq 1284$	$\geq 802$	399
	6	$\geq 2701$	$\geq 1336$	$\geq 762$	391
	7	$\geq 2654$	$\geq 1288$	$\geq 788$	392
	8	$\geq 2696$	$\geq 1336$	$\geq 811$	392
	9	$\geq 2788$	$\geq 1322$	$\geq 802$	401
	10	$\geq 2722$	$\geq 1315$	$\geq 796$	396
5000	1	$\geq 3042$	$\geq 1428$	$\geq 830$	$\geq 412$
	2	$\geq 2953$	$\geq 1400$	$\geq 829$	$\geq 428$
	3	$\geq 3003$	$\geq 1421$	$\geq 826$	$\geq 409$
	4	$\geq 3018$	$\geq 1386$	$\geq 820$	$\geq 421$
	5	$\geq 2957$	$\geq 1419$	$\geq 867$	$\geq 414$
	6	$\geq 3003$	$\geq 1446$	$\geq 810$	$\geq 423$
	7	$\geq 2938$	$\geq 1448$	$\geq 847$	$\geq 411$
	8	$\geq 2956$	$\geq 1434$	$\geq 855$	$\geq 416$
	9	$\geq 2929$	$\geq 1425$	$\geq 832$	$\geq 415$
	10	$\geq 3122$	$\geq 1440$	$\geq 875$	$\geq 412$

(b) Problems with  $\bar{\rho} = 0.2$ .

Variables ( $n$ )	Instance ( $k$ )	Maximum Values ( $\tau(f)$ )			
		$d = 25\%$	$d = 50\%$	$d = 75\%$	$d = 100\%$
500	1	$\geq 10\,596$	$\geq 7\,807$	$\geq 6\,190$	$\geq 5\,633$
	2	$\geq 10\,541$	$\geq 8\,026$	$\geq 6\,224$	$\geq 5\,286$
	3	$\geq 10\,385$	$\geq 8\,376$	$\geq 6\,267$	$\geq 5\,154$
	4	$\geq 10\,483$	$\geq 7\,783$	$\geq 6\,403$	$\geq 5\,362$
	5	$\geq 11\,135$	$\geq 7\,887$	$\geq 6\,116$	$\geq 5\,208$
	6	$\geq 10\,336$	$\geq 8\,171$	$\geq 6\,302$	$\geq 5\,454$
	7	$\geq 11\,107$	$\geq 7\,768$	$\geq 6\,565$	$\geq 5\,363$
	8	$\geq 11\,250$	$\geq 7\,611$	$\geq 6\,497$	$\geq 4\,953$
	9	$\geq 10\,894$	$\geq 7\,740$	$\geq 6\,292$	$\geq 5\,611$
	10	$\geq 12\,090$	$\geq 7\,824$	$\geq 6\,360$	$\geq 5\,380$
1 000	1	$\geq 16\,831$	$\geq 11\,646$	$\geq 8\,881$	$\geq 7\,179$
	2	$\geq 15\,834$	$\geq 11\,119$	$\geq 8\,699$	$\geq 7\,681$
	3	$\geq 15\,864$	$\geq 11\,214$	$\geq 8\,616$	$\geq 7\,037$
	4	$\geq 16\,004$	$\geq 11\,160$	$\geq 8\,818$	$\geq 7\,514$
	5	$\geq 16\,248$	$\geq 10\,996$	$\geq 8\,605$	$\geq 7\,239$
	6	$\geq 15\,818$	$\geq 10\,958$	$\geq 8\,687$	$\geq 7\,228$
	7	$\geq 16\,233$	$\geq 11\,287$	$\geq 8\,451$	$\geq 7\,179$
	8	$\geq 16\,723$	$\geq 11\,133$	$\geq 8\,621$	$\geq 7\,393$
	9	$\geq 16\,021$	$\geq 11\,015$	$\geq 8\,592$	$\geq 7\,232$
	10	$\geq 16\,553$	$\geq 11\,097$	$\geq 8\,705$	$\geq 7\,161$
2 500	1	$\geq 25\,276$	$\geq 15\,689$	$\geq 12\,094$	$\geq 9\,784$
	2	$\geq 25\,085$	$\geq 16\,130$	$\geq 12\,144$	$\geq 9\,502$
	3	$\geq 25\,277$	$\geq 16\,173$	$\geq 12\,358$	$\geq 9\,858$
	4	$\geq 24\,934$	$\geq 16\,235$	$\geq 12\,072$	$\geq 10\,114$
	5	$\geq 24\,884$	$\geq 16\,247$	$\geq 12\,051$	$\geq 9\,784$
	6	$\geq 25\,206$	$\geq 16\,062$	$\geq 12\,223$	$\geq 9\,835$
	7	$\geq 25\,319$	$\geq 16\,316$	$\geq 12\,321$	$\geq 10\,315$
	8	$\geq 24\,953$	$\geq 15\,863$	$\geq 12\,218$	$\geq 9\,821$
	9	$\geq 24\,881$	$\geq 15\,747$	$\geq 12\,093$	$\geq 9\,654$
	10	$\geq 24\,799$	$\geq 16\,271$	$\geq 12\,109$	$\geq 10\,130$
5 000	1	$\geq 32\,102$	$\geq 19\,832$	$\geq 14\,422$	$\geq 11\,688$
	2	$\geq 31\,220$	$\geq 19\,790$	$\geq 14\,291$	$\geq 11\,843$
	3	$\geq 31\,645$	$\geq 19\,938$	$\geq 14\,402$	$\geq 11\,310$
	4	$\geq 31\,553$	$\geq 19\,644$	$\geq 14\,583$	$\geq 12\,079$
	5	$\geq 30\,862$	$\geq 19\,636$	$\geq 14\,205$	$\geq 11\,680$
	6	$\geq 31\,516$	$\geq 19\,440$	$\geq 14\,499$	$\geq 11\,330$
	7	$\geq 31\,721$	$\geq 19\,260$	$\geq 14\,512$	$\geq 11\,514$
	8	$\geq 32\,005$	$\geq 20\,171$	$\geq 14\,443$	$\geq 11\,771$
	9	$\geq 31\,542$	$\geq 19\,413$	$\geq 14\,336$	$\geq 11\,588$
	10	$\geq 30\,815$	$\geq 19\,632$	$\geq 15\,336$	$\geq 11\,656$

(c) Problems with  $\bar{\rho} = 0.5$ .

Variables ( $n$ )	Instance ( $k$ )	Maximum Values ( $\tau(f)$ )			
		$d = 25\%$	$d = 50\%$	$d = 75\%$	$d = 100\%$
500	1	$\geq 193\,720$	$\geq 279\,570$	$\geq 303\,504$	$\geq 388\,093$
	2	$\geq 188\,081$	$\geq 267\,860$	$\geq 335\,519$	$\geq 393\,598$
	3	$\geq 195\,303$	$\geq 249\,201$	$\geq 331\,281$	$\geq 370\,484$
	4	$\geq 190\,422$	$\geq 268\,906$	$\geq 347\,094$	$\geq 429\,240$
	5	$\geq 201\,675$	$\geq 282\,127$	$\geq 344\,228$	$\geq 420\,516$
	6	$\geq 196\,179$	$\geq 271\,222$	$\geq 313\,344$	$\geq 352\,323$
	7	$\geq 181\,627$	$\geq 310\,386$	$\geq 327\,042$	$\geq 372\,387$
	8	$\geq 187\,568$	$\geq 273\,350$	$\geq 348\,639$	$\geq 390\,844$
	9	$\geq 211\,365$	$\geq 295\,870$	$\geq 301\,635$	$\geq 388\,741$
	10	$\geq 194\,761$	$\geq 269\,929$	$\geq 355\,210$	$\geq 377\,461$
1 000	1	$\geq 552\,555$	$\geq 767\,581$	$\geq 903\,348$	$\geq 1\,083\,464$
	2	$\geq 565\,454$	$\geq 788\,852$	$\geq 968\,276$	$\geq 1\,026\,577$
	3	$\geq 528\,104$	$\geq 788\,089$	$\geq 960\,572$	$\geq 1\,073\,194$
	4	$\geq 542\,066$	$\geq 763\,249$	$\geq 938\,678$	$\geq 1\,085\,743$
	5	$\geq 571\,010$	$\geq 829\,640$	$\geq 982\,533$	$\geq 1\,089\,956$
	6	$\geq 563\,361$	$\geq 791\,024$	$\geq 915\,896$	$\geq 1\,085\,129$
	7	$\geq 543\,531$	$\geq 777\,923$	$\geq 955\,619$	$\geq 1\,054\,291$
	8	$\geq 552\,963$	$\geq 773\,215$	$\geq 930\,066$	$\geq 1\,051\,310$
	9	$\geq 558\,924$	$\geq 804\,173$	$\geq 966\,388$	$\geq 1\,132\,824$
	10	$\geq 579\,971$	$\geq 748\,120$	$\geq 933\,109$	$\geq 1\,107\,758$
2 500	1	$\geq 2\,117\,885$	$\geq 2\,921\,232$	$\geq 3\,748\,994$	$\geq 4\,244\,425$
	2	$\geq 2\,206\,568$	$\geq 3\,159\,973$	$\geq 3\,750\,561$	$\geq 4\,302\,240$
	3	$\geq 2\,179\,960$	$\geq 3\,085\,238$	$\geq 3\,742\,426$	$\geq 4\,556\,506$
	4	$\geq 2\,061\,958$	$\geq 3\,095\,310$	$\geq 3\,753\,850$	$\geq 4\,237\,821$
	5	$\geq 2\,063\,035$	$\geq 3\,094\,166$	$\geq 3\,776\,627$	$\geq 4\,326\,133$
	6	$\geq 2\,150\,995$	$\geq 3\,067\,753$	$\geq 3\,766\,020$	$\geq 4\,435\,310$
	7	$\geq 2\,152\,229$	$\geq 3\,023\,673$	$\geq 3\,826\,756$	$\geq 4\,303\,320$
	8	$\geq 2\,182\,669$	$\geq 3\,067\,367$	$\geq 3\,673\,748$	$\geq 4\,343\,147$
	9	$\geq 2\,161\,980$	$\geq 3\,073\,944$	$\geq 3\,627\,131$	$\geq 4\,331\,541$
	10	$\geq 2\,136\,842$	$\geq 2\,968\,921$	$\geq 3\,883\,188$	$\geq 4\,294\,991$
5 000	1	$\geq 5\,965\,812$	$\geq 8\,658\,526$	$\geq 10\,885\,506$	$\geq 12\,265\,662$
	2	$\geq 5\,990\,057$	$\geq 8\,771\,424$	$\geq 10\,947\,831$	$\geq 12\,131\,166$
	3	$\geq 6\,148\,221$	$\geq 8\,679\,817$	$\geq 10\,734\,928$	$\geq 12\,180\,956$
	4	$\geq 5\,963\,305$	$\geq 8\,633\,975$	$\geq 10\,485\,856$	$\geq 12\,073\,331$
	5	$\geq 6\,133\,071$	$\geq 8\,573\,093$	$\geq 10\,460\,969$	$\geq 12\,356\,295$
	6	$\geq 6\,079\,756$	$\geq 8\,533\,052$	$\geq 10\,637\,130$	$\geq 12\,206\,640$
	7	$\geq 6\,116\,353$	$\geq 8\,670\,991$	$\geq 10\,538\,428$	$\geq 12\,030\,678$
	8	$\geq 6\,117\,382$	$\geq 8\,772\,400$	$\geq 10\,797\,015$	$\geq 11\,815\,372$
	9	$\geq 6\,178\,704$	$\geq 8\,513\,475$	$\geq 10\,439\,669$	$\geq 12\,273\,264$
	10	$\geq 6\,176\,659$	$\geq 8\,518\,653$	$\geq 10\,238\,785$	$\geq 12\,249\,156$

Table A.8: Best known cuts of the *Hamilton* graphs.(a) Group number one ( $k = 1$ ).

Exterior Field ( $h$ )	Edge's Weights ( $[w^-, w^+]$ )	Vertices ( $ V $ )	Weighted MAX-CUT ( $\tau(f)$ )			
			2 cycles	4 cycles	6 cycles	8 cycles
0	1	250	$\geq 432$	$\geq 762$	$\geq 1072$	$\geq 1370$
		500	$\geq 862$	$\geq 1524$	$\geq 2156$	$\geq 2746$
		1000	$\geq 1730$	$\geq 3056$	$\geq 4298$	$\geq 5500$
		2000	$\geq 3458$	$\geq 6106$	$\geq 8606$	$\geq 11010$
0	[-50, 50]	250	5066	$\geq 7161$	$\geq 8832$	$\geq 11807$
		500	$\geq 10645$	$\geq 15556$	$\geq 19599$	$\geq 22802$
		1000	$\geq 19618$	$\geq 28792$	$\geq 38054$	$\geq 47412$
		2000	$\geq 40920$	$\geq 61500$	$\geq 74248$	$\geq 88216$
25	[-50, 50]	250	9214	$\geq 11286$	$\geq 12988$	$\geq 15035$
		500	$\geq 18170$	$\geq 21037$	$\geq 26618$	$\geq 29143$
		1000	$\geq 37080$	$\geq 45208$	$\geq 53728$	$\geq 58431$
		2000	$\geq 72726$	$\geq 89568$	$\geq 102105$	$\geq 113604$
-75	[50, 100]	250	$\geq 23999$	$\geq 48771$	$\geq 71787$	$\geq 93689$
		500	$\geq 47300$	$\geq 96926$	$\geq 143087$	$\geq 187871$
		1000	$\geq 94895$	$\geq 195912$	$\geq 288124$	$\geq 378970$
		2000	$\geq 190347$	$\geq 388399$	$\geq 575993$	$\geq 753810$
75	[50, 100]	250	$\geq 42866$	$\geq 67629$	$\geq 90684$	$\geq 112618$
		500	$\geq 85259$	$\geq 135183$	$\geq 181214$	$\geq 226677$
		1000	$\geq 170633$	$\geq 270547$	$\geq 362323$	$\geq 453210$
		2000	$\geq 339274$	$\geq 538610$	$\geq 726052$	$\geq 906275$

(b) Group number two ( $k = 2$ ).

Exterior Field ( $h$ )	Edge's Weights ( $[w^-, w^+]$ )	Vertices ( $ V $ )	Weighted MAX-CUT ( $\tau(f)$ )			
			2 cycles	4 cycles	6 cycles	8 cycles
0	1	250	$\geq 432$	$\geq 766$	$\geq 1078$	$\geq 1368$
		500	$\geq 866$	$\geq 1526$	$\geq 2154$	$\geq 2750$
		1000	$\geq 1728$	$\geq 3050$	$\geq 4306$	$\geq 5510$
		2000	$\geq 3458$	$\geq 6104$	$\geq 8600$	$\geq 11010$
0	[-50, 50]	250	5294	$\geq 7905$	$\geq 9616$	$\geq 11187$
		500	$\geq 10646$	$\geq 15080$	$\geq 19259$	$\geq 21889$
		1000	$\geq 20429$	$\geq 31360$	$\geq 36794$	$\geq 44499$
		2000	$\geq 41320$	$\geq 60689$	$\geq 72583$	$\geq 87528$
25	[-50, 50]	250	8889	$\geq 11289$	$\geq 12619$	$\geq 14590$
		500	17871	$\geq 22606$	$\geq 24946$	$\geq 28826$
		1000	$\geq 35371$	$\geq 46124$	$\geq 52950$	$\geq 57618$
		2000	$\geq 71112$	$\geq 89317$	$\geq 105612$	$\geq 114356$
-75	[50, 100]	250	$\geq 23609$	$\geq 48150$	$\geq 71550$	$\geq 95059$
		500	$\geq 47659$	$\geq 96798$	$\geq 143424$	$\geq 190579$
		1000	$\geq 95238$	$\geq 194868$	$\geq 289076$	$\geq 378579$
		2000	$\geq 189262$	$\geq 389692$	$\geq 576861$	$\geq 753951$
75	[50, 100]	250	$\geq 42786$	$\geq 67277$	$\geq 91074$	$\geq 112563$
		500	$\geq 85469$	$\geq 133994$	$\geq 181180$	$\geq 227791$
		1000	$\geq 170473$	$\geq 269941$	$\geq 362123$	$\geq 453730$
		2000	$\geq 341075$	$\geq 537083$	$\geq 724955$	$\geq 905677$

(c) Group number three ( $k = 3$ ).

<i>Exterior Field</i> ( $h$ )	<i>Edge's Weights</i> ( $[w^-, w^+]$ )	<i>Vertices</i> ( $ V $ )	<i>Weighted MAX-CUT</i> ( $\tau(f)$ )			
			2 cycles	4 cycles	6 cycles	8 cycles
0	1	250	$\geq 428$	$\geq 758$	$\geq 1,076$	$\geq 1,378$
		500	$\geq 864$	$\geq 1\,524$	$\geq 2\,152$	$\geq 2\,750$
		1 000	$\geq 1\,732$	$\geq 3\,056$	$\geq 4\,306$	$\geq 5\,498$
		2 000	$\geq 3\,464$	$\geq 6\,114$	$\geq 8\,596$	$\geq 11\,000$
0	[-50, 50]	250	4 880	$\geq 7\,437$	$\geq 9\,707$	$\geq 10\,984$
		500	$\geq 9\,936$	$\geq 15\,893$	$\geq 18\,685$	$\geq 21\,482$
		1 000	$\geq 20\,123$	$\geq 30\,601$	$\geq 38\,188$	$\geq 43\,714$
		2 000	$\geq 41\,427$	$\geq 61\,183$	$\geq 76\,649$	$\geq 85\,448$
25	[-50, 50]	250	9 401	$\geq 10\,722$	$\geq 12\,970$	$\geq 14\,915$
		500	17 066	$\geq 21\,669$	$\geq 25\,188$	$\geq 28\,524$
		1 000	$\geq 34\,798$	$\geq 44\,328$	$\geq 49\,428$	$\geq 56\,768$
		2 000	$\geq 71\,957$	$\geq 89\,353$	$\geq 103\,163$	$\geq 112\,632$
-75	[50, 100]	250	$\geq 24\,421$	$\geq 48\,712$	$\geq 71\,442$	$\geq 93\,877$
		500	$\geq 47\,398$	$\geq 97\,304$	$\geq 143\,755$	$\geq 189\,146$
		1 000	$\geq 95\,736$	$\geq 192\,805$	$\geq 289\,256$	$\geq 378\,515$
		2 000	$\geq 190\,735$	$\geq 387\,780$	$\geq 575\,452$	$\geq 755\,498$
75	[50, 100]	250	$\geq 42\,674$	$\geq 67\,436$	$\geq 90\,052$	$\geq 112\,631$
		500	$\geq 84\,391$	$\geq 134\,772$	$\geq 181\,278$	$\geq 227\,221$
		1 000	$\geq 170\,709$	$\geq 270\,208$	$\geq 362\,367$	$\geq 453\,514$
		2 000	$\geq 337\,682$	$\geq 537\,761$	$\geq 723\,906$	$\geq 908\,119$

Table A.9: Best known values of the randomly generated MAX-2-SAT problems.

(a) SAT subfamily.

Profile ID	Instance ( $k$ )	Minimum False Clauses ( $\nu(f)$ )			
		$n = 50$	$n = 100$	$n = 200$	$n = 400$
1	1	169	720	2 905	12 085
	2	182	793	3 023	12 164
	3	188	748	2 935	12 138
	4	177	723	3 036	11 993
	5	184	772	2 973	11 952
2	1	86	360	1 397	5 605
	2	83	324	1 451	5 655
	3	86	346	1 396	5 743
	4	86	351	1 375	5 692
	5	85	338	1 377	5 705
3	1	233	$\leq 982$	$\leq 4 235$	$\leq 17 907$
	2	218	$\leq 975$	$\leq 4 258$	$\leq 17 905$
	3	226	$\leq 1 016$	$\leq 4 258$	$\leq 17 873$
	4	217	$\leq 1 000$	$\leq 4 259$	$\leq 17 889$
	5	227	$\leq 1 006$	$\leq 4 282$	$\leq 17 898$
4	1	121	565	$\leq 2 523$	$\leq 10 472$
	2	125	568	$\leq 2 393$	$\leq 10 342$
	3	122	566	$\leq 2 476$	$\leq 10 408$
	4	116	$\leq 557$	$\leq 2 405$	$\leq 10 345$
	5	133	535	$\leq 2 441$	$\leq 10 446$
5	1	36	175	$\leq 834$	$\leq 3 804$
	2	43	180	$\leq 855$	$\leq 3 792$
	3	40	192	$\leq 864$	$\leq 3 801$
	4	39	195	$\leq 864$	$\leq 3 812$
	5	37	185	$\leq 858$	$\leq 3 790$
6	1	53	260	$\leq 1 211$	$\leq 5 280$
	2	50	284	$\leq 1 209$	$\leq 5 273$
	3	54	252	$\leq 1 171$	$\leq 5 160$
	4	55	266	$\leq 1 243$	$\leq 5 264$
	5	55	280	$\leq 1 214$	$\leq 5 267$
7	1	200	$\leq 871$	$\leq 3 765$	$\leq 15 787$
	2	191	$\leq 912$	$\leq 3 819$	$\leq 15 698$
	3	196	$\leq 899$	$\leq 3 794$	$\leq 15 765$
	4	208	$\leq 887$	$\leq 3 815$	$\leq 15 833$
	5	202	$\leq 866$	$\leq 3 717$	$\leq 15 873$
8	1	105	470	$\leq 1 981$	$\leq 8 555$
	2	102	$\leq 464$	$\leq 2 044$	$\leq 8 587$
	3	104	$\leq 486$	$\leq 1 988$	$\leq 8 613$
	4	96	$\leq 453$	$\leq 2 008$	$\leq 8 599$
	5	112	$\leq 458$	$\leq 1 990$	$\leq 8 537$

(b) WSAT-[1,10] subfamily.

Profile ID	Instance (k)	Minimum False Clauses ( $\nu(f)$ )			
		n = 50	n = 100	n = 200	n = 400
1	1	1 078	4 130	16 478	66 431
	2	1 018	4 125	16 008	66 401
	3	929	4 224	16 578	66 095
	4	978	4 253	16 263	65 780
	5	1 000	4 105	16 047	65 918
2	1	421	2 005	7 966	31 313
	2	463	1 812	8 074	31 071
	3	448	2 027	7 842	31 403
	4	449	1 882	7 832	31 009
	5	522	2 018	7 936	31 247
3	1	1 176	$\leq 5\,305$	$\leq 22\,797$	$\leq 96\,101$
	2	1 206	$\leq 5\,076$	$\leq 22\,786$	$\leq 97\,147$
	3	1 214	$\leq 5\,227$	$\leq 22\,711$	$\leq 96\,421$
	4	1 214	$\leq 5\,199$	$\leq 22\,836$	$\leq 96\,970$
	5	1 165	$\leq 5\,140$	$\leq 23\,299$	$\leq 95\,920$
4	1	729	2 883	$\leq 13\,050$	$\leq 56\,515$
	2	632	$\leq 2\,851$	$\leq 12\,710$	$\leq 56\,177$
	3	597	$\leq 3\,052$	$\leq 13\,273$	$\leq 55\,938$
	4	554	$\leq 2\,974$	$\leq 13\,251$	$\leq 56\,664$
	5	641	$\leq 2\,961$	$\leq 12\,977$	$\leq 55\,737$
5	1	178	919	$\leq 4\,417$	$\leq 20\,677$
	2	209	1 008	$\leq 4\,530$	$\leq 19\,893$
	3	225	1 017	$\leq 4\,217$	$\leq 20\,200$
	4	247	995	$\leq 4\,514$	$\leq 20\,570$
	5	222	968	$\leq 4\,327$	$\leq 19\,844$
6	1	274	1 461	$\leq 6\,401$	$\leq 27\,981$
	2	233	1 315	$\leq 6\,318$	$\leq 28\,200$
	3	261	1 451	$\leq 6\,605$	$\leq 28\,003$
	4	250	1 346	$\leq 6\,147$	$\leq 27\,721$
	5	264	1 340	$\leq 6\,432$	$\leq 27\,813$
7	1	1 045	$\leq 4\,763$	$\leq 20\,320$	$\leq 85\,630$
	2	1 087	$\leq 4\,694$	$\leq 20\,521$	$\leq 84\,484$
	3	1 056	$\leq 4\,693$	$\leq 20\,458$	$\leq 85\,968$
	4	942	$\leq 4\,650$	$\leq 20\,373$	$\leq 86\,340$
	5	1 045	$\leq 4\,802$	$\leq 20\,435$	$\leq 86\,317$
8	1	447	$\leq 2\,528$	$\leq 10\,552$	$\leq 45\,986$
	2	505	$\leq 2\,595$	$\leq 10\,636$	$\leq 45\,532$
	3	561	2 126	$\leq 10\,558$	$\leq 45\,956$
	4	450	$\leq 2\,329$	$\leq 10\,542$	$\leq 45\,909$
	5	441	2 269	$\leq 10\,372$	$\leq 46\,578$

(c) WSAT-[1,100] subfamily.

Profile ID	Instance (k)	Minimum False Clauses ( $\nu(f)$ )			
		n = 50	n = 100	n = 200	n = 400
1	1	9 419	37 500	151 086	607 061
	2	8 917	36 193	146 401	614 767
	3	8 765	35 686	148 810	612 242
	4	8 422	37 616	151 897	601 632
	5	8 932	38 640	150 880	608 370
2	1	4 283	18 783	69 314	282 480
	2	4 395	17 045	73 054	285 126
	3	4 513	18 659	70 779	286 189
	4	3 671	20 014	71 461	285 728
	5	4 442	17 372	68 558	285 748
3	1	9 910	$\leq 48\,151$	$\leq 209\,851$	$\leq 879\,814$
	2	10 945	$\leq 47\,719$	$\leq 211\,381$	$\leq 886\,390$
	3	10 675	$\leq 46\,486$	$\leq 208\,549$	$\leq 883\,307$
	4	9 553	$\leq 47\,905$	$\leq 209\,637$	$\leq 888\,593$
	5	9 918	$\leq 47\,149$	$\leq 210\,136$	$\leq 887\,649$
4	1	5 737	26 982	$\leq 118\,651$	$\leq 508\,048$
	2	5 396	26 791	$\leq 117\,795$	$\leq 507\,736$
	3	5 469	26 310	$\leq 115\,447$	$\leq 512\,393$
	4	5 614	25 694	$\leq 118\,758$	$\leq 512\,348$
	5	5 438	$\leq 27\,848$	$\leq 118\,207$	$\leq 512\,381$
5	1	1 674	8 094	$\leq 40\,145$	$\leq 183\,267$
	2	1 646	9 147	$\leq 41\,567$	$\leq 183\,940$
	3	1 628	8 998	$\leq 41\,648$	$\leq 182\,193$
	4	1 414	7 850	$\leq 40\,131$	$\leq 187\,085$
	5	1 808	8 776	$\leq 40\,195$	$\leq 183\,009$
6	1	2 727	12 609	$\leq 58\,569$	$\leq 256\,252$
	2	2 324	12 731	$\leq 57\,005$	$\leq 256\,466$
	3	2 630	13 490	$\leq 56\,487$	$\leq 260\,561$
	4	2 963	13 444	$\leq 59\,273$	$\leq 254\,874$
	5	2 653	11 809	$\leq 60\,888$	$\leq 262\,830$
7	1	9 238	$\leq 43\,985$	$\leq 188\,313$	$\leq 784\,089$
	2	9 333	$\leq 43\,023$	$\leq 189\,879$	$\leq 788\,164$
	3	8 255	$\leq 41\,023$	$\leq 185\,516$	$\leq 778\,369$
	4	9 782	$\leq 42\,204$	$\leq 188\,299$	$\leq 785\,689$
	5	8 941	$\leq 42\,636$	$\leq 185\,158$	$\leq 785\,042$
8	1	4 594	$\leq 21\,495$	$\leq 96\,609$	$\leq 420\,688$
	2	4 775	21 326	$\leq 98\,571$	$\leq 418\,497$
	3	4 436	21 163	$\leq 97\,878$	$\leq 414\,141$
	4	4 956	20 788	$\leq 97\,544$	$\leq 422\,297$
	5	4 345	$\leq 21\,308$	$\leq 97\,808$	$\leq 420\,553$

(d) WSAT-[90,100] subfamily.

Profile ID	Instance (k)	Minimum False Clauses ( $\nu(f)$ )			
		n = 50	n = 100	n = 200	n = 400
1	1	18 101	76 763	287 294	1 142 997
	2	17 048	71 122	289 869	1 169 220
	3	17 714	71 183	288 847	1 146 747
	4	17 764	72 169	288 639	1 149 711
	5	17 622	72 557	279 593	1 149 335
2	1	7 103	34 777	135 249	534 493
	2	9 482	35 700	131 482	539 314
	3	7 862	32 484	137 411	541 226
	4	8 248	35 493	133 072	537 420
	5	7 209	34 973	137 376	531 988
3	1	21 784	$\leq 94\,016$	$\leq 404\,845$	$\leq 1\,692\,809$
	2	20 610	$\leq 92\,533$	$\leq 406\,774$	$\leq 1\,690\,477$
	3	22 277	$\leq 94\,139$	$\leq 406\,902$	$\leq 1\,702\,333$
	4	22 279	$\leq 95\,507$	$\leq 408\,499$	$\leq 1\,697\,074$
	5	20 710	$\leq 94\,944$	$\leq 406\,305$	$\leq 1\,695\,128$
4	1	12 033	49 973	$\leq 229\,164$	$\leq 980\,318$
	2	12 711	55 079	$\leq 231\,308$	$\leq 981\,967$
	3	11 760	54 834	$\leq 230\,894$	$\leq 982\,992$
	4	11 260	$\leq 53\,853$	$\leq 230\,990$	$\leq 987\,019$
	5	11 919	$\leq 52\,703$	$\leq 230\,842$	$\leq 992\,394$
5	1	3 907	17 280	$\leq 84\,354$	$\leq 362\,540$
	2	3 689	18 356	$\leq 82\,811$	$\leq 360\,733$
	3	3 983	17 391	$\leq 81\,687$	$\leq 362\,398$
	4	4 353	18 659	$\leq 82\,243$	$\leq 367\,793$
	5	3 433	18 668	$\leq 79\,091$	$\leq 367\,033$
6	1	5 145	24 351	$\leq 115\,599$	$\leq 506\,993$
	2	4 844	23 675	$\leq 114\,335$	$\leq 498\,561$
	3	4 616	24 749	$\leq 114\,243$	$\leq 497\,285$
	4	5 108	24 612	$\leq 115\,770$	$\leq 496\,892$
	5	5 022	26 204	$\leq 114\,638$	$\leq 496\,572$
7	1	18 433	$\leq 84\,856$	$\leq 361\,069$	$\leq 1\,505\,424$
	2	17 928	$\leq 84\,670$	$\leq 355\,410$	$\leq 1\,496\,948$
	3	18 896	$\leq 86\,320$	$\leq 360\,624$	$\leq 1\,502\,605$
	4	19 582	$\leq 79\,947$	$\leq 361\,107$	$\leq 1\,488\,725$
	5	18 498	$\leq 84\,859$	$\leq 362\,244$	$\leq 1\,500\,950$
8	1	10 229	$\leq 43\,426$	$\leq 188\,321$	$\leq 804\,306$
	2	8 444	$\leq 42\,100$	$\leq 188\,230$	$\leq 814\,102$
	3	9 568	$\leq 42\,491$	$\leq 194\,599$	$\leq 806\,611$
	4	10 246	$\leq 43\,267$	$\leq 189\,896$	$\leq 814\,702$
	5	9 447	39 857	$\leq 194\,276$	$\leq 820\,735$

## Appendix B

### Heuristics Statistics

Table B.1: Least squares fitting of performance ratios ( $r_{ACSIOM}$ ) in the *medium* test problems with  $\rho \leq 0.525$ .

(a) Model ANOVA.

	<i>Sum of Squares</i>	<i>df</i>	<i>Mean Square</i>	<i>F</i>	<i>sign.-F</i>
<i>Regression</i>	35.3429	3	11.7810	3217.4483	0.0000
<i>Residual</i>	9.5055	2596	0.0037		
<i>Total</i>	44.8484	2599			

(b) Model summary.

<i>Multiple R</i>	$R^2$	<i>Adjusted R<sup>2</sup></i>	<i>Standard Error</i>	<i>Number Observations</i>
0.8877	0.7881	0.7878	0.0605	2600

(c) Model coefficients.

<i>Regressor</i>	<i>Unstandardized Coefficients</i>		<i>t-stat.</i>	<i>sign.-t</i>	<i>95% Confidence Interval for Coefficients</i>	
	<i>Value</i>	<i>Std. Err.</i>			<i>Low. Bound</i>	<i>Up. Bound</i>
intercept	$a_0 = 6.8725E - 01$	0.0044	155.7904	0.0000	$6.7860E - 01$	$6.9590E - 01$
$n$	$a_n = -2.9132E - 05$	0.0000	-13.7230	0.0000	$-3.3295E - 05$	$-2.4970E - 05$
$d$	$a_d = -7.9109E - 02$	0.0042	-18.8547	0.0000	$-8.7336E - 02$	$-7.0881E - 02$
$\rho$	$a_\rho = 7.5261E - 01$	0.0079	95.4015	0.0000	$7.3714E - 01$	$7.6808E - 01$

Table B.2: Number of variables ( $n$ ) versus density ( $d$ ) analysis on the performance ratios values ( $r_{ACSIOM}$ ) for the *medium* problems.

(a)  $\mathcal{F} = \text{Medium}_{\rho \leq 0.15}$ .

$\rho \leq 0.15$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	61.4%	56.1%	54.5%	48.1%	38.3%
	75.8% $\pm$ 6.9%	71.5% $\pm$ 9.0%	71.1% $\pm$ 8.5%	67.5% $\pm$ 10.3%	64.0% $\pm$ 13.7%
$n = 1000$	59.1%	53.5%	47.1%	42.0%	44.8%
	71.2% $\pm$ 5.2%	68.8% $\pm$ 8.5%	66.0% $\pm$ 9.0%	60.9% $\pm$ 10.3%	66.0% $\pm$ 11.3%
$n = 1500$	54.3%	47.6%	46.6%	45.2%	43.2%
	67.8% $\pm$ 6.5%	66.7% $\pm$ 7.4%	64.7% $\pm$ 7.9%	67.6% $\pm$ 9.6%	64.4% $\pm$ 9.8%
$n = 2000$	53.1%	53.1%	45.8%	53.9%	45.3%
	68.7% $\pm$ 6.5%	68.0% $\pm$ 7.1%	66.0% $\pm$ 6.5%	68.2% $\pm$ 9.1%	63.1% $\pm$ 9.9%
	86.7%	82.5%	75.5%	90.5%	85.8%

(b)  $\mathcal{F} = \text{Medium}_{0.15 < \rho \leq 0.3}$ .

$0.15 < \rho \leq 0.3$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	68.5%	67.9%	61.5%	56.5%	56.7%
	84.0% $\pm$ 5.9%	81.9% $\pm$ 5.6%	78.9% $\pm$ 6.9%	76.6% $\pm$ 9.0%	76.3% $\pm$ 9.8%
$n = 1000$	66.9%	68.6%	60.8%	52.3%	51.8%
	81.2% $\pm$ 5.7%	78.3% $\pm$ 6.5%	76.4% $\pm$ 7.4%	74.7% $\pm$ 7.5%	71.8% $\pm$ 8.3%
$n = 1500$	64.4%	62.0%	56.4%	58.7%	49.2%
	79.5% $\pm$ 4.7%	76.4% $\pm$ 6.2%	74.0% $\pm$ 6.5%	74.3% $\pm$ 7.2%	69.5% $\pm$ 8.4%
$n = 2000$	65.7%	63.9%	64.1%	57.8%	52.5%
	78.5% $\pm$ 4.9%	75.7% $\pm$ 5.3%	72.5% $\pm$ 5.6%	71.5% $\pm$ 5.9%	68.8% $\pm$ 6.6%
	87.1%	85.1%	81.9%	85.8%	80.9%

(c)  $\mathcal{F} = \text{Medium}_{0.3 < \rho \leq 0.45}$ .

$0.3 < \rho \leq 0.45$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	85.5%	78.7%	80.1%	73.7%	73.1%
	95.0% $\pm$ 3.4%	92.5% $\pm$ 5.2%	91.6% $\pm$ 5.0%	89.8% $\pm$ 5.9%	87.4% $\pm$ 7.3%
$n = 1000$	87.9%	78.9%	77.3%	72.4%	72.3%
	92.9% $\pm$ 3.1%	89.8% $\pm$ 5.0%	88.9% $\pm$ 5.4%	87.5% $\pm$ 6.0%	86.3% $\pm$ 5.8%
$n = 1500$	78.9%	79.1%	68.9%	72.4%	70.4%
	91.4% $\pm$ 4.3%	88.4% $\pm$ 4.8%	86.0% $\pm$ 6.4%	84.1% $\pm$ 6.1%	84.0% $\pm$ 6.1%
$n = 2000$	82.0%	75.3%	69.4%	72.3%	68.4%
	89.7% $\pm$ 4.1%	86.7% $\pm$ 5.2%	84.6% $\pm$ 6.7%	83.2% $\pm$ 6.2%	82.7% $\pm$ 6.5%
	96.3%	95.6%	93.9%	93.2%	94.6%

(d)  $\mathcal{F} = \text{Medium}_{0.45 < \rho \leq 0.6}$ .

$0.45 < \rho \leq 0.6$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	97.9%	96.7%	96.1%	95.5%	96.0%
	99.6% $\pm$ 0.6%	99.4% $\pm$ 0.8%	99.5% $\pm$ 0.9%	99.4% $\pm$ 1.1%	99.3% $\pm$ 1.1%
$n = 1000$	97.8%	95.8%	95.7%	95.0%	94.5%
	99.5% $\pm$ 0.6%	99.2% $\pm$ 1.2%	99.2% $\pm$ 1.2%	99.1% $\pm$ 1.4%	99.2% $\pm$ 1.5%
$n = 1500$	97.0%	95.7%	94.7%	92.0%	93.2%
	99.4% $\pm$ 0.8%	99.3% $\pm$ 1.1%	98.9% $\pm$ 1.7%	98.8% $\pm$ 2.0%	98.9% $\pm$ 1.9%
$n = 2000$	96.6%	94.6%	93.4%	93.0%	92.7%
	99.3% $\pm$ 1.0%	99.2% $\pm$ 1.3%	99.0% $\pm$ 1.8%	98.7% $\pm$ 2.2%	98.6% $\pm$ 2.2%
	100.0%	100.0%	100.0%	100.0%	100.0%

Table B.3: Number of variables ( $n$ ) versus density ( $d$ ) analysis of number of roundings per variable necessary by  $A_{ACSIOM}$  in the problems of the *medium* family.

(a)  $\mathcal{F} = \text{Medium}_{\rho \leq 0.15}$ .

$\rho \leq 0.15$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	1.00600	1.00400	1.00400	1.00200	1.00200
	1.01977 $\pm$ 0.00703	1.01331 $\pm$ 0.00649	1.01217 $\pm$ 0.00459	1.00971 $\pm$ 0.00527	1.00714 $\pm$ 0.00344
$n = 1000$	1.00500	1.00300	1.00200	1.00200	1.00200
	1.01414 $\pm$ 0.00508	1.01003 $\pm$ 0.00383	1.00769 $\pm$ 0.00289	1.00589 $\pm$ 0.00275	1.00571 $\pm$ 0.00333
$n = 1500$	1.00600	1.00400	1.00333	1.00267	1.00133
	1.01095 $\pm$ 0.00302	1.00741 $\pm$ 0.00268	1.00623 $\pm$ 0.00184	1.00486 $\pm$ 0.00147	1.00360 $\pm$ 0.00159
$n = 2000$	1.00600	1.00350	1.00200	1.00200	1.00150
	1.01030 $\pm$ 0.00249	1.00644 $\pm$ 0.00176	1.00467 $\pm$ 0.00176	1.00404 $\pm$ 0.00166	1.00326 $\pm$ 0.00149

(b)  $\mathcal{F} = \text{Medium}_{0.15 < \rho \leq 0.3}$ .

$0.15 < \rho \leq 0.3$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	1.01000	1.00800	1.00800	1.00400	1.00400
	1.02543 $\pm$ 0.01074	1.02069 $\pm$ 0.00803	1.01789 $\pm$ 0.00755	1.01497 $\pm$ 0.00626	1.01429 $\pm$ 0.00629
$n = 1000$	1.01100	1.00700	1.00500	1.00400	1.00200
	1.01974 $\pm$ 0.00637	1.01706 $\pm$ 0.00712	1.01394 $\pm$ 0.00654	1.01274 $\pm$ 0.00515	1.01097 $\pm$ 0.00485
$n = 1500$	1.00600	1.00667	1.00467	1.00467	1.00200
	1.01838 $\pm$ 0.00568	1.01571 $\pm$ 0.00576	1.01091 $\pm$ 0.00425	1.01038 $\pm$ 0.00408	1.00709 $\pm$ 0.00306
$n = 2000$	1.00750	1.00550	1.00450	1.00400	1.00250
	1.01967 $\pm$ 0.00611	1.01233 $\pm$ 0.00448	1.01019 $\pm$ 0.00387	1.00813 $\pm$ 0.00217	1.00696 $\pm$ 0.00314

(c)  $\mathcal{F} = \text{Medium}_{0.3 < \rho \leq 0.45}$ .

$0.3 < \rho \leq 0.45$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	1.00600	1.00800	1.00600	1.00800	1.00800
	1.03190 $\pm$ 0.01235	1.02875 $\pm$ 0.01122	1.02790 $\pm$ 0.01279	1.03085 $\pm$ 0.01594	1.03035 $\pm$ 0.01661
$n = 1000$	1.01600	1.01000	1.00900	1.01100	1.00600
	1.03280 $\pm$ 0.01111	1.03090 $\pm$ 0.01249	1.02913 $\pm$ 0.01208	1.02905 $\pm$ 0.01127	1.02693 $\pm$ 0.01252
$n = 1500$	1.01133	1.01400	1.00933	1.00933	1.00867
	1.03167 $\pm$ 0.01115	1.03278 $\pm$ 0.01247	1.02755 $\pm$ 0.01199	1.02415 $\pm$ 0.01049	1.02170 $\pm$ 0.00964
$n = 2000$	1.01600	1.01250	1.00950	1.00850	1.00800
	1.03231 $\pm$ 0.00884	1.02859 $\pm$ 0.01001	1.02711 $\pm$ 0.01309	1.02230 $\pm$ 0.01096	1.02316 $\pm$ 0.01272

(d)  $\mathcal{F} = \text{Medium}_{0.45 < \rho \leq 0.6}$ .

$0.45 < \rho \leq 0.6$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 100\%$
$n = 500$	1.00200	1.00200	1.00200	1.00200	1.00200
	1.01389 $\pm$ 0.01148	1.01257 $\pm$ 0.01104	1.01286 $\pm$ 0.01136	1.01046 $\pm$ 0.01048	1.01566 $\pm$ 0.01783
$n = 1000$	1.00100	1.00100	1.00100	1.00100	1.00100
	1.01291 $\pm$ 0.01338	1.01231 $\pm$ 0.01390	1.01169 $\pm$ 0.01193	1.01386 $\pm$ 0.01690	1.01386 $\pm$ 0.01709
$n = 1500$	1.00067	1.00067	1.00067	1.00067	1.00067
	1.01364 $\pm$ 0.01298	1.01307 $\pm$ 0.01590	1.01331 $\pm$ 0.01562	1.01411 $\pm$ 0.01726	1.01320 $\pm$ 0.01639
$n = 2000$	1.00050	1.00050	1.00050	1.00050	1.00050
	1.01344 $\pm$ 0.01452	1.01336 $\pm$ 0.01529	1.01353 $\pm$ 0.01583	1.01647 $\pm$ 0.01984	1.01303 $\pm$ 0.01574

## Appendix C

### Preprocessing Statistics

Several statistical counters were included in the algorithm PREPRO. Let us remark that all the computing times presented in this Appendix include the time needed for this statistical gathering. Next, we describe the statistical counters that we have placed in PREPRO:

- $t_n$  – Time in seconds used by NETWORK;
- $n_s$  – Number of strong persistencies found by NETWORK;
- $n_w$  – Number of weak persistencies found by NETWORK;
- $t_p$  – Time used by PROBING in seconds;
- $n_b$  – Number of persistencies found by bounding in PROBING;
- $n_f$  – Number of linear persistencies found by quadratic consensus in PROBING;
- $n_e$  – Number of equality relations found by quadratic consensus in PROBING;
- $t_c$  – Time used by COORDINATION in seconds;
- $n_c$  – Number of equality relations found in COORDINATION;

We have also obtained several statistics immediately after the execution of PREPRO.

Namely:

- $t$  – Total computing time of PREPRO in seconds;
- $q$  – Number of quadratic persistencies of the resulting posiform;

- $g$  – Relative gap  $\frac{|U-L|}{|U|}$  between the upper bound  $U$  and the lower bound  $L$  returned by PREPRO;
- $c$  – Number of strong components of the resulting posiform;
- $s$  – Number of variables of the largest strong component of the resulting posiform;
- $r$  – Relative percentage of the number of variables fixed by PREPRO.

Table C.1: PREPRO statistical report on the QUBO problems of Glover, Kochenberger and Alidaee [108].

Problem		Preprocessing Tools Statistics									After Preprocessing					
		Roof-Duality			Probing				Coordination		Total Time (t)	Quad. Rel. (q)	Relat. Gap (g)	Comp. Left		Variab. Reduc. (r)
		Time (t <sub>n</sub> )	Strong (n <sub>s</sub> )	Weak (n <sub>w</sub> )	Time (t <sub>p</sub> )	Bound. (n <sub>b</sub> )	Fix. (n <sub>f</sub> )	Eq. (n <sub>e</sub> )	Time (t <sub>c</sub> )	Eq. (n <sub>c</sub> )				Num. (c)	Larg. (s)	
Family	Numb.															
A	1	0.0	48	2	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	2	0.0	60	0	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	3	0.0	17	0	0.0	3	31	19	0.1	0	0.1	0	0.0%	0	0	100.0%
	4	0.0	78	0	0.0	0	0	0	0.0	2	0.0	0	0.0%	0	0	100.0%
	5	0.0	5	1	0.0	0	0	0	0.0	0	0.0	4	8.7%	1	44	12.0%
	6	0.0	0	0	0.0	0	0	0	0.0	0	0.0	2	15.9%	1	30	0.0%
	7	0.0	0	0	0.0	0	0	0	0.0	0	0.0	0	16.4%	1	30	0.0%
	8	0.0	100	0	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
B	1	0.0	8	0	0.0	11	0	0	0.0	1	0.0	0	0.0%	0	0	100.0%
	2	0.0	9	0	0.0	18	0	1	0.0	2	0.1	0	0.0%	0	0	100.0%
	3	0.0	6	1	0.0	28	0	3	0.1	2	0.1	0	0.0%	0	0	100.0%
	4	0.0	1	1	0.2	13	1	0	0.2	0	0.4	466	77.9%	1	34	32.0%
	5	0.0	1	0	0.2	39	0	0	0.4	0	0.6	141	66.3%	1	20	66.7%
	6	0.1	6	0	0.3	62	0	1	0.8	1	1.2	0	0.0%	0	0	100.0%
	7	0.1	4	0	0.7	67	5	4	1.5	0	2.4	0	0.0%	0	0	100.0%
	8	0.1	0	1	1.8	31	0	0	2.2	0	4.2	1325	90.0%	1	58	35.6%
	9	0.1	0	1	2.8	20	0	0	2.8	0	5.8	2585	92.0%	1	79	21.0%
	10	0.3	0	3	6.1	30	0	0	7.5	0	14.0	3495	93.1%	1	92	26.4%
C	1	0.0	0	0	0.1	0	0	0	0.0	0	0.1	0	36.0%	1	40	0.0%
	2	0.0	0	0	0.1	0	0	0	0.0	0	0.1	0	36.5%	1	50	0.0%
	3	0.0	0	0	0.1	0	0	0	0.0	0	0.1	0	27.7%	1	60	0.0%
	4	0.0	0	0	0.1	0	0	0	0.0	0	0.1	2	24.1%	1	70	0.0%
	5	0.0	6	0	0.4	2	0	1	0.0	0	0.4	5	14.7%	1	71	11.3%
	6	0.0	87	0	0.0	3	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	7	0.0	100	0	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
D	1	0.0	8	0	0.6	6	0	2	0.1	0	0.8	19	6.6%	1	84	16.0%
	2	0.0	0	0	0.3	0	0	0	0.0	0	0.3	14	45.7%	1	100	0.0%
	3	0.0	0	0	0.4	0	0	0	0.0	0	0.4	0	47.5%	1	100	0.0%
	4	0.0	0	0	0.7	0	0	0	0.0	0	0.7	0	56.1%	1	100	0.0%
	5	0.0	0	0	0.9	0	0	0	0.0	0	0.9	0	62.1%	1	100	0.0%
	6	0.0	0	0	1.2	0	0	0	0.0	0	1.3	0	61.3%	1	100	0.0%
	7	0.0	0	0	1.5	0	0	0	0.0	0	1.5	0	66.8%	1	100	0.0%
	8	0.0	0	0	1.8	0	0	0	0.0	0	1.8	0	67.0%	1	100	0.0%
	9	0.0	0	0	2.0	0	0	0	0.0	0	2.1	0	71.2%	1	100	0.0%
	10	0.0	0	0	2.4	0	0	0	0.0	0	2.4	0	69.4%	1	100	0.0%
E	1	0.0	0	0	1.5	0	0	0	0.0	0	1.6	32	29.6%	1	200	0.0%
	2	0.0	0	0	3.3	0	0	0	0.0	0	3.4	0	53.8%	1	200	0.0%
	3	0.0	0	0	5.1	0	0	0	0.0	0	5.2	0	65.2%	1	200	0.0%
	4	0.1	0	0	7.0	0	0	0	0.1	0	7.1	0	64.8%	1	200	0.0%
	5	0.1	0	0	8.9	0	0	0	0.1	0	9.0	0	72.6%	1	200	0.0%
F <sub>1</sub>	a	0.1	0	0	28.2	0	0	0	0.3	0	28.6	0	61.4%	1	500	0.0%
	b	0.3	0	0	66.3	0	0	0	0.4	0	67.0	0	75.0%	1	500	0.0%
	c	0.9	0	0	132.8	0	0	0	0.5	0	134.1	0	82.7%	1	500	0.0%
	d	1.6	0	0	211.1	0	0	0	0.5	0	213.4	0	85.6%	1	500	0.0%
	e	2.6	0	0	304.1	0	0	0	0.7	0	307.5	0	88.1%	1	500	0.0%

Table C.2: PREPRO statistical report on the QUBO problems of Beasley [37].

Problem		Preprocessing Tools Statistics									After Preprocessing					
		Roof-Duality			Probing				Coordination		Total Time (t)	Quad. Rel. (q)	Relat. Gap (g)	Comp. Left		Variab. Reduc. (r)
		Time (t <sub>n</sub> )	Strong (n <sub>s</sub> )	Weak (n <sub>w</sub> )	Time (t <sub>p</sub> )	Bound. (n <sub>b</sub> )	Fix. (n <sub>f</sub> )	Eq. (n <sub>e</sub> )	Time (t <sub>c</sub> )	Eq. (n <sub>c</sub> )				Num. (c)	Larg. (s)	
Family	Numb.															
ORL 50	1	0.0	46	4	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	2	0.0	45	5	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	3	0.0	50	0	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	4	0.0	48	2	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	5	0.0	49	1	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	6	0.0	49	0	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	7	0.0	49	1	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	8	0.0	50	0	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	9	0.0	48	0	0.0	2	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	10	0.0	45	3	0.0	2	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
ORL 100	1	0.0	4	0	0.3	0	0	4	0.1	0	0.4	45	19.3%	1	92	8.0%
	2	0.0	0	1	0.4	2	1	2	0.0	0	0.5	37	6.4%	1	94	6.0%
	3	0.0	85	0	0.3	8	6	1	0.1	0	0.4	0	0.0%	0	0	100.0%
	4	0.0	1	0	0.2	0	0	0	0.0	0	0.2	43	11.3%	1	99	1.0%
	5	0.0	3	2	0.4	2	1	0	0.0	0	0.4	39	11.1%	1	92	8.0%
	6	0.0	5	0	0.3	0	0	1	0.0	0	0.3	24	19.6%	1	94	6.0%
	7	0.0	0	2	0.2	0	0	0	0.0	0	0.2	44	14.1%	1	98	2.0%
	8	0.0	23	2	0.2	5	1	1	0.1	0	0.3	25	6.3%	1	68	32.0%
	9	0.0	81	1	0.2	14	0	0	0.1	4	0.3	0	0.0%	0	0	100.0%
	10	0.0	7	1	0.5	17	74	1	0.2	0	0.7	0	0.0%	0	0	100.0%
ORL 250	1	0.0	0	0	3.4	0	0	0	0.0	0	3.4	8	41.5%	1	250	0.0%
	2	0.0	0	0	3.2	0	0	0	0.1	0	3.3	1	42.9%	1	250	0.0%
	3	0.0	0	0	3.2	0	0	0	0.1	0	3.3	0	38.8%	1	250	0.0%
	4	0.0	0	0	3.4	0	0	0	0.0	0	3.4	0	45.1%	1	250	0.0%
	5	0.0	0	0	3.3	0	0	0	0.0	0	3.4	0	39.7%	1	250	0.0%
	6	0.0	0	0	3.4	0	0	0	0.0	0	3.5	0	47.8%	1	250	0.0%
	7	0.0	0	0	3.3	0	0	0	0.0	0	3.4	0	41.1%	1	250	0.0%
	8	0.0	0	0	3.2	0	0	0	0.0	0	3.3	0	51.5%	1	250	0.0%
	9	0.0	0	0	3.4	0	0	0	0.0	0	3.4	0	40.0%	1	250	0.0%
	10	0.0	0	0	3.2	0	0	0	0.0	0	3.3	0	46.2%	1	250	0.0%
ORL 500	1	0.1	0	0	28.2	0	0	0	0.2	0	28.5	0	62.9%	1	500	0.0%
	2	0.1	0	0	27.9	0	0	0	0.2	0	28.2	0	58.5%	1	500	0.0%
	3	0.1	0	0	28.8	0	0	0	0.3	0	29.2	0	58.8%	1	500	0.0%
	4	0.1	0	0	27.9	0	0	0	0.2	0	28.3	0	58.8%	1	500	0.0%
	5	0.1	0	0	28.3	0	0	0	0.2	0	28.6	0	60.0%	1	500	0.0%
	6	0.1	0	0	28.0	0	0	0	0.2	0	28.3	0	61.1%	1	500	0.0%
	7	0.1	0	0	28.3	0	0	0	0.2	0	28.6	0	61.2%	1	500	0.0%
	8	0.1	0	0	28.0	0	0	0	0.2	0	28.4	0	60.8%	1	500	0.0%
	9	0.1	0	0	28.1	0	0	0	0.2	0	28.5	0	62.0%	1	500	0.0%
	10	0.1	0	0	28.1	0	0	0	0.2	0	28.5	0	59.1%	1	500	0.0%

Table C.3: PREPRO statistical report on the  $c$ -fat and Hamming graphs.

Problem		Preprocessing Tools Statistics									After Preprocessing					
		Roof-Duality			Probing			Coordination		Total	Quad.	Relat.	Comp. Left		Variab.	
Family	Name	Time ( $t_n$ )	Strong ( $n_s$ )	Weak ( $n_w$ )	Time ( $t_p$ )	Bound. ( $n_b$ )	Fix. ( $n_f$ )	Eq. ( $n_e$ )	Time ( $t_c$ )	Eq. ( $n_c$ )	Time ( $t$ )	Rel. ( $g$ )	Gap ( $g$ )	Num. ( $c$ )	Larg. ( $s$ )	Reduc. ( $r$ )
$c$ -fat	c-fat200-1	0.5	17	3	0.9	5	40	134	19.6	1	21.1	0	0.0%	0	0	100.0%
	c-fat200-2	0.3	53	1	0.8	13	20	113	16.1	0	17.4	0	0.0%	0	0	100.0%
	c-fat200-5	0.0	1	0	0.2	0	30	169	3.3	0	3.7	0	0.0%	0	0	100.0%
	c-fat500-1	6.4	9	3	12.3	12	106	369	655.3	1	677.0	0	0.0%	0	0	100.0%
	c-fat500-2	3.2	29	5	7.9	5	51	409	364.1	1	377.1	0	0.0%	0	0	100.0%
	c-fat500-5	1.3	35	0	4.8	26	11	428	159.1	0	166.2	0	0.0%	0	0	100.0%
	c-fat500-10	0.6	1	0	3.0	0	126	373	85.5	0	89.5	0	0.0%	0	0	100.0%
Hamming	hamming6-2	0.0	0	64	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	hamming8-2	0.0	0	256	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	hamming10-2	0.1	0	1024	0.0	0	0	0	0.0	0	0.1	0	0.0%	0	0	100.0%
	hamming6-4	0.0	0	0	0.1	0	0	0	0.0	0	0.1	0	84.6%	1	64	0.0%
	hamming8-4	0.0	0	0	7.8	0	0	0	1.2	0	8.9	0	86.8%	1	256	0.0%
	hamming10-4	0.1	0	0	229.0	0	0	0	36.2	0	265.4	0	93.6%	1	1024	0.0%

Table C.4: PREPRO statistical report on the minimum vertex cover of the RUDY planar graphs.

Problem		Preprocessing Tools Statistics									After Preprocessing					
		Roof-Duality			Probing				Coordination		Total Time (t)	Quad. Rel. (q)	Relat. Gap (g)	Comp. Left		Variab. Reduc. (r)
		Time (t <sub>n</sub> )	Strong (n <sub>s</sub> )	Weak (n <sub>w</sub> )	Time (t <sub>p</sub> )	Bound. (n <sub>b</sub> )	Fix. (n <sub>f</sub> )	Eq. (n <sub>e</sub> )	Time (t <sub>c</sub> )	Eq. (n <sub>c</sub> )				Num. (c)	Larg. (s)	
Family	Name															
50 000 (10%)	50000-10-1	26.2	9581	40419	0.0	0	0	0	0.0	0	65.7	0	0.0%	0	0	100.0%
	50000-10-2	23.7	9377	40619	0.0	0	0	4	0.0	0	63.3	0	0.0%	0	0	100.0%
	50000-10-3	23.3	9311	40689	0.0	0	0	0	0.0	0	62.5	0	0.0%	0	0	100.0%
50 000 (50%)	50000-50-1	194.9	23591	26295	0.0	0	24	86	0.0	4	206.3	0	0.0%	0	0	100.0%
	50000-50-2	197.3	23560	26342	0.0	0	23	73	0.0	2	209.0	0	0.0%	0	0	100.0%
	50000-50-3	204.4	23673	26208	0.0	0	30	88	0.0	1	216.2	0	0.0%	0	0	100.0%
50 000 (90%)	50000-90-1	85.3	8120	41871	0.0	0	0	4	0.0	5	94.0	0	0.0%	0	0	100.0%
	50000-90-2	85.2	8338	41639	0.0	0	6	12	0.1	5	94.6	0	0.0%	0	0	100.0%
	50000-90-3	86.2	8575	41408	0.0	0	4	10	0.1	3	94.8	0	0.0%	0	0	100.0%
100 000 (10%)	100000-10-1	109.0	18986	81002	0.0	0	2	10	0.0	0	266.5	0	0.0%	0	0	100.0%
	100000-10-2	99.5	19138	80858	0.0	0	0	4	0.0	0	256.4	0	0.0%	0	0	100.0%
	100000-10-3	105.4	18929	81067	0.0	0	0	4	0.0	0	263.1	0	0.0%	0	0	100.0%
100 000 (50%)	100000-50-1	836.7	47472	52358	0.0	0	39	126	0.0	5	883.3	0	0.0%	0	0	100.0%
	100000-50-2	871.4	47671	52145	0.0	0	40	141	0.0	3	917.2	0	0.0%	0	0	100.0%
	100000-50-3	887.8	47900	51974	0.0	1	32	93	0.0	0	933.5	0	0.0%	0	0	100.0%
100 000 (90%)	100000-90-1	354.5	16065	83897	0.0	1	4	20	0.2	13	386.8	0	0.0%	0	0	100.0%
	100000-90-2	360.0	16517	83426	0.0	1	9	32	0.2	15	392.7	0	0.0%	0	0	100.0%
	100000-90-3	358.0	16675	83286	0.0	0	4	20	0.2	15	391.7	0	0.0%	0	0	100.0%
250 000 (10%)	250000-10-1	634.6	47140	202852	0.0	0	1	7	0.0	0	1619.8	0	0.0%	0	0	100.0%
	250000-10-2	671.5	47175	202817	0.0	0	1	7	0.0	0	1655.7	0	0.0%	0	0	100.0%
	250000-10-3	689.2	47367	202625	0.0	0	1	7	0.0	0	1669.5	0	0.0%	0	0	100.0%
250 000 (50%)	250000-50-1	5754.9	119519	129972	0.0	0	109	392	0.0	8	6047.3	0	0.0%	0	0	100.0%
	250000-50-2	5612.8	119704	129840	0.0	1	111	335	0.2	9	5906.4	0	0.0%	0	0	100.0%
	250000-50-3	6054.2	120021	129512	0.0	1	102	356	0.1	8	6346.9	0	0.0%	0	0	100.0%
250 000 (90%)	250000-90-1	2304.1	42022	207847	0.0	1	31	75	0.6	24	2507.9	0	0.0%	0	0	100.0%
	250000-90-2	2370.9	43822	206093	0.0	0	15	57	0.5	13	2574.0	0	0.0%	0	0	100.0%
	250000-90-3	2318.8	41872	207970	0.0	0	59	77	0.6	22	2528.4	0	0.0%	0	0	100.0%
500 000 (10%)	500000-10-1	3026.0	94462	405510	0.0	0	5	23	0.0	0	6943.9	0	0.0%	0	0	100.0%
	500000-10-2	2850.7	94371	405609	0.0	0	4	16	0.0	0	6767.6	0	0.0%	0	0	100.0%
	500000-10-3	2680.8	95190	404798	0.0	0	2	10	0.0	0	6601.3	0	0.0%	0	0	100.0%
500 000 (50%)	500000-50-1	22013.2	240672	258449	0.0	1	205	656	0.2	17	23175.5	0	0.0%	0	0	100.0%
	500000-50-2	24020.2	238932	260334	0.0	1	163	543	0.2	27	25188.5	0	0.0%	0	0	100.0%
	500000-50-3	22178.5	239687	259501	0.1	0	165	632	0.3	15	23340.3	0	0.0%	0	0	100.0%
500 000 (90%)	500000-90-1	9564.2	86359	413422	0.0	1	46	108	1.3	64	10363.3	0	0.0%	0	0	100.0%
	500000-90-2	9427.4	83741	416025	0.0	1	40	136	1.3	57	10219.4	0	0.0%	0	0	100.0%
	500000-90-3	9547.5	84980	414793	0.0	0	47	134	1.4	46	10364.0	0	0.0%	0	0	100.0%

Table C.5: PREPRO statistical report on the MAX-CUT graphs.

Problem		Preprocessing Tools Statistics									After Preprocessing					
		Roof-Duality			Probing				Coordination		Total Time (t)	Quad. Rel. (q)	Relat. Gap (g)	Comp. Left		Variab. Reduc. (r)
		Time (t <sub>n</sub> )	Strong (n <sub>s</sub> )	Weak (n <sub>w</sub> )	Time (t <sub>p</sub> )	Bound. (n <sub>b</sub> )	Fix. (n <sub>f</sub> )	Eq. (n <sub>e</sub> )	Time (t <sub>c</sub> )	Eq. (n <sub>c</sub> )				Num. (c)	Larg. (s)	
Family	Name															
Torus	pm3-8-50	0.0	0	0	3.7	0	0	0	0.2	0	4.0	0	43.6%	1	511	0.2%
	pm3-15-50	0.1	0	0	267.0	0	0	0	11.8	0	279.3	0	45.5%	1	3374	0.0%
	g3-8	0.1	1	0	7.1	0	0	31	7.2	0	14.7	3	31.2%	1	479	6.4%
	g3-15	6.7	0	0	509.7	0	0	196	2513.8	0	3115.0	4	33.0%	1	3178	5.8%
R	R1000	0.1	0	0	46.8	0	0	1	5.7	0	52.7	0	29.0%	1	998	0.2%
	R2000	0.3	0	0	224.3	0	0	4	64.7	0	290.1	0	28.6%	1	1995	0.2%
	R3000	0.6	0	0	525.9	0	0	7	247.8	0	777.4	6	28.8%	1	2992	0.3%
	R4000	0.7	0	0	1205.3	0	0	7	436.1	0	1647.5	0	28.7%	1	3992	0.2%
	R5000	1.3	0	0	1697.1	0	0	12	1150.6	0	2862.9	0	28.6%	1	4987	0.3%
	R6000	1.7	0	0	2152.8	0	0	12	1625.3	0	3799.1	0	28.8%	1	5987	0.2%
	R7000	2.6	0	2	3784.1	0	0	16	2952.4	0	6773.9	0	28.8%	1	6981	0.3%
	R8000	3.5	0	0	4752.3	0	0	19	4489.5	0	9298.4	3	28.7%	1	7980	0.2%
Via	via.c1n	0.1	455	115	0.3	4	84	169	1.0	0	1.5	0	0.0%	0	0	100.0%
	via.c2n	0.4	102	67	3.3	3	18	789	45.5	0	52.6	0	0.0%	0	0	100.0%
	via.c3n	0.9	147	18	10.2	3	9	1149	76.3	0	94.5	0	0.0%	0	0	100.0%
	via.c4n	1.1	151	18	10.2	0	5	1191	185.8	0	208.9	0	0.0%	0	0	100.0%
	via.c5n	0.9	21	23	6.8	1	66	1090	129.4	0	145.1	0	0.0%	0	0	100.0%
	via.c1y	0.0	775	53	0.0	0	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	via.c2y	0.1	928	52	0.0	0	0	0	0.0	0	0.1	0	0.0%	0	0	100.0%
	via.c3y	0.1	1256	71	0.0	0	0	0	0.0	0	0.1	0	0.0%	0	0	100.0%
	via.c4y	0.1	1317	49	0.0	0	0	0	0.0	0	0.1	0	0.0%	0	0	100.0%
	via.c5y	0.1	1119	83	0.0	0	0	0	0.0	0	0.1	0	0.0%	0	0	100.0%
G-500	G500.2.5	0.1	0	72	1.0	0	0	235	7.6	0	9.8	12	7.7%	1	192	61.6%
	G500.05	0.2	0	7	4.9	0	0	60	13.1	0	18.8	9	19.1%	1	432	13.6%
	G500.10	0.0	0	0	5.9	0	0	1	1.0	0	6.9	0	27.2%	1	498	0.4%
	G500.20	0.0	0	0	10.8	0	0	0	1.0	0	11.9	0	35.2%	1	499	0.2%
G-1000	G1000.2.5	0.6	1	95	6.3	0	0	521	91.2	0	108.5	18	8.3%	1	382	61.8%
	G1000.05	1.0	0	6	30.7	0	0	121	154.6	0	191.0	9	19.8%	1	872	12.8%
	G1000.10	0.1	0	0	40.0	0	0	2	8.8	0	49.0	0	29.1%	1	997	0.3%
	G1000.20	0.1	0	0	50.4	0	0	0	4.3	0	54.8	0	35.0%	1	999	0.1%
U-500	U500.05	0.1	14	39	1.1	0	8	150	4.9	0	7.3	59	20.2%	4	151	42.4%
	U500.10	0.1	0	4	12.0	0	0	11	7.5	0	20.1	37	34.0%	1	484	3.2%
	U500.20	0.2	0	0	8.0	0	0	0	2.4	0	10.9	25	39.1%	1	499	0.2%
	U500.40	0.1	0	0	14.5	0	0	0	4.2	0	18.9	7	41.7%	1	499	0.2%
U-1000	U1000.05	0.9	16	89	8.3	0	6	264	57.5	0	74.5	160	21.6%	3	316	37.6%
	U1000.10	0.7	0	6	43.7	0	0	12	43.0	0	91.3	96	34.4%	1	981	1.9%
	U1000.20	0.7	0	0	36.5	0	0	0	12.0	0	51.7	44	38.8%	1	999	0.1%
	U1000.40	0.8	0	0	63.0	0	0	0	21.8	0	87.2	30	41.9%	1	999	0.1%

Table C.6: PREPRO statistical report on the MAX-2-SAT formulas of Borchers and Furman [47].

Problem		Preprocessing Tools Statistics									After Preprocessing					
		Roof-Duality			Probing				Coordination		Total Time (t)	Quad. Rel. (q)	Relat. Gap (g)	Comp. Left		Variab. Reduc. (r)
		Time (t <sub>n</sub> )	Strong (n <sub>s</sub> )	Weak (n <sub>w</sub> )	Time (t <sub>p</sub> )	Bound. (n <sub>b</sub> )	Fix. (n <sub>f</sub> )	Eq. (n <sub>e</sub> )	Time (t <sub>c</sub> )	Eq. (n <sub>c</sub> )				Num. (c)	Larg. (s)	
Family	Name															
BF-50	BF-50-100	0.0	2	38	0.0	0	0	6	0.0	4	0.0	0	0.0%	0	0	100.0%
	BF-50-150	0.0	1	18	0.0	4	15	12	0.0	0	0.0	0	0.0%	0	0	100.0%
	BF-50-200	0.0	1	5	0.0	0	0	2	0.0	0	0.0	15	100.0%	1	42	16.0%
	BF-50-250	0.0	0	1	0.1	2	0	2	0.0	0	0.2	14	109.5%	1	45	10.0%
	BF-50-300	0.0	0	0	0.0	0	0	0	0.0	0	0.0	14	204.8%	1	50	0.0%
	BF-50-350	0.0	0	0	0.1	1	0	0	0.0	0	0.1	13	186.7%	1	49	2.0%
	BF-50-400	0.0	1	0	0.1	1	0	0	0.0	0	0.1	14	141.0%	1	48	4.0%
	BF-50-450	0.0	0	0	0.0	0	0	0	0.0	0	0.1	3	190.9%	1	50	0.0%
BF-50-500	0.0	0	0	0.0	0	0	0	0.0	0	0.0	0	157.7%	1	50	0.0%	
BF-100	BF-100-200	0.0	2	50	0.0	0	17	31	0.0	0	0.0	0	0.0%	0	0	100.0%
	BF-100-300	0.0	0	14	0.2	0	0	9	0.1	0	0.2	49	328.6%	1	77	23.0%
	BF-100-400	0.0	0	10	0.2	0	0	1	0.0	0	0.2	43	215.8%	1	89	11.0%
	BF-100-500	0.0	0	3	0.1	0	0	0	0.0	0	0.1	25	475.0%	1	97	3.0%
	BF-100-600	0.0	0	1	0.1	0	0	0	0.0	0	0.1	6	340.0%	1	99	1.0%
BF-150	BF-150-300	0.0	1	79	0.1	0	31	34	0.1	5	0.2	0	0.0%	0	0	100.0%
	BF-150-450	0.0	0	22	0.4	0	0	5	0.1	0	0.5	85	475.0%	1	123	18.0%
	BF-150-600	0.0	0	4	0.6	0	3	6	0.2	0	0.8	80	300.0%	1	137	8.7%
BFW-50	BFW-50-100	0.0	4	13	0.0	2	14	17	0.0	0	0.0	0	0.0%	0	0	100.0%
	BFW-50-150	0.0	11	28	0.1	5	1	4	0.0	1	0.1	0	0.0%	0	0	100.0%
	BFW-50-200	0.0	1	3	0.1	1	0	1	0.0	0	0.1	14	102.8%	1	44	12.0%
	BFW-50-250	0.0	0	0	0.1	0	0	0	0.0	0	0.1	32	209.1%	1	50	0.0%
	BFW-50-300	0.0	0	1	0.1	0	0	0	0.0	0	0.1	12	149.1%	1	49	2.0%
	BFW-50-350	0.0	0	0	0.1	0	0	0	0.0	0	0.1	6	236.2%	1	50	0.0%
	BFW-50-400	0.0	0	0	0.1	2	0	0	0.0	0	0.1	4	126.5%	1	48	4.0%
	BFW-50-450	0.0	0	0	0.1	0	0	0	0.0	0	0.1	1	169.1%	1	50	0.0%
BFW-50-500	0.0	0	0	0.1	0	0	0	0.0	0	0.1	0	205.7%	1	50	0.0%	
BFW-100	BFW-100-200	0.0	54	45	0.0	1	0	0	0.0	0	0.0	0	0.0%	0	0	100.0%
	BFW-100-300	0.0	1	10	0.4	1	0	10	0.1	0	0.4	69	193.6%	1	78	22.0%
	BFW-100-400	0.0	0	2	0.4	0	1	5	0.1	0	0.5	81	476.7%	1	92	8.0%
	BFW-100-500	0.0	0	1	0.1	0	0	0	0.0	0	0.2	25	557.1%	1	99	1.0%
	BFW-100-600	0.0	0	0	0.2	0	0	0	0.0	0	0.2	18	450.0%	1	100	0.0%
BFW-150	BFW-150-300	0.0	39	55	0.5	4	7	44	0.3	1	0.8	0	0.0%	0	0	100.0%
	BFW-150-450	0.0	1	24	0.9	2	2	15	0.3	0	1.2	142	175.0%	1	106	29.3%
	BFW-150-600	0.0	0	6	0.6	0	3	3	0.1	0	0.7	59	620.7%	1	138	8.0%

**Appendix D**  
**Iterated Roof-Duality Experiments**

Table D.1: MAX-CUT (Kim et al. [157]).

(a) Upper bounds.

Family	Problem Name	Vert. (n)	Edges	Max. Cut	Upper Bounds to the Maximum		
					Semidefinite Relaxation	Roof Dual ( $\rho$ )	Iter. Roof Dual ( $\hat{\rho}$ )
$G_{n,d}$	G500.2.5	500	625	574	598.15	620.5	<b>590.50</b>
	G500.05		1 223	$\geq 1 008$	<b>1070.06</b>	1 217.0	1 086.00
	G500.10		2 355	$\geq 1 735$	<b>1847.97</b>	2 346.0	1 960.64
	G500.20		5 120	$\geq 3 390$	<b>3566.74</b>	5 103.0	4 006.70
	G1000.2.5	1 000	1 272	$\geq 1 173$	1 223.01	1 268.5	<b>1212.91</b>
	G1000.05		2 496	$\geq 2 053$	<b>2191.80</b>	2 490.5	2 232.66
	G1000.10		5 064	$\geq 3 705$	<b>3954.67</b>	5 052.5	4 245.18
	G1000.20		10 107	$\geq 6 729$	<b>7105.60</b>	10 090.0	8 059.02
$U_{n,d}$	U500.05	500	1 282	900	<b>922.42</b>	1 274.0	962.00
	U500.10		2 355	$\geq 1 546$	<b>1587.86</b>	2 345.0	1 716.09
	U500.20		4 549	$\geq 2 783$	<b>2864.27</b>	4 534.0	3 229.47
	U500.40		8 793	$\geq 5 181$	<b>5303.45</b>	8 765.0	6 164.78
	U1000.05	1 000	2 394	$\geq 1 711$	<b>1752.76</b>	2 388.5	1 830.25
	U1000.10		4 696	$\geq 3 073$	<b>3158.95</b>	4 686.5	3 424.81
	U1000.20		9 339	$\geq 5 737$	<b>5890.78</b>	9 319.5	6 617.69
	U1000.40		18 015	$\geq 10 560$	<b>10851.01</b>	17 986.0	12 593.03

(b) Computing times.

Problem	Semidefinite Programs			Roof-Dual Algorithms	
	DSDP**	SBM**	SDPA*	RDA*	IRDA*
G500.2.5	3.35 s	1 279.20 s	85.84 s	<0.005 s	0.05 s
G500.05	3.72 s	70.67 s	105.98 s	<0.005 s	0.09 s
G500.10	4.80 s	4.28 s	104.16 s	<0.005 s	0.31 s
G500.20	8.54 s	4.64 s	106.08 s	<0.005 s	0.84 s
G1000.2.5	17.85 s	4 056.78 s	694.05 s	<0.005 s	0.23 s
G1000.05	29.56 s	138.02 s	843.69 s	<0.005 s	0.53 s
G1000.10	47.80 s	11.37 s	871.16 s	<0.005 s	1.47 s
G1000.20	69.98 s	15.00 s	826.34 s	0.02 s	3.36 s
U500.05	3.71 s	2179.61 s	2179.61 s	<0.005 s	0.17 s
U500.10	3.15 s	52.15 s	107.58 s	<0.005 s	0.38 s
U500.20	3.74 s	6.32 s	111.56 s	<0.005 s	0.89 s
U500.40	4.41 s	3.85 s	106.80 s	<0.005 s	1.81 s
U1000.05	21.96 s	3596.08 s	794.62 s	<0.005 s	0.82 s
U1000.10	20.79 s	403.90 s	837.03 s	<0.005 s	1.88 s
U1000.20	22.87 s	22.24 s	871.67 s	<0.005 s	3.91 s
U1000.40	29.90 s	13.87 s	941.30 s	0.02 s	7.66 s

\*Computed on computer system I.

\*\*Computed on computer system II.

Table D.2: MAX-CUT (Homer and Peinado [145]).

(a) Upper bounds.

Family	Problem Name	Vertices (n)	Edges	Max. Cut	Upper Bounds to the Maximum		
					Semidefinite Relaxation	Roof Dual ( $\rho$ )	Iter. Roof Dual ( $\hat{\rho}$ )
R	R1000	1 000	5 033	$\geq 3\,687$	<b>3934.5</b>	5 021.5	4 220.96
	R2000	2 000	9 943	$\geq 7\,308$	<b>7820.0</b>	9 932.5	8 454.98
	R3000	3 000	14 965	$\geq 10\,997$	<b>11790.6</b>	14 953.5	12 800.36
	R4000	4 000	19 939	$\geq 14\,684$	<b>15729.2</b>	19 927.0	17 118.74
	R5000	5 000	24 794	$\geq 18\,225$	<b>19587.1</b>	24 783.0	21 362.02
	R6000	6 000	29 862	$\geq 21\,937$	<b>23602.7</b>	29 849.0	25 798.04
	R7000	7 000	35 110	$\geq 25\,763$	<b>27730.6</b>	35 097.0	30 363.51
	R8000	8 000	39 642	$\geq 29\,140$	<b>31382.1</b>	39 629.5	34 375.45
via	via.c1n	828	1 389	6 150	6 182.42	6 339.0	<b>6150.00</b>
	via.c2n	980	1 712	7 098	7 117.75	7 473.0	<b>7098.00</b>
	via.c3n	1 327	2 393	6 898	6 943.72	7 282.0	<b>6906.25</b>
	via.c4n	1 366	2 539	10 098	10 110.59	10 437.0	<b>10098.00</b>
	via.c5n	1 202	2 129	7 956	8 003.15	8 427.0	<b>7962.00</b>
	via.c1y	829	1 693	7 746	7 795.87	<b>7746.0</b>	<b>7746.00</b>
	via.c2y	981	2 039	8 226	8 276.36	<b>8226.0</b>	<b>8226.00</b>
	via.c3y	1 328	2 757	9 502	9 572.56	<b>9502.0</b>	<b>9502.00</b>
	via.c4y	1 367	2 848	12 516	12 556.58	<b>12516.0</b>	<b>12516.00</b>
	via.c5y	1 203	2 452	10 248	10 327.99	<b>10248.0</b>	<b>10248.00</b>

(b) Computing times.

Problem	Semidefinite Programs			Roof-Dual Algorithms	
	DSDP**	SBM**	SDPA*	RDA*	IRDA*
R1000	43.15 s	13.09 s	815.81 s	0.02 s	1.47 s
R2000	389.74 s	55.28 s	8 053.94 s	0.02 s	5.75 s
R3000	8 751.11 s	102.93 s	n/a <sup>†</sup>	0.02 s	13.00 s
R4000	2 492.46 s	244.18 s	n/a <sup>†</sup>	0.02 s	22.74 s
R5000	4 625.69 s	352.35 s	n/a <sup>†</sup>	0.02 s	35.59 s
R6000	9 055.98 s	472.75 s	n/a <sup>†</sup>	0.03 s	50.50 s
R7000	13 566.78 s	1 725.05 s	n/a <sup>†</sup>	0.03 s	69.23 s
R8000	18 212.75 s	853.85 s	n/a <sup>†</sup>	0.05 s	87.92 s
via.c1n	14.23 s	92.94 s	506.45 s	0.02 s	0.09 s
via.c2n	20.02 s	133.56 s	825.53 s	0.02 s	0.39 s
via.c3n	47.97 s	376.64 s	2 242.39 s	0.02 s	0.16 s
via.c4n	51.55 s	239.53 s	2 431.20 s	0.03 s	0.19 s
via.c5n	34.69 s	227.62 s	1 644.06 s	0.00 s	0.13 s
via.c1y	19.97 s	493.52 s	576.74 s	0.03 s	0.05 s
via.c2y	28.83 s	636.06 s	944.25 s	0.03 s	0.05 s
via.c3y	70.66 s	2 502.97 s	2 363.20 s	0.05 s	0.08 s
via.c4y	77.51 s	1,098.55 s	2 849.97 s	0.05 s	0.06 s
via.c5y	54.55 s	3716.97 s	1 759.91 s	0.03 s	0.06 s

\*Computed on computer system I.

\*\*Computed on computer system II.

<sup>†</sup>Memory exceeded.

Table D.3: MAX-CUT on cubic lattice graphs (Burer et al. [74]).

(a) Upper bounds.

Family	Problem Name	Vert. (n)	Edges	Max. Cut	Upper Bounds to the Maximum		
					Semidefinite Relaxation	Roof Dual ( $\rho$ )	Iter. Roof Dual ( $\hat{\rho}$ )
sg3dl05	sg3dl051000	125	375	110	126.53	185	<b>126.38</b>
	sg3dl052000	125	375	112	128.20	185	<b>127.73</b>
	sg3dl053000	125	375	106	<b>123.98</b>	185	126.50
	sg3dl054000	125	375	114	128.18	185	<b>124.09</b>
	sg3dl055000	125	375	112	127.06	185	<b>129.13</b>
	sg3dl056000	125	375	110	126.88	185	<b>128.25</b>
	sg3dl057000	125	375	112	126.81	185	<b>126.00</b>
	sg3dl058000	125	375	108	125.48	185	<b>123.38</b>
	sg3dl059000	125	375	110	126.00	185	<b>127.25</b>
	sg3dl0510000	125	375	112	127.68	185	<b>124.31</b>
sg3dl10	sg3dl101000	1 000	3 000	$\geq 896$	1 025.91	1 497	<b>1001.31</b>
	sg3dl102000	1 000	3 000	$\geq 900$	1 036.47	1 497	<b>1008.46</b>
	sg3dl103000	1 000	3 000	$\geq 892$	1 021.92	1 497	<b>1003.93</b>
	sg3dl104000	1 000	3 000	$\geq 898$	1 031.34	1 497	<b>1011.13</b>
	sg3dl105000	1 000	3 000	$\geq 886$	1 021.29	1 497	<b>1001.17</b>
	sg3dl106000	1 000	3 000	$\geq 888$	1 023.34	1 497	<b>1001.66</b>
	sg3dl107000	1 000	3 000	$\geq 900$	1 030.06	1 497	<b>1014.06</b>
	sg3dl108000	1 000	3 000	$\geq 882$	1 023.74	1 497	<b>1006.17</b>
	sg3dl109000	1 000	3 000	$\geq 902$	1 029.24	1 497	<b>1010.40</b>
	sg3dl1010000	1 000	3 000	$\geq 894$	1 027.65	1 497	<b>1005.20</b>
sg3dl14	sg3dl141000	2 744	8 232	$\geq 2 446$	2816.90	4 113	<b>2773.56</b>
	sg3dl142000	2 744	8 232	$\geq 2 458$	2825.79	4 113	<b>2762.61</b>
	sg3dl143000	2 744	8 232	$\geq 2 442$	2815.40	4 113	<b>2762.61</b>
	sg3dl144000	2 744	8 232	$\geq 2 450$	2817.45	4 113	<b>2764.10</b>
	sg3dl145000	2 744	8 232	$\geq 2 446$	2809.86	4 113	<b>2772.49</b>
	sg3dl146000	2 744	8 232	$\geq 2 450$	2822.92	4 113	<b>2765.19</b>
	sg3dl147000	2 744	8 232	$\geq 2 444$	2813.08	4 113	<b>2757.21</b>
	sg3dl148000	2 744	8 232	$\geq 2 446$	2818.70	4 113	<b>2771.18</b>
	sg3dl149000	2 744	8 232	$\geq 2 424$	2793.42	4 113	<b>2744.38</b>
	sg3dl1410000	2 744	8 232	$\geq 2 458$	2826.35	4 113	<b>2763.24</b>

(b) Average computing times.

Family	Semidefinite Programs			Roof-Dual Algorithms	
	DSDP**	SBM**	SDPA*	RDA*	IRDA*
sg3dl05	0.19 s	0.87 s	1.72 s	<0.005 s	0.01 s
sg3dl10	25.29 s	14.07 s	871.23 s	<0.005 s	0.91 s
sg3dl14	431.17 s	113.09 s	n/a <sup>†</sup>	0.01 s	7.16 s

\*Computed on computer system I.

\*\*Computed on computer system II.

<sup>†</sup>Memory exceeded.

Table D.4: MAX-CUT for torus graphs (7th DIMACS Implementation Challenge).

(a) Upper bounds.

Problem Name	Vert. ( $n$ )	Edges	Max. Cut	Upper Bounds to the Maximum		
				Semidefinite Relaxation	Roof Dual ( $\rho$ )	Iter. Roof Dual ( $\hat{\rho}$ )
toruspm3-8-50	512	1 536	458	527.81	765.0	<b>523.05</b>
toruspm3-15-50	3 375	10 125	$\geq 3\,016$	3 475.13	5 060.0	<b>3414.49</b>
torusg3-8	512	1 536	41 684 814	45 735 854.8	58 921 474.5	<b>45100733.03</b>
torusg3-15	3 375	10 125	$\geq 285\,790\,637$	313 457 107.3	402 667 673.0	<b>308433472.25</b>

(b) Computing times.

Problem	Semidefinite Programs			Roof-Dual Algorithms	
	DSDP**	SBM**	SDPA*	RDA*	IRDA*
toruspm3-8-50	4.16 s	5.00 s	114.78 s	<0.005 s	0.22 s
toruspm3-15-50	763.58 s	226.89 s	n/a <sup>†</sup>	0.02 s	10.92 s
torusg3-8	8.03 s	6.35 s	186.98 s	<0.005 s	0.14 s
torusg3-15	1301.45 s	224.62 s	n/a <sup>†</sup>	0.02 s	6.70 s

\* Computed on computer system I.

\*\* Computed on computer system II.

<sup>†</sup> Memory exceeded.

Table D.5: Upper bounds of 10% dense QUBO maximization problems (Beasley [37]).

Family ( $n$ )	Problem Number	Maximum	Upper Bounds to the Maximum		
			Semidefinite Relaxation	Roof Dual ( $\rho$ )	Iter. Roof Dual ( $\hat{\rho}$ )
ORL-100  (100)	1	7 970	<b>8721.11</b>	10 160.5	8 725.50
	2	11 036	11 704.18	12 285.5	<b>11245.50</b>
	3	12 723	13 336.70	13 664.5	<b>12864.00</b>
	4	10 368	10 927.93	12 099.0	<b>10656.00</b>
	5	9 083	9 736.93	10 617.0	<b>9339.50</b>
	6	10 210	11 073.07	13 086.5	<b>11042.00</b>
	7	10 125	10 906.86	12 016.5	<b>10489.00</b>
	8	11 435	12 078.48	12 638.0	<b>11542.06</b>
	9	11 455	11 926.97	12 235.0	<b>11581.00</b>
	10	12 565	13 151.28	13 686.0	<b>12749.00</b>
ORL-250  (250)	1	45 607	<b>48732.37</b>	78 321.0	52 528.63
	2	44 810	<b>48093.50</b>	78 258.5	52 728.61
	3	49 037	<b>51745.40</b>	80 919.0	55 145.06
	4	41 274	<b>44391.58</b>	75 411.0	49 577.34
	5	47 961	<b>50803.63</b>	79 972.5	54 165.75
	6	$\geq 41 014$	<b>44547.53</b>	78 452.5	50 704.70
	7	46 757	<b>49709.76</b>	80 040.0	54 096.54
	8	$\geq 35 726$	<b>40005.60</b>	72 599.5	46 508.41
	9	48 916	<b>52330.23</b>	81 838.5	56 244.03
	10	40 442	<b>44026.14</b>	75 752.5	49 320.57
ORL-500  (500)	1	$\geq 116 586$	<b>128402.72</b>	308 706.5	171 327.46
	2	$\geq 128 339$	<b>138237.20</b>	309 825.5	175 209.92
	3	$\geq 130 812$	<b>140738.05</b>	317 653.5	181 089.10
	4	$\geq 130 097$	<b>141602.11</b>	315 733.0	180 490.21
	5	$\geq 125 487$	<b>136578.72</b>	311 891.5	176 444.61
	6	$\geq 121 772$	<b>132960.20</b>	310 139.5	173 953.18
	7	$\geq 122 201$	<b>134273.56</b>	312 285.5	175 700.29
	8	$\geq 123 559$	<b>135438.79</b>	313 878.5	177 959.90
	9	$\geq 120 798$	<b>132615.73</b>	312 183.0	175 938.65
	10	$\geq 130 619$	<b>141076.28</b>	317 514.5	180 860.23
ORL-1000  (1 000)	1	$\geq 371 438$	<b>403684.0</b>	1 256 488.0	627 769.04
	2	$\geq 354 932$	<b>390028.8</b>	1 251 578.0	619 465.32
	3	$\geq 371 236$	<b>404445.7</b>	1 263 836.0	628 844.76
	4	$\geq 370 675$	<b>403911.4</b>	1 269 344.0	633 482.54
	5	$\geq 352 760$	<b>388304.0</b>	1 260 413.5	622 233.15
	6	$\geq 359 629$	<b>392175.5</b>	1 257 474.5	620 005.81
	7	$\geq 371 193$	<b>405621.7</b>	1 259 282.5	628 620.56
	8	$\geq 351 994$	<b>388940.6</b>	1 253 255.0	620 828.19
	9	$\geq 349 337$	<b>385204.7</b>	1 254 976.0	617 834.57
	10	$\geq 351 415$	<b>385664.6</b>	1 240 515.5	613 573.00
ORL-2500  (2 500)	1	$\geq 1 515 944$	<b>1652473.3</b>	7 886 424.0	3 417 034.96
	2	$\geq 1 471 392$	<b>1614710.7</b>	7 843 106.0	3 384 231.31
	3	$\geq 1 414 192$	<b>1558172.2</b>	7 810 572.5	3 354 353.86
	4	$\geq 1 507 701$	<b>1642588.4</b>	7 860 349.5	3 408 753.16
	5	$\geq 1 491 816$	<b>1626210.5</b>	7 858 834.0	3 390 133.74
	6	$\geq 1 469 162$	<b>1608890.8</b>	7 827 394.0	3 377 820.75
	7	$\geq 1 479 040$	<b>1619037.2</b>	7 852 577.0	3 386 395.44
	8	$\geq 1 484 199$	<b>1616263.5</b>	7 831 767.5	3 381 058.32
	9	$\geq 1 482 413$	<b>1622399.3</b>	7 868 242.0	3 400 093.17
	10	$\geq 1 483 355$	<b>1625693.3</b>	7 840 749.5	3 392 232.40

## References

- [1] The dimacs library of mixed semidefinite-quadratic-linear programs (November 2003).  
URL <http://dimacs.rutgers.edu/Challenges/Seventh/Instances>.
- [2] Hearin Center for Enterprise Science: Benchmarks for unconstrained binary quadratic problems (March 2003).  
URL <http://hces.bus.olemiss.edu/tools.html>.
- [3] MAXSAT, a Davis-Putnam like code for MAX-SAT problems (December 2004).  
URL <http://www.nmt.edu/~borchers/maxsat.html>.
- [4] The CAIDA data page for comparative analysis of the Internet AS-level topologies (April 2006).  
URL [http://www.caida.org/analysis/topology/as\\_topo\\_comparisons](http://www.caida.org/analysis/topology/as_topo_comparisons).
- [5] CAIDA Internet router-level graph (April 2006).  
URL [http://www.caida.org/tools/measurement/skitter/router\\_topology](http://www.caida.org/tools/measurement/skitter/router_topology).
- [6] CCNR network databases (April 2006).  
URL <http://www.nd.edu/networks/resources.htm>.
- [7] The Erdős Number project (April 2006).  
URL <http://www.oakland.edu/enp/index.html>.
- [8] KEGG Biomolecular Relations in Information Transmission and Expression (BRITE) database generalized protein interactions (April 2006).  
URL [http://www.genome.jp/brite/generalized\\_interactions.html](http://www.genome.jp/brite/generalized_interactions.html).
- [9] Ilog Cplex (March 2008).  
URL <http://www.ilog.com/products/cplex>.
- [10] National Laboratory for Applied Network Research (NLANR) project (March 2008).  
URL <http://moat.nlanr.net>.
- [11] Abu-Khzam, F.N., R.L. Collins, M.R. Fellows, M.A. Langston, W.H. Suters and C.T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX)* (New Orleans, Louisiana, 2004).
- [12] Adams, W.P. and P.M. Dearing. On the equivalence between roof duality and lagrangian duality for unconstrained 0-1 quadratic programming. *Discrete Applied Mathematics* **48**, (1994), pp. 1–20.

- [13] Alber, J., F. Dorn and R. Niedermeier. Experimental evaluation of a tree decomposition based algorithm for vertex cover on planar graphs. *Discrete Applied Mathematics* **145**, (2005), pp. 219–231.
- [14] Alexe, G., P.L. Hammer, V.V. Lozin and D. de Werra. Struction revisited. *Discrete Applied Mathematics* **132**, (2004), pp. 27–46.
- [15] Alidaee, B., G.A. Kochenberger and A. Ahmadian. 0-1 quadratic programming approach for the optimal solution of two scheduling problems. *International Journal of Systems Science* **25**, (1994), pp. 401–408.
- [16] Alsinet, T., F. Manyá and J. Planes. Improved branch and bound algorithms for max-2-sat and weighted max-2-sat. In *Proceedings of sixth Catalan Conference on Artificial Intelligence (CCIA 2003)* (2003).
- [17] Alsinet, T., F. Manyá and J. Planes. Improved branch and bound algorithms for max-sat. In *Proceedings of sixth international conference on theory and applications of satisfiability testing (SAT 2003)* (2003), pp. 408–415.
- [18] Amini, M.M., B. Alidaee and G.A. Kochenberger. A scatter search approach to unconstrained quadratic binary programs. In *New ideas in optimisation* (D. Corne, M. Dorigo and F. Glover, eds.), pp. 317–329 (McGraw-Hill, London, 1999).
- [19] Anjos, M., E. Boros and G. Tavares. A private communication (2007).
- [20] Anjos, M., B. Ghaddar and F. Liers. *A branch-and-cut algorithm based on semi-definite programming for the minimum k-partition problem*. Research report, Combinatorial Optimization in Physics (COPhy) (July 2007).
- [21] Ashford, R. Mixed integer programming: A historical perspective with Xpress-MP. *Annals of Operations Research* **149**(1), (2007), pp. 5–17.
- [22] Aspvall, B., M.F. Plass and R.E. Tarjan. A linear time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters* **8**, (1979), pp. 121–123.
- [23] Atamtürk, A., G.L. Nemhauser and M.W.P. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research* **121**, (2000), pp. 40–55.
- [24] Axehill, D. and A. Hansson. *A preprocessing algorithm for MIQP solvers with applications to MPC*. Tech. rep., Department of Electrical Engineering, Linköping University (2004).
- [25] Badics, T. *Approximation of some nonlinear binary optimization problems*. Ph.D. thesis, RUTCOR, Rutgers University (1996).
- [26] Badics, T. and E. Boros. Minimization of half-products. *Mathematics of Operations Research* **23**, (1998), pp. 649–660.
- [27] Balas, E. and J.B. Mazzola. Nonlinear 0-1 programming: I. Linearization techniques and ii. Dominance relations and algorithms. *Mathematical Programming* **30**, (1984), pp. 1–45.

- [28] Balasundaram, B., S. Butenko and S. Trukhanov. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization* **10**(1), (2005), pp. 23–39.
- [29] Balinski, M.L. On maximum matching, minimum covering and their connection. In *Proceedings of the Princeton symposium on mathematical programming* (H.W. Kuhn, ed.) (Princeton University Press, Princeton, NJ, 1970).
- [30] Balinski, M.L. *Integer Programming: Methods, uses, computation* (W. H. Freeman, San Francisco, 1979).
- [31] Barabási, A.L., H. Jeong, S.P. Mason and Z.N. Oltvai. Centrality and lethality of protein networks. *Nature* **411**, (2001), pp. 41–42.
- [32] Barahona, F., M. Grötschel, M. Jünger and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research* **36**, (1988), pp. 493–513.
- [33] Barahona, F., M. Jünger and G. Reinelt. Experiments in quadratic 0-1 programming. *Mathematical Programming* **44**, (1989), pp. 127–137.
- [34] Barahona, F. and L. Ladányi. Branch and cut based on the volume algorithm: Steiner trees in graphs and max-cut. *RAIRO Recherche Opérationnelle* **40**(1), (2006), pp. 53–73.
- [35] Batagelj, V. and A. Mrvar. Pajek datasets (April 2006).  
URL <http://vlado.fmf.uni-lj.si/pub/networks/data>.
- [36] Beasley, J.E. Or-library: Distributing test problems by electronic mail. *Journal of Operations Research Society* **41**, (1990), pp. 1069–1072.
- [37] Beasley, J.E. *Heuristic algorithms for the unconstrained binary quadratic programming problem*. Tech. rep., Management School, Imperial College, London, UK (1998).
- [38] Beasley, J.E. OR-Library: Unconstrained binary quadratic programming (November 2003).  
URL <http://mscmga.ms.ic.ac.uk/jeb/orlib/bqpinfo.html>.
- [39] Benson, S.J., Y. Ye and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization* **10**(2), (2000), pp. 443–461.
- [40] Berman, P. and A. Pelc. Distributed fault diagnosis for multiprocessor systems. In *Proceedings of the 20th annual international symposium on fault-tolerant computing* (Newcastle, UK, 1990), pp. 340–346.
- [41] Billionnet, A. and S. Elloumi. *Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem*. Rapport Scientifique CEDRIC no. 466, CNAM (2003).
- [42] Billionnet, A. and B. Jaumard. A decomposition method for minimizing quadratic pseudo-Boolean functions. *Operations Research Letters* **8**(3), (1989), pp. 161–163.

- [43] Billionnet, A. and B. Jaumard. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Mathematical Programming* **100**(1), (2007), pp. 55–68.
- [44] Billionnet, A. and A. Sutter. Minimization of a quadratic pseudo-Boolean function. *European Journal of Operational Research* **78**, (1994), pp. 106–115.
- [45] Bomze, I.M., M. Budinich, P.M. Pardalos and M. Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization* (D.Z. Du and P.M. Pardalos, eds.), pp. 38–64 (Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999).
- [46] Bonami, P. and M. Minoux. Exact max-2sat solution via lift-and-project closure. *Operations Research Letters* **34**, (2006), p. 387393.
- [47] Borchers, B. and J. Furman. A two-phase exact algorithm for max-sat and weighted max-sat problems. *Journal of Combinatorial Optimization* **2**(4), (1998), pp. 299–306.
- [48] Boros, E., Y. Crama and P.L. Hammer. Polynomial-time inference of all valid implications for horn and related formulae. *Annals of Mathematics and Artificial Intelligence* **1**, (1990), pp. 21–32.
- [49] Boros, E., Y. Crama and P.L. Hammer. Upper-bounds for quadratic 0-1 maximization. *Operations Research Letters* **9**, (1990), pp. 73–79.
- [50] Boros, E., Y. Crama and P.L. Hammer. Chvátal cuts and odd cycle inequalities in quadratic 0-1 optimization. *SIAM Journal on Discrete Mathematics* **5**(2), (1992), pp. 163–177.
- [51] Boros, E. and P.L. Hammer. *A max-flow approach to improved roof-duality in quadratic 0-1 minimization*. Research Report RRR 15-1989, RUTCOR, Rutgers University (1989).
- [52] Boros, E. and P.L. Hammer. The max-cut problem and quadratic 0-1 optimization, polyhedral aspects, relaxations and bounds. *Annals of Operations Research* **33**, (1991), pp. 151–180.
- [53] Boros, E. and P.L. Hammer. Cut-polytopes, Boolean quadric polytopes and nonnegative quadratic pseudo-Boolean functions. *Mathematics of Operations Research* **18**, (1993), pp. 245–253.
- [54] Boros, E. and P.L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics* **123**, (2002), pp. 155–225.
- [55] Boros, E., P.L. Hammer, M. Minoux and D. Rader. Optimal cell flipping to minimize channel density in vlsi design and pseudo-Boolean optimization. *Discrete Applied Mathematics* **90**, (1999), pp. 69–88.
- [56] Boros, E., P.L. Hammer, R. Sun and G. Tavares. *A max-flow approach to improved lower bounds for quadratic 0-1 minimization*. Research Report RRR 7-2006, RUTCOR, Rutgers University (March 2006).

- [57] Boros, E., P.L. Hammer, R. Sun and G. Tavares. A max-flow approach to improved lower bounds for Quadratic Unconstrained Binary Optimization (QUBO). *Discrete Optimization* **5**(2), (2008), pp. 501–529.
- [58] Boros, E., P.L. Hammer and X. Sun. *The DDT method for quadratic 0-1 optimization*. Research Report RRR 39-1989, RUTCOR, Rutgers University (1989).
- [59] Boros, E., P.L. Hammer and X. Sun. *Network flows and minimization of quadratic pseudo-Boolean functions*. Research Report RRR 17-1991, RUTCOR, Rutgers University (1991).
- [60] Boros, E., P.L. Hammer and X. Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics* **55**(1), (1994), pp. 1–13.
- [61] Boros, E., P.L. Hammer and G. Tavares. *Probabilistic one-pass heuristics for Quadratic Unconstrained Binary Optimization (QUBO)*. Technical Report 1-2006, RUTCOR, Rutgers University (2006).
- [62] Boros, E., P.L. Hammer and G. Tavares. Local search heuristics for Unconstrained Quadratic Binary Optimization (QUBO). *Journal of Heuristics* **13**(2), (2007), pp. 99–132.
- [63] Boros, E., I. Lari and B. Simeone. Block linear majorants in quadratic 0-1 optimization. *Discrete Applied Mathematics* **145**(1), (2004), pp. 52–71.
- [64] Boros, E. and A. Prékopa. Probabilistic bounds and algorithms for the maximum satisfiability problem. *Annals of Operations Research* **21**, (1989), pp. 109–126.
- [65] Bourjolly, J.M., P.L. Hammer, W.R. Pulleyblank and B. Simeone. *Combinatorial methods for bounding quadratic pseudo-Boolean functions*. Research Report CORR 89-21, Faculty of Mathematics, University of Waterloo (May 1989).
- [66] Bourjolly, J.M., P.L. Hammer, W.R. Pulleyblank and B. Simeone. *Combinatorial methods for bounding quadratic pseudo-Boolean functions*. Research Report RRR 27-1989, RUTCOR, Rutgers University (August 1989).
- [67] Bourjolly, J.M., P.L. Hammer, W.R. Pulleyblank and B. Simeone. *Boolean-combinatorial bounding of maximum 2-satisfiability*. Research Report Serie A no. 9, Department of Statistics, “La Sapienza” University (1992).
- [68] Bourjolly, J.M., P.L. Hammer, W.R. Pulleyblank and B. Simeone. Boolean-combinatorial bounding of maximum 2-satisfiability. In *Computer Science and Operations Research (Williamsburg, VA 1992)*, pp. 23–42 (Pergamon, Oxford, 1999).
- [69] Boykov, Y. and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 26 (2004).
- [70] Boykov, Y., O. Veksler and R. Zabih. Fast approximate energy minimization via graph cuts. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23 (2001).

- [71] Brockington, M. and J.C. Culberson. Camouflaging independent sets in quasi-random graphs. In *Second DIMACS challenge: Cliques, coloring and satisfiability* (Rutgers University, New Brunswick, NJ, 1993).
- [72] Buchheim, C. and G. Rinaldi. *Compact integer programming formulations for Boolean optimization problems*. Research report, Zentrum für Angewandte Informatik Köln, Lehrstuhl Jünger (September 2007).
- [73] Burer, S. and R.D.C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization **95**(2), (2003), pp. 329–357.
- [74] Burer, S., R.D.C. Monteiro and Y. Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization* **12**(2), (2001), pp. 503–521.
- [75] Burer, S., R.D.C. Monteiro and Y. Zhang. Maximum stable set formulations and heuristics based on continuous optimization. *Mathematical Programming, Series B* **94**, (2002), pp. 137–166.
- [76] Bushnell, M.L. and I.P. Shaik. Robust delay fault built-in self-testing method and apparatus. *United States Patent # 5,422,891* .
- [77] Butenko, S. and S. Trukhanov. Using critical sets to solve the maximum independent set problem. *Operations Research Letters* doi:10.1016/j.orl.2006.07.004.
- [78] Butz, L., P.L. Hammer and D. Haussmann. Reduction methods for the vertex packing problem. In *Proceedings of the 17th conference on probability theory* (M. Iosifescu, S. Grigorescu and T. Postelnicu, eds.). The Centre of Mathematical Statistics of the National Institute of Metrology Bucharest (VNU Science Press, Utrecht, 1985, Brasov, Romania, 1982).
- [79] Carraghan, M. and P.M. Pardalos. An algorithm for the maximum clique problem. *Operations Research Letters* **9**, (1990), pp. 375–382.
- [80] Carter, M.W. The indefinite zero-one quadratic problem. *Discrete Applied Mathematics* **7**, (1984), pp. 23–44.
- [81] Cartwright, D. and F. Harary. Structural balance: A generalization of Heider’s theory. *Physical Review* **63**, (1956), pp. 277–292.
- [82] Chardaire, P. and A. Sutter. A decomposition method for quadratic zero-one programming. *Management Science* **41**(4), (1995), pp. 704–712.
- [83] Chen, R.W., Y. Kajitani and S.P. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Transactions on Circuits and Systems* **30**(5), (1983), pp. 284–299.
- [84] Chen, S. and D. Li. Image binarization focusing on objects. *Neurocomputing* **69**(16-18), (2006), pp. 2411–2415.
- [85] Cheng, C.K., S.Z. Yao and T.C. Hu. The orientation of modules based on graph decomposition. *IEEE Transactions on Computers* **40**(6), (1991), pp. 774–780.

- [86] Chlebík, M. and J. Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics* **156**(3), (2008), pp. 292–312.
- [87] Chopra, S. and M.R. Rao. Facets of the  $k$ -partition problem. *Discrete Applied Mathematics* **61**, (1995), pp. 27–48.
- [88] Corràdi, K. and S. Szabò. A combinatorial approach for Keller’s conjecture. *Periodica Mathematica Hungarica* **21**(2), (1990), pp. 95–100.
- [89] Crama, Y. and P.L. Hammer. *Boolean functions: Theory, algorithms and applications* (Cambridge University Press, 2007). Forthcoming.
- [90] Crama, Y., P.L. Hammer, B. Jaumard and B. Simeone. Product form parametric representation of the solutions to a quadratic Boolean equation. *RAIRO Recherche Opérationnelle* **21**(4).
- [91] Crama, Y. and J.B. Mazzola. Valid inequalities and facets for a hypergraph model of the nonlinear knapsack and fms part-selection problems. *Annals of Operations Research* **58**, (1995), pp. 99–128.
- [92] de Givry, S., M. Zytnicki, F. Heras and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proc. of IJCAI-0* (2005), pp. 84–89.
- [93] Dimitropoulos, X., K.C. Claffy, M. Fomenkov, B. Huffaker, D. Krioukov, P. Mahadevan and A. Vahdat. *Lessons from three views of the internet topology*. Technical report, Cooperative Association for Internet Data Analysis (CAIDA) (2005).
- [94] Ebenegger, C., P.L. Hammer and D. de Werra. Pseudo-Boolean functions and stability of graphs. *Annals of Discrete Mathematics* **19**, (1984), pp. 83–98.
- [95] Elloumi, S., A. Faye and E. Soutif. Decomposition and linearization for 0-1 quadratic programming. *Annals of Operations Research* **99**, (1999), pp. 79–93.
- [96] Ferrante, A., G. Pandurangan and K. Park. *Complexity of combinatorial optimization in power-law graphs*. Technical report (2006).
- [97] Festa, P., P.M. Pardalos, M.G.C. Resende and C.C. Ribeiro. Randomized heuristics for the max-cut problem. *Optimization Methods and Software* **7**, (2002), pp. 1033–1058.
- [98] Fischer, I., G. Gruber, F. Rendl and R. Sotirov. Computational experience with a bundle approach for semidefinite cutting plane relaxations of max-cut and equipartition. *Mathematical Programming* **105**(2-3), (2006), pp. 451–469.
- [99] Flake, G.W., R.E. Tarjan and K. Tsoutsoulis. Graph clustering and minimum cut trees. *Internet Mathematics* **1**(4), (2004), pp. 385–408.
- [100] Fortet, R. L’algebre de Boole en recherche operationelle. *Revue Francaise de Recherche Operationelle* **4**, (1960), pp. 17–26.
- [101] Fraenkel, A.S. and P.L. Hammer. Pseudo-Boolean functions and their graphs. *Annals of Discrete Mathematics* **20**, (1984), pp. 137–146.

- [102] Fujisawa, K., M. Fukuda, M. Kojima and K. Nakata. Numerical evaluation of the SDPA (SemiDefinite Programming Algorithm), booktitle = High performance optimization, year = 2000, editor = H. Frenk and K. Roos and T. Terlaky and S. Zhang, publisher = Kluwer Academic Press, pages = 267-301.
- [103] Gallo, G., P.L. Hammer and B. Simeone. Quadratic knapsack problems. *Mathematical Programming* **12**, (1980), pp. 132–149.
- [104] Garey, M.R. and D.S. Johnson. *Computers and intractability: An introduction to the theory of s-completeness* (W. H. Freeman, San Francisco, 1979).
- [105] Garey, M.R., D.S. Johnson and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science* **1**, (1976), pp. 237–267.
- [106] Gendreau, M., P. Soriano and L. Salvail. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research* **41**, (1993), pp. 385–403.
- [107] Glover, F., B. Alidaee, C. Rego and G. Kochenberger. One-pass heuristics for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research* **137**(2), (2002), pp. 272–287.
- [108] Glover, F., G. Kochenberger and B. Alidaee. Adaptative memory tabu search for binary quadratic programs. *Management Science* **44**(3), (1998), pp. 336–345.
- [109] Glover, F., G.A. Kochenberger, B. Alidaee and M. Amini. Tabu search with critical event memory: An enhanced application for binary quadratic programs. In *Meta-heuristics - Advances and trends in local search paradigms for optimization* (S. Voss, S. Martello, I. Osman and C. Roucairol, eds.), pp. 83–109 (Kluwer Academic Publishers, 1998).
- [110] Glover, F. and R.E. Woolsey. Further reduction of zero-one polynomial programs to zero-one linear programming problems. *Operations Research* **21**, (1973), pp. 156–161.
- [111] Glover, F. and R.E. Woolsey. Note on converting of 0-1 polynomial programming problems to zero-one linear programming problems. *Operations Research* **22**, (1974), pp. 180–181.
- [112] Goemans, M.X. and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* **42**, (1995), pp. 1115–1145.
- [113] Gramm, J. and R. Niedermeier. Faster exact solutions for max2sat. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC 2000)* (G. Bongiovanni, G. Gambosi and R. Petreschi, eds.) (volume 1767 in Lecture Notes in Computer Science, 2000), pp. 174–186.
- [114] Grötschel, M., M. Jünger and G. Reinelt. Via minimization with pin preassignments and layer preference. *Zeitschrift für Angewandte Mathematik und Mechanik* **69**(11), (1989), pp. 393–399.

- [115] Gueye, S. and P. Michelon. “miniaturized” linearizations for quadratic 0/1 problems. *Annals of Operations Research* **95**(1), (2005), pp. 235–261.
- [116] Gulati, S.K., S.K. Gupta and A.K. Mittal. Unconstrained quadratic bivalent programming problem. *European Journal of Operational Research* **15**, (1980), pp. 121–125.
- [117] Hadlock, F. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal of Computing* **4**(3), (1975), p. 221225.
- [118] Hammer, P.L. Plant location - a pseudo-Boolean approach. *Israel Journal of Technology* **6**, (1968), pp. 330–332.
- [119] Hammer, P.L. Pseudo-Boolean remarks on balanced graphs. *International Series of Numerical Mathematics* **36**, (1977), pp. 69–78.
- [120] Hammer, P.L. *The conflict graph of a pseudo-Boolean function*. Research report, Bell Laboratories (August 1978).
- [121] Hammer, P.L. and P. Hansen. Logical relations in quadratic 0-1 programming. *Romanian Journal of Pure and Applied Mathematics* **26**(3), (1981), pp. 421–429.
- [122] Hammer, P.L., P. Hansen and B. Simeone. Upper planes of quadratic 0-1 functions and stability in graphs. In *Nonlinear Programming 4* (O. Mangasarian, R.R. Meyer and S.M. Robinson, eds.), pp. 395–414 (Academic Press, New York, 1981).
- [123] Hammer, P.L., P. Hansen and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming* **28**, (1984), pp. 121–155.
- [124] Hammer, P.L. and A. Hertz. *On a transformation which preserves the stability number*. Research Report RRR 69-1991, RUTCOR, Rutgers University (1991).
- [125] Hammer, P.L. and R. Holzman. Approximations of pseudo-Boolean functions; applications to game theory. *ZOR - Methods and Models of Operations Research* **36**, (1992), pp. 3–21.
- [126] Hammer, P.L., N.V.R. Mahadev and D. de Werra. The struction of a graph: Application to CN-free graphs. *Combinatorica* **5**, (1985), pp. 141–147.
- [127] Hammer, P.L. and I. Rosenberg. Linear decomposition of a positive group-Boolean function. In *Numerische Methoden bei Optimierung, Vol II* (L. Collatz and W. Wetterling, eds.), pp. 51–62 (Birkhauser-Verlag, Basel-Stuttgart, 1974).
- [128] Hammer, P.L., I. Rosenberg and S. Rudeanu. Application of discrete linear programming to the minimization of Boolean functions. *Rev. Mat. Pures Appl.* **8**, (1963), pp. 459–475.
- [129] Hammer, P.L., I. Rosenberg and S. Rudeanu. On the determination of the minima of pseudo-Boolean functions. *Stud. Cerc. Mat.* **14**, (1963), pp. 359–364. (in Romanian).

- [130] Hammer, P.L. and A.A. Rubin. Some remarks on quadratic programming with 0-1 variables. *Revue Fr. Infor. Rech. Oper.* **4**, (1970), pp. 67–79.
- [131] Hammer, P.L. and S. Rudeanu. *Boolean methods in operations research and related areas* (Springer-Verlag, Berlin, Heidelberg, New York, 1968).
- [132] Hammer, P.L. and E. Shliffer. Applications of pseudo-Boolean methods to economic problems. *Theory and Decision* **1**, (1971), pp. 296–308.
- [133] Hammer, P.L. and B. Simeone. Quadratic functions of binary variable. *Combinatorial Optimization, Lecture Notes in Mathematics* **1403**, (1989), pp. 1–56.
- [134] Hansen, P. and B. Jaumard. Uniquely solvable quadratic Boolean equations. *Discrete Applied Mathematics* **12**, (1985), pp. 145–154.
- [135] Hansen, P., B. Jaumard and C. Meyer. *A simple enumerative algorithm for unconstrained 0-1 quadratic programming*. Technical Report G-2000-59, Les Cahiers du GERAD (2000).
- [136] Hasselberg, J., P.M. Pardalos and G. Vairaktarakis. Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization* **3**(4), (1993), pp. 463–482.
- [137] Helmberg, C. Numerical evaluation of sbmethod. *Mathematical Programming* **95**(2), (2003), pp. 381–406.
- [138] Helmberg, C. and K.C. Kiwiel. A spectral bundle method with bounds. *Mathematical Programming* **93**, (2002), pp. 173–194.
- [139] Helmberg, C. and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming* **82**, (1998), pp. 291–315.
- [140] Helmberg, C. and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization* **10**(3), (2000), pp. 673–696.
- [141] Hertz, A. Polynomially solvable cases for the maximum stable set problem. *Discrete Applied Mathematics* **60**, (1995), pp. 195–210.
- [142] Hertz, A. On the use of Boolean methods for the computation of the stability number. *Discrete Applied Mathematics* **76**, (1997), pp. 183–203.
- [143] Hertz, A. On a transformation which preserves the stability number. *Yugoslav Journal of Operations Research* **10/1**, (2000), pp. 1–12.
- [144] Hillier, F.S. *The evaluation of risky interrelated investments* (North-Holland, Amsterdam, 1969).
- [145] Homer, S. and M. Peinado. Design and performance of parallel and distributed approximation algorithms for maxcut. *Journal of Parallel and Distributed Computing* **46**, (1997), pp. 48–61.

- [146] Huang, H.X., P.M. Pardalos and O.A. Prokopyev. Lower bound improvement and forcing rule for quadratic binary programming. *Computational Optimization and Applications* **33**(2-3), (2006), pp. 187–208.
- [147] Ibaraki, T., T. Imamichi, Y. Koga, K. Nonobe, H. Nagamochi and M. Yagiura. Efficient branch-and-bound algorithms for weighted max-2-sat. In *Proceedings of the third international Asian Applied Computing Conference (AACC)* (Nepal, 2005).
- [148] Ibaraki, T., T. Imamichi, Y. Koga, K. Nonobe, H. Nagamochi and M. Yagiura. *Efficient branch-and-bound algorithms for weighted MAX-2-SAT*. Tech. Rep. Technical Report 2007-011, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University (May 2007).
- [149] J. F. Banzhaf, I. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review* **19**, (1965), pp. 317–343.
- [150] Johnson, D.S., C.R. Aragon, L.A. McGeoch and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research* **39**, (1991), pp. 378–406.
- [151] Johnson, D.S. and M.A. Trick. Cliques, coloring, and satisfiability. In *2nd DIMACS Implementation Challenge in New Brunswick, NJ, October 11-13, 1993* (D.S. in Discrete Mathematics and Theoretical Computer Science (26), eds.) (American Mathematical Society, Providence, RI, 1996).
- [152] Jünger, M., A. Martin, G. Reinelt and R. Weismantel. Quadratic 0-1 optimization and a decomposition approach for the placement of electronic circuits. *Mathematical Programming* **63**, (1994), pp. 257–279.
- [153] Kalantari, B. and A. Bagchi. An algorithm for quadratic zero-one programs. *Naval Research Logistics* **37**, (1990), pp. 527–538.
- [154] Karger, D.R. and M.S. Levine. Finding maximum flows in undirected graphs seems easier than bipartite matching. *STOC 1998*, pp. 69–78.
- [155] Katayama, K. and H. Narihisa. Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. *European Journal of Operational Research* **134**, (2001), pp. 103–119.
- [156] Katzgraber, H.G. and A.P. Young. Monte Carlo studies of the one-dimensional ising spin glass with power-law interactions. *Physical Review B* **67**(13), (2003), pp. 134410–134417.
- [157] Kim, S.H., Y.H. Kim and B.R. Moon. A hybrid genetic algorithm for the max cut problem. In *Genetic and Evolutionary Computation Conference* (2001), pp. 416–423.
- [158] Kochenberger, G., B. Alidaee and M. Amini. *Heuristic algorithms for the unconstrained binary quadratic programming problem*. Working paper, University of Colorado (1998).

- [159] Kolmogorov, V., V. Lempitsky, C. Rother and M. Szummer. Optimizing binary MRFs via extended roof duality. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2007).
- [160] Kolmogorov, V. and R. Zabih. What energy functions can be minimized via graph cuts? In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2004).
- [161] Krarup, J. and P.M. Pruzan. Computer-aided layout design. *Mathematical Programming Study* **9**, (1978), pp. 75–94.
- [162] Kubiak, W. New results on the completion time variance minimization. *Discrete Applied Mathematics* **58**, (1995), pp. 157–168.
- [163] Kullmann, O. Worst-case analysis, 3-SAT decision and lower bounds: Approaches for improved SAT algorithms. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* (D. Du, J. Gu and P.M. Pardalos, eds.), pp. 261–313 (American Mathematical Society, 1997).
- [164] Lagarias, J.C. and P.W. Shor. Keller’s cube-tiling conjecture is false in high dimensions. *Bulletin of the AMS* **27**, (1992), pp. 279–283.
- [165] Larrosa, J. and T. Schiex. Solving weighted csp by maintaining arc-consistency. *Artificial Intelligence* **159**(1-2), (2004), pp. 1–26.
- [166] Laughhunn, D.J. Quadratic binary programming with applications to capital budgeting problems. *Operations Research* **18**, (1970), pp. 454–461.
- [167] Laughhunn, D.J. and D.E. Peterson. Computational experience with capital expenditure programming models under risk. *J. Business Finance* **3**, (1971), pp. 43–48.
- [168] Laundry, R., M. Perregaard, G. Tavares, H. Tipi and A. Vazacopoulos. *Solving hard mixed integer programming problems with Xpress-MP: A MIPLIB 2003 case study*. Research Report RRR 2-2007, RUTCOR, Rutgers University (2007).
- [169] Lee, H. and K. Park. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. In *Proceedings of ACM SIGCOMM* (2001), pp. 15–26.
- [170] LI, C.M., F. Manya and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research* **30**, (2007), pp. 321–359.
- [171] Li, X. and Z. Tian. Optimum cut-based clustering. *Signal Processing* **87**(11), (2007), pp. 2491–2502.
- [172] Liers, F. *Contributions to determining exact ground-states of Ising spin-glasses and to their physics*. Ph.D. thesis, Universität zu Köln (2004).
- [173] Lovász, L. On the ratio of optimal integral and fractional covers. *Discrete Mathematics* **13**, (1975), pp. 383–390.
- [174] MacWilliams, F.J. and N.J.A. Sloane. *The theory of error-correcting codes* (North-Holland, Amsterdam, 1979).

- [175] McBride, R.D. and J.S. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science* **26**, (1980), pp. 282–296.
- [176] Mehlhorn, K. and S. Näher. *The LEDA platform of combinatorial and geometric computing* (Cambridge University Press, Cambridge, England, 1999).
- [177] Merz, P. and B. Freisleben. Genetic algorithms for binary quadratic programming. In *Proceedings of the 1999 international Genetic and Evolutionary Computation Conference (GECCO'99)* (1999), pp. 417–424.
- [178] Merz, P. and B. Freisleben. Greedy and local search heuristics for the unconstrained binary quadratic programming problem. *Journal of Heuristics* **8**(2), (2002), pp. 197–213.
- [179] Mittelman, H.D. An independent benchmarking of SDP and SOCP solvers. *Mathematical Programming* .
- [180] Naclerio, N.J., S. Masuda and K. Nakajima. The via minimization problem is NP-complete. *IEEE Transactions on Computers* **38**(11), (1989), pp. 1604–1608.
- [181] Nelson, H.F. Beebe's bibliographies page (April 2006).  
URL <http://www.math.utah.edu/~beebe/bibliographies.html>.
- [182] Nemhauser, G.L. and L.E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming* **8**, (1975), pp. 232–248.
- [183] Palubeckis, G.
- [184] Palubeckis, G. Heuristics with a worst-case bound for unconstrained quadratic 0-1 programming. *Informatika* **3**(2), (1992), pp. 225–240.
- [185] Palubeckis, G. A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. *Computing* **54**, (1995), pp. 283–301.
- [186] Palubeckis, G. Steepest ascent: A generalization of the greedy algorithm for the maximum clique problem. *Information Technology and Control* **28**(3), (2003), pp. 7–13.
- [187] Palubeckis, G. Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research* **131**, (2004), pp. 259–282.
- [188] Palubeckis, G. and V. Krivickiene. Application of multistart tabu search to the max-cut problem. *Information Technology and Control* **31**(2), (2004), pp. 29–35.
- [189] Palubeckis, G. and A. Tomkevičius. Grasp implementations for the unconstrained binary quadratic optimization problem. *Information Technology and Control* **24**(3), (2002), pp. 14–20.
- [190] Papaioannou, S.G. Optimal test generation in combinational networks by pseudo-Boolean programming. *IEEE Transactions on Computers* **26**, (1977), pp. 553–560.

- [191] Pardalos, P.M. Construction of test problems in quadratic bivalent programming. *ACM Transactions on Mathematical Software* **17**(1), (1991), pp. 74–87.
- [192] Pardalos, P.M. and N. Desai. An algorithm for finding a maximum weighted independent set in an arbitrary graph. *International Journal of Computer Mathematics* **38**, (1991), pp. 163–175.
- [193] Pardalos, P.M., H.X. Huang and O.A. Prokopyev. *Multi-quadratic binary programming*. Research report, University of Florida (2004).
- [194] Pardalos, P.M. and S. Jha. Complexity of uniqueness and local search in quadratic 0-1 programming. *Operations Research Letters* **11**, (1992), pp. 119–123.
- [195] Pardalos, P.M. and G.P. Rodgers. Computational aspects of a branch and bound algorithm for quadratic 0-1 programming. *Computing* **45**, (1990), pp. 131–144.
- [196] Pardalos, P.M. and G.P. Rodgers. A branch and bound algorithm for the maximum clique problem. *Computers and Operations Research* **19**, (1992), pp. 363–375.
- [197] Pardalos, P.M. and J. Xue. The maximum clique problem. *Journal of Global Optimization* **4**, (1994), pp. 301–328.
- [198] Phillips, A.T. and J.B. Rosen. A quadratic assignment formulation for the molecular conformation problem. *Journal of Global Optimization* **4**, (1994), pp. 229–241.
- [199] Picard, J.C. and M. Queyranne. On the integer-valued variables in the linear vertex packing problem. *Mathematical Programming* **12**, (1977), pp. 97–101.
- [200] Picard, J.C. and H.D. Ratliff. Minimum cuts and related problems. *Networks* **5**, (1975), pp. 357–370.
- [201] Picard, J.C. and H.D. Ratliff. A cut approach to the rectilinear facility location problem. *Operations Research* **26**, (1978), pp. 422–433.
- [202] Pinter, R.Y. Optimal layer assignment for interconnect. *Advances in VLSI and Computer Systems* **1**(2), (1984), pp. 123–137.
- [203] Raj, A., G. Singh and R. Zabih. MRF’s for MRI’s: Bayesian reconstruction of MR images via graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR’06)* (2006).
- [204] Ranyard, R.H. An algorithm for maximum likelihood ranking and slater’s  $i$  from paired comparisons. *British Journal of Mathematical and Statistical Psychology* **29**, (1976), pp. 242–248.
- [205] Rao, M.R. Cluster analysis and mathematical programming. *Journal of the American Statistical Association* **66**, (1971), pp. 622–626.
- [206] Rendl, F., G. Rinaldi and A. Wiegele. *A branch and bound algorithm for max-cut based on combining semidefinite and polyhedral relaxations*. Tech. rep., Alpen-Adria-Universität Klagenfurt, Inst. f. Mathematik (2006).

- [207] Rendl, F., G. Rinaldi and A. Wiegele. Biq Mac solver - Binary Quadratic and Max Cut solver (November 2007).  
URL <http://biqmac.uni-klu.ac.at>.
- [208] Rendl, F., G. Rinaldi and A. Wiegele. A branch and bound algorithm for max-cut based on combining semidefinite and polyhedral relaxations. In *Integer Programming and Combinatorial Optimization*, vol. 4513 of *Lecture Notes in Computer Science* (Springer, Berlin, Germany, 2007).
- [209] Resende, M.G.C. MAX-CUT problem data (March 2008).  
URL <http://www.research.att.com/mgcr/data/maxcut.tar.gz>.
- [210] Rhys, J. A selection problem of shared fixed costs and network flows. *Management Science* **17**, (1970), pp. 200–207.
- [211] Rinaldi, G. Rudy: A generator for random graphs (1996).
- [212] Rosenberg, I.G. 0-1 optimization and non-linear programming. *Revue Française d'Automatique, d'Informatique et de Recherche Opérationnelle (Série Bleue)* **2**, (1972), pp. 95–97.
- [213] Rosenberg, I.G. Reduction of bivalent maximization to the quadratic case. *Cahiers du Centre d'Etudes de Recherche Operationnelle* **17**, (1975), pp. 71–74.
- [214] Sanchis, L. Test case construction for the vertex cover problem. In *DIMACS workshop on computational support for discrete mathematics* (DIMACS, New Brunswick, NJ, 1992).
- [215] Sanchis, L. and A. Jagota. *Some experimental and theoretical results on test case generators for the maximum clique problem*. Technical Report 93-69, DIMACS (1993).
- [216] Savelsbergh, M.W.P. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing* **6**, (1994), pp. 445–454.
- [217] Shaik, I.P. *An optimization approach to robust delay-fault built-in testing*. Ph.D. thesis, Electrical Engineering Department, Rutgers University (1996).
- [218] Shapley, L.S. A value for n-person games. In *Contributions to the theory of games, II* (H.W. Kuhn and A.W. Tucker, eds.), pp. 307–317 (Annals of Mathematics Studies No. 28, Princeton University Press, Princeton, NJ, 1953).
- [219] Shen, H. and H. Zhang. An empirical study of max-2-sat phase transitions. In *Proceedings of LICS workshop on phase transitions* (Ottawa, Canada, 2003).
- [220] Shen, H. and H. Zhang. Improving exact algorithms for max-2-sat. In *Eighth international symposium on artificial intelligence and mathematics* (Fort Lauderdale, Florida, 2004).
- [221] Shen, H. and H. Zhang. Improving exact algorithms for max-2-sat. *Annals of Mathematics and Artificial Intelligence* **44**(4), (2005), pp. 419–436.
- [222] Simeone, B. *Quadratic 0-1 programming, Boolean functions and graphs*. Ph.D. thesis, University of Waterloo (1979).

- [223] Simone, C.D., M. Diehl, M. Jünger, P. Mutzel, G. Reinelt and G. Rinaldi. Exact ground states of ising spin glasses: New experimental results with a branch and cut algorithm. *Journal of Statistical Physics* **80**, (1995), pp. 487–496.
- [224] Sloane, N.J.A. Unsolved problems in graph theory arising from the study of codes. *Graph Theory Notes of New York XVIII* pp. 11–20.
- [225] Strogatz, S.H. and D.J. Watts. Collective dynamics of 'small-world' networks. *Nature* p. 440.
- [226] Sun, X. *Combinatorial algorithms for Boolean and pseudo-Boolean functions*. Ph.D. thesis, RUTCOR, Rutgers University (1991).
- [227] Tarjan, R.E. Depth-first search and linear graph algorithms. *SIAM Journal of Computing* **1**, (1972), pp. 146–160.
- [228] Tavares, G. *A bibliography of Data Envelopment Analysis (DEA) (1978-2001)*. Research Report RRR 01-2002, RUTCOR, Rutgers University (January 2002).
- [229] Tavares, G. The pseudo-Boolean optimization website (March 2008). URL <http://rutcor.rutgers.edu/~pbo>.
- [230] Taxt, T. and Øivind Due Trier. Evaluation of binarization methods for document images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(3), (1995), pp. 312–315.
- [231] Wang, H., B. Alidaee and G. Kochenberger. *Evaluating a clique partitioning problem model for clustering data mining*. Tech. Rep. HCES-06-04, Hearin Center for Enterprise Science, University of Mississippi (2004).
- [232] Warszawski, A. Pseudo-Boolean solutions to multidimensional location problems. *Operations Research* **22**, (1974), pp. 1081–1085.
- [233] Watters, L.G. Reduction of integer polynomial problems to zero-one linear programming problems. *Operations Research* **15**, (1967), pp. 1171–1174.
- [234] Weingartner, H.M. Capital budgeting of interrelated projects: Survey and synthesis. *Management Science* **12**, (1966), pp. 485–516.
- [235] Williams, A.C. *Quadratic 0-1 programming using the roof dual with computational results*. Research Report RRR 8-1985, RUTCOR, Rutgers University (December 1985).
- [236] Xing, Z. and W. Zhang. Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence* **164**(1-2), (2005), pp. 47–80.
- [237] Xu, K. and W. Li. *Many hard examples in exact phase transitions with application to generating hard satisfiable instances*. CoRR Report cs.CC/0302001 (2003).
- [238] Yang, Y. and H. Yan. An adaptive logical method for binarization of degraded document images. *Pattern Recognition* **33**, (2000), pp. 787–807.

- [239] Ye, X., M. Cheriet and C.Y. Suen. Stroke-model-based character extraction from gray-level document images. *IEEE Transactions on Image Processing* **10**(8), (2001), pp. 1152–1161.
- [240] Zemel, E. Measuring the quality of approximate solutions to zero-one programming problems. *Mathematics of Operations Research* **6**(3), (1981), pp. 319–332.
- [241] Zhang, H., H. Shen and F. Manyà. Exact algorithms for max-sat. *Electronic Notes in Theoretical Computer Science* **86**(1).

## Vita

### Gabriel Tavares

- 1988-93** B.Sc. in Computer Engineering, Coimbra University, Portugal.
- 1995-98** M.Sc. in Information Systems and Technologies, Coimbra University, Portugal.
- 2000-08** Ph.D. in Operations Research, Rutgers University.
- 1993-99** Assistant Professor in the Department of Computer Engineering, Polytechnic Institute of Guarda, Portugal.
- 2000-03** Scholarship awarded by the Portuguese FCT and by the FSE in the context of the III Quadro Comunitário de Apoio.
- 2004-06** Graduate Assistant, RUTCOR, Rutgers University, New Brunswick, New Jersey.
- 2005-07** Senior Consultant, Dash Optimization, Englewood Cliffs, New Jersey.
- 2008** Lead Consultant, Fair Isaac Corporation, Englewood Cliffs, New Jersey.
- 2006** J. Figueira, G. Tavares and M. Wiecek. Labelling algorithms for multiple criteria knapsack problems, Internal Report 5/2006, Instituto Superior Técnico, Universidade Técnica de Lisboa.
- 2006** E. Boros, P. L. Hammer and G. Tavares. Preprocessing of unconstrained quadratic binary optimization, RRR 10-2006, RUTCOR - Rutgers Center for Operations Research, Rutgers University.
- 2007** E. Boros, P. L. Hammer and G. Tavares. Local search heuristics for Quadratic Unconstrained Binary Optimization (QUBO), *Journal of Heuristics* 13 (2), pp. 99–132.
- 2008** E. Boros, P. L. Hammer and G. Tavares, A max-flow approach to improved lower bounds for quadratic 0-1 minimization, *Discrete Optimization* 5 (2), In Memory of George B. Dantzig (1914-2005), pp. 501–529.
- 2008** R. Laundry, M. Perregaard, G. Tavares, H. Tipi, and A. Vazacopoulos. Solving hard integer programming problems with Xpress-MP: A MIPLIB 2003 case study. *INFORMS Journal on Computing*, accepted.