A GENERAL CLASS OF HEURISTICS FOR CONSTRAINED FOREST AND CYCLE PROBLEMS

BY CELINA Z. IMIELINSKA

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 Computer Science

Written under the direction of

Professor Bahman Kalantari

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 1993

ABSTRACT OF THE DISSERTATION

A General Class of Heuristics for Constrained Forest and Cycle Problems

by Celina Z. Imielinska , Ph.D.

Dissertation Director: Professor Bahman Kalantari

Let $G = (V, E)$ be a connected, undirected, edge-weighted graph. A subgraph of G satisfying a certain criteria will be called an optimal solution if its total edge-weight is minimum among all subgraphs satisfying the given criteria. The *minimum spanning tree (MST)* of G is its optimal spanning tree. The *constrained forest problem* (CFP), a more general version of the MST, is the problem of finding an optimal spanning forest in which each tree spans at least m, a given number of vertices. When $n = |V|$ is a multiple of m, a variation of the CFP is the *exact constrained forest problem (ECFP)*, where each tree spans exactly m vertices. We shall also refer to $ECFP$ as the *m-subtree problem.* For $m = 2$, the m-subtree problem reduces to the minimum weight perfect matching problem which is the problem of finding an optimal set of edges such that each vertex is incident to exactly one edge.

The *traveling salesman problem (TSP)* is the problem of finding an optimal tour (Hamiltonian cycle) covering all the vertices in V. A more general version of the TSP is the *constrained cycle problem (CCP)*, which is the problem of finding an optimal set of vertex-disjoint cycles each covering at least m vertices. For $m = 3$ the CCP reduces to the so-called *2-matching problem* and for $m = 4$ to the triangle-free *2-matching problem.* When n is a multiple of m, a variation of the CCP is the *exact constrained*

cycle problem (ECCP) where each cycle covers exactly m vertices. We shall also refer to the $ECCP$ as the *m-perfect matching problem.*

Except for the boundary cases of the above mentioned problems e.g. MST, the perfect matching problem, these problems are NP-hard. A crucial component of the thesis is a fast 2-approximate heuristic for the CFP which is proved to be NP-hard for $m \leq 3$. From this heuristic, under the assumption of the triangle inequality, we obtain a 4-approximate heuristic for the CCP and a general class of heuristics for the $ECFP$ and the $ECCP$. In addition to the heuristic for the CFP, this general class of heuristics is a powerful combination of the *Onethird* and the *hypergreedy* heuristics for perfect matching. This class of heuristics allows adjustable time complexity and is capable of producing approximate solutions within either a constant or a very slowly growing function of n times the optimal weight.

Acknowledgements

I would like to thank my advisor, Prof. Bahman Kalantari, for inspiring my thesis research and working with me for many years.

I would like to thank the other members of my committee: Prof. Diane Souvaine, Prof. Bill Steiger and Dr. Subhash Suri, for very meaningful comments which helped me to improve the presentation of my thesis.

I would like to thank Prof. Ann Yasuhara for very important discussions.

This thesis would never be done without the support of Tomasz, my husband and my best friend, who always believed in me and inspired me whenever I thought about quitting. Thanks to him I am here reaching an important goal.

Dedication

to Tomasz , Marcin and Konrad

Table of Contents

List of Figures

Chapter 1

Introduction.

There are many combinatorial optimization problems for which either exact solutions are hard to find, e.g. for NP-complete problems, or an implementation of exact polynomial-time solutions is too slow in practice. We will focus in this thesis on general approximation techniques for particular classes of graph covering problems, for which we will find fast and easy to implement heuristics that produce near optimal solutions within guaranteed error bound. By the *error* of a heuristic algorithm we mean the worst case ratio, of the cost of an approximate solution produced by the heuristic to the cost of the optimal solution. A heuristic algorithm is said to have *performance guarantee* of α if it produces a solution within an error of α. If additionally the heuristic runs in a polynomial time, we call it an α-approximate algorithm. In general we denote the error of a heuristic by $f(n)$, a function of the number of vertices n in a given graph.

In this thesis we will deal with families of combinatorial optimization problems on undirected edge-weighted graphs. Given an edge-weighted graph $K(V)$, $|V| = n$ and a natural number m, a subgraph of $K(V)$ satisfying certain criteria will be called *an optimal solution* it its total edge-weight is minimum among all subgraphs satisfying the given criteria. We consider a class of graph covering problems with an optimal set of vertex-disjoint connected components which are either trees of cycles satisfying certain requirements on their size, i.e. the number of vertices they will cover. If the size of each tree and cycle is *at least* m, we call the problems *constrained forest problem, (CFP)* and the *constrained cycle problem, (CCP)*, respectively. For $m = 3$, the CCP becomes the so-called 2-matching problem, where a graph is covered with an optimal set of cycles, each spanning at least 3 vertices, and for $m = 4$, it reduces to the triangle-free 2-matching problem. For $m = n$, the CFP becomes the minimum spanning tree

problem.

If the number of vertices in each tree and cycle is exactly m and in addition we assume that m divides n, we consider another class of the *exact constrained forest (ECFP)* and the *exact constrained cycle (ECCP)* problems, to which we will refer throughout the thesis as the *optimal m-subtree problem* and the *optimal m-perfect matching*, respectively. For $m = 2$ the optimal m-subtree problem becomes the minimum-weight perfect matching, which is a minimum-weight cover of a graph, where each vertex is incident to exactly one edge; and for $m = n$, it reduces to the minimum spanning tree problem. For $m = n$, the optimal m-perfect matching becomes the traveling salesman problem, which is a minimum-weight cycle covering all the vertices in a graph. For all these problems our proposed heuristics are capable of producing fast approximate solutions, whose worst-case error factors are bounded either by constants or very slowly growing functions of n. In some cases we assume that the edge-weights satisfy the triangle inequality. Except for the boundary cases of the above mentioned problems e.g. MST and the minimum-weight perfect matching, these problems are NP-hard.

The minimum-weight perfect matching problem can be solved exactly in polynomial time by the original Edmonds' algorithm [17], which was implemented by Gabow [21] in $O(n^3)$ time for dense graphs. The fastest polynomial time implementation of Edmonds' algorithm is due to Gabow [23], and its running time is $O(n(m + n\log n))$, where m is the number of edges in the graph. Since the algorithm is too slow in practice, for large n, there has been a need for faster and simpler approximate algorithms for the problem. There are several heuristics for the minimum-weight perfect matching that run faster that the exact algorithm, like Reingold and Tarjan's *greedy* heuristic [47], Papadimitriou's *minimum spanning tree* heuristic, in [47], Supowit, Plaisted and Reingold's *hypergreedy* heuristic [45], [50], Grigoriadis and Kalantari's *Onethird* class of heuristics [29], and Gabow, Goemans and Williamson's algorithm [24], [27].

An approximate solution to the traveling salesman problem (TSP), which is NP-complete, can be obtained by using either the well known 2-approximate MST-heuristic in $O(n^2)$ time [44] or Christofides' $(\frac{3}{2})$-approximate heuristic which runs in $O(n^3)$ time.

The above heuristics for the minimum-weight matching and the traveling salesman

problem are designed for single problems, rather than for a class of problems which can be solved using the same methodology. We propose in the thesis, general classes of heuristic algorithms which produce solutions for a spectrum of constrained forest and cycle problems, which includes the minimum-weight perfect matching, 2-matching problem, traveling salesman problem and minimum spanning tree as special cases. In the general classes of heuristics, we exploit the techniques used in the two existing heuristics for the minimum-weight perfect matching, (specifically the *hypergreedy* heuristic [45], [50], and the *Onethird* class of heuristics [29]), and a heuristic which we designed for the CFP. This combination of the three heuristics, in which the heuristic for the CFP plays the crucial role, results in obtaining a methodology for finding approximate solutions for the CFP, CCP, optimal m-perfect matching and optimal m-subtree problem.

For the CFP, which we show is NP-hard for $m \geq 3$, we develop a 2-approximate heuristic, which runs within the time needed to compute a minimum spanning tree. This fast and very simple heuristic is extensively used in obtaining approximate solutions for the other families of problems. Even though the heuristic is as simple as the trivial MST heuristic for the traveling salesman problem, proving the error bound of 2 required a sophisticated proof. This heuristic gives rise, in linear time, to a 4-approximate solution to the CCP. Thus, for $m = 3$, this becomes a fast 4-approximate algorithm for the 2-matching problem, and for $m = 4$ a fast 4-approximate heuristic for the triangle-free 2-matching problem.

We propose a class of heuristics for the optimal m-subtree problem and the optimal m-perfect matching problem called the (t, k)-heuristic, where t and k are integer parameters ranging from 0 to $\log n$. The (t, k)-heuristic is a generalization of the *Onethird* class of heuristics for perfect matching by Grigoriadis and Kalantari [29], and in addition uses a combination of the heuristic for the CFP and the *hypergreedy* by Plaisted, Reingold and Supowit [45], [50]. We consider a subclass of the (t, k)-heuristic, for $k = 1$ and $t = \lceil \log(\frac{n}{m}) \rceil$ (for $m = 2$, $t = \lfloor \log_3 n \rfloor$), called the *Generalized Hypergreedy*, which is a combination of the heuristic for the CFP and the *hypergreedy* heuristic. Because of its relative simplicity and its potential use in other heuristics, the Generalized Hypergreedy will be treated, throughout the thesis, as a separate class of heuristics. Therefore

we will analyze the error and the time complexity of the Generalized Hypergreedy separately. This will also allow to understand better the more complex analysis of the (t,k)-heuristic.

The error and the time complexity of the Generalized Hypergreedy for the m-subtree problem and the m-perfect matching are shown in the following table:

Problem	Error	Time Complexity
m-subtree problem $m = 2$	$2\lfloor \log_3 n \rfloor$	$O(n^2 \log n)$
m-subtree problem $3 \leq m \leq \frac{n}{m}$	$4\frac{m-1}{m}[\lceil \log(\frac{n}{m}) \rceil]$	$O(n^2 \log(\frac{n}{m}))$
m-perfect matching $m = 2$	$2\lfloor \log_3 n \rfloor$	$O(n^2 \log n)$
m-perfect $3 \leq m \leq \frac{n}{m}$	$4\frac{m-1}{m}(2\frac{m-1}{m} + \lceil \log(\frac{n}{m}) \rceil - 1)$	$O(n^2 \log(\frac{n}{m}))$

For $m = 2$, the Generalized Hypergreedy, for the m-subtree problem, reduces to the hypergreedy heuristic for perfect matching, and runs in $O(n^2 \log n)$ time. For large m, ($m = \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \ldots$), the Generalized Hypergreedy is a constant-error heuristic, which runs in $O(n^2)$ time, while for small m, it becomes a $(\log n)$-error heuristic running in $O(n^2 \log n)$ time. For $m = n$ the Generalized Hypergreedy reduces to the exact minimum spanning tree algorithm and the minimum spanning tree heuristic for the traveling traveling salesman problem for the m-subtree problem and the m-perfect matching, respectively.

The (t,k)-heuristic, for given t and k, consists of $(k+1)$ stages, and we obtain at the first k stages an approximate solution, which possibly covers all the vertices in V. If after the first k stages, there are n_k unmatched vertices, the heuristic uses, at the last $(k+1)$ stage, an algorithm $\mathcal{A}$, which can be either another heuristic or an exact algorithm for the problem. We mean by $f_{\mathcal{A}}(n_k)$ the error of the algorithm $\mathcal{A}$, applied

to a complete graph with n_k vertices. The error and the time complexity of the (t,k)-heuristic for the m-subtree problem and the m-perfect matching are described in the following table:

Problem	Error	Time Complexity
m-subtree problem	$[\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_1)^k - \frac{4}{9}\frac{m}{m-1}$ where $a_1 = 2\frac{m-1}{m}[1 + 4t(\frac{m-1}{m})]$	$O(tn^2)$
m-perfect matching	$[\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_1)^k - \frac{4}{9}\frac{m}{m-1}$ where $a_1 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(\frac{m-2}{m} + t)]$	$O(tn^2)$

For $m = 2$, $t = \lfloor \log_3 n \rfloor$, and $k = 1$, the (t,k)-heuristic for the m-subtree problem, reduces to the hypergreedy. For $m = 2$ and $t = 1$, the (t,k)-heuristic for m-subtree problem, becomes the Onethird class of heuristics. In Fig.1.1 we illustrate the relationships between the (t,k)-heuristic, the Generalized Hypergreedy, hypergreedy and the Onethird.

We show that all our heuristics run considerably faster for Euclidean points in the plane, while their error bounds increase only slightly, in some of the cases. The heuristics for the CFP and CCP use the algorithm for the Euclidean minimum spanning tree, and run in $O(n \log n)$ time. The (t,k)-heuristic and the Generalized Hypergreedy make use of approximation of the complete graph $K(V)$ by the Delaunay triangulation of V [46]. The Delaunay triangulation is a planar graph, which approximates complete Euclidean graphs [9], [37], within a small constant error bound. The best bound on this approximation has been shown by Keil and Gutwin [37] to be:

$$\alpha = 2.42. \tag{1.1}$$

Thus, by using the Delaunay triangulation of V, instead of $K(V)$, the size of the input problem reduces from $O(n^2)$ to $O(n)$.

For Euclidean points in the plane, the error and time complexity of the Generalized Hypergreedy are described below:

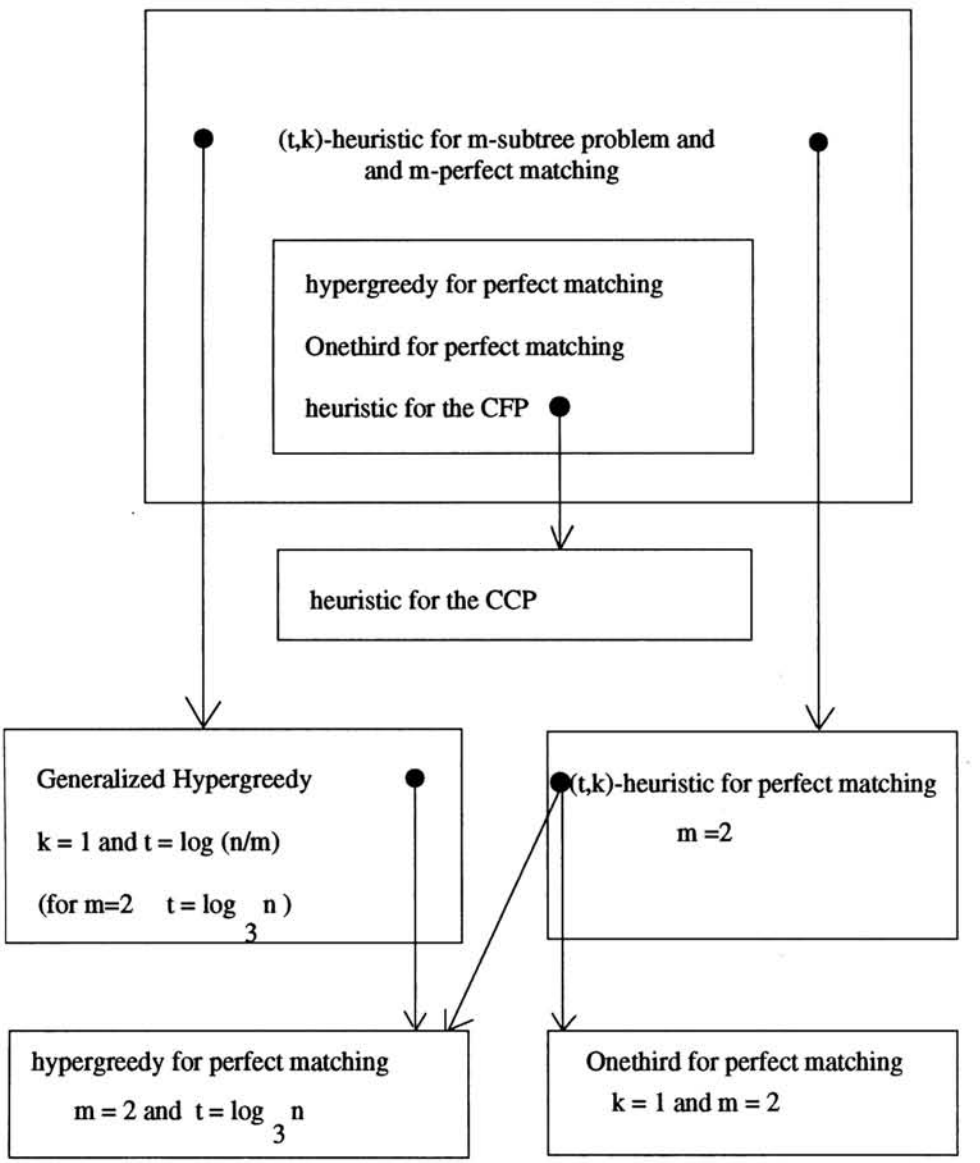


Figure 1.1: Outline of the results of the thesis

Problem	Error	Time Complexity
m-subtree problem $m = 2$	$2\alpha\lfloor \log_3 n \rfloor$	$O(n \log^2 n)$
m-subtree problem $3 \leq m \leq \frac{n}{m}$	$4\frac{m-1}{m}[1 + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]$	$O(n \log n \log(\frac{n}{m}))$
m-perfect matching $m = 2$	$2\alpha\lfloor \log_3 n \rfloor$	$O(n \log^2 n)$
m-perfect matching $3 \leq m \leq \frac{n}{m}$	$4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]$	$O(n \log n \log(\frac{n}{m}))$

For large m, ($m = \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \ldots$), the Generalized Hypergreedy for Euclidean points in the plane produces constant-error heuristics in $O(n \log n)$ time. For small m, it becomes a $(\log n)$-error heuristic running in $O(n \log^2 n)$ time. In particular, for $m = 2$, the Generalized Hypergreedy for the m-subtree problem becomes the hypergreedy heuristic for perfect matching for points in the Euclidean plane, which runs in $O(n \log^2 n)$ time, with the error bounded above by $2.42(2\lfloor \log_3 n \rfloor + 1$. This heuristic is faster than Vaidya's $O(n \log^3 n)$-time and $(3 \log_3 \frac{3}{2} n)$-error heuristic for Euclidean points in the plane [53].

For Euclidean points in the plane, the error and the time complexity of the (t, k)-heuristic for the m-subtree problem and the m-perfect matching

Problem	Error	Time Complexity
m-subtree problem	$[\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_2)^k - \frac{4}{9}\frac{m}{m-1}$ where $a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(1 + \alpha(t-1))]$	$O(tn \log n)$
m-perfect matching	$[\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_2)^k - \frac{4}{9}\frac{m}{m-1}$ where $a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(2\frac{m-1}{m} + \alpha(t-1))]$	$O(tn \log n)$

We introduce two exact dynamic programming algorithms for optimal m-perfect matching and the optimal m-subtree problem, which run in $O(m^2 2^{n+m} + 2^{2n-m})$ and $O(2^{2n-m})$ time, for $m < n$, respectively. For $m = n$, the dynamic programming algorithm for the m-perfect matching reduces to the $O(n^2 2^n)$-time dynamic programming

algorithm for the traveling salesman problem, and the dynamic programming algorithm for the m-subtree problem becomes the $O(n^2)$-time algorithm for minimum spanning tree. These heuristics can be used in the last stage of the (t,k)-heuristic, when the size of the problem becomes small enough.

We develop a heuristic for the minimum-weight perfect matching, called the *t-hypergreedy*, which is a combination of the hypergreedy heuristic with an exact minimum-weight perfect matching algorithm. In the following table we present the error and time complexity of the t-hypergreedy for weights satisfying the triangle inequality:

Heuristic	Error	Time Complexity
t-hypergreedy	$2t+1$	$O(max\{tn^2, \frac{n^3}{3^t}\})$

For Euclidean points in the plane, the error and the time complexity of the t-hypergreedy are following:

Heuristic	Error	Time Complexity
t-hypergreedy	$\alpha(2t+1)$	$O(max\{tn\log n, \frac{n^2\log n}{3^t}, \frac{n^3}{3^{3t}}\})$

Goemans and Williamson [27] proposed a general approximation technique for graph covering problems with a constant error bound and time complexity of $O(n^2 \log n)$, which was recently improved by Gabow, Goemans and Williamson to $O(n^2\sqrt{\log\log n})$ [24]. From now on, the Gabow, Goemans and Williamson's algorithm will be referred to as the *GGW* algorithm. Our 2-approximate heuristic for the *CFP* has the same error bound as the *GGW* algorithm for the problem, and its $O(n^2)$ and $O(n\log n)$ running time, for general weights and for Euclidean points in the plane, respectively, is faster than that of the 2-approximate *GGW* algorithm. Thus our algorithm is superior to the corresponding *GGW* algorithm. Also, the 4-approximate and heuristic for the *CCP*, which runs in the same time as the heuristic for the *CFP*, is faster than the corresponding 2-approximate *GGW* algorithm.

We use the *GGW* algorithm in the last stage of the (t,k)-heuristic. In the following

table we compare other heuristics for the perfect matching with instances of the (t, k)-heuristic combined with the *GGW*:

Heuristic	Error	Time Complexity
greedy (Reingold and Tarjan)	$\frac{4}{3}n^{0.585}$	$O(n^2 \log n)$
MST heuristic Papadimitriou	$\frac{n}{2}$	$O(n^2)$
hypergreedy (Plaisted, Supowit and Reingold)	$2\log_3 \frac{3}{2}n$	$O(n^2 \log n$
GGW, (Gabow, Goemans and Williamson)	2	$O(n^2\sqrt{\log\log n})$
Onethird with *GGW* (Grigoriadis and Kalantari) $t = 1$ and $k = \frac{1}{4}\log_3\log_3\log_3 n$	$3(\log_3\log_3 n)^{0.25} - 1$	$O(n^2)$
(t, k)-heuristic with *GGW* $t = 4$ and $k = \frac{1}{16}\log_3\log_3\log_3 n$	$3(\log_3\log_3 n)^{0.125} - 1$	$O(n^2)$
(t, k)-heuristic with *GGW* $k = 1$ and $t = \frac{1}{4}\log_3\log_3\log_3 n$	$\frac{3}{2}(\log_3\log_3\log_3 n) + 2$	$O(n^2 \log\log\log n)$

The (t, k)-heuristic with the *GGW* produces similar results for other values of m. For example, for $m = 4$:

Heuristic	Error	Time Complexity
GGW	4	$O(n^2\sqrt{\log\log n})$
(t, k)-heuristic with *GGW* $t = 4$ and $k = \frac{1}{16}\log_3\log_3\log_3 n$	$2.66(\log\log n)^{0.27} - .59$	$O(n^2 \log\log\log n)$

Finally we present an example of a parallel implementation of a perfect matching heuristic. We show that we can implement the basic Onethird heuristic in $O(\log^3 n)$

time using $O(n)$ processors, for the Euclidean points in the plane; and in $O(\log^2 n)$-time and using $O(n^2)$ processors, for general weights satisfying the triangle inequality.

The organization of the thesis is following. The next chapter contains formal definitions of the optimal perfect matching, CFP, CCP, optimal m-subtree and the optimal m-perfect matching problems; and include more detailed description of the results. In Chapter 3, an analysis of the heuristic for the CFP and CCP is presented. It is also shown in the chapter that the CFP is NP-hard for $m \geq 3$. The construction of the t-basic graph is described in Chapter 4. This chapter also contains the analysis of the error and time complexity. The implementation of t-basic graph, for the Euclidean case, is presented in Chapter 5. The Generalized Hypergreedy heuristic for the m-subtree problem and the m-perfect matching is described and analyzed in Chapter 6. The following Chapter 7 contains the analysis of the (t,k)-heuristic for the m-subtree problem and the m-perfect matching. In Chapter 8, two dynamic programming algorithms for the m-subtree problem and the m-perfect matching, respectively, are presented. Chapter 9 contains the description and analysis of the t-hypergreedy, a heuristic for perfect matching. In Chapter 10 a parallel implementation of the simplest Onethird heuristic is presented. The last chapter contains the concluding remarks.

1.1 Applications.

We describe here examples of application problems which can be formulated as one of the following problems: the minimum-weight perfect matching, the CFP, the m-subtree problem and the m-perfect matching. We will indicate which of the above optimization problems represent an exact solution, and which of the heuristics presented in this thesis could be used to obtain an approximate solution.

Efficient Use of Mechanical Plotter

Consider the problem of plotting a graph $G = (V, E)$, which represents a street map of a city (or VLSI circuits). Each vertex is given by its (x, y) coordinates. Since, in general, such a graph does not contain an Euler tour, i.e. a cycle which would visit each edge in the graph exactly once, it cannot be drawn without wasted pen movements.

Thus the pen has to be moved in the up position to a new location, and the distance corresponding to the motion is wasted. A graph is Eulerian if the degrees of all its vertices are even. We can make G Eulerian by matching the vertices with odd degrees, such that each odd-degree vertex will be incident to exactly one edge in the matching. Now we can extract an Euler tour in the graph. We will draw the graph by following the tour and lifting the pen every time an edge from the matching will be visited. The total cost of the wasted pen movement corresponds to the total weight of the matching of the odd-degree vertices in G. If the vertices are matched using arbitrary matching, the wasted pen movement can be substantial. If the matching is a minimum-weight matching of the odd-degree vertices, the waste would be minimized. Thus we can define the wasted pen movements as the *minimum-weight perfect matching.* Since the size of the graph in real life can be very large, in *tens of thousands* of vertices, we would rather use an approximate perfect matching algorithm instead of an exact one.

Vehicle Routing Problem

There is a collection of drivers to deliver to a set of n customers, where each driver has to serve the same number of m customers, where m divides n. To make a decision about which subset of m customers will be assigned to each driver, we want to represent the collection of single vehicle problems as the minimum-distance collection of closed tours, each covering exactly m customers. The exact solution to this problem problem is the *m-perfect matching*, and we can solve it approximately using the Generalized Hypergreedy or the (t, k)-heuristic. The size of the problem, i.e. the number of customers, could be in hundreds.

Clustering Problem

Given a black-and-white image, we wish to learn about its shape, by finding first clusters of black pixels, where each cluster contains at least a given minimum number of pixels. We can formulate the problem as the CFP and use our heuristic to obtain a 2-approximate solution. Given the clusters, we can further extract the shape of the image. The size of the problem corresponding to a number of pixels in an image, could be in range of 2^{18}.

Design of Communication Networks

We want to design a collection of local area networks, which would satisfy the *scalability* property, that all the networks will serve a balanced load, i.e. it will serve exactly the same number of m workstations. We will assume the topology of each such local area network to be either a tree or a cycle. Given a collection of n sites (m divides n), we wish to cover the sites with $\frac{n}{m}$ trees (cycles) of size exactly m, such that the total cost of the links between the sites would be minimum. Clearly this problem can be formulated as either the *m-subtree problem* or the *m-perfect matching*. We anticipate that the local area networks can be interconnected, but the cost of these relatively unused links connecting the servers in the networks, is irrelevant.

Chapter 2

An Extended Description of the Problems and the Results.

In Section 2.1, we will present the problem of finding the minimum-weight perfect matching in a weighted complete undirected graph. Edmonds' algorithm [17] for the problem is an example of a successful search for a polynomial-time algorithm for an integer programming problem. Because of the application of the minimum-weight perfect matching in areas, where large-scale problems have to be solved, a number of fast and near-optimal perfect matching heuristics have been proposed during last ten years. We will describe some of them, especially those which were used in designing of our heuristics for the problem. The results obtained by our heuristics will be compared to those obtained by the other heuristics.

There has been little done in the area of developing heuristics for the general class of graph covering problems, considered in this thesis. Until the recent result, the *GGW* algorithm by Gabow, Goemans and Williamson [27], [24], there had been no known heuristic for a class of weighted graph covering problem. In the subsequent sections, the CFP, CCP, m-perfect matching and the m-subtree problem will be defined formally. The results obtained by our heuristics for the problems would be outlined and compared to the corresponding results obtained by the *GGW* algorithm.

2.1 Minimum-Weight Perfect Matching.

Let V be a set of vertices, where $n = |V|$ is an even number, and let $K(V)$ be a complete edge weighted graph on V satisfying the triangle inequality, see Fig.2.1. A *perfect matching* of V is a set of edges such that each vertex of V is incident to exactly one edge. An *optimal perfect matching* of V is a perfect matching with minimum total

edge weight, see Fig.2.2.

The optimal perfect matching can be obtained by the Edmonds' algorithm [17], [21], in $O(n^3)$ time, using the well-known augmentation method [44]. For many years the heuristics for solving the perfect matching had not received much attention, because it was one of the rare combinatorial problems, for which existed a polynomial-time algorithm. But in the last decade new developments have occurred in relevant application areas (e.g. pen-plotting, VLSI), where for large-size problems even an $O(n^3)$ implementation of Edmonds' algorithm is simply too expensive. This has motivated search for faster and simpler approximate algorithm for the problem. There are several heuristics for the optimal perfect matching problem which run faster than the exact algorithm, like Reingold and Tarjan's $\frac{4}{3}n^{0.585}$-approximate and $O(n \log n)$-time *greedy* heuristic [47], that repeatedly matches the two closest unmatched vertices. The bound on the error is shown to be tight. Another algorithm the Papadimitriou's *minimum spanning tree (MST)* $\frac{n}{2}$-approximate and $O(n^2)$-time heuristic, described in [50], starts with spanning tree on the vertices, and converts it into a matching by repeatedly matching the leaves in the tree. The $\frac{n}{2}$ error bound is asymptotically achievable. Both algorithms apply to general non-negative weights.

Plaisted, Reingold and Supowit [50] and Plaisted and [45] have proposed a $2\log_3(\frac{3}{2}n)$-approximate *hypergreedy* heuristic, which runs in $O(n^2 \log n)$ time for edge weighted graphs satisfying the triangle inequality. For the same class of weighted graphs, Grigoriadis and Kalantari's have obtained a generic class of heuristics *Onethird* [29], which for each nonnegative integer $k \leq \log_3 n$, and for any perfect matching algorithm that runs in $t_A(n)$ time and has an error bound of $f_A(n)$ time the optimal weight, produces a $[(\frac{7}{3})^k(1 + f_A(3^{-k}n)) - 1]$-approximate heuristic in $O(\max\{n^2, t_A(3^{-k}n)\})$ time. The hypergreedy and the Onethird heuristics, will be outlined later. The recent Gabow, Goemans and Williamson's (*GGW*) algorithm [27], which represents a general approximation technique for a class of graph covering problems, finds a perfect matching in $O(n^2\sqrt{\log \log n})$ time and within error bound of 2. Grigoriadis and Kalantari showed in [28] that any perfect matching heuristic that produces a solution within a finite error bound runs in $\Omega(n \log n)$ time.

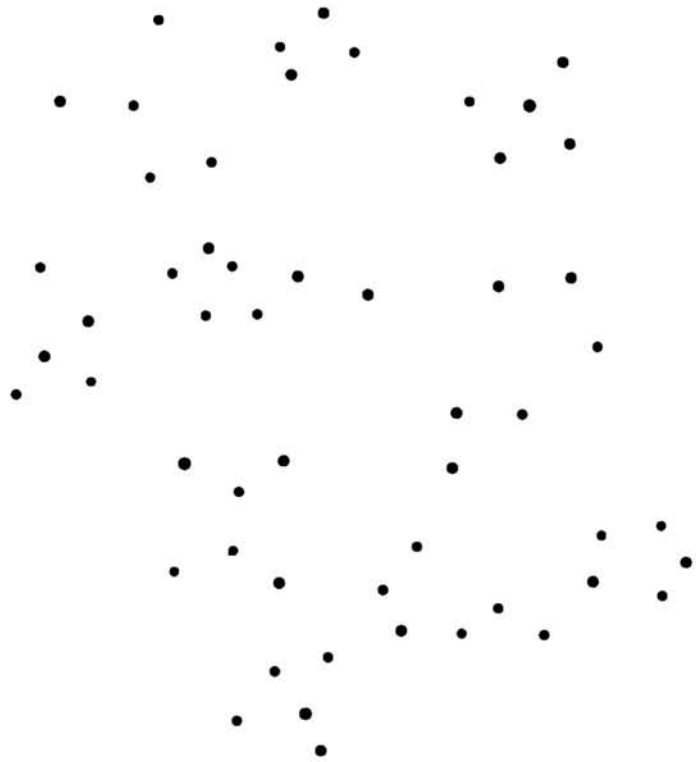


Figure 2.1: $K(V)$ - a complete edge weighted graph with $V = n$ vertices.

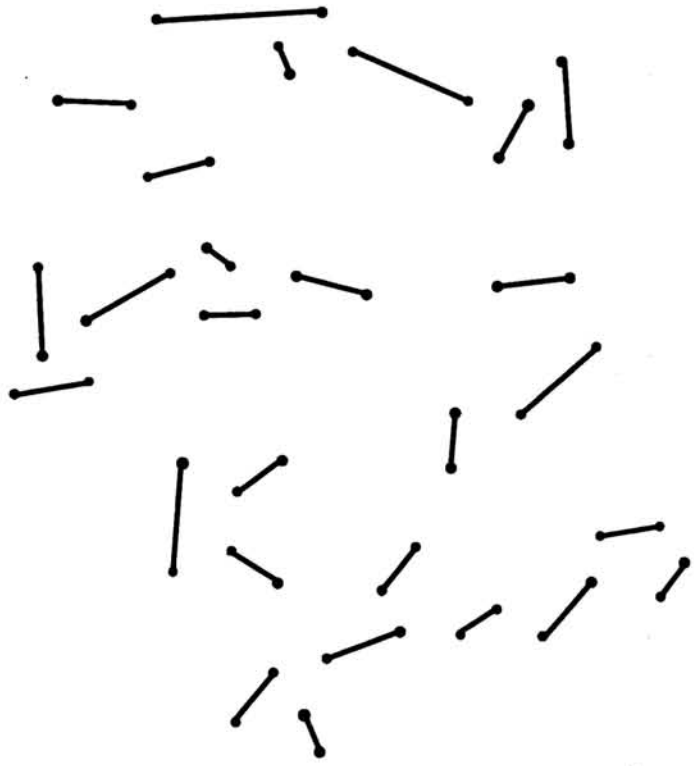


Figure 2.2: Optimal perfect matching of $K(V)$.

There are other exact algorithms which solve the minimum weight perfect matching for weights satisfying certain conditions. The fastest algorithm for the case of Euclidean points in the plane, due to Vaidya [51], runs in $O(n^{2.5}\log^4 n)$ time. Special cases of the problem can be solved faster. For example, Marcotte and Suri [41] proposed an $O(n \log n)$ time exact algorithm for the case where the points are the vertices of a convex polygon.

We propose two classes of heuristics for optimal perfect matching, one called the (t,k)-heuristic, where t and k are integer parameters satisfying $0 \leq t, k \leq \lfloor \log_3 n \rfloor$, and the other called the *t-hypergreedy*, where $0 \leq t \leq \lfloor \log_3 n \rfloor$. The (t,k)-heuristic for perfect matching is a generalization of the *hypergreedy* heuristic and the *Onethird* class of heuristics, which will outlined below. The hypergreedy algorithm, which is presented here, is our version of the original hypergreedy algorithm:

The hypergreedy

Input: $K(V)$, whose weights satisfy the triangle inequality

Output: a perfect matching of $K(V)$

1. Construct a sparse graph of $K(V)$ in $\lfloor \log_3 n \rfloor$ steps whose total weight is bounded above by $\lfloor \log_3 n \rfloor$ times the weight of the optimal solution

2. In every connected component (which covers an even number of vertices):

- duplicate the edges

- extract an Euler tour (a cycle visiting each edge exactly once)

- convert the tour into a Hamiltonian cycle (a cycle visiting each vertex exactly once)

2. Find the maximum cardinality minimum-weight matching

The sparse subgraph constructed in the Step 1. of the hypergreedy is obtained in $\lfloor \log_3 n \rfloor$ stages, by repeatedly finding a nearest neighbor graph of connected components which span an odd number vertices. Since we will extend this construction to obtain the t-basic graph, which is used extensively in our heuristics, its detailed description is presented in Chapter 5. The hypergreedy is a $2\lfloor \log_3 n \rfloor$-approximate heuristic running

in $O(n^2 \log n)$ time. The following is the outline of the Onethird class of heuristics, which consists of k stages, $0 \leq k \leq \lfloor \log_3 n \rfloor$, at at the last $(k+1)$ stage it uses an auxiliary minimum-weight perfect matching algorithm $\mathcal{A}$:

The Onethird (for weights satisfying the triangle inequality)

Input: $K(V)$, $V_0 = V$

Output: a perfect matching of $K(V)$

At each stage j, $j = 0, ..., k-1$

1. For each vertex in V_j find its nearest neighbor

2. In every connected component of the nearest neighbor graph:

- duplicate the edges
- extract an Euler tour
- convert the tour into a Hamiltonian cycle

3. From the union of the Hamiltonians select a partial matching with exactly $\lfloor \frac{1}{3} \rfloor$ edges

4. All unmatched vertices form set V_{j+1} and $K(V_{j+1})$.

After k stages: If $k < \log_3 n$, match the remaining unmatched vertices by an auxiliary perfect matching algorithm $\mathcal{A}$.

For each nonnegative $k \leq \lfloor \log_3 n \rfloor$ and any existing minimum-weight perfect matching algorithm $\mathcal{A}$ that runs in $t_{\mathcal{A}}(n)$ time and has an error bound of $f_{\mathcal{A}}(n)$, the Onethird produces a heuristic that runs in

$$T(n) = O(\max\{n^2, t_{\mathcal{A}}(3^{-k}n)\}) \tag{2.1}$$

time and has an error bound of

$$f(n) = (\frac{7}{3})^k (1 + f_{\mathcal{A}}(3^{-k}n)) - 1. \tag{2.2}$$

The (t,k)-heuristic generalizes the Onethird heuristic and uses some properties of the hypergreedy heuristic. This combination results in a class of heuristics, which improve the error bounds of the corresponding Onethird heuristics. Similarly to the Onethird,

the (t,k)-heuristic consists of $(k+1)$ stages. At each stage j, $0 \leq j \leq k-1$, it extracts from $K(V_j)$, $V_j \subset V(V_0 = V)$, in t steps, a sparse subgraph, using the same construction as in the first part of the hypergreedy. From the sparse graph, it selects the maximum cardinality minimum weight perfect matching which forms a partial matching of $K(V_j)$. After first k stages the heuristic possibly matches all the vertices of V. If this is not the case, then in the last $(k+1)$-th stage the algorithm makes use of an auxiliary exact or approximate perfect matching algorithm, $\mathcal{A}$, to match the remaining unmatched vertices.

The error of the (t,k)-heuristic, for weights satisfying the triangle inequality, is bounded above by

$$f(n) = (2t+1)^k[1 + f_A(n_k)] - 1 \tag{2.3}$$

where $f_{\mathcal{A}}(n_k)$ is the error of the algorithm $\mathcal{A}$, applied to a complete graph with $n_k \leq (\frac{1}{3t})^k n$ vertices. With appropriate choice of k the time complexity of the (t,k)-heuristic is $O(tn^2)$. For $t = \lfloor \log_3 n \rfloor$ and $k = 1$ the (t,k)-heuristic reduces to the hypergreedy and runs in $O(n^2 \log n)$ time. If $t = 1$ and $0 \leq k \leq \lfloor \log_3 n \rfloor$, the (t,k)-heuristic becomes the Onethird class of heuristics. One should note that in the (t,k)-heuristic, a simplified version of the Onethird is generalized, therefore for $t = \lfloor \log_3 n \rfloor$ and $k = 1$ the actual error bounds of the (t,k)-heuristic are slightly higher than than the corresponding bounds obtained for the Onethird. In the conclusion chapter tables illustrating the behavior of the (t,k)-heuristic are presented.

The *GGW* $O(n^2\sqrt{\log \log n})$-time algorithm finds a perfect matching whose weight is bounded above by 2 times the weight of the optimal perfect matching. Although the *GGW* algorithm has the interesting property of obtaining solutions within a constant error of two, one might be interested in obtaining even faster heuristics with reasonably small theoretical error. Clearly, on the surface the *GGW* algorithm is superior to the hypergreedy and does not leave any incentive to ever use the latter. In fact using the *GGW* algorithm in the last stage of Onethird, already produces a better heuristic than the hypergreedy (see below). As we shall see, the (t,k)-heuristic is more powerful than the Onethird, it makes use of the hypergreedy, and in conjunction with the *GGW*

algorithm results in a $\log_3 \log_3 \log_3 n$-error heuristic which runs in $O(n^2 \log\log\log n)$ time.

More specifically, suppose that in the $(k+1)$-th stage the *GGW* algorithm is used. Onethird (the (t,k)-heuristic with $t=1$) with $k = \frac{1}{4}\log_3 \log_3 \log_3 n$, gives a heuristic with time complexity of $O(n^2)$, and a reasonable error of $3(\log_3 \log_3 n)^{0.25} - 1$. Even this heuristic is better than the hypergreedy, both in time complexity and error. The (t,k)-heuristic with $t=4$, $k = \frac{1}{16}\log_3 \log_3 \log_3 n$, again has the time complexity of $O(n^2)$ and the error bounded above by $3(\log_3 \log_3 n)^{0.125} - 1$, which is even better than that of the Onethird. Finally, for $k=1$, $t = \frac{1}{4}\log_3 \log_3 \log_3 n$, a solution is obtained, which is bounded above by $(1.5 \log_3 \log_3 \log_3 n + 2)$ times the weight of the optimal solution. The corresponding time complexity is $O(n^2 \log\log\log n)$, still an improvement over the $O(n^2\sqrt{\log\log n})$ time of the *GGW* algorithm.

The other heuristic, the t-hypergreedy heuristic [31], which is a generalization of the hypergreedy, provides an approximate solution with the error bounded above by

$$f(n) = (2t+1), \tag{2.4}$$

and the time complexity of

$$T(n) = O(\max\{tn^2, \frac{n^3}{3^t}\}), \tag{2.5}$$

if at the last $(t+1)$ stage the exact optimal perfect matching algorithm is used. For $t = \lfloor \log_3 n \rfloor$, this heuristic reduces to the hypergreedy. Note that the exact perfect matching algorithm can be used at the last stage of the t-heuristic, for any t, because the perfect matching is polynomialy solvable. Since this does not apply to the m-perfect matching and m-subtree problem, for $m \geq 3$, it is impossible to obtain a generalization of the t-hypergreedy for those problems.

For Euclidean points in the plane, the t-hypergreedy is modified, such that its error is bounded above by $\alpha(2t+1)$, where $\alpha = 2.42$ is the constant corresponding to the worst case ratio of the weight of the shortest path in the Delaunay triangulation to the Euclidean distance in the plane [37]. The time complexity of the heuristic is

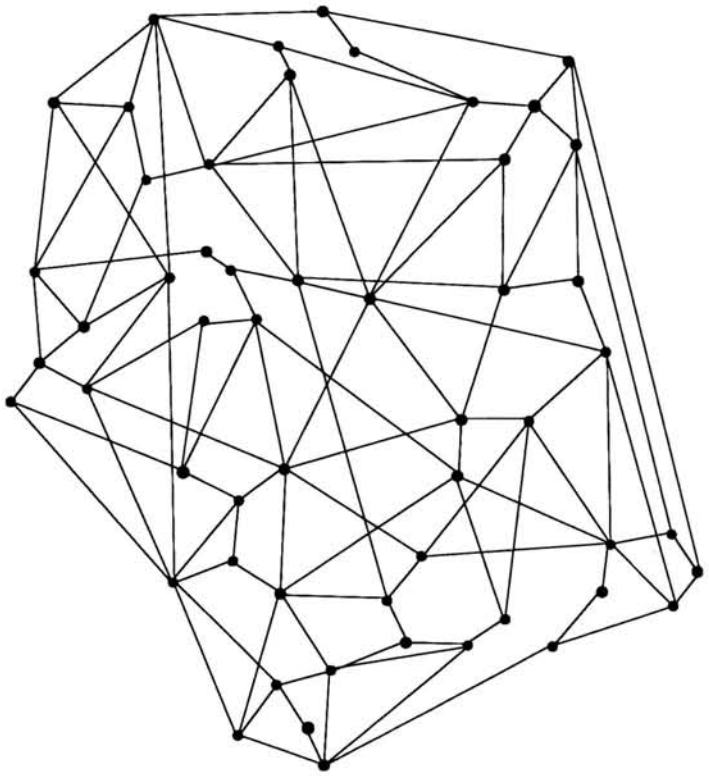


Figure 2.3: $G = (V, E)$ - an undirected edge-weighted graph.

$$T(n) = O(\max\{tn \log n, \frac{n^2 \log n}{3^t}, \frac{n^3}{3^{3t}}\}). \tag{2.6}$$

In particular, for $t = \lfloor \log_3 n \rfloor$ the modified t-hypergreedy has error bounded above by $\alpha(2\lfloor \log_3 n \rfloor + 1)$, and it can be implemented in $O(n \log^2 n)$ time. This time complexity is also favorable with respect to Vaidya's $O(n \log^3 n)$ time heuristic [53], for points in the Euclidean plane, whose error is bounded above by $3 \log_3 \frac{3}{2} n$.

2.2 Constrained Forest Problem.

Given an undirected edge-weighted graph $G = (V, E)$ on the vertex set V, $|V| = n$, with edge set E, see Fig.2.3, and a natural number $2 \leq m \leq n$, the *constrained forest problem (CFP)* is a forest, denoted by F_m^*, of minimum weight (sum of the weights of the edges in F_m^*), spanning all the vertices of G and such that each tree spans at least m vertices, see in Fig.2.4 the CFP of $G(V, E) = K(V)$, for $m = 6$.

Analogous to the classical minimum spanning tree problem, *CFP* finds applications

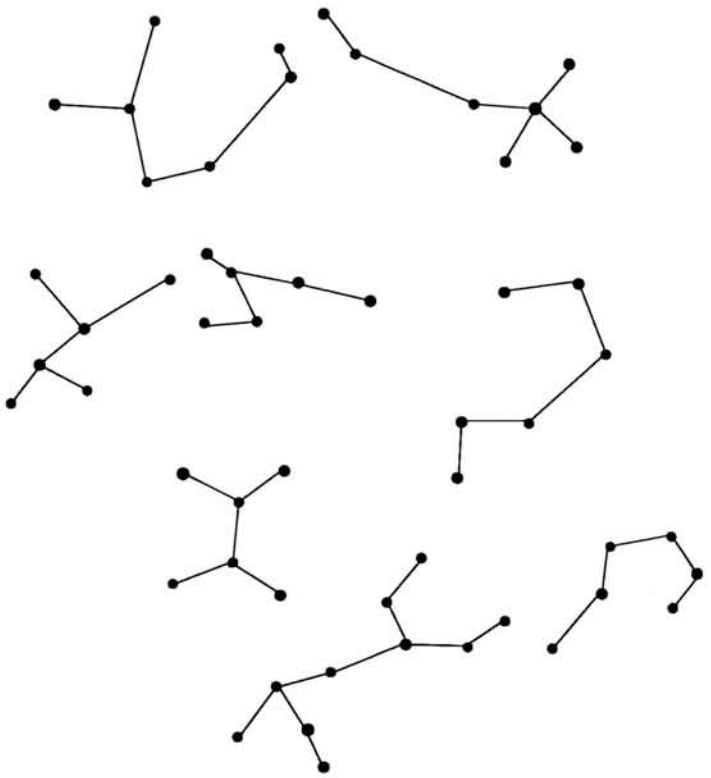


Figure 2.4: Constrained forest problem (F_m^*) for $m = 6$.

either directly or within the context of approximation algorithms for other combinatorial problems. For instance, approximate solutions to F_2^* obtained from a nearest neighbor graph of G have been used within heuristics for minimum-weight perfect matching, see e.g. Supowit, Plaisted, and Reingold [50], Plaisted [45], Grigoriadis and Kalantari [29], Imielinska and Kalantari [31] [32]. Moreover, for $m > 2$, approximate solutions to F_m^* have been considered in Goemans and Williamson [27] and Imielinska and Kalantari [33] in order to find approximate solutions for the more general NP-hard version of the optimal perfect matching problem, e.g. the optimal m-perfect matching problem and optimal m-subtree problem.

We prove that for $m \geq 4$, *CFP* is NP-hard. A simple greedy heuristic [33] is proposed for this problem, which is a modification of a minimum spanning tree algorithm, producing a solution whose weight is at most twice the optimal weight, for $m < n$. For $m = n$, our algorithm reduces to a minimum spanning tree algorithm, i.e. it provides an exact solution to the CFP. The time complexity of the algorithm is the same as

the time complexity for finding a minimum spanning tree. In particular, our algorithm runs in $O(|E|+|V|\log|V|)$ time for general edge-weighted graphs, and in $O(|V|\log|V|)$ time for Euclidean points in the plane. This 2-approximate heuristic improves the result by Gabow, Goemans and Williamson [27], whose 2-approximate algorithm runs in $O(|V|^2+|V|\sqrt{|E|\log\log|V|})$ time [24]).

2.3 Constrained Cycle Problem.

Given a complete undirected edge weighted graph $K(V)$ on the vertex set V, $|V|=n$, whose weights satisfy the triangle inequality, and a natural number $2 \leq m \leq n$, the *constrained cycle problem (CCP)* is a set of minimum weight, vertex disjoint cycles, denoted by C_m^*, covering all the vertices of G and such that each cycle has at least m vertices, see in Fig.2.5 the CCP of graph $K(V)$ from Fig.2.1. It is known that for $m \geq 5$ CCP is NP-hard (see [43], [55], [14]), polynomialy solvable for $m=2$ and $m=3$ [19], and for $m=4$ is open [27]. When $m=3$, the CCP becomes the 2-matching problem, and when $m=4$, it is the triangle-free 2-matching problem [13]. In the presence of the triangle inequality, analogous to the minimum spanning tree heuristic for the traveling salesman problem, [44], it is easy to show that a 2-approximate solution of F_m^*, in linear time gives rise to a 4-approximate solution for the CCP, for $2 \leq m < n$. Thus our 4-approximate algorithm which runs in $O(n^2)$ and $O(n\log n)$ time for the weights satisfying the triangle inequality and for points in the Euclidean plane, respectively, is faster than $O(n^2\sqrt{\log\log n})$ 2-approximate GGW algorithm.

Note, that for $m=n$, the CCP becomes the traveling salesman problem, and our algorithm reduces to the minimum spanning tree 2-approximate heuristic for the traveling salesman problem.

2.4 Optimal m-Perfect Matching.

Given a complete edge weighted graph $K(V)$, $|V|=n$, satisfying the triangle inequality, and a natural number m, dividing n, the *m-perfect matching* of $K(V)$ is a set of vertex disjoint cycles, of size exactly m, covering all the vertices V. *An optimal m-perfect*

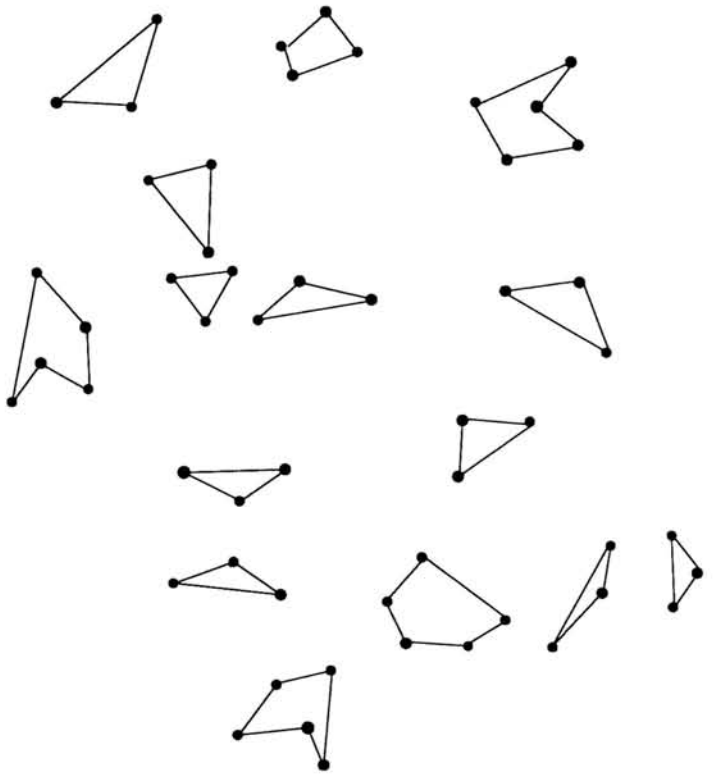


Figure 2.5: Constrained cycle problem (C_m^*) for $m = 3$.

matching of $K(V)$, denoted by $M_m^*(V)$, is an m-perfect matching with the minimum weight, see in Fig.2.6 the optimal 3-perfect matching of $K(V)$ from Fig.2.1. In particular, each cycle in the optimal m-perfect matching is an optimal Hamiltonian on the set of its vertices.

If $m = 2$, the optimal m-perfect matching is a duplicated minimum weight perfect matching, i.e. the set of minimum weight, vertex disjoint 2-cycles, covering the set of vertices V. While for $m = n$ the $M_m^*(V)$ reduces to the *traveling salesman problem (TSP)* [44], i.e. the minimum weight Hamiltonian cycle on the vertex set V.

Unlike the ordinary minimum weight perfect matching, which can be solved, in $O(n^3)$, by Edmonds' algorithm [17], the optimal m-perfect matching is *NP*-hard for $m \geq 3$ [26].

We develop two families of heuristic algorithms for the optimal m-perfect matching, called the *(t,k)-heuristic*, where t and k are integer parameters ranging from 1 to $\log n$, and the *Generalized Hypergreedy*. As is was mentioned in the introduction,

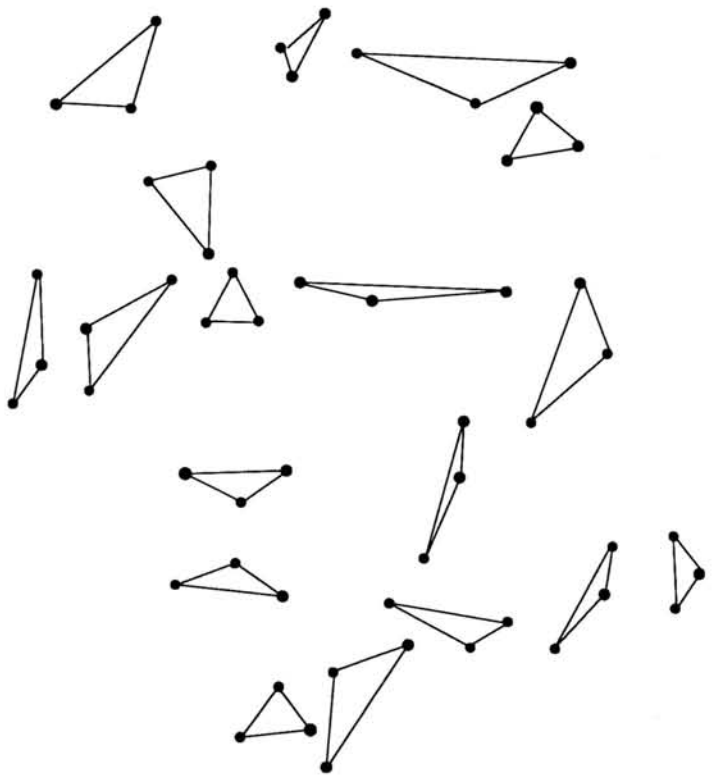


Figure 2.6: Optimal m-perfect matching (M_m^*) for $m = 3$.

the Generalized Hypergreedy is in fact a subclass of the (t, k)-heuristic, but it is analyzed separately in this thesis. Also a dynamic programming algorithm for m-perfect matching is introduced, which can be used in the above two classes of heuristics.

The (t, k)-heuristic for m-perfect matching is a generalization of the hypergreedy and Onethird heuristics for perfect matching, and additionally it makes use of the heuristic for the CFP. The heuristic consists of *(k+1)* stages; at each stage j, $0 \leq j \leq k-1$, it selects a *partial m-perfect matching*, which is a set of vertex disjoint m-cycles covering a fraction of vertices in V_j, $V_j \subset V$. The central structure, at each stage of the (t, k)-heuristic, is the t-basic graph, which is a sparse subgraph of $K(V_j)$, obtained in $O(tn^2)$ from a combination of the heuristic for the CFP and an algorithm based on the hypergreedy heuristic. The t-basic graph has the property that its edge-weight can be related to the weight of the optimal m-perfect matching of $K(V_j)$.

If after first k stages of the (t,k)-heuristic there are some vertices in V left unmatched, it uses an m-perfect matching algorithm $\mathcal{A}$, which can be either the Generalized Hypergreedy, the *GGW* algorithm, or our dynamic programming m-perfect matching algorithm.

The error of the *(t,k)-heuristic*, for weights satisfying the triangle inequality, is bounded above by by

$$f(n) = [\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_1)^k - \frac{4}{9}\frac{m}{m-1} \tag{2.7}$$

for $2 \leq m \leq \frac{n}{2}$, where n_k is the number of unmatched vertices after k stages, $f_{\mathcal{A}}(n_k)$ is the error of the algorithm $\mathcal{A}$, applied to a complete graph with n_k vertices, and

$$a_1 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(\frac{m-2}{m} + t)]. \tag{2.8}$$

For $m = n$, the (t,k)-heuristic for the m-perfect matching problem, reduces to the minimum spanning tree, 2-approximate heuristic, for the traveling salesman problem.

For Euclidean points in the plane, the (t,k)-heuristic uses, at each stage j, $0 \leq j \leq k-1$, an approximate t-basic graph, obtained from the Delaunay triangulation of V_j, in $O(tn\log n)$ time. The error of the (t,k)-heuristic for Euclidean points in the plane is bounded above by

$$f(n) = [\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_2)^k - \frac{4}{9}\frac{m}{m-1}, \tag{2.9}$$

for $2 \leq m \leq \frac{n}{2}$, where $a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(2\frac{m-1}{m} + \alpha(t-1))]$ and $\alpha = 2.42$. For $m = n$, the (t,k)-heuristic reduces to the 2-approximate minimum spanning tree heuristic for the Euclidean traveling salesman problem.

The time complexity of the (t,k)-heuristic, when k is selected appropriately, is $O(tn^2)$ for weights satisfying the triangle inequality, and $O(tn\log n)$ for the Euclidean points in the plane.

The second class of heuristics, the Generalized Hypergreedy is a combination of the heuristic for the CFP and the an algorithm based on the hypergreedy. Its central structure is also, as for the (t,k)-heuristic, the t-basic graph (the approximate t-basic

graph, for Euclidean points in the plane). The algorithm constructs the t-basic graph (the approximate t-basic graph), for such t, that all the resulting connected components in the graph admit m-perfect matchings, which form an approximate solution. The error of the Generalized Hypergreedy is bounded above by

$$f(n) = 2\lfloor \log_3 n \rfloor - 1 \tag{2.10}$$

for $m = 2$; and by

$$f(n) = 4\frac{m-1}{m}[2\frac{m-1}{m} + \lceil \log(\frac{n}{m}) \rceil - 1] \tag{2.11}$$

for $3 \leq m \leq \frac{n}{2}$. For $m = n$ this heuristic reduces to the minimum spanning tree 2-approximate heuristic for traveling salesman problem. For $m = \frac{n}{2}$ the error is $f(n) = 8$, for $m = \frac{n}{3}, \frac{n}{4}$ the error is $f(n) = 12$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 16$.

For Euclidean points in the plane, the error of the Generalized Hypergreedy is bounded above by

$$f(n) = 2\alpha\lfloor \log_3 n \rfloor \tag{2.12}$$

for $m = 2$; and by

$$f(n) = 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)], \tag{2.13}$$

for $3 \leq m \leq \frac{n}{2}$, where $\alpha = 2.42$. For $m = n$, this heuristic reduces to the 2-approximate minimum spanning tree heuristic for the Euclidean traveling salesman problem. For $m = \frac{n}{2}$, the error is $f(n) = 8$, for $m = \frac{n}{3}, \frac{n}{4}$, the error is $f(n) = 4(2 + \alpha)$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 8(1 + \alpha)$.

The time complexity of the Generalized Hypergreedy is

$$T(n) = O(n^2 \log(\frac{n}{m})), \tag{2.14}$$

for general weights satisfying the triangle inequality and

$$T(n) = O(n \log(\frac{n}{m}) \log n) \tag{2.15}$$

for the Euclidean points in the plane. For large $m = \frac{n}{k}$, where k is a small natural number dividing n, the Generalized hypergreedy becomes a constant-error heuristic for the m-perfect matching running in $O(n^2)$ and $O(n \log n)$ time for general weights satisfying the triangle inequality and for Euclidean points in the plane, respectively.

Both heuristics make use of the t-basic graph, denoted by $BG_t(W)$, where $W \subseteq V$ and $|W|$ is divisible by m, and $1 \leq t < \log n$. The t-basic graph is constructed in t stages and it is a collection of two classes of sparse connected components, which are trees, classified as of *type A* and *type B*. A type A connected component contains a multiple of m vertices and it can admit an m-perfect matching. Any connected component which is not type A, is classified as the type B connected component. Given W, the $BG_1(W)$, which is a forest of trees obtained by the heuristic for the CFP, where each tree has size at least m. For $m \leq \frac{|W|}{2}$, the total edge weight of $BG_1(W)$ is bounded above by $2\frac{(m-1)}{m}$ times the weight of $M^*_m(W)$. For $m = |W|$, the $BG_1(W)$ becomes a minimum spanning tree of W and its weight is bounded above by $\frac{(m-1)}{m}$ time the weight of $M^*_m(W)$, the solution to the traveling salesman problem of $K(W)$. Note that for $m = n$ we can construct only the 1-basic graph. For $m \neq |W|$, the total edge weight of the t-basic graph, does not exceed $[\frac{(m-2)}{m} + t]$ times the weight of the optimal m-perfect matching of $K(W)$. At each stage j, $0 \leq j \leq k-1$, of the (t,k)-heuristic, for a given t, we extract from the t-basic graph an m-perfect matching covering a fraction of vertices in $V_j \subset V$.

The Generalized Hypergreedy constructs the t-basic graph, for

$$t = (\lceil \log(\frac{n}{m}) \rceil). \tag{2.16}$$

Such t-basic graph guarantees that each connected component is of type A and admits an m-perfect matching. The union of the m-perfect matchings, obtained from all the connected components, forms an approximate solution.

For Euclidean points in the plane, the (t,k)-heuristic and the Generalized Hyper-greedy use approximate t-basic graph, denoted by $\widehat{BG}_t(W)$, where each Euclidean distance is replaced by the corresponding shortest path in the Delaunay triangulation, and its total weight is bounded above by

$$(2\frac{m-1}{m} + \alpha(t-1)) \tag{2.17}$$

times the weight of $M^*_m(W)$. This modified t-basic graph uses instead of $O(n^2)$ edges of the complete graph $K(V)$, only $O(n)$ edges of the Delaunay triangulation. The Delaunay triangulation is a planar graph and provides a good α-approximation for the Euclidean distances. Because the Delaunay triangulation can be constructed in $O(n \log n)$, using the approximated t-basic graph results in a significant reduction of the time complexity of the corresponding heuristics, while their errors are affected only by the constant ratio of α.

The *(t,k)*-heuristic selects at each stage $j = 1, 2, ..., k$, a partial m-perfect matching from the t-basic graph $BG_t(V_j)$ (from the approximate t-basic graph, $\widehat{BG}_t(V_j)$, for the Euclidean points in the plane), where $V_j \subseteq V$. The edge weight of the partial m-perfect matching does not exceed

$$4\frac{m-1}{m}(\frac{m-2}{m} + t) \tag{2.18}$$

times the weight of the optimal solution, for weights satisfying the triangle inequality and

$$4\frac{m-1}{m}(2\frac{m-1}{m} + \alpha(t-1)) \tag{2.19}$$

times the weight of the optimal solution, for Euclidean points in the plane) times the weight of the optimal m-perfect matching of $K(V_j)$. After k stages, an m-perfect matching algorithm $\mathcal{A}$ is applied to the remaining n_k unmatched vertices. The union of the m-perfect matchings obtained from all the $(k+1)$ stages forms an m-perfect matching of $K(V)$, whose total error, for general weights satisfying the triangle inequality and the Euclidean points in the plane, we described above. In the last stage, either

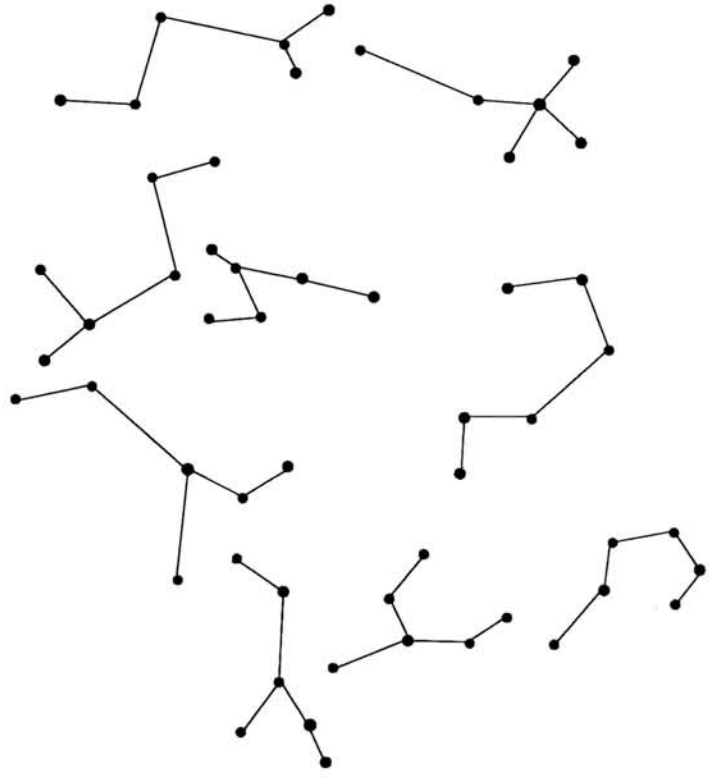


Figure 2.7: Optimal m-subtree problem (T_m^*) for $m = 6$.

our exact dynamic programming algorithm for the optimal m-perfect matching or the *GGW* algorithm can be used. The time complexity of this exact dynamic programming algorithm is $O(m^2 2^{n+m} + 2^{2n-m})$, a big improvement over a brute force method which would compare all possible m-perfect matchings in $O((\frac{(m-1)!}{m!})^{\frac{n}{m}} n!)$.

2.5 Optimal m-Subtree Problem.

Given a complete edge weighted graph $K(V)$, $|V| = n$, satisfying the triangle inequality, and a natural number m, dividing n, the *m-subtree problem* of $K(V)$ is a set of vertex disjoint trees, each of size exactly m, covering all the vertices V. *An optimal m-subtree problem* of $K(V)$, denoted by T_m^*, is an m-subtree problem with minimum weight, see in Fig.2.7 the optimal 6-subtree problem of $K(V)$ from Fig.2.1. In particular, each tree in the optimal m-subtree problem is a minimum spanning tree on the set of its vertices. It can be proved that the optimal m-subtree problem in *NP*-hard for $4 \leq m \leq \frac{n}{2}$.

For $m = 2$ the optimal m-subtree problem becomes the minimum weight perfect

matching problem, while for $m = n$ it reduces to the minimum spanning tree problem on the vertex set V.

Similarly to the heuristic algorithms for the optimal m-perfect matching, we introduce the same two classes of heuristics for the m-subtree problem, the *(t,k)-heuristic*, with parameters t and k ranging from *0* to $\log n$, and the *Generalized Hypergreedy*. Also an exact dynamic programming algorithm will be presented for this problem.

The t-basic graph, $BG_t(W)$, $W \subset V$, (approximate t-basic graph) which is used in both, the (t,k)-heuristic and the Generalized Hypergreedy, has its weight is bounded above by $2t$ and (by $2(1+\alpha(t-1))$) times weight of $T^*_m(W)$.

The (t,k)-heuristic consists of *(k+1)* stages; at each stage j, $1 \leq j \leq k$, it selects a *partial* solution to T^*_m spanning a fraction of vertices in V_j, $V_j \subset V$. After k stages, if there are some vertices in V left, which are not spanned, the heuristic uses an algorithm $\mathcal{A}$ for m-subtree problem, which can be either the Generalized Hypergreedy, the *GGW* algorithm, or an exact dynamic programming algorithm.

The error of the *(t,k)-heuristic*, for weights satisfying the triangle inequality, is bounded above by Equation. 2.7 for $2 \leq m \leq \frac{n}{2}$, where n_k is the number of unmatched vertices after k stages, $f_{\mathcal{A}}(n_k)$ is the error of the algorithm $\mathcal{A}$, applied to a complete graph with n_k vertices, and

$$a_1 = 2\frac{m-1}{m}[1 + 4t\frac{m-1}{m}]. \tag{2.20}$$

For Euclidean points in the plane, the error of the (t,k)-heuristic is bounded above by Equation. 2.9 for $2 \leq m \leq \frac{n}{2}$, where

$$a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(1+\alpha(t-1))]. \tag{2.21}$$

The time complexity of the (t,k)-heuristic, when k is selected appropriately, is $O(tn^2)$ for weights satisfying the triangle inequality, and $O(tn\log n)$ for the Euclidean points in the plane.

The error of the Generalized Hypergreedy, for general weights satisfying the triangle inequality, is bounded above by

$$f(n) = 2\lfloor \log_3 n \rfloor \tag{2.22}$$

for $m = 2$, which it the error of the hypergreedy for perfect matching; and by $f(n) = 4\frac{m-1}{m}[\lceil \log(\frac{n}{m}) \rceil]$, for $3 \leq m \leq \frac{n}{2}$. The time complexity of the Generalized Hypergreedy is $O(n^2 \log(\frac{n}{m}))$, for general weights satisfying the triangle inequality.

For $m = n$, the the heuristic reduces to the exact minimum spanning tree algorithm, since the optimal m-subtree problem becomes the minimum spanning tree problem. For $m = \frac{n}{2}$ the error is $f(n) = 4$, for $m = \frac{n}{3}, \frac{n}{4}$ the error is $f(n) = 8$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 12$. For all the above values of m, this heuristic runs in $O(n^2)$ time, a better time than that the $O(n^2\sqrt{\log \log n})$ time of the respective 4-approximate *GGW* algorithm. For $m = 2$, the Generalized Hypergreedy for the m-subtree problem, becomes the $O(n^2 \log n)$-time hypergreedy heuristic for perfect matching.

For Euclidean points in the plane, the error of the Generalized Hypergreedy is bounded above by $f(n) = 2\alpha\lfloor \log_3 n \rfloor$, for $m = 2$, which corresponds to the error of our implementation of the Euclidean hypergreedy heuristic; and by

$$f(n) = 4\frac{m-1}{m}[1 + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)], \tag{2.23}$$

for $3 \leq m \leq \frac{n}{2}$. The time complexity of the Generalized Hypergreedy for Euclidean points in the plane is $O(n \log(\frac{n}{m}) \log n)$.

For $m = n$, this heuristic reduces to the exact Euclidean minimum spanning tree algorithm. For $m = \frac{m}{2}$ the error of the heuristic is $f(n) = 4$, for $m = \frac{n}{3}, \frac{n}{4}$ the error is $f(n) = 4(1 + \alpha)$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 4(2 + 2\alpha)$. For all the above values of m, this heuristic runs in $O(n \log n)$ time. For $m = 2$, the Generalized Hypergreedy reduces to our $O(n \log^2 n)$-time implementation of the Euclidean hypergreedy.

A dynamic programming algorithm for optimal m-subtree problem is proposed, which is similar to the dynamic programming algorithm for the optimal m-perfect matching, and has the time complexity of 2^{2n-m}. This algorithm can be used in the last stage of the (t, k)-heuristic, when the size of the problem is small enough, $O(\log n)$,

to improve the overall error of the heuristic.

Chapter 3

An Approximation Algorithm For the Constrained Forest and Cycle Problems

In this chapter we will present an approximation algorithm for the constrained forest problem CFP, and use it to derive an approximation algorithm for the constrained cycle problem CCP. For $m \geq 4$, we show that the CFP problem is NP-hard. We describe a simple 2-approximate greedy heuristic for the CFP that runs within the time needed to compute a minimum spanning tree. If the edge weights satisfy the triangle inequality, any such a 2-approximate solution, in linear time, can be converted into a 4-approximate solution for the CCP.

3.1 *NP*-hardness.

Theorem 3.1.1 *For $m \geq 4$, the CFP is NP-hard.*

Proof: The theorem will be proved for $m = 4$. The proof is easily generalizable for $m > 4$. We reduce the 3-dimensional matching problem to the *CFP*. Consider a given tripartite graph $G_1 = (V_1, E_1)$, where V_1 is the union of three disjoint set of vertices A, B, and C, each of the same cardinality l. A triplet (a, b, c), where $a \in A$, $b \in B$, and $c \in C$ is *acceptable* if the edges (a, b), (a, c), and (b, c) are in E_1. The 3-dimensional matching problem is to determine if all the vertices in V_1 can be covered by vertex disjoint acceptable triplets, and it is NP-complete (see [26]).

Given G_1, consider the edge-weighted layered graph $G_2 = (V_2, E_2)$ with layers, where $V_2 = V_1 \bigcup D \bigcup R$; see Fig.3.1. The bottom layer consists of $3l$ vertices in V_1. The middle layer consists of the vertex set D with Δ vertices, where Δ is the number of acceptable triplets (a, b, c) in G_1. Each vertex $d \in D$ is connected to the vertices

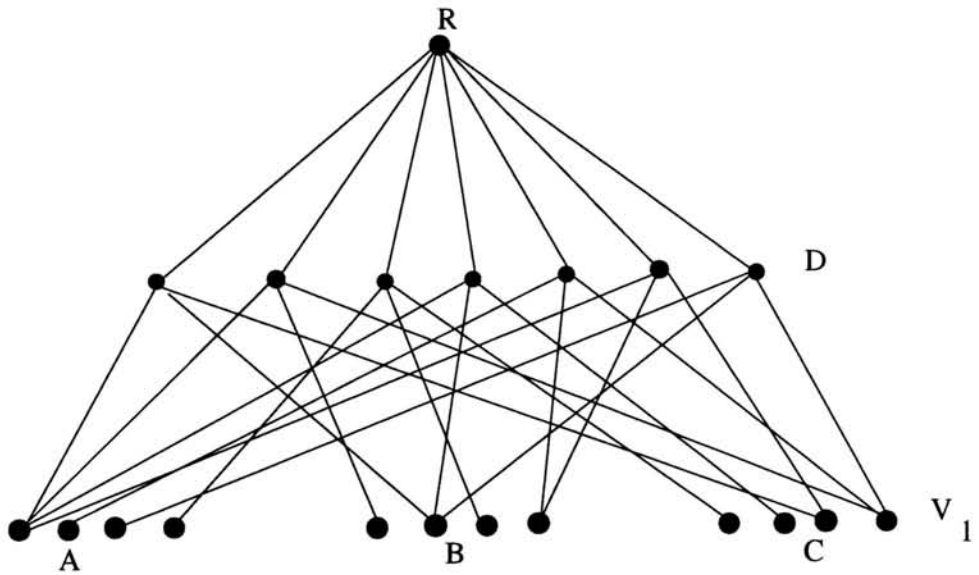


Figure 3.1: Graph $G_2 = (V_2, E_2)$.

in A, B, and C which define the corresponding triplet by edges of weight 1. The top layer R consists of m copies of an auxiliary vertex. Let us assume that all these copies are connected to each other by edges of weight zero. In addition one of these copies is connected to each vertex in D by an edge of weight $0 < \epsilon < \frac{1}{\Delta}$. We assume that G_2 is connected, i.e. each vertex in V_1 is connected to a vertex in D. Otherwise, G_1 does not admit a 3-dimensional matching.

Let F_m^* be an optimal constrained forest of G_2 and let $\omega(F_m^*)$ denote its total edge weight. We claim that $\omega(F_m^*) = 3l + \epsilon(\Delta - l)$ if and only if G_1 admits a 3-dimensional matching. To prove this let us first prove some facts.

(i): There do not exist vertices $v \in V_1$, $d, d' \in D$ such that the edges (d, v), (d', v) are in F_m^*. Otherwise, replacing one of these edges by the two edges (d, R) and (d', R), results in another feasible forest whose weight is less than $\omega(F_m^*)$.

Let D_3 be the set of vertices $d \in D$ such that the three edges of the tree connecting d to its corresponding triplet (a, b, c) belong to F_m^*.

(ii): $\omega(F_m^*) = 3l + \epsilon(\Delta - |D_3|) \geq 3l + \epsilon(\Delta - l)$. To prove this one can observe that since each vertex in V_1 is incident to an edge in F_m^*, the integral part of $\omega(F_m^*)$ is at least $3l$. On the other hand $\omega(F_m^*)$ is bounded above by $(3l + \epsilon\Delta)$ which is the weight of the minimum spanning tree of G_2 which in turn is less than $(3l + 1)$. Thus, the integral

part of $\omega(F_m^*)$ is exactly $3l$. Let d be a vertex in $D \setminus D_3$. From (i), the edge (d, R) must be in F_m^*. Also, from the optimality of F_m^*, if $d \in D_3$, then the edge (d, R) is not in F_m^*. Hence the fractional part of $\omega(F_m^*)$ is $\epsilon(\Delta - |D_3|)$. To prove the lower bound, observe that if $|D_3| > l$, then $\omega(F_m^*) > 3|D_3| > (3l + 1)$, a contradiction.

Now we prove the claim. Suppose $\omega(F_m^*) = 3l + \epsilon(\Delta - l)$. From (ii), this implies that $|D_3| = l$. From (i), the trees of F_m^* containing vertices in D_3 must contain all the vertices of V_1. Hence, the set of the triplets (a, b, c) corresponding to the vertices in D_3 give a 3-dimensional matching of G_1. Conversely, suppose that G_1 admits a 3-dimensional matching. If in G_2 vertices of each triplet are connected, in the 3-dimensional matching, to their corresponding vertex in D, there are l trees each with $m = 4$ vertices and of weight equal to 3. Next, connecting all the remaining $\Delta - l$ vertices in D to R, gives a tree of weight equal to $\epsilon(\Delta - l)$. Thus the total weight of these $(l+1)$ trees is $3l + \epsilon(\Delta - l)$ which from (ii) implies optimality of the corresponding forest. Clearly, this proof can be generalized to $m > 4$. □

Next, we show that the NP-hardness extends to the case where the weights satisfy the triangle inequality.

Corollary 3.1.1 *For $m \geq 4$, CFP remains to be NP-hard for complete graphs satisfying the triangle inequality.*

Proof: Consider G_2 as in Theorem 2.1, with F_m^* as its optimal constrained forest. Let $\widehat{G}_2$ be the complete graph on V_2 whose edge weights are the weights of the shortest paths in G_2. Also let $\widehat{F}_m^*$ be an optimal constraint forest of $\widehat{G}_2$. Clearly, $\omega(F_m^*) \geq \omega(\widehat{F}_m^*)$. To prove the opposite inequality, suppose that each edge in $\widehat{F}_m^*$ is replaced with its corresponding shortest path in G_2. This results in a subgraph from which a feasible constrained forest can be obtained for G_2, with weight not exceeding $\omega(\widehat{F}_m^*)$. □

3.2 A greedy heuristic for CFP.

In this section we describe a simple heuristic which assumes that a minimum spanning tree of G whose edges are denoted by MST, is at hand. Using the edges in MST,

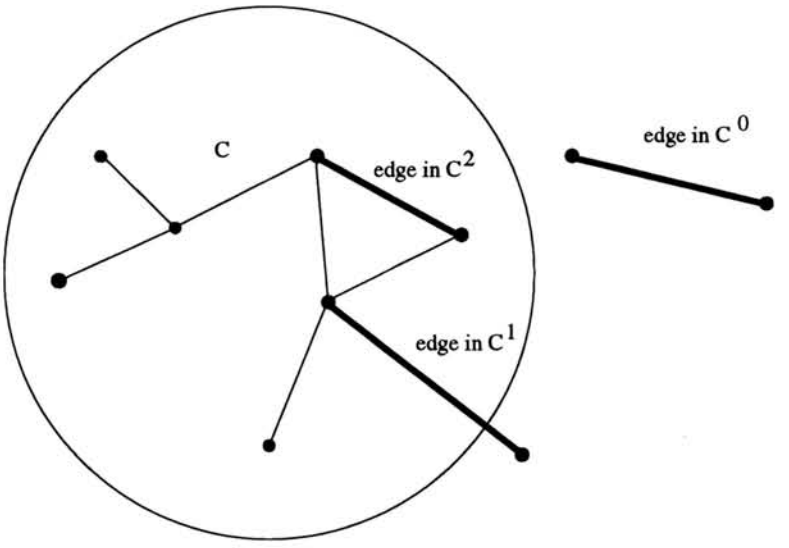


Figure 3.2: Edges in $C^j \subset G$, $j = 0, 1, 2$, (in bold), incident to a connected component C.

sorted according to their weight, the heuristic selects in a greedy fashion a forest F_m such that each of its trees is incident to at least m vertices.

Algorithm Greedy

Input: $W = MST$, $F_m = \emptyset$

Output: F_m forest of trees, each spanning at least m vertices.

begin

while F_m does not span V **do**

if $W \neq \emptyset$ *and the shortest edge* $e \in W$ *does not connect two trees of size at least* m *in* F_m **then** $F_m = F_m \bigcup \{e\}$;

$W = W \setminus \{e\}$;

end

In the remaining of this chapter we show that the weight of the final spanning forest F_m is bounded above by twice the weight of F_m^*. In order to prove this bound, first we will show the existence of a function $\xi_* : F_m \rightarrow F_m^*$ such that:

1. the weight of each edge in F_m is bounded above by the weight of its image.

2. ξ_* maps at most two edges in F_m onto each edge in F_m^*.

Let us first give some definitions. For any subgraph C of G, let *size(C)* be the number of vertices incident to the edges in C. If x is a vertex or an edge of C then $x \in C$. Let us denote by $\omega(C)$ the sum of the weights of the edges in C. Let C^j, $j = 0, 1, 2$ be the set of edges in G which are incident to j vertices in C. Note that an edge in C^2 is not necessarily in C. Fig.3.2 illustrates a connected component C of the approximate solution F_m and some of the edges in $C_j \subset G$, $j = 0, 1, 2$, (in bold).

Lemma 3.2.1 *Let S_k be the set of first k edges selected into F_m by the Greedy algorithm. Let C be any connected component of S_k satisfying size$(C) < m$. If $f \in C^1$, then there exists an edge $g \in MST \bigcap C^1$ satisfying $\omega(g) \leq \omega(f)$.*

Proof: Assume $f = (v_1, v_2)$, with $v_2 \in C$. If $f \in MST$, then the lemma is obviously true. Assume otherwise. Adding f to MST gives a cycle $(v_1, v_2, ..., v_j, v_1)$, $j \geq 3$. By a well known property of MST, the weight of f is greater than or equal to the weight of each edge in the cycle. If $v_3 \notin C$, let $g = (v_2, v_3)$. Otherwise, if $v_4 \notin C$, let $g = (v_3, v_4)$, and so on. If $v_3, v_4, ..., v_j$ are all in C then $g = (v_j, v_1)$ is the desired edge, since $v_1 \notin C$. □

Lemma 3.2.2 *Let S_k be as in Lemma 3.2.1 and C any connected component of S_k. There exists a mapping ξ_k: $S_k \rightarrow F_m^*$ such that*

(i): $\omega(e) \leq \omega(\xi_k(e))$ for any $e \in S_k$.

(ii): If size$(C) < m$, there exists $f \in C^1 \bigcap F_m^*$, such that $|\xi_k^{-1}(f) \bigcap C| = 0$, i.e. ξ_k does not map any edge in C onto f.

(iii): If $f \in C^j \bigcap F_m^*$, $j = 0, 1, 2$, then $|\xi_k^{-1}(f) \bigcap C| \leq j$, i.e. ξ_k maps at most j edges in C onto f.

Proof: The lemma is proved by induction on k.

(*Base*:) For $k = 1$, $S_k = \{e\} = C$, see Fig.3.3(a). Suppose $e \in F_m^*$, let us define $\xi_1(e) = f = e$. If $C^1 \bigcap F_m^* = \emptyset$, then $m = 2$ and condition (ii) is void. Also (i) and (iii) are satisfied. If $C^1 \bigcap F_m^* \neq \emptyset$ containing say f', (ii) is satisfied, see Fig.3.3(b).

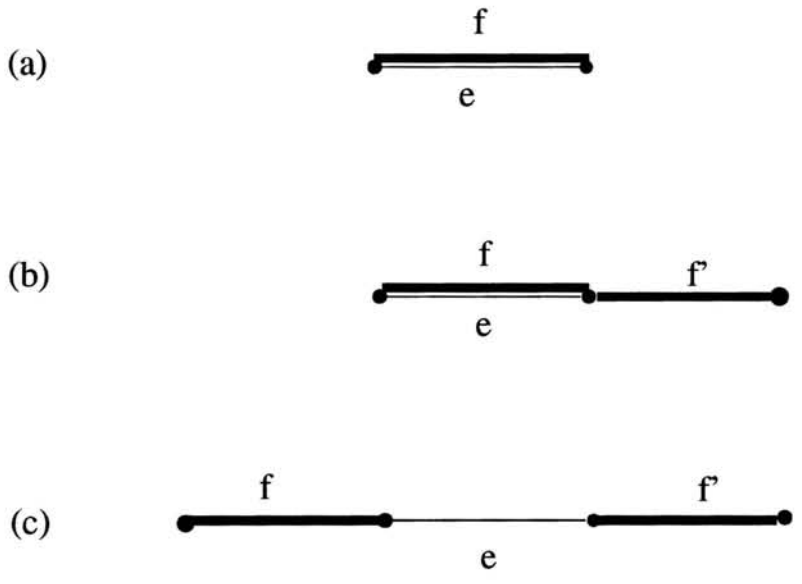


Figure 3.3: $S_1 = \{e\}$.

Conditions (i) and (iii) are again true. If $e \notin F_m^*$, see Fig.3.3(c), then there must be at least two edges in $C^1 \bigcap F_m^*$, say f and f'. Define $\xi_1(e) = f$. Since an edge with the smallest weight in MST is necessarily an edge with the smallest weight in the entire graph G, then $\omega(e) \leq \omega(f)$. Condition (ii) is trivially true for f' and (iii) is obvious.

(*Induction*:) Assume that ξ_k exists for a certain k. We show how to construct ξ_{k+1} from ξ_k. Let $e_{k+1} = (a, b)$ be the $(k+1)$-th edge selected into F_m by the algorithm. Let C_a and C_b be the connected components of S_k incident to the vertices a and b, respectively. Note that by our algorithm either $size(C_a) < m$ or $size(C_b) < m$. Assume that $size(C_a) < m$. Let $f_a \in C_a^1 \bigcap F_m^*$ satisfy (ii), which exists by the inductive hypothesis. Also if $size(C_b) < m$, let $f_b \in C_b^1 \bigcap F_m^*$ satisfy (ii). Let $C = C_a \bigcup C_b \bigcup \{e_{k+1}\}$.

The following cases are possible:

case(1): $size(C) \geq m$, see Fig.3.4.

case(2): $size(C) < m$ and $f_b \in C^1 \bigcap F_m^*$, see Fig.3.5.

case(3): $size(C) < m$ and both $f_a, f_b \in C^2 \bigcap F_m^*$, see Fig.3.6.

In Fig.3.4 and Fig.3.5 the two copies of f_a, solid and dashed, indicate the possibility that f_a may belong to $C^1 \bigcap F_m^*$ or $C^2 \bigcap F_m^*$. We will prove that the function

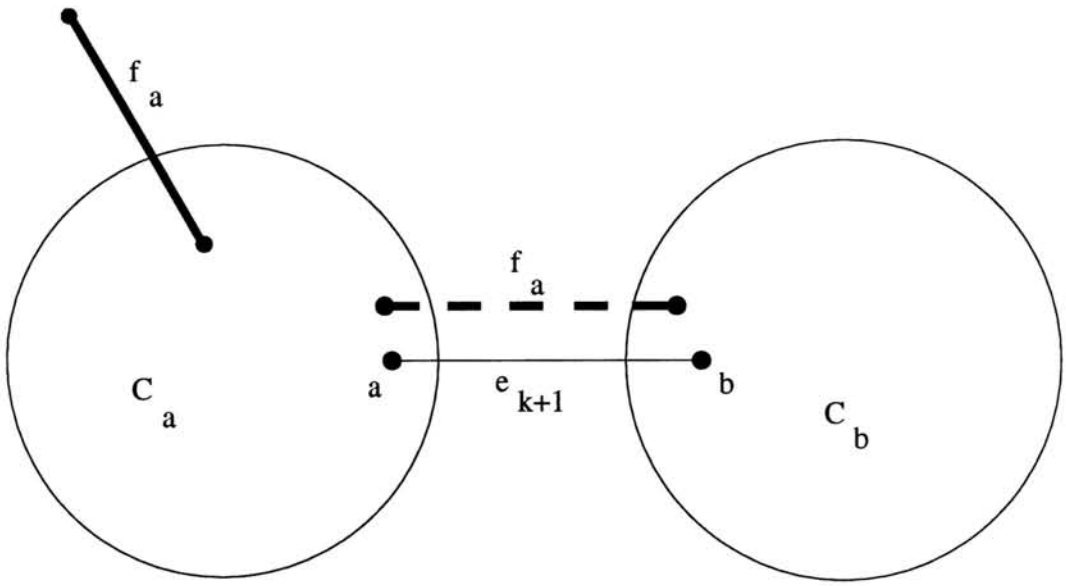


Figure 3.4: $size(C) \geq m$ and $f_a \in C_a^1 \bigcap F_m^*$.

$$\xi_{k+1}(e) = \begin{cases} \xi_k(e), & for\ any\ e \in S_k \\ f_a, & for\ e = e_{k+1} \end{cases} \qquad (1)$$

satisfies the conditions (i) - (iii) of the lemma, for $S_{k+1} = S_k \bigcup \{e_{k+1}\}$, in cases(1) and (2). To verify (i) one only has to show that $\omega(e_{k+1}) \leq \omega(f_a)$, since the other inequalities follow from the inductive hypothesis for ξ_k. This inequality is valid because otherwise, by Lemma 3.1, there is an edge $g \in MST \bigcap C_a^1$ such that $\omega(g) \leq \omega(f_a)$, and the algorithm would have selected the smallest such an edge instead of e_{k+1}.

Next it would be shown that condition (iii) is satisfied in cases(1) and (2). Let f be an edge in F_m^*. For $f \neq f_a$ the set of edges which map onto f stays the same. If no edges in S_k map onto f, then obviously (iii) still holds for f. If exactly one edge in S_k maps onto f, then (iii) still holds. If two edges in S_k map onto f, then even if one is in C_a and the other is in C_b, which are now joined, (iii) is still satisfied. Since at most one edge in S_k map onto f_a and no edges from C_a do, one could conclude that (iii) holds for $f = f_a$ as well.

Condition (ii) is clearly satisfied by the inductive hypothesis for case(1) (it does not apply to C but will apply to other connected components of *size* less than m). Fore case(2), the edge f_b satisfies (ii).

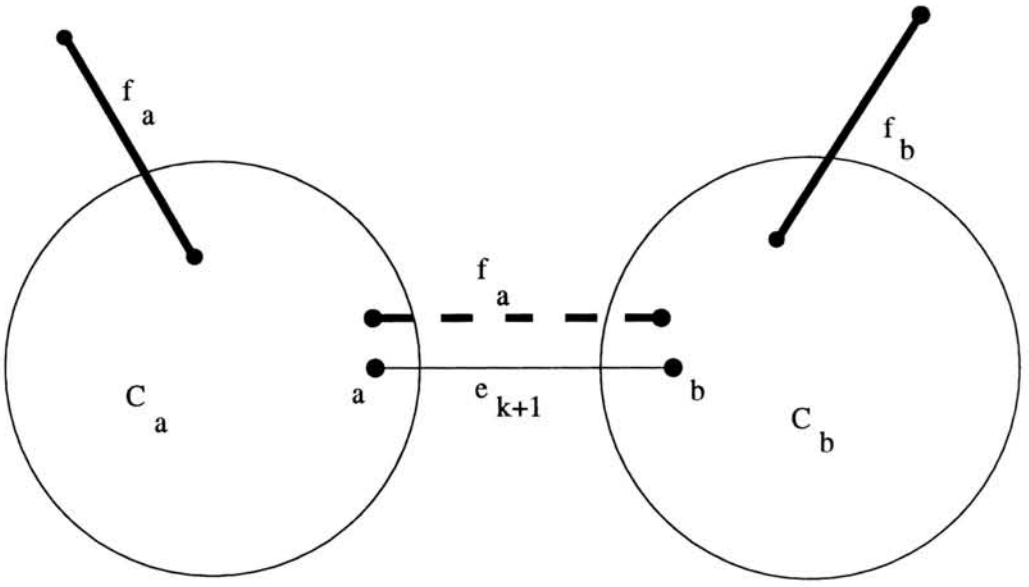


Figure 3.5: $size(C) < m$ and $f_b \in C^1 \bigcap F_m^*$.

Now consider case(3). Since $size(C) < m$, there must exist an edge $f^* \in C^1 \bigcap F_m^*$. If $|\xi_k^{-1}(f^*) \bigcap (C_a \bigcup C_b)| = 0$, i.e. ξ_k does not map any edge in $C_a \bigcup C_b$ onto f^*, then ξ_{k+1} defined as in (1) still satisfies (i) - (iii): Clearly (i) is true as before. Also f^* satisfies (ii). To check (iii), suppose $f_a \neq f_b$. Since $|\xi_k^{-1}(f_a) \bigcap C_a| = 0$, and $|\xi_k^{-1}(f_a) \bigcap C_b| \leq 1$, then $|\xi_{k+1}^{-1}(f_a) \bigcap C| \leq 2$. If $f_a = f_b$, then $|\xi_k^{-1}(f_a) \bigcap (C_a \bigcup C_b)| = 0$. Hence, $|\xi_{k+1}^{-1}(f_a) \bigcap C| = 1$. As before for other edges of F_m^* the appropriate inequalities will hold.

Now suppose that $|\xi_k^{-1}(f^*) \bigcap (C_a \bigcup C_b)| = 1$, then ξ_{k+1} has to be modified. In this case let e^* be an edge in $C_a \bigcup C_b$ such that $\xi_k(e^*) = f^*$. The ξ_{k+1} is defined as follows:

$$\xi_{k+1}(e) = \begin{cases} \xi_k(e), & for\ e \in S_k\ and\ e \neq e^* \\ f_a, & for\ e = e^* \\ f_b, & for\ e = e_{k+1} \end{cases} \qquad (2)$$

To verify (i), one can observe that by the construction of the algorithm and Lemma 3.1, $\omega(e^*) \leq \omega(e_{k+1}) \leq \min\{\omega(f_a), \omega(f_b)\}$. The edge f^* satisfies (ii). Regardless of the equality of f_a and f_b, (iii) will follow as argued before.

Theorem 3.2.1 $\omega(F_m) \leq 2\omega(F_m^*)$.

Proof: Let $\xi_* : F_m \rightarrow F_m^*$, where $\xi_* = \xi_k$, for a k for which $S_k = F_m$. By (i) of Lemma 3.2, for every $e \in F_m$ there is $\omega(e) \leq \omega(\xi_*(e))$. To prove the theorem one only need

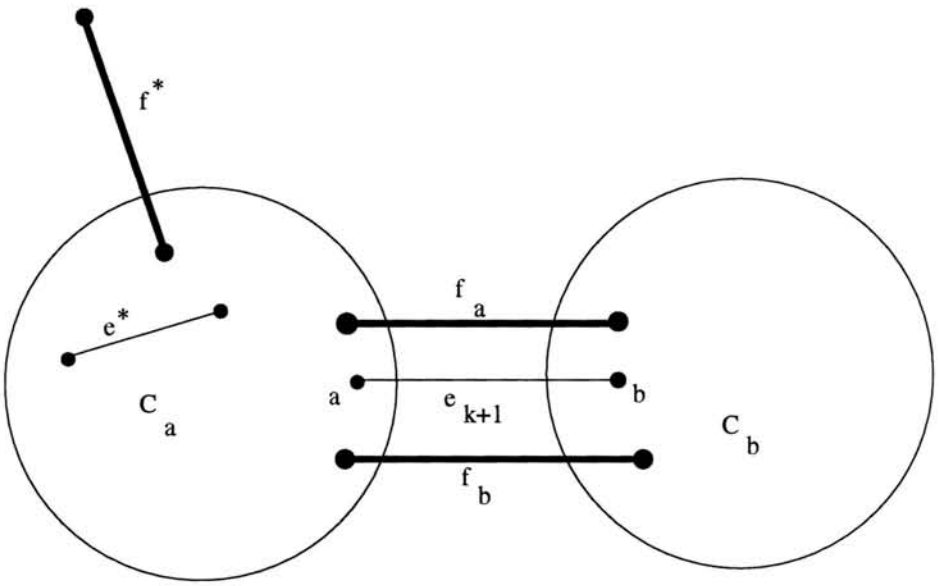


Figure 3.6: $size(C) < m$ and $f_a, f_b \in C^2 \bigcap F_m^*$.

to argue that for each $f \in F_m^*$, $|\xi_*^{-1}(f)| \leq 2$. Condition (iii) of Lemma 3.2 for $j = 0$, implies that only edges from connected components of F_m which are incident to f may be mapped onto f. If f is incident to two different connected components of F_m, then condition (iii), for $j = 1$ implies that at most one edge from each of the two connected component are mapped onto f. Therefore there are at most two edges mapped onto f. If f is incident to a single connected component F_m, then condition (iii) for $j = 2$ implies that at most two edges of the connected component are mapped onto f. Thus, the desired inequality holds. □

The time complexity of the heuristic is the same as of the minimum spanning tree algorithm, since in the first stage a minimum spanning tree is built, than in the second stage an algorithm is run which is again a modified minimum spanning tree algorithm on the edge set obtained in the first stage. Thus the overall time complexity is $O(|E| + |V| \log |V|)$ (if the Prim's minimum spanning algorithm [12] is used) and $O(|V| \log |V|)$ (if the Euclidean minimum spanning tree algorithm [46] is applied) for general weights and for Euclidean points in the plane, respectively. This makes our heuristic superior to the $O(|V|^2 + |V|\sqrt{|E| \log \log |V|}$-time GGW algorithm, which provides a 2-approximate solution for the CFP.

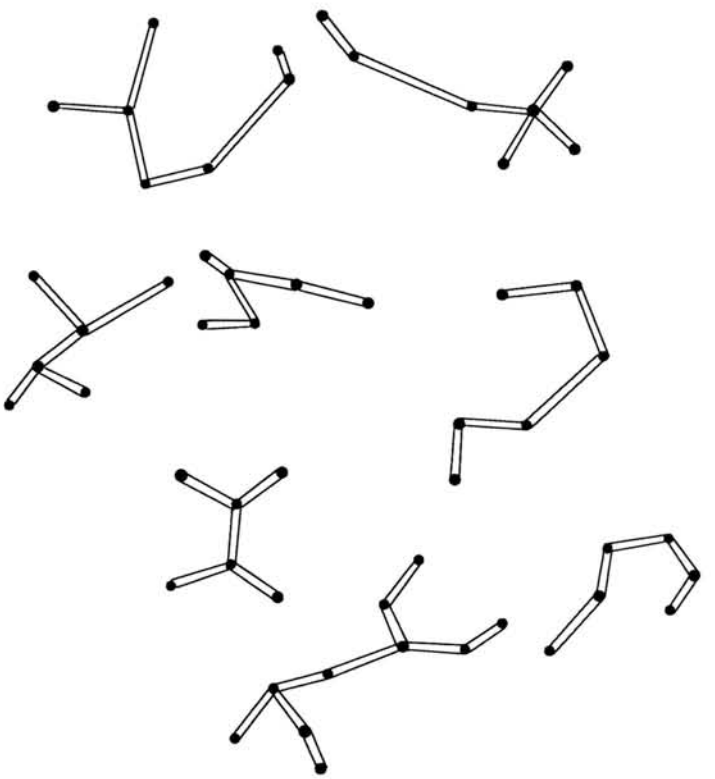


Figure 3.7: An approximate solution to the CFP of $K(V)$ with duplicated edges.

3.3 Heuristic for CCP.

Given a complete undirected edge weighted graph $K(V)$ on the vertex set V, $|V| = n$, whose weights satisfy the triangle inequality, the the *constrained cycle problem* (CCP), denoted by C_m^*, is the problem of finding a set of disjoint cycles of minimum-weight, covering all the vertices of a given graph, so that each cycle covers at least m vertices. As it was said before, in the presence of the triangle inequality, given a 2-approximate solution to the CFP on $K(V)$, one can construct, in linear time, a 4-approximate solution to the CCP, analogous to the minimum spanning tree heuristic for the traveling salesman problem [44].

First, we will show that the weight of F_m, an approximate solution to the CFP is bounded above by two times the weight of a solution to the CCP, i.e. $\omega(F_m) \leq 2\omega(C_m^*)$. The solution to the CCP which is a collection of cycles, each of size at least m, after removing from each cycle an edge becomes a forest, where each tree spans at least m vertices. Thus $\omega(F_m^*) \leq \omega(C_m^*)$ and $\omega(F_m) \leq 2\omega(F_m^*) \leq 2\omega(C_m^*)$.

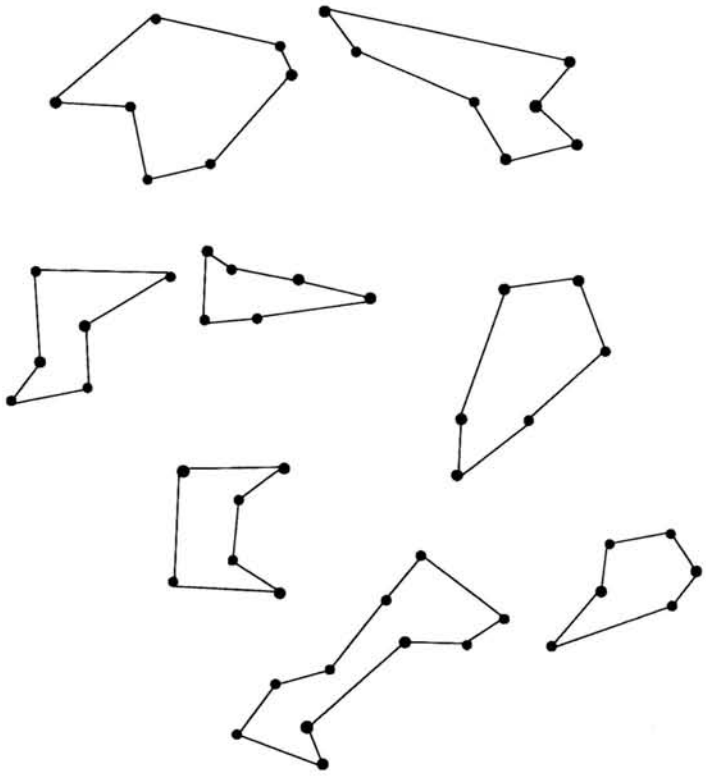


Figure 3.8: An approximate solution to the CCP.

Given an approximate solution to the CFP, by converting each tree into a cycle, an approximate solution to the CCP is obtained, as follows. The edges are duplicated in each tree in F_m, Fig.3.7 illustrates such duplicated forest, and an Euler tour is extracted from each component, which is a cycle which passes through each edge exactly once. The weight of the union of the Euler tours is twice the weight of the forest. Now, using the triangle inequality, each Euler tour is converted into a Hamiltonian cycle, a cycle which passes through each vertex exactly once, whose weight is bounded above by the one of the tour. The union of the Hamiltonian cycles, see Fig.3.8, whose total weight bounded by 2 the weight of the solution to CFP, forms a 4-approximate solution to the CCP.

Chapter 4

The t-basic Graph

The *t-basic graph* is used by the (t, k)-heuristic, the Generalized Hypergreedy, and the t-hypergreedy. In this section we will describe the construction of the t-basic graph, and analyze its time complexity. The weight of the t-basic graph extracted from a complete graph would be related to the weight of the solution for the optimal m-perfect matching and the optimal m-subtree problem, respectively. We will analyze the time complexity for general weights satisfying the triangle inequality.

4.1 The Construction of the t-basic Graph.

Let W be a subset of vertices in V, let us denote by $BG_t(W)$, the t-basic graph of W in $K(W)$, where $1 \leq t \leq \log n$, which is a collection of trees spanning W. The t-basic graph is constructed recursively from the *(t-1)*-basic graph using edges in a complete graph $K(W)$. The 1-basic graph, or $BG_1(W)$, is the forest of trees, F_m, obtained by our heuristic for the CFP. The main features of the t-basic graph is that it is a forest of trees and its total weight is bounded above by a certain factor times the weight of the optimal m-perfect matching (and the solution to the optimal m-subtree problem) of $K(W)$. The t-basic graph is the key construction in all of our heuristics.

First, we will relate the weight of the 1-basic graph to the weight of $M_m^*(W)$, the optimal m-perfect matching of W, and to the weight of $T_m^*(W)$, the optimal m-subtree of W. For a graph G, let $\omega(G)$ be the edge-weight of G.

Lemma 4.1.1 *The weight of the* 1*-basic graph is bounded above by* $\omega(BG_1(W)) \leq \{2\frac{m-1}{m}\omega(M_m^*(W)), \omega(BG_1(W)) \leq 2\omega(T_m^*(W))\}$.

Proof: Given $M_m^*(W)$, which is a collection of cycles each of size exactly m, after

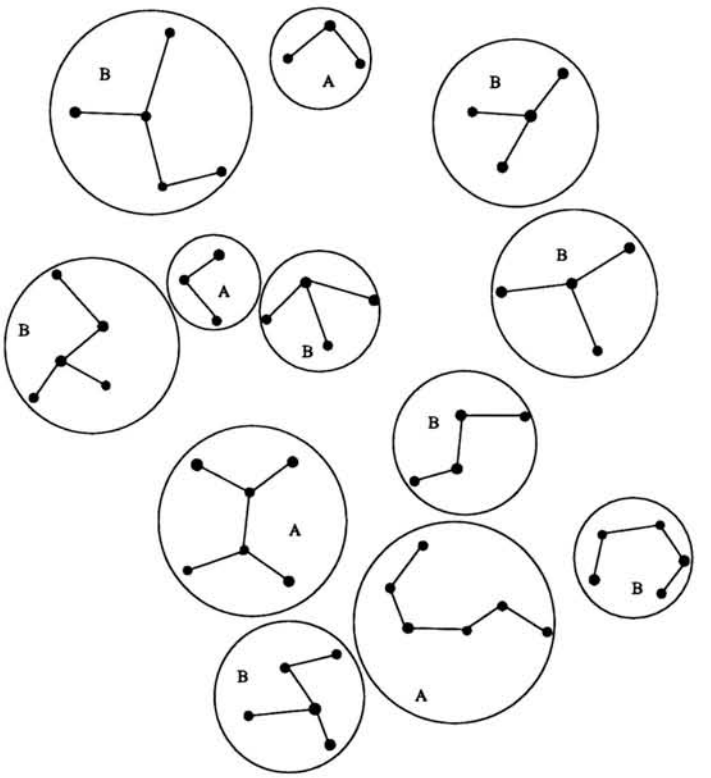


Figure 4.1: Type A and type B connected components in $BG_1(W)$, for $m = 3$.

removing the largest edge from each cycle, the resulting graph is a forest of trees, whose edge-weight is $(1-\frac{1}{m})$ times the weight of the corresponding cycles. Note that the weight of $F_m^*(W)$, the CFP of W, is bounded above by the weight of the forest. From this and by Theorem 3.2.1, it follows that $\omega(BG_1(W)) \leq 2\omega(F_m^*(W)) \leq 2\frac{m-1}{m}\omega(M_m^*(W))$.

Similarly, from $\omega(F_m^*(W)) \leq \omega(T_m^*(W))$ and by Theorem 3.2.1, it follows that $\omega(BG_1(W)) \leq 2\omega(T_m^*(W))$ □.

The 1-basic graph, see Fig.4.1, consists of two types of connected components: type A, denoted by $A_1(W)$, where each member has a multiple of m vertices, and type B, denoted by $B_1(W)$, which are all the other connected components. The components will be treated as hypervertices.

Now we will show one step of the recursive procedure, where the 2-basic graph is obtained from the 1-basic graph. To construct $BG_2(W)$ from $BG_1(W)$, for each type B hypervertex in $B_1(W)$ its nearest type B hypervertex is found. The two hypervertices are connected either by an edge or a set of edges forming a shortest path in $K(W)$,

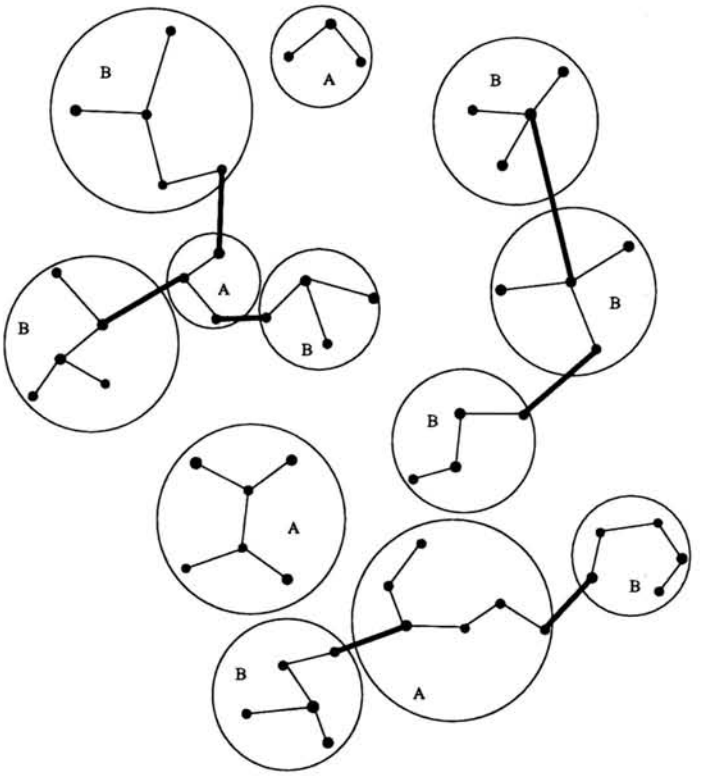


Figure 4.2: The nearest neighbor graph of type B connected components.

passing through type A hypervertices of $A_1(W)$. An auxiliary graph is constructed from which the nearest neighbor graph of type b hypervertices can be obtained. In particular any two hypervertices are connected, type A and type B, by an edge in $K(W)$, which is the shortest edge between them. The new graph, denoted by $\overline{BG_1}(W)$, can be viewed as a complete graph on all the hypervertices, of type A and type B. We find a nearest neighbor graph of type B hypervertices, where each type B hypervertex is connected to its nearest type B neighbor either by and edge in $\overline{BG_1}(W)$ or a shortest path in $\overline{BG_1}(W)$ passing through other type A hypervertices. Fig.4.2 illustrates such a nearest neighbor graph, where the thick edges represent the shortest paths. This graph becomes the 2-basic graph. The newly obtained connected components are clusters of old type A and type$B$ hypervertices. Also some type A hypervertices in $A_1(W)$ become type A hypervertices in $A_2(W)$. Thus $BG_2(W)$ contains old and new type A hypervertices, forming $A_2(W)$, and new type B hypervertices, now in $B_2(W)$, see fig.4.3.

This recursive procedure is repeated until the t-basic graph is formed. In general

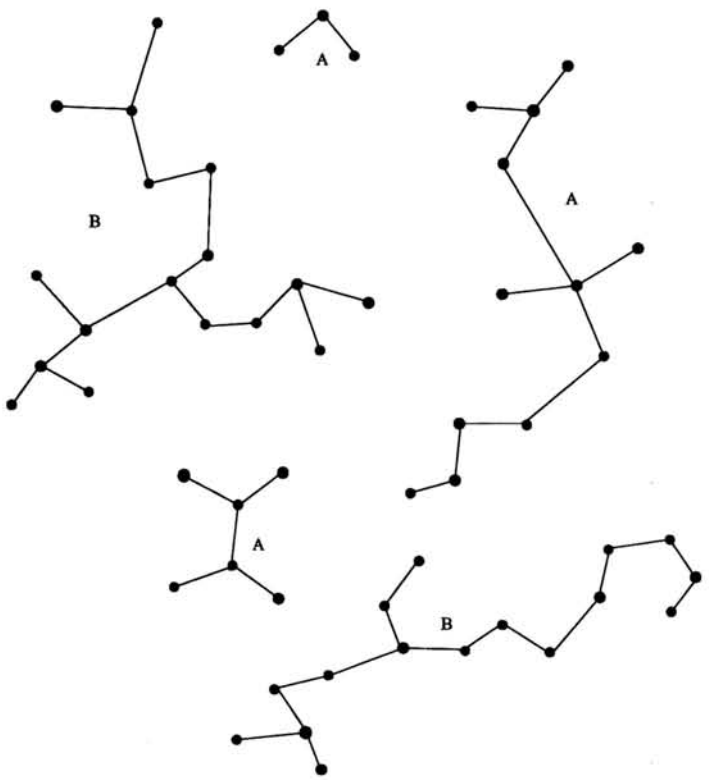


Figure 4.3: $BG_2(W)$ - a collection of old and new type A and new type Bconnected components, for $m = 3$.

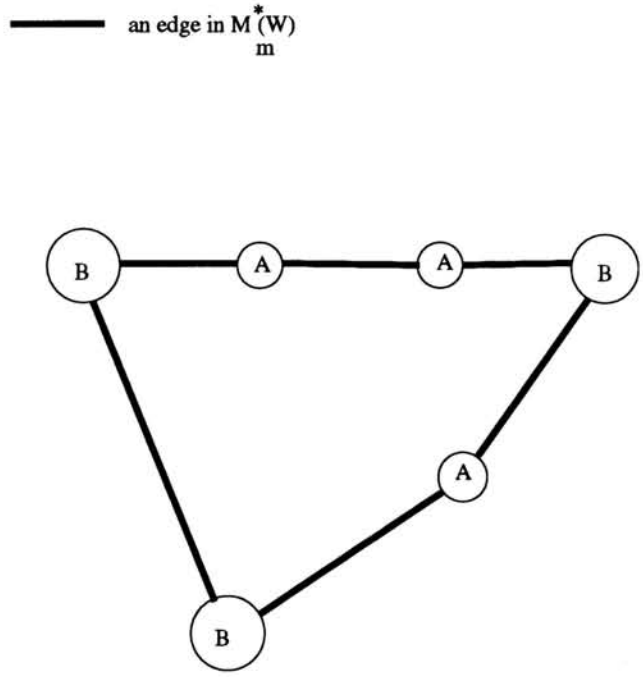


Figure 4.4: A connected component in $BG_i(W) \bigcup M_m^*(W)$.

given $BG_i(W)$, a $BG_{i+1}(W)$ is obtained by finding a nearest neighbor graph of type B connected components in $B_i(W)$, repeating the same process, and adding a set of edges in $K(W)$. The total weight of the edges is bounded in the following lemma.

Lemma 4.1.2 *For each $1 \leq i < t$ the total weight of all edges added to $BG_i(W)$, to form a nearest neighbor graph of its type B hypervertices is bounded above by $\{\omega(M_m^*(W)), 2\omega(T_m^*(W)\}$.*

Proof: First we will prove the bound with respect to $\omega(M_m^*(W))$. Let us consider, for a given i, the union of $BG_i(W)$ and $M_m^*(W)$. Some of the edges of $M_m^*(W)$ would be hidden inside the type A and type B hypervertices of $A_i(W)$ and $B_i(W)$, while the other will form connections between the components (treated as hypervertices). We observe that each type B hypervertex is connected with its nearest type B hypervertex either by an edge or a path passing through type A hypervertices, see Fig.4.4. Hence by adding edges from $M_m^*(W)$ to $BG_i(W)$, the i-basic graph is partitioned into connected components, whose nodes are type A and B hypervertices, and the connected components are themselves of type A.

Let us denote by $M_m'(W)$ the graph $BG_i(W) \bigcup M_m^*(W)$, where the type B hypervertices are nodes and the edges correspond to the chains between type B hypervertices,

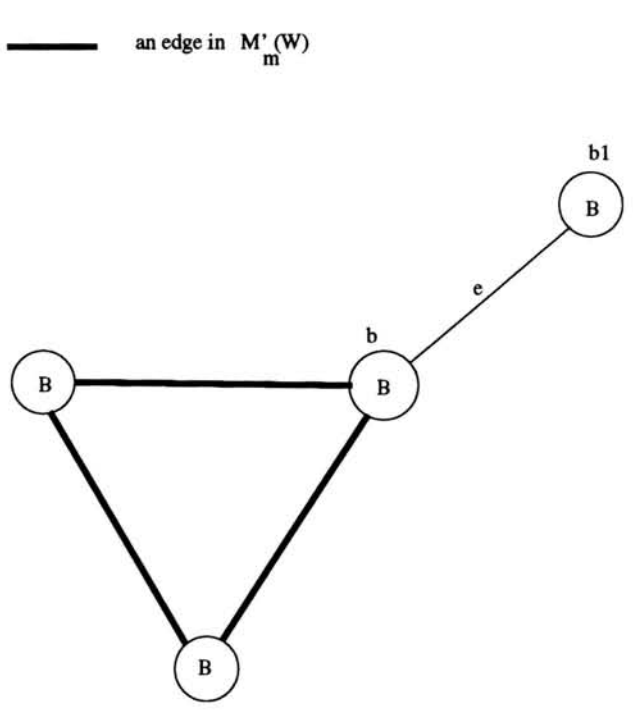


Figure 4.5: A type B hypervertex in $M'_m(W)$ and its nearest type B neighbor.

passing through type A hypervertices. The total weight of the edges in $M'_m(W)$ is bounded above by $\omega(M^*_m(W))$. Every connected component in $M'_m(W)$ is of type A and there is an even number of edges incident to each node. This comes from the fact that $M^*_m(W)$ is a collection of cycles and if a cycle in $M^*_m(W)$ is incident to a node in $BG_i(W) \bigcup M^*_m(W)$ it "enters" and "leaves" the node. Since each node in $M'_m(W)$ is incident to at least one cycle, its degree is even.

Let b be a node in $M'_m(W)$ (i.e. a type B hypervertex), there is an even number of edges incident to b in $M'_m(W)$. Without loss of generality we assume that degree of each node in $M'_m(W)$ is exactly 2, i.e. each connected component in $M'_m(W)$ is a cycle (see Fig.4.5). Let b_1, be the nearest type B neighbor of b in $\overline{BG_i}(W)$, and $e = (b, b_1)$ corresponds to a shortest path between them consisting of edges in $\overline{BG_i}(W)$. The weight of e is bounded above by the weight of the shortest edge incident to b in $M'_m(W)$, i.e. its weight is less or equal to the weight of any of the two edges incident to b. The same is true for any other node in the cycle; the weight of the shortest path between the node and its nearest type B neighbor can be bounded by the weight of any edge incident to the node in $M'_m(W)$. We can define trivially one-to-one mapping,

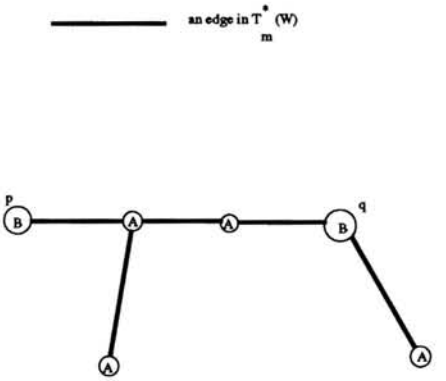


Figure 4.6: A connected component in $BG_i(W) \bigcup T_m^*(W)$.

which maps each such shortest path into an edge in $M_m'(W)$, and such that the weight of the shortest path is bounded above by the weight of its image. Hence the total weight of the nearest neighbor graph of type B hypervertices is bounded above by $\omega(M_m^*(W))$.

Similarly, we will prove the bound with respect to $\omega(T_m^*(W))$. Consider graph $BG_i(W) \bigcup T_m^*(W)$, where the edges in $T_m^*(W)$ form connections between the hypervertices of $BG_i(W)$. Each type B hypervertex, which is incident to at least one edge in $T_m^*(W)$, is connected to another type B hypervertex either by an edge or a path passing through type A hypervertices. Note that all the connected components in $BG_i(W) \bigcup T_m^*(W)$ are of type A, Fig.4.6 illustrates one of them. Let $T_m'(W)$ be the graph $BG_i(W) \bigcup T_m^*(W)$, where paths between type B hypervertices are replaced by edges whose weights are equal to those of the corresponding paths. Let p and q be nodes in $T_m'(W)$, which are connected by an edge e, corresponding to a path in $BG_i(W) \bigcup T_m^*(W)$, see Fig.4.7. Let p_1 andq_1 be the nearest type B neighbors in

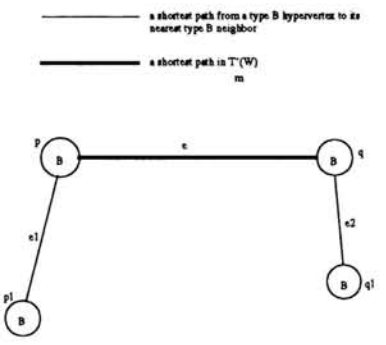


Figure 4.7: Type B hypervertices in $T'_m(W)$ and their nearest type B neighbors.

$\overline{BG_i}(W)$ of p and q, respectively, and let $e_1 = (p, p_1)$ and $e_2 = (q, q_1)$ correspond to shortest paths in $\overline{BG_i}(W)$. The weight of e_1 and e_2 is bounded above by the weight e. Hence the total weight of the nearest neighbor graph of type B hypervertices is bounded above by $2\omega(T^*_m(W))$ $\square$.

At each i-th step, when the i-basic graph is constructed, a number of edges of $K(W)$ are added, whose total weight is bounded above by $\omega(M^*_m(W))$ and $2\omega(T^*_m(W))$. This is a "penalty" which is paid, for growing the size of each type B hypervertex. We want to construct large type B hypervertices and at the same time maintain the weight of the i-basic graph as low as possible. The total weight of $BG_t(W)$ graph is bounded above by the weight of $BG_1(W)$ and the weight of all the edges added in the $(t-1)$ stages of forming the t-basic graph. Thus

Theorem 4.1.1 *The total edge weight of the t-basic graph $B_t(W)$ of $K(W)$, is bounded above by: $min\{[2\frac{(m-1)}{m} + (t-1)]\omega(M^*_m(W)), 2t\omega(T^*_m(W))\}$* $\square$.

In particular, when $t = \lfloor log_3|W| \rfloor - 1$ and $m = 2$, the *t-basic graph* is the collection of only even connected components (or type A hypervertices in general), and if additionally

$W = V$, this corresponds to the original *hypergreedy* heuristic by Plaisted, Reingold and Supowit [45], [50]. This is the case when the optimal m-subtree problem reduces to an optimal perfect matching. In the *hypergreedy*, the even connected components are converted into a perfect matching of the input graph, and the total edge weight of the matching does not exceed the weight of the t-basic graph. Thus the ratio of the weight of $BG_t(W)$ (in this case $W = V$) to the weight of $M_m^*(W)$, which is bounded above $2\lfloor log_3|W|\rfloor - 1$, is the error of the hypergreedy.

4.2 Time Complexity.

In this section, we derive a bound on the worst case time complexity for constructing the t-basic graph. The t-basic graph is obtained in t steps using edges in $K(W)$. The time complexity of construction of the $BG_1(W)$ graph is $O(|W|^2)$, for weights satisfying the triangle inequality. The time complexity of one step of the recursive procedure, when the $(i+1)$-basic graph is obtained from the i-basic graph, $1 \leq i \leq t$ can be derived as follows. The $\overline{BG_i}(W)$ auxiliary graph is constructed in $O(|W|^2)$ time, by finding for any two hypervertices of type A and type B, the shortest edge between them. We will show that although the edge weights of $\overline{BG_i}(W)$ do not satisfy the triangle inequality, the subgraph of all nearest type B neighbors selected from $\overline{BG_i}(W)$ can be constructed in $O(|W|^2)$ time, for general weights satisfying the triangle inequality. We find for every type B hypervertex in $B_i(W)$ its nearest type B neighbor, using edges in $\overline{BG_i}(W)$. Given $\overline{BG_i}(W)$, a structure is built, called the *Generalized Voronoi Diagram (GVD)* which is used for constructing the nearest neighbor graph. The *GVD* for graph $\overline{BG_i}(W)$ relative to the set of type B hypervertices, defined similarly as in [50], is a partition of all hypervertices in $\overline{BG_i}(W)$ with respect to which type B hypervertex they are closest to (each type B hypervertex is closest to itself), see Fig.4.8. The *Generalized Voronoi Region (GVR)* associated with each type B hypervertex b is a tree with the root at b. The remaining nodes in the tree are those hypervertices of type A which are closer to b than to any other type B hypervertex. At each node, a distance which is the length of a shortest path from the node to b in $\overline{BG_i}(W)$ is stored. Two *GVR*'s are *adjacent* if there is an edge in $\overline{BG_i}(W)$ with endpoints in the two regions, and obviously for

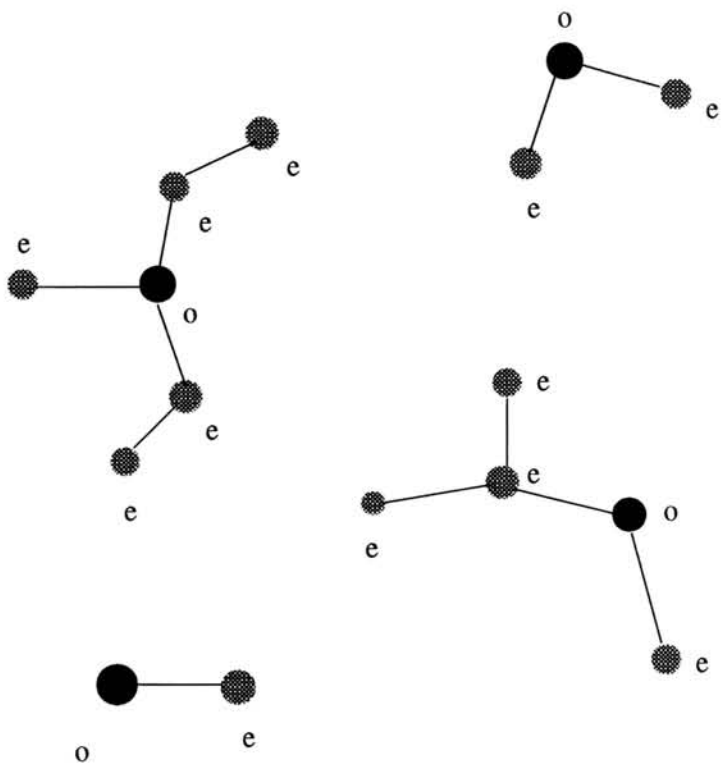


Figure 4.8: Generalized Voronoi Diagram.

$K(W)$, each two GVR's are adjacent.

Before the worst-case the time complexity of the construction of the t-basic graph will be derived, we will consider the problem of constructing the nearest neighbor graph of type B hypervertices.

Theorem 4.2.1 *Computing the nearest neighbor graph of type B hypervertices in* $\overline{BG_i}(W)$, *for general weights, can be done in* $(|W|^2)$ *time.*

The proof of the above theorem follows from the next two lemmas.

Lemma 4.2.1 *Given* $\overline{BG_i}(W)$, *the Generalized Voronoi Diagram for graph* $\overline{BG_i}(W)$, *with respect to the set of type B hypervertices can be constructed in* $O(|W|^2)$ *time, for general weights.*

Proof: Graph $\overline{BG_i}(W)$ is a complete graph whose vertices are type A and B hypervertices. We find, for each type A hypervertex, the shortest distance to a type B hypervertex, and also the name of the predecessor in the shortest path, using a modified

Dijkstra's algorithm.

Initially all the type A hypervertices are in the set $UNMARKED$, and all the type B hypervertices in the set $MARKED$. We will use a modified Dijkstra's shortest path algorithm, see [44], which finds for every hypervertex a in $UNMARKED$ its nearest hypervertex b in $MARKED$, such that they are connected by an edge in $\overline{BG_i}(W)$. In order to do this all the edges in $\overline{BG_i}(W)$ have to be examined.

At any stage there is a set $MARKED$ of type A and type B hypervertices, set $UNMARKED$ of type A hypervertices, and labels $p(a)$ for all type A hypervertices, such that

$p(a)$ = *shortest distance of any path from a to a type B hypervertex*

using intermediate type A hypervertices in MARKED, if there are any

The hypervertex $a \in UNMARKED$ with the smallest label $p(a)$ is selected. The shortest path from a type B hypervertex to this a uses only type A hypervertices in $MARKED$ as intermediate vertices. Otherwise the $p(a)$ would not be the smallest.

The hypervertex a is moved from $UNMARKED$ to $MARKED$ and the labels $p(v)$ are updated for all $v \in UNMARKED$, as follows:

$$p(v) = min\{p(v), p(a) + \omega(v, a)\}$$

This means that the new label of each hypervertex $a \in UNMARKED$ is either the old label or it is the sum of the shortest distance from type B hypervertex to a, which corresponds to the weight of the shortest path passing through type A hypervertices in $MARKED$, and the weight of the edge (v, a). The algorithm stops when $UNMARKED = \emptyset$, and each type A hypervertex $a \in MARKED$ has its label $p(a)$ which is the weight of the shortest path from a to its nearest type B hypervertex. The path can be reconstructed by keeping track at each node of where its label comes from. The resulting graph can be viewed as a forest of trees rooted at type B hypervertices with shortest paths to all type A hypervertices. More formally,

Modified Dijkstra's Algorithm for GVD for complete graphs

Input: A graph $\overline{BG_i}(W)$ with nonnegative weights and two sets of hypervertices, type A and B.

Output: The shortest path from each type A hypervertex to its nearest type B hypervertex.

begin

$UNMARKED = \{$ *all type A hypervertices*$\}$;

$MARKED = \{$ *all type B hypervertices*$\}$;

for all $v \in UNMARKED$ **do** $p(v) = \omega(v, b)$, where $b \in MARKED$;

while $UNMARKED \neq \emptyset$

begin

find $min\ \{p(v); v \in UNMARKED\}$, say $p(a)$;

$MARKED = MARKED \bigcup \{a\}$;

$UNMARKED = UNMARKED \setminus \{a\}$;

for all $v \in UNMARKED$ **do**

$p(v) = min\ \{p(v), p(a) + \omega(v, a)\}$;

end

end

There are $O(|W|)$ iterations of the algorithm, where each iteration is proportional to the number of hypervertices in *UNMARKED*. Thus the time complexity of constructing GVD of $\overline{BG_i}(W)$ is $O(|W|^2)$. □

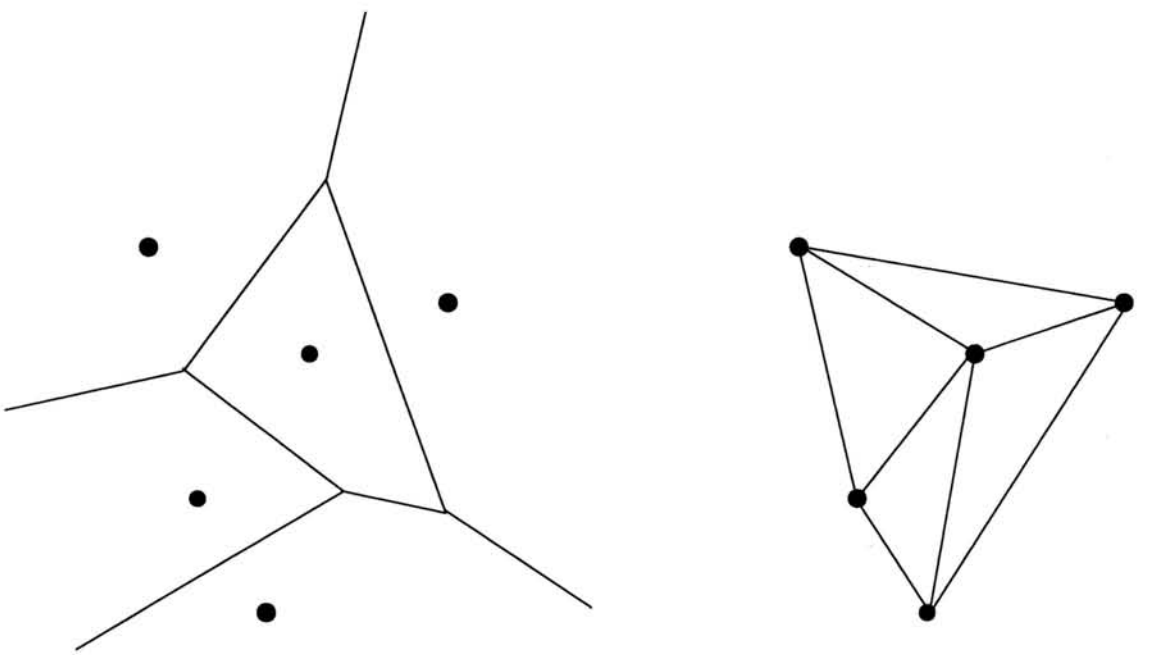
The following lemma was proved in [45]. We replace in the original lemma "odd hypervertex" by "type B hypervertex".

Lemma 4.2.2 *Given a type B hypervertex y and GVR(y), the GVR containing y, assume z is nearest type B neighbor of y. Then, there exits an edge (u,v), such that shortest path from y to z consists of the path from y to u, $u \in GVR(y)$, the edge (u,v), $v \in GVR(z)$, and the path from V to z.*

By the above lemma, to determine for each type B hypervertex its nearest type B neighbor, all the edges in $\overline{BG_i}(W)$ have to be examined, and retained those with endpoints in different GVR's. Then for each pair of GVR's such an edge which minimizes the length of the shortest path between the corresponding type B hypervertices is selected. This takes $O(|W|^2)$ operations.

Lemma 4.2.3 *Given $K(W)$, the construction of the t-basic graph $BG_t(W)$ requires, for weights satisfying the triangle inequality, $O(t|W|^2)$ operations.*

Proof: The graph $\overline{BG_i}(W)$ is constructed from the *i-basic graph* by joining each its two hypervertices by the shortest edge in $K(W)$. This requires $O(|W|^2)$ time. By Lemma 4.2.2, finding the nearest neighbor graph of type B hypervertices requires $O(|W|^2)$ operations. The resulting new type A and type B hypervertices and old type A hypervertices form the $(i+1)$-basic graph, $BG_{i+1}(W)$. Therefore constructing the $(i+1)$-basic graph from the i-basic graph takes $O(|W|^2)$ time. The parameter t is the depth of the recursion, thus the total work for constructing $BG_t(W)$ is $O(t|W|^2)$ □.



(a) (b)

Figure 4.9: (a) Voronoi diagram, (b) Delaunay triangulation.

Chapter 5

An Approximate t-basic Graph for Euclidean Points in the Plane.

In this section we will describe the construction of the approximate t-basic graph for the Euclidean points in the plane and analyze its time complexity. The weight of the approximate t-basic graph will be related to the weight of solution of the the optimal m-perfect matching and the optimal m-subtree problem, respectively. By using the approximate t-basic graph in all of the three classes of heuristics, we achieve a dramatic reduction in their respective time complexities, while their error bounds are affected only by the constant factor of $\alpha = 2.42$.

5.1 The construction of an approximate t-basic graph.

Suppose V is a set of points in the Euclidean plane. In this case all three classes of heuristics build an approximate t-basic graph, denoted by $\widehat{BG}_t(W)$, where $W \subseteq V$, which is a subgraph of the *Delaunay triangulation* [46]. For $i = 1$, the subgraph $\widehat{BG}_t(W)$ coincides with the forest of trees, where each tree has size at least m, constructed by the Euclidean heuristic for CFP. For $1 \leq i < t$, to obtain $\widehat{BG}_{i+1}(W)$ from $\widehat{BG}_i(W)$, the nearest neighbor graph of the type B hypervertices with respect to the Delaunay triangulation of V is constructed.

The Delaunay triangulation, which is a sparse graph with $O(|W|)$ edges, is obtained from the *Voronoi diagram*, see Fig.4.9(a). The Voronoi diagram is a partition of the plane into *Voronoi regions* of points closer to one point of W than to others, see Preparata and Shamos [46]. The Voronoi region of a point $v \in W$ is a set of points in the plane closer to v than to any other point $u \in W$, $v \neq u$. Two Voronoi regions are *adjacent* if they share a Voronoi edge. The Delaunay triangulation of W can be derived

from the corresponding Voronoi Diagram by joining each two points whose Voronoi regions are adjacent, see Fig.4.9(b), in $O(|W|)$ time. Chew [9] and Dobkin and Friedman and Supowit [16] showed that complete Euclidean graphs can be approximated by Delaunay graphs. The best bound on this approximation is due to Keil and Gutwin in [37], by proving that the Delaunay triangulation has the property that the weight of the shortest path between each pair of vertices is bounded above by $\alpha = 2.42$ times the Euclidean distance between the two vertices.

Consider a type B hypervertex b. Let b_1 be the nearest type B hypervertex of b with respect to $K(W)$, and let $(v_1, v_2), (v_3, v_4)..., (v_{k-1}, v_k)$ be the edges of this shortest path. From the above, for each edge (v_i, v_{i+1}), there exists a path in the Delaunay triangulation whose weight is bounded above by α times the straight line distance between v_i and v_{i+1}. Thus

Lemma 5.1.1 *The weight of the shortest path from a type B hypervertex to its nearest type B hypervertex with respect to the Delaunay triangulation, is bounded above by α times the weight of the shortest path from that type B hypervertex to its nearest type B hypervertex with respect to K(V).* □

From Theorem 4.1.1 and the above lemma, it follows, for points in the Euclidean plane, that

Corollary 5.1.1 *The weight of edges of $\widehat{BG}_t(W)$ is bounded above by $[2\frac{m-1}{m} + \alpha(t-1)]\omega(M^*_m(W))$ and $2[1+\alpha(t-1)]\omega(T^*_m(W))$* □.

5.2 Time complexity.

In this section we will analyze the time complexity of an approximate t-basic graph, $\widehat{BG}_t(W)$, for points in the Euclidean plane.

Given a set of W points in the Euclidean plane, the $\widehat{BG}_t(W)$ which coincides with $BG_1(W)$ is obtained in $O(|W|log|W|)$ time, as follows. First the Voronoi diagram of W is constructed in $O(|W|log|W|)$ time, see [46], then the Delaunay triangulation is derived from the corresponding Voronoi diagram in $O(|W|)$ time. The edges in the

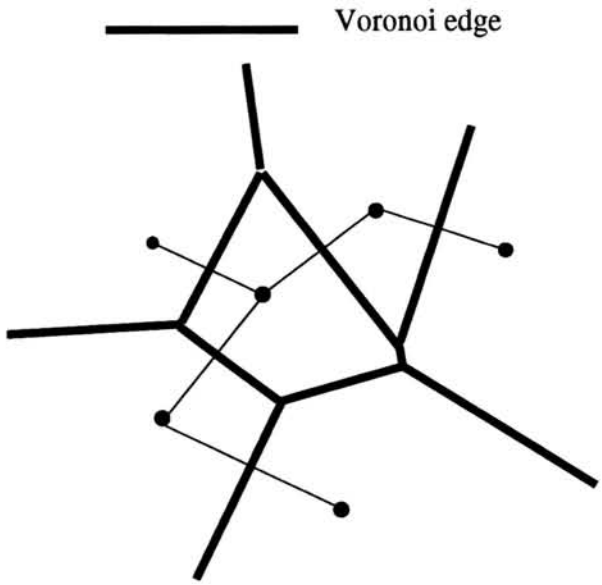


Figure 5.1: Voronoi region of a hypervertex (VRH).

Delaunay triangulation are used to obtain $\widehat{BG}_t(W)$. The $\widehat{BG}_t(W)$ graph contains type A and type B hypervertices.

Given $\widehat{BG}_t(W)$ and the edges in the Delaunay triangulation, the $\widehat{BG}_2(W)$ is obtained in $O(|W|log|W|)$ time. The Generalized Voronoi Diagram (GVD) relative to the set of the type B hypervertices is built using edges in the Delaunay triangulation. In the GVD, each type A hypervertex is connected to its nearest type B hypervertex by a shortest path, possibly passing through other type A hypervertices. A GVD is a collection of Generalized Voronoi Regions (GVR's), and each GVR associated with a type B hypervertex, say b, is a tree rooted at b, where the remaining nodes are type A hypervertices and the edges come from the Delaunay triangulation. At each type A node, the shortest distance from the node to the root (its nearest type B neighbor) is stored. For the Euclidean case in the plane, two GVR's are *adjacent* if there is an edge in the Delaunay triangulation with endpoints in the two GVR's.

Before the time complexity of constructing of the nearest neighbor graph of the type B hypervertices is analyzed, we will consider the following problem. Let $G = (V, E)$ be a sparse edge-weighted graph with $|V| = n$ vertices and $|E| = m$ edges and whose weights are nonnegative. Suppose that V is partitioned into $\mathcal{A}$ and $\mathcal{B}$ labeled as type A and type B, respectively. The GVD is defined as before, i.e as clusters of type A

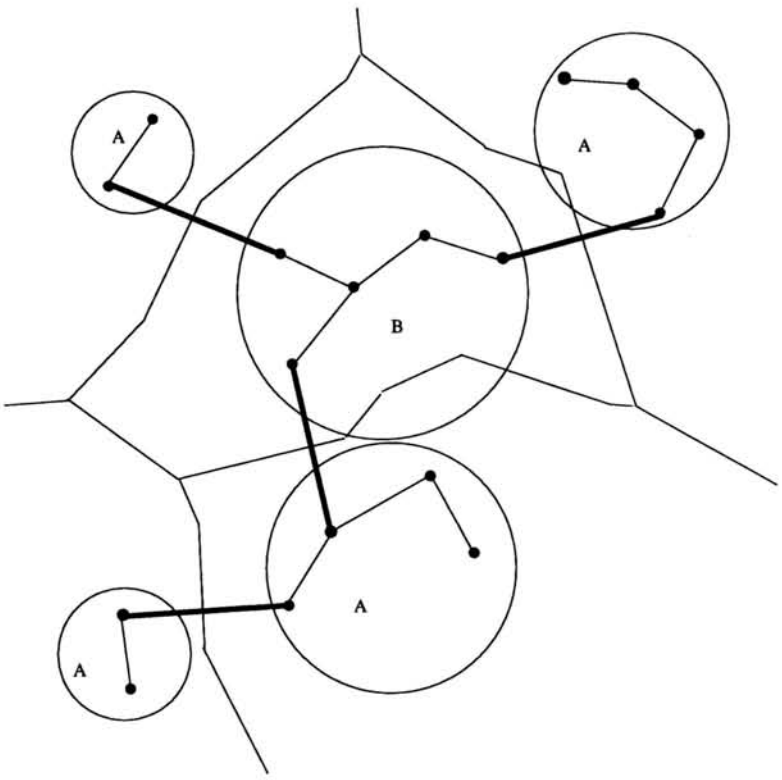


Figure 5.2: Representation of a GVR, as a cluster of VRH's.

vertices connected to their nearest type B vertices by the corresponding shortest paths, then

Lemma 5.2.1 *The GVD of $G = (V, E)$ can be constructed in $O(m \log n\})$ time.*

Proof: Let $\mathcal{A}$ and $\mathcal{B}$ contain initially the type A and type B vertices, respectively. Consider the following modified Dijkstra's shortest path algorithm (see [44], [45]).

When the algorithm proceeds, some of the type A vertices would be moved from $\mathcal{A}$ to $\mathcal{B}$. First let us examine edges between $\mathcal{A}$ and $\mathcal{B}$. For each type A vertex $q \in \mathcal{A}$ its nearest type B neighbor $o \in \mathcal{B}$ is found, if there is an edge $(q, o) \in E$, and a label $p(q) = \omega(q, o)$ is assigned to it, i.e. the weight of the edge. All the remaining type A vertices in $q \in \mathcal{A}$ will have labels $p(q) = \infty$.

At any stage there will be labels $p(q)$ for all type A vertices in $\mathcal{A}$, such that

$p(q)$= shortest distance of any path from q to an type B vertex

using intermediate even vertices in $\mathcal{B}$.

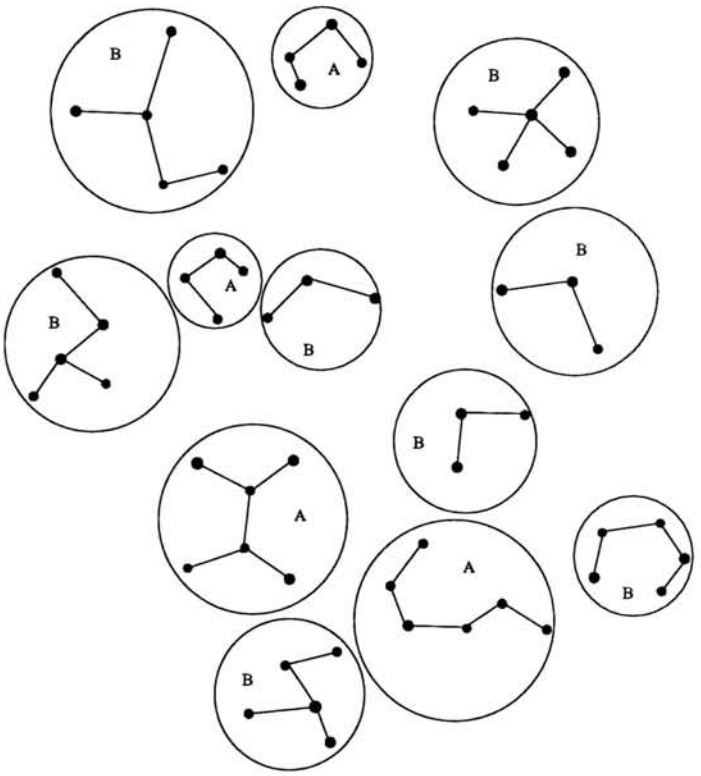


Figure 5.3: The 1-basic graph for Euclidean points in the plane ($m = 2$).

Now select the vertex $q \in A$ with the smallest label $p(q)$ is selected. The shortest path from a type B vertex to this q uses only type A vertices in $\mathcal{B}$ as intermediate vertices. Otherwise the $p(q)$ would not be the smallest.

The vertex q is moved from $\mathcal{A}$ to $\mathcal{B}$ and the labels $p(v)$ for all $v \in \mathcal{A}$ are updated for which there is an edge $(v, q) \in E$ as follows:

$p(v) = min\{p(v), p(q) + \omega(v, q)\}$ This means that the new label of each vertex $v \in \mathcal{A}$ is either the old label or it is the sum of the shortest distance from a type B vertex to q through type A vertices in $\mathcal{B}$ and the weight of the edge (v, q). The algorithm stops when $\mathcal{A} = \emptyset$, and each type A vertex $q \in \mathcal{B}$ has its label $p(q)$ which is the weight of the shortest path from q to its nearest type B vertex. The path can be reconstructed by tracing the labels in the path. The resulting graph can be viewed as a forest of trees rooted at type B vertices with shortest path to all type A vertices, More formally,

Modified Dijkstra's Algorithm for GVD

Input: A sparse graph $G = (V, E)$ with nonnegative edge weights and with V partitioned into two subsets $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$= { vertices labeled as type$A$ } and $\mathcal{B}$= { vertices labeled as type B }. Initially each vertex $v \in \mathcal{A}$ has a label $p(v) = \infty$.

Output: The weight shortest path from each type A vertex v to its nearest type B vertex, possibly passing through intermediate type A vertices, stored in $p(v)$. Also each type A vertex will store the next vertex in its shortest path.

begin

for all $v \in \mathcal{A}$ such that there is an edge $(v, o) \in E$, where $o \in \mathcal{B}$ **do**

$p(v) = min\{\omega(v, o) : o \in \mathcal{B}$ *and* $(v, o) \in E\}$;

while $\mathcal{A} \neq \emptyset$;

begin find $min\{p(v); v \in \mathcal{A}\}$, say $p(q)$;

$\mathcal{B}=\mathcal{B}\bigcup\{q\}$;

$\mathcal{A}=\mathcal{A}\backslash\{q\}$;

for all $v \in \mathcal{A}$ s.t. $(v, q) \in E$ **do**

$p(v) = min\{p(v), p(q) + \omega(v, q)\}$

end

end

For $G = (V, E)$, which is sparse, all the labels $p(v)$ can be stored on a heap to maintain a priority queue which returns and removes the label with the smallest value, see [12]. The size of the heap is $O(n)$ and it can be constructed it in $O(n)$ time. The algorithm will require m operations on the priority queue where each operation takes $O(\log n)$ time. Thus the time complexity is $O(m \log n)$. □

By Lemma 4.2.2, given GVD the nearest type B neighbor for each type B hypervertex in $\widehat{BG}_t(W)$ with respect to the Delaunay triangulation can be determined in $O(|W| \log |W|)$ operations. All the edges in the Delaunay triangulation are examined and retained only those with endpoints in different GVR's. For each pair of adjacent GVR's such an edge is selected which minimizes the length of the shortest path between the corresponding type B hypervertices, with respect to the Delaunay triangulation. Given GVD, this can be done in $O(n)$ operations.

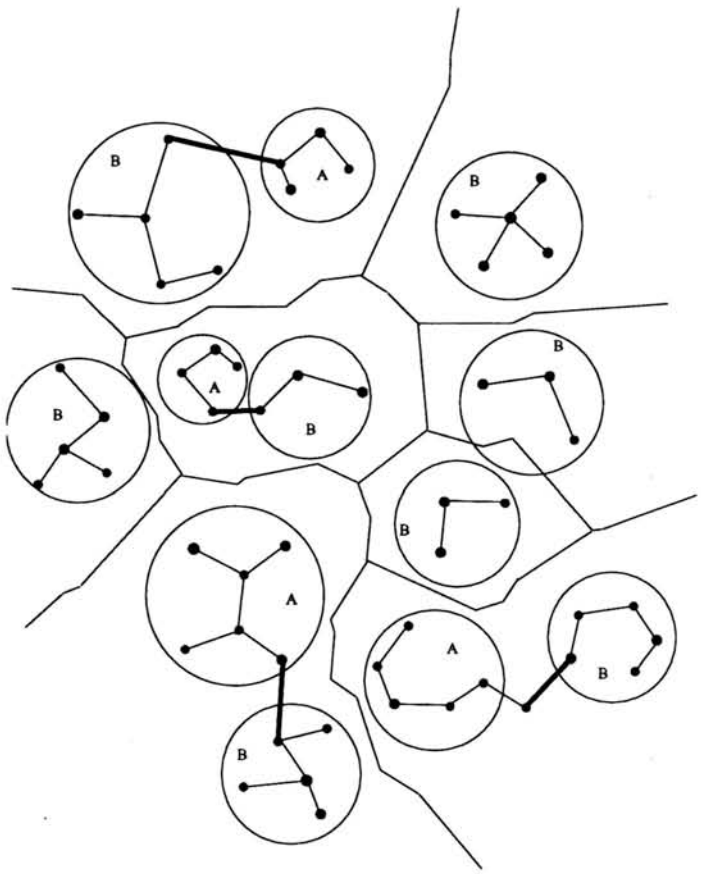


Figure 5.4: GVD with GVR's separated by their Voronoi diagram boundaries.

Each vertex from the set W defines its Voronoi region of points closer to it than to any other vertex in W. Each hypervertex, type A and type B, of $\widehat{BG}_1(W)$, is represented by a cluster of Voronoi regions of all the vertices in the hypervertex. Let us denote this representation by the *Voronoi region of a hypervertex*, (*VRH*), see Fig.5.1.

Two *VRH*'s are *adjacent* if at least one pair of their constituent Voronoi regions, each in different *VHR*, is adjacent. Each *GVR* is represented as a cluster of *VRH*'s of all the hypervertices in the *GVR*, see Fig.5.2. Two *GVR*'s are *adjacent* if at least one pair of their constituent *VRH*'s, each in different *GVR*, is adjacent. Given the 1-basic graph, see Fig.5.3, and the above representation of its *GVD*, see Fig.5.4, to find for each type B hypervertex its nearest type B hypervertex with respect to the Delaunay triangulation, all the edges in the Delaunay triangulation have to be examined. Only those edges with endpoints in different (adjacent) GVR's are retained. Then for each pair of adjacent GVR's such an edge is selected which minimizes the length of the shortest path between the corresponding type B hypervertices. Fig.5.5 illustrates such nearest neighbor graph of type B hypervertices. Thus

Theorem 5.2.1 *The nearest neighbor graph of type B hypervertices can be computed in and in* $O(|W|\log|W|)$ *time for Euclidean points in the plane with respect to the Delaunay triangulation.* □

To construct $\widehat{BG}_i(V)$ from $\widehat{BG}_{i-1}(V)$, for every type B hypervertex its nearest type B hypervertex has to be found. By Theorem 5.2.1, this requires and $O(|W|\log|W|)$ time for Euclidean points in the plane. Thus

Corollary 5.2.1 *The time complexity of constructing* $\widehat{BG}_t(V)$ *is* $O(t|W|\log|W|)$ *for Euclidean points in the plane.*

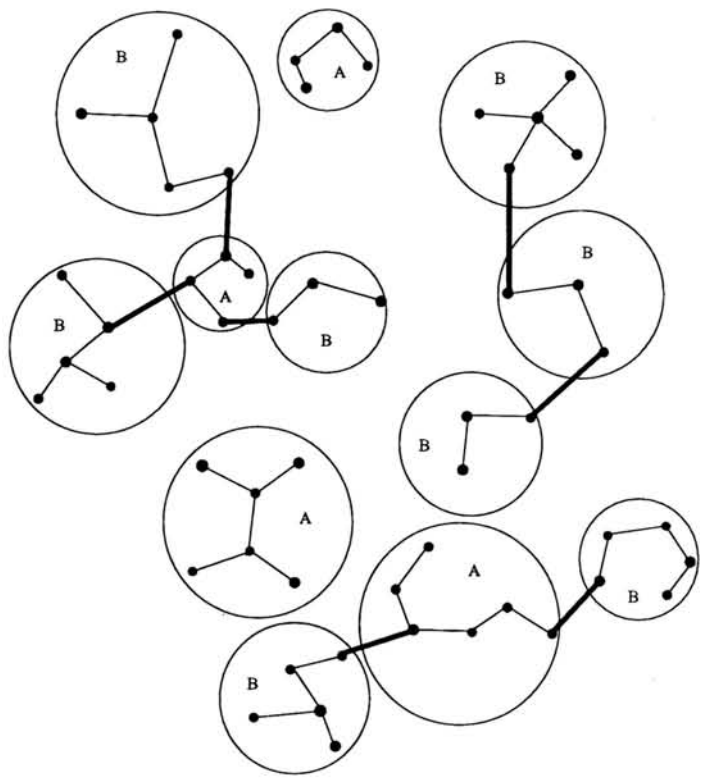


Figure 5.5: A nearest neighbor graph of type B hypervertices.

Chapter 6

The Generalized Hypergreedy.

In the following three sections we will describe the Generalized Hypergreedy heuristic and analyze its error and time complexity.

6.1 Description of the Generalized Hypergreedy.

The *Generalized Hypergreedy* heuristic for the m-perfect matching and the m-subtree problem is a combination of the heuristic for the CFP and an algorithm based on the *hypergreedy* heuristic for perfect matching. Given $K(V)$, a complete edge-weighted graph, whose weights satisfy the triangle inequality, and $n = |V|$ is a multiple of a given natural number m, we construct the t-basic graph (the approximate t-basic graph for the Euclidean case), where the 1-basic graph is obtained by the CFP heuristic, for $t = \lceil log(\frac{n}{m}) \rceil$, see in Fig.6.1 such t-basic graph. Each connected component in the t-basic graph (approximate t-basic graph) of $K(V)$, is a tree with a multiple of m vertices, i.e. it admits an m-perfect matching and a solution to the m-subtree problem. Each such connected component is processed separately: its edges are duplicated, which results in an Eulerian graph, from which an Euler tour is extracted; then using the triangle inequality, the tour is converted into a Hamiltonian cycle, whose weight does not exceed that of the tour, see Fig.6.2. The maximum-cardinality minimum-weight m-chain cover is found in the cycle, which covers all its vertices. The union of all such covers from all the Hamiltonian cycles forms a solution to the m-subtree problem, see Fig.6.3. To obtain a solution to the m-perfect matching problem, each m-chain in the m-subtree problem solution is converted into a cycle, by adding a closing edge, whose weight is bounded above by that of the chain, see Fig.6.4. The Generalized Hypergreedy is a special case of the (t, k)-heuristic. Because the Generalized Hypergreedy is a simple

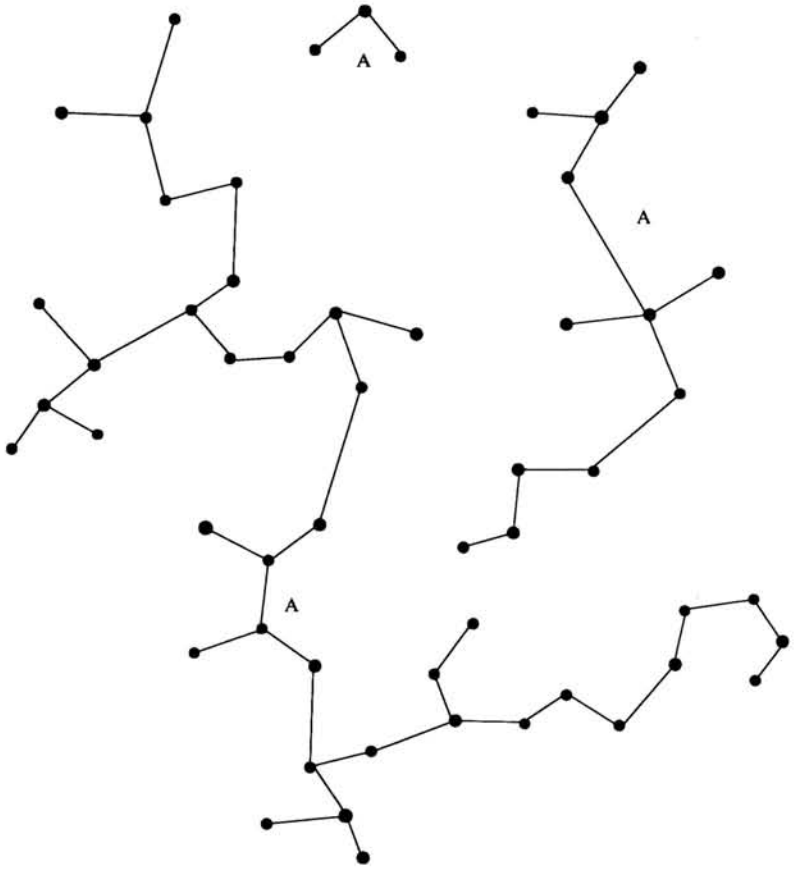


Figure 6.1: The t-basic graph with all the type A connected components, $m = 3$.

(t, k)-heuristic, it can be used in the future for developing new heuristics, in conjunction with other techniques, different from the ones used in the (t, k)-heuristic.

The Generalized Hypergreedy

Input: $K(V)$ (V, set of vertices in the Euclidean plane)

Output: m-perfect matching of $K(V)$ (a solution to the m-subtree problem of $K(V)$)

1. Construct the t-basic graph for $t = \lceil log(\frac{n}{m}) \rceil$;

(Construct an approximate t-basic graph for $t = \lceil log(\frac{n}{M}) \rceil$)

2. In every connected component of $BG_t(V)$:

- duplicate edges
- extract an Euler tour
- convert the tour into a Hamiltonian cycle

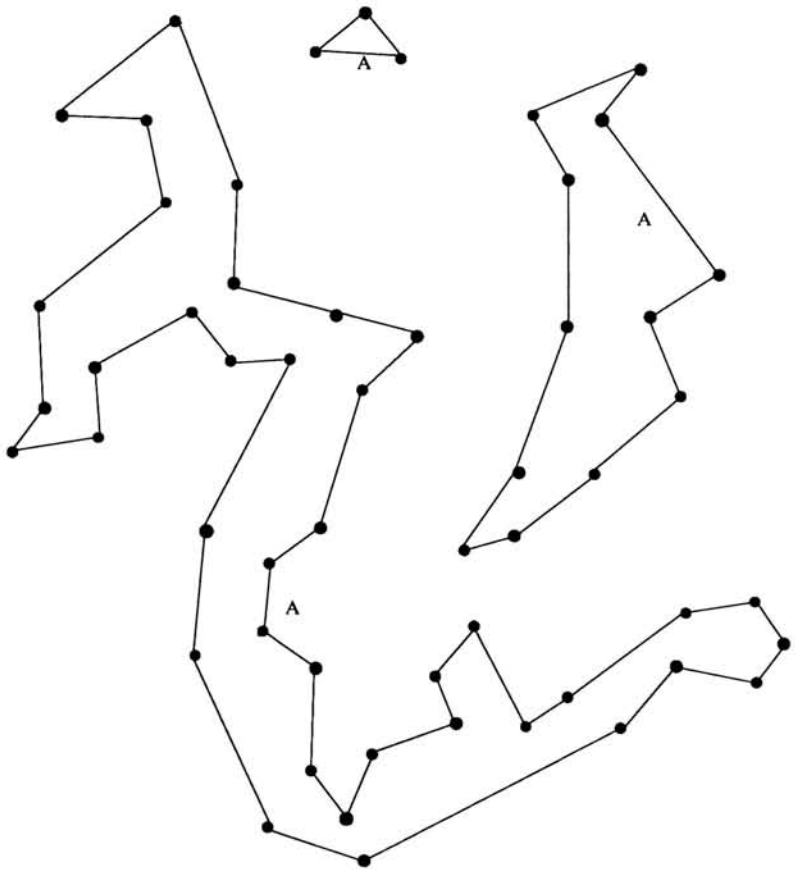


Figure 6.2: Hamiltonian cycles extracted from the duplicated t-basic graph.

- find the maximum-cardinality minimum-weight m-chain cover
- **(only for m-perfect matching)** convert each m-chain into an m-cycle

6.2 Error.

In the following two lemmas we will prove the error bounds for the m-perfect matching and the m-subtree problem.

Lemma 6.2.1 *The error of the Generalized Hypergreedy, which finds an m-perfect matching, in a complete edge-weighted graph $K(V)$, for $3 \leq m \leq \frac{n}{2}$, is bounded above by*

$$f(n) = 4\frac{m-1}{m}(2\frac{m-1}{m} + \lceil \log(\frac{n}{m}) \rceil - 1)$$

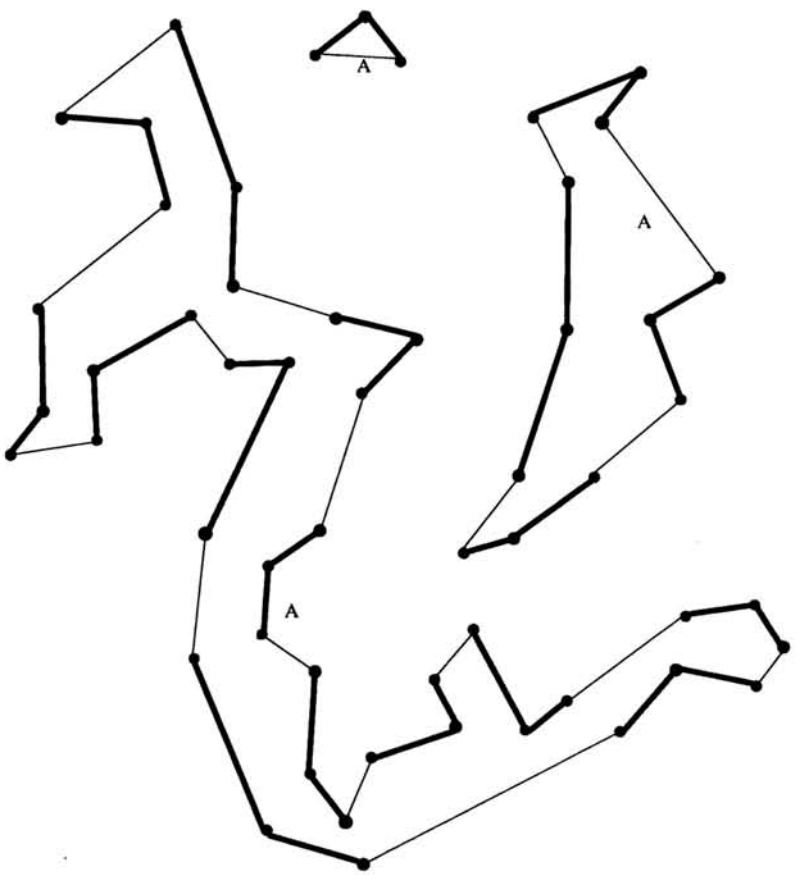


Figure 6.3: The maximum-cardinality minimum-weight m-chain cover - a solution to the m-subtree problem, $m = 3$.

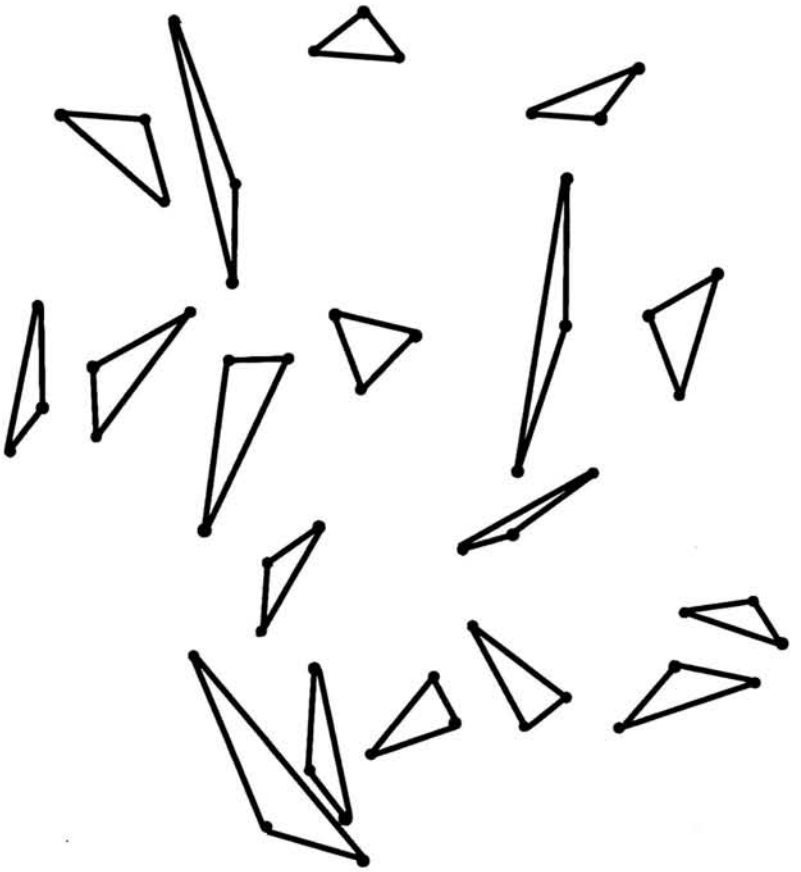


Figure 6.4: A solution to the m-perfect matching, $m = 3$

for general weights satisfying the triangle inequality, and by

$$f(n) = 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]$$

for Euclidean points in the plane.

Proof: From Theorem 4.1.1 and Corollary 5.1.1, for $t = \lceil log(\frac{n}{m}) \rceil$, the weight of the t-basic graph and the approximate t-basic graph is bounded above by $(2\frac{m-1}{m} + \lceil \log(\frac{n}{m}) \rceil - 1)\omega(M_m^*(V))$ and $[2\frac{m-1}{m} + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]\omega(M_m^*(V))$, respectively. The edges of the t-basic graph (of the approximate t-basic graph) are duplicated, i.e. the weight of the graph is doubled. An Euler tour is extracted from each connected component in the resulting graph, and the weight of the tour is bounded above by the weight of the component. Each Euler tour is converted into a Hamiltonian cycle, and by the triangle inequality, the weight the cycle obtained from an Euler tour is bounded above by the weight of the tour. The maximum-cardinality minimum-weight m-chain cover is selected in the cycle, and the weight of the cover does not exceed $\frac{m-1}{m}$ times the weight of the cycle. A closing edge is added to each m-chain cover, and the weight of the edge is bounded above by the weight of the m-chain, to form an m-cycle. The union of all such -cycles obtained from all the covers, which has weight twice the weight of the Hamiltonians, form an m-perfect matching of V. Thus the total weight of an m-perfect matching obtained by the Generalized Hypergreedy, is bounded above by $4\frac{m-1}{m}(2\frac{m-1}{m} + \lceil \log(\frac{n}{m}) \rceil - 1)\omega(M_m^*(V))$ and $4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]\omega(M_m^*(V))$ for general weights satisfying the triangle inequality and for Euclidean points in the plane, respectively.□

For $m = n$ this heuristic reduces to the minimum spanning tree 2-approximate heuristic for traveling salesman problem. For large m the error of the heuristic has a constant bound. For weights satisfying the triangle inequality, for $m = \frac{n}{2}$ the error is $f(n) = 8$, for $m = \frac{n}{3}, \frac{n}{4}$ the error is $f(n) = 12$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 16$.

For Euclidean points in the plane, for $m = n$, this heuristic reduces to the 2-approximate minimum spanning tree heuristic for the Euclidean traveling salesman

problem. For $m = \frac{n}{2}$, the error is $f(n) = 8$, for $m = \frac{n}{3}, \frac{n}{4}$, the error is $f(n) = 4(2+\alpha)$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 8(1+\alpha)$.

Lemma 6.2.2 *The error of the Generalized Hypergreedy, which finds a solution to the m-subtree problem, in a complete edge-weighted graph $K(V)$, for $3 \leq m \leq \frac{n}{2}$, is bounded above by*

$$f(n) = 4\frac{m-1}{m}(\lceil \log(\frac{n}{m}) \rceil)$$

for general weights satisfying the triangle inequality, and by

$$f(n) = 4\frac{m-1}{m}[1 + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]$$

for Euclidean points in the plane.

Proof: From Theorem 4.1.1 and Corollary 5.1.1, for $t = \lceil log(\frac{n}{m}) \rceil$, the weight of the t-basic graph and the approximate t-basic graph is bounded above by $2(\lceil \log(\frac{n}{m}) \rceil)\omega(T_m^*(V))$ and $2[1 + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]\omega(T_m^*(V))$, respectively. The weight of the Hamiltonians extracted from the duplicated t-basic graph (the approximate t-basic graph) is bounded twice the weight of the graph. From each Hamiltonian cycle select the maximum-cardinality minimum-weight m-chain cover in the cycle, and the weight of the cover does not exceed $\frac{m-1}{m}$ times the weight of the cycle. The union of the cover forms a solution to the m-subtree problem, whose weight is bounded above by $4(\lceil \log(\frac{n}{m}) \rceil)\omega(T_m^*(V))$ and $4[1 + \alpha(\lceil \log(\frac{n}{m}) \rceil - 1)]\omega(T_m^*(V))$. □

For general weights satisfying the triangle inequality, for $m = n$, the the heuristic reduces to the exact minimum spanning tree algorithm, since the optimal m-subtree problem becomes the minimum spanning tree problem. For $m = \frac{n}{2}$ the error is $f(n) = 4$, for $m = \frac{n}{3}, \frac{n}{4}$ the error is $f(n) = 8$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 12$. For $m = 2$, the Generalized Hypergreedy for the m-subtree problem, becomes the hypergreedy heuristic for perfect matching.

For Euclidean points in the plane, for $m = n$, this heuristic reduces to the exact Euclidean minimum spanning tree algorithm. For $m = \frac{m}{2}$ the error of the heuristic is $f(n) = 4$, for $m = \frac{n}{3}, \frac{n}{4}$ the error is $f(n) = 4(1+\alpha)$, and for $m = \frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$ the error is $f(n) = 4(1+2\alpha)$.

6.3 Time complexity.

Theorem 6.3.1 *The time complexity of Generalized Hypergreedy is*

$$T(n) = O(n^2 log(\frac{n}{m}))$$

for weights satisfying the triangle inequality, and

$$T(n) = O(nlog(\frac{n}{m})logn)$$

for the Euclidean points in the plane

Proof: From Lemma 4.2.3 and Corollary 5.2.1, the time for $t = \lceil log(\frac{n}{m}) \rceil$, the time complexity of constructing the t-basic graph and the approximate t-basic graph is $O(n^2 log(\frac{n}{m}))$ and $O(nlog(\frac{n}{m})logn)$, respectively. All the other steps of the algorithm can be implemented in additional $O(n)$ time. □

For large m, the Generalized hypergreedy becomes a constant-error heuristic for the m-perfect matching and for the m-subtree problem, running in $O(n^2)$ and $O(n \log n)$ time for general weights satisfying the triangle inequality and for Euclidean points in the plane, respectively.

Chapter 7

The *(t,k)*-heuristic.

In the following sections we will describe the *(t,k)*-heuristic, derive its error with respect to the optimal m-perfect matching and the optimal m-subtree, and analyze its time complexity.

7.1 Description of the (t,k)-heuristic.

The *(t,k)-heuristic*, a class of heuristics for the optimal m-perfect matching and for the optimal m-subtree problem, where t and k are integer parameters ranging from 0 to *logn*. It consists of $(k+1)$ stages. At each stage $j = 0, 1, ..., k-1$, a complete graph $K(V_j)$, where $V_0 = V$, $V_j \subseteq V$ and $|V_j| = n_j$, is processed. Given $K(V_j)$, first the t-basic graph, $BG_t(V_j)$ (approximate t-basic graph, $\widehat{BG}_t(V_j)$ for Euclidean points in the plane), is extracted. From the t-basic graph (approximate t-basic graph), which is a forest of trees, a partial m-matching (a partial solution to the m-subtree problem)is selected, denoted by S_j, which is a collection of vertex disjoint cycles (trees), each of size exactly m, covering (possibly) a portion of vertices in V_j. All the vertices matched by S_j are removed from V_j, while all the remaining vertices form a complete graph $K(V_{j+1})$ to be processed in the next stage. After k stages, the remaining unmatched vertices (if any), in V_{k+1}, are matched by an m-perfect matching algorithm (an algorithm for the m-subtree problem), $\mathcal{A}$, which is either another heuristic or an exact algorithm for the problem.

Let $M_m^*(V)$ and $M_m^*(V_j)$ be an optimal m-perfect matching of graphs $K(V)$ and $K(V_j)$, respectively, and $\omega(M_m^*(V))$ and $\omega(M_m^*(V_j))$, their corresponding edge-weights. Let $T_m^*(V)$ and $T_m^*(V_j)$ be the optimal m-subtree problem of of graphs $K(V)$ and $K(V_j)$, respectively, and $\omega(T_m^*(V))$ and $\omega(T_m^*(V_j))$, their corresponding edge-weights.

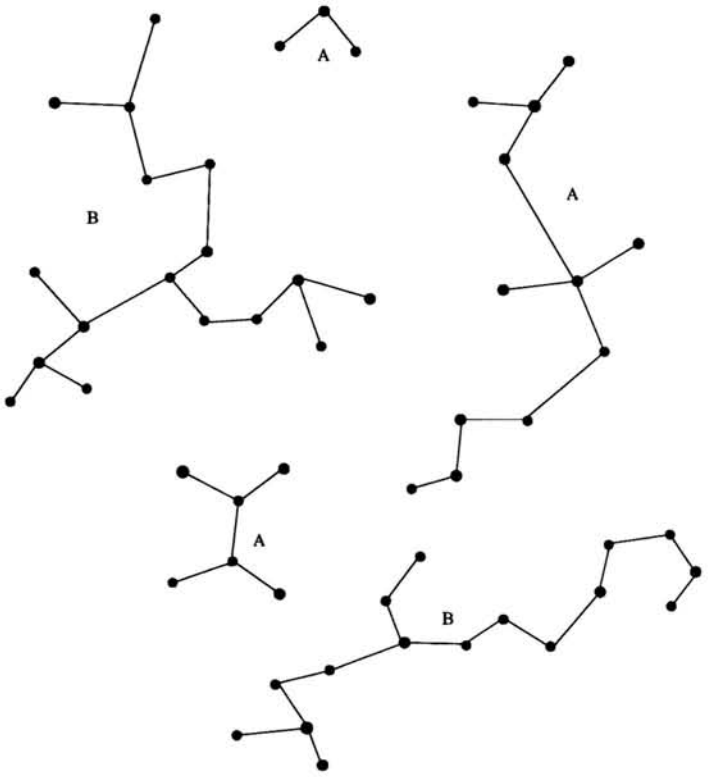


Figure 7.1: The t-basic graph with type A and type B hypervertices.

More specifically, at each stage j of the (t,k)-heuristic, the t-basic graph, $BG_t(V_j)$, (an approximate t-basic graph, $\widehat{BG}_t(V_j)$, for Euclidean points in the plane) is constructed, whose total weight does not exceed $(\frac{m-2}{m}+t)\omega(M^*_m(V_j))$ and $2t\omega(T^*_m(V_j))$ (for Euclidean points in the plane $(2\frac{m-1}{m}+\alpha(t-1))\omega(M^*_m(V_j))$ and $2[1+\alpha(t-1)]\omega(T^*_m(V_j))$). There are type A and type B hypervertices in the t-basic graph (approximate t-basic graph), and only the latter admit an m-perfect matching and an m-subtree problem. Fig.7.1 illustrates the t-basic graph for $m=3$.

A partial m-matching (a partial solution to the m-subtree problem) of $K(V_j)$, denoted by S_j, is selected from the t-basic graph. The edges in $BG_t(V_j)$ (in $\widehat{BG}_t(V_j)$) are doubled which results in a new graph, where each vertex has an even degree and each connected component is Eulerian. A graph is said to be Eulerian if there is a closed tour, called an Euler tour, which visits each edge exactly once. An Euler tour is extracted in every connected component of the duplicated t-basic graph (duplicated approximate t-basic graph), and ET_j denotes a union of all such tours. Thus

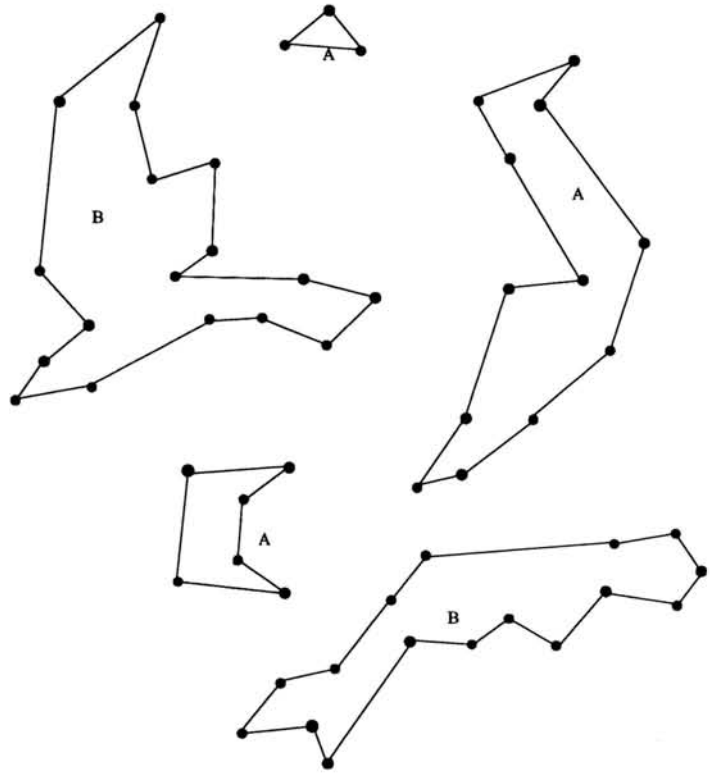


Figure 7.2: Union of Hamiltonian cycles extracted from each hypervertex in $BG_t(V_j)$.

Lemma 7.1.1 *The weight of the edges in ET_j is bounded above by*

$$2(\frac{m-2}{m}+t)\omega(M_m^*(V_j))$$

and by

$$4t\omega(T_m^*(V_j))$$

for weights satisfying the triangle inequality, and for Euclidean points in the plane by

$$2[2\frac{m-1}{m}+\alpha(t-1)]\omega(M_m^*(V_j))$$

and by

$$4[1+\alpha(t-1)]\omega(T_m^*(V_j))$$

Using the triangle inequality, the Euler tours are converted into Hamiltonian cycles, denoted by H_j, whose weight is bounded above by that of the corresponding tours, see

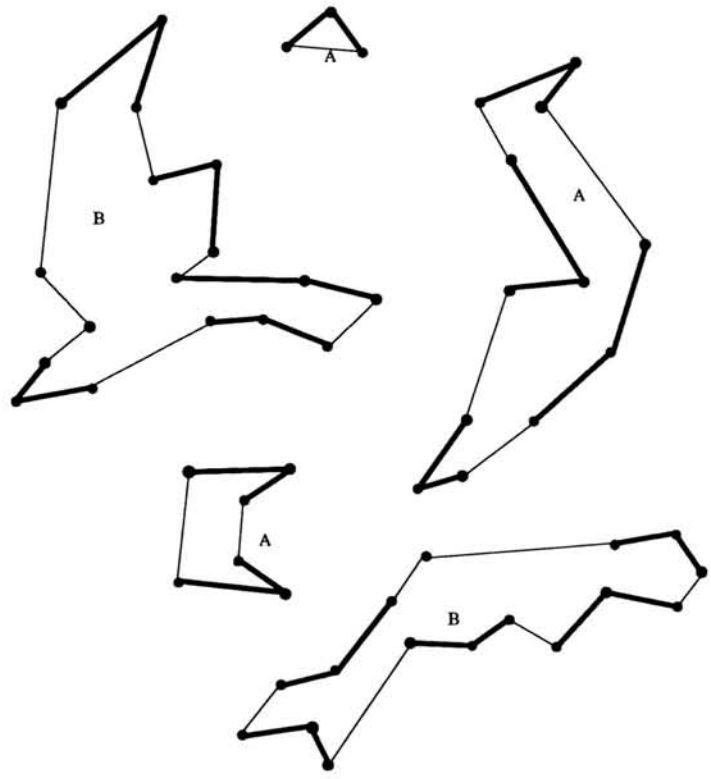


Figure 7.3: The maximum cardinality minimum weight m-chain cover of H_j.

Fig.7.2. A Hamiltonian cycle of a connected component is a cycle where each vertex is visited exactly once. Thus

Lemma 7.1.2 *The total edge weight of Hamiltonian cycles in H_j, selected by respective (t,k)-heuristic for the m-perfect matching and the m-subtree problem, is bounded above by the corresponding optimal weights for $M_m^*(V_j)$ and $M_m^*(V_j)$, as follows:*

$$\omega(H_j) \leq 2(\frac{m-2}{m} + t)\omega(M_m^*(V_j))$$

$$\omega(H_j) \leq 4t\omega(T_m^*(V_j))$$

for general weights satisfying the triangle inequality, and by

$$\omega(H_j) \leq 2[2\frac{m-1}{m} + \alpha(t-1)]\omega(M_m^*(V_j))$$

$$\omega(H_j) \leq 4[1 + \alpha(t-1)]\omega(T_m^*(V_j))$$

for Euclidean points in the plane. □

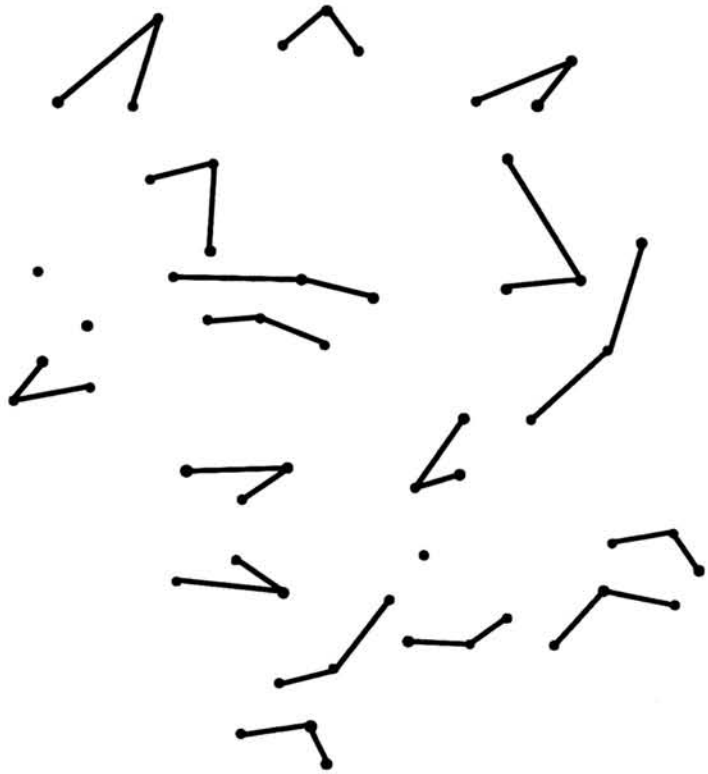


Figure 7.4: A partial solution to m-subtree problem of V_j.

We wish to extract from H_j the minimum weight solution, S_j, to the m-perfect matching and the m-subtree problem on each Hamiltonian cycle. The union of such solutions will form, respectively, a partial m matching and a partial solution to the m-subtree problem for $K(V_j)$. It will cover only a portion of the vertices in V_j, if there are any type B connected components in H_j. All the vertices which will not be covered at this stage, will form the vertex set V_{j+1}, to be processed in the next stage.

First, let us define the m-chain to be a chain of m consecutive vertices and $(m-1)$ consecutive edges on a cycle. The weight of an m-chain is the sum of the weights of its edges. The *maximum-cardinality minimum-weight m-chain cover* of a cycle is the maximum-cardinality minimum-weight set of vertex disjoint m-chains, which cover the cycle. If the cycle is of type A then the m-chain cover is a complete solution to the m-subtree problem for the cycle. From each cycle a maximum-cardinality minimum-weight m-chain cover is selected, see Fig.7.3.

For $m = 2$, when the m-subtree problem reduces to the optimal perfect matching,

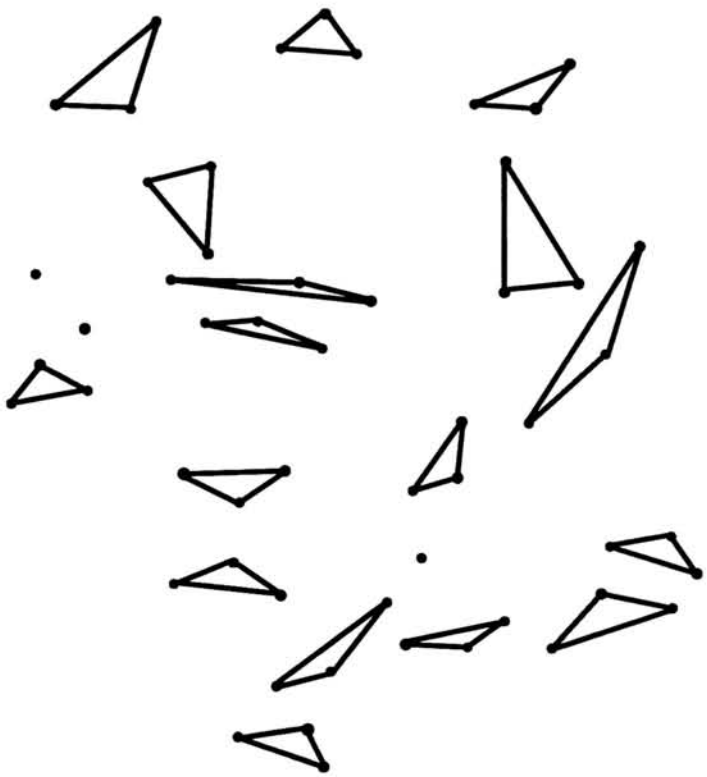


Figure 7.5: A partial m-perfect matching of V_j.

the union of such maximum-cardinality minimum-weight m-chain covers from all the cycles, contains at most half of all the edges in H_j, and their total weight is bounded above by the half of the total weight of H_j. We will show that for $m = 2$, H_j always admits the maximum cardinality minimum weight m-chain cover with at least $\lceil \frac{1}{2}\frac{3^t-1}{3^t}n_j \rceil$ edges. The number of edges in such S_j increases with t, since the minimum size of the type B connected components in H_j grows with t. This forms a partial solution to the 2-subtree problem, i.e. the perfect matching of V_j, whose weight is bounded above by $(\frac{1}{2})\omega(H_j)$. To obtain a partial 2-perfect matching, i.e. a partial duplicated perfect matching of V_j, each edge in the minimum-weight maximum-cardinality 2-cover is closed by an identical edge to form a 2-cycle. The weight of such partial 2-perfect matching is bounded above by $\omega(H_j)$.

For $3 \leq m \leq \frac{n}{2}$, S_j covers a portion of the vertices in V_j, and the number of the vertices in S_j depends on the parameters t and m. The size of a type B connected component in H_j is bounded below by a function of t, the bigger t the bigger the size

of smallest type B connected component. Let C_i be a type B cycle in H_j, it admits an m-chain cover with at least

$$\lceil \lfloor \frac{b_t}{m} \rfloor \frac{|C_i|}{b_t} \rceil \tag{7.1}$$

m-chains, where b_t is the size of a smallest type B cycle in H_j, which contains a multiple of m minus 1 vertices. Such a chain cover forms S_j for the m-subtree problem, see Fig.7.4, which consists of at least

$$\lceil \lfloor \frac{b_t}{m} \rfloor \frac{|V_j|}{b_t} \rceil \tag{7.2}$$

m-chains, and whose total weight is bounded above $\frac{m-1}{m}\omega(H_j)$.

To obtain S_j for the m-perfect matching, each m-chain is converted, in the maximum-cardinality minimum-weight m-chain cover of H_j, into a m-cycle, by adding a closing edge, whose weight is bounded above by the weight of the chain, see Fig.7.5. The weight of such S_j is bounded above by $2\frac{m-1}{m}\omega(H_j)$.

All the remaining vertices in V_j, unmatched by S_j, form a new complete graph, $K(V_{j+1})$, to be processed in the $j+1$ stage, see again Fig.7.4 and Fig.7.5. We repeat this procedure, and at the completion of the *k-th* stage, we apply to the set of

$$n_k \leq (\frac{m-1}{b_t})^k |V| \tag{7.3}$$

unmatched vertices an m-perfect matching (m-subtree problem) algorithm $\mathcal{A}$.

The central structure of the *(t,k)*-heuristic is the t-basic graph, and the crucial feature of the heuristic is the minimum size of a type B connected component. For $m = 2$, every odd connected component has size at least 3^t, and the maximum cardinality minimum weight 2-chain cover can be selected with at least $\lceil (\frac{1}{2})\frac{3^t-1}{3^t} n_j \rceil$ edges.

For $m \geq 3$, let

$$b_t = pm - 1 \tag{7.4}$$

be the size of the smallest type B connected component, where p is a natural number. This size is a function of parameters t and m, and represents the "worst-case size" of a

type B connected component, for which the S_j leaves the largest fraction of unmatched vertices. At each stage of construction of the t-basic graph, by finding a nearest neighbor graph of type B connected components, the b_t grows. Thus, for a given m, the larger the parameter t, the more m-chains can be selected in one stage of the (t,k)-heuristic to form S_j, and the smaller is the number of stages of the heuristic. This should lead to a smaller overall error of the approximate solution. But one should note that the parameter t is restricted by the size of V, and also the time complexity of the (t,k)-heuristic grows with bigger t.

The time complexity of the (t,k)-heuristic, when k is selected appropriately, is $O(tn^2)$ for weights satisfying the triangle inequality, and $O(tn\log n)$ for the Euclidean points in the plane.

At stage $(k+1)$ the *(t,k)*-heuristic applies an m-perfect matching (an m-subtree problem) algorithm $\mathcal{A}$, which is either an exact algorithm (for $m=2$, the exact perfect matching algorithm, and for $m \geq 3$, a dynamic programming algorithm) or a heuristic algorithm (foe example, the Generalized Hypergreedy and GGW). We always want to select such parameter k and t, which would guarantee the overall time complexity of the *(t,k)*-heuristic to be $O(tn^2)$, for weights satisfying the triangle inequality, and $O(tnlogn)$ for Euclidean points in the plane. The parameter k has to satisfy the property, that at stage k the number of remaining unmatched vertices, n_k, has to be small enough, so the time complexity of the algorithm $\mathcal{A}$ applied to the restricted problem, would not exceed the overall time complexity of stages $0,...,k-1$. Yet we want to minimize k. The (t,k)-heuristic is presented, more formally, in the following pseudocode:

(t,k)-heuristic

Input: $K(V)$, $V_0 = V$ ($V_0 = V$, set of vertices in the Euclidean plane)

Output: an m-perfect matching of $K(V)$ (a solution to the m-subtree problem of $K(V)$)

At each stage j, $j = 0,...,k-1$**:**

1. Construct the t-basic graph.

(Construct an approximate t-basic graph.)

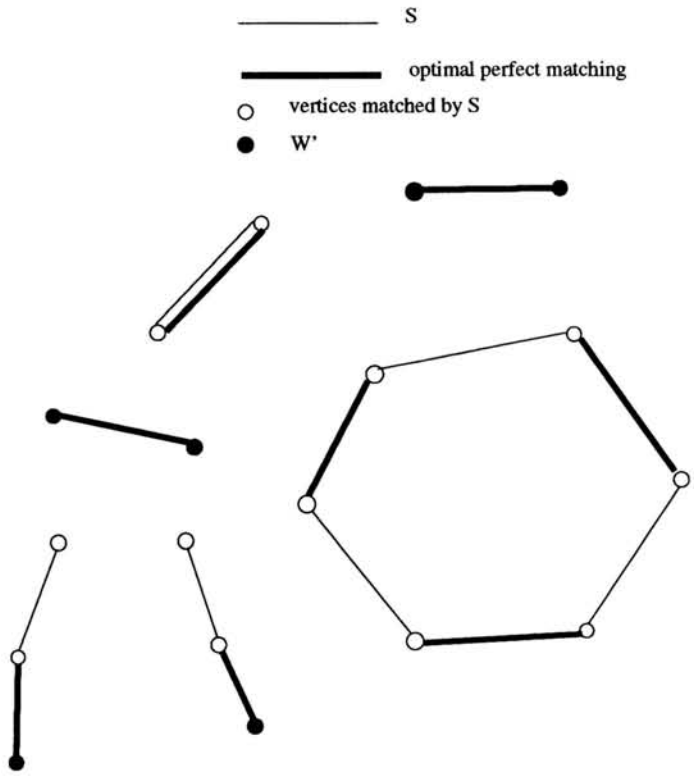


Figure 7.6: $S \bigcup T_m^*(W)$, for $m = 2$.

2. In every connected component of $BG_t(V_j)$ ($\widehat{BG}_t(V_j)$):

- duplicate the edges
- extract an Euler tour
- convert the tour into a Hamiltonian cycle

3. Find the maximum cardinality minimum weight m-chain cover, which forms a partial solution to the m-subtree problem.

4. (only for m-perfect matching) Add and edge to each m-chain in the cover to form a partial m-matching.

5. All unmatched vertices form $K(V_{j+1})$

After k stages: Match all the remaining vertices, if there are any left, using an auxiliary algorithm $\mathcal{A}$ for m-perfect matching (m-subtree problem of V).

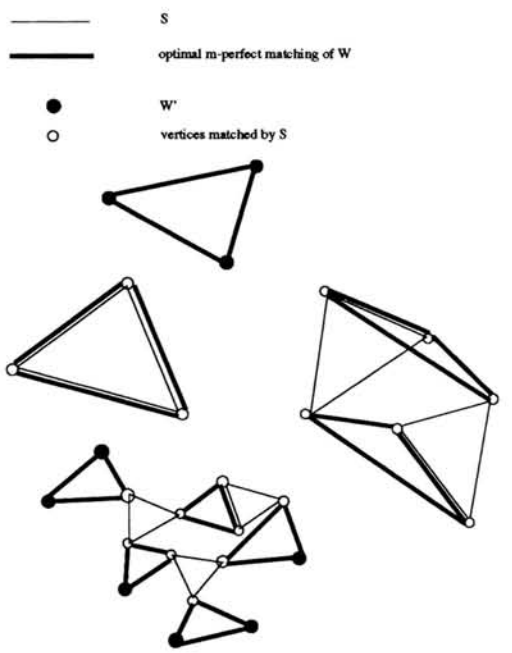


Figure 7.7: $S \bigcup M_m^*(W)$.

7.2 Error Analysis.

In this section we analyze the error of *(t,k)*-heuristic. First we introduce a generalization of Grigoriadis and Kalantari's lemma [29], which relates the weight of an optimal perfect matching of $K(V_{j+1})$ to the weights of a partial perfect matching, S_j, and an optimal perfect matching of $K(V_j)$. We will prove a new lemma which generalizes the Grigoriadis and Kalantari's lemma, to describe similar relationship for the optimal m-perfect matching and the optimal m-subtree problem. These two lemmas play a crucial role in deriving the overall error of the (t, k)-heuristic. We will use the results to relate the bound on the error obtained for one stage of the (t, k)-heuristic to that of the next stage. Finally, this relationship will allow to bound the overall error $f(n)$.

The following lemma has been introduced by Grigoriadis and Kalantari [29]. Given is a subset W of vertices of $K(V)$. Let S be a partial matching selected from W, and let

$T_2^*(W)$ and $T_2^*(W')$ be the optimal perfect matching of W and W' respectively, where W' is the set of unmatched vertices left after selecting S. Let $\omega(T_2^*(W))$, $\omega(T_2^*(W'))$, and $\omega(S)$ be the respective weights of $T_2^*(W)$, $T_2^*(W')$, and $\omega(S)$. In the lemma we relate the weights of $T_2^*(W')$, $T_2^*(W)$ and S.

Lemma 7.2.1

$$\omega(T_2^*(W')) \leq \omega(T_2^*(W)) + \omega(S)$$

Proof: (from [29]) Consider $S \bigcup T_2^*(W)$, see Fig.7.6. This union consists of a disjoint set of edges of $T_2^*(W)$, and of path and cycles whose edges alternate between S and T_2^*. No vertex of a cycle in $S \bigcup T_2^*(W)$ can belong to W' since all of its vertices are matched by S. Similarly, only the terminal vertices of a path can belong to W'. Let $P = \{(v_1, v_2), ..., (v_{t-1}, v_t)\}$ be such path. By triangle inequality we have $\omega(P) \geq \omega(e)$, where $e = (v_1, v_t)$. The set of such edges e, one for each $P \subseteq S \bigcup T_2^*(W)$, and all the disjoint edges of $T_2^*(W)$ in $S \bigcup T_2^*(W)$, is a perfect matching M' of W' with weight $\omega(T_2^*(W')) \leq \omega(M') \leq \omega(S) + \omega(T_2^*(W))$.□

We generalize the above lemma to the define a similar relationship for the optimal m-perfect matching and the optimal m-subtree problem. Given is a subset W of of vertices of $K(V)$. Let S be a partial m-perfect matching (a partial solution to m-subtree problem), selected from W, and let $M_m^*(W)$ and $M_m^*(W')$ (again $T_m^*(W)$ and $T_m^*(W')$), be the optimal m-perfect matching (optimal m-subtree problem) of W and W', respectively, where W' is the set of unmatched vertices left after selecting S. In the lemma we relate the weights of $M_m^*(W')$ and $M_m^*(W)$, for m-perfect matching, and the weights of $T_m^*(W')$ and $T_m^*(W)$, for m-subtree problem, and S, for $m \geq 3$.

Lemma 7.2.2

$$\omega(M_m^*(W')) \leq \frac{2(m-1)}{m}[\omega(M_m^*(W)) + \omega(S)]$$

$$\omega(T_m^*(W')) \leq \frac{2(m-1)}{m}[\omega(T_m^*(W)) + \omega(S)]$$

Proof: The union of $S \bigcup M_m^*(W)$ (the union of $S \bigcup T_m^*(W)$) is a collection of connected components, such that each of them is either a disjoint cycle (a disjoint tree) of $M_m^*(W)$

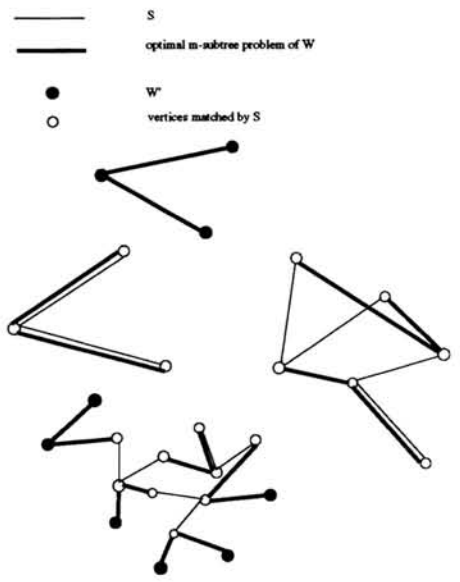


Figure 7.8: $S \bigcup T_m^*(W)$ $m \geq 3$.

(of $T_m^*(W)$), of size exactly m, or a combination of cycles (a combination of trees) of size m which alternate between those of $M_m^*(W)$ (those of $T_m^*(W)$) and S, see Fig.7.7 (see Fig.7.8). All the vertices matched by cycles (trees) in $M_m^*(W)$ (in $T_m^*(W)$), but not by S, belong to W', the set of vertices to be matched later. We observe that any connected component of $S \bigcup M_m^*(W)$ (any connected component of $S \bigcup T_m^*(W)$) contains either none or a multiple of m vertices in W'. Also, there is no vertex in $S \bigcup M_m^*(W)$ (no vertex in $S \bigcup T_m^*(W)$) which is matched only by S, but not by.

To prove the lemma, first the Euler tours will be extracted from the connected components of $S \bigcup M_m^*(W)$ and $S \bigcup T_m^*(W)$. The graph $S \bigcup M_m^*(W)$ is Eulerian, because each of its vertices which is matched either by one cycle of $M_m^*(W)$ or two cycles: one in $M_m^*(W)$ and one in S, has an even degree. Thus an Euler tour can be extracted, from each connected component, without duplicating edges in the graph. The graph $S \bigcup T_m^*(W)$ is not Eulerian, therefore to obtain Euler tours from its connected components, its edges have to be duplicated.

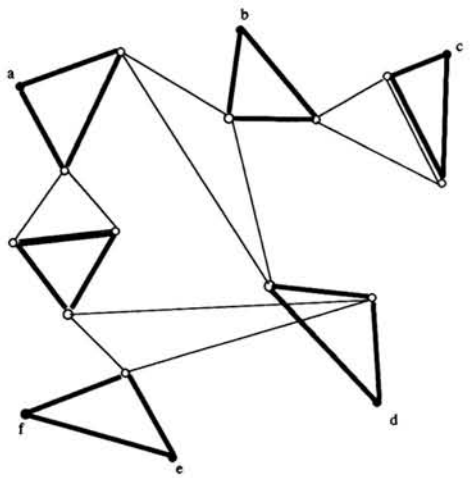


Figure 7.9: A connected component in $S \bigcup M_m^*(W)$.

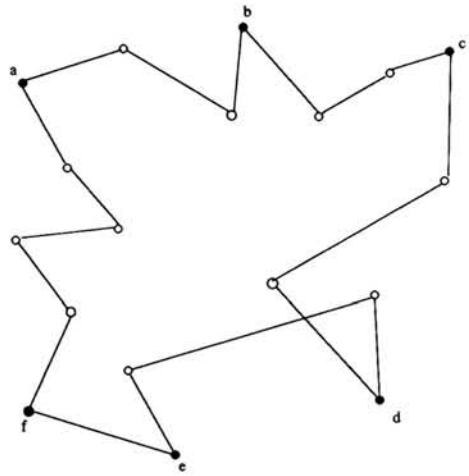


Figure 7.10: A Hamiltonian cycle obtained from a connected component in $S \bigcup M_m^*(W)$.

First we will prove the lemma for the optimal m-perfect matching. Let us assume, without loss of generality, that there is just one connected component in $S \bigcup M_m^*(W)$, see Fig.7.9, and we extract from it a single Euler tour, then using the triangle inequality we convert it into a Hamiltonian cycle on the set of vertices W, whose weight is bounded above by that of the tour, see Fig.7.10. Next, we retain in the cycle only those vertices in W', and replace chains between each two consecutive vertices in W' by an edge whose weight is equal to that of the chain. The new cycle has the same weight as that of the original Euler tour and all its vertices are in W', see Fig.7.11. Let us denote by E and $\omega(E)$ the cycle and its weight, respectively, and let mp be the number of vertices in the cycle, where p is a natural number. We show that maximum-cardinality minimum-weight m-chain cover of the cycle, denoted by M', has the weight bounded above by $\frac{(m-1)}{m}\omega(E)$, and the weight of the minimum-weight m-perfect matching of E, obtained from the cover, is bounded above by $2\frac{(m-1)}{m}\omega(E)$.

Let L be the complement of M' in E. Hence L is a collection of p single disjoint edges, see in Fig.7.12. We observe that there are mp distinct m-chains in all distinct maximum-cardinality m-chain covers of the cycle, and whose total weight is equal to $(m-1)\omega(E)$. There are exactly p m-chains in M'. Moreover there are m distinct maximum-cardinality m-chain covers in E, say $M^{(1)}, M^{(2)}, ..., M^{(m)}$. We associate with each such m-chain cover $M^{(i)}$ its unique complement $L^{(i)}$, $i = 1, 2, ..., m$, and the following can be said about $L^{(i)}$.

$$\bigcup_{i=1}^{m} L^{(i)} = E$$

$$\textit{and} \quad L^{(i)} \bigcap L^{(j)} = \{\textit{no common edges}\} \quad \textit{for} \quad i \neq j \quad \textit{and} \quad i, j = 1, 2, ..., m$$

$$\textit{Also} \quad \sum_{i=1}^{m} \omega(L^{(i)}) = \omega(E) \quad \textit{and} \quad \omega(L) = \omega(L^{(1)}) \geq \omega(L^{(2)}) \geq ... \geq \omega(L^{(m)})$$

Therefore, L with the biggest weight satisfies $\omega(L) \geq \frac{\omega(E)}{m}$, and the weight of its complement, the corresponding maximum-cardinality minimum-weight m-chain cover, M', is bounded above by $\omega(M') \leq (1 - \frac{1}{m})\omega(E) = \frac{m-1}{m}\omega(E)$. The M' is converted

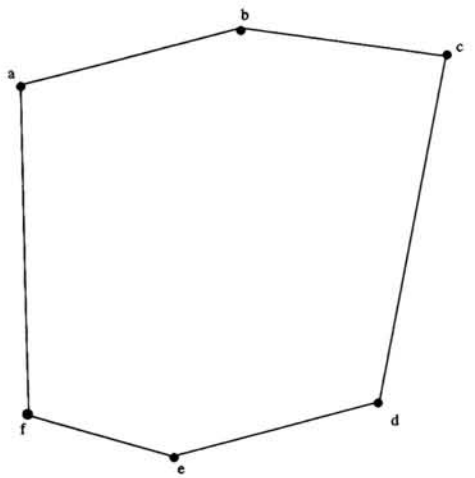


Figure 7.11: A modified Hamiltonian cycle with vertices in W'.

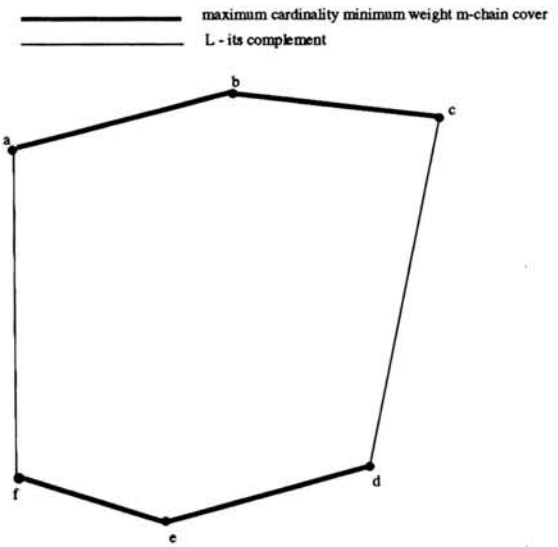


Figure 7.12: The maximum cardinality minimum weight m-chain cover.

into a minimum-weight m-perfect matching of the vertex set W', by adding to each m-cycle in M' a closing edge whose weight is bounded by the weight of the cycle. The weight of the m-perfect matching is bounded above by $2\frac{(m-1)}{m}\omega(E)$. Consequently, since $E = S \bigcup M_m^*(W)$ and $\omega(M_m^*(W')) \leq 2\frac{(m-1)}{m}\omega(E) = 2\frac{(m-1)}{m}[\omega(M_m^*(W)) + \omega(S)]$

To prove the lemma for the m-subtree problem, we will consider, as before, a single connected component in $S \bigcup T_m^*(W)$, see Fig.7.13. Since $S \bigcup T_m^*(W)$ is not Eulerian, we duplicate its edges, see Fig.7.14, extract and Euler tour from such a duplicated graph, and convert it into a Hamiltonian cycle whose weight is bounded by the weight of the tour, see Fig.7.15. We retain in the cycle only those vertices which are in W', and replace chains between each two consecutive vertices in W' by an edge whose weight is equal to that of the chain. The new cycle, denoted as before by E, has the weight bounded above by the weight of the Euler tour, which is *twice* the weight of $S \bigcup T_m^*(W)$, see again Fig.7.11. We find the maximum-cardinality minimum-weight m-chain cover of E, see Fig.7.12, which is an optimal solution to the m-subtree problem of W'. Thus since E is the duplicated $S \bigcup T_m^*(W)$ and $\omega(T_m^*(W)) \leq \frac{(m-1)}{m}\omega(E)$, then $\omega(T_m^*(W)) \leq \frac{(m-1)}{m}\omega(E) = 2\frac{(m-1)}{m}[\omega(T_m^*(W)) + \omega(S)]$. □

At each stage $j = 0, 1, ..., k-1$, of the *(t,k)*-heuristic, a partial solution S_j is selected from $K(V_j)$. We will derive bounds on $\omega(S_j)$ for the m-perfect matching and the m-subtree problem. We will obtain this bound with respect to $\omega(T_2^*(V_j))$, i.e. the weight of minimum-weight perfect matching, separately. First, we will show how many edges we can select into S_j.

Lemma 7.2.3 *At each stage $j = 0, 1, ..., k-1$ of the (t,k)-heuristic for $T_m^*(V)$, for $m = 2$, we can select from $K(V_j)$ a partial matching S_j containing at least $\lceil \frac{1}{2}\frac{3^t-1}{3^t}n_j \rceil$ edges.*

Proof: First we claim that the size of each odd hypervertex in the t-basic graph is at least 3^t. Clearly, this is true for $t = 1$. For $t > 1$, we only need to observe that an odd hypervertex in the t-basic graph is created from at least 3 odd hypervertices in the $(t-1)$-basic graph. Let $C_1, C_2, ..., C_l$ be the odd cycles, and let $C_{l+1}, ..., C_q$ be the even cycles in H_j. There are $\frac{|C_i|-1}{2} \geq \frac{3^t-1}{2}$, $i = 1, ..., l$, and $\frac{|C_i|}{2}$, $i = l+1, ..., q$, edges in any

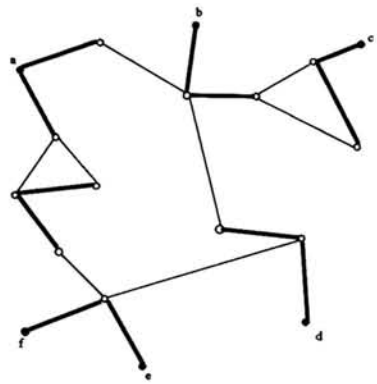


Figure 7.13: A connected component in $S \bigcup T_m^*(W)$.

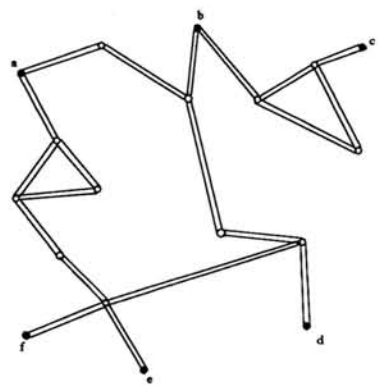


Figure 7.14: A duplicated connected component in $S \bigcup T_m^*(W)$.

maximum cardinality matching of each odd and even cycle, respectively. Thus we can select from each cycle at least $\lceil \frac{3^t-1}{2}\frac{|C_i|}{3^t} \rceil$, $i = 1, ..., q$, vertex disjoint edges. The proof of the lemma is now immediate from the following inequalities

$$\lceil \frac{1}{2}(\frac{3^t-1}{3^t})n_j \rceil \leq \sum_{i=1}^{q} \lceil \frac{3^t-1}{2}\frac{|C_i|}{3^t} \rceil. \ \square$$

For $m \geq 3$, we will show what is the minimum number of edges which always can be selected into a maximum-cardinality minimum-weight m-chain cover of H_j, at each stage j, $j = 0, 1, ..., k-1$, of the heuristic, where b_t is defined in Equation. 7.4.

Lemma 7.2.4 *At each stage $j = 0, 1, ..., k-1$ of the (t,k)-heuristic, for $m \geq 3$, a maximum-cardinality minimum-weight m-chain cover of H_j can be selected from $K(V_j)$, containing at least $(m-1)\lceil \frac{\lfloor \frac{b_t}{m} \rfloor}{b_t} n_j \rceil$ edges.*

Proof: Given the size of the "worst-case" type B component of H_j, $b_t = pm - 1$, where p is a natural number, let $C_1, C_2, ..., C_l$ be the type B cycles, and let $C_{l+1}, ..., C_q$ be the type A cycles in H_j. There are at least $(m-1)\lceil \frac{\lfloor \frac{b_t}{m} \rfloor}{b_t}|C_i| \rceil = (m-1)\lceil \frac{p-1}{pm-1}|C_i| \rceil$, $i = 1, ..., l$, and $(m-1)\frac{|C_i|}{m}$, $i = l+1, ..., q$, edges in any maximum-cardinality minimum-weight m-chain cover of each type B and type A cycle, respectively. Thus at least $(m-1)\lceil \frac{p-1}{pm-1}|C_i| \rceil$, $i = 1, ..., q$, edges can be selected from each cycle into the cover. The proof of the lemma is now immediate from the following inequalities

$$(m-1)\lceil \frac{\lfloor \frac{b_t}{m} \rfloor}{b_t} n_j \rceil = (m-1)\lceil \frac{p-1}{pm-1} n_j \rceil \leq \sum_{i=1}^{q} (m-1)\lceil \frac{p-1}{pm-1}|C_i| \rceil. \ \square$$

Lemma 7.2.5 *The partial matching S_j selected at each stage j, $j = 0, 1, ..., k-1$ of the (t,k)-heuristic for the perfect matching, and for general weights satisfying the triangle inequality, has weight bounded above by*

$$\omega(S_j) \leq 2t\omega(T_2^*(V_j))$$

Proof: The partial matching S_j is selected from H_j. The total weight of H_j is bounded above by $4t\omega(T_2^*(V_j))$, twice the weight of the t-basic graph of $K(V_j)$. We find the maximum cardinality minimum weight matching in each Hamiltonian cycle, and the eight of the matching is bounded above by half the weight of the cycle. From the

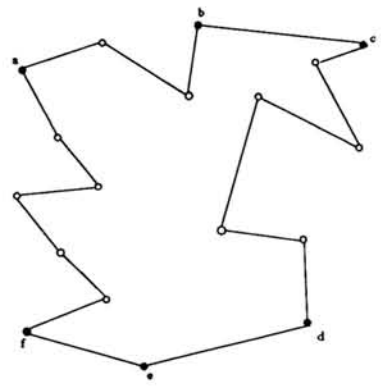


Figure 7.15: A Hamiltonian cycle obtained from a duplicated $S \bigcup T_m^*(W)$.

previous lemma, there could be at least $\lceil \frac{1}{2}\frac{3^t-1}{3^t} n_j \rceil$ edges selected into the maximum cardinality minimum weight matching of H_j, and those edges form a partial matching S_j. Clearly, $\omega(S_j) \leq \frac{1}{2}\omega(H_j)$. Hence, we have $\omega(S_j) \leq 2t\omega(T_2^*(V_j))$. □

Lemma 7.2.6 *The partial matching S_j selected at each stage j, $j = 0, 1, ..., k-1$, of (t,k)-heuristic for perfect matching, has weight bounded above by:*

$$\omega(S_j) \leq 2[1 + \alpha(t-1)]\omega(T_2^*(V_j)$$

for Euclidean points in the plane.

Proof: The proof is identical to the proof of previous lemma. The factor α in the bound comes from the fact that the $\omega(H_j) \leq 4[1 + \alpha(t-1)]\omega(T_2^*(V_j)$, and the Hamiltonian cycles are obtained from an approximate t-basic graph of $K(V_j)$. □

Now we will prove similar bounds on the weight of S_j, for $m \geq 3$, with respect to the weight of the optimal m-perfect matching and the optimal m-subtree problem.

Lemma 7.2.7 *A partial m-perfect matching selected at each stage j, $j = 0, 1, ..., k-1$, of (t,k)-heuristic, for $3 \leq m < n$, has weight bounded above by:*

$$\omega(S_j) \leq 4\frac{m-1}{m}[\frac{m-2}{m} + t]\omega(M_m^*(V_j))$$

for general weights satisfying the triangle inequality, and by

$$\omega(S_j) \leq 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(t-1)]\omega(M_m^*(V_j))$$

for Euclidean points in the plane.

Proof: We select in each cycle C_i, $i = 1, ..., q$, in H_j, a maximum-cardinality minimum-weight m-chain cover whose weight is bounded above by $\frac{m-1}{m}\omega(C_i)$. To obtain S_j, we convert each m-chain in the union of the maximum-cardinality minimum-weight m-chain covers obtained from all the Hamiltonian cycles, into a cycle by adding a closing edge. Thus

$$\omega(S_j) \leq \sum_{i=1}^{q} 2\frac{m-1}{m}\omega(C_i) = 2\frac{m-1}{m}\sum_{i=1}^{q}\omega(C_i) = 2\frac{m-1}{m}\omega(H_j)$$

,

and from Lemma 6.1.1, we have $\omega(S_j) \leq 4\frac{m-1}{m}[\frac{m-2}{m} + t]\omega(M_m^*(V_j))$, for general weights satisfying the triangle inequality, and $\omega(S_j) \leq 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(t-1)]\omega(M_m^*(V_j))$, for Euclidean points in the plane.

Lemma 7.2.8 *A partial solution to the m-subtree problem, selected at each stage j, $j = 0, 1, ..., k-1$, of (t,k)-heuristic, for $3 \leq m < n$, has weight bounded above by:*

$$\omega(S_j) \leq 4t\frac{m-1}{m}\omega(T_m^*(V_j))$$

for general weights satisfying the triangle inequality, and by

$$\omega(S_j) \leq 4\frac{m-1}{m}[1 + \alpha(t-1)]\omega(M_m^*(V_j))$$

for Euclidean points in the plane.

Proof: Proof of this lemma is similar the proof of the previous lemma.

The (t,k)-heuristic produce, in the first k stages, an approximate solution, which is the union of the partial solutions, S_j, $j = 0,1,...,k-1$. At each stage of the heuristic we are able to describe the error for the stage with relation to the error in the next stage. This important relationship will allow to derive the overall error of the heuristic. We will Let M'_j be a solution for the graph $K(V_j)$ produced by the *(t,k)*-heuristic. M'_j is a union of partial solutions produced at stages $j, j+1, ..., k$.

$$M'_j = \bigcup_{i=j}^{k} S_j$$

The ratio of the weight of M'_j to the weight of the optimal solution for $K(V_j)$, i.e. $M^*_m(V_j)$, for the optimal m-perfect matching and $T^*_m(V_j)$, for the optimal m-subtree problem, bounds above the error for the stage j. The error for each stage j is denoted by $f(n_j)$.

$$f(n_j) = \frac{\omega(M'_j)}{\omega(M^*_m(V_j))}, \text{ if } \omega(M^*_m(V_j)) \neq 0$$

$$f(n_j) = \frac{\omega(M'_j)}{\omega(T^*_m(V_j))}, \text{ if } \omega(T^*_m(V_j)) \neq 0$$

For a degenerate case, when $\omega(M^*_m(V_j)) = 0$ and $|V_j| \neq 0$ (when $\omega(T^*_m(V_j)) = 0$ and $|V_j| \neq 0$), i.e. V_j can be viewed as a set of points, each of multiplicity m, and the distance between each two within a group of m such points is 0, the solution M' produced by the *(t,k)*-heuristic, is identical to $M^*_m(V_j)$ (identical to $T^*_m(V_j)$), and $f(n_j) = 1$. If $|V_j| = 0$ then $f(n_j) = 0$, for $V_j = n_j$. In the following lemmas we relate errors in two consecutive stages.

Lemma 7.2.9 *The error for each stage j, $j = 0,1,...,k-1$, of (t,k)-heuristic for m-perfect matching is bounded above by:*

$$f(n_j) \leq 4\frac{m-1}{m}(\frac{m-2}{m} + t) + 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(\frac{m-2}{m} + t)]f(n_{j+1})$$

for general weights satisfying the triangle inequality, and

$$f(n_j) \leq 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(t-1)] + 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(2\frac{m-1}{m} + \alpha(t-1))]f(n_{j+1})$$

for Euclidean points in the plane.

Proof: From Lemma 6.2.2, $\omega(M_m^*(V_{j+1})) \leq 2\frac{m-1}{m}[\omega(M_m^*(V_j)) + \omega(S_j)]$, and from Lemma 6.2.7 $\omega(S_j) \leq 4\frac{m-1}{m}(\frac{m-2}{m} + t)\omega(M_m^*(V_j))$, for general weights satisfying the triangle inequality, and $\omega(S_j) \leq 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(t-1)]$, for Euclidean points in the plane. It gives

$$\frac{\omega(M_m^*(V_{j+1}))}{\omega(M_m^*(V_j))} \leq 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(\frac{m-2}{m} + t)]$$

for general weights satisfying the triangle inequality, and

$$\frac{\omega(M_m^*(V_{j+1}))}{\omega(M_m^*(V_j))} \leq 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(2\frac{m-1}{m} + \alpha(t-1))]$$

for Euclidean points in the plane.

The (t,k)-heuristic produces m-perfect matchings, M_j' and M_{j+1}' for $K(V_j)$ and $K(V_{j+1})$, respectively, and $\omega(M_j') = \omega(S_j) + \omega(M_{j+1}')$. The substitution of $\omega(S_j)$ gives

$$\omega(M_j') \leq 4\frac{m-1}{m}(\frac{m-2}{m} + t)\omega(M_m^*(V_j)) + \omega(M_{j+1}')$$

for general weights satisfying the triangle inequality, and

$$\omega(M_j') \leq 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(t-1)]\omega(M_m^*(V_j)) + \omega(M_{j+1}')$$

for Euclidean points in the plane. The proof of the lemma, for general weights satisfying the triangle inequality, follows by dividing by $\omega(M_m^*(V_j))$, writing

$$\frac{\omega(M_j')}{\omega(M_m^*(V_j))} \leq 4\frac{m-1}{m}(\frac{m-2}{m} + t) + \frac{\omega(M_{j+1}')}{\omega(M_m^*(V_{j+1}))}\frac{\omega(M_m^*(V_{j+1}))}{\omega(M_m^*(V_j))}$$

and using the bound of the last ratio. Similarly, for Euclidean points in the plane, we prove the lemma by replacing the last ration in the following expression,

$$\frac{\omega(M_j')}{\omega(M_m^*(V_j))} \leq 4\frac{m-1}{m}[2\frac{m-1}{m} + \alpha(t-1)] + \frac{\omega(M_{j+1}')}{\omega(M_m^*(V_{j+1}))}\frac{\omega(M_m^*(V_{j+1}))}{\omega(M_m^*(V_j))}. \quad \square$$

Lemma 7.2.10 *The error for each stage j, $j = 0, 1, ..., k-1$, of (t,k)-heuristic for m-subtree matching is bounded above by:*

$$f(n_j) \leq 4t\frac{m-1}{m} + 2\frac{m-1}{m}[1 + 4t\frac{m-1}{m}]f(n_{j+1})$$

for general weights satisfying the triangle inequality, and

$$f(n_j) \leq 4\frac{m-1}{m}[1 + \alpha(t-1)] + 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(1 + \alpha(t-1))]f(n_{j+1})$$

for Euclidean points in the plane.

Proof: Similar to the proof of previous lemma.

The overall error of the *(t,k)*-heuristic can be expressed by the ratio $f(n) = f(n_0) = \frac{\omega(M_0')}{\omega(M_m^*(V_0))}$, for the m-perfect matching and by the ratio $f(n) = f(n_0) = \frac{\omega(M_0')}{\omega(T_m^*(V_0))}$, for the m-subtree problem, where M_0' is a solution for $K(V)$, produced by *(t,k)*-heuristic, and $M_m^*(V_0)$ and $T_m^*(V_0)$ is an optimal m-perfect matching and optimal m-subtree problem of $K(V)$. We obtain the overall error formula, recursively, from the error bound formula defined for one stage of the *(t,k)*-heuristic.

Theorem 7.2.1 *The error of (t,k)-heuristic, which finds a solution to the m-subtree problem and the m-perfect matching, for $2 \leq m < n$, in a complete edge-weighted graph K(V), whose weights satisfy the triangle inequality, is bounded above by:*

$$f(n) \leq [\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_1)^k - \frac{4}{9}\frac{m}{m-1}$$

for general weights satisfying the triangle inequality, where $f_{\mathcal{A}}(n_k)$ is the error of an auxiliary algorithm $\mathcal{A}$ for the problem; and $a_1 = 2\frac{m-1}{m}[1 + 4t\frac{m-1}{m}]$ and $a_1 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(\frac{m-2}{m} + t)]$ for the m-subtree problem and the m-perfect matching, respectively.

Proof: Let us write the error bound for each stage j, from Lemma 7.2.9 and Lemma 7.2.10, for weights satisfying the triangle inequality, in the following form: $f(n_j) \leq A + Bf(n_{j+1})$, where $B = a_1$, and $A = 4t\frac{m-1}{m}\frac{m-1}{m}$ for the m-subtree problem and $A = 4\frac{m-1}{m}(\frac{m-2}{m} + t)$ for the m-perfect matching. We show that the number of vertices at stage j is bounded above by $n_j \leq (\frac{m-1}{b_t})^j n$. After k stages either or the vertices are

matched or there are n_k vertices to be matched by a M-perfect matching algorithm $\mathcal{A}$. We apply recursively Lemma 7.2.9 (Lemma 7.2.10) and obtain the overall error bound $f(n)$:

$$f(n) \leq A\sum_{i=0}^{k-1}(B)^i + (A)^k f_{\mathcal{A}}(n_k)$$

$$f(n) \leq \frac{A(B-1)}{B-1}\sum_{i=0}^{k-1}(B)^i + (B)^k f_{\mathcal{A}}(n_k)$$

$$f(n) \leq \frac{A}{B-1}\sum_{l=1}^{k}(B)^l - \frac{A}{B-1}\sum_{i=0}^{k-1}(B)^i + (B)^k f(n_k)$$

$$f(n) \leq \frac{A}{B-1}\sum_{l=1}^{k}(B)^l + \frac{A}{B-1} - \frac{A}{B-1} - \frac{A}{B-1}\sum_{i=0}^{k-1}(B)^i + (B)^k f_{\mathcal{A}}(n_k)$$

$$f(n) \leq \frac{A}{B-1}\sum_{l=0}^{k-1}(B)^l - \sum_{i=0}^{k-1}(B)^i + \frac{A}{B-1}(B)^k + (B)^k f_{\mathcal{A}}(n_k) - \frac{A}{B-1}$$

$$f(n) \leq [\frac{A}{B-1} + f_{\mathcal{A}}(n_k)](B)^k - \frac{A}{B-1}$$

We can show that $\frac{4}{9}\frac{m}{m-1} < \frac{A}{B-1} \leq \frac{m}{2(m-1)}$. □

Theorem 7.2.2 *The error of (t,k)-heuristic, which finds a solution to the m-subtree problem and the m-perfect matching for $2 \leq m < n$, for Euclidean points in the plane is bounded above by:*

$$f(n) \leq [\frac{m}{2(m-1)} + f_{\mathcal{A}}(n_k)](a_2)^k - \frac{4}{9}\frac{m}{m-1}$$

where $f_{\mathcal{A}}(n_k)$ is the error of an auxiliary algorithm $\mathcal{A}$; $a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(1 + \alpha(t-1))]$ and $a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(2\frac{m-1}{m} + \alpha(t-1))]$ for the m-subtree problem and the m-perfect matching, respectively.

Proof: Same as the proof of the above theorem.

Lemma 7.2.11 *The number of vertices n_j to be processed in each stage j, $j = 0, 1, ..., k-1$, of the (t,k)-heuristic, for given m and t, is bounded above by*

$$n_j \leq (\frac{m-1}{b_t})^j n,$$

where b_t is defined in Equation. 7.4.

Proof: We showed in Lemma 6.2.4, that we select from $K(V_j)$, $|V_j| = n_j$, a maximum-cardinality minimum-weight m-chain cover, with at least $(m-1)\lceil \frac{\lfloor \frac{b_t}{m} \rfloor}{b_t} n_j \rceil$ edges, where $b_t = pm - 1$ and p is a natural number. This means that we match in n_j at least $m\lceil \frac{\lfloor \frac{b_t}{m} \rfloor}{b_t} n_j \rceil$ vertices, and the remaining n_{j+1}, which will be processed in the next stage satisfy

$$n_{j+1} = n_j - m\lceil \frac{\lfloor \frac{b_t}{m} \rfloor}{b_t} n_j \rceil$$

We can express $\lfloor \frac{b_t}{m} \rfloor n_j = b_t l - r$, where l and $0 \leq r \leq b_t - 1$ are integers. Thus

$$n_{j+1} = n_j - m\lceil \frac{b_t l}{b_t} - \frac{r}{b_t} \rceil = n_j - lm$$

Since $l = \lfloor \frac{b_t}{m} \rfloor n_j + \frac{r}{b_t}$, we get

$$n_{j+1} = n_j - m\lfloor \frac{b_t}{m} \rfloor n_j - m\frac{r}{b_t} \leq n_j[1 - m\frac{\lfloor \frac{pm-1}{m} \rfloor}{b_t}] \leq n_j[1 - \frac{m(p-1)}{b_t}] = \frac{m-1}{b_t} n_j$$

From the above formula we can derive a bound on n_j

$$n_j \leq (\frac{m-1}{b_t})^j n. \square$$

In particular, for $m = 2$, $b_t = 3^t$ and $n_j \leq (\frac{1}{3^t})^j n$. For $m \geq 3$, an exact formula on b_t can't be provided, but one can show that $b_t \geq m2^t$ and $n_j \leq (\frac{(m-1)}{m2^t})^j n$.

In the following corollaries we will describe special cases of Theorem 6.2.1 and Theorem 6.2.2. First we will show the error bounds for the cases where we do not use the auxiliary algorithm in the last $(k+1)$ stage of the (t,k)-heuristic, i.e. the heuristic finds a complete solution in the first k stages, if k is selected appropriately.

Corollary 7.2.1 *Let $K(V)$ be a complete edge weighted graph with n vertices, the error of (t,k)-heuristic for m-perfect matching, where $m \geq 3$, $k = \frac{1}{t}\log\frac{n}{m}$, and $1, 2, ..., \log n$ is bounded above by:*

$$f(n) \leq [\frac{m}{2(m-1)} + 1](\frac{n}{m})^{\frac{1}{t}a_1} - \frac{4}{9}\frac{m}{m-1}$$

for general weights satisfying the triangle inequality, and

$$f(n) \leq [\frac{m}{2(m-1)} + 1](\frac{n}{m})^{\frac{1}{t}a_2} - \frac{4}{9}\frac{m}{m-1}$$

for Euclidean points in the plane, where $a_1 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(\frac{m-2}{m} + t)]$ and $a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(2\frac{m-1}{m} + \alpha(t-1))]$

Corollary 7.2.2 *Let $K(V)$ be a complete edge weighted graph with n vertices, the error of (t,k)-heuristic for m-subtree problem, where $m \geq 3$, $k = \frac{1}{t}\log\frac{n}{m}$, and $1, 2, ..., \log n$ is bounded above by:*

$$f(n) \leq [\frac{m}{2(m-1)} + 1](\frac{n}{m})^{\frac{1}{t}a_1} - \frac{4}{9}\frac{m}{m-1}$$

for general weights satisfying the triangle inequality, and

$$f(n) \leq [\frac{m}{2(m-1)} + 1](\frac{n}{m})^{\frac{1}{t}a_2} - \frac{4}{9}\frac{m}{m-1}$$

for Euclidean points in the plane, where $a_1 = 2\frac{m-1}{m}[1 + 4t\frac{m-1}{m}]$ and $a_2 = 2\frac{m-1}{m}[1 + 4\frac{m-1}{m}(1 + \alpha(t-1))]$

Corollary 7.2.3 *Given is a complete edge weighted graph $K(V)$, the error of the (t,k)-heuristic for perfect matching, ($T_m^*(V)$ form $= 2$), for $t = 1, 2, ..., \lfloor\log_3 n\rfloor$ and $k = \lfloor\log_{3^t} n\rfloor$, is bounded above by:*

$$f(n) \leq 2n^{\log_{3^t}[2t+1]} - 1$$

for general weights satisfying the triangle inequality, and

$$f(n) \leq 2n^{\log_{3^t}[2\alpha(t-1)+3]} - 1$$

for Euclidean points in the plane.

For $t = 1$, such (t, k)-heuristic becomes the weakest Onethird heuristic.

We show that if at the last $(k+1)$ stage of the (t, k)-heuristic the *GGW* algorithm is applied, the error of the resulting heuristic is a slowly growing function of n, and its time complexity is of $O(tn^2)$.

Corollary 7.2.4 *The error of the (t, k)-heuristic, which finds an m-perfect matching of $K(V)$, satisfying the triangle inequality, for $m \geq 3$ $t = 1, 2, ..., \frac{1}{4}\log\log\log n$, $k = \frac{1}{4t}\log\log\log n$, is bounded above by:*

$$f(n) \leq [\frac{m}{2(m-1)} + 2](\log\log n)^{\frac{1}{4t}\log a_1} - \frac{4}{9}\frac{m}{m-1}$$

where $a_1 = 2^{\frac{m-1}{m}}[1 + 4^{\frac{m-1}{m}}(\frac{m-2}{m} + t)]$, and for $m = 2$, $t = 1, 2, ..., \frac{1}{4}\log_3\log_3\log_3 n$, $k = \frac{1}{4t}\log_3\log_3\log_3 n$, the error is bounded above by:

$$f(n) \leq [\frac{m}{2(m-1)} + 2](\log_3\log_3 n)^{\frac{1}{4t}\log_3(1+2t)} - 1$$

Proof: Let $T_{\mathcal{A}}(n)$ be the time complexity of any perfect matching algorithm $\mathcal{A}$ used in the last stage of the (t, k)-heuristic. We want the parameter k to satisfy $O(T_{\mathcal{A}}(n_k)) = O(tn^2)$, which would guarantee the overall time complexity of the resulting heuristic to remain to be $O(tn^2)$ time. The *GGW* algorithm which runs in $O(n^2\sqrt{\log\log n})$ time is the fastest algorithm for the m-perfect matching with a constant error bound. We will use the algorithm to match the remaining n_k vertices, after k stages of the (t, k)-heuristic. In order to find the appropriate choice for parameters k and t, the following has to be satisfied:

$$(n_k)^2\sqrt{\log\log n} \leq tn^2$$

for $m \geq 3$, and by

$$(n_k)^2\sqrt{\log_3\log_3 n} \leq tn^2$$

for $m = 2$. Recall that $n_k \leq (\frac{m-1}{b_t})^k n$. It is to check that this is satisfied for $m \geq$, when $k = \frac{1}{4t}\log\log\log n$ and $1 \leq t \leq \frac{1}{4}\log_3\log_3\log_3 n$; and for $m = 2$ when $k = \frac{1}{4}\log_3\log_3\log_3 n$ and $1 \leq t \leq \frac{1}{4}\log_3\log_3\log_3 n$. Thus the rest of the proof follows from the error formula in Theorem 6.2.1. □

We obtain a similar result for the m-subtree problem in the following corollary:

Corollary 7.2.5 *The error of the (t,k)-heuristic, which finds a solution to the m-subtree problem of $K(V)$, for $m \geq 3$ $t = 1, 2, ..., \frac{1}{4}\log\log\log n$, $k = \frac{1}{4t}\log\log\log n$, is bounded above by:*

$$f(n) \leq [\frac{m}{2(m-1)} + 2](\log\log n)^{\frac{1}{4t}\log a_1} - \frac{4}{9}\frac{m}{m-1}$$

where $a_1 = 2^{\frac{m-1}{m}}[1 + 4t^{\frac{m-1}{m}}]$, and for $m = 2$, $t = 1, 2, ..., \frac{1}{4}\log_3\log_3\log_3 n$, $k = \frac{1}{4t}\log_3\log_3\log_3 n$, the error is bounded above by:

$$f(n) \leq [\frac{m}{2(m-1)} + 2](\log_3\log_3 n)^{\frac{1}{4t}\log_3(1+2t)} - 1$$

For example, for $m = 2$, from the above corollary we can obtain the following heuristics for perfect matching. The (t,k)-heuristic, for $t = 1$ (it becomes a Onethird heuristic) and $k = \frac{1}{4}\log_3\log_3\log_3 n$, gives a $O(n^2)$-time heuristic with a reasonable error of $3(\log_3\log_3 n)^{0.25} - 1$. This heuristic is better than the hypergreedy, both in time and error. The (t,k)-heuristic with $t = 4$ and $k = \frac{1}{16}\log_3\log_3\log_3 n$, gives also a $O(n^2)$-time heuristic with the error bounded above by $3(\log_3\log_3 n)^{0.125} - 1$, which is even better than that of the Onethird. Finally, for $k = 1$, $t = \frac{1}{4}\log_3\log_3\log_3 n$, we obtain a solution bounded above by $(1.5\log_3\log_3\log_3 n + 2)$ time the weights of the optimal solution. The corresponding time complexity is $O(n^2\log\log\log n)$, still an improvement over the $O(n^2\sqrt{\log_3\log_3 n})$ time of the *GGW* algorithm.

We can derive similar bound for other values of m. For example, for $m = 4$, $t = 4$ and $k = \frac{1}{16}\log\log\log n$, the (t,k)-heuristic gives a $O(n^2)$-time heuristic for m-subtree problem with a reasonable error of $2.66(\log\log n)^{0.27} - .59$, a faster heuristic than the *GGW* algorithm.

7.3 Time Complexity.

In this section we derive a bound on the worst case time complexity for the *(t,k)*-heuristic.

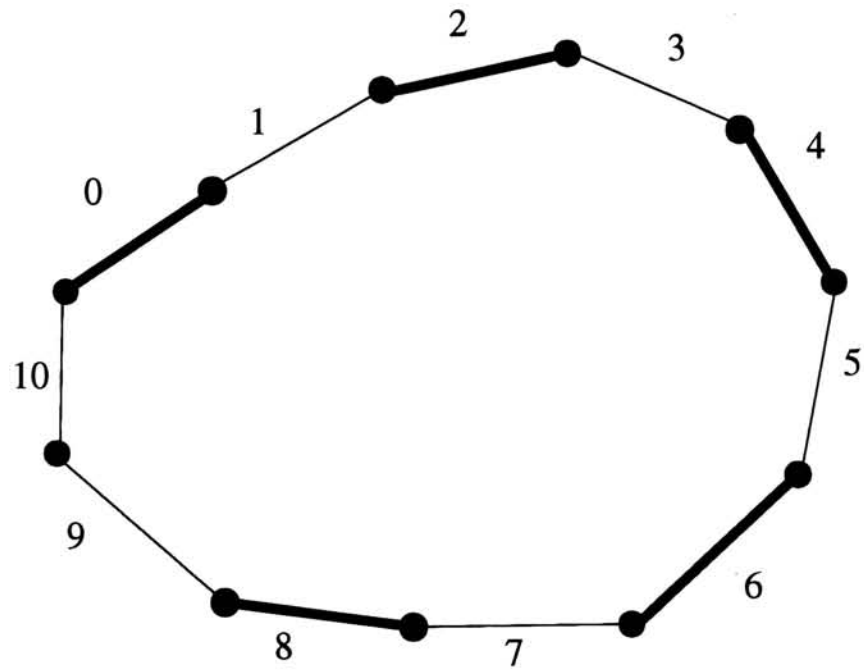


Figure 7.16: One of 11 maximum cardinality m-chain covers of a cycle of size 11, for $m = 2$.

Theorem 7.3.1 *The time complexity of the (t,k)-heuristic, for weights satisfying the triangle inequality is* $T(n) = O(tn^2)$, *and for the Euclidean points in the plane is* $T(n) = O(tnlogn)$.

Proof: For $j = 0, 1, ..., k-1$, from Lemma 4.2.3 and Corollary 5.2.1, the t-basic graph and the approximate t-basic graph of $K(V_j)$ can be implemented in $O(tn_j)$ and $O(tn_j \log n_j)$ time, respectively. From the t-basic graph (approximate t-basic graph), we construct a collection of Euler tours, then Hamiltonian cycles, in $O(n_j)$ time. We show that finding a maximum cardinality minimum weight matching of a Hamiltonian cycle is linear in the size of the cycle. Let the sequence $(0, 1, ..., q-1)$ correspond to the edges in a cycle of size q, in the order they appear. Without loss of generality we will show it for $m = 2$. If q is even, then there are 2 distinct maximum-cardinality 2-chain covers, each of size$\frac{q}{2}$, and we compute each of them and select the one with the smaller weight. For q odd, there are q different maximum cardinality 2-chain covers, each of size $\frac{q-1}{2}$. We can view the cycle as a circular list of q edges and each of its maximum cardinality 2-chain cover as a list of $\frac{q-1}{2}$ edges, see Fig 7.16. First we generate the

maximum cardinality 2-chain cover starting with the first list $(0, 2, 4, ..., q-3)$ of the consecutive vertex disjoint edges. We compute the weight of this cover in $O(q)$ time. We remove the first edge from the list (the edge 0), add the vertex disjoint edge in the cycle following the last edge in the list (the edge $(q-1)$), and compute in a constant time the weight of the resulting new cover accordingly. We repeat the process $O(q)$ time and obtain the weight of all the maximum cardinality 2-chain covers from which we select the one with the smallest weight.

The union of the maximum cardinality minimum-weight m-chain covers of all the Hamiltonian cycles, becomes a partial solution, S_j (for m-perfect matching, we convert all the m-chains into cycles in total $O(n_j)$ time). Therefore one stage of the (t,k)-heuristic can be implemented in $O(t(n_j)^2)$ and $O(tn_j \log n_j)$ time, for general weights satisfying the triangle inequality and for Euclidean points in the plane, respectively. Since $n_j \leq (\frac{m-1}{b_t})^j n$, the overall time complexity of the (t,k)-heuristic, for the general weights satisfying the triangle inequality, is bounded above by

$$T(n) = O(\sum_{j=0}^{k-1} t(n_j)^2) = O(\sum_{j=0}^{k-1} t(\frac{m-1}{b_t})^{2j} n^2)$$

$$T(n) = O(tn^2 \sum_{j=0}^{k-1} (\frac{m-1}{b_t})^{2j}) = O(tn^2 \frac{1}{1-(\frac{m-1}{b_t})^2}) = O(tn^2).$$

The overall time complexity of the *(t,k)*-heuristic for the Euclidean points in the plane is bounded above by,

$$T(n) = O(\sum_{j=0}^{k-1} t|n_j|log|n_j|) = O(\sum_{j=0}^{k-1} t(\frac{m-1}{b_t})^j n[log(\frac{m-1}{b_t})^j n])$$

$$T(n) = O(tnlogn \sum_{j=0}^{k-1} (\frac{m-1}{b_t})^j) = O(tnlogn) \ \square.$$

Chapter 8

The t-Hypergreedy heuristic

The t-hypergreedy is a generalization of the hypergreedy algorithm for minimum weight perfect matching on a complete edge weighted graphs whose weights satisfy the triangle inequality. Since the t-hypergreedy uses the exact algorithm for perfect matching, we couldn't extend the heuristic to get a solution to the m-perfect matching and the m-subtree problem, for any m.

8.1 Description of the t-hypergreedy.

Let $K(V)$ be a complete edge weighted graph with an even number, $n = |V|$, of vertices. We assume that the edge weights satisfy the triangle inequality. We show that for points in the Euclidean plane the error of a modified t-hypergreedy is bounded above by $\alpha(2t+1)$, where $\alpha = 2.42$, and its time complexity is $O(max\{tn\log n, \frac{n^2\log n}{3^t}, \frac{n^3}{3^{3t}}\})$. In particular, for $t = \lfloor \log_3 n \rfloor$ the modified t-hypergreedy has error bounded above by $\alpha(2\lfloor \log_3 n \rfloor + 1)$ and it can be implemented in $O(n\log^2 n)$ time. This time complexity is also favorable with respect to Vaidya's $O(n\log^3 n)$ time heuristic [53], for points in the Euclidean plane, whose error is bounded above by $3\log_3 \frac{3}{2}n$.

We now describe the t-hypergreedy heuristic. First, we construct the t-basic graph, $BG_t(V)$ (approximate t-basic graph, $\widehat{BG}_t(V)$ for the Euclidean case). This graph contains even and (possibly) odd hypervertices. If there are any odd hypervertices in the t-basic graph, we match them using an optimal perfect matching algorithm. This requires a preprocessing which consists of the computation of the shortest paths, possibly passing through even hypervertices, between every pair of odd hypervertices. Each shortest path is replaced by an edge which has the same weight as the weight

of the path. This gives a complete graph whose nodes are the odd hypervertices. Next we compute an optimal perfect matching on the reduced graph. This type of problem reduction was exploited in Grigoriadis and Kalantari [29]. The matching is then replaced by the corresponding set of paths. The edges in the paths are added to $BG_t(V)$. The new graph contains only even connected components. Each component admits a perfect matching and is processed separately. The matching is obtained in a similar fashion as the minimum spanning tree heuristic for the traveling salesman problem: The edges in each connected component are doubled. This results in a new graph, where each component is Eulerian. We extract an Euler tour in every connected component. Using the triangle inequality, Euler tours are replaced with Hamiltonian cycles of lesser weight. Each Hamiltonian cycle is the union of two disjoint perfect matchings. We select the one with the smaller weight. The union of the selected perfect matchings of the components forms a perfect matching of V. The weight of the optimal perfect matching of the odd hypervertices is bounded above by $\omega(T_2^*(V))$. To show it, we consider, n a similar fashion, as in the proof of Lemma 4.1.2, the union of the t-basic graph and $T_2^*(V)$. This union gives even cycles on the the odd hypervertices. Now it is easy to argue that in each cycle the total weight of edges from the optimal perfect matching of the odd hypervertices is less than equal to the total weight of edges from $T_2^*(V)$. Thus

Theorem 8.1.1 *The error of the t-hypergreedy for weights satisfying the triangle inequality, is bounded above by* $(2t+1)$, *for* $1 \leq t \leq \lfloor \log_3 n \rfloor$. □

Theorem 8.1.2 *The error of the approximate t-hypergreedy for points in the Euclidean plane is bounded above by* $\alpha(2t+1)$, *for* $1 \leq t \leq \lfloor \log_3 n \rfloor$.

8.2 Time Complexity.

We now analyze the time complexity of the second stage of the t-hypergreedy, i.e. the time needed to find the optimal perfect matching of the odd hypervertices either in $BG_t(V)$ or $\widehat{BG}_t(V)$. There is an even number of odd hypervertices in $BG_t(V)$ or $\widehat{BG}_t(V)$, if any. Having found an optimal perfect matching of the odd hypervertices,

we add the corresponding edges. This will result in a graph with hypervertices which are even only.

For each t, the number of vertices in each odd hypervertex of $BG_t(V)$ or $\widehat{BG}_t(V)$ is at least 3^t. This follows from the fact that at each stage a new odd hypervertex is formed by an odd number of odd hypervertices (at least 3 of them); thus the number of vertices in an odd hypervertex grows by a factor of at least 3. In particular, when $t = \lfloor \log_3 n \rfloor$ the heuristic reduces to the hypergreedy heuristic. For this t, there could be at most two remaining odd hypervertices. Otherwise, the graph would have at least $3^{\lfloor \log_3 n \rfloor + 1}$ vertices, which is impossible. Thus, for this t, the construction of the second part of the t-hypergreedy, i.e., the construction of the optimal perfect matching of the odd hypervertices is essentially the construction of the shortest path between the two odd hypervertices. This can be done in $O(n^2)$ time for general weights and $O(n \log n)$ time for the Euclidean case.

Given $BG_t(V)$, where odd and even hypervertices are treated as nodes, the t-hypergreedy proceeds as follows. We form a graph with those edges in $K(V)$, whose endpoints are in different hypervertices. In this graph, for all its $O(\frac{n}{3^t})$ odd nodes, the shortest paths between every pair of vertices is found. This can be done in $O(\frac{n^3}{3^t})$ time by applying the shortest path algorithm of Dijkstra $O(\frac{n}{3^t})$ times. We replace each such shortest path by an edge whose weight is equal to that of the path, and we match odd hypervertices in the complete graph using an exact perfect matching algorithm, in $O(\frac{n^3}{3^{3t}})$ time. Now all the connected components are even. The construction of Euler and Hamiltonian cycle requires $O(n)$ time. Each Hamiltonian gives rise to two different perfect matchings on its vertices, and the better of the two is selected. Thus we have

Theorem 8.2.1 *The time complexity of the t-hypergreedy is* $T(n) = O(max\{tn^2, \frac{n^3}{3^t}\})$, *where* $1 \leq t \leq \lfloor \log_3 n \rfloor$. □

Given $\widehat{BG}_t(V)$ for points in the Euclidean plane, the second part of the modified t-heuristic is implemented as follows. We form a graph where nodes are all even and odd hypervertices of $\widehat{BG}_t(V)$, and they are connected by those edges in the Delaunay triangulation whose endpoints are in different hypervertices. In this graph for each odd

hypervertex we find a shortest path to all other odd hypervertices in $O(\frac{n^2}{3^t}\log n)$ time using Dijkstra's shortest path algorithm $O(\frac{n}{3^t})$ times. As before, we form a complete graph whose nodes are all the odd hypervertices and edges correspond to the shortest paths between the hypervertices, followed by the computation of an optimal perfect matching. Thus we have

Theorem 8.2.2 *For points in the Euclidean plane, the time complexity of the modified t-hypergreedy is* $T(n) = O(max\{tn\log n, \frac{n^2\log n}{3^t}, \frac{n^3}{3^{3t}}\})$, *where* $1 \leq t \leq \lfloor \log_3 n \rfloor$. □

Chapter 9

Dynamic Programming Algorithms for Graph Covering Problems.

In this chapter we will present dynamic programming algorithms for the optimal m-perfect matching and optimal m-subtree problem, which can be used in the last stage of the (t,k)-heuristics, for $m \geq 3$. When the size of the problem in the last stage of the (t,k)-heuristic is small enough, the exact dynamic programming algorithm can be applied to obtain better error bound of the heuristic.

9.1 A Dynamic Programming Algorithm for the Optimal m-Perfect Matching.

Let $K(V)$ be a complete edge weighted graph, satisfying the triangle inequality, with $|V| = n$ vertices, where n is a multiple of m. Let $S_m, S_{2m}, ..., S_n$, be any subsets of V with m, $2m$,..., and n vertices, respectively. The *m-MATCH(V)* is an optimal m-perfect matching of $K(V)$.

The problem can be represented as a dynamic programming problem, as follows:

$$\omega(m\text{-}MATCH(V)) = min_{<v_1,...,v_m>}\{\omega(C(v_1,...,v_m))+\omega(m\text{-}MATCH(V\backslash\{v_1,...,v_m\}))\}$$

where $C(v_1,...,v_m)$ is a cycle of size m. We select the best m-perfect matching of V, i.e. *m-MATCH(V)* provided we know all the best m-perfect matchings of $m - MATCH(V \setminus \{v_1,...,v_m\})$ for every subset of V of size $|V| - m$. The number of computations necessary to find *m-MATCH(V)* is obtained as follows. Let us consider the following steps of the algorithms.

STEP.1: Take $\begin{pmatrix} n \\ m \end{pmatrix}$ choices of m points. For each choice S_m compute $m\text{-}MATCH(S_m)$,

i.e. solve the traveling salesman problem on S_m, using a dynamic programming algorithm. The cost of finding $m - MATCH(S_m)$ is $O(m^2 2^m)$, i.e. the cost of a dynamic programming algorithm for finding the best Hamiltonian cycle, given m nodes, [44].

STEP.2: Now consider finding of the two best m-cycles for each subset of V of size $2m$, i.e compute $m - MATCH(S_{2m})$ for $\begin{pmatrix} n \\ 2m \end{pmatrix}$ choices. The cost of finding the $m - MATCH(S_{2m})$ is:

$$\omega(m-MATCH(S_{2m})) = min_{<v_1,...,v_m>}\{\omega(C(v_1,...,v_m)+m-MATCH(S_{2m}\backslash\{v_1,...,v_m\}))\}$$

The total number of computations for finding m-$MATCH(V)$ is obtained recursively,

$$T(n) = m^2 2^m \begin{pmatrix} n \\ m \end{pmatrix} + \sum_{k=2}^{k=\frac{n}{m}} \begin{pmatrix} n \\ km \end{pmatrix} \cdot \begin{pmatrix} km \\ m \end{pmatrix} = m^2 2^m \begin{pmatrix} n \\ m \end{pmatrix} + \begin{pmatrix} m \\ n \end{pmatrix} \cdot \sum_{k=2}^{k=\frac{n}{m}} \begin{pmatrix} n-m \\ (k-1)m \end{pmatrix}$$

The expression is bounded by:

$$T(n) \leq \begin{pmatrix} n \\ m \end{pmatrix} (m^2 2^m + 2^{n-m})$$

For $m < n$ this gives:

$$T(n) = O(2^n(m^2 2^m + 2^{n-m}) = O(m^2 2^{n+m} + 2^{2n-m})$$

For $m = n$, the time complexity reduces to $T(n) = O(n^2 2^n)$, the running time of the dynamic programming algorithm for the traveling salesman problem.

In general, our dynamic programming algorithm is a big improvement over a brute force method which would compare all possible m-perfect matchings of $K(V)$ in $O((\frac{(m-1)!}{m!})^{\frac{n}{m}} n!)$ time.

9.2 A Dynamic Programming Algorithm for the Optimal m-Subtree Problem.

Let $K(V)$ be a complete edge weighted graph, satisfying the triangle inequality, with $|V| = n$ vertices, where n is a multiple of m. Let $S_m, S_{2m}, ..., S_n$, be subsets of V

with m, $2m$,..., and n vertices, respectively. The *m-FOREST(V)* is an optimal m-subtree problem of $K(V)$. The problem can be represented as a dynamic programming problem, as follows:

$$\omega(m\text{-}FOREST(V)) = min_{<v_1,...,v_m>}\{\omega(T(v_1,...,v_m))+\omega(m\text{-}FOREST(V\backslash\{v_1,...,v_m\}))\}$$

where $T(v_1,...,v_m)$ is a spanning tree of size m. The best solution to the m-subtree problem of V is selected, i.e. *m-TREE(V)* provided we know all the best solutions for the m-subtree problems of $m - TREE(V \setminus \{v_1,...,v_m\})$ for every subset of V of size $|V| - m$. To find the number of computations necessary to obtain *m-MATCH(V)*, consider the following steps of the algorithms.

STEP.1: Take $\begin{pmatrix} n \\ m \end{pmatrix}$ choices of m points. For each choice S_m compute $m\text{-}FOREST(S_m)$, i.e. solve the minimum spanning tree problem on S_m in $O(m^2)$ time.

STEP.2: Now consider finding of the two best m-trees for each subset of V of size $2m$, i.e compute $m - FOREST(S_{2m})$ for $\begin{pmatrix} n \\ 2m \end{pmatrix}$ choices. The cost of finding the $m - FOREST(S_{2m})$ is:

$$\omega(m\text{-}FOREST(S_{2m})) = min_{<v_1,...,v_m>}\{\omega(T(v_1,...,v_m)+m\text{-}FOREST(S_{2m}\backslash\{v_1,...,v_m\}))\}$$

The total number of computations for finding *m-FOREST(V)* is obtained recursively,

$$T(n) = m^2\begin{pmatrix} n \\ m \end{pmatrix} + \sum_{k=2}^{k=\frac{n}{m}} \begin{pmatrix} n \\ km \end{pmatrix} \cdot \begin{pmatrix} km \\ m \end{pmatrix} = m^2\begin{pmatrix} n \\ m \end{pmatrix} + \begin{pmatrix} m \\ n \end{pmatrix} \cdot \sum_{k=2}^{k=\frac{n}{m}} \begin{pmatrix} n-m \\ (k-1)m \end{pmatrix}$$

The expression is bounded by:

$$T(n) \leq \begin{pmatrix} n \\ m \end{pmatrix} (m^2 + 2^{n-m})$$

For $m < n$ this gives:

$$T(n) = O(2^n(m^2 + 2^{n-m}) = O(2^{2n-m})$$

For $m = n$, the time complexity reduces to $T(n) = O(n^2)$, the running time of the minimum spanning tree algorithm, for a complete edge-weighted graph with general weights.

Chapter 10

Parallel Implementation of a Perfect Matching Heuristic.

We give an example of a parallel implementation of a perfect matching heuristic. We hope that in the future we would be able to implement the other heuristics presented in the thesis.

We assume the following model of parallel computation. It is a parallel random access machine (PRAM), where the processors communicate through a common memory in constant time. In the exclusive-read exclusive-write (EREW) PRAM model, both read and write access is disallowed by more than one process to the same memory location. In the concurrent-read exclusive-write (CREW) PRAM, simultaneous reading from the same memory location is allowed, but not simultaneous writing [3].

By "fast" parallel algorithm using "small" number of processors we mean a parallel algorithm running in polylog $O((\log n)^k)$ time and using a polynomial number of processors. This class is called *NC*. Karp and Widgerson [36], showed that the maximal matching problem is in *NC*. They proposed a parallel algorithm which runs in $O((\log n)^4)$ parallel time and uses $O((\frac{n}{\log n})^3)$ processors.

In this paper we describe a parallel implementation of the simplest Grigoriadis and Kalantari's Onethird heuristic, which we call *Onethird-parallel*, for $k = \lfloor \log_3 n \rfloor$, The algorithm runs in $O((\log n)^3)$ and $O((\log n)^2)$ time and uses $O(n)$ and $O(n^2)$ processors for Euclidean points in the plane and the weights satisfying triangle inequality, respectively. We use the *CREW PRAM* model.

In the following section, we outline the Onethird, next we describe the parallel implementation of the algorithm and we analyze the time complexity of the parallel implementation of one stage of the algorithm for Euclidean weights and weights satisfying the triangle inequality, respectively. Finally, we obtain the overall time complexity.

10.1 Outline of the Algorithm

We denote a complete edge weighted graph $K(V)$, satisfying the triangle inequality, with $|V| = n$, an even number of vertices. For an edge $e \in K(V)$, we denote by $\omega(e)$ the weight of e. For$S \subseteq K(V)$, $\omega(S)$ is the total weight of its edges. Our heuristic algorithm finds a perfect matching T_V^* of $K(V)$ in $(k+1)$ stages, where $k = \lfloor \log_3 n \rfloor$ or $k = \lfloor \log_3 n - \log_3 \log_3(\frac{n}{9}) \rfloor$. In each stage it selects a partial matching S_j of $K(V_j)$, where V_j is the set of all-so-far unmatched vertices. S_j consists of $\lfloor (\frac{1}{3})n_j \rfloor$ edges, no two of which are incident upon the same vertex, where $n_j = |V_j|$. Also $\omega(S_j)$ is not greater than a certain portion of the optimal weight. We discard all the vertices matched by S_j, and the remaining unmatched vertices, denoted by V_{j+1}, form $K(V_{j+1})$ to be processed in the next stage. We repeat the same process $(k-1)$ times. For $k = \lfloor \log_3 n \rfloor$ a perfect matching of $K(V)$ is found. For $k = \lfloor \log_3 n - \log_3 \log_3(\frac{n}{9}) \rfloor$ and the Euclidean points in the plane, when $|V_{k+1}| \leq \lfloor \log_3 n \rfloor$, we apply Edmonds' serial algorithm for optimal perfect matching to obtain a perfect matching of V_{j+1}.

At each stage j of the algorithm we do the following. For the Euclidean points in the plane, first we find, in parallel, a Voronoi Diagram of V_j. Next we extract a nearest neighbor graph, which is a forest of trees. In each tree we duplicate edges, and find an Euler tour, which visits each edge exactly once. Each tour is further reduced to Hamiltonian cycle, which visits each node exactly once, and whose weight does not exceed that of the Euler tour. For each Hamiltonian cycle a maximum cardinality minimum weight matching is selected (or any matching whose weight does not exceed half of the weight of the Hamiltonian). Then all the matched edges in the union of the Hamiltonians are sorted by weight and exactly $\lfloor (\frac{n}{3})n_j \rfloor$ shortest edges are selected to form a partial matching S_j. For weights satisfying the triangle inequality, first we find a nearest neighbor graph of V_j, then proceed with the steps described above.

One of the differences between serial Onethird and Onethird-parallel is that we have to fix the parameter k, which is $k = \lfloor \log_3 n \rfloor$ and $k = \log_3 n - \log_3 \log_3(\frac{n}{9}) \rfloor$ for weights satisfying the triangle inequality and for Euclidean points in the plane, respectively. In the original algorithm k could be chosen as any integer between 1 and $\lfloor \log_3 n \rfloor$, and in

the $(k+1)$ stage an optimal perfect matching of graph $K(V_{k+1})$ is found using other perfect matching algorithm. Thus the worst case error bound and the time complexity of the algorithm depend on the choice of k; the smaller k the smaller worst case error bound and the higher time complexity. Since there is no fast parallel algorithm for exact minimum weight perfect matching, we are not able to match $K(V_{j+1})$, for an arbitrary k, using other parallel perfect matching algorithm. When the size of the problem becomes $n_{k+1} \leq \lfloor \log_3 n \rfloor$, we can apply Edmonds' serial algorithm which runs in $O((\log n_{k+1})^3)$ time, i.e. in time which does not exceed overall time complexity of the first k stages of our algorithm. But this applies only to points in the Euclidean plane.

10.2 The Algorithm Onethird.

procedure **Onethird**

INPUT: a complete edge weighted graph $K(V)$ satisfying the triangle inequality.

OUTPUT: a approximate perfect matching of $K(V)$.

$j \leftarrow 0$, $V_j \leftarrow V$

STEP.1 (omitted if the input is not a set of Euclidean points in the plane)
Find Voronoi diagram of V_j.

STEP.2 Find a nearest neighbor graph of $K(V_j)$.

STEP.3 For each connected component find an Euler tour.

STEP.4 Reduce each tour to a Hamiltonian cycle.

STEP.5 Select a partial matching S_j with $\lfloor (\frac{1}{3})n_j \rfloor$ edges whose weight does not exceed one third of the weight of the Hamiltonians.

STEP.6 Remove from V_j all the vertices matched by S_j and form a new complete graph $K(V_{j+1})$ from the remaining V_{j+1} vertices; go to STEP.1.

10.3 Parallel Implementation of Onethird for Euclidean Weights.

In this section we show a parallel implementation of one stage of Onethird-parallel, for the Euclidean points in the plane.

procedure **Onethird-parallel**

INPUT: A set V of n Euclidean points in the plane.

OUTPUT: a perfect matching of V.

Result: an $O((\log n)^3)$ time parallel algorithm using $O(n)$ processors and $O(n)$ space for the *CREW PRAM*.

$j \leftarrow 0$, $V_j \leftarrow V$.

Implementation of STEP.1:

Aggarwal et. all [1] proposed a parallel algorithm for Voronoi Diagram, running in $O((\log n_j)^3)$ time and using $O(n_j)$ processors. The more recent, improved, result has been obtained by Dadoun and Kirkpatrick [15]. Their algorithm, which we choose for the implementation, runs in $O((\log n_j)^2)$ time and uses $O(n_j)$ processors and $O(n_j)$ memory.

Implementation of STEP 2:

A nearest neighbor graph of V_j is a subgraph of a Delaunay triangulation, which is a dual graph of the Voronoi diagram. We find the nearest graph as follows. Given a Voronoi diagram of V_j, we assign one processor to every edge in the Voronoi diagram. There are no more than $3n_j - 1$ edges in a Voronoi diagram (see [46]). We form n_j classes, each class associated with one vertex in V_j, i.e. with one Voronoi region. Since each Voronoi edge is shared by two Voronoi regions, every processor belongs to two different classes. We solve the following minimum selection problem for each class. Let v be a vertex in V_j; a Voronoi region containing v is bounded by t edges, which are shared with t neighboring Voronoi regions, each containing one vertex of V_j. A processor assigned to each of the t edges computes a distance between v and the corresponding Voronoi neighbor of v. Since each processor stores exactly one distance between v and

its neighbor, finding a nearest neighbor of v is equivalent to finding the minimum of t elements. We can do it in $O(\log\log t)$ parallel time, using $O(n)$ processors, ([48], [54]). Since any vertex in V_j has at most $n_j - 1$ Voronoi neighbors, finding a nearest neighbor graph in V_j requires, in the worst case, $O(\log\log n_j)$ parallel time and $O(n)$ processors.

Implementation of STEP 3:

We identify all the connected components in the nearest neighbor graph of V_j using Shiloah and Vishkin's parallel connectivity algorithm [49], which runs in $O(\log n_j)$ parallel time and uses $O(n_j)$ number of processors and $O(n_j)$ memory. We double edges in each connected component and construct an Euler tour using Atallah's algorithm [6]. Finding all the Euler tours in the connected components requires $O(\log n_j)$ parallel time using $O(n_j)$ processors and $O(n_j)$ memory.

Implementation of STEP 4:

Let us assign directions to the edges in an Euler tour. The tour enters each vertex through an in-edge and leaves it through an out-edge. A vertex has in- and out-degree which is the number of in- and out-edges incident to it, respectively. The in- and out-degree of each vertex is the same. Each vertex, which appears in the tour more than once has out-degree greater than 1. Let v be a vertex in an Euler tour and (v, k) be the *k-th* copy of the vertex, when the tour visits it for the *k-th* time. We can view the tour as a cycle with k, $1 \leq k \leq out - degree(v)$ copies of each vertex v, which appear in the order they are visited by the tour. We represent the Euler tour by a double circular linked list, so that each vertex will know its unique predecessor and successor. To reduce an Euler tour to a Hamiltonian cycle we have to retain exactly one occurrence of each vertex in the cycle, let say we keep its $(v, 1)$ copy and discard all $\{(v, k) \mid 2 \leq k \leq out - degree(v)\}$ copies, and we do not change the order of appearance of the retained copies. Since the vertices in the Hamiltonian cycle will stay in the same relative order as in the original Euler tour, the weight of the Hamiltonian cannot exceed that of the Euler tour. This step requires $O(\log n_j)$ parallel time using $O(n_j)$ processors and $O(n_j)$ memory.

Implementation of STEP 5:

To select a partial matching, first we find maximum cardinality minimum weight matching for every Hamiltonian cycle. Next we sort all the matched edges, by their weights, in the union of the Hamiltonians and select $\lfloor(\frac{1}{3})n_j\rfloor$ shortest edges to form a partial matching. Let t be the number of vertices in a Hamiltonian cycle and $e_1, e_2, ..., e_t$ be the edges in the cycle in the order they appear.

If t is even, then there are two substrings of edges, each of size $(\frac{t}{2})$, which form a matching. We have to compute the weight of the two matchings, and choose the one which is smaller. We can find a sum of t numbers in $O(\log t)$ parallel time using $O(t)$ processors and $O(t)$ memory, see Akl [3].

For t odd, finding the maximum cardinality minimum weight matching is more complex, but we are able to do it within the same time complexity. There are t different matchings, in a cycle of size t, each containing $\lfloor\frac{t}{2}\rfloor$ edges. We compute the weights of the t matchings and select the one with the smallest weight. We don't have to find the weight of every matching separately. First we find the following groups of partial sums.

$$s_k = \sum_{i=1}^{\lfloor\frac{t}{2}\rfloor-k} \omega(e_{2i-1}), \quad k = 0, ..., \lfloor\frac{t}{2}\rfloor - 1$$

$$p_k = \sum_{i=\lfloor\frac{t}{2}\rfloor-k+1}^{\lfloor\frac{t}{2}\rfloor} \omega(e_{2i}), \quad k = 1, ..., \lfloor\frac{t}{2}\rfloor$$

$$z_k = \sum_{i=1}^{\lfloor\frac{t}{2}\rfloor-k} \omega(e_{2i}), \quad k = 1, ..., \lfloor\frac{t}{2}\rfloor - 1$$

$$r_k = \sum_{i=\lfloor\frac{t}{2}\rfloor-k+1}^{\lfloor\frac{t}{2}\rfloor} \omega(e_{2i+1}), \quad k = 1, ..., \lfloor\frac{t}{2}\rfloor$$

Each group of the partial sums $(s_1, ..., s_{\lfloor\frac{t}{2}\rfloor-1})$, $(p_1, ..., p_{\lfloor\frac{t}{2}\rfloor})$, $(z_1, ..., z_{\lfloor\frac{t}{2}\rfloor-1})$, $(r_1, ..., r_{\lfloor\frac{t}{2}\rfloor})$ can be computed in $O(\log t)$ parallel time with $O(t)$ processors and $O(t)$ memory. Now using a simple arithmetic we can compute in $O(1)$ parallel steps the weights of the t matchings. Let denote by $\omega(m_0)$, $\omega(m_1)$,...,$\omega(m_{t-1})$ the weights of t different matchings of a Hamiltonian cycle of odd size t.

$$\omega(m_0) = s_0$$

$$\omega(m_k) = s_k + p_k, \quad k = 1, ..., \lfloor \frac{t}{2} \rfloor - 1$$

$$\omega(m_{\lfloor \frac{t}{2} \rfloor}) = s_{\lfloor \frac{t}{2} \rfloor}$$

$$\omega(m_{\lfloor \frac{t}{2} \rfloor + k}) = z_k + r_k, \quad k = 1, ..., \lfloor \frac{t}{2} \rfloor - 1$$

$$\omega(m_{t-1}) = z_{\lfloor \frac{t}{2} \rfloor}$$

The maximum cardinality minimum weight matching of an odd cycle, which is the matching with the smallest weight, can be found in $O(\log\log t)$ parallel time. The largest cycle can have size $O(n_j)$, therefore the worst case time complexity of this step is $@O(\log n_j)$.

There are at least $\lfloor \frac{n_j}{3} \rfloor$ and at most $\lfloor \frac{n_j}{2} \rfloor$ matched edges in the union of the Hamiltonians. We sort the matched edges by the weights in $O(\log n_j)$ parallel time using $O(n_j)$ processors, see [11]. Then we choose $O(\lfloor \frac{n_j}{3} \rfloor$ shortest edges to form a partial matching S_j. We'll show later that the weight of the partial matching does not exceed a certain portion of the optimal matching of the union of the Hamiltonians.

implementation of STEP 6: Removing all the vertices matched by S_j from V_j can be executed in $O(1)$ parallel steps.

The overall time complexity The implementation of one stage of the Onethird-parallel , for Euclidean points in the plane, requires $O((\log n_j)^2)$ parallel time (time complexity of *STEP 1*), using $O(n_j)$ processors and $O(n_j)$ space.

10.4 Parallel Implementation of Onethird for Weights Satisfying the Triangle Inequality.

In this section we show a parallel implementation of Onethird for weights satisfying the triangle inequality. The implementation starts with *STEP 2* of the Onethird algorithm. We also omit description of those steps of the Onethird-parallel algorithm which are identical to the algorithm for the Euclidean case.

*procedure***Onethird-parallel**

INPUT: A complete edge weighted graph $K(V)$ satisfying the triangle inequality.

OUTPUT: A perfect matching of $K(V)$.

Result: $O((\log n)^2)$ time algorithm using $O(n^2)$ processors on *CREW-PRAM* model.

$j \leftarrow 0$; $V_j \leftarrow V$

implementation of STEP 2:

We assign one processor to every edge in $K(V_j)$. We are able to find a nearest neighbor graph of V_j in $O(\log\log n_j)$ parallel time using $O((n_j)^2)$ processors, [48].

Implementation of *STEPS 3-6* is identical to the one for the Euclidean case. The time complexity of one stage of the Onethird-parallel for weights satisfying the triangle inequality is $O(\log n_j)$ using $O((n_j)^2)$ processors.

10.5 Analysis of the Worst-case Error

In the implementation of *STEP 5* of the Onethird-parallel we find the maximum cardinality minimum weight matching for every Hamiltonian cycle, then sort all the matched edges and select $\lfloor \frac{n}{3} \rfloor$ shortest edges to form a partial matching. In the original Onethird-parallel algorithm, all edges are sorted in the union of all Hamiltonians, instead, and $\lfloor \frac{n}{3} \rfloor$ shortest edges are selected such that no two of them are incident upon the same vertex (they are vertex-disjoint). Such selection of $\lfloor \frac{n}{3} \rfloor$ vertex disjoint edges is strictly

serial (to select the next shortest edge which does not share a vertex with any of so-far-selected edges, we have to know in advance all these edges). We can show that the problem is P-complete, by reducing it to lexicographically minimum solution, [5]. This is why we had to replace *STEP 5* of Onethird-parallel with something else.

In the implementation of *STEP 5*, we find maximum cardinality minimum weight matching in the union of the Hamiltonians, denoted by W_j, then select $\lfloor \frac{n_j}{3} \rfloor$ shortest matched edges. We will show in Lemma 1 that the total weight of the $\lfloor \frac{n}{3} \rfloor$ selected edges is bounded above by $\frac{\lfloor \frac{n_j}{3} \rfloor}{\lfloor \frac{n_j}{2} \rfloor} \omega(W_j)$. Since the weight of W_j is no bigger than the half of the total weight of all the Hamiltonians (each optimal matching of a cycle has weight at most half of the weight of the cycle), then the weight of the partial matching S_j, selected by our algorithm satisfies the Lemma 2 from Grigoriadis and Kalantari's paper, [29]. This lemma says that at any stage of the algorithm the weight of the selected partial matching is no bigger than one third of the total weight of the union of all Hamiltonians.

Lemma 10.5.1 *For every stage $j = 1, ..., k$ the set of edges S_j is a partial matching of $K(V_j)$, satisfying:*

$$\omega(S_j) \leq \frac{\lfloor \frac{n_j}{3} \rfloor}{\lfloor \frac{n_j}{2} \rfloor} \omega(W_j)$$

Proof: Let $\lfloor \frac{n_j}{3} \rfloor = p$ and $\lfloor \frac{n_j}{2} \rfloor = q$. Vector *MATCH-EDGE* contains q ordered numbers $a_1 \leq a_2 \leq ... \leq a_p \leq ... \leq a_q$.

$$\omega(S_j) = \sum_{i=1}^{p} a_i$$

$$\omega(W_j) = \sum_{i=1}^{q} a_i$$

$$Let \quad \omega(S') = \sum_{i=1}^{p} a_i + (q-p)a_p = \omega(S_j) + (q-p)a_p \leq \omega(W_j)$$

$$\frac{\omega(S')}{\omega(S')} = \frac{\omega(S_j) + (q-p)a_p}{\omega(S')}$$

$$1 = \frac{\omega(S_j)}{\omega(S')} + \frac{(q-p)a_p}{\omega(S')}$$

$$\frac{\omega(S_j)}{\omega(S')} = 1 - \frac{(q-p)a_p}{\omega(S')} \leq 1 - \frac{(q-p)a_p}{qa_p} = \frac{p}{q}$$

□.

Let denote by $\omega(M)$ the weight of an approximate perfect matching produced by our algorithm, and by $\omega(M^*)$ the weight of the optimal perfect matching of $K(V)$. We can express the error of the approximation as the ratio $f(n) = \frac{\omega(M)}{\omega(M^*)}$. At each stage j of our algorithm we match $\lfloor \frac{n_j}{3} \rfloor$ edges which form a partial matching S_j. In the following lemma we show a relation between the weight of an optimal matching of any complete graph $K(V_j)$ to the weights of a partial matching S_j and of an optimal matching of a complete graph $K(V_{j+1})$. Graph $K(V_{j+1})$ is a subgraph of the graph $K(V_j)$ after removal of all vertices matched in the set S_j and all edges incident upon them. Let $\omega(M_j^*)$ and $\omega(M_{j+1}^*)$ be the weights of the optimal perfect matchings of $K(V_j)$ and $K(V_{j+1})$, respectively. Then

Lemma 10.5.2 *(Lemma 1 in [29])*

$$\omega(M_{j+1}^*) \leq \omega(M_j^*) + \omega(S_j); \quad j = 0, 1, ..., k-1$$

Proof: see [29].

Now we will summarize the worst case error analysis from [29]. Before we derive a bound on $f(n)$, we find the worst-case error for every stage of our algorithm. We denote by $M_j = \sum_{i=j}^{k} S_j$ the matched edges selected in stages j through k of our algorithm; those edges form a perfect matching of $K(V_j)$. Thus $M_0 = M$ is a perfect matching of $K(V)$, produced by our algorithm. The error for each stage of the algorithm is a ratio $f(n_j) = \frac{\omega(M_j)}{\omega(M_j^*)}$, if $\omega(M_j^*) \neq 0$, otherwise M_j will be an optimal matching of $K(V_j)$, and the error we say is $f(n_j) = 1$. In the following lemma we relate the worst case error at a stage to the error in the next stage.

Lemma 10.5.3 *(Lemma 3 in [29])*

$$f(n_j) \leq (\frac{4}{3}) + (\frac{7}{3}) f(n_j) \quad for \quad j = 0, 1, ..., k$$

Proof: To prove Lemma 3 we use Lemma 1 and Lemma 2. For detailed description see [29].

Using Lemma 1 and Lemma 2 we get that $\frac{\omega(M^*_{j+1})}{\omega(M^*_j)} \leq \frac{7}{3}$. In the following theorem we derive a bound on the error $f(n)$:

Theorem 10.5.1 *(theorem 1 in [29]*

$$f(n) \leq 2(\frac{7}{3})^k - 1.$$

In our algorithm we fix the number of stages k. If $k = \lfloor \log_3 n \rfloor$, then by Theorem 1 the worst-case error bound for the approximate perfect matching, produced by the algorithm is $f(n) \leq 2(\frac{7}{3})^{\log_3 n} - 1 \leq 2n^{0.771} - 1$. For Euclidean points in the plane we can achieve a slight improvement on the ratio $f(n)$ by decreasing the number of stages of Onethird-parallel. The second choice of k is based upon the observation that at each stage of our algorithm the size of the problem is reduced to approximately one third of its size at the previous stage. At stage j, the size of the problem has a bound $n_j \leq (\frac{1}{3})^j n + 2$, for $j = 1, 2, ..., k$. We can show that after at most $k = \lfloor \log_3 n - \log_3 \log_3(\frac{n}{9}) \rfloor$ stages the size of the problem is bounded by $\lfloor \log_3 n \rfloor$. Then an optimal perfect matching is extracted (using serial Edmonds' algorithm), in the last $(k+1)$-st stage, which produces no error, and the ratio $f(n)$ has slightly better bound over the one for $k = \lfloor \log_3 n \rfloor$.

$$f(n) \leq 2(\frac{7}{3})^{\log_3 n - \log_3 \log_3(\frac{n}{9})} - 1 \leq 2\frac{n^{\log_3(\frac{7}{3})}}{[\log_3(\frac{n}{9})]^{\log_3(\frac{7}{3})}} - 1 \leq 2[\frac{n}{\log_3(\frac{n}{9})}]^{0.7713} - 1$$

10.6 Time Complexity

At each stage j of our algorithm, the size of the problem n_j is equal at most one third of its size in the previous stage, i.e. $n_j \leq (\frac{1}{3})^j n + 2$, for $j = 1, 2, ..., k$, see Lemma 4 in [29].

Theorem 10.6.1 *Onethird for Euclidean points in the plane can be implemented in $O((\log n)^3)$ parallel time using $O(n)$ processors and $O(n)$ space.*

Proof:

As we have shown in section 4 Onethird-parallel finds a partial matching of a set V of points in the Euclidean plane in $O((\log n_j)^2)$ parallel time using $O(n_j)$ processors and $O(n_j)$ space. We find the overall time complexity as follows:

$$T(n) \leq c \cdot \sum_{i=1}^{k} (\log n_j)^2, \quad where \quad n_0 = n$$

$$n_j \leq (\frac{1}{3})^j n_0 + 2 \leq (\frac{1}{3})^{j-1} n_0, \quad k = \lfloor \log_3 n \rfloor$$

$$T(n) \leq c \cdot \log n \cdot \sum_{i=1}^{k} \log n_j = c \cdot \log n \cdot \log[\prod_{i=1}^{k} n_i]$$

$$T(n) \leq c \cdot \log n \cdot \log[1 \cdot (\frac{1}{3}) \cdot (\frac{1}{3})^2 \cdot \ldots \cdot (\frac{1}{3})^{k-1} \cdot n^k]$$

$$T(n) \leq c \cdot \log n \cdot \log[(\frac{1}{3})^{\frac{(k-1)k}{2}} \cdot n^k]$$

$$T(n) \leq c \cdot \log n \cdot \log[(\frac{1}{n})^{\frac{k}{2}} \cdot n^{\frac{1}{2}} \cdot n^k]$$

$$T(n) \leq c \cdot (\frac{k}{2} + \frac{1}{2}) \cdot (\log n)^2 \leq (\log n)^3$$

□.

Theorem 10.6.2 *Onethird-parallel for weights satisfying the triangle inequality can be implemented in $O((\log n)^2)$ parallel time using $O(n^2)$ processors.*

Proof: Similar to the proof of Theorem 2.

10.7 Comparison with other heuristics

Onethird-parallel which is a parallel implementation of Onethird has the same worst-case error bound as Onethird for $k = \lfloor \log_3 n \rfloor$ and (for the Euclidean points in the plane) $k = \lfloor \log_3 n - \log_3 \log_3(\frac{n}{9}) \rfloor$. The worst case error bound for $k\lfloor \log_3 n \rfloor$ is $2n^{0.7713} - 1$ and, for $k = \lfloor \log_3 n - \log_3 \log_3(\frac{n}{3}) \rfloor$ is $2[\frac{n}{\log_3(\frac{n}{9})}]^{0.7713} - 1$. The second bound increases slower in function of n than the first one.

Now let us compare Onethird with the other known simple heuristics for perfect matching. The *greedy* heuristic which has the worst-case error bound of $(\frac{4}{3})n^{0.585}$, performs better than Onethird with $k = \lfloor \log_3 n \rfloor$ or $k = \lfloor \log_3 n - \log_3 \log_3(\frac{n}{9}) \rfloor$ stages. In the greedy, two nearest unmatched vertices are repeatedly matched and the process seems to be strictly serial. Also Vaidya's heuristic for Euclidean points in the plane, is superior (as far as the worst-case error bound) to the Onethird but does not have the simplicity of the Onethird. The same we can say about the hypergreedy, and the (t, k)-heuristic, which consist of some strictly serial steps. The open problem is whether one can replace the strictly serial parts of other heuristics, which provide better error bounds than the simplest Onethird heuristic, and implement them in parallel.

Chapter 11

Conclusions.

We presented in this thesis families of heuristics for spectra of graph covering problems. The 2-approximate heuristic for the CFP problem, a problem which we also proved to be NP-hard, plays a crucial role in designing of other classes of heuristics. The heuristic for the CFP, which works for any weights, and runs within the time needed to compute a minimum spanning tree, is superior to the corresponding 2-approximate and GGW algorithm, because of the same error bound and faster time complexity. Even though the heuristic for the CFP is as simple as the trivial MST heuristic for the traveling salesman problem, showing that its error is bounded by 2 required a sophisticated proof. From this heuristic, we obtained, under the assumption of the triangle inequality, a 4-approximate heuristic for the CCP which has the same time complexity. It is faster than the corresponding 2-approximate GGW algorithm for the problem. In the following table we show the error and the time complexity of the GGW algorithm for all the four classes of problems:

Problem	Error	Time Complexity
CCP	2	$O(n^2\sqrt{\log logn})$
CCP	2	$O(n^2\sqrt{\log logn})$
perfect matching	2	$O(n^2\sqrt{\log logn})$
m-subtree problem	4	$O(n^2\sqrt{\log logn})$
m-perfect matching	4	$O(n^2\sqrt{\log\log n})$

The error and the time complexity of the Generalized Hypergreedy for the m-subtree problem and the m-perfect matching, for weights satisfying the triangle inequality, are

shown in the following two table, for selected m:

Generalized Hypergreedy for the m-subtree problem:

(general weight satisfying the triangle inequality)

m	Error	Time Complexity
2	$2\lfloor \log_3 n \rfloor$	$O(n^2 \log n)$ (hypergreedy)
$\frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$	12	$O(n^2)$
$\frac{n}{3}, \frac{n}{4}$	8	$O(n^2)$
$\frac{n}{2}$	4	$O(n^2)$
n	1	$O(n^2)$ exact MST alg.

For $m = \frac{n}{2}$ the Generalized Hypergreedy is better than the corresponding 4-approximate and $O(n^2\sqrt{\log \log n})$-time GGW algorithm.

Generalized Hypergreedy for the m-perfect matching:

(general weight satisfying the triangle inequality)

m	Error	Time Complexity
2	$2\lfloor \log_3 n \rfloor$	$O(n^2 \log n)$
$\frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$	16	$O(n^2)$
$\frac{n}{3}, \frac{n}{4}$	12	$O(n^2)$
$\frac{n}{2}$	8	$O(n^2)$
n	2	$O(n^2)$ MST heuristic for TSP

For Euclidean points in the plane the error and time complexity of the Generalized

Hypergreedy, for selected m, are described in the following two tables for the Euclidean m-subtree problem and the m-perfect matching, respectively, where $\alpha = 2.42$:

Generalized Hypergreedy for the m-subtree problem:

(Euclidean points in the plane)

m	Error	Time Complexity
2	$2\alpha\lfloor \log_3 n \rfloor$	$O(n \log^2 n)$ (Euclidean hypergreedy)
$\frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$	$4(1+2\alpha)$	$O(n \log n)$
$\frac{n}{3}, \frac{n}{4}$	$4(1+\alpha)$	$O(n \log n)$
$\frac{n}{2}$	4	$O(n \log n)$
n	1	$O(n \log n)$ exact Euclidean MST alg.

Generalized Hypergreedy for the m-perfect matching:

(Euclidean points in the plane)

m	Error	Time Complexity
2	$2\alpha\lfloor \log_3 n \rfloor$	$O(n \log^2 n)$
$\frac{n}{5}, \frac{n}{6}, \frac{n}{7}, \frac{n}{8}$	$8(1+\alpha)$	$O(n \log n)$
$\frac{n}{3}, \frac{n}{4}$	$4(2+\alpha)$	$O(n \log n)$
$\frac{n}{2}$	8	$O(n \log n)$
n	2	$O(n \log n)$ MST heuristic for Euclidean TSP

The Generalized Hypergreedy for the the m-subtree problem and the m-perfect matching, for Euclidean points in the plane, runs much faster than the corresponding *GGW* algorithm.

The error and the time complexity of the (t,k)-heuristic for the m-subtree problem and the m-perfect matching, for selected m will be shown in the following tables. The results are self explanatory.

(t,k)-heuristic for the m-subtree problem:

(general weight satisfying the triangle inequality)

$m = 2$ and $k = log_3 n$ stages of the algorithm

t	Error	Time Complexity
1	$2n^{0.771} - 1$	$O(n^2)$, (Onethird)
5	$2n^{0.435} - 1$	$O(n^2)$
10	$2n^{0.277} - 1$	$O(n^2)$
$\lfloor \log_3 n \rfloor$	$2\lfloor \log_3 n \rfloor$	$O(n \log^2 n)$ (hypergreedy)

In the above table we see that the (t,k)-heuristic, for $m = 2$, spans the Onethird and the hypergreedy. The adjustable time complexity depends on the choice of t, provided that k was selected appropriately. In the following table, the exact algorithm is applied at the last stage of the (t,k)-heuristic, for $m = 2$. This is a theoretical results, since the implementation of the exact Edmonds' algorithm is not trivial:

(t,k)-heuristic for the m-subtree problem, $m = 2$:

(general weight satisfying the triangle inequality)

an exact perfect matching algorithm is used at stage $k = (\frac{1}{3t}) \log_3 (\frac{n}{t})$

t	Error	Time Complexity
1	$2n^{0.257} - 1$	$O(n^2)$, (Onethird)
5	$2(\frac{n}{5})^{0.145} - 1$	$O(n^2)$
10	$2(\frac{n}{10})^{0.092} - 1$	$O(n^2)$
$\lfloor \log_3 n \rfloor$	$2\lfloor \log_3 n \rfloor$	$O(n \log^2 n)$ (hypergreedy)

Applying an exact algorithm for carefully selected k and t results in improved error bounds for growing t. We have to note that even though for $t = 10$ the asymptotic error decreases considerably, the constant in the time complexity, which is proportional to t, grows. Finally, we present the best, so far, combination of the (t, k)-heuristic with an auxiliary heuristic, which is the combination of the (t, k)-heuristic with the *GGW* algorithm.

(t, k)-heuristic for the m-subtree problem, $m = 2$:
(general weight satisfying the triangle inequality)
the *GGW* algorithm is used at $k = (\frac{1}{4t}) \log_3 \log_3 \log_3 n$

t	Error	Time Complexity
1	$3(\log_3 \log_3 n)^{0.25} - 1$	$O(n^2)$, (Onethird)
2	$3(\log_3 \log_3 n)^{0.183} - 1$	$O(n^2)$
3	$3(\log_3 \log_3 n)^{0.0.147} - 1$	$O(n^2)$
4	$3(\log_3 \log_3 n)^{0.125} - 1$	$O(n^2)$
10	$3(\log_3 \log_3 n)^{0.07} - 1$	$O(n^2)$
$\frac{1}{4} \log_3 \log_3 \log_3 n$	$\frac{3}{2}(\log_3 \log_3 \log_3 n) + 2$	$O(n^2 \log\log\log n)$

From this table we observe that the combination of the (t, k)-heuristic with the *GGW* algorithm, results in an $O(n^2)$-time heuristics whose error bounds are very slowly growing functions of n. These heuristics are faster than the *GGW* algorithm, and their error is relatively small, even for a large-size problems. The last $O(n^2 \log\log\log n)$ heuristic in the above table is also faster the *GGW* algorithm with a $\log\log\log n$-error. We obtain a similar results for other m. For example:

(t, k)-heuristic for the m-subtree problem, $m = 3$:
(general weight satisfying the triangle inequality)
the *GGW* algorithm is used at stage $k = (\frac{1}{4t}) \log\log\log n$

t	Error	Time Complexity
1	$2.75(\log\log n)^{0.57} - .66$	$O(n^2)$
2	$2.75(\log\log n)^{0.38} - .66$	$O(n^2)$
3	$2.75(\log\log n)^{0.29} - .66$	$O(n^2)$
4	$2.75(\log\log n)^{0.21} - .66$	$O(n^2)$
10	$2.75(\log\log n)^{0.13} - .66$	$O(n^2)$

We introduce two exact dynamic programming algorithms for optimal m-perfect matching and the optimal m-subtree problem, which run in $O(m^2 2^{n+m} + 2^{2n-m})$ and $O(2^{2n-m})$ time, for $m < n$, respectively. For $m = n$, the dynamic programming algorithm for the m-perfect matching reduces to the $O(n^2 2^n)$-time dynamic programming algorithm for the traveling salesman problem, and the dynamic programming algorithm for the m-subtree problem becomes the $O(n^2)$-time algorithm for minimum spanning tree. These heuristics can be also used in the last stage of the (t, k)-heuristic, when the size of the problem becomes small enough.

We develop a heuristic for the minimum-weight perfect matching, called the *t-hypergreedy*, which is a combination of the hypergreedy heuristic with an exact minimum-weight perfect matching algorithm. In the following table we present the error and time complexity of the t-hypergreedy for weights satisfying the triangle inequality:

Heuristic	Error	Time Complexity
t-hypergreedy	$2t+1$	$O(max\{tn^2, \frac{n^3}{3^t}\})$

For Euclidean points in the plane, the error and the time complexity of the t-hypergreedy are following:

Heuristic	Error	Time Complexity
t-hypergreedy	$\alpha(2t+1)$	$O(max\{tn\log n, \frac{n^2\log n}{3^t}, \frac{n^3}{3^{3t}}\})$

Our $O(n\log^2 n)$ implementation of the Euclidean hypergreedy is a special case of

the t-hypergreedy for $t = \lfloor \log_3 n \rfloor$, whose error is bounded by $2\alpha\lfloor \log_3 n \rfloor$, $\alpha = 2.42$. It is the fastest heuristic for the Euclidean perfect matching with a $\log n$-error.

We may consider, in the future, the following extensions of the results presented in the thesis:

- Developing of a t-Generalized Hypergreedy, which would be a generalization of the t-hypergreedy for any m. This would apply only to large m.
- Finding better error bound of the heuristic for the CCP. The conjecture is that the bound should be 2 instead of 4. The bound of 4 which we have obtained is not tight.
- Defining other graph covering problems, which can be solved using a variation of the (t, k)-heuristic, e.g. we may cover an edge-weighted graph with other connected components than trees and cycles.
- Implementation of all of our heuristics and comparison of their performance to the performance of other algorithms that have been implemented.
- Finding which of the heuristics can be implemented in parallel.

References

[1] A. Aggarval, B. Chazelle, L. Guibas, C. O'Dunlaing, C. K. Yap, Parallel computational geometry, *Algorithmica*, (3), 293-327, 1988.

[2] A. Aggarwal, P. Shor, A linear algorithm for computing the Voronoi Diagram of convex polygon, *STOC*, (1987).

[3] S. G. Akl, *Parallel sorting algorithms*, Notes and Reports in Computer Science and Applied Mathematics, Academic Press Inc., New York, (1985).

[4] S. G. Akl, *The design and analysis of parallel algorithms*, Prentice Hall, Englewood Cliffs, New Jersey, (1989).

[5] R. J. Anderson, *The complexity of parallel algorithm*, Ph.D. Thesis, Stanford University, Palo Alto, (1986).

[6] M. Atallah, Finding Euler tour in parallel, *Journal of computer and System Sciences*, (29), 330-337, (1984).

[7] D. Avis, Worst case bounds for the Euclidean matching problem, *Comput. Math. Appl.*, (7), 251-257, (1981).

[8] D. Avis, A Survey of heuristics for weighted matching problem, *Networks*, (13), 475-493, (1983).

[9] L. P. Chew, There is a planar graph almost as good as the complete graph, Proceedings of the 2nd Symposium on Computational Geometry, Yorktown Heights, NY, 169-177, (1986).

[10] N. Christofides, Worst-case analysis of a new heuristic for the Traveling Salesman Problem, Tech. Report, G.S.I.A., Carnegie-Mellon University, Pittsburgh, PA, (1976).

[11] R. Cole, Parallel merge sort, Courant Institute of Math. Sciences, NYU, New York, (1986).

[12] T. H. Cormen, C.E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press and McGraw-Hill, Cambridge, (1989).

[13] G. Cornuejols and W. Pullyblank, A matching problem with side constraints, *Disc. Math.*, 29, 135-159, (1980).

[14] W. H. Cunnigham and A. B. Marsh III, A primal algorithm for optimum matching, *Math. Prog. Study*, 8, 50-72, (1978).

[15] N. Dadoun and D.G. Kirkpatrick, Parallel Processing for efficient subdivision search, (1986).

[16] D. P. Dobkin, S. J. Friedman, K. J. Supowit, Delaunay graphs are almost as good as complete graphs, *FOCS*, (1987).

[17] J. Edmonds, Paths, trees and flowers, *Canadian Journal of Mathematics*, (17), 449-467, (1965).

[18] J. Edmonds, Matching and a polyhedron with 0-1 vertices, *Journal of Research National Bureau of Standards*, 69B, 125-130, (1965).

[19] J. Edmonds and E. L. Johnson, Matching: a well-solved class of integer linear programs, *Proc. Calgary Intern. Conf. on Combinatorial Structures and Their Applications*, 89-92, (1970).

[20] R. Fredman and R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, (1984).

[21] H. N. Gabow, Implementation of algorithms on nonbipartite graphs, Ph.D thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, (1973).

[22] H. N. Gabow, A scaling algorithm for weighted matching on general graphs, *FOCS*, 90-100, (1985).

[23] H. N. Gabow, Data structure for weighted matching and nearest common ancestors with linking", *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, 434-443, (1990).

[24] H. N. Gabow, M. X. Goemans, and D. P. Williamson, An efficient approximate algorithm for the survivable network design problems, to appear in *Proc. of the Third Conference on Integer Programming and Combinatorial Optimization.*

[25] H. N. Gabow and R. E. Tarjan, Faster scaling algorithms for general graph matching problems, Tech. Report CU-CS-432-89, Univ. of Colorado, Boulder, (1989).

[26] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., San Francisco, (1979).

[27] M. X. Goemans and D. P. Williamson, A general approximation technique for constrained forest problems, *Proc. Third Annual ACM-SIAM Symposium on Discrete Algorithms*, 307-316, (1992).

[28] M. D. Grigoriadis and B.Kalantari, A lower bound to the complexity of Euclidean and rectilinear matching algorithms, *Information Processing Letters*, (22), 73-76, (1986).

[29] M. D. Grigoriadis and B. Kalantari, A new class of heuristic algorithms for weighted perfect matching, *JACM*, 769-776, (1988).

[30] M. D. Grigoriadis, B. Kalantari and C. Y. Lai, On existence of weak greedy matching heuristics, *Operations Research Letters*, (33), 243-259, (1985).

[31] C. Imielinska and B. Kalantari, A generalized hypergreedy algorithm for perfect matching, Tech. Rep. LCSR-TR-170, Rutgers University. To appear in BIT.

[32] C. Imielinska and B. Kalantari, A general class of heuristics for minimum perfect matching and a $\log_3 \log_3 n$-error special case in $O(n^2 \log\log n)$ time, Tech. Rep. LCSR-TR-181, Rutgers University, (submitted to *Algorithmica*).

[33] C. Imielinska, B. Kalantari and L. Khachiyan, A Greedy Heuristic for a minimum-weight forest problem, to appear in *Operations Research Letters.*

[34] M. Iri, K. Murota, and S. Matsui, Linear-time approximation algorithms for finding the minimum weight perfect matching on a plane, *Information Processing Letters*, (12), 206-209, (1981).

[35] M. Iri, K. Murota, and S. Matsui, Linear-time approximation algorithms for finding the minimum weight perfect matching on a plane with an application to the plotter algorithm, *Networks*, (13), 67-92, (1983).

[36] R. M. Karp and A. Widgerson, A fast parallel algorithm for weighted perfect matching, *STOC*, 266-272, (1984).

[37] J. M. Keil and C. A. Gutwin, Classes of graphs which approximate the complete Euclidean graphs, *Discrete and Computational Geometry*, (7), 13-28, (1992).

[38] J. M. Kurtzengberg, On approximation methods for the assignment problems, *JACM*, (9), 419-439, (1962).

[39] C. W. Lai, A heuristic for the assignment problem and related bounds, Tech. Report 81.20, McGill University, Montreal, (1981).

[40] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, (1976).

[41] O. Marcote and S. Suri, Fast matching algorithm for points on a polygon, *SIAM J. of Computing*, 20 (3), 405-422, (1991).

[42] C. N. K. Osiakwan, Parallel computation of weighted matchings in graphs, Ph.D. thesis, Department of Computing and Information Science, Queen's University, Kingston, Ontario, 1991.

[43] C. H. Papadimitriou, in: G. Cornuéjols and W. Pulleyblank, A matching problem with side constraints, *Discr. Math.*, 29, 135-159, (1980).

[44] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, New Jersey, (1982).

[45] D. A. Plaisted, Heuristic matching for graphs satisfying the triangle inequality, *J.Algorithms*, 5(5), 163-179, (1984).

[46] F. P. Preparata and K. J. Shamos, *Computational Geometry*, Springer-Verlag, New York, (1985).

[47] E. M. Reingold and R. E. Tarjan, On greedy heuristic for complete matching, *SIAM J.Computing* (10), 676-681, (1981).

[48] Y. Shiloah and U. Vishkin, Finding the maximum, merging and sorting in parallel, *J. of Algorithms*, (2), 88-102, (1981).

[49] Y. Shiloah and U. Vishkin, An O(nlogn) parallel connectivity algorithm, *J. of Algorithms*, (3), 57-67, (1982).

[50] K. J. Supowit, D. A. Plaisted, and E. M. Reingold, Heuristic for weighted matching, *STOC*, 398-419, (1980).

[51] P. Vaidya, Geometry Helps in Matching, (extended abstract), *STOC*, 422-425, (1988).

[52] P. M. Vaidya, An optimal algorithm for the all-nearest-neighbor problems, *FOCS*, 117-122, (1986).

[53] P. M. Vaidya, Approximate minimum weight matching on points in k-dimensional space, *Algorithmica*, (4), 569-583, (1989).

[54] L. G. Valiant, Parallelism in comparison problems, *SIAM Journal on Computing*, (4), 348-355, (1975).

[55] O. Vornberger, Complexity of path problems in graphs, Ph.D. Thesis, Universität-GH-Paderborn, (1979).

[56] M. Werman, S. Peleg and R. Melter, Bipartite graph matching for points on a line or a circle, Tech. Report, University of Maryland, (1984).

Vita

Celina Z. Imielinska

1980 M.S. in Electrical Engineering, Technical University of Gdansk.

1980-1981 Research and Teaching Assistant, Technical University of Gdansk, Gdansk, Poland.

1981-1982 Research Assistant, Institute of Computer Science, Polish Academy of Sciences, Warsaw Poland.

1982-1983 Teaching assistant, McGill University, Montreal Canada.

1983-1987 Teaching Assistant, Rutgers University.

1985 M.S. in Computer Science, Rutgers University.

1987-1989 Research Associate, General Robotics and Active Sensory Perception Laboratory, School of Engineering and Applied Science, University of Pennsylvania.

1991- present Assistant Professor, Stevens Institute of Technology.

1993 Ph.D. in Computer Science.