© 2020

Marta Sofia Pimentel Cavaleiro

ALL RIGHTS RESERVED

SIMPLEX-LIKE METHODS FOR SPHERICAL ENCLOSURE OF POINTS AND SPHERES - ALGORITHMS AND APPLICATIONS

BY

MARTA SOFIA PIMENTEL CAVALEIRO

A dissertation submitted to the
School of Graduate Studies
Rutgers, The State University of New Jersey
In partial fulfillment of the requirements
For the degree of
Doctor of Philosophy
Graduate Program in Operations Research
Written under the direction of
Farid Alizadeh
And approved by

_____

_____

_____

_____

_____

_____

New Brunswick, New Jersey

January, 2020

ABSTRACT OF THE DISSERTATION

Simplex-like Methods for Spherical Enclosure of Points and Spheres - Algorithms and Applications

By Marta Sofia Pimentel Cavaleiro

Dissertation Director: Farid Alizadeh

Given a set of $\mathfrak{m}$ points in the Euclidean space $\mathbb{R}^{\mathfrak{n}}$, the problem of the *minimum enclosing ball of points*, "MB problem of points" for short, seeks the $\mathfrak{n}$-dimensional Euclidean hypersphere (ball) of smallest radius that encloses all points in the set. A generalization of this problem is the *minimum enclosing ball of balls*, also referred as the "MB problem of balls", where, instead of enclosing points, one wishes to enclose a set of given balls. Both are important problems in computational geometry and optimization, with applications in facility location, computer graphics, machine learning, etc.

The MB problem of points/balls shares some combinatorial properties with linear programs (LP), which makes it part of the class of *LP-type problems*. Methods that resemble the simplex method for LP (simplex-like methods) have been proposed to solve the MB problem of points. However, no simplex-like method has been developed for the case of balls.

We start by considering one of such simplex-like methods for the MB problem of points, the dual algorithm by Dearing and Zeck [33]. We modify its main step, the directional search procedure, improving its computational complexity from $\mathcal{O}(\mathfrak{n}^3)$ to $\mathcal{O}(\mathfrak{n}^2)$. We show that this modification yields in practice faster running times as we increase the dimension.

Next, we consider the problem of *partial enclosure*: instead of enclosing all $\mathtt{m}$ given points, we want to enclose at least $\mathtt{k}$ with the smallest radius ball. This problem is called the *minimum* $\mathtt{k}$*-enclosing ball problem*, or "$\mathrm{MB}_{\mathtt{k}}$ problem" for short, and it is NP-hard. We present a branch-and-bound (B&B) algorithm on the tree of the subsets with size $\mathtt{k}$ to solve this problem. The nodes on the tree are ordered in a suitable way, which, complemented with a last-in-first-out search strategy, allows for only a small fraction of nodes to be explored. We use the dual simplex-like algorithm by Dearing and Zeck to solve the subproblem at each node of the search tree. We show that our B&B is able to solve the $\mathrm{MB}_{\mathtt{k}}$ problem exactly in a short amount of time for small and medium sized datasets.

Finally, we address the MB problem of balls, a second-order cone program, and propose a dual simplex-like algorithm to solve it. At each iteration of the algorithm, the next iterate is found by performing an exact search on a well-defined curve. The algorithm can be seen as an extension to the case of enclosing balls of the algorithm by Dearing and Zeck. The algorithm's implementation is based on the Cholesky factorization. Our computational results show that the algorithm is very efficient in solving even large instances.

Acknowledgements

First and foremost, I would like to thank my research advisor and mentor, Prof. Farid Alizadeh, for his persistent support and intellectual guidance. I am truly grateful for the freedom he has given me to pursue my ideas, which he has always respected, at my own pace. It has been a privilege working with him.

Next, I would like to express my appreciation and thanks to Prof. Adi Ben-Israel, Prof. Endre Boros, Prof. Jonathan Eckstein, and Prof. Fatma Kılınç-Karzan, for kindly agreeing to be in my defense committee and for their insightful feedback about my work.

I am immensely grateful for the opportunity to pursue my doctoral studies in Operations Research at Rutgers University. I am lucky to have learned with exceptional professors at Rutgers, especially from RUTCOR. All of them have contributed somehow to this dissertation.

Finally, a special thanks to those in my inner circle: to my family, in particular my parents; to my partner Gianluca; and to all my friends, especially Tó-Zé. Thank you for your encouragement and love.

Dedication

To my parents, José and Licínia.

Table of Contents

List of Notations

$\|x\|$	Euclidean norm of vector $x \in \mathbb{R}^n$
$\langle x, y \rangle$	inner product in $\mathbb{R}^n$ of vectors $x, y \in \mathbb{R}^n$
$[a, b]$	closed interval in $\mathbb{R}$
$]a, b[$	open interval in $\mathbb{R}$
$B(c, r)$	n-dimensional hypershpere $\{x \in \mathbb{R}^n : \|x - c\| \leq r\}$
I_n	identity matrix in $\mathbb{R}^{n \times n}$
$e_j \in \mathbb{R}^n$	j-th column of I_n
1_n	n-dimensional vector with all entries 1
$\mathrm{Span}(A)$	column range of matrix A
$\mathrm{Null}(A)$	null space of matrix A
$\emptyset$	empty set
$\|\mathcal{S}\|$	cardinality of set $\mathcal{S}$
$2^{\mathcal{S}}$	set of all subsets of $\mathcal{S}$
$\mathrm{conv}(\mathcal{S})$	convex hull of set $\mathcal{S}$
$\mathrm{aff}(\mathcal{S})$	affine hull of set $\mathcal{S}$
$\mathrm{int}\,\mathcal{S}$	interior of set $\mathcal{S}$
$\mathrm{ri}\,\mathcal{S}$	relative interior of set $\mathcal{S}$
$\partial\mathcal{S}$	boundary of set $\mathcal{S}$
$\mathcal{Q}$	Second-order cone $\{x := (x_0; \bar{x}) \in \mathbb{R}^n : \|\bar{x}\|_2 \leq x_0\}$
$x \succeq_{\mathcal{Q}} y$	$x - y \in \mathcal{Q}$
$x \succ_{\mathcal{Q}} y$	$x - y \in \mathrm{int}(\mathcal{Q})$

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation revolves around the analysis, development, and application of *simplex-like* methods for spherical enclosure problems of points and spheres (informally referred to as balls). This introductory chapter presents some background concepts, reviews relevant work on the topic, and outlines the novel contributions of this dissertation.

1.1 Overview

1.1.1 The minimum enclosing ball of points and balls

The *minimum enclosing ball (MB) problem* can be formulated as follows: given a set of $\mathfrak{m}$ objects in the Euclidean space $\mathbb{R}^{\mathfrak{n}}$, find the $\mathfrak{n}$-dimensional hypersphere (ball) with the smallest radius that encloses all objects in the set. The scope of this dissertation is concerned with the case when the objects are either points or balls. The minimum enclosing ball of points can be seen as a particular case of the minimum enclosing ball of balls where all given balls have a radius equal to zero.

The problem of enclosing a given set of points/balls with a minimum radius ball is an important and active problem in computational geometry and optimization. The problem is also referred in the literature as *minimum covering ball*, *smallest enclosing disk*, *minimum spanning ball*, *minmax location*, *Euclidean* 1*-center*, *optimal bounding sphere*, etc.

Applications include facility location [54, 75, 87]; computer graphics [59, 66]; and industrial manufacturing [39, 58]. Further typically high-dimensional applications arise in machine learning, and involve, e.g., support vector machines [22, 85, 102], clustering [12, 18], and farthest-neighbor approximation [49]. In all of these applications, when the data are not spherically-shaped, one may apply a linear transformation to it in an attempt to make it spherically-shaped, and then enclose it with a sphere. That can be accomplished by calculating the spectral decomposition of the covariance matrix of the data, and use it to rotate and rescale the data.

Combinatorial nature

It is well known that the minimum enclosing ball of a set of points is unique and defined by at least 2 and at most $n+1$ points lying on the ball's boundary. Thus, a *brute-force* computation of the optimal ball can be performed by considering all possible subsets of 2, 3,..., and $n+1$ points, and computing the smallest radius ball that passes through the points of each subset, when it exists. Finding such ball boils down to a linear system of equations which is inconsistent when such ball does not exist. Of course, such a brute-force method is prohibitively slow.

Similarly, the minimum enclosing ball of balls is defined by at least 1 and at most $n+1$ balls tangent to the boundary of the optimal enclosing ball. The problem of finding the smallest ball that is tangent to and encloses $k \leq n$ balls, instead, boils down to a relatively simple nonlinear system of equations.

Early work

The minimum enclosing ball problem of points has been studied since as early as 1857 when Sylvester studied the planar case [98]. An interesting account of the work that followed Sylvester is presented in a paper from 1941 by Blumenthal and Wahlin [14].

Much of the early work on the MB problem of points focused on the two or three-dimensional case [68, 11, 37, 93, 60, 13, 96]. A breakthrough occurred in 1983 when Megiddo gave the first deterministic algorithm based on a prune-and-search technique that solves the MB problem in $O(m)$ time when the dimension n is fixed [73]. This result has more

of a theoretical meaning since the actual implementation of the method is impractical. Dyer [34] further improved Megiddo's technique, and later, Megiddo extended it to the case of enclosing balls instead of points [74].

In 1991, Welzl proposed a simple randomized algorithm for the minimum enclosing ball of points with expected running time $\mathcal{O}(\mathfrak{n}\mathfrak{n}!\mathfrak{m})$ [104]. Gärtner later improved Welzl's algorithm [44], but it remained practical only for small dimensions ($\mathfrak{n} < 30$). Fischer and Gärtner proved that Welzl's algorithm cannot be generalized to the case of the minimum enclosing ball of balls, except when the centers of the given balls are affinely independent [41]. After Welzl's paper, some of the subsequent advances on the MB problem were accomplished under the umbrella of a more general abstract framework of optimization problems called *LP-type problems*.

LP - type problems

Introduced by Sharir and Welzl in [94], the abstract framework of *LP-type problems* is a class of problems that captures some combinatorial properties of linear programing. An LP-type problem is given by a pair $(\mathcal{H}, w)$, where $\mathcal{H}$ is the finite set representing the constraints, and w is a function, $w : 2^{\mathcal{H}} \to \mathcal{W}$, for a totally ordered[1] set $(\mathcal{W}, \leq)$ with minimal element $-\infty$. For every subset $\mathcal{G} \subseteq \mathcal{H}$, $w(\mathcal{G})$ is denoted by the *value of* $\mathcal{G}$ and corresponds to the minimum value of the objective function when subject to the constraints in $\mathcal{G}$. Such a pair $(\mathcal{H}, w)$ is an LP-type problem if it satisfies the *monotonicity* and *locality* axioms:

Monotonicity axiom: For any $\mathcal{F}, \mathcal{G}$ with $\mathcal{F} \subseteq \mathcal{G} \subseteq \mathcal{H}$, we have $w(\mathcal{F}) \leq w(\mathcal{G})$.

Locality axiom: For any $\mathcal{F} \subseteq \mathcal{G} \subseteq \mathcal{H}$ with $-\infty \neq w(\mathcal{F}) = w(\mathcal{G})$ and any $\mathfrak{h} \in \mathcal{H}$, $w(\mathcal{G}) < w(\mathcal{G} \cup \{\mathfrak{h}\})$ implies $w(\mathcal{F}) < w(\mathcal{F} \cup \{\mathfrak{h}\})$.

The goal of an LP-type problem is to compute an inclusion-minimal[2] subset $\mathcal{B} \subseteq \mathcal{H}$ such that $w(\mathcal{B}) = w(\mathcal{H})$, that is, $\mathcal{B}$ is a *basis of* $\mathcal{H}$. In other words, $\mathcal{B}$ is a basis in the LP-type sense if it contains no elements that are redundant for determining the optimum solution of

[1] A binary relation $\leq$ is a total order (or linear order) on $\mathcal{W}$ if the following three properties hold for any elements $u, v, w \in \mathcal{W}$: if $u \leq v$ and $v \leq u$ then $u = v$ *(antisymmetry)*; if $u \leq v$ and $v \leq w$ then $u \leq w$ *(transitivity)*; either $u \leq v$ or $v \leq u$ *(connexity)*.

[2] A set is inclusion-minimal with respect to some property if it fulfills the property but none of its subsets does.

minimizing the objective function subject to $\mathcal{B}$. The *combinatorial dimension* of an LP-type problem is the maximum cardinality of a basis of any subset of $\mathcal{H}$.

It turns out that the MB problem of balls (and thus of points) is an LP-type problem [41, 69]. It is easy to see that both axioms hold for $(\mathcal{H}, w)$, with $\mathcal{H}$ the set of balls to be enclosed, and $w(\mathcal{F})$ the radius of the smallest ball that encloses $\mathcal{F} \subseteq \mathcal{H}$ assuming $w(\emptyset) = -\infty$ (thus $\mathcal{W} = \mathbb{R}_0^+ \cup \{-\infty\}$). The combinatorial dimension of the MB problem is $n + 1$ for both the case of points and balls.

Other examples of LP-type problems are finding the minimum distance between two closed polytopes, computing the ellipsoid with the smallest volume containing a given set of points, or computing the ball with the smallest radius that intersects a given set of closed convex objects. For more examples, see [46, 69]. As the name suggests, linear programs can be formulated as an LP-type problem [35].

The algorithms for LP-type problems rely on the following two primitive operations:

Violation test: Given a basis $\mathcal{B} \subseteq \mathcal{H}$ and a constraint $h \in \mathcal{H}$, determine whether $w(\mathcal{B}) < w(\mathcal{B} \cup \{h\})$.

Basis computation: Given a basis $\mathcal{B} \subseteq \mathcal{H}$ and a constraint $h \in \mathcal{H}$, find a basis of $\mathcal{B} \cup \{h\}$.

Once an optimization problem is known to be an LP-type problem and an implementation of the two primitive operations is available, several algorithms can be applied to solve the problem. Examples of such algorithms are the randomized algorithms by Sharir and Welzl [94], Clarkson [26], and Matoušek et al. [69, 71], or the derandomized version of Clarkson's algorithm by Chazelle and Matoušek [23]. Of theoretical relevance is the result that any LP-type of combinatorial dimension δ with size $m = |\mathcal{H}|$ can be solved with an expected number of $\mathcal{O}(\delta m) + e^{\mathcal{O}(\sqrt{\delta \log \delta})}$ primitive operations, provided some initial basis is available [46]. Thus, if the primitive operations can be implemented in polynomial time (or subexponential) in δ and m, then the problem can be solved in an expected subexponential time.

In terms of the MB problem, the violation test is an easy task. However, the basis computation is not trivial. An exhaustive search is always an (undesirable) possibility.

Gärtner showed in [43] how the basis computation can be done in subexponential time for the case of the MB problem of points. A more practical approach was given in [47], by presenting a randomized algorithm with expected complexity $\mathcal{O}(1.5^n)$. This bound could be further improved by transforming the problem into a *unique sink orientation (USO)*[3], which can be solved in an expected $\mathcal{O}(1.47^n)$ time [99].

For the case of the MB of balls, Fischer and Gärtner showed in 2004 that Welzl's algorithm [104] could be used to solve the basis computation operation by embedding the balls in $\mathbb{R}^{n+1}$ and perturbing the centers so they are affinely independent [41]. This option, however, requires an exponential amount of time. In the same paper, they also show how one can reduce the basis computation to a USO problem.

In 2009, in the context of a dual simplex-like algorithm for the MB of points, Dearing and Zeck presented a polynomial method to perform the basis computation [33]. We discuss this algorithm in Chapter 2, where we also present an improvement of the method. In Chapter 4 we present an algorithm that can be seen as a generalization of Dearing and Zeck's for the case of the MB of balls, which as a byproduct also yields a polynomial-time basis computation for this problem.

For a comprehensive survey and historic account on LP-type problems we refer the interested reader to [35].

Convex programming approach

The MB problem for points can easily be converted to a quadratic program (QP) that is convex. Quadratic programs are typically solved by *interior-point methods* or *active set methods*. If the Hessian of the objective function and/or the constraint matrix of the QP is large and sparse, then an interior-point method is usually the method of choice. For small to medium-sized problems with dense matrices, active set methods are usually preferable.

[3] A unique sink orientation (USO) is an orientation of the edges of a hypercube such that every face of the cube has in its induced subgraph a unique vertex with no outgoing edges *(sink)*. The USO problem consists of finding the sink of the induced graph.

Active-set methods are iterative methods that solve a sequence of equality-constrained quadratic subproblems, with the goal of predicting the *active set*, that is, the set of active constraints at the optimal solution. Active-set methods generally require a large number of iterations in which each search direction is relatively inexpensive to compute since it involves only a subset of the variables of the problem. The most famous active set method in the optimization literature is the *Simplex method* for linear programs (LP), introduced in 1947 by Dantzig [29]. There are several extensions of the Simplex method to solve QPs, see for instance [105, 103, 51, 45]. In particular, the method of [45] was developed for solving QPs that arise from problems in computational geometry, such as the MB of points. The methods that Chapters 2 and 4 revolve around can also be considered to belong to the class of active set methods.

The MB problem for balls is not known to be a QP, but it is a second-order cone program (SOCP). Thus its solution usually relies on interior-point methods, such as the *primal-dual path following algorithms*, [82, 83, 84]. Interior-point algorithms generate interior-points for the primal and dual problems that follow the so-called *central path*, which converges to a primal-dual optimal solution in the limit. Each interior-point iteration consists of solving a linear system resulting from the application of Newton's method to the KKT conditions of the modified problem with a *logarithmic barrier function*. In terms of the MB problem, each interior-point iteration has $\mathcal{O}(mn^2+n^3)$ computational complexity [65, 110], and an interior point method would be guaranteed to get an approximate solution with an error of at most ϵ in $\mathcal{O}(\sqrt{n}\log\frac{1}{\epsilon})$ iterations. In practice, the number of iterations in interior-point methods is often very small and independent of the problem dimension.

Simplex-like methods

Consider a linear program in *standard form*

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0, \end{aligned}$$

such that $A \in \mathbb{R}^{m \times n}$, and its dual

$$\begin{aligned} \min_{y \in \mathbb{R}^m} \quad & b^T y \\ \text{s.t.} \quad & A^T y \leq c. \end{aligned}$$

A *basis*, in the context of linear programming, is a subset of m linearly independent columns of A. Let $\mathcal{B}$ be the indexes of the columns of a basis and B a matrix containing those columns. A *basic solution* is a vector x such that $x_j = (B^{-1}b)_j$, for $j \in \mathcal{B}$ (these are the basic variables), and $x_j = 0$ for $j \notin \mathcal{B}$ (the non-basic variables). A basic solution that is (primal) feasible ($x \geq 0$), is called a *basic feasible solution (b.f.s.)*. A *dual basic solution (d.b.s.)* is a vector y such that $B^T y = c$. If y is feasible for the dual problem ($A^T y \leq c$), then it is called a *dual basic feasible solution (d.b.f.s.)*.

Now, we briefly explain how a linear program can be solved by the Simplex method, focusing on an interpretation of the algorithm's mechanics that easily allow it to be extended to other problems. For the sake of simplicity, we disregard issues such as unboundedness or degeneracy. For a detailed discussion on the Simplex method for LP, see e.g., [25].

At each iteration, called *pivot*, the Simplex method moves from one basic solution to another basic solution, by replacing a basic variable by a non-basic variable. The primal variant of the Simplex method does so while maintaining primal feasibility and decreasing the value of the objective function at each iteration. An iteration of the primal Simplex method is composed of two main parts: *pricing*, which either certifies that the current b.f.s. is optimal by checking whether the corresponding d.b.s. is dual feasible, or selects a non-basic variable corresponding to an infeasible dual constraint to become basic; and the *ratio test*, which basically consists in solving the problem of moving as far as possible in the direction that maintains primal feasibility, keeps the non-basic variables zero, and decreases the value of the objective function. When we consider moving in that direction, the search stops when one of the basic variables becomes zero. Such a variable becomes non-basic, and a new iteration starts with the new b.f.s..

The dual Simplex method works analogously. It starts with a d.b.f.s. and keeps dual feasibility throughout the iterations. Each iteration starts with the *dual pricing* phase,

which either certifies optimality by checking primal feasibility of the primal basic solution corresponding to the current d.b.f.s., or selects a basic (primal) variable that has negative value to become non-basic. The *ratio test* solves the following problem: how to move and how far in the direction that keeps all dual constraints feasible, in particular, keeps the dual constraints corresponding to basic variables active, and increases the value of the objective function (of the primal). The non-basic variable that corresponds to the dual constraint that first becomes active as we move in that direction is the one to become basic. After the ratio test, a new iteration starts.

A *simplex-like* algorithm is an algorithm that in its essence behaves like the Simplex methods described above. As we have mentioned, there are several simplex-like methods for QP, e.g., [105, 103, 51, 45]. Contrarily to LP, where there is always an exchange in the basis at each iteration, in QP, that may not be the case: it can happen that a variable enters the basis, but there is no leaving variable; and it can happen that even if some leaving variable is found, the solution at that point is not basic. In this case, the pivot step continues, and more variables may leave the basis until another basic solution is discovered.

Simplex-like algorithms have been proposed for the *minimum enclosing ball of points problem*. First, Fischer and Gärtner proposed in 2003, [42], a primal algorithm for the MB problem of points based on previous ideas from [58]. Their algorithm starts with a ball that encloses all points, and at each iteration, a *pivot-like* procedure deflates the ball until it cannot shrink any more without losing coverage of a point. To avoid *cycling*, they adapt Bland's rule for the Simplex method for LP. In practice, the algorithm is shown to be able to solve large instances ($\mathfrak{n} \leq 10000$) in a practical amount of time. In Section 2.3, we present a brief overview of this algorithm. Later, in 2009, Dearing and Zeck proposed a dual simplex-like algorithm for the same problem [33]. Their algorithm starts with the smallest ball that encloses a subset of the points. At each iteration, a point that is not yet enclosed is selected, and a *pivot-like* procedure inflates the ball until the selected point is enclosed. The radius of the ball strictly increases at each iteration, so *cycling* cannot occur. This algorithm is the subject of Chapter 2, and as we will show, it can also deal with large dimensions efficiently. The algorithms in [42] and [33] have been extended to the problem of

solving the *weighted minimum enclosing ball of points* in [31] and [32], respectively.

To the best of our knowledge, no simplex-like algorithm, other than the one introduced in this dissertation (Chapter 4), has been proposed to solve the problem of the *minimum enclosing ball of balls*.

Besides providing an exact solution, an advantage of simplex-like methods over interior-point methods is that the former generates *basic solutions*. In the case of the MB problem, that means knowing which input points (or balls) determine the optimal enclosing ball, which can be of importance for some applications. Additionally, if, after solving the problem, the problem undergoes small changes (e.g., a new constraint is added), a simplex-like method will often require a small number of iterations to calculate the new solution when it starts from a solution to the original problem. One situation of this type is, for instance, branch-and-bound algorithms for solving integer programs, where each node in the branch-and-bound tree requires the solution of a linear program that differs only slightly from the one already solved in the parent node. However, we do not have any polynomial complexity guarantees for simplex-like algorithms as we have for interior-point methods.

Approximation algorithms

So far, we have only mentioned *exact algorithms* to solve the MB problem of points/balls (we will restrict ourselves to such methods in this dissertation). However, several *approximation algorithms* have also been proposed in the literature. As an example, we have the algorithms for the MB of points of Ritter [89] and of Wu [106]. In particular, approximation algorithms based on the concept of ϵ-*core-sets* have received much attention. A ϵ-*core-set* is a subset of the input points whose minimum enclosing ball, when scaled by a factor of $1+\epsilon$, is guaranteed to enclose the entire input point set [17]. A remarkable property is the fact that any point set admits a ϵ-core-set of size $\lceil \frac{1}{\epsilon} \rceil$ (independent of the dimension) [15, 16]. Several $(1+\epsilon)$-approximation algorithms focused on finding ϵ-core-sets have been proposed, e.g. [17, 65, 1, 107, 67], being in general able to deal with large dimensions in a practical amount of time. The algorithms in this family with the best theoretical complexity take $\mathcal{O}(\frac{mn}{\epsilon})$ time [86, 107]. The results and algorithms involving ϵ-core-sets are also valid for the

case of enclosing balls.

Approximation algorithms based on the SOCP formulation of the minimum enclosing ball of points/balls have also been considered. In [65], Kumar et al. use ϵ-core-sets on top of the SOCP formulation to obtain an approximation algorithm. In [110] instead, Zhou et al. propose an interior-point algorithm that exploits the particular structure of the matrix necessary to calculate the search direction at each iteration. In both cases, large-dimensional problems can be handled. In [110], the authors present another method based on a smooth approximation of the non-differentiable minimax formulation[4] of the MB problem.

More recently, $(1+\epsilon)$-approximation algorithms were proposed by Saha et al. [92] and Allen-Zhu et al. [9] which achieve time complexity of $\mathcal{O}(C\frac{mn}{\sqrt{\epsilon}})$ (C is an upper bound on the norm of the input points) and $\mathcal{O}(mn + \frac{m\sqrt{n}}{\sqrt{\epsilon}})$, respectively.

Finally, streaming algorithms, that is, algorithms that only allow one pass over the input data, have also been studied for the MB of points. Zarrabi-Zadeh and Chan [109] gave a 1.5-approximation streaming algorithm, and later an algorithm by Agarwal and Sharathkumar [3] was able to improve the approximation factor to 1.22 [21].

Filtering heuristics

Heuristics that identify points on the interior of the optimal enclosing ball of points have been proposed in the literature, using various techniques [5, 63, 88]. With these heuristics, one can discard some points from the input set, possibly allowing a faster solution of the MB problem of points.

The minimum-volume enclosing ellipsoid

A generalization of the MB problem of points is the problem of enclosing the points with a minimum-volume ellipsoid instead of a minimum-radius ball. Minimum-volume enclosing ellipsoids play an important role in several applications such as optimal design, computational geometry, statistics, etc. For a comprehensive study of this problem, we refer the reader to [100].

[4] See Chapter 2.

1.1.2 The minimum k-enclosing ball of points

Consider a dataset $\mathcal{P}$ of m points in $\mathbb{R}^n$. In real-world applications we often need to assume the presence of outliers in the given dataset. In such cases, we may be given a fraction of the points that we either want to enclose or to leave out. The *minimum* k*-enclosing ball problem*, *MB_k problem* in short, is a natural generalization of the MB problem, where one seeks to enclose at least k of the m points with a ball with the smallest radius. The requirement to enclose only a fraction of the points with a minimum enclosing ball makes the problem not only non-convex but also highly combinatorial. The MB_k problem is NP-hard and admits no fully polynomial-time approximation scheme (FPTAS)[5] (assuming $P \neq NP$) [95].

Most of the applications of the MB_k problem are variants of the applications of the MB problem (previously described in Section 1.1.1) that are robust to outliers. Another application arises in robust regression in statistics, with the problem of minimum volume ball estimator (MVBE) [90, 28]. The MVBE is a least-median-squares-location estimator, corresponding to finding a point that minimizes the median of its Euclidean distances to points in a dataset. Thus, it consists of finding the minimum volume hypersphere that covers at least half of the data points ($k = \lceil \frac{1}{2} m \rceil$). This estimator has a high breakdown point (it can detect a large number of outliers) and can be used in the covariance estimator of multivariate data. The MVBE can be generalized to quantiles $0 < q < 1$ other than the median, yielding an MB_k problem with $k = \lceil qm \rceil$. The MVBE is a particular case of the minimum volume ellipsoid estimator.

Previous work

The initial approaches to solving the MB_k focused on the exact solution of the planar case. One approach is based on the construction of high-order Voronoi diagrams[6] and a search in some or all of its cells [4]. Efrat et al. [36] gave two algorithms that make use of the parametric search technique (due to Megiddo [72]) and that solve the problem in

[5] An $(1+\epsilon)$-approximation algorithm is an FPTAS if its running time is polynomial in both the problem size and $1/\epsilon$.

[6] Given a set of points $\mathcal{P}$ in the plane, a k-th order Voronoi diagram is a division of the plane in regions (cells). Each cell is associated with a k-subset of $\mathcal{P}$ and corresponds to the points in the plane that are closer to the points of that k-subset than to any other points.

$\mathcal{O}(mk \log^2 m)$; Matoušek developed a simple randomized search algorithm that has expected running time $\mathcal{O}(m \log m + mk)$ [70].

Eppstein and Erickson [38] developed an algorithm that works for any dimension, and that reduces the problem to $\mathcal{O}(m/k)$ subproblems of $\mathcal{O}(k)$ points each, by finding $\mathcal{O}(k)$ nearest neighbors for each point. Later, Datta et al. [30] proposes the same reduction but using techniques for closest pair problems instead of nearest neighbors. Both algorithms in [38, 30] solve the problem in $\mathcal{O}(m \log m + mk^{n-1} \log^2 k)$ time.

Approximation algorithms started to be developed more recently. Mount et al. [78] proposed a $(1+\epsilon)$-approximation algorithm that runs in $\mathcal{O}(m \log m + (\frac{1}{\epsilon})^{\mathcal{O}(n)})$, and that can be applied to other k-enclosing problems. Har-Peled and Mazumdar [55] focused on the planar case and presented a linear 2-approximation algorithm by dividing the plane in *grids*, based on which they developed both a randomized exact algorithm and a $(1+\epsilon)$-approximation algorithm for the MB_k problem. The same ideas were further explored in [56] for large dimensions, where an $(1+\epsilon)$-algorithm with expected $\mathcal{O}(m/\epsilon^n)$ time was shown as an application of a more general framework to solve several computational geometry problems. Other approximation algorithms have been developed based on the computation of a *(robust) core-set* (subset of the input point-set whose MB_k solution is a $(1+\epsilon)$-approximation of the solution of $MB_k(\mathcal{P})$) [2, 57]. These algorithms find such a subset of size $\mathcal{O}((n-k)/\epsilon^{\mathcal{O}(d)})$ in $\mathcal{O}(n+(n-k)/\epsilon^{\mathcal{O}(d)})$ time. Another approximation algorithm with $\mathcal{O}(m^{1/\epsilon^2+1}n)$ time is discussed in [95].

Variations of the MB_k problem have also been studied, such as the MB_k problem in the plane with center restricted to a given line [64], or the case when each point has a color associated and we want to find the MB_k ball that encloses at least k points of a given color [10].

Branch-and-bound methods

The *branch-and-bound (B&B)* paradigm is a widely used tool in combinatorial optimization. Hence it is a natural choice to obtain an exact solution of the MB_k problem. A B&B algorithm searches the feasible set by dividing it into disjoint subsets organized in a tree

structure. The use of bounds for the function to be optimized combined with the value of the current best solution enables the algorithm to discard parts of the feasible set, meaning only a portion of the nodes of the tree is visited. An important question in B&B is how to specify the order in which nodes are visited. An effective node ordering strategy guides the search to promising areas in the tree and improves the chance of quickly finding a good incumbent solution, which can be used to rule out other nodes.

There are several ways one could employ a B&B framework for the MB_k problem. One possibility is a B&B algorithm on the tree of all k-subsets of the input point set (subsets of size k). Chapter 3 focuses on the study of such an algorithm.

1.2 Contributions and outline of the dissertation

This dissertation focuses on the study, development, and application of *simplex-like* methods for the minimum enclosing ball of points and of balls.

Chapter 2 addresses the dual simplex-like algorithm proposed by Dearing and Zeck to solve the MB problem of points [33]. The main contribution of Chapter 2 is a modification of the algorithm's directional search that makes it computationally more efficient. This modification, together with an implementation that uses efficient updates of the QR factorization of a particular matrix, yields a directional search with $\mathcal{O}(n^2)$ complexity, where n is the dimension of the points from the input set. This is an improvement with respect to the original algorithm's directional search procedure, which could not achieve a complexity better than $\mathcal{O}(n^3)$. We conclude with some computational results that show that the modification we propose decreases the running time considerably when we increase the dimension. The novel contributions presented in this chapter are published in [20].

Chapter 3 covers an application of the algorithm studied in Chapter 2. In this chapter, we consider the problem of the minimum k-enclosing ball of points. Our main contribution consists in a branch-and-bound algorithm that builds the tree of all k-subsets of the input point set, that is, subsets of size k, and that uses the algorithm by Dearing and Zeck [33] to solve the MB problem of points in each node. The nodes on the tree are ordered in a

suitable way, which, complemented with a last-in-first-out search strategy, allows for only a small fraction of nodes to be explored. We study several aspects of the algorithm and present a thorough analysis of its computational performance for different datasets.

Chapter 4 covers the most significant contribution of this dissertation. In this chapter, we address the MB problem of balls and present a dual simplex-like algorithm to solve it. To the best of our knowledge, this is the first simplex-like algorithm to be developed for this problem. We consider the SOCP formulation of the MB problem of balls and, since other geometric problems, like the *smallest intersecting ball of a set of balls*, also share the same formulation, we refer to it as the problem of finding the *infimum with respect to the Second-Order cone* $\mathcal{Q}$, or $\text{Inf}_{\mathcal{Q}}$ problem for short. We start by studying the $\text{Inf}_{\mathcal{Q}}$ problem in depth, presenting many of its properties and laying the groundwork for the algorithm. Then, we introduce a dual simplex-like algorithm to solve the $\text{Inf}_{\mathcal{Q}}$ problem, whose main feature is a pivot-like step that consists of an exact curve search. We also describe how the algorithm can be efficiently implemented by keeping the Cholesky factorization of a particular matrix, and present computational results that show that our algorithm can handle even large datasets in practical time.

Chapter 2

The minimum enclosing ball of points

Given a set $\mathcal{P} = \{p_1, \ldots, p_m\}$ of distinct points in $\mathbb{R}^n$, the problem of the *minimum enclosing ball of points* (*MB problem* for short or simply $\mathsf{MB}(\mathcal{P})$), consists on determining the ball of smallest radius that encloses all points in $\mathcal{P}$. Figure 2.1 illustrates the problem in $\mathbb{R}^2$. Considering x and r, the center and radius, respectively, the MB problem of points can be formulated as

$$\mathsf{MB}(\mathcal{P}): \quad \begin{aligned} \min_{x \in \mathbb{R}^n,\, r \in \mathbb{R}} \quad & r \\ \text{s.t.} \quad & \|p_i - x\| \leq r, \quad \forall p_i \in \mathcal{P}. \end{aligned} \tag{2.1}$$

One can also use the equivalent *min-max* formulation:

$$\mathsf{MB}(\mathcal{P}): \quad \min_{x \in \mathbb{R}^n} \max_{p_i \in \mathcal{P}} \|p_i - x\|.$$

This chapter's main contribution is a modification of the dual simplex-like algorithm proposed by Dearing and Zeck [33] to solve the MB problem of points. Each iteration of the original algorithm performs a sequence of directional searches. Each one of these directional searches consists, among other things, in finding the intersection of a line with $\mathcal{O}(n)$ hyperplanes. This results in a directional search procedure with computational complexity no better than $\mathcal{O}(n^3)$. We modify the directional search procedure in such a way that, together

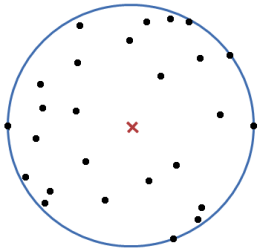


Figure 2.1: Illustration of the minimum enclosing ball problem in $\mathbb{R}^2$.

with an implementation using QR updates, allows it to be done in $\mathcal{O}(n^2)$ time. The results shown in this chapter are published in [20].

This chapter is organized as follows. Section 2.1 presents some essential background about the problem such as optimality conditions and duality. Section 2.2 summarizes the algorithm that is the subject of our work, the dual simplex-like algorithm by Dearing and Zeck [33]. For the sake of completeness, we briefly describe the primal simplex-like method for the MB problem proposed by Fischer, Gärtner, and Kutz [42], in Section 2.3. Section 2.4 introduces a modification to the algorithm by Dearing and Zeck that improves its running time, and Section 2.5 presents the details of how one can implement the new algorithm. Finally, Section 2.6 gives some computational results and concludes with the practical impact of our work.

In this chapter, whenever we refer to the minimum enclosing ball problem or the MB problem, we are always referring to the case of enclosing points.

2.1 Properties of the MB problem

This section briefly presents some properties of the MB problem of points that are relevant to the subsequent sections. We omit many details since an extensive study of the same properties for the more general case of the MB problem of balls is presented in Section 4.2.

Problem (2.1) can easily be transformed into a quadratic program (QP), by squaring

the objective function and both sides of the constraints, and considering $w = r^2 - x^T x$:

$$\begin{aligned} \min_{x \in \mathbb{R}^n, w \in \mathbb{R}} \quad & x^T x + w \\ \text{s.t.} \quad & 2p_i^T x + w \geq p_i^T p_i, \quad \forall\, p_i \in \mathcal{P}. \end{aligned}$$

The Hessian of the objective function is positive semi-definite, we have a *convex problem*. Although the MB problem is not *strictly convex*, its solution is unique. For proof of this claim, we refer the reader to Theorem 4.3 from Chapter 4, where the uniqueness of the solution of a more general problem is proved.

Optimality conditions and geometric interpretation

It is easy to see that *Slater's constraint qualification condition* is satisfied. To obtain a strictly feasible solution for (2.1) simply consider $x = p_k$ for any $p_k \in \mathcal{P}$, and $r = \max_{j=1,\ldots,m} \|p_j - p_k\| + \epsilon$, for any $\epsilon > 0$. This, together with the fact that problem (2.1) is convex, implies that the the optimal solution satisfies the Karush-Kuhn-Tucker conditions and these are also sufficient (see for example [91, §3]).

Theorem 2.1 (Karush-Kuhn-Tucker conditions for the MB problem)**.** *Let* x^* *and* r^* *be feasible for problem (2.1). Then* x^* *and* r^* *constitute the optimal solution to (2.1) if and only if there exists* Lagrange multipliers $\lambda^* \in \mathbb{R}^m$ *such that*

$$x^* = \sum_{i=1}^{m} \lambda_i^* p_i, \quad \sum_{i=1}^{m} \lambda_i^* = 1, \quad \lambda_i^* \geq 0, \quad i = 1, \ldots, m,$$

and the complementary slackness conditions *are satisfied*

$$\lambda_i^* \left((r^*)^2 - \|p_i - x^*\|^2 \right) = 0, \quad i = 1, \ldots, m.$$

The complementary slackness conditions imply that the Lagrange multipliers corresponding to points that are not on the boundary of the optimal ball must be zero. Therefore, we can conclude that the center of the optimal ball is a convex combination of the points on the boundary of the optimal ball. Theorem 2.1 can then be rephrased in geometric terms (Corollary 2.2).

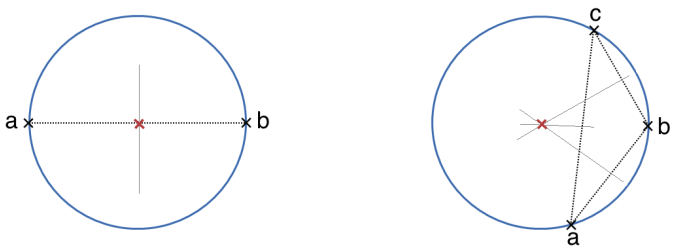


Figure 2.2: Illustration in $\mathbb{R}^2$ of the unique ball with center in the affine space of a set of affinely independent points on its boundary. The lines represent the bisectors.

Corollary 2.2 (Geometric interpretation of the KKT conditions)**.** *Consider a ball* $B(x^*, r^*)$ *with center at* x^* *and radius* r^* *that encloses all points in* $\mathcal{P}$*. Let* $\mathcal{S} \subseteq \mathcal{P}$ *be the set of points on the boundary of* $B(x^*, r^*)$*. The ball* $B(x^*, r^*)$ *is the minimum enclosing ball of* $\mathcal{P}$ *if and only if* $x^* \in \text{conv}(\mathcal{S})$.

Let $\mathcal{S} := \{p_1, ..., p_s\}$ be the set of points on the boundary of ball $B(x, r)$. Then, the center x must satisfy

$$\|x - p_i\|^2 = \|x - p_1\|^2, \quad \forall p_i \in \mathcal{S} \setminus \{p_1\}. \tag{2.2}$$

These conditions are easily transformed in linear equations that define hyperplanes often denoted as *bisectors*. Given, p_i and p_j, a pair of points in $\mathbb{R}^n$, we define the *bisector* $B_{i,j}$ as the hyperplane that bisects and is orthogonal to line segment $\overline{p_i p_j}$, that is,

$$B_{i,j} := \left\{ x \in \mathbb{R}^n : (p_i - p_j)^T x = \frac{1}{2}(p_i - p_j)^T (p_i + p_j) \right\}.$$

Thus, conditions (2.2) are equivalent to $x \in B_{i,1}$, for $i \geq 2$.

Given a set of points $\mathcal{S}$, there may be many balls that have $\mathcal{S}$ on the boundary and satisfy (2.2). However, if $\mathcal{S}$ is affinely independent then there is a unique ball with center at $\text{aff}(\mathcal{S})$ that passes through the points of $\mathcal{S}$ (Figure 2.2). This claim is easy to prove from a linear algebra argument.

For a comprehensive study on the geometric properties of minimum enclosing balls, we refer the reader to [40].

Duality

The dual problem of (2.1) can be formulated as:

$$\begin{aligned} \max_{\lambda \in \mathbb{R}^m} \quad & \sum_{i=1}^{m} \lambda_i \|p_i\|^2 - \left\| \sum_{i=1}^{m} \lambda_i p_i \right\|^2 \\ \text{s.t.} \quad & \sum_{i=1}^{m} \lambda_i = 1 \\ & \lambda_i \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

The optimal solution λ^* to the dual problem gives the coefficients of the convex combination of the center of the optimal ball in terms of the points on its boundary. Given this interpretation, it becomes clear that the dual problem may not have a unique solution.As a consequence of the convexity of (2.1) and the satisfaction of Slater's condition, *strong duality* holds.

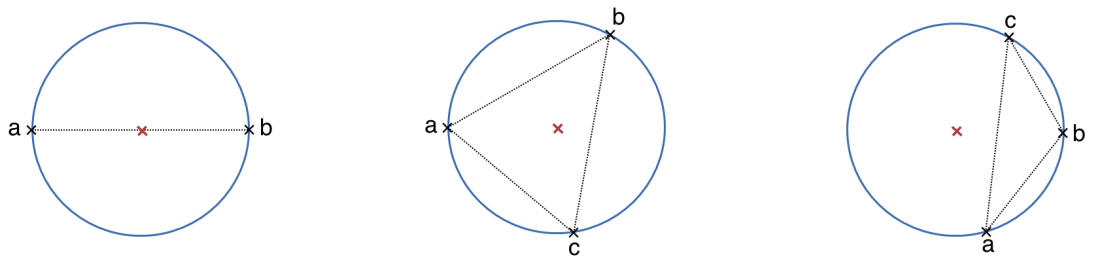
The definition of basis

As mentioned in Section 1.1.1 the minimum enclosing ball problem is an LP-type problem. If we adopt the LP-type definition of basis, we obtain Definition 2.3.

Definition 2.3 (Basis for the MB problem)**.** A subset $S \subseteq \mathcal{P}$ is called a *basis* if no proper subset S' of S is such that $MB(S') = MB(S)$. A subset $S \subseteq \mathcal{P}$ is said to be an *optimal basis* if S is a basis and $MB(S) = MB(\mathcal{P})$.

From Corollary 2.2, we conclude that the points of an optimal basis must be on the boundary of the optimal ball. Moreover, the inclusion-minimal property implies that a basis must be an affinely independent set. That is easy to see: if a ball has an affinely dependent set S on its boundary then there is at least one point $p \in S$ such that $MB(S) = MB(S \setminus \{p\})$. A basis has therefore at most $n + 1$ points. However, not all affinely independent sets of points on the boundary of a ball are bases, as Figure 2.3 illustrates.

The following result follows trivially.



(a) $\{a, b\}$ *is a basis* (b) $\{a, b, c\}$ *is a basis* (c) $\{a, b, c\}$ *is not a basis*

Figure 2.3: Illustration of a basis in $\mathbb{R}^2$.

Theorem 2.4. *Let* $S \subseteq \mathcal{P}$ *be a set of affinely independent points on the boundary of the minimum enclosing ball of* $\mathcal{P}$, $B(x^*, r^*)$. *Then* S *is a basis if and only if* $x^* \in \operatorname{ri}\operatorname{conv}(S)$ *and* $\|x^* - p_i\| = r^*$, *for all* $p_i \in S$.

2.2 The dual simplex-like algorithm by Dearing and Zeck

Dearing and Zeck proposed in 2009 a dual simplex-like algorithm to solve the MB problem in [33]. We will refer to this algorithm by "DZ algorithm". Their algorithm is a generalization of the one by Elzinga and Hearn [37] to dimensions larger than 2.

The DZ algorithm works as follows. At the beginning of each main iteration $j = 0, 1, 2, ...$, we have x^j and r^j the solution of $MB(S^j)$, where $S^j \subseteq \mathcal{P}$ is a *basis*. Hence, the points in S^j are affinely independent. If the current solution corresponds to a ball that encloses all points in $\mathcal{P}$, then it solves $MB(\mathcal{P})$. Otherwise, a point p that is not yet enclosed is chosen to enter S^j. In the next step, S^j is updated: either p is added to S^j or an existing point in S^j is replaced by p. In either case, the algorithm guarantees that points in S^j remain affinely independent. The next and main step of the algorithm consists of finding the next iterate. That will be accomplished in several sub-iterations, where an exact directional search is performed in each one, and a point is removed from S^j, until the point p becomes enclosed and the remaining S^j constitutes a basis.

The algorithm is finite since the radius strictly increases at each main iteration, and the algorithm stops when all points have been enclosed. Though the concept of basis is

never mentioned in the original paper, as we will observe, at the end of each main iteration, we always have that $\mathcal{S}^j \subseteq \mathcal{P}$ is a basis. At the end, when all points of $\mathcal{P}$ are enclosed, the algorithm returns an optimal basis of $MB(\mathcal{P})$.

Before we present the algorithm in detail, let us introduce some definitions and notations. Let $\mathcal{S} = \{p_1, ..., p_s\} \in \mathbb{R}^n$ be an affinely independent set ($s \leq n+1$). We define:

- $\text{Sub}(\mathcal{S})$, the $(s-1)$-dimensional linear space parallel to $\text{aff}(\mathcal{S})$, that is, the column space of matrix $S = [p_2 - p_1 \dots p_s - p_1]$ (note that S is full column rank, as a consequence of the affine independence of $\mathcal{S}$);
- $\text{Null}(\mathcal{S})$, the null space of matrix S, that is, the set of orthogonal vectors to $\text{Sub}(\mathcal{S})$;
- $F_j := \text{conv}(\mathcal{S} \setminus \{p_j\})$, the facet of $\text{conv}(\mathcal{S})$ opposed to point $p_j \in \mathcal{S}$, $j = 1, ..., s$ (that is, the facet that does not contain p_j).

The DZ algorithm maintains the following invariants throughout each iteration:

- $\mathcal{S}^j$ is affinely independent,
- $x^j \in \text{conv}(\mathcal{S}^j)$.

Moreover, with the exception of the point p that is selected to be enclosed, the primal constraints corresponding to $\mathcal{S}^j$ are kept active. So, the algorithm maintains dual feasibility and the complementary slackness conditions. The goal is to make all primal constraints feasible, that is, enclose all points in $\mathcal{P}$.

Initialization:

The routine starts with a basis $\mathcal{S}^0 \subseteq \mathcal{P}$, and x^0 and r^0, the solution to $MB(\mathcal{S}^0)$. If such data is not available, the algorithm picks any two points $p_{0_1}, p_{0_2} \in \mathcal{P}$, and considers

$$\mathcal{S}^0 = \{p_{0_1}, p_{0_2}\}, \quad x^0 = \tfrac{1}{2}(p_{0_1} + p_{0_2}), \quad \text{and} \quad r^0 = \|x^0 - p_{0_1}\|.$$

Main iteration j:

At the beginning of each main iteration, we have the minimum enclosing ball of a basis $\mathcal{S}^j = \{p_{j_1}, ..., p_{j_s}\} \subseteq \mathcal{P}$. Thus, $x^j \in \text{ri}\,\text{conv}(\mathcal{S}^j)$.

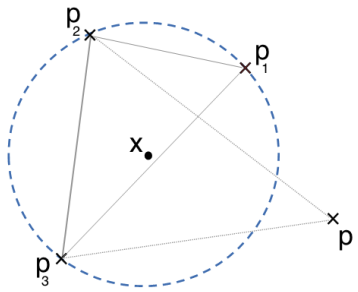


Figure 2.4: Illustration of the *Update* $\mathcal{S}^j$ step: Initially $\mathcal{S}^j = \{p_1, p_2, p_3\}$ and p is the infeasible point chosen. After this step, p_1 is dropped and $\mathcal{S}^j = \{p_2, p_3, p\}$.

Step 1. ***Optimality check*:** If all points of $\mathcal{P}$ are enclosed by $B(x^j, r^j)$ then the current solution is optimal to $MB(\mathcal{P})$ and $\mathcal{S}^j$ is an optimal basis of $MB(\mathcal{P})$. Otherwise, the algorithm picks a point $p \in \mathcal{P}$ that is not yet enclosed.

Step 2. ***Update* $\mathcal{S}^j$:** If $\mathcal{S}^j \cup \{p\}$ is affinely independent, then $\mathcal{S}^j = \mathcal{S}^j \cup \{p\}$, and the invariants are maintained for the current $\mathcal{S}^j$ and x^j. Otherwise, if $\mathcal{S}^j \cup \{p\}$ is affinely dependent, then $\mathcal{S}^j = \mathcal{S}^j \setminus \{p_{j_k}\} \cup \{p\}$ for a point $p_{j_k} \in \mathcal{S}^j$ such that $\mathcal{S}^j \setminus \{p_{j_k}\} \cup \{p\}$ is affinely independent and $x^j \in \text{conv}(\mathcal{S}^j \setminus \{p_{j_k}\} \cup \{p\})$ (see Figure 2.4). Such a point p_{j_k} can be found by considering

$$\frac{\tau_k}{-\sigma_k} = \min_{i=1,\ldots,s} \left\{ \frac{\tau_i}{-\sigma_i} : \sigma_i < 0 \right\},$$

for $\tau_1, \ldots, \tau_s$, and $\sigma_1, \ldots, \sigma_s$, the respective solutions of

$$\sum_{i=1}^{s} \tau_i p_{j_i} = x^j, \ \sum_{i=1}^{s} \tau_i = 1, \quad \text{and} \quad \sum_{i=1}^{s} \sigma_i p_{j_i} = -p, \ \sum_{i=1}^{s} \sigma_i = -1.$$

Step 3. ***Finding the new iterate and basis (sub-iteration)*:** This step consists itself of an iterative process. To find the next iterate, the algorithm uses a directional search procedure that starts at the current center x^j, and proceeds along a direction d. Thus, the search is done on the ray ℓ^+ defined by

$$\ell^+ := \{x^j + \alpha d : \alpha \geq 0\}.$$

As the algorithm moves on ℓ^+, it stops when the first of the following events happen:

Case 1. It hits a facet of $\text{conv}(\mathcal{S}^j)$. In this case, a point from $\mathcal{S}^j$ is dropped, and a new directional search is performed next.

Case 2. It finds the optimal solution to $\text{MB}(\mathcal{S}^j)$ (the intersection of the bisectors of the edges of $\text{conv}(\mathcal{S}^j)$ with $\text{aff}(\mathcal{S}^j)$), and it is in $\text{ri}\,\text{conv}(\mathcal{S}^j)$. In this case $\mathcal{S}^j$ is a basis (Theorem 2.4), and the center of $\text{MB}(\mathcal{S}^j)$ is the next iterate.

The details of this directional search are described below.

i. ***The direction*** d**:** The direction d along which the line search is performed satisfies:

(a) d is parallel to the intersection of the bisectors of the edges of the polytope $\text{conv}(\mathcal{S}^j \setminus \{p\})$, or, equivalently, it is orthogonal to $\text{Sub}(\mathcal{S}^j \setminus \{p\})$. So given any point $p_{j_k} \in \mathcal{S}^j \setminus \{p\}$, we have

$$(p_{j_i} - p_{j_k})^T d = 0, \quad \forall\, p_{j_i} \in \mathcal{S}^j \setminus \{p_{j_k}, p\}; \tag{2.3a}$$

(b) d "points towards" p, in the sense that the distance to p from any point on ℓ^+ decreases as α increases, that is, given any point $p_{j_k} \in \mathcal{S}^j \setminus \{p\}$, we have

$$(p - p_{j_k})^T d = 1; \tag{2.3b}$$

(c) $d \in \text{Sub}(\mathcal{S}^j)$, that is, given $\{u_i\}_i$ a basis for $\text{Null}(\mathcal{S}^j)$, we have

$$u_i^T d = 0, \quad i = 1, ..., n - |\mathcal{S}^j| + 1. \tag{2.3c}$$

To find the direction d, the algorithm solves the linear system defined by conditions (2.3) (note that it is always determined).

ii. ***The directional search***: The ray ℓ^+ intersects both the intersection of the bisectors of the edges of $\text{conv}(\mathcal{S}^j)$ and one (or the intersection of several) facet(s) of $\text{conv}(\mathcal{S}^j)$. Let α_b the point on ℓ^+ that corresponds to the former. It can be calculated by:

$$\alpha_b = \frac{(p - p_{j_k})^T \left(\frac{1}{2}(p + p_{j_k}) - x^j\right)}{(p - p_{j_k})^T d}, \tag{2.4}$$

where p_{j_k} is any point of $\mathcal{S}^j \setminus \{p\}$.

Let α_f correspond to the intersection of ℓ^+ with the boundary of $\mathrm{conv}(\mathcal{S}^j)$. To find α_f, first, the algorithm determines the intersection of ℓ, defined as

$$\ell := \{x^j + \alpha d : \alpha \in \mathbb{R}\},$$

with the supporting hyperplanes of all the facets of $\mathrm{conv}(\mathcal{S}^j)$. Note that only the facets that contain p are relevant. Denote by α_k, $k = 0, \ldots, |\mathcal{S}^j| - 1$, the value of α corresponding to the intersection of ℓ with F_k, the facet opposed to $p_{j_k} \in \mathcal{S}^j \setminus \{p\}$. Let $\{w_i\}_i$ be a basis for the null space of $\mathrm{Null}(\mathcal{S}^j \setminus \{p_{j_k}\})$. Then

$$\alpha_k = \frac{(p - x^j)^T w_i}{d^T w_i}, \text{ for any } i \text{ s.t. } d^T w_i \neq 0. \tag{2.5}$$

The value α_f can be found by choosing the smallest non-negative α_i, that is, $\alpha_f = \alpha_l$ for l given by

$$l = \arg \min_{k=0,\ldots,|\mathcal{S}^j|-1} \{\alpha_k : \alpha_i \geq 0\}. \tag{2.6}$$

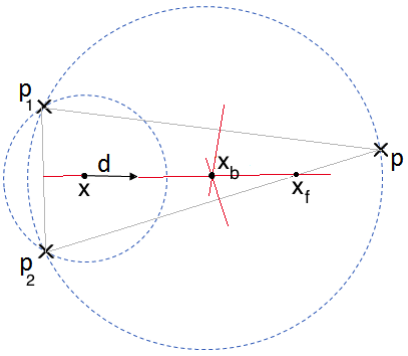
And the facet (or one of the facets) that is intersected by ℓ is the one opposed to point p_{j_l}.

Two cases are now possible (see Figure 2.5):

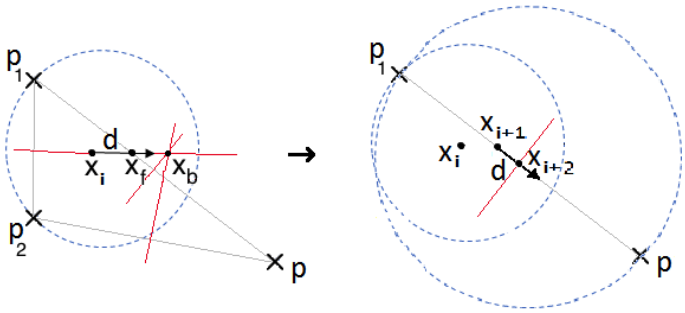
Case 1: $\alpha_b < \alpha_f$, that is, the intersection with the bisectors occurs first. If this is the case, the solution to $\mathrm{MB}(\mathcal{S}^j)$ is the point $x^j + \alpha_b d$, and the algorithm goes back to Step 1 with $x^{j+1} = x^j + \alpha_b d$ and $\mathcal{S}^{j+1} = \mathcal{S}^j$.

Case 2: $\alpha_b \geq \alpha_f$. In this case the opposite point $p_{j_l} \in \mathcal{S}^j$ to the (or one of the) intersected facet(s) is removed from $\mathcal{S}^j$: $\mathcal{S}^j = \mathcal{S}^j \setminus \{p_{j_l}\}$. Note that p_{j_l} can never be p. The algorithm now returns to the beginning of Step 3 with the new $\mathcal{S}^j$ and $x^j = x^j + \alpha_f d$.

Since the algorithm only goes to the beginning of a new iteration when $\alpha_b < \alpha_f$, we have $x^{j+1} \in \mathrm{ri}\,\mathrm{conv}(\mathcal{S}^{j+1})$. Thus, at the end of this procedure, we have that $\mathcal{S}^{j+1}$ is a basis.



(a) Case 1: $\mathcal{S}^j = \{p_1, p_2, p\}$. Since $x_b = x^j + \alpha_b d$ happens before $x_f = x^j + \alpha_f d$, x_b is the center of the ball that encloses $\mathcal{S}^j$. The algorithm goes back to Step 1 with $\mathcal{S}^{j+1} = \{p_1, p_2, p\}$.



(b) Case 2: In the first picture: $\mathcal{S}^j = \{p_1, p_2, p\}$, $x^j = x_i$ is the current center. After the line search, the next iterate is x_f and p_2 is dropped from $\mathcal{S}^j$. The algorithm then goes back to Step 3, illustrated in the second picture, with $\mathcal{S}^j = \{p_1, p\}$ and $x^j = x_{i+1} = x_f$. After the line search, the ball encloses $\mathcal{S}^j$. The algorithm then goes back to Step 1 with $\mathcal{S}^{j+1} = \mathcal{S}^j$ and $x^{j+1} = x_{i+2}$.

Figure 2.5: Illustration of the directional search procedure in $\mathbb{R}^2$.

It is possible to prove that the radius strictly increases at each main iteration [33], thus the algorithm is finite. Note that the algorithm does not suffer from degeneracy. A degenerate situation would occur when we have a step size 0 in the directional search. That is, ℓ intersects one or more facets simultaneously at the current iterate. In that event, the opposite point(s) to those facets are dropped from $\mathcal{S}^j$ at as many sub-iterations as necessary without x^j ever moving. Eventually, the center will finally be allowed to move, and a strictly positive step size will happen, and the ball will inflate. Thus, *cycling* cannot occur.

2.3 The primal simplex-like algorithm by Fischer et al.

Before we proceed to a modification of Dearing and Zeck's algorithm, for the sake of completeness, we briefly present a primal version of their algorithm. A primal simplex-like algorithm for the minimum enclosing ball problem was proposed in 2003 by Fischer, Gärtner and Kutz [42]. We will refer to this algorithm by the FGK algorithm. Their algorithm is based on an idea first proposed by Hopp & Reeve in [58], and it basically starts with a ball that encloses all the points and then deflates it, until it cannot shrink anymore without losing a point.

Consider the unique ball that passes through the points of the affinely independent set S, and that has center in $\text{aff}(S)$. Let $\text{cc}(S)$ denote the center of such ball.

Now, we introduce the main ideas behind the algorithm. Let x^j, and r^j be the center and radius of the ball, respectively, at iteration $j = 0, 1, 2, ...$, and let $S^j \subseteq \mathcal{P}$ the set of points that lie on the boundary of the ball $B(x^j, r^j)$. At each iteration, the algorithm maintains the following invariants:

- S^j is affinely independent;
- all points of $\mathcal{P}$ are enclosed by the current ball.

Primal feasibility and complementary slackness are kept throughout the algorithm. The goal is to achieve $x^j \in \text{conv}(S^j)$, that is, dual feasibility.

Initialization:

To get S^0 and x^0, the algorithm picks any point $p \in \mathcal{P}$, and makes $x^0 = p$ and $S^0 = \{q\}$, where $q \in \mathcal{P}$ is a point at maximal distance from x^0.

Iteration:

At the beginning of the j-th iteration, we have $S^j := \{p_{j_1}, ..., p_{j_s}\}$ affinely independent and x^j, the center of a ball that goes through S^j and encloses all points of $\mathcal{P}$.

Step 1. ***Optimality check*:** The first step of the iteration is to check dual feasibility, that is, whether $x^j \in \text{conv}(S^j)$. If so, x^j is the center of the minimum enclosing ball of $\mathcal{P}$.

Otherwise the algorithm goes to the *Walking phase* possibly preceded by a *Dropping phase* in case $x^j \in \text{aff}(S^j)$.

Step 2. *Dropping phase*: If $x^j \in \text{aff}(S^j)$, then the algorithm enters this phase, where a point is dropped from S^j. If $x^j \in \text{aff}(S^j)$, then $x^j = \text{cc}(S^j)$. Since $x^j \notin \text{conv}(S^j)$ there must be at least one point $p_{j_k} \in S^j$ whose coefficient in the affine combination in terms of S^j is negative. Such a point is removed from S^j: $S^j = S^j \setminus \{p_{j_k}\}$, and the algorithm enters the *Walking phase* next.

Step 3. *Walking phase*: At this stage, $x^j \notin \text{aff}(S^j)$, and the algorithm moves x^j orthogonally to $\text{aff}(S^j)$ in direction of $\text{cc}(S^j)$ in such a way that all points of S^j remain on the boundary of the ball. As the center "walks", the ball decreases its radius, and the movement stops when the first of the following happens:

Case 1. A point $p \in \mathcal{P}$ hits the boundary. In this case, the algorithm goes back to the beginning of a new iteration with $S^{j+1} = S^j \cup \{p\}$ and x^{j+1} the point where the center stopped. For this case, the algorithm has to check for new points to hit the shrinking boundary. It is possible to prove that the points "behind" $\text{aff}(S^j)$, that is, points $q \in \mathcal{P}$ that satisfy

$$(q - x)^T(\text{cc}(S^j) - x^j) \geq \left\|\text{cc}(S^j) - x^j\right\|^2,$$

cannot hit the boundary of the ball with center $x' \in [x^j, \text{cc}(S^j)]$. Thus, such points do not need to be checked. Moreover, if a point was dropped in the *Dropping phase* that preceded the *Walking phase*, then it can also be ignored.

Case 2. The center reaches $\text{aff}(S^j)$: If no point stops the "walk", the center $\text{cc}(S^j)$ of the unique ball that goes through S^j and has center in $\text{aff}(S^j)$ is reached. A new iteration then starts with the same set $S^{j+1} = S^j$ and $x^{j+1} = \text{cc}(S^j)$.

The algorithm is not guaranteed to terminate for an arbitrary set $\mathcal{P}$. The algorithm may encounter a situation where it is not possible to move the center because a point is on the boundary of the current ball. In principle, such situation may happen in several consecutive iterations, where points are added and dropped from S^j but x^j does not change, and eventually, the algorithm finds itself with a S^j that had been considered in the past. That

is, *cycling* occurs. To prevent the *cycling* phenomenon, the authors of the FGK algorithm adopt a rule similar to Bland's rule for the simplex method in linear programming.

Finally, we mention that, in order to obtain an optimal basis at the end of the FGK algorithm, one has to remove from the final $\mathcal{S}^j$ the points corresponding to a zero coefficient in the convex combination of the optimal center in terms of the points in $\mathcal{S}^j$.

2.4 An improvement to the dual simplex-like algorithm

In this section, we propose a modification of the algorithm by Dearing and Zeck with the goal of improving its running time.

As described in Section 2.2, during the *directional phase*, the dual algorithm by Dearing and Zeck finds α_f, corresponding to the point where the ray ℓ^+ intersects the boundary of $\text{conv}(\mathcal{S}^j)$, by calculating the intersection of ℓ with each one of the facets of $\text{conv}(\mathcal{S}^j)$ except the one opposed to p (equation (2.6)). These facets are as many as the number of points in $\mathcal{S}^j \setminus \{p\}$, which can be as many as n. This fact is the central reason why, even using $\mathcal{O}(n^2)$ updates to some factorization of some suitable matrix, the directional search step (and thus each sub-iteration) of the original algorithm could not have a better complexity than $\mathcal{O}(n^3)$, since such updates would need to be done $\mathcal{O}(n)$ times. In this section, we show how one can find α_f without having to check each facet, which will ultimately result in a $\mathcal{O}(n^2)$ sub-iteration, as shown in Section 2.5.

The idea consists of projecting the polytope $\text{conv}(\mathcal{S}^j)$ and the line ℓ orthogonally onto $\text{aff}(\mathcal{S}^j \setminus \{p\})$. Recall that ℓ is perpendicular to $\text{aff}(\mathcal{S}^j \setminus \{p\})$, so its projection will be a single point that coincides with the projection of x^j. In order to find the intersected facet, first, we find in which at most two projected facets of $\text{conv}(\mathcal{S}^j)$ the projection of x^j falls into. We then calculate the intersection of $t\ell^+$ with those two facets to find which facet of $\text{conv}(\mathcal{S}^j)$ is intersected first by the ray ℓ^+.

For simplicity, and since the operations described in this section all happen within the same iteration, we drop the iteration index from $\mathcal{S}^j$ and x^j, and simply consider $\mathcal{S}$ and x.

Before we proceed into the details, consider the following notation, some of which has been introduced before:

- $\mathcal{S} = \{p_1, \ldots, p_s, p\}$, current basis, and $\mathcal{S}' = \mathcal{S} \setminus \{p\}$, so $|\mathcal{S}'| = s$;
- $\mathcal{C} = \text{conv}(\mathcal{S})$ and $\partial\mathcal{C}$ its boundary;
- $F_j := \text{conv}(\mathcal{S} \setminus \{p_j\})$, the facet of $\mathcal{C}$ opposed to point $p_j \in \mathcal{S}$, $j = 1, ..., s$;
- $F_0 := \text{conv}(\mathcal{S} \setminus \{p\})$, the facet of $\mathcal{C}$ opposed to p;
- α_j, $j = 0, \ldots, s$, value of α corresponding to the intersection of ℓ with F_j.

Recall that, at the beginning of Step 3ii, the directional search, we have:

- $\mathcal{S}$, an affinely independent set, consequently $p \notin \text{aff}(\mathcal{S}')$;
- d is a direction in $\text{Sub}(\mathcal{S})$, orthogonal to $\text{aff}(\mathcal{S}')$, that points towards p, and passes through the intersection of the bisectors of the facets of $\text{conv}(\mathcal{S})$;
- x, the current solution, is such that $x \in \text{conv}(\mathcal{S})$.

Let x' and p' be the orthogonal projections of x and p onto $\text{aff}(\mathcal{S}')$, respectively. Let

$$x' = \sum_{j=1}^{s} \pi_j p_j \text{ s.t. } \sum_{j=1}^{s} \pi_j = 1, \quad \text{and} \quad p' = \sum_{j=1}^{s} \omega_j p_j \text{ s.t. } \sum_{j=1}^{s} \omega_j = 1, \tag{2.7}$$

be their unique representations as affine combinations in terms of the points in $\mathcal{S}'$. An important observation is that the projection of ℓ onto $\text{aff}(\mathcal{S}')$ coincides with x'.

We now introduce two useful lemmas.

Lemma 2.5. $x' \in \text{conv}(\mathcal{S}' \cup \{p'\})$.

Proof. This is a consequence of $x \in \text{conv}(\mathcal{S})$ and the linearity of the projection operator. □

Lemma 2.6. *Consider the representations (2.7). We have that* $\pi_j < 0$ *implies* $\omega_j < 0$.

Proof. From Lemma 2.5 we know that there exists $\beta \geq 0$ and $\beta' \geq 0$ such that

$$x' = \sum_{j=1}^{s} \beta_j p_j + \beta' p', \quad \text{with} \quad \sum_{j=1}^{s} \beta_j + \beta' = 1.$$

Therefore,

$$x' = \sum_{j=1}^{s} \beta_j p_j + \beta' \sum_{j=1}^{s} \omega_j p_j = \sum_{j=1}^{s} (\beta_j + \beta' \omega_j) p_j.$$

Let $\delta_j = \beta_j + \beta' \omega_j$. Note that $\sum_{j=1}^{s} \delta_j = 1$. Since $\mathcal{S}$ is an affinely independent set, the representation of x' as an affine combination of the points in $\mathcal{S}$ is unique, therefore we must have $\pi_j = \delta_j = \beta_j + \beta' \omega_j$, for all $j = 1, ..., s$. Thus, if $\pi_j < 0$, then $\omega_j < 0$. Note that $\pi_j = 0$ does not imply $\omega_j \leq 0$. □

Consider the general case where the intersection of the line ℓ with $\partial\mathcal{C}$ is two points z_1 and z_2. Let F_{k_1} be one of the facets containing z_1 and F_{k_2} be one containing z_2. Point z_1 or z_2 may be on the intersection of several facets, but knowing one of them suffices. The projection of both z_1 and z_2 onto $\text{aff}(\mathcal{S}')$ is x', therefore x' will be written as a unique convex combination of the projections of the points that form F_{k_1} and also as a unique convex combination of the projections of the points that form F_{k_2}, and only of those and no other projected facets. In the particular cases when ℓ intersects $\partial\mathcal{C}$ on a single point or an infinite number of points, x' will still be written as a convex combination of the projected points of one of the intersected facets. Lemma 2.7 states this fact. Note that the intersection always exists since $x \in \ell \cap \mathcal{C}$.

Lemma 2.7. *Line ℓ intersects facet F_k if and only if $x' \in \text{conv}(\mathcal{S}' \setminus \{p_k\} \cup \{p'\})$.*

Proof. We only need to prove that, if $x' \in \text{conv}(\mathcal{S}' \setminus \{p_k\} \cup \{p'\})$, then ℓ intersects F_k, since the opposite is trivial as a consequence of the linearity of the projection operator. Let $A = [p_2 - p_1, \ldots, p_s - p_1]$. Then $p' = A(A^T A)^{-1} A^T (p - p_1) + p_1$. First, we prove that there exists a $\gamma > 0$ such that $d = \gamma(p - p')$. Clearly, $p - p' \in \text{Sub}(\mathcal{S})$, and $p - p'$ is orthogonal to $\text{Sub}(\mathcal{S}')$ since

$$A^T(p - p') = A^T(p - p_1) - A^T(p - p_1) = 0$$

(recall that $(p - p_s)^T d > 0$). Moreover, d and $p - p'$ have the same direction since

$$(p - p_s)^T (p - p') = \|p - p'\|^2 + (p' - p_s)^T (p - p') = \|p - p'\|^2 > 0.$$

Thus we conclude that there exists a $\gamma > 0$ such that $d = \gamma(p - p')$.

Now, suppose $x' \in \text{conv}(S' \setminus \{p_k\} \cup \{p'\})$, that is,

$$x' = \sum_{\substack{j=1 \\ j \neq k}}^{s} \beta_j p_j + \beta' p' \text{ with } \sum_{\substack{j=1 \\ j \neq k}}^{s} \beta_j + \beta' = 1 \text{ and } \beta' \geq 0,\ \beta_j \geq 0,\ j = 1, ..., s.$$

Observe that for any α we have

$$x' + \alpha d = \sum_{\substack{j=1 \\ j \neq k}}^{s} \beta_j p_j + \beta' p' + \alpha\gamma(p - p') = \sum_{\substack{j=1 \\ j \neq k}}^{s} \beta_j p_j + (\beta' - \alpha\gamma)p' + \alpha\gamma p.$$

Let $\alpha' = \frac{\beta'}{\gamma}$. We have that $x' + \alpha' d \in \text{conv}(S' \setminus \{p_k\} \cup \{p\})$ since $\sum_{j \neq k} \beta_j + \alpha'\gamma = 1$. This implies that there exists an α such that $x + \alpha d \in F_k$, that is, ℓ intersects F_k. □

Finally, Theorem 2.8 shows how to calculate the intersection of ℓ^+ with $\partial\mathcal{C}$.

Theorem 2.8. *Consider the representations (2.7) of x' and p'. Suppose F_l is the facet that is first intersected by ℓ^+, and let α_f be the value of α at which the intersection occurs. To find the point opposed to F_l and α_f, there are two possible cases:*

- *Case 1. If there is a $k = 1, ..., s$ such that $\pi_k = \omega_k = 0$, then p_k is the point opposed to the facet intersected first, and $\alpha_f = 0$.*
- *Case 2. Suppose case 1 does not hold. First, find k_1 such that*

$$\frac{\pi_{k_1}}{\omega_{k_1}} = \min_{j=1,...,s} \left\{ \frac{\pi_j}{\omega_j} : \pi_j \geq 0, \omega_j > 0 \right\}. \tag{2.8a}$$

Then, find α_{k_1} as in (2.5). Let $\mathcal{J} := \{j : \pi_j \leq 0, \omega_j < 0\}$. If $\mathcal{J} \neq \emptyset$, find k_2 such that

$$\frac{\pi_{k_2}}{\omega_{k_2}} = \max_{j=1,...,s} \left\{ \frac{\pi_j}{\omega_j} : \pi_j \leq 0, \omega_j < 0 \right\}, \tag{2.8b}$$

and find α_{k_2} as in (2.5). Otherwise, if $\mathcal{J} = \emptyset$ simply consider $\alpha_{k_2} = -\infty$. The facet first intersected by ℓ^+, F_l, is such that $l = \arg\min_{j=k_1,k_2}\{\alpha_j : \alpha_j \geq 0\}$, and $\alpha_f = \alpha_l$.

Proof. Case 1. Suppose there is a $k \in \{1, \ldots, s\}$ such that $\omega_k = \pi_k = 0$. Then, since

$x = x' + \delta d$, for some $\delta \geq 0$, and that there is a $\gamma > 0$ such that $d = \gamma(p - p')$ (see the proof of Lemma 2.7) we have:

$$x = x' + \delta\gamma(p - p') = \sum_{\substack{j=1 \\ j\neq k}}^{s} \pi_j p_j + \delta\gamma \left(p - \sum_{\substack{j=1 \\ j\neq k}}^{s} \omega_j p_j \right) = \sum_{\substack{j=1 \\ j\neq k}}^{s} (\pi_j - \delta\gamma\omega_j)\, p_j + \delta\gamma p.$$

This representation of x as a convex combination of S is unique, since S is affinely independent. Consequently, $\pi_j - \delta\gamma\omega_j \geq 0$ and $\delta\gamma \geq 0$, concluding that $x \in \text{conv}(S'\backslash\{p_k\}\cup\{p\}) \equiv F_k$. Therefore ℓ^+ intersects F_k at $\alpha_f = \alpha_k = 0$.

Case 2. Since $x' \in \text{conv}(S' \cup \{p'\})$, from the proof of Lemma 2.6, there exists $\beta' \geq 0$ such that

$$x' = \sum_{j=1}^{s} \beta_j p_j + \beta' p' = \sum_{j=1}^{s} (\pi_j - \beta'\omega_j) p_j + \beta' p'. \tag{2.9}$$

Formula (2.9) gives all possible ways to represent x' as a convex combination of $S' \cup \{p'\}$ as a function of β'. We are now interested in knowing the minimum and maximum values of β', β'_{min} and β'_{max}, respectively.

Suppose $\pi_j \geq 0$, for all $j = 1, ..., s$, that is, $x' \in \text{conv}(S' \cup \{p'\})$. Then $\beta_{min} = 0$. It is easy to see that, in such case, $\beta_{max} = \frac{\pi_{k_1}}{\omega_{k_1}}$ as in (2.8a), and that any $\beta \in [0, \beta_{max}]$ yields $\pi_j - \beta\omega_j \geq 0$. We conclude that $x' \in \text{conv}(S' \setminus \{p_{i_{k_1}}\} \cup \{p'\})$, and so, from Lemma 2.7, ℓ intersects F_{k_1}. Note that β_{max} may be 0. When $\beta_{max} > 0$, observe that there is no other $\beta \in]0, \beta_{max}[$ such that $\pi_j - \beta\omega_j = 0$, so there is no other way to write x' as a convex combination of p' and $s - 1$ points of S'.

Now suppose that there exists $\pi_j < 0$. Then $x' \notin \text{conv}(S')$ and therefore $\beta_{min} > 0$. Since β' must be positive, in order to have a convex combination in (2.9), β' must satisfy simultaneously the following conditions:

$$\beta' \geq \frac{\pi_j}{\omega_j}, \quad \forall j : \omega_j < 0, \pi_j \leq 0 \quad \Rightarrow \quad \beta' \geq \frac{\pi_{k_2}}{\omega_{k_2}},$$

$$\beta' \leq \frac{\pi_j}{\omega_j}, \quad \forall j : \omega_j > 0, \pi_j \geq 0 \quad \Rightarrow \quad \beta' \leq \frac{\pi_{k_1}}{\omega_{k_1}},$$

Algorithm 2.1 Find α_f, the intersection of ℓ^+ with $\partial\mathcal{C}$.

input: $\mathcal{S}'$, p, x.
output: α_f, and p_l, the point opposed to the intersected facet.

Calculate x' and p', the orthogonal projections of x and p, respectively, onto $\text{aff}(\mathcal{S}')$;
Find π and ω as in (2.7);
if $\exists k = 1, ..., s$ s.t. $\pi_k = \omega_k = 0$ **then**
    $\alpha_f = 0$ and $p_l = p_k$.
else
    Find k_1 as in (2.8a), and calculate α_{k_1};
    if $\{j : \pi_j \leq 0, \omega_j < 0\} \neq \emptyset$ **then**
        Find k_2 as in (2.8b), and calculate α_{k_2};
    else
        $\alpha_{k_2} = -\infty$;
    end if
    $l = \text{argmin}_{j=k_1,k_2}\{\alpha_j : \alpha_j \geq 0\}$;
    $\alpha_f = \alpha_l$ and $p_f = p_l$.
end if

for k_1 and k_2 as in (2.8a) and (2.8b), respectively. The conditions above are feasible since there must be such a β', and because of Lemma 2.6. Thus, $\frac{\pi_{k_2}}{\omega_{k_2}} \leq \frac{\pi_{k_1}}{\omega_{k_1}}$, so $\beta_{min} = \frac{\pi_{k_2}}{\omega_{k_2}}$ and $\beta_{max} = \frac{\pi_{k_1}}{\omega_{k_1}}$. Finally, we conclude that $x' \in \text{conv}(\mathcal{S}' \setminus \{p_{k_1}\} \cup \{p'\})$ and $x' \in \text{conv}(\mathcal{S}' \setminus \{p_{k_2}\} \cup \{p'\})$, with $k_1 \neq k_2$, meaning ℓ intersects both facets F_{k_1} and F_{k_2}. □

The procedure to find α_f is summarized in Algorithm 2.4.

2.5 Implementation details

Recall the QR factorization of a matrix $A \in \mathbb{R}^{n\times m}$ into the product of matrices Q and R, where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $R \in \mathbb{R}^{n\times m}$ is upper triangular. For our purposes, we will consider the case when $n \geq m$. If A is full column rank, then the diagonal of R is non-zero, and Q can be partitioned in $[V\ U]$, for $V \in \mathbb{R}^{n\times m}$ and $U \in \mathbb{R}^{n\times(n-m)}$, such that the columns of V form a basis of $\text{Span}(A)$, the column range of A, and the columns of U form a basis of $\text{Null}(A)$, the null space of A. Different algorithms are available to find a QR factorization of a matrix, but in terms of computational work, and for a general $(n \times n)$-matrix, they all need $\mathcal{O}(n^3)$ steps [52, §5.2]. However, the QR factorization of A can be "recycled" and used to calculate the QR factorization of a matrix obtained from A

by either rank-one changes, appending a row or column to A, or deleting a row or column from A [52, §12.5]. These factorization updates are accomplished by using *Givens rotations* (e.g. [52, §5.1.8]), and, in the case when $m = n$, need $\mathcal{O}(n^2)$ steps.

We now describe in detail how one can implement the algorithm taking advantage of the efficient updates of a QR factorization. At the beginning of each iteration, we have the QR factorization of the $(n \times s)$-matrix S, whose columns are the points in the current basis $\mathcal{S}^j = \{p_{j_1}, ..., p_{j_s}\}$:

$$S = \begin{bmatrix} p_{j_1} & p_{j_2} & \cdots & p_{j_s} \end{bmatrix}.$$

Let Q_S and R_S be the matrices of the QR factorization of S

$$S = Q_S R_S.$$

The first iteration is the only time a QR factorization of a matrix S is calculated from scratch. After that, the QR factorization of S is updated every time a point is added or removed from $\mathcal{S}^j$, that is, a column is added or removed from S, respectively.

In the next sections we will be presenting some pseudo-code for different parts of the algorithm. In this pseudo-code we will refer to procedures `QRinsert`, `QRdelete`, and `QRupdate` defined below:

- `QRinsert`: Receives as input the QR factors of some matrix A and either a column vector or a row vector, and outputs the QR factorization of the matrix resulting from appending the column vector or the row vector to A;
- `QRdelete`: Receives as input the QR factors of some matrix A and the index of a column or a row, and outputs the QR factorization of the matrix resulting from deleting that column or row from A;
- `QRupdate`: Receives as input the QR factors of some matrix A and two vectors u, v, and outputs the QR factorization of matrix $A + uv^T$.

2.5.1 The "Update $\mathcal{S}^j$" procedure

The first step of this procedure is to check whether $\mathcal{S}^j \cup \{p\}$ is affinely independent. If it is not, the next step is to find $p_{j_k} \in \mathcal{S}^j$, such that $\mathcal{S}^j \setminus \{p_{j_k}\} \cup \{p\}$ is affinely independent and $x^j \in \text{conv}(\mathcal{S}^j \setminus \{p_{j_k}\} \cup \{p\})$. In order to do this, consider matrices B and $\bar{B}$ as follows:

$$B = \begin{bmatrix} p_{j_1} & \dots & p_{j_s} \\ 1 & \dots & 1 \end{bmatrix}, \qquad \bar{B} = \begin{bmatrix} p_{j_1} & \dots & p_{j_s} & p \\ 1 & \dots & 1 & 1 \end{bmatrix}. \tag{2.10}$$

Matrix B is full column rank since $\mathcal{S}^j$ is affinely independent. When $\mathcal{S}^j$ has $n+1$ points we automatically know that $\mathcal{S}^j \cup \{p\}$ is affinely dependent. Consider then $s \leq n$, which implies that $\bar{B}$ has at least as many columns as rows. $\mathcal{S}^j \cup \{p\}$ is affinely independent if and only if $\bar{B}$ is full column rank, which can be found out by looking at the $(s+1, s+1)$-entry of $R_{\bar{B}}$, where $\bar{B} = Q_{\bar{B}} R_{\bar{B}}$. If that entry is zero or close to zero (to account for precision errors) then $\mathcal{S}^j \cup \{p\}$ is affinely dependent, otherwise it is affinely independent.

Matrix B can be obtained by inserting a row of ones in S, and $\bar{B}$ can then be obtained by inserting the column $\binom{p}{1}$ in B. Thus, the respective QR factorizations can be efficiently computed from the QR factorization of S and of B, respectively.

If $\mathcal{S}^j \cup \{p\}$ is affinely independent, we add p to $\mathcal{S}^j$: $\mathcal{S}^j = \mathcal{S}^j \cup \{p\}$ (but we leave matrix S as is). Otherwise, the following linear systems need to be solved

$$B\tau = \binom{x^j}{1}, \quad \text{and} \quad B\sigma = \binom{-p}{-1}. \tag{2.11}$$

These linear systems can be reduced to linear systems with upper triangular matrices, using the QR factorization of B

$$R_B \tau = Q_B^T \binom{x^j}{1}, \qquad R_B \sigma = Q_B^T \binom{-p}{-1},$$

which can be solved using *Back Substitution* (performed in $\mathcal{O}(n^2)$ [52, §3.1]). After finding the point to be removed from $\mathcal{S}^j$, we get the new Q_S and R_S, the QR factorization of the new matrix S, obtained from the previous one by removing the column corresponding to

Algorithm 2.2 `Update_`$\mathcal{S}^j$

input: Q_S, R_S, p, $\mathcal{S}^j$, x^j
output: Q_S, R_S, $\mathcal{S}^j$

1: Q_B, $R_B \leftarrow$ `QRinsert`$(Q_S, R_S,$ row $1_s)$
2: $Q_{\bar{B}}$, $R_{\bar{B}} \leftarrow$ `QRinsert`$(Q_B, R_B,$ column $\binom{p}{1})$
3: **if** $(s+1, s+1)$-entry of $R_{\bar{B}}$ is positive **then**
4: $\quad \mathcal{S}^j = \mathcal{S}^j \cup \{p\}$.
5: **else**
6: $\quad$ Get τ and σ by solving $R_B\tau = Q_B^T\binom{x^j}{1}$ and $R_B\sigma = Q_B^T\binom{-p}{-1}$;
7: $\quad$ Find p_{j_k}, such that $\frac{\pi_k}{-\omega_k} = \min_{j=1,\ldots,s}\left\{\frac{\pi_j}{-\omega_j} : \omega_j < 0\right\}$;
8: $\quad \mathcal{S}^j = \mathcal{S}^j \setminus \{p_{j_k}\} \cup \{p\}$;
9: $\quad Q_S$, $R_S \leftarrow$ `QRdelete`$(Q_S, R_S,$ column $(p_{j_k}))$.
10: **end if**

that point. We do not add p yet to S; we leave that to the very end of the iteration when p is finally covered.

Algorithm 2.2 summarizes these ideas.

2.5.2 Calculating the direction d

Recall that, at this stage, Q_S and R_S are the QR factors of matrix $S = [p_{j_1} \ \ldots \ p_{j_s}]$ that contains the points of $\mathcal{S}^j \setminus \{p\}$. Define matrix C as

$$C = \begin{bmatrix} p_{j_1} - p_{j_s} & \cdots & p_{j_{s-1}} - p_{j_s} & p - p_{j_s} \end{bmatrix}.$$

Let $C = Q_C R_C$ be the QR factorization of C, obtained by updating the QR factorization of S twice, since C can be obtained from S by adding two rank one matrices;

$$C = S + (p - p_{j_s})e_n^T - p_{j_s}1_n^T.$$

Matrix C is full column rank since $\mathcal{S}^j \cup \{p\}$ is affinely independent, so we can partition $Q_C = [V_C \ U_C]$, where V_C is a $(n \times s)$-matrix whose columns are a basis to Span(C), and U_C is a $(n \times (n-s))$-matrix whose columns are a basis to Null(C). Finally, to find d we

Algorithm 2.3 `Calculate_d`

input: Q_S, R_S, p, $\mathcal{S}^j$
output: d, Q_C, R_C

1: Q_C, $R_C \leftarrow$ `QRupdate`$(Q_S, R_S, p - p_{j_s}, e_n)$;
2: Q_C, $R_C \leftarrow$ `QRupdate`$(Q_C, R_C, -p_{j_s}, 1_n)$;
3: Build $\hat{C}$ and solve $\hat{C}g = e_s$;
4: $d = Q_C g$.

need to solve the linear system resulting from conditions (2.3):

$$\begin{bmatrix} C^T \\ U_C^T \end{bmatrix} d = e_s \quad \Longleftrightarrow \quad \hat{C} Q_C^T d = e_s, \quad \text{with} \quad \hat{C} = \left[\begin{array}{c|c} & R_C^T \\ \hline 0_{(n-s)\times s} & I_{n-s} \end{array}\right].$$

Note that $\hat{C}$ is a lower triangular matrix.

Algorithm 2.3 summarizes the procedure to calculate d.

2.5.3 Calculating the next iterate

The first step now is to calculate x' and p', the orthogonal projection of x^j and p respectively onto $\mathrm{aff}(\mathcal{S}')$, where $\mathcal{S}' = \mathcal{S}^j \setminus \{p\}$. Define matrix D,

$$D = \begin{bmatrix} p_{j_1} - p_{j_s} & \cdots & p_{j_{s-1}} - p_{j_s} \end{bmatrix},$$

and let Q_D and R_D be the matrices of its QR factorization. The QR factorization of D can be obtained easily from the QR factorization of matrix C, since D can be obtained from C by removing the last column. Let V_D be the matrix with the first $s-1$ columns of Q_D, which form an orthogonal basis to $\mathrm{Sub}(\mathcal{S}')$. Consider $\mathrm{aff}(\mathcal{S}') = p_{j_1} + \mathrm{Sub}(\mathcal{S}')$. The projections x' and p' can be calculated as

$$x' = V_D V_D^T (x^j - p_{j_1}) + p_{j_1}, \quad \text{and} \quad p' = V_D V_D^T (p - p_{j_1}) + p_{j_1}.$$

Algorithm 2.4 `Calculate_π_ω`

input: Q_C, R_C, p, x^j
output: π, ω

1: Q_D, $R_D \leftarrow$ `QRdelete`$(Q_C, R_C,$ column $p - p_{j_s})$;
2: Get V_D, the matrix with the first $s-1$ columns of Q_D;
3: $x' = V_D V_D^T(x^j - p_{j_1}) + p_{j_1}$; $\quad p' = V_D V_D^T(p - p_{j_1}) + p_{j_1}$;
4: Q_E, $R_E \leftarrow$ `QRinsert`$(Q_S, R_S,$ row $1_s)$;
5: Get π and ω by solving $R_E\pi = Q_E^T\begin{pmatrix} x' \\ 1 \end{pmatrix}$ and $R_E\omega = Q_E^T\begin{pmatrix} p' \\ 1 \end{pmatrix}$.

Now, the linear systems corresponding to (2.7) need to be solved:

$$E\pi = \begin{pmatrix} x' \\ 1 \end{pmatrix} \quad \text{and} \quad E\omega = \begin{pmatrix} p' \\ 1 \end{pmatrix}, \quad \text{with} \quad E = \begin{bmatrix} p_{j_1} & \cdots & p_{j_s} \\ 1 & \cdots & 1 \end{bmatrix}. \tag{2.12}$$

These linear systems are similar to (2.11) so they are solved following similar steps (Algorithm 2.4). Note that E may not be the same as B from the *Update* $\mathcal{S}$ *procedure*.

After using π and ω to find k_1 and possibly k_2, as in (2.8), we need to calculate α_{k_1} and α_{k_2}. To calculate α_{k_1}, we need a basis for $\text{Null}(\mathcal{S}^j \setminus \{p_{j_{k_1}}\})$, in order to apply formula (2.5). If $k_1 < s$ then $\text{Sub}(\mathcal{S}^j \setminus \{p_{j_{k_1}}\}) \equiv \text{Span}(F)$ with

$$F = \begin{bmatrix} p_{j_1} - p_{j_s} & \cdots & p_{j_{k_1-1}} - p_{j_s} & p_{j_{k_1+1}} - p_{j_s} & \cdots & p_{j_{s-1}} - p_{j_s} & p - p_{j_s} \end{bmatrix}.$$

Matrix F can be obtained from C by deleting its k_1-th column. On the other hand, if $k_1 = s$ then $\text{Sub}(\mathcal{S}^j \setminus \{p_{j_s}\}) \equiv \text{Span}(F)$ with

$$F = \begin{bmatrix} p_{j_1} - p & \cdots & p_{j_{s-1}} - p \end{bmatrix},$$

and F can be obtained from C by deleting the last column and then adding the rank one matrix $(p_{j_s} - p)1_n^T$. Therefore, in both cases, the QR factorization of F can be obtained by updating the QR factorization of C. A basis for $\text{Null}(\mathcal{S}^j \setminus \{p_{k_1}\})$ is then formed by the last $n - s + 1$ columns of Q_F. The value α_{k_2} is calculated in an analogous way. Algorithm 2.5 summarizes these ideas.

Algorithm 2.5 `Calculate_`α_{k_i}

input: Q_C, R_C, p, d, $\mathcal{S}^j$, x^j, k_i
output: α_{k_i}

1: Q_F, $R_F \leftarrow$ `QRdelete`$(Q_C, R_C, k_i-$th column$)$;
2: **if** $k_i = s$ **then**
3: $\quad Q_F$, $R_F \leftarrow$ `QRupdate`$(Q_F, R_F, (p_{j_s} - p), 1_n)$;
4: **end if**
5: Get W_F, the matrix with the last $n - s + 1$ columns of Q_F;
6: $\alpha_{k_i} = \frac{(p - x^j)^T w_j}{d^T w_j}$ for any column w_j of W_F such that $d^T w_j \neq 0$.

2.5.4 Pseudo-code of the algorithm

The only time a QR factorization of a matrix is calculated from scratch is at the beginning of the algorithm. From the discussion of the previous sections, we conclude that each sub-iteration requires a constant number of "QR updates" that are performed to matrices with n rows and at most n columns, so such updates take $\mathcal{O}(n^2)$ steps. This results in a sub-iteration that is done in quadratic time. Since the optimality check is $\mathcal{O}(mn)$ and each main iteration may need as many as n sub-iterations, we conclude that each (main) iteration of the algorithm has complexity of at most $\mathcal{O}(mn + n^3)$. Recall that there is no polynomial bound on the number of iterations of the algorithm.

We now aggregate the results/discussion of the previous sections in Algorithm 2.6.

2.6 Computational results

We implemented the original algorithm by Dearing and Zeck, as presented in Section 2.2, using efficient updates of the QR factorization of a certain matrix. We denote this version by "DZ original". We also implemented the algorithm with the modification presented in Section 2.4 and following the implementation directions of Section 2.5. We denote this version by "DZ new". The goal is to study the practical impact of the modification presented in Section 2.4.

Our experiments were conducted using MATLAB R2014a (version 8.3) on a PC with an Intel Core i5 2.30 GHz processor, with 4 GB RAM. We considered datasets generated

Algorithm 2.6 Faster dual algorithm for the minimum enclosing ball of points

input: $\mathcal{P}$

output: x, r, $\mathcal{S}$, the optimal solution and a basis, respectively.

Initialization:

1: Choose any two points $\{p_{0_1}, p_{0_2}\} \in \mathcal{P}$;
2: Set $\mathcal{S}^0 = \{p_{0_1}, p_{0_2}\}$, $x^0 = \frac{p_{0_1}+p_{0_2}}{2}$, $r = \|x^0 - p_{0_1}\|$;
3: Get Q_S, R_S the QR factorization of $[p_{0_1}\ p_{0_2}]$;

Loop:

4: **for** $j = 1, 2, \ldots.$ **do**

Optimality check:

5: **if** $\left\|x^j - p_i\right\| \leq r$ for all $p_i \in \mathcal{P}$ **then**
6: (x^j, r^j) is the optimal solution and $\mathcal{S}^j$ is an optimal basis. **Stop.**
7: **else**
8: Get $p \in \mathcal{P}$ s.t. $\left\|x^j - p\right\| > r$;
9: **end if**
10: $Q_S, R_S, \mathcal{S}^j \leftarrow$ `Update_`$\mathcal{S}$ $(Q_S, R_S, p, \mathcal{S}^j, x^j)$;

Finding the new iterate and basis (sub-iteration):

11: $d, Q_C, R_C \leftarrow$ `Calculate_d` $(Q_S, R_S, p, \mathcal{S}^j)$;
12: Calculate α_b as in (2.4);
13: $\pi, \omega \leftarrow$ `Calculate_`π_ω (Q_C, R_C, p, x^j);
14: **if** $\exists\ k = 1, ..., s$ s.t. $\pi_k = \omega_k = 0$ **then**
15: $\alpha_f = 0$ and $p_f = p_{j_k}$;
16: **else**
17: Find k_1 as in (2.8a);
18: $\alpha_{k_1} \leftarrow$ `Calculate_`α_{k_i} $(Q_C, R_C, p, d, \mathcal{S}^j, x^j, k_1)$;
19: **if** $\{j : \pi_j \leq 0, \omega_j < 0\} \neq \emptyset$ **then**
20: Find k_2 as in (2.8b);
21: $\alpha_{k_2} \leftarrow$ `Calculate_`α_{k_i} $(Q_C, R_C, p, d, \mathcal{S}^j, x^j, k_2)$;
22: **else**
23: $\alpha_{k_2} = -\infty$;
24: **end if**
25: Find k s.t. $k = \arg\min_{j=k_1,k_2}\{\alpha_j :\ \alpha_j \geq 0\}$;
26: $\alpha_f = \alpha_k$ and $p_f = p_k$.
27: **end if**
28: **if** $\alpha_b < \alpha_f$ **then**
29: $x^{j+1} = x^j + \alpha_b d$; $r^{j+1} = \left\|x^{j+1} - p\right\|$;
30: $Q_S, R_S \leftarrow$ `QRinsert`(Q_S, R_S, p);
31: Start a new iteration: Got to *line 5.*
32: **else**
33: $x^j = x^j + \alpha_f d$; $\mathcal{S}^j = \mathcal{S}^j \setminus \{p_f\}$;
34: $Q_S, R_S \leftarrow$ `QRdelete`$(Q_S, R_S, k$—th column$)$;
35: Start a new sub-iteration: Go to *line 11.*
36: **end if**
37: **end for**

randomly from a uniform distribution at the unit cube. Tables 2.1 and 2.2 show the average running times for different dimensions of the two versions of the algorithm on instances with 1000 and $10,000$ points, respectively.

We observed that for smaller dimensions, the original algorithm is slightly faster. But, as the dimension increases, we observe that the new version of the algorithm with the changes we proposed in Section 2.4 is considerably faster, for example, for $n = 5000$ we observe a 10-fold improvement in the running times.

2.7 Conclusion

In this chapter, we studied the dual simplex-like algorithm proposed by Dearing and Zeck for the problem of enclosing a set of points with a ball of smallest radius. We realized that the directional search of the original algorithm was inefficient due to the fact that it requires finding the intersection of a line with $\mathcal{O}(n)$ hyperplanes. This fact, yields slow running times for large dimensions, since each directional search (and consequently, each sub-iteration) could not be done in time better than $\mathcal{O}(n^3)$.

In order to improve the running times of the algorithm, we proposed a modification of the directional search procedure, which together with using efficient updates of the QR factorization of a certain matrix, allows each sub-iteration to be performed in $\mathcal{O}(n^2)$ time. We observe that this modification improved considerably the running times of the algorithm, especially for large values of the dimension n.

Problem		Time in seconds	
n	m	DZ original	DZ new
50	1000	0.15	0.32
100	1000	0.42	0.96
500	1000	32.52	20.13
1000	1000	140.45	57.50
5000	1000	8852.20	887.15

Table 2.1: Running time (in seconds) of the two versions of the algorithm, for 1000-point datasets with points uniformly distributed in a unit cube, for different dimensions n. The reported numbers correspond to the averages corresponding to 25 runs for each n.

Problem		Time in seconds	
n	m	DZ original	DZ new
50	10,000	2.48	2.95
100	10,000	5.03	5.97
500	10,000	70.04	42.55
1000	10,000	267.68	114.88
5000	10,000	17044.10	1463.20

Table 2.2: Running time (in seconds) of the two versions of the algorithm, for $10,000$-point datasets with points uniformly distributed in a unit cube, for different dimensions n. The reported numbers correspond to the averages corresponding to 25 runs for each n.

Chapter 3

The minimum k-enclosing ball of points

Let $\mathcal{P}$ be a set of m points, $p_1, \ldots, p_m$, in $\mathbb{R}^n$. We define a k-*enclosing ball* of $\mathcal{P}$ as an n-dimensional Euclidean hypersphere that covers at least k points of $\mathcal{P}$. The *minimum* k-*enclosing ball of points* problem, or *MB*$_k$ *problem* in short, aims at finding the k-enclosing ball of $\mathcal{P}$ with smallest radius. The MB$_k$ problem can be formulated as:

$$\text{MB}_k(\mathcal{P}): \quad \begin{array}{ll} \min\limits_{x \in \mathbb{R}^n, r \in \mathbb{R}} & r \\ \text{s.t.} & |\{p_j \in \mathcal{P} : \|x - p_j\| \leq r\}| \geq k. \end{array} \tag{3.1}$$

Figure 3.1 illustrates the problem in $\mathbb{R}^2$.

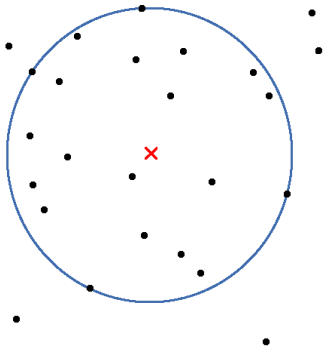


Figure 3.1: Illustration of the minimum 20-enclosing ball for a 25-point dataset in $\mathbb{R}^2$.

In this chapter we study the use of the DZ algorithm from Section 2.2 in a branch-and-bound (B&B) scheme to find an exact solution to the minimum k-enclosing ball problem. The search tree is the tree of all k-subsets[1] of $\mathcal{P}$, where each node corresponds to a subset of at most k points. Our B&B shares similarities with the branch-and-bound algorithm proposed by Candela in [19], and also considered by Ahipaşaoğlu in [6], for the problem of enclosing k points of $\mathcal{P}$ with the smallest volume ellipsoid.

Similarly to Candela, we use a depth-first search strategy for selecting the next node to explore and combine it with a method that aims at placing nodes with large bounds at the root of large subtrees. This combination results in a behavior that shares the advantages of both depth-first-search and best-first-search techniques, and is able to prune very large subtrees early on. Ahipaşaoğlu, instead, uses first-in-first-out and builds the tree following the lexicographic order of points. Contrarily to Candela, every time a node is created, we solve the subproblem associated with that node which consists of enclosing the subset corresponding to that node with the smallest enclosing ball. As we will see, that is often done in a small number of iterations (often only 1) by the DZ algorithm. Before solving each node's subproblem, we make use of lower bounds to check if the node can be pruned immediately and avoid calling the DZ algorithm.

Our results show that the B&B algorithm can handle in practical time small and medium-sized instances of the MB_k problem. However, it is not suitable to solve large instances. Exactly solving large instances requires tighter bounds and a B&B developed for multiple processors.

This chapter is organized as follows. We start by presenting the details of the B&B algorithm in Section 3.1. In Section 3.2, we show the computational performance of the algorithm and derive conclusions on its behavior for different dataset features. Section 3.3 presents some empirical studies on further aspects of the algorithm, such as the effect of providing an initial solution and the application of lower bounds based on relaxing the mixed-integer formulations of problem (3.1). Finally, Section 3.4 draws some conclusions.

[1] A k-subset is a subset with cardinality k.

3.1 The proposed branch-and-bound framework

We can think as the MB_k problem as the problem of finding the k-subset of $\mathcal{P}$ whose minimum enclosing ball is the one with the smallest radius among all k-enclosing balls. Then, the search space is the set of all k-subsets of $\mathcal{P}$, which can be represented as a tree, as Figure 3.2 shows for the case of $m = 5$ and $k = 3$.

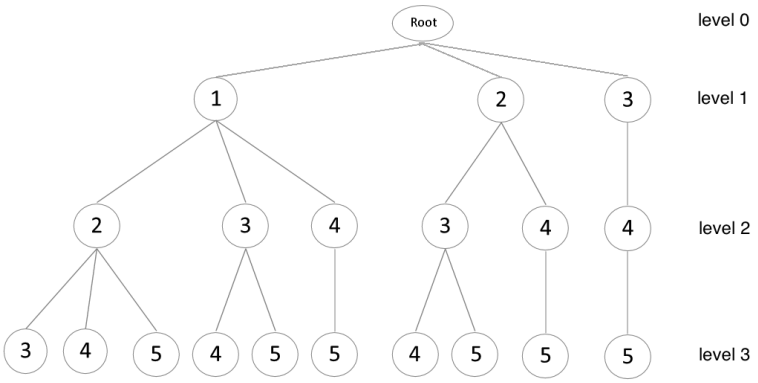


Figure 3.2: Search tree for $m = 5$ and $k = 3$.

The root of the tree represents the empty set. Each node on the tree is assigned one point of $\mathcal{P}$. The set of nodes on the path from the root to a node corresponds to a subset of $\mathcal{P}$ with at most k points. Thus, each path from the root to a leaf represents a different k-subset of $\mathcal{P}$.

One way to assign points to nodes of the tree is as Figure 3.2 shows, that is, the shape is static and points are assigned to nodes in lexicographic order. Instead, in our branch-and-bound algorithm the tree is dynamic, meaning that which points are assigned to which nodes is decided as we search the tree in a way that aims at decreasing the number of explored nodes. Still, each path from the root to the leaf corresponds to a different k-subset.

Besides having been considered in [19, 6] as mentioned before, branch-and-bound methods using the same search tree have also been used in [81, 108, 24] to address the feature selection problem.

3.1.1 Preliminaries

Before proceeding, we introduce some notations about the search tree.

- N_0: root node;
- $l(N)$, level of node N: number of nodes on the path from N to the root (without counting the root); by definition the level of the root is 0;
- $p(N)$: point of $\mathcal{P}$ assigned to node N;
- $Pt(N)$: set of points assigned to nodes on the path from N_0 to N (including N);
- $ST(N)$: set of points assigned to nodes on the subtree of N (excluding N);
- $r(N)$, $x(N)$: radius and center of $MB(Pt(N))$ respectively;
- $\mathcal{C}(N)$: list of child nodes of N, that is, successor nodes of N at level $l(N)+1$;
- r^*, x^*: radius and center, respectively, of the best solution found at any stage (also called *incumbent* or *upper bound*).

One possible way to assign points to nodes is lexicographically, as it was shown in Figure 3.2. That is, the $m-k+1$ child nodes of the root are assigned to points p_1, p_2, ..., p_{m-k+1}, in this order from left to right; and, every time a node at level l assigned to p_j is branched, its child nodes are assigned to points p_{j+1}, p_{j+2}, ..., $p_{m-k+l+1}$, in this order from left to right. We introduce one last useful notation:

- $i(N)$, index of node N: index of the point from $\mathcal{P}$ assigned to N, assuming the nodes on the tree are organized lexicographically; by definition, the index of the root is 0.

The tree corresponding to m points and $k \leq m$, has the following properties:

i. The number of nodes at level l is $\binom{m-k+l}{l}$, for $l = 1, \ldots, k$; the tree has $\binom{m}{k}$ leaves;
ii. The number of different points of $\mathcal{P}$ assigned to successor nodes of N is $m - i(N)$;
iii. $|\mathcal{C}(N)| = m - k + l(N) - i(N) + 1$;
iv. The indexes of the child nodes of N are $i(N)+1$, $i(N)+2$, ..., $m-k+l(N)+1$, assigned in this order from left to right.

The branch-and-bound algorithm starts with an initial solution x^*, r^*, corresponding to the center and radius of a k-enclosing ball. The search begins at level 0, when the root

node is created and added to the pool of *live* nodes $\mathcal{L}$. A *live* node is a node that has been explored, that is, whose associated problem has been solved, but whose immediate successors (child nodes) have not been generated yet. The problem associated with a node N is the following: find $x(N)$ and $r(N)$, the center and radius, respectively, of the minimum enclosing ball that encloses the points assigned to nodes on the path from the root to N, that is,

$$\begin{aligned} MB(Pt(N)): \quad & \min_{x\in\mathbb{R}^n,\, r\in\mathbb{R}} \quad r \\ & \text{s.t.} \qquad \|p_i - x\| \le r, \quad \forall p_i \in Pt(N). \end{aligned} \tag{3.2}$$

At each iteration, and following some selection rule, a node N is removed from $\mathcal{L}$ to be branched. If, for that node, $r(N) \ge r^*$, then N's successor nodes can be pruned since $r(N)$ is a lower bound on the best solution that can be found in its subtree. On the other hand, if $r(N) < r^*$, each child node is created and the subproblem (3.2) corresponding to each child is solved. If the radius r corresponding to a child node is such that $r \ge r^*$, then that child node can be immediately disregarded. Otherwise, if the ball corresponding to that child encloses k points of $\mathcal{P}$, then its radius becomes the new incumbent. If neither of these situations occurs, the child node is added to $\mathcal{L}$ to be branched later. The algorithm terminates when $\mathcal{L}$ is empty, meaning that all nodes were either analyzed or cut off from the tree. The solution of (3.1) is the radius r^* and center x^* as defined at termination.

These considerations are summarized formally in Algorithm 3.1.

3.1.2 Solving the MB problem at each node

At each node of the search tree, we propose using the DZ dual algorithm (see Section 2.2) to solve the MB problem corresponding to that node. Suppose node N was selected to be branched from the pool of live nodes. Let $C_1, ..., C_{|\mathcal{C}(N)|}$ denote its child nodes. Once N is branched, we need to solve problem $MB(Pt(C_j))$ for each child node C_j, where $Pt(C_j) = Pt(N) \cup \{p(C_j)\}$, with $p(C_j)$ the point assigned to C_j. If $p(C_j)$ is covered by $B(x(N), r(N))$, then the solution of $MB(Pt(C_j))$ is the same as of $MB(Pt(N))$. Otherwise, we apply the DZ algorithm to solve $MB(Pt(C_j))$, starting from $r(N)$ and $x(N)$, and considering $p(C_j)$ as the point to enter the basis. Note that, $r(N)$ and $x(N)$ is a dual feasible solution to $MB(Pt(C_j))$.

Algorithm 3.1 A branch-and-bound algorithm for the MB_k problem

input: $\mathcal{P}$, and an initial solution x^*, r^*;
output: x^*, r^*;

1: $\mathcal{L} = \{N_0\}$.
2: **while** $\mathcal{L} \neq \emptyset$ **do**
3: Select a node N from $\mathcal{L}$.
4: **if** $r(N) \geq r^*$ **then**
5: Prune N; Go to line 3;
6: **end if**
7: Get $\mathcal{C}(N) = \{C_1, C_2 \ldots, C_{|\mathcal{C}(N)|}\}$. ▷ Branch node N.
8: **for each** node C_j in $\mathcal{C}(N)$ **do**
9: Get $r(C_j)$ and $x(C_j)$ by solving $\mathrm{MB}(\mathrm{Pt}(C_j))$; ▷ Explore each child node.
10: **if** $r(C_j) \geq r^*$ **then**
11: Prune C_j;
12: **else**
13: **if** $l(C_j) = k$ or $B(x(C_j), r(C_j))$ encloses at least k points of $\mathcal{P}$ **then**
14: $r^* = r(C_j)$; $x^* = x(C_j)$; ▷ Update incumbent.
15: Prune C_j;
16: **else**
17: Add C_j to $\mathcal{L}$;
18: **end if**
19: **end if**
20: **end for**
21: **end while**

Typically, solving $\mathrm{MB}(\mathrm{Pt}(N) \cup \{p(C_j)\})$ starting from the solution of $\mathrm{MB}(\mathrm{Pt}(N)$ requires a modest number of iterations (often 1 iteration suffices). Moreover, since at each DZ iteration the radius increases, if it ever becomes larger than the incumbent, we can immediately prune that child node and its successors.

A lower bound on the solution of $\mathrm{MB}(\mathrm{Pt}(C_j))$

Before solving $\mathrm{MB}(\mathrm{Pt}(C_j))$, we can obtain a quick lower bound on its optimal radius, and use it to prune C_j without having to call the DZ algorithm. If $p(C_j)$, the point corresponding to C_j, is not covered by $\mathrm{MB}(\mathrm{Pt}(N))$, then it is known to be part of the optimal basis of $\mathrm{MB}(\mathrm{Pt}(C_j))$ (see e.g. [33] or Theorem 4.14). Since $p(C_j)$ is on the boundary of the optimal ball from $\mathrm{MB}(\mathrm{Pt}(C_j))$, from the triangle inequality, we conclude that

$$\|p(C_j) - p\| \leq \|p - x(C_j)\| + \|p(C_j) - x(C_j)\| \leq 2r(C_j), \quad \forall p \in \mathrm{Pt}(N).$$

Using the same observation, we can conclude that the diameter of the optimal ball obtained from $MB(Pt(C_j))$ is smaller than the distance between $p(C_j)$ and the farthest point on the boundary of $B(x(N), r(N))$, that is $\|p(C_j) - x(N)\| + r(N)$. Thus, we obtain:

$$\max_{p \in Pt(N)} \tfrac{1}{2} \|p(C_j) - p\| \leq r(C_j) \leq \tfrac{1}{2} \left(\|p(C_j) - x(N)\| + r(N) \right). \tag{3.3}$$

We calculate the pairwise distances between points of $\mathcal{P}$ in a preprocessing phase, in order to be able to apply the lower bound above in every node. Note that it is not necessary to scan the distances corresponding to all the points of $Pt(N)$ to find (3.3). Only the points $p \in Pt(N)$ that satisfy $\langle p - x(N), p(C_j) - x(N) \rangle \leq 0$ need to be considered [42]. However, since the pairwise distances are always available, calculating such inner products at every node for all points in $Pt(N)$ offers no advantage.

The next lemma gives us a weaker lower bound that can be used to avoid calculating the lower bound in (3.3). This can be advantageous, especially when the number of points in $Pt(N)$ is large. A proof can be found, for instance, in [49].

Lemma 3.1. *Let $\mathcal{P} \subset \mathbb{R}^n$ be a set of points. Any closed halfspace that passes through the center of the optimal ball of $MB(\mathcal{P})$ contains at least one point on the boundary of the ball.*

Consider the hyperplane perpendicular to $p(C_j) - x(N)$ that passes through $x(N)$. The distance between $p(C_j)$ and v, any point on the intersection of the hyperplane with the boundary of $B(x(N), r(N))$, is a lower bound on the maximum distance between $p(C_j)$ and any point covered by $B(x(N), r(N))$. Thus,

$$\tfrac{1}{2} \sqrt{\|p(C_j) - x(N)\|^2 + r(N)^2} = \tfrac{1}{2} \|p(C_j) - v\| \leq \max_{p \in Pt(N)} \tfrac{1}{2} \|p(C_j) - p\|. \tag{3.4}$$

Note that, this lower bound is only useful when $\|p(C_j) - x(N)\| > \sqrt{3} r(N)$, otherwise it yields a value smaller or equal to $r(N)$.

Algorithm 3.2 summarizes how to solve $MB(Pt(C_j))$ employing the lower bounds in this discussion above.

Algorithm 3.2 Solving $MB(Pt(C_j))$ for each child node of N

input: $x(N), r(N), p(C_j), r^*$
output: $r(C_j), x(C_j)$

Calculate $t = \|p(C_j) - x(N)\|$.
if $t \leq r(N)$ **then**
    $r(C_j) = r(N)$; $x(C_j) = x(N)$; **return** ▷ $p(C_j)$ *is enclosed by* $B(x(N), r(N))$
end if
if $r^* \leq \frac{1}{2}(t + r(N))$ **then**
    if $\frac{t}{r(N)} > \sqrt{3}$ and $r^* \leq \frac{1}{2}\sqrt{t^2 + r(N)^2}$ **then**
        $r(C_j) = +\infty$; $x(C_j) = [\,]$; **return** ▷ C_j *is pruned*
    else
        if $r^* \leq \max_{p \in Pt(N)} \frac{1}{2}\|p(C_j) - p\|$ **then**
            $r(C_j) = +\infty$; $x(C_j) = [\,]$; **return** ▷ C_j *is pruned*
        end if
    end if
end if
Get $r(C_j)$ and $x(C_j)$, by solving $MB(Pt(C_j))$ starting from $x(N)$ and $r(N)$.
return

3.1.3 The tree design

Given a tree topology, different point-node assignments may be defined. A straightforward way would be to simply design the tree using the lexicographic order of the points (as shown in Figure 3.2), or to assign points to nodes randomly. Though simple, these strategies are not efficient for general inputs since they do not exploit the geometrical aspects of the problem nor the particular structure of the search tree.

A dynamic point-node assignment scheme

The tree of all k-subsets of a set is not symmetric, in the sense that different nodes have different subtree sizes and shapes, depending on their position on the tree. We aim at assigning points to nodes on the denser part of the tree (left side of the tree whose nodes have a large number of successors) that yield large radii; and the opposite for the less dense part of the tree (right side, whose nodes have few successors). As a consequence, provided we have an incumbent that is good enough, nodes on the denser part are more likely to be pruned.

We propose the following node-point assignment scheme. When a live node N is

Algorithm 3.3 Branch node N

input: Node N

output: The sorted list of child nodes $\{C_1, C_2 \ldots, C_{|\mathcal{C}(N)|}\}$

for each $q_i \in ST(N)$ **do**
    Calculate $t_i = \|q_i - x(N)\|$.
end for
Get $\hat{t}$, the vector of the $|\mathcal{C}(N)|$ largest values of vector t sorted in descending order.
for $j = 1, ..., |\mathcal{C}(N)|$ **do**
    Create node C_j.
    Get $\hat{q}_j \in ST(N)$ s.t. $\hat{t}_j = \|\hat{q}_j - x(N)\|$.
    $p(C_j) = \hat{q}_j$;
    $Pt(C_j) = Pt(N) \cup \{\hat{q}_j\}$;
    $ST(C_j) = ST(N) \setminus \{\hat{q}_1, \ldots, \hat{q}_j\}$.
end for

selected to be branched, first, the distances between $x(N)$ and each of the points of $ST(N)$ are calculated and sorted in descending order; then the points are assigned to nodes such that the j-th child of N corresponds to the point with the j-th largest distance from $x(N)$. This in-level node-point assignment scheme is expected to be effective because, for a general point-set Q and a point q, the distance from q to the optimal center of $MB(Q)$ is a good predictor of the magnitude of the optimal radius of $MB(Q \cup \{q\})$. Additionally, the rightmost child has the closest points to the parent's ball in its subtree (a path), which may result in finding a new (and good) incumbent.

This node-point assignment procedure is formalized in Algorithm 3.3. Every time a node is created, the distances between $x(N)$ and each of the $m - i(N)$ points assigned to nodes on N's subtree need to be calculated. Then, the $|\mathcal{C}(N)|$ largest distances need to be found and sorted. This procedure can be accomplished in $\mathcal{O}\left(m - i(N) + |\mathcal{C}(N)| \log(|\mathcal{C}(N)|)\right)$ for each node N [2].

Children of the root node

Now, we consider the problem of which points to assign to the $m - k + 1$ child nodes of the root. One option would be to select the first $m - k + 1$ points of $\mathcal{P}$ or simply randomly. We

[2] Finding the k-th largest element in a set of n numbers, corresponds to the *selection problem* (finding the $(n - k + 1)^{\text{th}}$-order statistic) which can be solved in $\mathcal{O}(n)$ time. After finding the k-the largest element, finding the $k - 1$ elements larger than it, can be done in $\mathcal{O}(n)$ time. Finally, sorting of those $k - 1$ largest elements can be accomplished in $\mathcal{O}(k \log k)$ time. See [27, §9] for details.

propose an approach that performed well in practice, and that consists of first calculating the minimum enclosing ball of all points in $\mathcal{P}$, and then selecting the $m-k+1$ farthest points from its center as the children of the root. The intuition behind this heuristic is that each subtree of a child of the root will be assigned to explore feasible solutions of diametrically opposite areas of the input set, which will hopefully result in less redundant work.

The minimum solution tree

Take any node N from the tree. The "last" child of N, the rightmost successor node, has a path as a subtree. Instead of evaluating each node on such path one at a time, we can jump directly to the leaf and solve the corresponding MB problem. The path below such one-degree nodes can, therefore, be merged into a *special* single node, yielding a modified topology of the search tree called the *minimum solution tree*[3], as shown in Figure 3.3.

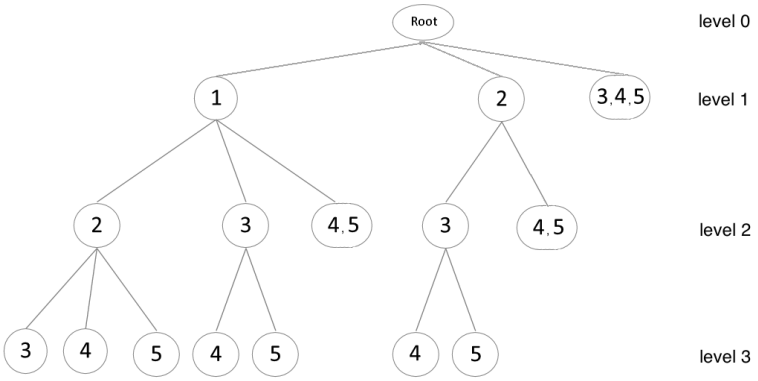


Figure 3.3: Minimum solution search tree for $m = 5$ and $k = 3$.

Using the minimum solution tree yields a decrease in the total number of DZ iterations performed, because the number of iterations necessary to solve the subproblem at a *special* node is always less or equal than the total number of iterations corresponding to solve all the subproblems from the path corresponding to that node.

[3]The concept of minimum solution tree was first introduced by Yu and Yuan in [108] for a branch-and-bound algorithm for solving feature selection problems.

3.1.4 Search strategy

There are many ways one can choose which live node is going to be explored next. Choosing a successful strategy for a general input is highly intertwined with the design of the tree.

A depth-oriented search strategy seems to be the most advantageous in this case, as it allows reaching leaves faster and keeps the size of $\mathcal{L}$ relatively small. Additionally, given the shape of the tree, it seems best to search the tree in a right-to-left manner, that is, a strategy that explores the less dense part of the tree early in an attempt to find a good bound before the dense part of the tree is explored.

Whenever a node is branched, assume that its child nodes are generated in left-to-right order. We propose using a pure *depth-first-search (DFS)* strategy. In this strategy, $\mathcal{L}$ is maintained as a stack; that is, the last node added is the first one to be removed (*last-in-first-out* (LIFO)). As a consequence, when a node is branched, its child nodes are added to $\mathcal{L}$ as they are created, that is, in a left-to-right manner. Given the point-node assignment scheme proposed in Section 3.1.3, that implies that the less dense part of a node's subtree, which is also the most promising in finding a good lower bound, is searched first. Thus, unless a leaf is reached or a node is pruned, at every iteration the B&B descends one level, therefore (good) feasible solutions are expected to be found early on. Note that the point-node assignment combined with DFS can be seen as a combination of depth-first-search and best-first-search (DFS is the overall principle, and when the choice is to be made between nodes at the same level of the tree, the most promising one is selected).

Finally, when DFS is adopted, the maximum size of $\mathcal{L}$ for a given $\mathtt{k}$ can be determined, as Theorem 3.2 states, allowing the required memory to be allocated beforehand.

Theorem 3.2 (Maximum length of $\mathcal{L}$)**.** *When DFS is used and the minimum solution tree is adopted, the maximum length of* $\mathcal{L}$ *is* $\mathtt{m} - \mathtt{k}$*. This upper bound is always attained.*

Proof. Suppose the worst case scenario: no subtree is ever pruned, that is, every node will be added to $\mathcal{L}$ at some point, with the exception of the leaves (which are disregarded after being evaluated). Consider any stage of the algorithm when a node $\mathtt{N}$ was removed from $\mathcal{L}$

to be branched. Note that N cannot be a leaf, therefore $l(N) \leq k-1$. Let $N_0, N_1, \ldots, N_{l(N)}$ be the nodes on the path from the root N_0 to $N \equiv N_{l(N)}$. Recall the definition of index of N, $i(N)$. Together, $i(N)$ and $l(N)$ uniquely define the shape of N's subtree. As mentioned before, the number of immediate successors of any node N is $|\mathcal{C}(N)| = m-k+l(N)-i(N)+1$. Moreover, $i(N) \leq m-k+l(N)$ always holds.

If $l(N) = 0$, then N is the root, N_0, and $\mathcal{L}$ is empty. By definition $i(N_0) = 0$, so $b(N_0) = m-k+1$. Each child of N_0 will be added to $\mathcal{L}$ with the exception of the last one (since it is a leaf). After N_0 is branched, $\mathcal{L}$ has size $m-k$.

Consider now the case $l(N) \geq 1$. Since N was removed from $\mathcal{L}$, it means all nodes on the path from the root to N cannot be in $\mathcal{L}$. Additionally, $\mathcal{L}$ cannot contain any nodes of higher level than $l(N)$. Let j be any level between 1 and $l(N)$. The indexes of the nodes at level j that are still in $\mathcal{L}$ are $i(N_{j-1})+1, \ldots, i(N_j)-1$. Thus, the number of nodes in $\mathcal{L}$ is:

$$\sum_{j=1}^{l(N)} (i(N_j) - i(N_{j-1}) - 1) = i(N) - l(N).$$

Once N is branched, its $m-k+l(N)-i(N)+1$ immediate successor nodes are created and evaluated, and with the exception of the last one, they are all added to $\mathcal{L}$. The size of $\mathcal{L}$ then becomes $m-k$. □

Remark 3.3. In Section 3.1.1 we mentioned that the algorithm explores the child nodes first and only then adds them to the pool of live nodes, that is, it follows an *eager* strategy. Given that DFS is used, this is a natural choice over the *lazy* strategy (where the child nodes are generated and added to the pool with their parent's lower bound instead, and only explored after selected out of the pool). The lazy strategy makes sense if a better incumbent than the parent's node is expected to be found before its children are selected out of the pool, avoiding the need to solve the children's subproblems. However, that cannot happen when DFS is used, because the nodes that will be explored after a node N is taken out of the pool are going to be its successors, whose radius will always be worse than N's radius. Thus, an incumbent with radius better than the parent's radius is not possible to be found before N's child nodes are selected out of the tree.

3.1.5 Finding an initial solution

An initial solution, that is, a k-enclosing ball of $\mathcal{P}$, can be used to initialize $\mathtt{r}^*$ in order to speed up pruning. There are many possible approaches to get such an upper bound for the optimal radius of the $\mathrm{MB}_\mathtt{k}$ problem. Some quick approaches include randomly selecting a k-subset of $\mathcal{P}$ and find its minimum enclosing ball, or picking a random point in $\mathcal{P}$, find the $(\mathtt{k}-1)$-closest points to it, and find the minimum enclosing ball of the resulting set.

Two other options are called *spherical ordering* and *spherical peeling*[4]. Spherical ordering starts by finding the minimum enclosing ball of $\mathcal{P}$, then it picks the k closest points to the center of that ball and finds the minimum enclosing ball of those k points. In spherical peeling, we also start by finding the minimum enclosing ball of the whole set $\mathcal{P}$, and then, iteratively we discard one of the points lying on the boundary of the ball and find the minimum enclosing ball of the remaining points, until only k points remain on the set to enclose. This method is more suitable for the case when k is large, and it can be done efficiently using the FSW algorithm for the MB problem (see Section 2.3). Instead of starting with the minimum enclosing ball of $\mathcal{P}$, spherical ordering and spherical peeling may also start with the minimum enclosing ball of a subset of $\mathcal{P}$ with k points randomly picked.

One can also use more elaborate methods such as some of the approximation algorithms discussed in Section 1.1.2. However, the amount of effort in finding a good initial solution should be in line with how much of an advantage it can offer in reducing the computational work of the B&B algorithm. In Section 3.3.1, we provide some studies on that aspect.

3.2 Computational study

In this section, we present some computational results of the B&B algorithm. We start by showing results for randomly sampled datasets corresponding to different distributions, number of points, dimension, and values of k. Then, we consider the application of facility location with outliers and show some results using real datasets. Finally, we compare the

[4] Spherical ordering and spherical peeling were also suggested in [7] for the more general problem of the minimum volume ellipsoid with partial inclusion.

B&B against the general purpose solver Gurobi.

Our experiments were conducted on MATLAB R2018b (version 9.5.0) on a Mac with an Intel Core i5 1.6 GHz processor, 8GB RAM, and running Mac OS X version 10.11.6.

Since we learned in Section 2.6 that our implementation of the original DZ algorithm is often faster for small dimensions than the implementation with the modification we propose in Section 2.4, we used the former in our computational experiments of the B&B.

3.2.1 Performance of the branch-and-bound algorithm

With the goal of studying the behavior of the B&B in sets with different properties, we considered the following randomly generated datasets:

Uniform: points uniformly sampled on a unit ball;

Normal: points sampled from a multivariate standard normal distribution;

Exponential: each coordinate of the points independently sampled from an exponential distribution with mean equal to 1;

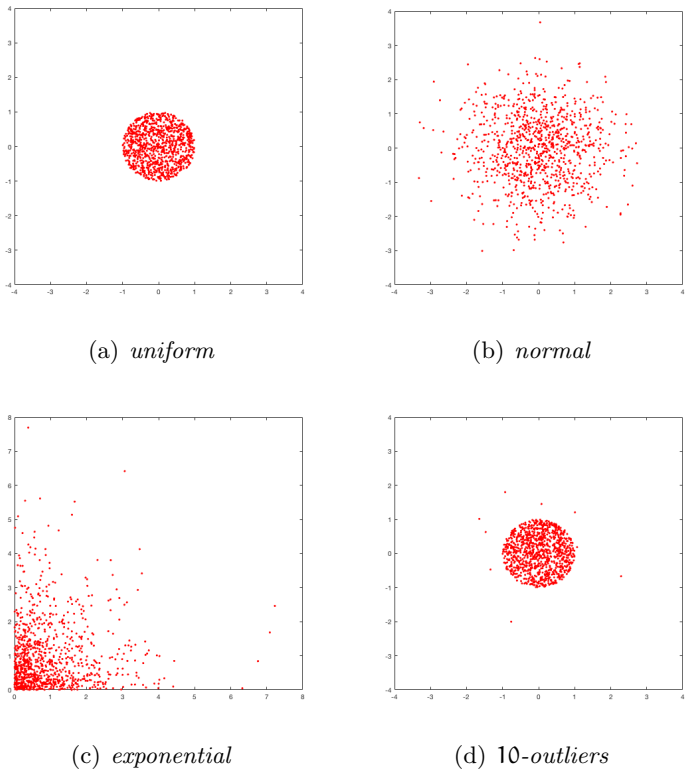
$\mathtt{b}$-*outliers*: points uniformly sampled on a unit ball and $\mathtt{b}$ artificial outliers uniformly sampled from the ring with same center and radius between 1 and 3.

Figure 3.4 illustrates 2-dimensional instances of each of these datasets.

The methods used to obtain an initial solution were the following: for the datasets *normal*, *ball*, and $\mathtt{b}$-*outliers*, we used the spherical ordering method; and for the *exponential* dataset, we used spherical peeling. Based on the characteristics of the datasets, we expect these methods to provide good upper bounds.

Tables 3.1, 3.2, and 3.3 report different parameters observed when we run the BB algorithm for the datasets *normal*, *uniform*, and *exponential*, for different dimensions, number of points, and values of $\mathtt{k}$. The results from Table 3.4 concern the $\mathtt{b}$-*outliers* datasets with $10,000$ points. Tables 3.1-3.4 report the average and relative standard deviation[5] of the following parameters:

[5]Labeled as "dev", the relative standard deviation is the standard deviation divided by the mean and multiplied by 100.



(a) *uniform* (b) *normal*

(c) *exponential* (d) 10-*outliers*

Figure 3.4: Illustration of the different randomly generated datasets with 1000 points in $\mathbb{R}^2$.

- **nodes**: number of explored nodes;
- **iters**: number of DZ algorithm's iterations;
- **iters/node**: ratio between the number of DZ iterations and explored nodes;
- **time**: total elapsed CPU time in seconds.

A first look at the tables reveals that the algorithm explores a very small fraction of the nodes of the tree. For instance, in the case of 1000 points, the total number of nodes of the tree is $9.96\mathsf{E}+84$ when $\mathsf{k}=50$, and $6.43\mathsf{E}+242$ when $\mathsf{k}=250$. In Table 3.1, we observe that the number of explored nodes is in the order of $1\mathsf{E}+4$ for $\mathsf{k}=50$, and less than $1\mathsf{E}+7$ for $\mathsf{k}=250$, for all the studied datasets.

In general, we observe that the algorithm performs best when k is close to m. In this case, the tree nodes have subtrees that are "tall and skinny", and each node has a small number of children (when we compare to the case when k is small). This fact, combined

		nodes		iters		iters/node	time	
	k	mean	dev	mean	dev	mean	mean	dev
normal	50	8.41E+03	53%	9.12E+03	53%	1.086	7.60	14%
	100	4.39E+04	60%	4.58E+04	60%	1.045	14.44	39%
	250	3.40E+05	26%	3.46E+05	26%	1.018	83.54	30%
	750	2.28E+05	25%	2.31E+05	25%	1.013	53.22	25%
	900	2.45E+04	34%	2.52E+04	34%	1.029	5.59	43%
	990	9.49E+01	80%	1.18E+02	77%	1.295	0.03	73%
uniform	50	2.07E+04	31%	2.23E+04	30%	1.081	10.12	15%
	100	1.40E+05	16%	1.46E+05	16%	1.039	36.31	16%
	250	1.36E+06	32%	1.38E+06	32%	1.016	362.93	34%
	750	3.24E+06	16%	3.25E+06	16%	1.004	609.59	15%
	900	5.28E+05	14%	5.32E+05	14%	1.006	90.86	13%
	990	7.77E+02	25%	8.75E+02	25%	1.128	0.28	45%
exponential	50	2.22E+04	22%	2.38E+04	22%	1.073	11.98	14%
	100	4.34E+04	14%	4.51E+04	14%	1.040	17.57	10%
	250	8.86E+04	37%	9.05E+04	36%	1.022	33.14	36%
	750	1.72E+05	34%	1.75E+05	34%	1.016	50.04	35%
	900	2.53E+04	30%	2.62E+04	30%	1.032	7.69	34%
	990	1.19E+02	24%	1.50E+02	21%	1.265	0.08	37%

Table 3.1: Performance of the B&B for different 2-dimensional datasets with 1000 points. The results correspond to the averages of 10 instances of each dataset.

		nodes		iters		iters/node	time	
	k	mean	dev	mean	dev	mean	mean	dev
norm.	50	4.37E+07	55%	4.52E+07	55%	1.034	9508.72	53%
	990	4.08E+03	52%	5.70E+03	53%	1.381	2.26	56%

Table 3.2: Performance of the B&B for 10-dimensional *normal* datasets with 1000 points. The results correspond to the averages of 10 instances of each dataset.

		nodes		iters		iters/node	time	
	k	mean	dev	mean	dev	mean	mean	dev
normal	10	3.18E+03	58%	3.51E+03	57%	1.112	0.73	53%
	30	3.94E+04	38%	4.37E+04	39%	1.109	9.01	41%
	55	3.98E+04	48%	4.54E+04	48%	1.139	9.56	53%
	75	1.30E+04	35%	1.56E+04	37%	1.193	3.54	44%
	90	9.15E+02	35%	1.24E+03	35%	1.353	0.32	57%
uniform	10	1.34E+04	26%	1.45E+04	27%	1.079	2.66	23%
	30	6.54E+05	37%	7.06E+05	37%	1.079	144.69	39%
	55	3.06E+06	69%	3.39E+06	69%	1.109	837.38	73%
	75	1.20E+06	41%	1.42E+06	40%	1.183	392.83	42%
	90	3.76E+04	64%	5.36E+04	66%	1.416	15.98	72%
exponential	10	2.51E+03	37%	2.93E+03	35%	1.174	0.68	33%
	30	1.63E+04	42%	1.84E+04	41%	1.130	4.01	43%
	55	1.33E+04	51%	1.55E+04	49%	1.182	3.80	46%
	75	3.96E+03	32%	4.82E+03	32%	1.217	1.07	36%
	90	2.54E+02	67%	3.75E+02	62%	1.510	0.08	58%

Table 3.3: Performance of the B&B for different 10-dimensional datasets with 100 points. The results correspond to the averages of 10 instances of each dataset.

			nodes		iters		iters/node	time	
b	n	k	mean	dev	mean	dev	node	mean	dev
10	10	9990	6.83E+01	71%	2.70E+02	94%	3.75	0.31	99%
50	10	9950	3.59E+03	106%	1.17E+04	116%	2.84	14.85	115%
100	10	9900	4.30E+04	173%	1.29E+05	194%	2.44	178.83	208%
10	100	9990	5.56E+01	101%	6.23E+02	64%	16.02	10.29	52%
50	100	9950	7.85E+02	76%	2.31E+03	68%	3.29	28.27	62%
100	100	9900	9.55E+03	131%	1.50E+04	120%	1.81	166.60	120%
250	100	9750	1.38E+05	111%	1.77E+05	112%	1.25	1814.38	119%

Table 3.4: Performance of the B&B for 10000-point b-*outliers* datasets for different values of b and dimension n. The results correspond to the averages of 10 instances of each dataset.

with the point-node assignment scheme we use, tends to create large radii at nodes from the upper levels of the tree, resulting in a large number of nodes pruned in the early stages of the run. In general, the hardest situations for the B&B are when k is close to $\frac{m}{2}$, which is natural since that is when the number of total nodes of the tree attains its maximum.

The algorithm is affected by the *curse of dimensionality*. This is easily seen when we compare Tables 3.1 and 3.2 for the *normal* datasets with $k = 990$. Just by increasing the dimension from 2 to 10 causes the number of explored nodes to increase dramatically. No other values of k were possible to be tested in Table 3.2 as they were not possible to run in a practical amount of time (the case $k = 50$ took on average about 3 hours to run). This increase in complexity is explained by the following. Consider a node that is branched, and its minimum ball is inflated in order to cover a point corresponding to one of its child nodes. When the dimension is small, this inflated ball has a good chance of covering many other points in the vicinity. This is true, especially if the point to cover is far from the current center. When the dimension is large, however, the inflated ball has less chance to cover many other points unless the number of points also increases (exponentially) with the dimension. In other words, by increasing the dimension, we have a lot more room to disperse the points and so the chance of an inflating ball to capture many other points decreases.

Another issue with increasing the dimension is that the number of dual iterations per node increases. When a ball is inflated and translated to enclose a new point, as the dimension increases, the chance that a previously enclosed point becomes unenclosed also increases (this is also more frequent when k is large). However, the number of dual iterations per node remains small for small dimensions. This validates the choice of the DZ algorithm for solving minimum enclosing ball subproblems for each node. One can also consider modifying the algorithm and, instead of solving up to optimality each node's subproblem, only doing one, or another fixed number of, DZ iterations at each node, and then solve up to optimality when a leaf is reached. This may reduce the number of DZ iterations for larger dimensions. However the bounds produced at each node will not be as strong, so the number of explored nodes will likely increase.

The most challenging datasets for the B&B seem to be those that contain many k-subsets whose minimum enclosing ball radii are close to the radius of the optimal k-enclosing ball. This is the case of the *uniform* datasets. In these instances, a large portion of the tree has to be scanned until the optimal solution is found, and its optimality is proved. On the other hand, when the data has many outlying points, like in the *normal* or the *exponential* cases, the B&B tends to place many of the outlying points on the left side of the tree, and, since these points generate large radii, the corresponding nodes are quickly pruned together with their large subtrees. This behavior is observed in the b-*outliers* datasets (Table 3.4) where instances with $10,000$ points and dimensions 10 and 100 were possible to be solved in a practical amount of time.

Finally, we present some results with real data. Considering the application in facility location, we tested our B&B algorithm in datasets[6] whose points correspond to cities in Canada, Japan, and the U.S.A. Table 3.5 contains the number of points in the dataset (m), the percentage of points to be enclosed, the number of explored nodes, DZ iterations, and CPU time in seconds for these datasets. Figures 3.5 show the optimal k-enclosing balls. We observe that, despite the large number of points in these datasets, the B&B algorithm managed to solve these instances in a short amount of time.

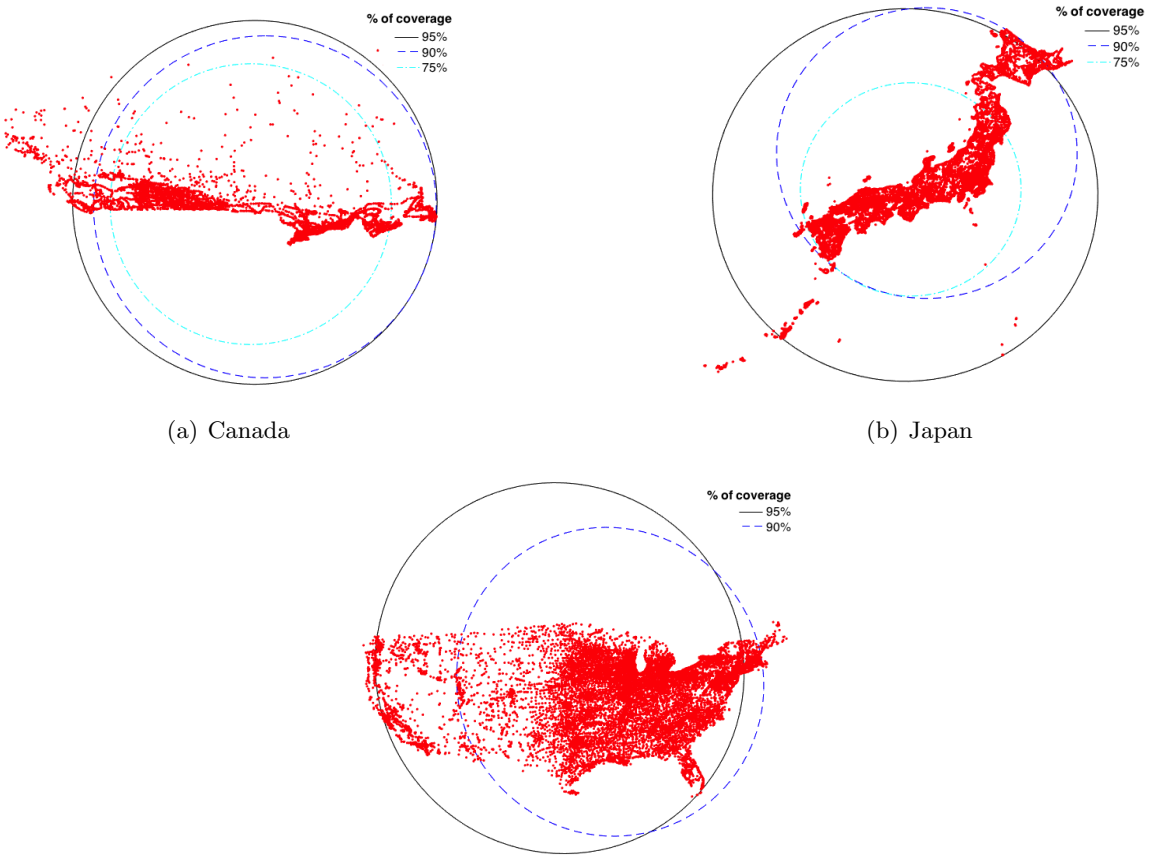
3.2.2 Comparison with other methodologies

Considered the following mixed-integer quadratic programming (MIQP) formulation of the minimum k-enclosing ball problem:

$$\begin{aligned} \min_{r,x,\beta_j} \quad & r^2 \\ \text{s.t.} \quad & \|p_j - x\|^2 \leq r^2 + (1-\beta_j)M, \quad p_j \in \mathcal{P} \\ & \sum_{j=1}^{m} \beta_j \geq k \\ & \beta_j \in \{0,1\}, \quad j = 1, ..., m, \end{aligned}$$

with $M = (\text{diam}(\mathcal{P}))^2$.

[6] These datasets are available in http://www.math.uwaterloo.ca/tsp/world/countries.html.



(a) Canada

(b) Japan

(c) U.S.A.

Figure 3.5: Illustration of the optimal k-enclosing balls for different levels of coverage.

Country	m	%coverage	k	nodes	iters	time
Canada	4663	75%	3498	1.51E+05	1.56E+05	71.31
		90%	4197	1.54E+04	1.66E+04	8.38
		95%	4430	4.16E+03	4.64E+03	3.16
Japan	9847	75%	7385	1.65E+05	1.69E+05	161.85
		90%	8862	5.76E+04	5.91E+04	86.57
		95%	9354	2.19E+04	2.26E+04	17.52
U.S.A.	13509	90%	12159	3.53E+05	3.59E+05	1060.60
		95%	12834	2.18E+06	2.22E+06	2026.80

Table 3.5: Performance of the B&B for different 2-dimensional datasets corresponding to Canada, Japan, and the U.S.A.

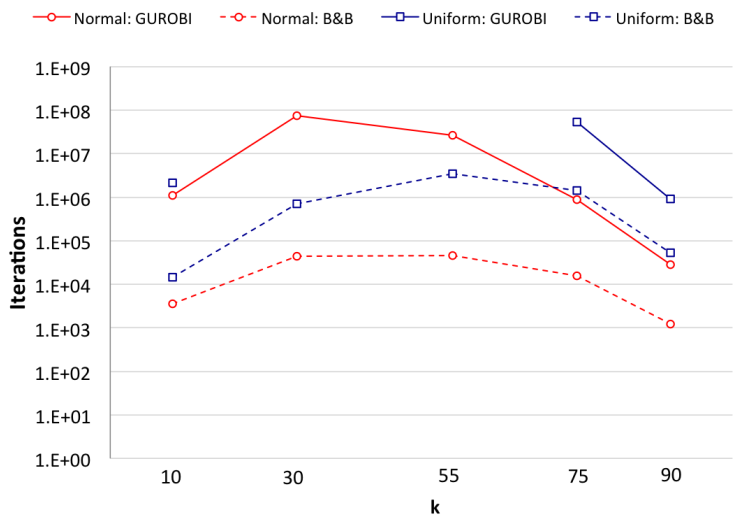
In this section, we compare the results of Table 3.3 with those of applying the general-purpose MIQP solver of Gurobi [53]. We used Gurobi version 8.0.0 with its MATLAB interface. We run Gurobi's solver on the instances of the same dataset that generated the results of Table 3.3. Additionally, the same initial solutions were given to Gurobi as a warm start. All Gurobi parameters were kept at their default values[7], with only the following exception: the dual simplex method was the algorithm chosen for solving the relaxation at each node, and the maximum running time allowed was 5000 seconds.

Figure 3.6 (a) reports the average number of Gurobi's dual simplex iterations in its Branch-and-Cut algorithm and the average number of Dearing and Zeck's iterations in our B&B, and Fig. 3.6 (b) reports the average CPU time of Gurobi and our B&B. We observe that our method was faster for all the datasets studied, except for the case of the uniform datasets when $k = 90$, in which we observe Gurobi to perform about twice as fast. The data corresponding to the *uniform* datasets for $k = 30$ and $k = 55$ is missing since Gurobi was not able to solve these problems to optimality in the allowed time. It is, of course, not surprising that a dedicated algorithm is superior to a general-purpose code. Still, the comparison is necessary in order to argue that off-the-shelf methods cannot successfully compete with our approach. Our comparison is based on the 100-point instances since Gurobi was not able to solve the instances with 1000 points within the allocated time (except for $k = 990$).

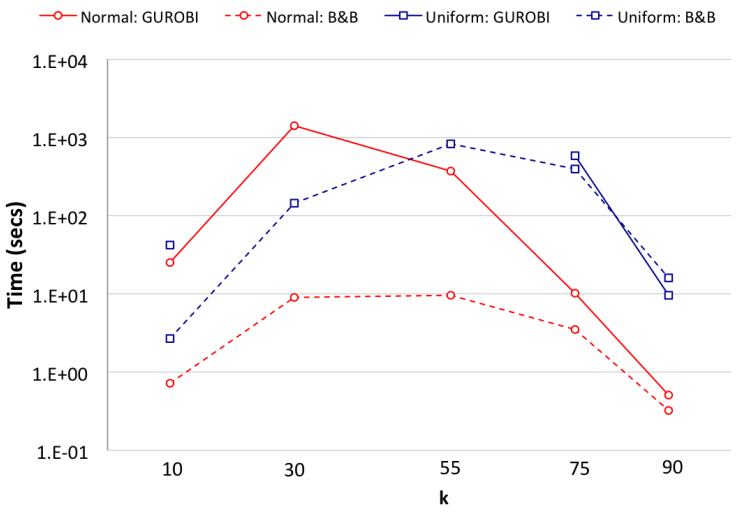
3.3 Further computational studies

In this section, we empirically study several aspects of the B&B algorithm presented in Section 3.1. We start by studying the effect of providing an initial solution as a warm start. Then, we introduce lower bounds resulting from relaxing the problem corresponding to finding the best solution on a node's subtree, and study how much of a reduction in the number of explored nodes these lower bounds produce. Finally, we compare the point-node assignment heuristic introduced in Section 3.1.3 with two other alternatives: assigning the

[7] We realize that by fine-tuning these parameters Gurobi may have a much better performance, but the idea here is to show that our special-purpose algorithm works better than a generic and default branch-and-bound method using the dual QP Simplex algorithm.



(a) Average total number of iterations as a function of k



(b) Average CPU time in seconds as a function of k

Figure 3.6: Gurobi vs. B&B for different 10-dimensional *normal* and *uniform* datasets with 100 points. The results correspond to the averages of 10 instances of each dataset.

points to nodes in a lexicographic order; and using the radius of each node as criterion to assign points to nodes.

3.3.1 The impact of using an initial solution

Starting the B&B algorithm with the knowledge of a good upper bound on the optimal solution can potentially result in a decrease in the number of explored nodes. To draw some conclusions of how advantageous a good initial solution can be, we performed two diametrically opposite experiments: for each dataset, we first ran our B&B algorithm without any initial solution, and recorded the optimal solution; next, we gave the optimal solution as initial solution to the B&B and ran it on the same dataset. The results are presented in Tables 3.6 and 3.7 show the average over 10 instances of the ratio (in %) corresponding to the number of explored nodes when the optimum solution was given as initial solution over the number of explored nodes when no initial solution was given. A value close to 100% means that giving an initial solution had little effect reducing the number of explored nodes.

In Tables 3.6 and 3.7, we observe that it is not unusual for the average ratios to be above 80%. This means that the existence of a good initial solution is observed to not cause a considerable reduction in the number of explored nodes. This is explained by either, when no initial solution is given, a good upper bound on the optimal radius is found early; or, a similar number of nodes needs to be scanned to prove optimality independently of the knowledge of a good upper bound. For instance, when there are many k-subsets with an enclosing ball with a radius close to the optimal k-enclosing ball radius, the latter case is likely to happen. This happens, for example, in the case of *uniform* datasets that are dense, like when $m = 1000$ and $n = 2$,

However, we have also observed some cases where the average ratio is small, meaning, unless a good initial solution is given, a good upper bound is found in a later stage of the algorithm's run. This is visible in the *exponential* datasets, in particular, when $n = 2$ and $m = 1000$. In this case, the knowledge of a good initial solution can make a notorious difference.

	k	nodes ratio (%)
normal	50	96.4%
	100	94.3%
	250	90.2%
	750	93.8%
	900	93.7%
	990	93.1%
uniform	50	90.4%
	100	81.4%
	250	86.3%
	750	93.8%
	900	95.7%
	990	93.9%
exponential	50	63.6%
	100	49.5%
	250	70.7%
	750	35.5%
	900	34.4%
	990	65.9%

Table 3.6: The ratio in percentage of the number of explored nodes when the optimum solution is given as initial solution over the number of explored nodes when no initial solution is given. The results report the averages for 10 instances with 1000 points and dimension 2.

	k	nodes ratio (%)
normal	10	86.4%
	30	87.7%
	55	84.6%
	75	83.3%
	90	85.6%
uniform	10	69.3%
	30	66.6%
	55	68.0%
	75	76.2%
	90	77.7%
exponential	10	67.5%
	30	57.7%
	55	64.5%
	75	76.6%
	90	89.2%

Table 3.7: The ratio in percentage of the number of explored nodes when the optimum solution is given as initial solution over the number of explored nodes when no initial solution is given. The results report the averages for 10 instances with 100 points and dimension 10.

The main conclusion, therefore, is that although an initial solution can sometimes reduce the number of explored nodes considerably, there is a good chance it may not. Therefore, spending too much effort at the outset of the algorithm to find a good upper bound can often be wasteful.

3.3.2 Additional lower bounds

Obtaining a lower bound for the radius of the optimal solution of the MB_k problem can be useful for cases where we are satisfied with suboptimal solutions, or cases where there is a limit on the number of iterations or in the running time of the algorithm. Having a good lower bound will indicate approximately how good our current best solution is.

A simple way to obtain such lower bound comes from the observation that the diameter of the minimum enclosing ball of any set of points, say $\mathcal{Q}$, is larger than or equal to $\text{diam}(\mathcal{Q})$, the diameter of set $\mathcal{Q}$ (the maximum Euclidean distance between any two points of $\mathcal{Q}$). Suppose the solution of the MB_k problem corresponds to a ball that encloses subset $\mathcal{Q} \subseteq \mathcal{P}$, such that $|\mathcal{Q}| \geq k$, and with radius r^*. We then have

$$\begin{aligned} 2r^* &\geq \text{diam}(\mathcal{Q}) \\ &= \left(\tfrac{|\mathcal{Q}|(|\mathcal{Q}|-1)}{2}\right)\text{-th smallest pairwise distance of } \mathcal{Q} \\ &\geq \left(\tfrac{|\mathcal{Q}|(|\mathcal{Q}|-1)}{2}\right)\text{-th smallest pairwise distance of } \mathcal{P} \\ &\geq \left(\tfrac{k(k-1)}{2}\right)\text{-th smallest pairwise distance of } \mathcal{P}. \end{aligned} \tag{3.5}$$

A lower bound for the solution of the MB_k problem can also be obtained based on the relaxations of mixed-integer formulations of (3.1). It is possible to formulate (3.1) as a mixed-integer-quadratic problem as follows:

$$\begin{aligned} \min_{r,x,\beta_j} \quad & r^2 \\ \text{s.t.} \quad & \|p_j - x\|^2 \leq r^2 + (1-\beta_j)M, \quad p_j \in \mathcal{P} \\ & \sum_{j=1}^{m} \beta_j \geq k \\ & \beta_j \in \{0,1\}, \quad j = 1, ..., m, \end{aligned} \tag{3.6a}$$

where M is a constant that is "large enough". Relaxing the integrality constraints (the binary constraints are replaced by $0 \leq \beta_j \leq 1$) yields a lower bound for r^*. However, the value of the constant M needs to be very large to not jeopardize feasibility. Without any knowledge about the location of the minimum k-enclosing ball of $\mathcal{P}$, all we can conclude is that $M \leq (\text{diam}(\mathcal{P}))^2$, which could be quite large. As a result, the relaxation (3.6a) tends to not be very useful, except when k is very close to $|\mathcal{P}|$.

Problem (3.1) can also be formulated as a mixed-integer-second-order-cone program:

$$\begin{aligned} \min_{r,x,\beta_j} \quad & r \\ \text{s.t.} \quad & \|\beta_j p_j + (1-\beta_j)s - x\| \leq r, \quad p_j \in \mathcal{P} \\ & \sum_{j=1}^{m} \beta_j \geq k \\ & \beta_j \in \{0,1\}, \quad j = 1, ..., m, \end{aligned} \tag{3.6b}$$

with s being any point known to be enclosed by the optimal k-enclosing ball of $\mathcal{P}$. A lower bound for the solution of (3.1) can be found using a relaxation of (3.6b) as follows . First, select a subset $\mathcal{Q}$ of $\mathcal{P}$ with $m-k+1$ points; then, solve the relaxation $m-k+1$ times, each time considering s a different point from $\mathcal{Q}$. Since, at least one of the points from $\mathcal{Q}$ is known to be covered by the optimal k-enclosing ball, we can obtain a lower bound for r^* by selecting the minimum radius among all the obtained solutions of the relaxations.

Lower bounds on the best solution on a subtree

The formulations (3.6) can be adapted to calculate the best solution that can be found on a node's subtree.

Consider node N of the search tree. Recall that $l(N) = |Pt(N)|)$ and $|ST(N)| = m - i(N)$. The smallest radius of a k-enclosing ball that can be found in any of N's subtree nodes is given by the problem of finding the minimum radius k-enclosing ball that covers $Pt(N)$ and at least $k - l(N)$ points of $ST(N)$. That problem can be formulated based on

(3.6a) as

$$\begin{aligned}
\min_{r,x,\beta_j} \quad & r^2 \\
\text{s.t.} \quad & \|p_i - x\|^2 \leq r^2, \quad p_i \in Pt(N) \\
& \|q_j - x\|^2 \leq r^2 + (1-\beta_j)M, \quad q_j \in ST(N) \\
& \sum_{j=1}^{|ST(N)|} \beta_j \geq k - l(N) \\
& \beta_j \in \{0,1\}, \quad j = 1, ..., m - i(N).
\end{aligned} \tag{3.7a}$$

Similarly, based on (3.6b), we have

$$\begin{aligned}
\min_{r,x,\beta_j} \quad & r \\
\text{s.t.} \quad & \|p_i - x\| \leq r, \quad p_i \in Pt(N) \\
& \|\beta_j q_j + (1-\beta_j)s_j - x\| \leq r, \quad q_j \in ST(N) \\
& \sum_{j=1}^{|ST(N)|} \beta_j \geq k - l(N) \\
& \beta_j \in \{0,1\}, \quad j = 1, ..., m - i(N),
\end{aligned} \tag{3.7b}$$

where s_j are points that are guaranteed to be covered by the smallest k-enclosing ball found on N's subtree. Point s_j can be the same for all the constraints, for instance, any point of $Pt(N)$ would do. However, picking s_j such that the distance between s_j and q_j is small gives better bounds. A simple option is, for instance, choosing s_j to be the closest point of $Pt(N)$ to q_j.

Relaxing the integrality constraints in (3.7) yields formulations that can be used to calculate, at each node, a lower bound on the best solution on that node's subtree. These can be applied to some or all nodes of the search tree in order to prune parts of the tree more aggressively. Applying these lower bounds at every node is not practical as solving each formulation many times is very time-consuming, especially (3.7b). For nodes whose subtrees contain a small number of points, that is $m - i(N)$ is small, it may even be faster to explore all nodes on the subtree instead of solving any of the problems above.

As observed before, due to the large magnitude of the M constant in (3.7a), we expect that the SOCP relaxation (3.7b) yields better lower bounds than the QP relaxation (3.7a).

Moreover, both relaxations are not expected to provide useful lower bounds when $k - l(N)$ is small (unless $m - i(N)$ is small, in which case, as discussed, we might as well scan all nodes on the subtree). When $k - l(N)$ is small, the constraint $\sum \beta_j \geq k - l(N)$ is likely to allow very small values for β_j causing the solution of either relaxation to be only slightly larger than $r(N)$.

Remark 3.4. A lower bound on the best solution of a node's subtree can be obtained based on (3.5). However, this lower bound is not useful, given the point-node assignment method we adopt. The points of $Pt(N)$ are among the ones with higher pairwise distances of the set $Pt(N) \cup ST(N)$, given how they were selected. Therefore, in most cases, half the $(k(k-1)/2)$-smallest distance between the points in $Pt(N) \cup ST(N)$ will be smaller than $r(N)$ (the radius of the minimum enclosing ball of $Pt(N)$).

We carried out some experiments to understand how much the relaxations (3.7) can reduce the number of explored nodes. Both relaxations were applied separately for each dataset. We did not compute these lower bounds on every single node, as that would have been unnecessarily time consuming, but only when all the following conditions were met: the number of points involved in the relaxation was larger than k (otherwise, the relaxation is simply an MB problem, also, this excludes leaves); and the number of points on the subtree of a node was at least 0.1 of the total number of nodes. Note that these rules still imply that the relaxations are applied in a vast number of the nodes, which is still impractical. We tested this strategy just for the sake of studying the maximum impact we can expect from using such relaxations in the B&B.

In our tests, we gave the algorithm initial solutions. As we did before, we used the Spherical Ordering method for the datasets *normal* and *uniform*; and for the *exponential* dataset we used Spherical Peeling.

The results are presented in Tables 3.8 and 3.9, where the percent ratios of the number of explored nodes when each relaxation was used under the conditions described above, over the number of explored nodes when no such relaxation was used, are presented. These experiments required a significant amount of time to run, and it was not possible to run such

		nodes ratios (%)	
	k	SOCP	QP
normal	50	96.1%	99.9%
	100	85.3%	99.6%
	250	84.2%	99.4%
	750	89.1%	99.6%
	900	85.4%	99.2%
	990	91.3%	96.4%
uniform	50	88.3%	99.6%
	100	65.3%	99.4%
	990	98.6%	99.7%
exponential	50	79.8%	98.8%
	100	60.8%	97.2%
	250	48.6%	98.0%
	750	74.0%	98.6%
	900	86.0%	98.7%
	990	93.5%	95.3%

Table 3.8: The ratio in percentage of the number of explored nodes when the relaxations (3.7) are used over the number of explored nodes when no such relaxation is used, for different 2-dimensional datasets with 1000 points.

		nodes ratios (%)	
	k	SOCP	QP
normal	10	98.6%	99.4%
	30	91.3%	99.3%
	55	89.5%	99.2%
	75	90.5%	99.3%
	90	93.1%	98.9%
unif.	10	99.0%	99.4%
	90	98.6%	100.0%
exponential	10	94.8%	97.0%
	30	87.3%	98.3%
	55	85.3%	98.1%
	75	86.1%	97.7%
	90	88.8%	95.6%

Table 3.9: The ratio in percentage of the number of explored nodes when the relaxations (3.7) are used over the number of explored nodes when no such relaxation is used, for different 10-dimensional datasets with 100 points.

experiments for the *uniform* datasets and $k = 100, 250, 750$, due to the excessive running times.

In Tables 3.8 and 3.9, we observe that the QP relaxation had almost no impact in decreasing the number of explored nodes. This occurs because the value of M is usually very large, which causes the lower bound given by the relaxation to be little better than the one we already have, that is, the radius of $MB(Pt(N))$. However, a small reduction was observed in some cases when k is very large. In this case, since the value of most of the β_j will be close to 1, the effect of M is neutralized for most of the constraints. The SOCP relaxation performs better than the QP one. However, its application is much more time-consuming, and our tests show that, when k is either small or very large, the SOCP relaxation (3.7b) is often not very helpful. As observed previously, when k is small, that fact is a consequence of the formulation. When k is large, since the optimal radius is very large, these relaxations would only be able to decrease the number of explored nodes considerably if they could produce a large value as a lower bound, which is not the case. So, it makes sense that our results show that these relaxations have an impact mostly for middle values of k.

As a final comment, the (usually low) impact of using these relaxations cannot be only justified by formulation features. We must also consider the fact that, because of the node-point assignment scheme, at each node N, the value of $r(N)$ is already a very good lower bound, undermining the effect of these relaxations. These relaxations, when applied to a B&B based on the lexicographic point-node assignment, had a much more significant impact.

3.3.3 On the point-node assignment scheme

In Section 3.1.3 we introduced a point-node assignment scheme that, every time a node N is branched, assigns points to N's children in descending order of distance to $x(N)$. To observe empirically the effect of adopting this scheme, in addition to implementing a B&B that employs it (which will be denoted by BB_{dist}), we have also implemented a B&B that uses the lexicographical order to assign points to nodes (BB_{lexic}), and a B&B that orders

the children nodes according to their radius in decreasing order as we explain next (referred to as BB_{rad}).

A B&B that orders the children nodes according to their radius would work as follows: pick the points to be assigned to the children nodes, say $q_1, ..., q_{b(N)}$, to be the ones with largest distance from $x(N)$ (like in the scheme we propose); get r_j by solving the subproblem $MB(ST(N) \cup q_j)$ corresponding to each point q_j; and assign the points to nodes in descending order by radius r_j instead of by distance. When we do this, on one hand, the left-most children will have a lower bound that is at least as good as the one resulting from the distance-based order. This, may result in a reduction in the number of explored nodes. On the other hand, note that all child nodes need to be explored in order to be sorted, including the one that would be the last child. Only after knowing which node is the last child, its path can be compressed into a single node, and solved. This may in fact increase the total number of dual iterations.

For each dataset, we run the three B&B algorithms with the same initial solution. The results of our experiments are reported in Tables 3.10 and 3.11, and contain the following measures:

- **total tree nodes**: the total number of nodes of the search tree;
- **nodes**: the average number of explored nodes of the corresponding B&B;
- **iters**: the average total number of dual iterations from Dearing and Zeck's algorithm from the corresponding B&B.

Table 3.10 shows how the lexicographic scheme is inefficient even for relatively easy problems such as the case of 50-point datasets. Increasing the number of points from 50 to 100 makes the lexicographical order an inviable option, due to the very large number of explored nodes and consequent impractical running times. In Table 3.10, we barely observe a difference in using the distance versus the radius. That difference is better shown when we increase the number of points, see Table 3.11. In general, we see that, as expected, when the nodes are ordered based on the radius value, the number of explored nodes is smaller in general, though not significantly. However, this reduction is not enough to make up for the extra dual iterations resulting from exploring the last child of each node of the minimum

		total	BB_{dist}		BB_{rad}		BB_{lexic}	
	k	tree nodes	nodes	iters	nodes	iters	nodes	iters
normal	5	2.35E+06	65	65	65	65	149	149
	10	1.28E+10	91	91	91	91	1.72E+03	1.87E+03
	15	3.19E+12	108	108	107	107	1.83E+04	2.07E+04
	30	1.14E+14	98	98	97	97	1.12E+06	1.31E+06
	45	1.80E+07	23	23	22	23	2.33E+04	2.67E+04

Table 3.10: Comparison of explored nodes (nodes) and dual iterations (iters) from different point-node assignment schemes for 2-dimensional normal datasets with 50 points. The results report the averages for 10 instances.

		total	BB_{dist}		BB_{rad}		ratios (%)	
	k	tree nodes	nodes(n_1)	iters (i_1)	nodes(n_2)	iters (i_2)	n_2/n_1	i_2/i_1
normal	10	1.92E+13	3.43E+03	3.65E+03	3.41E+03	3.75E+03	99%	103%
	30	4.19E+25	4.28E+04	4.75E+04	4.15E+04	4.98E+04	97%	105%
	55	1.35E+29	4.32E+04	4.90E+04	4.11E+04	5.43E+04	95%	111%
	75	9.42E+23	1.35E+04	1.62E+04	1.29E+04	1.71E+04	96%	106%
	90	1.59E+14	1.32E+03	1.79E+03	1.26E+03	1.89E+03	96%	106%
uniform	10	1.92E+13	2.00E+04	2.10E+04	1.98E+04	2.23E+04	99%	106%
	30	4.19E+25	6.62E+05	7.14E+05	6.21E+05	7.40E+05	94%	104%
	55	1.35E+29	2.78E+06	3.09E+06	2.42E+06	3.05E+06	87%	99%
	75	9.42E+23	9.73E+05	1.16E+06	8.42E+05	1.15E+06	87%	99%
	90	1.59E+14	3.27E+04	4.63E+04	2.87E+04	4.58E+04	88%	99%

Table 3.11: Comparison of explored nodes (nodes) and dual iterations (iters) from different point-node assignment schemes for 10-dimensional normal and uniform datasets with 100 points. The results report the averages for 10 instances.

solution tree. We observe that the total number of dual iterations is usually larger for the B&B using the radius, yielding higher computational times when compared to the B&B that uses the distance.

3.4 Conclusion

In this chapter we proposed a branch-and-bound algorithm to solve the NP-hard problem of finding the ball of minimum radius that encloses at least k points from a given set. Our algorithm makes a correspondence between the subsets of points and the tree nodes. It uses a heuristic that aims at placing subsets of points enclosed in larger balls at the root of large subtrees. As a result, it has the ability to prune very large subsets of points from the search tree early on. This heuristic, combined with a last-in-first-out strategy for selecting the next node to explore, results in a combination of depth-first-search and best-first-search techniques. The algorithm retains the advantages of both these methods without the typical disadvantages. In fact, an important consequence of our LIFO approach is that the memory required to hold the set of live nodes is bounded by $m - k$.

Our experimental results show that, in general, only a small fraction of the nodes of the search tree need to be explored. They also support our choice of the DZ dual algorithm to solve the subproblems at each node, since in general we need a very small number of iterations per node.

In general, the algorithm does quite well on datasets of small to medium size, independently of k. If we consider situations where the goal is to discard outliers, then we will be interested in large values of k. We observe that, when k is large, the algorithm performs very well in general. We even managed to solve $10,000$-point instances with artificial outliers in dimensions up to 100 in a practical amount of time. The most difficult datasets for the algorithm seem to be those with a uniform distribution of the points drawn from a ball. However, a dataset with such characteristics would be unlikely to arise from an application of the MB_k problem.

Finally, we observed that the use of an initial good solution is not very consequential

in reducing the number of explored nodes. We also studied two relaxations of the MB_k problem in order to find better lower bounds at each node. One was based on a SOCP while another was based on a QP. Our study revealed that the lower bounds given by these relaxations are not very effective given our node/point attribution rule.

Chapter 4

The minimum enclosing ball of balls

In this chapter, we investigate the problem of the *minimum enclosing ball of balls*. We consider its second-order cone formulation, and conclude that other problems can also be solved using the same formulation. We will denote this general formulation as the problem of finding the *infimum with respect to* $\mathcal{Q}$. We start by defining this problem.

Let $\mathcal{Q}$ denote the second-order cone $\{x := (x_0; \overline{x}) \in \mathbb{R}^n : \|\overline{x}\|_2 \leq x_0\}$. Consider the generalized inequalities[1]

$$x \preceq_{\mathcal{Q}} y \quad \text{iff} \quad y - x \in \mathcal{Q} \quad \text{and} \quad x \prec_{\mathcal{Q}} y \quad \text{iff} \quad y - x \in \operatorname{int} \mathcal{Q}.$$

Given a set of distinct points $\mathcal{P} = \{p_1, ..., p_m\} \subset \mathbb{R}^n$, consider the following problem

$$\begin{aligned} \operatorname{Inf}_{\mathcal{Q}}(\mathcal{P}) := \max_{x \in \mathbb{R}^n} \quad & \langle e_0, x \rangle \\ \text{s.t.} \quad & x \preceq_{\mathcal{Q}} p_i,\ i = 1, ..., m \end{aligned} \tag{4.1}$$

where $e_0 = (1, 0, ..., 0) \in \mathbb{R}^n$. We call this problem the *infimum of* $\mathcal{P}$ *with respect to* $\mathcal{Q}$ or the $\operatorname{Inf}_{\mathcal{Q}}$ *problem* in short, due to its resemblance with the linear programming formulation

[1] The relation $\preceq_{\mathcal{Q}}$ defines a partial order on $\mathbb{R}^n$, that is, the following holds for any elements $u, v, w \in \mathbb{R}^n$: if $u \preceq_{\mathcal{Q}} v$ and $v \preceq_{\mathcal{Q}} u$ then $u = v$ *(antisymmetry)*; if $u \preceq_{\mathcal{Q}} v$ and $v \preceq_{\mathcal{Q}} w$ then $u \preceq_{\mathcal{Q}} w$ *(transitivity)*; $u \preceq_{\mathcal{Q}} u$ *(reflexivity)*.

of finding the minimum of a given set of numbers.

In this chapter, we introduce a dual simplex-like algorithm to solve the $\text{Inf}_{\mathcal{Q}}$ problem. Our algorithm shares similarities with the dual active-set algorithm for strictly convex quadratic programs by Goldfarb and Idnani [51], and with the dual algorithm for the minimum enclosing ball of points by Dearing and Zeck [33] (studied in Chapter 2).

This chapter is organized as follows. In Section 4.1, we show that the *minimum enclosing ball of balls problem*, as well as other related problems, are $\text{Inf}_{\mathcal{Q}}$ instances. Section 4.2 presents important theoretical background about the $\text{Inf}_{\mathcal{Q}}$ problem that provide a foundation to the algorithm. The dual simplex-like algorithm for the $\text{Inf}_{\mathcal{Q}}$ problem is introduced in detail in Section 4.3, and Section 4.4 explains how the algorithm can be efficiently implemented using the Cholesky factorization. Finally, computational results are presented in Section 4.5, and conclusions are drawn in Section 4.6.

4.1 Equivalent problems in computational geometry

Let x^* be the optimal solution to (4.1). First note that, a straightforward geometric interpretation for problem (4.1) is as follows: x^* is the "highest" point (considering x_0 as the height of point x) such that $x^* + \mathcal{Q}$ covers all points of set $\mathcal{P}$ (see Figure 4.1).

Denote by $B(c, r)$ the Euclidean ball with center at $c \in \mathbb{R}^{n-1}$ and radius $r \geq 0$.

Proposition 4.1. $B(c_i, r_i) \subseteq B(c, r)$ *if and only if* $\|c - c_i\| \leq r - r_i$.

Proof. Suppose $B(c_i, r_i) \subseteq B(c, r)$. Consider

$$w = c - \gamma(c - c_i) \quad \text{with} \quad \gamma = \frac{\|c - c_i\| + r_i}{\|c - c_i\|}.$$

We have that $\|w - c_i\| = |1 - \gamma| \, \|c - c_i\| = r_i$. Therefore, $w \in B(c, r)$ which implies

$$\|w - c\| = \|-\gamma(c - c_i)\| \leq r \quad \Leftrightarrow \quad \|c - c_i\| + r_i \leq r.$$

The other implication is a direct consequence of the triangle inequality. □

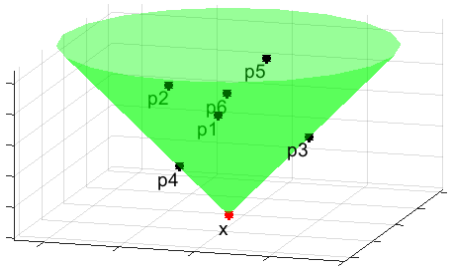


Figure 4.1: A geometric interpretation of the $\text{Inf}_{\mathcal{Q}}$ problem: its solution is the point x^* with maximum x_0 such that $p_i \in x^* + \mathcal{Q}$, for all given points p_i.

As a consequence of Proposition 4.1, the *smallest enclosing ball of balls*, defined as the problem of enclosing a given set of balls $B(c_i, r_i)$, $i = 1, ..., m$, with a ball $B(c, r)$ of smallest radius, can be reduced to an $\text{Inf}_{\mathcal{Q}}$ instance by considering $\mathcal{P} = \{(-r_i; c_i),\ i = 1, ..., m\}$:

$$\begin{array}{ll} \max\limits_{x \in \mathbb{R}^n} & x_0 \\ \text{s.t.} & \binom{x_0}{\bar{x}} \preceq_{\mathcal{Q}} \binom{-r_i}{c_i},\ i = 1, ..., m. \end{array} \tag{4.2}$$

The solution x^* to (4.2) is such that $x_0^* \leq 0$. The optimal ball, $B(c, r)$, has center $c = \bar{x}^*$ and radius $r = -x_0^*$. Figure 4.2 illustrates this situation: the minimum radius enclosing ball of a set of balls is the intersection of the cone $x^* + \mathcal{Q}$ with the plane $x_0 = 0$. When $r_i = 0$, for all $i = 1, ..., m$, problem (4.2) corresponds to the smallest enclosing ball of points problem studied in Chapter 2.

Consider now the problem of finding the ball $B(c, r)$ with smallest radius that intersects all balls $B(c_i, r_i)$, $i = 1, ..., m$, that is, the *smallest intersecting ball problem*. This problem also reduces to an $\text{Inf}_{\mathcal{Q}}$ instance as a consequence of the following proposition:

Proposition 4.2. $B(c_1, r_1) \cap B(c_2, r_2) \neq \emptyset$ *if and only if* $\|c_2 - c_1\| \leq r_2 + r_1$.

Proof. Suppose $\|c_2 - c_1\| \leq r_2 + r_1$. Consider the following point

$$x = \frac{r_2 c_1 + r_1 c_2}{r_1 + r_2}.$$

We have that $x \in B(c_1, r_1)$ since

$$\|x - c_1\| = \left\| \frac{r_1 c_2 - r_1 c_1}{r_1 + r_2} \right\| = \frac{r_1}{r_1 + r_2} \|c_1 - c_2\| \leq r_1.$$

Similarly, one has $x \in B(c_2, r_2)$, so the intersection of the two balls is non-empty. The other implication is a direct consequence of the triangle inequality. □

Thus, to find $B(c, r)$ we simply consider $\mathcal{P} = \{(r_i; c_i),\ i = 1, ..., m\}$ in (4.1), obtaining:

$$\begin{array}{ll} \max\limits_{x \in \mathbb{R}^n} & x_0 \\ \text{s.t.} & \binom{x_0}{\bar{x}} \preceq_Q \binom{r_i}{c_i},\ i = 1, ..., m. \end{array} \tag{4.3}$$

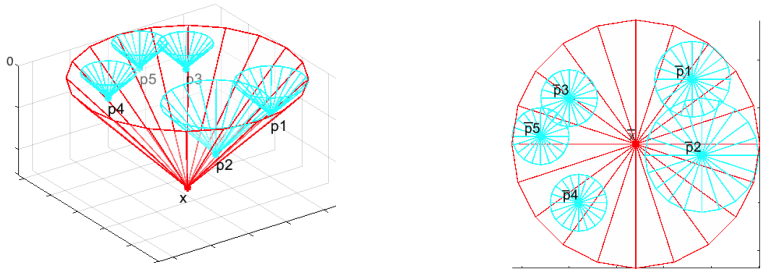
The smallest ball that intersects all given balls has center $c = \bar{x}^*$ and radius $r = -x_0^*$, where x^* is the optimal solution to (4.3). Figure 4.3 illustrates this case. For previous work on the *smallest intersecting ball* problem see [76, 80] and the references therein. An application of this problem can be found in [61].

When the intersection of balls $B(c_i, r_i)$, $i = 1, ..., m$, is non-empty, the smallest intersecting ball could be considered any point of the intersection. In this case, it turns out the solution to (4.3) has a different meaning: the solution is such that $x_0^* > 0$, and the ball $B(\bar{x}, x_0^*)$ is the largest radius ball that is enclosed in the intersection of all given balls. We shall refer to this problem as the *largest enclosed ball problem* (see Figure 4.4).

From the previous analysis, we conclude that the problem of finding a ball with smallest radius that simultaneously encloses balls $B(c_i, r_i)$, $i = 1, ..., m_1$, and intersects balls $B(c_j, r_j)$, $j = 1, ..., m_2$, can also be solved by considering $\mathcal{P} = \{(-r_i; c_i),\ i = 1, ..., m_1\} \cup \{(r_j; c_j),\ j = 1, ..., m_2\}$ in (4.1), resulting in

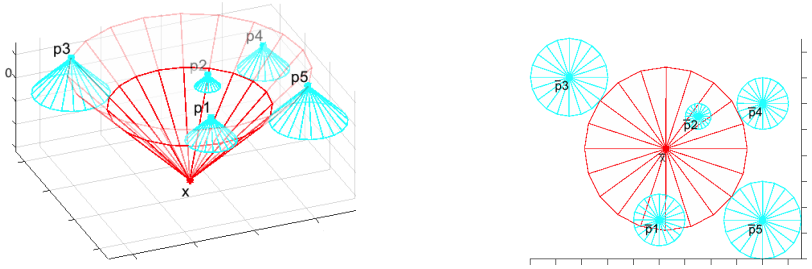
$$\begin{array}{ll} \max\limits_{x \in \mathbb{R}^n} & x_0 \\ \text{s.t.} & \binom{x_0}{\bar{x}} \preceq_Q \binom{-r_i}{c_i},\ i = 1, ..., m_1 \\ & \binom{x_0}{\bar{x}} \preceq_Q \binom{r_j}{c_j},\ j = 1, ..., m_2. \end{array}$$

The optimal ball has center $\bar{x}^*$ and radius $-x_0^*$. The problem is illustrated in Figure 4.5. Previous work on this problem includes [79].



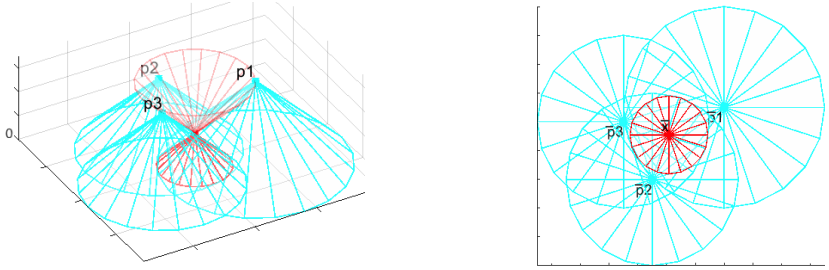
(a) *Cone view* (b) *Cross-section* $x_0 = 0$

Figure 4.2: The smallest enclosing ball of a set of balls.



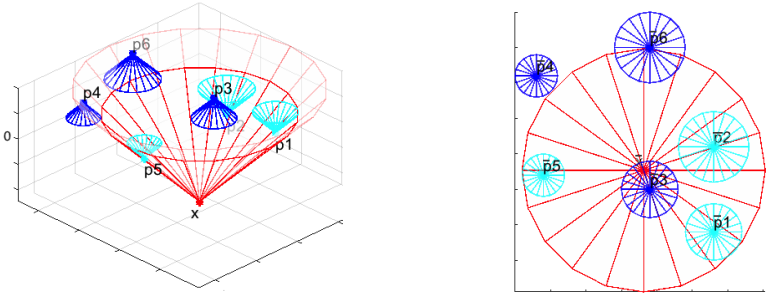
(a) *Cone view* (b) *Cross-section* $x_0 = 0$

Figure 4.3: The smallest intersecting ball of a set of balls.



(a) *Cone view* (b) *Cross-section* $x_0 = 0$

Figure 4.4: The largest enclosed ball in a set of balls.



(a) *Cone view* (b) *Cross-section* $x_0 = 0$

Figure 4.5: The smallest radius ball that simultaneously intersects a set of balls and encloses another set of balls.

4.2 Properties of the Inf_{Q} problem

Recall that $\mathcal{P} = \{p_1, ..., p_m\}$, and let $p_i = (p_{i0}; \overline{p}_i) \in \mathbb{R}^n$ with $\overline{p}_i \in \mathbb{R}^{n-1}$. We now present two theorems about the solution of the problem $\mathrm{Inf}_{Q}(\mathcal{P})$.

Theorem 4.3. *The solution to* $\mathrm{Inf}_{Q}(\mathcal{P})$ *exists and is unique.*

It is easy to see that the solution to $\mathrm{Inf}_{Q}(\mathcal{P})$ exists since the feasible set is always non-empty. For a proof of the uniqueness of the solution, we refer the reader to [40, §3.1].

Theorem 4.4. *The solution to* $\mathrm{Inf}_{Q}(\mathcal{P})$ *is* $p_k \in \mathcal{P}$*, for some* $k = 1, ..., m$*, if and only if* $p_k \preceq_{Q} p_i$*, for all* $i = 1, ..., m$*.*

Proof. First, note that, the optimal solution of $\mathrm{Inf}_{Q}(\mathcal{P})$, x^*, always satisfies $x_0^* \leq p_{i0}$, $i = 1, ..., m$. Assume there exists $p_k \in \mathcal{P}$, for some $k = 1, ..., m$, that is feasible for $\mathrm{Inf}_{Q}(\mathcal{P})$. Then, $x_0^* \geq p_{k0}$ since the problem is a maximization. Thus, we conclude that $x_0^* = p_{k0}$, and so $x^* = p_k$. The opposite implication is trivial. □

4.2.1 Duality and optimality conditions

Before we proceed, let us introduce some notation.

- $\overline{\mathcal{P}} := \{\overline{p}_i : p_i \in \mathcal{P}\} \subset \mathbb{R}^{n-1}$;
- $x = (x_0; \overline{x})$ is the (primal) variable of (4.1), and its $x^* = (x_0^*; \overline{x}^*)$ optimal solution;
- $y_i = (y_{i0}; \overline{y}_i)$, $i = 1, ..., m$, are the dual variables, and $y_i^* = (y_{i0}^*; \overline{y}_i^*)$, $i = 1, ..., m$, an optimal dual solution.

The dual problem of $\mathrm{Inf}_{Q}(\mathcal{P})$ is

$$\begin{aligned} \min_{y} \quad & \sum_{i=1}^{m} \langle p_i, y_i \rangle \\ \text{s.t.} \quad & \sum_{i=1}^{m} y_i = e_0 \\ & y_i \succeq_{Q} 0, \quad i = 1, ..., m. \end{aligned} \tag{4.4}$$

Whereas the solution of the primal problem is unique, the solution of (4.4) may not be. Example 4.5 shows an example.

Example 4.5. Consider $\mathcal{P} \subset \mathbb{R}^3$ the set of points $p_1 = (0,0,0)$, $p_2 = (0,1,0)$, $p_3 = (0,0,1)$, and $p_4 = (0,1,1)$. The optimal solution to $\mathrm{Inf}_Q(\mathcal{P})$ is $x^* = \left(-\frac{\sqrt{2}}{2}, \frac{1}{2}, \frac{1}{2}\right)$. The following three sets of dual solutions are all optimal for the corresponding dual problem:

i. $y_1^* = \left(\frac{1}{4}, \frac{\sqrt{2}}{8}, \frac{\sqrt{2}}{8}\right)$, $y_2^* = \left(\frac{1}{4}, -\frac{\sqrt{2}}{8}, \frac{\sqrt{2}}{8}\right)$, $y_3^* = \left(\frac{1}{4}, \frac{\sqrt{2}}{8}, -\frac{\sqrt{2}}{8}\right)$, $y_4^* = \left(\frac{1}{4}, -\frac{\sqrt{2}}{8}, -\frac{\sqrt{2}}{8}\right)$;

ii. $y_1^* = \left(\frac{1}{2}, \frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{4}\right)$, $y_2^* = y_3^* = (0,0,0)$, $y_4^* = \left(\frac{1}{2}, -\frac{\sqrt{2}}{4}, -\frac{\sqrt{2}}{4}\right)$;

iii. $y_1^* = y_4^* = (0,0,0)$, $y_2^* = \left(\frac{1}{2}, \frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{4}\right)$, $y_3^* = \left(\frac{1}{2}, -\frac{\sqrt{2}}{4}, -\frac{\sqrt{2}}{4}\right)$.

Lemma 4.6 (Strict feasibility)**.** *Both primal (4.1) and dual (4.4) problems are strictly feasible; i.e. there exists a primal feasible vector x such that $x \prec_Q p_i$, for all $i = 1, ..., m$, and there exist dual feasible $y_1, ..., y_m$ such that $y_i \succ_Q 0$, for all $i = 1, ..., m$.*

Proof. It is possible to construct a strictly primal feasible solution as follows. Consider $\overline{x} = \overline{p}_j$ for some $j = 1, ..., m$, and $x_0 = \min_{i=1,...,m}\{p_{i0} - \left\|\overline{p}_i - \overline{p}_j\right\|\} - \epsilon$, for some $\epsilon > 0$. It is easy to see that

$$\|\overline{p}_i - \overline{x}\| = \left\|\overline{p}_i - \overline{p}_j\right\| \leq p_{i0} - x_0 - \epsilon < p_{i0} - x_0, \quad i = 1, ..., m.$$

Note that $p_{i0} - x_0 \geq \|\overline{p}_i - \overline{p}_k\| + \epsilon > 0$. Finally, a trivial strictly dual feasible solution is $y_i = (\frac{1}{m}, 0, ..., 0)$, for all $i = 1, ..., m$. □

Since both primal and dual problems are strictly feasible, the duality gap is zero, that is, *strong duality* holds and the Karush-Kuhn-Tucker conditions are also sufficient, e.g. [8, pp. 25]. This is stated in Theorems 4.7 and 4.8.

Theorem 4.7 (Strong duality)**.** *Both primal (4.1) and dual (4.4) problems have optimal solutions, x^* and $y_1^*, ..., y_m^*$, respectively, and*

$$\langle e_0, x^* \rangle = \sum_{i=1}^{m} \langle p_i, y_i^* \rangle.$$

Theorem 4.8 (Optimality conditions)**.** *Let* x^* *and* y_i^*, $i = 1, ..., m$ *be any vectors in* $\mathbb{R}^n$. *The pair* $(x^*, \{y_i^*\}_{i=1,...,m})$ *is primal-dual optimal if and only if*

- *primal feasibility:* $x^* \preceq_Q p_i$, $i = 1, ..., m$;
- *dual feasibility:* $\sum_{i=1}^m y_i^* = e_0$ *and* $y_i^* \succeq_Q 0$, $i = 1, ..., m$;
- *complementary slackness:* $\langle p_i - x^*, y_i^* \rangle = 0$, $i = 1, ..., m$.

The next corollary follows from the complementary slackness conditions.

Corollary 4.9. *Let* x^* *and* $\{y_i^*\}_{i=1,...,m})$ *be primal and dual optimal, respectively. Then,*

- *if* $y_i^* \succ_Q 0$, *then* $x^* = p_i$ *(which can happen for a single* i, *and, in that case,* $y_i^* = e_0$ *and* $y_j^* = 0$, *for all* $j \neq i$*);*
- *if* $x^* \succ_Q p_i$, *then* $y_i^* = 0$;
- *if* $p_i - x^* \in \partial Q$ *and* $y_i^* \in \partial Q$, *then*

$$\overline{y}_i^* = -\frac{y_{i0}^*}{p_{i0} - x_0^*} (\overline{p}_i - \overline{x}^*) = -y_{i0}^* \frac{\overline{p}_i - \overline{x}^*}{\|\overline{p}_i - \overline{x}^*\|}. \tag{4.5}$$

As illustrated in Example 4.5, we may have $p_i - x^* \in \partial Q$ and $y_i^* = 0$ for some i. Thus, *strict complementarity*, that is, $p_i - x^* + y_i^* \succ_Q 0$ for all $i = 1, ..., m$, may not hold.

The following characterization of optimality is a consequence of Theorem 4.8:

Theorem 4.10. *The optimal solution to* $\mathrm{Inf}_Q(\mathcal{P})$ *is* x^*, *iff* x^* *is primal feasible, and*

$$\overline{x}^* \in \mathrm{conv}\left(\{\overline{p}_i \in \overline{\mathcal{P}} : \|\overline{p}_i - \overline{x}^*\| = p_{i0} - x_0^*\}\right). \tag{4.6}$$

Proof. If $x^* = p_k$, for some $p_k \in \mathcal{P}$, the theorem follows trivially from the optimality conditions. For the remainder of the proof, assume that $x^* \neq p_k$, $p_k \in \mathcal{P}$.

First, consider x^* is optimal for $\mathrm{Inf}_Q(\mathcal{P})$. Let $\mathcal{I} = \{i : \|\overline{p}_i - \overline{x}^*\| = p_{i0} - x_0^*\}$, and y_i^*, $i = 1, ..., m$, be an optimal dual solution. From complementary slackness and dual

feasibility, $y_i^* = 0$, for $i \notin \mathcal{I}$, and

$$0 = \sum_{i \in \mathcal{I}} \overline{y}_i^* = \sum_{i \in \mathcal{I}} \frac{y_{i0}^*}{p_{i0} - x_0^*} \left(\overline{p}_i - \overline{x}^*\right).$$

As a consequence, we get

$$\overline{x}^* = \sum_{i \in \mathcal{I}} \alpha_i \overline{p}_i, \quad \text{with} \quad \alpha_i = \frac{y_{i0}^* / \left(p_{i0} - x_0^*\right)}{\sum_{i \in \mathcal{I}} y_{i0}^* / \left(p_{i0} - x_0^*\right)}.$$

Since $\sum_i y_{i0}^* = 1$, we have that $\sum_{i \in \mathcal{I}} \alpha_i = 1$, and there must be at least one $\alpha_i \neq 0$. Finally, $y_i^* \succeq_Q 0$ implies $\alpha_i \geq 0$, for all i. Thus, $\sum_{i=1}^m \alpha_i = 1$, so (4.6) holds.

Conversely, suppose x^* is primal feasible and satisfies (4.6):

$$\overline{x}^* = \sum_{i \in \mathcal{I}} \alpha_i \overline{p}_i, \text{ with } \alpha_i \geq 0, \text{ for } i \in \mathcal{I}, \quad \text{and} \quad \sum_{i \in \mathcal{I}} \alpha_i = 1.$$

We will now build a dual solution that is feasible and, together with x^*, satisfies complementary slackness. Consider y_{i0}, $i = 1, ..., m$, such that $z_i = y_{i0}/(p_{i0} - x_0^*)$ and

$$\alpha_i = \frac{z_i}{\sum_{j=1}^m z_j}, \tag{4.7}$$

Let $\sigma_j = p_{i0} - x_0^*$. Equations (4.7) together with $\sum_i y_{i0} = 1$ result in the following systems of equations

$$\begin{aligned} &\alpha_i(z_1 + \ldots + z_m) = z_i, \quad i = 1, ..., m, \\ &\sigma_1 z_1 + \ldots + \sigma_m z_m = 1. \end{aligned}$$

In order to solve the system of equations for y_{i0}, first note that

$$\sigma_1 \alpha_1 (z_1 + \ldots + z_m) + \ldots + \sigma_m \alpha_m (z_1 + \ldots + z_m) = 1,$$

which yields

$$(z_1 + \ldots + z_m) = \frac{1}{\sigma_1 \alpha_1 + \ldots + \sigma_m \alpha_m}.$$

Therefore

$$z_i = \frac{\alpha_i}{\sigma_1\alpha_1 + \ldots + \sigma_m\alpha_m} \quad \Rightarrow \quad y_{i0} = \frac{\alpha_i(p_{i0} - x_0^*)}{\sum_{j=1}^{m} \alpha_j(p_{j0} - x_0^*)}.$$

Note that $y_{i0} \geq 0$, for $i \in \mathcal{I}$, and $y_{i0} = 0$, for $i \notin \mathcal{I}$. Finally, if we consider

$$\overline{y}_i = -\frac{y_{i0}}{p_{i0} - x_0^*}\left(\overline{p}_i - \overline{x}\right), \quad i = 1, ..., m,$$

we have that $(y_{i0}; \overline{y}_i)$ is feasible for the dual problem (4.4). It is easy to see that x^* and y_i, $i = 1, ..., m$, satisfy complementary slackness. Thus, they are optimal for the respective problems. □

Observation 4.11. From the proof of Theorem 4.10 we can conclude that the optimal solution of $\mathrm{Inf}_{\mathcal{Q}}(\mathcal{P})$ is such that $\overline{x}^* = \mathrm{ri}\,\mathrm{conv}(\{\overline{p}_i : y_{i0}^* > 0\})$. Note that $y_{i0}^* > 0$ could not be substituted by $y_i^* \in \partial\mathcal{Q}$, because when $x^* = p_k \in \mathcal{P}$ we have $y_k^* = e_0 \notin \partial\mathcal{Q}$.

A conclusion from Theorem 4.10, which will be important for of the algorithm presented in Section 4.3, is the corollary below.

Corollary 4.12. *Consider vector x, not necessarily primal feasible, that satisfies*

$$\overline{x} \in \mathrm{aff}(\{\overline{p}_i : i \in \mathcal{I}\}), \tag{4.8}$$

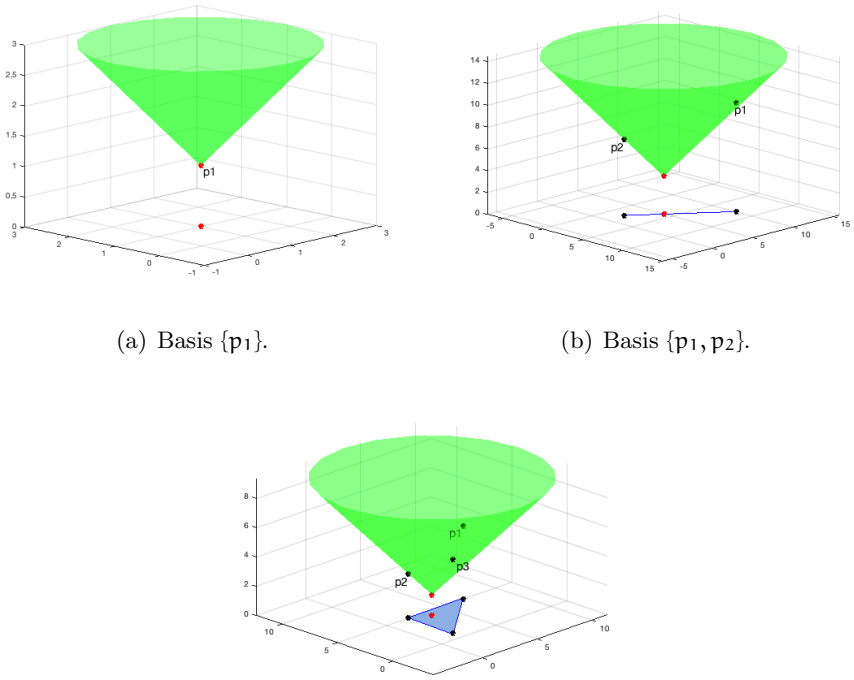
for $\mathcal{I} = \{i : \|\overline{p}_i - \overline{x}\| = p_{i0} - x_0\}$. Let $\alpha_1, ..., \alpha_m$ be the coefficients of the affine combination (4.8). Then there exists a dual solution given by

$$y_i = 0, \text{ for } i \notin \mathcal{I}, \quad \text{and} \quad y_{i0} = \frac{\alpha_i(p_{i0} - x_0)}{\sum_{j=1}^{m} \alpha_j(p_{j0} - x_0)}, \ \overline{y}_i = -\frac{y_{i0}}{p_{i0} - x_0}\left(\overline{p}_i - \overline{x}\right), \text{ for } i \in \mathcal{I},$$

that satisfies the dual constraint $\sum_{i=1}^{m} y_i = e_0$, and, together with x, the complementary slackness conditions.

4.2.2 Basis and dual feasible S-pair

We now define *basis* for the $\mathrm{Inf}_{\mathcal{Q}}(\mathcal{P})$ problem the same way a basis is defined for an LP-type problem (Section 1.1.1).



(a) Basis $\{p_1\}$.

(b) Basis $\{p_1, p_2\}$.

(c) Basis $\{p_1, p_2, p_3\}$.

Figure 4.6: Different bases in $\mathbb{R}^3$ with different cardinality.

Definition 4.13 (Basis)**.** A set $\mathcal{S} \subseteq \mathcal{P}$ is a *basis* if no proper subset $\mathcal{S}'$ of $\mathcal{S}$ is such that $\text{Inf}_Q(\mathcal{S}') = \text{Inf}_Q(\mathcal{S})$. A basis $\mathcal{S} \subseteq \mathcal{P}$ is *optimal* if $\text{Inf}_Q(\mathcal{S}) = \text{Inf}_Q(\mathcal{P})$.

Figure 4.6 illustrates the concept of basis in $\mathbb{R}^3$. An optimal basis may not be unique as shown in Example 4.5, in which we have two optimal bases: $\{p_1, p_4\}$ and $\{p_2, p_3\}$.

Theorem 4.14. *Let $\mathcal{S} \subset \mathcal{P}$ be a basis. If the constraint corresponding to $p^* \in \mathcal{P}$ is infeasible at the solution of $\text{Inf}_Q(\mathcal{S})$, then p^* belongs to an optimal basis of $\text{Inf}_Q(\mathcal{S} \cup \{p^*\})$.*

Proof. Let x solve $\text{Inf}_Q(\mathcal{S})$. Let $\mathcal{S}^*$ and x^* be an optimal basis and solution of $\text{Inf}_Q(\mathcal{S} \cup \{p^*\})$, respectively. Suppose, by contradiction, that $p^* \notin \mathcal{S}^*$. The fact that $\mathcal{S}^* \subseteq \mathcal{S} \subseteq \mathcal{S} \cup \{p^*\}$ implies $x_0^* \geq x_0 \geq x_0^*$. Since x^* is feasible for $\text{Inf}_Q(\mathcal{S})$ and $x_0^* = x_0$, then $x^* = x$. This contradicts the fact that the constraint corresponding to p^* is infeasible at x. □

Definition 4.15 (Dual feasible S-pair)**.** Let $\mathcal{S} \subseteq \mathcal{P}$ and $x \in \mathbb{R}^n$. We say $(\mathcal{S}, x)$ is a *dual feasible (solution) S-pair* if

(a) $\overline{\mathcal{S}}$ is affinely independent;

(b) $\|\overline{p}_i - \overline{x}\| = p_{i0} - x_0, \quad \forall\, p_i \in \mathcal{S}$;

(c) $\overline{x} \in \operatorname{ri}\operatorname{conv}(\overline{\mathcal{S}})$.

The next theorem presents an alternative characterization of a dual feasible S-pair.

Theorem 4.16. $(\mathcal{S}, x)$ *is a dual feasible S-pair iff* $\mathcal{S}$ *is a basis and* x *solves* $\operatorname{Inf}_Q(\mathcal{S})$.

Proof. The case $|\mathcal{S}^j| = 1$ is trivial. Consider a basis $\mathcal{S} = \{p_1, \ldots, p_s\}$, $s \geq 2$, and x the solution to $\operatorname{Inf}_Q(\mathcal{S})$. Let $\mathcal{S}' = \{p_i \in \mathcal{S} : \|\overline{p}_i - \overline{x}\| = p_{i0} - x_0\}$. From Theorem 4.10, we know that $\overline{x} \in \operatorname{conv}(\overline{\mathcal{S}}')$, thus x also solves $\operatorname{Inf}_Q(\mathcal{S}')$. Since $\mathcal{S}$ is minimal with respect to inclusion, we conclude that $\mathcal{S} = \mathcal{S}'$, proving property (b) of the dual feasible S-pair definition. Since no coefficient of the convex combination of $\overline{x}$ in terms of $\mathcal{S}$ can be 0 (otherwise, x would be optimal for a proper subset of $\mathcal{S}$), we conclude that $\overline{x} \in \operatorname{ri}\operatorname{conv}(\overline{\mathcal{S}})$, proving property (c). Let $\alpha_i > 0$ be the coefficients of $\overline{x}$ written as a convex combination of $\overline{\mathcal{S}}$. To prove property (a), assume, for the sake of contradiction, that $\overline{\mathcal{S}}$ is affinely dependent. Then, the set of vectors $\{\overline{p}_i - \overline{p}_1\}_{i \geq 2}$ is linearly dependent, that is, there exists coefficients β_i, not all equal to zero, such that $0 = \sum_{i=1}^{s} \beta_i \overline{p}_i$ and $\sum_{i=1}^{s} \beta_i = 0$. Thus,

$$\overline{x} = \sum_{i=1}^{s} \alpha_i \overline{p}_i + \delta \sum_{i=1}^{s} \beta_i \overline{p}_i, \quad \text{with} \quad \sum_{i=1}^{s} \alpha_i = 1,$$

for any $\delta \in \mathbb{R}$. Note that $\sum_{i=1}^{s} (\alpha_i + \delta\beta_i) = 1$. Consider $\delta = \min_i\{-\frac{\alpha_i}{\beta_i} : \beta_i < 0\}$. Then $\overline{x}$ becomes a convex combination of a proper subset $\mathcal{S}'$ of $\mathcal{S}$. By Theorem 4.10, x is optimal for $\operatorname{Inf}_Q(\mathcal{S}')$, contradicting the hypothesis that $\mathcal{S}$ is inclusion-minimal, proving property (a).

Let $(\mathcal{S}, x)$ be a dual feasible S-pair, and $\overline{x} = \sum_{i=1}^{s} \alpha_i \overline{p}_i$, with $\alpha_i > 0$, and $\sum_{1=1}^{s} \alpha_i = 1$. From Theorem 4.10, we know that properties (b) and (c) of the dual feasible S-pair definition imply that x is the solution to $\operatorname{Inf}_Q(\mathcal{S})$. So, it only remains to prove that $\mathcal{S}$ is inclusion-minimal. Suppose it is not, that is, there exists a proper subset $\mathcal{S}'$ of $\mathcal{S}$ such that x solves $\operatorname{Inf}_Q(\mathcal{S}')$. From Theorem 4.10, $\overline{x}$ can be written as a convex combination of $\mathcal{S}'$. Let β_i be

the corresponding coefficients and let $p_k \in \mathcal{S} \setminus \mathcal{S}'$. Then,

$$\overline{p}_k = \sum_{i:\, p_i \in \mathcal{S}'} \frac{\beta_i - \alpha_i}{\alpha_k} \overline{p}_i + \sum_{i \neq k:\, p_i \in \mathcal{S} \setminus \mathcal{S}'} \frac{\alpha_i}{\alpha_k} \overline{p}_i,$$

contradicting the fact that $\mathcal{S}$ is affinely independent, since the sum of the coefficients of the linear combination above is 1. Hence, $\mathcal{S}$ is inclusion-minimal. □

Observation 4.17. As a consequence of the previous theorem, a basis has at least 1 point and at most n points.

Observation 4.18. If $(\mathcal{S}, x)$ is a dual feasible S-pair, then x is a dual feasible solution to $\mathrm{Inf}_Q(\mathcal{P})$. If $(\mathcal{S}, x)$ is a dual feasible S-pair and x is primal feasible, then x solves $\mathrm{Inf}_Q(\mathcal{P})$ and $\mathcal{S}$ is an optimal basis.

Now, we present a Lemma about dual feasible S-pairs that will be useful later.

Lemma 4.19. *If $(\mathcal{S}, x)$ is a dual feasible S-pair such that $|\mathcal{S}| > 1$, then*

$$\left\|\overline{p}_j - \overline{p}_i\right\| > p_{j0} - p_{i0}, \quad \forall p_i, p_j \in \mathcal{S}.$$

Proof. First, suppose by contradiction that $\exists p_i \in \mathcal{S}$ s.t. $\left\|\overline{p}_j - \overline{p}_i\right\| < p_{j0} - p_{i0}$, for some $p_j \in \mathcal{S}$, and $p_{j0} - p_{i0} \geq 0$. By the triangle inequality,

$$\left\|\overline{p}_j - \overline{x}\right\| \leq \left\|\overline{p}_j - \overline{p}_i\right\| + \left\|\overline{p}_i - \overline{x}\right\| < p_{j0} - x_0,$$

contradicting the fact that $(\mathcal{S}, x)$ is a dual feasible S-pair.

Without loss of generality, now suppose by contradiction that $p_1, p_2 \in \mathcal{S}$ are s.t. $\|\overline{p}_2 - \overline{p}_1\| = p_{20} - p_{10} \geq 0$. Let $\mathcal{C}_i = \{z \in \mathbb{R}^n : \|\overline{p}_i - \bar{z}\| = p_{i0} - z_0\}$, $i = 1, 2$. The intersection of $\mathcal{C}_1$ and $\mathcal{C}_2$ is a line segment that passes through p_1 and p_2. As a consequence of $(\mathcal{S}, x)$ being a dual feasible S-pair, $x \in \mathcal{C}_1 \cap \mathcal{C}_2$. Thus, there exists $\beta \in \mathbb{R}$ such that $x = p_1 + \beta(p_2 - p_1)$. From $x_0 \leq p_{10}$ and $x_0 = p_{10} + \beta(p_{20} - p_{10})$, we conclude that $\beta \leq 0$.

From $\overline{x} = \sum_{i=1}^{s} \alpha_i \overline{p}_i$, for $\alpha_i > 0$, and $\sum_{i=1}^{s} \alpha_i = 1$, we obtain

$$\overline{p}_2 = \frac{\alpha_1 - 1 + \beta}{\beta - \alpha_2}\overline{p}_1 + \sum_{i \geq 3}^{s} \frac{\alpha_i}{\beta - \alpha_2}\overline{p}_i,$$

which contradicts the fact that $\overline{S}$ is affinely independent. □

Finally, we can use the result from Theorem 4.10 to calculate algebraically the solution to $\mathrm{Inf}_Q(\mathcal{P})$ and find the optimal basis, when $\mathcal{P}$ has only two points.

Theorem 4.20. *The solution to* $\mathrm{Inf}_Q(\{p_1, p_2\})$ *is given by*

$$x_0^* = \min\left(p_{10}, p_{20}, \tfrac{1}{2}\left(p_{10} + p_{20} - \|\overline{p}_1 - \overline{p}_2\|\right)\right), \quad \textit{and} \quad \overline{x}^* = \frac{(p_{10} - x_0^*)\overline{p}_2 + (p_{20} - x_0^*)\overline{p}_1}{(p_{10} - x_0^*) + (p_{20} - x_0^*)}.$$

If x^* *is either* p_1 *or* p_2*, then the optimal basis is* $S^* = \{x^*\}$*. Otherwise,* $S^* = \{p_1, p_2\}$*.*

Proof. Let $t = \frac{1}{2}\left(p_{10} + p_{20} - \|\overline{p}_1 - \overline{p}_2\|\right)$. Case 1: $x^* = p_1$. Recall from Theorem 4.4 that $x^* = p_1$ iff $\|\overline{p}_2 - \overline{p}_1\| \leq p_{20} - p_{10}$. Thus, $p_{10} \leq p_{20}$ and $t = p_{20}$. The case when $x^* = p_2$ is proved the same way.

Case 2: $\overline{x}^* \in \mathrm{ri}\,\mathrm{conv}(\overline{p}_1, \overline{p}_2)$, that is, $\overline{x}^* = \alpha\overline{p}_1 + (1-\alpha)\overline{p}_2$, for $0 < \alpha < 1$. Without loss of generality consider $p_{10} \leq p_{20}$. From the primal constraints binding at x^*:

$$\|\overline{p}_1 - \overline{x}^*\| = (1-\alpha)\|\overline{p}_1 - \overline{p}_2\| = p_{10} - x_0^*, \quad \|\overline{p}_2 - \overline{x}^*\| = \alpha\|\overline{p}_2 - \overline{p}_1\| = p_{20} - x_0^*,$$

we can find α:

$$\alpha = \frac{\|\overline{p}_1 - \overline{p}_2\| - (p_{10} - p_{20})}{2\|\overline{p}_1 - \overline{p}_2\|}.$$

As a consequence of Theorem 4.4, we have $\|\overline{p}_2 - \overline{p}_1\| > p_{20} - p_{10} \geq 0$, thus the value of α given by the formula above satisfies $0 < \alpha < 1$. We conclude that $x_0^* = t < p_{10} \leq p_{20}$. □

4.3 The dual simplex-like algorithm

In this section we introduce a dual simplex-like algorithm to solve the $\text{Inf}_{\mathcal{Q}}$ problem. The algorithm may be outlined as follows:

Input: A set of points $\mathcal{P} \in \mathbb{R}^n$.

Initialization: A dual feasible S-pair $(\mathcal{S}, x)$.

Loop: While x is primal infeasible:

(i) Get a $p^* \in \mathcal{P}$ corresponding to a violated constraint at x;

(ii) Obtain a dual feasible S-pair $(\mathcal{S}' \cup \{p^*\}, x')$, with $\mathcal{S}' \subseteq \mathcal{S}$ and $x'_0 < x_0$.

Output: x, the optimal solution to $\text{Inf}_{\mathcal{Q}}(\mathcal{P})$ and $\mathcal{S}$ an optimal basis.

The initialization step can be easily done by picking any point $p \in \mathcal{P}$ and considering $(\{p\}, p)$ as a dual feasible S-pair.

Step (ii) of the outline above is the core of the algorithm, and consists on a sequence of *curve searches* after which p^* becomes feasible and the value of the objective function strictly decreases.

Consider $(\mathcal{S}^j, x^j)$ the dual feasible S-pair at the beginning of iteration j. Let $\mathcal{S}^j := \{p_{j_1}, p_{j_2}, ..., p_{j_s}\}$, with $s = |\mathcal{S}^j|$, and $\overline{\mathcal{S}}^j := \{\overline{p}_{j_i} : p_{j_i} \in \mathcal{S}^j\}$. In order to maintain complementary slackness and dual feasibility, the algorithm restricts the search for the next iterate to the set of points that satisfy (4.9) and (4.10a):

$$\left\|\overline{p}_{j_i} - \overline{x}\right\| = p_{j_i 0} - x_0, \quad \forall\, p_{j_i} \in \mathcal{S}^j, \tag{4.9}$$

$$\overline{x} \in \text{conv}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}). \tag{4.10a}$$

Let us relax condition (4.10a), and simply consider

$$\overline{x} \in \text{aff}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}). \tag{4.10b}$$

In Section 4.3.1 we show that, if $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely independent, conditions (4.9, 4.10b) define a curve in $\mathbb{R}^n$. A *curve search* consists on the following: starting at x^j, we "move" on

the curve in the direction of decrease of x_0 until either dual feasibility is lost or p^* becomes feasible, whichever happens first. As we will see, the points where either dual feasibility is lost or p^* becomes feasible can be both calculated exactly, so the *curve search* is *exact*. If dual feasibility is lost first, a new curve search is performed next. Otherwise, when p^* becomes feasible first, a new loop iteration starts.

If $\overline{S}^j \cup \{\overline{p}^*\}$ is affinely dependent then (4.9, 4.10b) define a single point. In Section 4.3.2, we address this case explaining how to proceed in order to be able to perform a curve search afterward.

Before we proceed to the details of the algorithm, we first define some matrices and vectors that will be used throughout this discussion.

If $|\mathcal{S}^j| > 1$, we define the following matrices and vectors:

$$M = \begin{bmatrix} \overline{p}_{j_2} - \overline{p}_{j_1} & \overline{p}_{j_3} - \overline{p}_{j_1} & \dots & \overline{p}_{j_s} - \overline{p}_{j_1} \end{bmatrix}, \qquad M^+ = (M^T M)^{-1} M^T,$$

$$c = \begin{pmatrix} p_{j_2 0} - p_{j_1 0} \\ \vdots \\ p_{j_s 0} - p_{j_1 0} \end{pmatrix}, \qquad b = \frac{1}{2} \begin{pmatrix} \|\overline{p}_{j_2}\|^2 - p_{j_2 0}^2 - \|\overline{p}_{j_1}\|^2 + p_{j_1 0}^2 \\ \vdots \\ \|\overline{p}_{j_s}\|^2 - p_{j_s 0}^2 - \|\overline{p}_{j_1}\|^2 + p_{j_1 0}^2 \end{pmatrix},$$

$$u = (M^T M)^{-1}(b - M^T \overline{p}_{j_1}), \qquad v = (M^T M)^{-1} c,$$

$$w = -M^+(\overline{p}^* - \overline{p}_{j_1}), \qquad z = (I - MM^+)(\overline{p}^* - \overline{p}_{j_1}) = \overline{p}^* - \overline{p}_{j_1} + Mw.$$

M^+ is the Moore-Penrose inverse, or pseudo-inverse, of M. If $\overline{S}^j$ is affinely independent, then M is full column rank and M^+ is a left inverse in the sense that $M^+M = I$.

If $|\mathcal{S}^j| = 1$, then $M = [\,]$, $b = [\,]$, $c = [\,]$, $u = v = w = [\,]$, and $z = \overline{p}^* - \overline{p}_{j_1}$.

Lemma 4.21. *Conditions (4.9) are equivalent to:*

$$M^T \overline{x} = b + x_0 c, \tag{4.11a}$$

$$\left\|\overline{p}_{j_1} - \overline{x}\right\|^2 = (p_{j_1 0} - x_0)^2, \tag{4.11b}$$

$$x_0 \le \min_{p_{j_i} \in \mathcal{S}^j} \{p_{j_i 0}\}. \tag{4.11c}$$

Proof. If we square equations (4.9), we obtain

$$\left\|\overline{p}_{j_i} - \overline{x}\right\|^2 - (p_{j_i 0} - x_0)^2 = 0, \quad p_{j_i} - x_0 \geq 0, \quad \forall p_{j_i} \in \mathcal{S}^j,$$

which are equivalent to the following system of equations and one inequality:

$$\begin{aligned} x_0 &\leq \min_{p_{j_i} \in \mathcal{S}^j} \{p_{j_i 0}\}, \\ \left\|\overline{p}_{j_1} - \overline{x}\right\|^2 &= (p_{j_1 0} - x_0)^2, \\ \left\|\overline{p}_{j_i} - \overline{x}\right\|^2 - (p_{j_i 0} - x_0)^2 &= \left\|\overline{p}_{j_1} - \overline{x}\right\|^2 - (p_{j_1 0} - x_0)^2, \quad \forall p_{j_i} \in \mathcal{S}^j \setminus \{p_{j_1}\}. \end{aligned}$$

After several simplifications the last conditions are equivalent to

$$-(p_{j_i 0} - p_{j_1 0})x_0 + (p_{j_i} - p_{j_1})^T \overline{x} = \frac{1}{2}\left(\left\|\overline{p}_{j_i}\right\|^2 - p_{j_i 0}^2 - \left\|\overline{p}_{j_1}\right\|^2 + p_{j_1 0}^2\right), \quad \forall p_{j_i} \in \mathcal{S}^j,$$

that is, $-x_0 c + M^T \overline{x} = b$. □

Theorem 4.22. *Let $\overline{\mathcal{S}}^j$ be affinely independent. Conditions (4.9, 4.10b) are equivalent to*

$$\overline{x} = \overline{p}_{j_1} + M(u + x_0 v) + \alpha^* z, \tag{4.12a}$$

for $x_0 \leq \min_{p_{j_i} \in \mathcal{S}^j}\{p_{j_i 0}\}$ and $\alpha^* \in \mathbb{R}$ such that

$$(\alpha^*)^2 \|z\|^2 + \|M(u + x_0 v)\|^2 - (p_{10} - x_0)^2 = 0. \tag{4.12b}$$

Proof. From (4.10b) we know that there exist scalars $\alpha_1, \ldots, \alpha_s, \alpha^*$, such that $\sum_{i=1}^s \alpha_i + \alpha^* = 1$ and $\overline{x} = \sum_{i=1}^s \alpha_i \overline{p}_{j_i} + \alpha^* \overline{p}^*$, which implies

$$\overline{x} = M\alpha_{2:s} + \alpha^*(\overline{p}^* - \overline{p}_{j_1}) + \overline{p}_{j_1} \tag{4.13}$$

with $\alpha_{2:s}$ the vector with $\alpha_2, \ldots, \alpha_s$. Substituting (4.13) in (4.11a) we have

$$M^T M\alpha_{2:s} + \alpha^* M^T(\overline{p}^* - \overline{p}_{j_1}) + M^T \overline{p}_{j_1} = b + x_0 c.$$

Since $M^T M$ is invertible, this yields

$$\alpha_{2:s} = u + x_0 v + \alpha^* w, \tag{4.14}$$

with u, v, and w as defined before. Combining (4.13) and (4.14) we obtain (4.12a). Finally, by plugging (4.12a) in (4.11b) and observing that $M^T z = 0$, we obtain (4.12b). □

Note that the results of Lemma 4.21 and Theorem 4.22 are independent on x^j, and make no assumption on the affine dependence of $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$.

4.3.1 The curve search - $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely independent

At the beginning of a curve search, we always have that the pair $(\mathcal{S}^j, x^j)$ satisfies:

- $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely independent;
- $\left\|\overline{p}_{j_i} - \overline{x}^j\right\| = p_{j_i 0} - x_0^j, \quad \forall\, p_i \in \mathcal{S}^j$;
- $\overline{x}^j \in \operatorname{conv}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$, with $p^* \in \mathcal{P}$ corresponding to the chosen violated constraint.

The fact that these assumptions hold at the beginning of a curve search will be proved in Section 4.3.4. Note that, $(\mathcal{S}^j, x^j)$ may or may note be a dual feasible S-pair. Set $\mathcal{S}^j$ may not be a basis, but it always is a subset of a basis.

In this section we shall see that, provided $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely independent, the set of points that satisfy equations (4.9, 4.10b) topologically define a curve, that is, they are the image of a continuous function $\Gamma : I \subseteq \mathbb{R} \to \mathbb{R}^n$. Moreover, Γ can be written parametrically in terms of x_0.

The curve

If $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely independent, then $z \neq 0$. Thus, the quadratic equation (4.12b) can be solved for α^* (the coefficient of the affine combination associated with p^*) and we obtain:

$$\alpha^*(x_0) = \pm \frac{1}{\|z\|} \sqrt{(p_{j_1 0} - x_0)^2 - \|M(u + x_0 v)\|^2}. \tag{4.15}$$

Now, it is possible to write the points that satisfy (4.9, 4.10b) as a function of the single variable x_0, as Theorem 4.22 shows.

Theorem 4.23. *Consider $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ affinely independent. Conditions (4.9, 4.10b) define a curve $\Gamma : \left[-\infty, \min_{p_{j_i} \in \mathcal{S}^j}\{p_{j_i 0}\}\right] \to \mathbb{R}^n$ parameterized by x_0, such that $\Gamma(x_0) = (x_0; \overline{\Gamma}(x_0))$ with*

$$\overline{\Gamma}(x_0) := \overline{p}_{j_1} + M(u + x_0 v) \pm \frac{z}{\|z\|}\sqrt{(p_{j_1 0} - x_0)^2 - \|M(u + x_0 v)\|^2}.$$

We are not interested in the whole curve defined by Γ, but only on the portion that intersects $\text{conv}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$, condition (4.10a), that is, that corresponds to a nonnegative α^*. Corollary 4.24 shows how the points on the curve that are also in $\text{conv}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$ can be written as an affine combination of $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$. This result will allow the algorithm to control dual feasibility during the curve search.

Corollary 4.24. *Consider $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ affinely independent. Conditions (4.9, 4.10a) define a curve $\Gamma^+ : \left[-\infty, \min_{p_{j_i} \in \mathcal{S}^j}\{p_{j_i 0}\}\right] \to \mathbb{R}^n$ parameterized by x_0: $\Gamma^+(x_0) = (x_0; \overline{\Gamma}^+(x_0))$ with*

$$\overline{\Gamma}^+(x_0) := \overline{p}_{j_1} + M(u + x_0 v) + \frac{z}{\|z\|}\sqrt{(p_{j_1 0} - x_0)^2 - \|M(u + x_0 v)\|^2}.$$

Moreover,

$$\overline{\Gamma}^+(x_0) = \sum_{i=1}^{s} \alpha_i(x_0)\overline{p}_{j_i} + \alpha^*(x_0)\overline{p}^*, \tag{4.16}$$

with

$$\alpha^*(x_0) = \frac{1}{\|z\|}\sqrt{(p_{j_1 0} - x_0)^2 - \|M(u + x_0 v)\|^2}, \tag{4.17a}$$

$$\alpha_1(x_0) = 1 - 1_{s-1}^T(u + x_0 v) - \alpha^*(x_0)\left(1_{s-1}^T w + 1\right), \tag{4.17b}$$

$$\alpha_{2:s}(x_0) = u + x_0 v + \alpha^*(x_0) w, \tag{4.17c}$$

where $1_{s-1} \in \mathbb{R}^{s-1}$ with entries all 1, and $\alpha_{2:s}$ the vector $(\alpha_2, \alpha_3, ..., \alpha_s)$.

Note that the assumptions on x^j at the beginning of a curve search imply that x^j is on the curve defined by Γ^+, that is, $x^j = \Gamma^+(x_0^j)$.

Observation 4.25. It is now possible to write explicitly the dual variables that correspond

to $(x_0; \overline{\Gamma}^+(x_0))$ as functions of x_0:

$$
\begin{aligned}
y_{j_i 0}(x_0) &= \frac{\alpha_i(x_0)(p_{j_i 0} - x_0)}{\sum_{k=1}^{s} \alpha_k(x_0)(p_{j_k 0} - x_0) + \alpha^*(x_0)(p_0^* - x_0)}, && \text{for } p_{j_i} \in \mathcal{S}^j, \\
\overline{y}_{j_i}(x_0) &= -\frac{y_{j_i 0}(x_0)}{p_{j_i 0} - x_0} \left(\overline{p}_{j_i} - \overline{\Gamma}^+(x_0) \right), && \text{for } p_{j_i} \in \mathcal{S}^j, \\
y_0^*(x_0) &= \frac{\alpha_i(x_0)(p_0^* - x_0)}{\sum_{k=1}^{s} \alpha_k(x_0)(p_{j_k 0} - x_0) + \alpha^*(x_0)(p_0^* - x_0)}, && \text{for } p^*, \\
\overline{y}^*(x_0) &= -\frac{y_0^*(x_0)}{p_0^* - x_0} \left(\overline{p}^* - \overline{\Gamma}^+(x_0) \right), && \text{for } p^*, \\
y_k &= 0, && \text{for } p_k \notin \mathcal{S}^j \cup \{p^*\}.
\end{aligned}
$$

We only need to satisfy $\alpha_i(x_0) \geq 0$ and $\alpha^*(x_0) \geq 0$, in order to ensure that $(x_0, \overline{\Gamma}^+(x_0))$ corresponds to a dual feasible solution.

We now show some properties of the curve. Let N be a matrix whose columns constitute a basis to the nullspace of the linear space parallel to $\operatorname{aff}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$. Define $\mathcal{H}$ and $\mathcal{C}_1$ as

$$
\begin{aligned}
\mathcal{H} &:= \left\{ x \in \mathbb{R}^n : A^T x = \begin{pmatrix} b \\ N^T \overline{p}_{j_1} \end{pmatrix} \right\} \quad \text{with} \quad A = \left[\begin{array}{c|c} -c^T & 0 \\ \hline M & N \end{array}\right], \\
\mathcal{C}_1 &:= \left\{ x \in \mathbb{R}^n : \left\| \overline{p}_{j_1} - \overline{x} \right\|^2 = (p_{j_1 0} - x_0)^2 \right\}.
\end{aligned}
$$

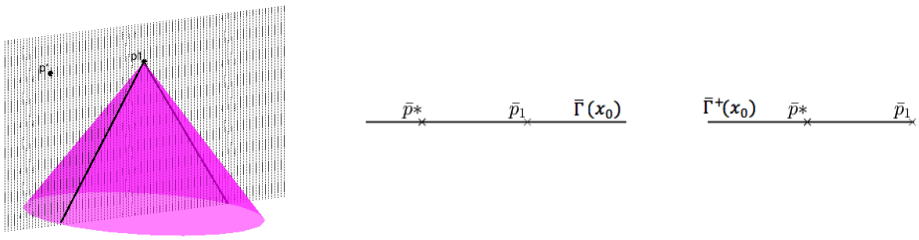
From Lemma 4.21 and condition (4.10b) we conclude that Γ corresponds to the intersection of $\mathcal{H}$ with $\mathcal{C}_1$, for x_0 satisfying the inequality (4.11c). When $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely independent, N has $n - 1 - s$ columns, and $[M|N]$ has $n - 2$ columns and is full column rank. Thus, A^T is full row rank, meaning $\mathcal{H}$ is a 2−dimensional affine space in $\mathbb{R}^n$ (a plane). The curve Γ is then the intersection of a plane with the lower nappe of $\mathcal{C}_1$, that is, a conic section. Note that $\overline{\Gamma}(x_0)$ is the projection of $\Gamma(x_0)$ onto the hyperplane $\{x \in \mathbb{R}^n : x_0 = 0\}$.

Proposition 4.26 (Curve shape)**.** *Consider $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ affinely independent.*

i. If $|\mathcal{S}^j| = 1$, then $\Gamma(x_0)$ is two line segments that intersect at p_{j_1}, and $\overline{\Gamma}(x_0)$ is a line;

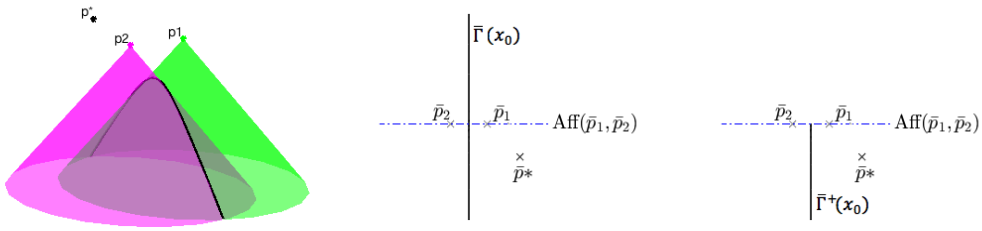
ii. If $|\mathcal{S}^j| > 1$, then $\Gamma(x_0)$ is a branch of a hyperbola. In particular, if $c = 0$, $\overline{\Gamma}(x_0)$ is a line.

iii. $\Gamma(x_0)$ is symmetric with respect to a reflection through $\{x \in \mathbb{R}^n : \overline{x} \in \operatorname{aff}(\overline{\mathcal{S}}^j)\}$.



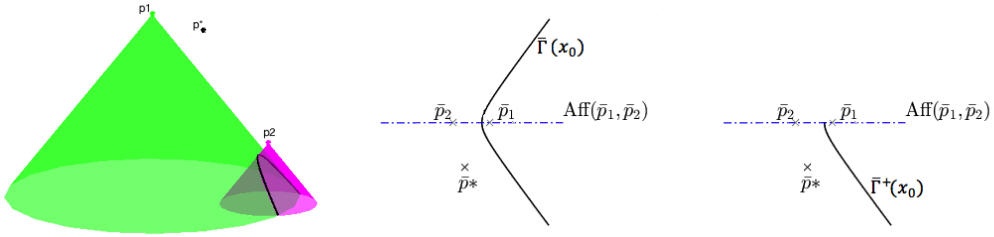
(a) $\Gamma(x_0)$, $x_0 \leq p_{10}$ (b) $\bar{\Gamma}(x_0)$, $x_0 \leq p_{10}$ (c) $\bar{\Gamma}^+(x_0)$, $x_0 \leq p_{10}$

Figure 4.7: Illustration of the curve when $\mathcal{S}^j = \{p_1\} \subset \mathbb{R}^3$.



(a) $\Gamma(x_0)$, $x_0 \leq p_{10}$ (b) $\bar{\Gamma}(x_0)$, $x_0 \leq p_{10}$ (c) $\bar{\Gamma}^+(x_0)$, $x_0 \leq p_{10}$

Figure 4.8: Illustration of the curve when $\mathcal{S}^j = \{p_1, p_2\} \subset \mathbb{R}^3$ and $c = 0$.



(a) $\Gamma(x_0)$, $x_0 \leq p_{10}$ (b) $\bar{\Gamma}(x_0)$, $x_0 \leq p_{10}$ (c) $\bar{\Gamma}^+(x_0)$, $x_0 \leq p_{10}$

Figure 4.9: Illustration of the curve when $\mathcal{S}^j = \{p_1, p_2\} \subset \mathbb{R}^3$ and $c \neq 0$.

Proof. i. Assume $\mathcal{S}^j$ has a single point p_{j_1}. Then $M = []$ and $b = c = 0$, and Γ becomes:

$$\Gamma(x_0) = \left(x_0; \overline{\Gamma}(x_0)\right) = \left(x_0, \overline{p}_{j_1} \pm \frac{z}{\|z\|} |p_{j_1 0} - x_0|\right), \quad x_0 \leq p_{10}.$$

Another way to see that the curve corresponds to two line segments, is to note that $\mathcal{H} = \{(x_0; \overline{x}) \in \mathbb{R}^n : N^T(\overline{x} - \overline{p}_{j_1}) = 0\}$. Thus, $\mathcal{H}$ is parallel to the axis of the cone $\mathcal{C}_1$ since the linear space corresponding to $\mathcal{H}$ contains the vector $(1, 0, ..., 0)$. Since $\mathcal{H}$ contains the vertex of $\mathcal{C}_1$, the intersection of $\mathcal{H}$ with $\mathcal{C}_1$ is two opposite rays of the cone (Figure 4.7).

ii. Assume $|\mathcal{S}^j| > 1$ and $c = 0$. Then $p_{j_i 0} = p_{j_1 0}$, for all $i = 2, ..., s$, and $\mathcal{H} = \{(x_0; \overline{x}) \in \mathbb{R}^n : N^T(\overline{x} - \overline{p}_{j_1}) = 0, M^T\overline{x} = b\}$. Consequently, $\mathcal{H}$ is parallel to the axis of the cone, but this time $p_{j_1} \notin \mathcal{H}$ since $M^T\overline{p}_{j_1} - b \neq 0$. The intersection of a double cone and a plane parallel to its axis that does not pass through its vertex is a hyperbola. In this case,

$$\Gamma(x_0) = \left(x_0; \overline{\Gamma}(x_0)\right) = \left(x_0, \overline{p}_{j_1} + Mu \pm \frac{z}{\|z\|}\sqrt{(p_{j_1 0} - x_0)^2 - \|Mu\|^2}\right).$$

Since we only care about the lower nappe of $\mathcal{C}_1$, we only have one of the branches (Figure 4.8). Note that $Mu \neq 0$, and the image of $\overline{\Gamma}(x_0)$ in $\mathbb{R}^{n-1}$ is a line.

Now, assume $|\mathcal{S}^j| > 1$ and $c \neq 0$. To prove that the image of Γ is a hyperbola, we prove that $\mathcal{H}$ intersects both nappes of $\mathcal{C}_1$. Consider the intersection of Γ with $\text{aff}(\overline{\mathcal{S}}^j)$, obtained by solving $\alpha^*(x_0) = 0$. The solution(s) are such that $x_0 = p_{j_1 0} \pm \|M(u + x_0 v)\|$, thus $x_0 \leq p_{10}$ and $x_0 \geq p_{10}$. Note that $M(u + p_{j_1 0} v) = 0$ iff $b - M^T\overline{p}_{j_1} + p_{j_1 0} c = 0$ iff $\left\|\overline{p}_{j_i} - \overline{p}_{j_1}\right\|^2 = (p_{j_i 0} - p_{j_1 0})^2$ for all $p_{j_i} \in \mathcal{S}^j \setminus \{p_{j_1}\}$. The latter equality is not true due to the fact that $\mathcal{S}^j$ is a subset of a basis (Lemma 4.19). As a result, $x_0 = p_{10}$ is not a solution. Therefore, $\alpha^*(x_0) = 0$ has two solutions, one such that $x_0 < p_{10}$ and the other such that $x_0 > p_{10}$, that is, each one corresponding to one of the nappes of $\mathcal{C}_1$ (Figure 4.9).

iii. Let x be a point on the curve. We know that $\overline{x} \in \text{aff}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$. In particular, $\overline{x} \in \text{aff}(\overline{\mathcal{S}}^j)$ iff $z^T(\overline{x} - \overline{p}_{j_1}) = 0$. The reflection operator with respect to the affine hyperplane $\text{aff}(\overline{\mathcal{S}}^j)$ for points in $\text{aff}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$ is $\mathcal{R}(\overline{x}) = \overline{x} - 2z^T(\overline{x} - \overline{p}_{j_1})\frac{\|z\|^2}{z}$. It is easy to prove that,

since $M^T z = 0$, it holds that $\mathcal{R}(\overline{\Gamma}^+(x_0)) = \overline{\Gamma}^-(x_0)$ and vice-versa, for

$$\overline{\Gamma}^-(x_0) := \overline{p}_{j_1} + M(u + x_0 v) - \frac{z}{\|z\|}\sqrt{(p_{j_1 0} - x_0)^2 - \|M(u + x_0 v)\|^2}.$$

□

The curve search procedure

The curve search consists on moving on the curve defined by $\Gamma^+(x_0)$ in the direction of decrease of x_0, starting at x^j. This movement stops either right before dual feasibility is lost or on the point where the violated constraint becomes feasible. Using the conclusions from the previous section, finding the point on the curve where to stop can be formally described by the following problem:

$$\begin{aligned}
\min \quad & x_0 \\
\text{s.t.} \quad & \overline{x} = \overline{\Gamma}^+(x_0) \\
& \|\overline{p}^* - \overline{x}\| \geq p_0^* - x_0 \\
& \alpha_i(x_0) \geq 0, \quad i = 1, ..., s \\
& \alpha^*(x_0) \geq 0 \\
& x_0 \leq x_0^j.
\end{aligned}$$

Note that in the problem above we do not have $x_0 \leq \min_{p_{j_i} \in \mathcal{S}^j}\{p_{j_i 0}\}$, since x_0^j already satisfies the same inequality.

We define *partial step* and *full step* as:

Partial step: the largest step on the curve without violating dual feasibility, when we move in the direction of decrease of x_0 starting at x^j. Equivalently, this is the largest step on the curve until one of the dual variables, which vary non-linearly as we move on the curve, becomes zero. We denote the point where this happens by

$$x^P := \Gamma^+\left(x_0^P\right) = \left(x_0^P; \overline{\Gamma}^+\left(x_0^P\right)\right).$$

Full step: the smallest step on the curve that makes the constraint corresponding to p^* feasible (active), when we move in the direction of decrease of x_0 starting at x^j. When it exists, we denote that point by

$$x^F := \Gamma^+\left(x_0^F\right) = \left(x_0^F; \overline{\Gamma}^+\left(x_0^F\right)\right).$$

If $x_0^F > x_0^P$, then the constraint corresponding to p^* becomes feasible while all dual variables associated with $\mathcal{S}^j$ are kept feasible and non-zero. That implies that x^F is the solution to $\mathrm{Inf}_Q(\mathcal{S}^j \cup \{p^*\})$. The algorithm then goes to the beginning of a new iteration with a new dual feasible S-pair $(\mathcal{S}^{j+1}, x^{j+1}) := (\mathcal{S}^j \cup \{p^*\}, x^F)$.

On the other hand, if $x_0^F \leq x_0^P$, then at least a dual variable becomes 0 before, or exactly at the same time that, p^* becomes primal feasible. When this is the case, the algorithm drops a point p_{j_k} from $\mathcal{S}^j$ that corresponds to a dual variable that became zero. The algorithm then performs a new curve search with the new $\mathcal{S}^j$ starting at x^P.

More than one dual variable may become zero simultaneously at x^P. In this case, the algorithm chooses any p_{j_k} corresponding to one of such zero dual variables to be deleted from $\mathcal{S}^j$. The following curve search will be a degenerate case where x will not change, and simply another point whose dual variable is zero at x^P is dropped. Cycling will not occur since at each degenerate curve search one point is deleted from $\mathcal{S}^j$. Eventually, after some number of points are deleted from $\mathcal{S}^j$, $x^P \in \mathrm{ri}\,\mathrm{conv}(\overline{\mathcal{S}}^j)$, and in the next curve search x will change (a movement on the curve occurs).

For the remainder of this section, unless mentioned otherwise, assume $|\mathcal{S}^j| > 1$. The particular case of $|\mathcal{S}^j| = 1$ will be addressed separately.

The full step - calculating x_0^F

Since $\left\|\overline{p}^* - \overline{x}^j\right\| \geq p_0^* - x_0^j$, then x^F is the first point on the curve that satisfies

$$\|\overline{p}^* - \overline{x}\| = p_0^* - x_0. \tag{4.18a}$$

Such point can be obtained using conditions (4.11) and (4.18b):

$$(\overline{p}^* - \overline{p}_{j_1})^T \overline{x} = b^* + x_0 c^*, \tag{4.18b}$$

with $c^* = p_0^* - p_{j_1 0}$, $b^* = \frac{1}{2}\left(\|\overline{p}^*\|^2 - (p_0^*)^2 - \|\overline{p}_{j_1}\|^2 + p_{j_1 0}^2\right)$, and $x_0 \leq p_0^*$. Therefore, x_0^F is the solution to the following problem:

$$\begin{aligned} \max \quad & x_0 \\ \text{s.t.} \quad & \overline{x} = \overline{\Gamma}^+(x_0) \\ & (\overline{p}^* - \overline{p}_{j_1})^T \overline{x} = b^* + x_0 c^* \\ & x_0 \leq \min\{p_0^*, x_0^j\}. \end{aligned} \tag{4.19}$$

It is easy to see that the solution to (4.19) may not exist: equations (4.9,4.18a) seek the position of the vertex of a second-order cone that has points $S^j \cup \{p^*\}$ on its boundary, which is not guaranteed to exist. To solve (4.19), we could plug the equation of $\overline{\Gamma}^+(x_0)$ in (4.18b), and solve for x_0. However, we realized this approach would result in rather intricate calculations involving square roots. So, instead, we go back to the results of Lemma 4.21 and Theorem 4.22 and proceed as follows. We start by plugging (4.12a) in (4.18b), solve for α^*, obtaining α^* as a function of x_0:

$$\alpha^*(x_0) = \frac{1}{(\overline{p}^* - \overline{p}_{j_1})^T z}(b^* + x_0 c^* - (\overline{p}^* - \overline{p}_{j_1})^T (M(u + x_0 v) + \overline{p}_{j_1})), \tag{4.20}$$

Now, if we plug (4.20) back in (4.12a), we get $\overline{x} = q + x_0 r + \overline{p}_{j_1}$, with

$$q = Mu + \frac{b^* - (\overline{p}^* - \overline{p}_{j_1})^T (Mu + \overline{p}_{j_1})}{(\overline{p}^* - \overline{p}_{j_1})^T z} \text{ and } r = Mv + \frac{c^* - (\overline{p}^* - \overline{p}_{j_1})^T (Mv)}{(\overline{p}^* - \overline{p}_{j_1})^T z}.$$

Finally, we plug $\overline{x} = q + x_0 r + \overline{p}_{j_1}$ in (4.11b), which becomes $\|q + x_0 r\|^2 = (p_{j_1 0} - x_0)^2$, and solve it for x_0.

Note that, in this process, we never imposed that $\alpha^*(x_0) \geq 0$ (recall that $\alpha^*(x_0)$ is given by (4.20)). If the solution(s) to the quadratic equation are real, we only keep the one(s) that satisfy both conditions $\alpha^*(x_0) \geq 0$ and $x_0 \leq \min\left\{x_0^j, p_0^*\right\}$. If there are more than one

Algorithm 4.1 Full step procedure

input: Matrix M, vectors u, v, w, z, points x^j, p^*, and set S^j
output: x_0^F

1: Define vectors c^*, b^*, q, and r;
2: $\hat{\mathcal{X}}^* \leftarrow \{x_0 : \|q + x_0 r\|^2 = (p_{j_1 0} - x_0)^2\}$;
3: $\mathcal{X}^* \leftarrow \{x_0 \in \hat{\mathcal{X}}^* : \alpha^*(x_0) \geq 0 \wedge x_0 \leq p_0^* \wedge x_0 \leq x_0^j\}$, for $\alpha^*(x_0)$ defined in (4.20);
4: **if** $\mathcal{X}^* = \emptyset$ **then**
5: $\quad x_0^F \leftarrow -\infty$.
6: **else**
7: $\quad x_0^F \leftarrow \max\{\mathcal{X}^*\}$.
8: **end if**

such solution, x_0^F is the largest one. If we are left with no solutions or the solutions of the quadratic equation are not real, then the solution to (4.19) does not exist. In this case, we shall consider $x_0^F = -\infty$. The details of calculating x_0^F are described in Algorithm 4.1.

The partial step - calculating x_0^P

When $\overline{S}^j \cup \{\overline{p}^*\}$ is affinely independent, Corollary 4.24 gives all the ingredients necessary to control the dual variables as we move on the curve. As a consequence, x_0^P is the solution to

$$\begin{aligned} \min \quad & x_0 \\ \text{s.t.} \quad & \alpha_i(x_0) \geq 0, \quad i = 1, ..., s \\ & \alpha^*(x_0) \geq 0 \\ & x_0 \leq x_0^j. \end{aligned} \tag{4.21}$$

Note that, if we consider the definitions of $\alpha_i(x_0)$, $i = 1, ..., m$, given by (4.17), the constraint $\alpha^*(x_0) \geq 0$ is already implied.

Since $\overline{x}^j \in \text{conv}(\overline{S}^j \cup \{\overline{p}^*\})$, we know that $\alpha^*(x_0^j) \geq 0$ and $\alpha_i(x_0^j) \geq 0$, for all $i = 1, ..., s$. So, the solution to (4.21) can be found by solving the s equations $\alpha_i(x_0) = 0$, $i = 1, ..., s$, and picking the largest of the solutions that satisfy $x_0 \leq x_0^j$. These equations are radical equations that can be easily solved by isolating the square root term in one side, squaring both sides, solving the resulting quadratic equation, and finally discarding any extraneous solutions.

Geometrically, the solution to (4.21) corresponds to the largest step on the curve where

Algorithm 4.2 Partial step procedure

input: Matrix M, vectors u, v, w, z, points x^j, p^*, and set $\mathcal{S}^j$
output: x_0^P and index k.

1: **for** $i = 1, ..., s$ **do**
2: Define $\alpha_i(x_0)$ as in (4.17);
3: $\hat{\mathcal{X}}_i \leftarrow \{x_0 : \alpha_i(x_0) = 0\}$;
4: $\mathcal{X}_i \leftarrow \left\{x_0 \in \hat{\mathcal{X}}_i : x_0 \text{ is real} \wedge x_0 \leq x_0^j\right\}$;
5: **end for**
6: $x_0^P \leftarrow \max\{\cup_{i=1,...,s} \mathcal{X}_i\}$;
7: $k \leftarrow i$ such that $x_0^P \in \mathcal{X}_i$.

the boundary of $\operatorname{conv}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$ is found. The point or points of $\mathcal{S}^j$ opposed to the intersected facet(s) of $\operatorname{conv}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$ correspond to the dual variable(s) that became zero. Note that, for some i the equation $\alpha_i(x_0) = 0$ may not have a solution, since the curve may not intersect the supporting hyperplane of the corresponding facet of $\operatorname{conv}(\overline{\mathcal{S}}^j \cup \{\overline{p}^*\})$.

The procedure of calculating x_0^P is summarized in Algorithm 4.2. Note that this procedure will only be performed when $|\mathcal{S}^j| > 1$.

The special case of $|\mathcal{S}^j| = 1$

The curve search procedure always imposes the next iterate to satisfy conditions (4.9) for a subset of $\mathcal{S}^j$. When p^* solves $\operatorname{Inf}_Q(\mathcal{S}^j \cup \{p^*\})$, after a series of partial steps where a point from $\mathcal{S}^j$ is dropped each time, the algorithm gets to a curve search where $\mathcal{S}^j$ has a single point, $\mathcal{S}^j = \{p_{j_1}\}$. The next iterate should be p^*, the solution of $\operatorname{Inf}_Q(\{p_{j_1}, p^*\})$, which does not satisfy (4.9) for p_{j_1}. Therefore, when $|\mathcal{S}^j| = 1$, the curve search procedure explained previously may fail to return the correct iterate. To avoid that, when $|\mathcal{S}^j| = 1$, we use the result of Theorem 4.20 to find the next iterate in replacement of a curve search. If the formulas of Theorem 4.20 return p^*, then the algorithm goes to the beginning of a new iteration with dual feasible S-pair $(\mathcal{S}^{j+1}, x^{j+1}) = (\{p^*\}, p^*)$. Otherwise, the formulas of Theorem 4.20 will return a solution $x^{j+1} \in \operatorname{ri}\operatorname{conv}(\{\overline{p}_{j_1}, \overline{p}^*\})$, and the algorithm goes to the beginning of a new iteration with dual feasible S-pair $(\mathcal{S}^{j+1}, x^{j+1})$ such that $\mathcal{S}^{j+1} = \{p_{j_1}, p^*\}$. Note that Theorem 4.20 can never return p_{j_1} as the solution, otherwise p^* would be feasible for the current iterate.

(a) Case $|\mathcal{S}^j| = 2$ and $p^* \in \text{aff}(\overline{\mathcal{S}}^j)$

(b) Case $|\mathcal{S}^j| = 3$

Figure 4.10: Illustration of the affinely dependent case in $\mathbb{R}^3$.

4.3.2 The case when $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely dependent

Consider $(\mathcal{S}^j, x^j)$ a dual feasible S-pair and p^* corresponding to the selected violated constraint. If $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely dependent, then $z = 0$, and equations (4.12) from Theorem 4.22 become:

$$\overline{x} = M(u + x_0 v) + \overline{p}_{j_1}, \tag{4.22a}$$

$$\|M(u + x_0 v)\|^2 = (p_{j_1 0} - x_0)^2, \tag{4.22b}$$

respectively, for $x_0 \leq \min_{p_{j_i} \in \mathcal{S}^j}\{p_{j_i 0}\}$. The current dual feasible solution, x^j, is the unique solution that satisfies (4.22) and the inequality. The other solution of (4.22b) is such that $x_0 > p_{j_1 0} \geq \min_{p_{j_i} \in \mathcal{S}^j}\{p_{j_i 0}\}$, since $\|M(u + x_0 v)\| > 0$ (otherwise $x = p_{j_1}$, $\mathcal{S}^j = \{p_{j_1}\}$, and we could not possibly have affinely dependence). Thus, when $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely dependent equations (4.9, 4.10a) do not define a piece of a curve, but the single point x^j. Figure 4.10 illustrates this situation.

Recall that equations (4.14) from the proof of Theorem 4.22 still hold for x_0^j, when $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely dependent. They yield

$$\alpha_{2:s}(\alpha^*) = u + x_0^j v + \alpha^* w, \tag{4.23a}$$

$$\alpha_1(\alpha^*) = 1 - \alpha^* - 1^T(u + x_0^j v + \alpha^* w). \tag{4.23b}$$

Algorithm 4.3 Update $\mathcal{S}^j$ procedure

input: Vectors u, v, w, point x^j.
output: Index k.

1: Define vectors ρ and σ:

$$\rho = \begin{pmatrix} 1 - 1^T(u + x_0^j v) \\ u + x_0^j v \end{pmatrix} \text{ and } \sigma = \begin{pmatrix} -1 - 1^T w \\ w \end{pmatrix}.$$

2: Get k such that

$$-\frac{\rho_k}{\sigma_k} := \min_{i=1,\dots,s} \left\{ -\frac{\rho_i}{\sigma_i} : \sigma_i < 0 \right\}. \tag{4.25}$$

Formulas (4.23) give all possible ways of writing $\overline{x}^j$ as an affine combination of $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$, as a function of the scalar α^*. We can use them to obtain $\overline{x}^j$ written as a convex combination of $\overline{p}^*$ and a proper subset of $\overline{\mathcal{S}}^j$. To do that, consider the following problem

$$\begin{aligned} \max \quad & \alpha^* \\ \text{s.t.} \quad & \alpha_i(\alpha^*) \geq 0, \ i = 1, \dots, s \\ & \alpha^* \geq 0, \end{aligned} \tag{4.24}$$

which can easily be solved using a minimum ratio rule, as explained in Algorithm 4.3. Note that, in Algorithm 4.3, ρ is the vector of the coefficients of $\overline{x}^j$ written as an affine combination of $\overline{\mathcal{S}}^j$, thus $\rho \geq 0$ by assumption. Similarly, from $z = \overline{p}^* + \overline{p}_1 + Mw = 0$, $-\sigma$ is the vector of the coefficients of $\overline{p}^*$ written as an affine combination of $\overline{\mathcal{S}}^j$.

Let $p_{j_k} \in \mathcal{S}^j$ correspond to the index obtained in (4.25) from Algorithm 4.3. In Section 4.3.4 it is proved that $\overline{\mathcal{S}}^j \backslash \{\overline{p}_{j_k}\} \cup \{\overline{p}^*\}$ is affinely independent and $\overline{x}^j \in \text{conv}(\overline{\mathcal{S}}^j \backslash \{\overline{p}_{j_k}\} \cup \{\overline{p}^*\})$. If we drop p_{j_k} from $\mathcal{S}^j$, the assumptions we made at the beginning of a curve search hold, and a curve can then be defined.

We can think of the procedure explained in this section as a degenerate curve search with a partial step of zero length. In fact, if we imagine p^* suffering a small perturbation such that $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ becomes affinely independent, the point p_{j_k} to be dropped would be the one selected by the partial step procedure too.

4.3.3 Pseudo-code

We now aggregate the results/discussion of the previous sections on Algorithm 4.4.

Algorithm 4.4 Dual algorithm for the infimum of $\mathcal{P}$ with respect to $\mathcal{Q}$

input: $\mathcal{P}$; dual feasible S-pair $(\mathcal{S}^0, x^0)$ (optional).
output: x, $\mathcal{S}$, the optimal solution and basis, respectively.

Initialization:
if $(\mathcal{S}^0, x^0)$ not given in input **then**
  Choose any point $p \in \mathcal{P}$; $x^0 \leftarrow p$; $\mathcal{S}^0 \leftarrow \{p\}$.
end if
Loop:
for $j = 0, 1, \ldots.$ **do**
  if $x^j \preceq_{\mathcal{Q}} p_i$ for all $p_i \in \mathcal{P}$ **then** ▷ Optimality check
    x^j is the optimal solution and $\mathcal{S}^j$ an optimal basis. **Stop.**
  else
    Get $p^* \in \mathcal{P}$ s.t. $x^j \succ_{\mathcal{Q}} p^*$.
  end if
  Define matrix M and vectors b, c, u, v, w, and z.
  if $\|z\| = 0$ **then** ▷ $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ affinely dependent
    Get k from Algorithm 4.3; $\mathcal{S}^j \leftarrow \mathcal{S}^j \setminus \{p_{j_k}\}$.
  end if
  Curve search:
  if $|\mathcal{S}^j| = 1$ **then**
    Get x^{j+1} and $\mathcal{S}^{j+1}$, the solution to $\mathrm{Inf}_{\mathcal{Q}}(\mathcal{S}^j \cup \{p^*\})$, from Theorem 4.20;
    Go to *line 5*.
  else
    Get x_0^F from Algorithm 4.1.
    Get x_0^P from Algorithm 4.2.
    if $x_0^F \leq x_0^P$ **then** ▷ Partial step is taken.
      $x^j \leftarrow (x_0^P; \overline{\Gamma}^+(x_0^P))$; $\mathcal{S}^j \leftarrow \mathcal{S}^j \setminus \{p_{j_k}\}$; Go to *line 14*.
    else ▷ Full step is taken.
      $x^{j+1} \leftarrow (x_0^F; \overline{\Gamma}^+(x_0^F))$; $\mathcal{S}^{j+1} \leftarrow \mathcal{S}^j \cup \{p^*\}$; Go to *line 5*.
    end if
  end if
end for

4.3.4 Finiteness and correctness of the algorithm

For the proof of the correctness of Algorithm 4.4, we will follow a similar approach as in [51]. We first introduce the following definition:

Definition 4.27. The triple $(x, \mathcal{S}, p)$ is said to be a *(violated) V-triple* if the following four conditions hold:

(a) $\overline{\mathcal{S}} \cup \{\overline{p}\}$ is affinely independent,

(b) $\|\overline{p} - \overline{x}\| \geq (p_0 - x_0)$,

(c) $\|\overline{p}_i - \overline{x}\| = (p_{i0} - x_0)$, for all $p_i \in \mathcal{S}$,

(d) $\overline{x} \in \operatorname{conv}(\overline{\mathcal{S}} \cup \{\overline{p}\})$.

The correctness of the algorithm relies on a series of lemmas that are proved next.

Lemma 4.28. *When $|\mathcal{S}^j| = 1$, $(\mathcal{S}^{j+1}, x^{j+1})$, obtained from applying Theorem 4.20, is a dual feasible S-pair.*

Proof. This is a trivial consequence of Theorem 4.20. □

Lemma 4.29. *Assume that, at the beginning of a curve search, we have a V-triple $(x^j, \mathcal{S}^j, p^*)$ such that $|\mathcal{S}^j| > 1$. Suppose a full step is taken, that is, the solution given by the full step procedure (Algorithm 4.1) exists and $x_0^F > x_0^P$. Then,*

i. $(\mathcal{S}^j \cup \{p^*\}, x^F)$ *is a dual feasible S-pair,*

ii. x^F *solves* $\operatorname{Inf}_Q(\mathcal{S}^j \cup \{p^*\})$,

iii. $x_0^F \leq x_0^j$.

Proof. It is easy to see that $(\mathcal{S}^j \cup \{p^*\}, x^F)$ is a dual feasible S-pair. First, $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely independent by assumption. Additionally, $\overline{x}^F \in \operatorname{ri}\operatorname{conv}(\overline{\mathcal{S}}^j)$, because otherwise there would exist an $\alpha_i(x_0^F) = 0$, meaning a partial step would have been taken instead, contradicting

the assumption that $x_0^F > x_0^P$. Finally, recall that the full step finds a point x^F such that

$$\begin{aligned} &\left\|\overline{p}^* - \overline{x}^F\right\| = p_0^* - x_0^F, \\ &\left\|\overline{p}_i - \overline{x}^F\right\| = p_{i0} - x_0^F, \quad \forall\, p_i \in \mathcal{S}^j \\ &\overline{x} \in \operatorname{aff}(\overline{\mathcal{S}} \cup \{\overline{p}^*\}), \end{aligned}$$

whenever these conditions are feasible. Finally, the fact that x^F solves $\operatorname{Inf}_Q(\mathcal{S}^j \cup \{p^*\})$ is a consequence of Theorem 4.16. Lastly, $x_0^F \leq x_0^j$ follows from the definition of the full step procedure. □

Lemma 4.30. *Assume that, at the beginning of a curve search, we have a V-triple* $(x^j, \mathcal{S}^j, p^*)$ *such that* $|\mathcal{S}^j| > 1$. *Suppose a partial step was taken, that is,* $x_0^F \leq x_0^P$. *Let* $p_k \in \mathcal{S}^j$ *be the point selected to be dropped by the partial step procedure (Algorithm 4.2). Then,*

i. $(x^P, \mathcal{S}^j \setminus \{p_k\}, p^*)$ *is a V-triple,*

ii. $x_0^P \leq x_0^j$.

Proof. Let us prove properties (a-d) from Definition 4.27 for $(x^P, \mathcal{S}^j \setminus \{p_k\}, p^*)$. Property (a) follows from the assumption that $(x^j, \mathcal{S}^j, p^*)$ is a V-triple. Property (b) is proved by a continuity argument: since the curve search starts at x^j such that $\left\|\overline{p}^* - \overline{x}^j\right\| \geq p_0^* - x_0^j$ and it ends at x^P such that $x_0^P \leq x_0^F$, then $\left\|\overline{p}^* - \overline{x}^P\right\| \geq p_0^* - x_0^P$ (recall that x^F corresponds to the point of smallest value $x_0 \leq x_0^j$ for which $\left\|\overline{p}^* - \overline{\Gamma}^+(x_0)\right\| \geq p_0^* - x_0$). Property (c) is maintained at any point of the curve so it holds for x^P, and property (d) also holds since at x^P we have $\alpha_i(x_0^P) \geq 0$ and $\alpha^*(x_0^P) \geq 0$. Finally, the fact that $x_0^P \leq x_0^j$ is a direct consequence of the definition of the partial step procedure. □

Lemma 4.31. *Let* $(\mathcal{S}^j, x^j)$ *be a dual feasible S-pair and* $p^* \in \mathcal{P}$ *be a point corresponding to an infeasible constraint at* x^j. *Suppose* $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ *is affinely dependent and let* $p_k \in \mathcal{S}^j$ *be the point whose index was returned by Algorithm 4.3. Then,* $(x^j, \mathcal{S}^j \setminus \{p_k\}, p^*)$ *is a V-triple.*

Proof. First note that the affine dependence assumption implies that $|\mathcal{S}^j| > 1$. To prove that $(x^j, \mathcal{S}^j \setminus \{p_k\}, p^*)$ is a V-triple, we need to prove the four properties from Definition 4.27.

Property (b) is trivial, and property (c) follows directly from $(\mathcal{S}^j, x^j)$ being a dual feasible S-pair. Now, in order to prove (a), suppose, by contradiction, that $\overline{\mathcal{S}}^j \setminus \{\overline{p}_k\} \cup \{\overline{p}^*\}$ is affinely dependent. Since $\overline{\mathcal{S}}^j \setminus \{\overline{p}_k\}$ is affinely independent, there exists β_i, $i \neq k$, such that

$$\overline{p}^* = \sum_{\substack{i=1 \\ i\neq k}}^{s} \beta_i \overline{p}_i, \quad \text{with} \quad \sum_{\substack{i=1 \\ i\neq k}}^{s} \beta_i = 1.$$

On the other hand, the affine dependence of $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ causes $z = \overline{p}^* - \overline{p}_1 + Mw = 0$. Thus, we can write $\overline{p}^*$ as:

$$\overline{p}^* = -\sum_{i=1}^{s} \sigma_i \overline{p}_i,$$

for σ_i, $i = 1, ..., s$, defined in Algorithm 4.3. Then,

$$\sum_{\substack{i=1 \\ i\neq k}}^{s} \beta_i \overline{p}_i = -\sum_{i=1}^{s} \sigma_i \overline{p}_i \quad \rightarrow \quad \overline{p}_k = \sum_{\substack{i=1 \\ i\neq k}}^{s} \frac{\beta_i + \sigma_i}{-\sigma_k} \overline{p}_i, \quad \text{with} \sum_{\substack{i=1 \\ i\neq k}}^{s} \frac{\beta_i + \sigma_i}{-\sigma_k} = 1.$$

Recall from (4.25) that $\sigma_k < 0$. We conclude that, $\overline{p}_k$ is an affine combination of $\overline{\mathcal{S}}^j \setminus \{\overline{p}_k\}$, which is a contradiction.

Now consider ρ as defined in Algorithm 4.3. Let $\delta = \rho - \frac{\rho_k}{\sigma_k}\sigma$ and $\delta^* = \frac{\rho_k}{\sigma_k}$. It is easy to see that $\delta \geq 0$ and, in particular, $\delta_k = 0$. Since $z = 0$, we have

$$\begin{aligned} M\delta_{2:s} + \delta^*(\overline{p}^* - \overline{p}_1) + \overline{p}_1 &= M\left(u + x_0 v - \frac{\rho_k}{\sigma_k} w\right) - \frac{\rho_k}{\sigma_k}(\overline{p}^* - \overline{p}_1) + \overline{p}_1 \\ &= M(u + x_0 v) + \overline{p}_1 - \frac{\rho_k}{\sigma_k}(\overline{p}^* - \overline{p}_1 + Mw) \\ &= M(u + x_0 v) + \overline{p}_1 - \frac{\rho_k}{\sigma_k} z \\ &= \overline{x}^j, \end{aligned}$$

which proves that property (d) holds. □

Recall that the algorithm is initialized with a dual feasible S-pair. From the previous lemmas, we conclude that we always have a V-triple before a curve search, and that at the beginning of a new iteration, when an infeasible point p^* is selected, we always have a

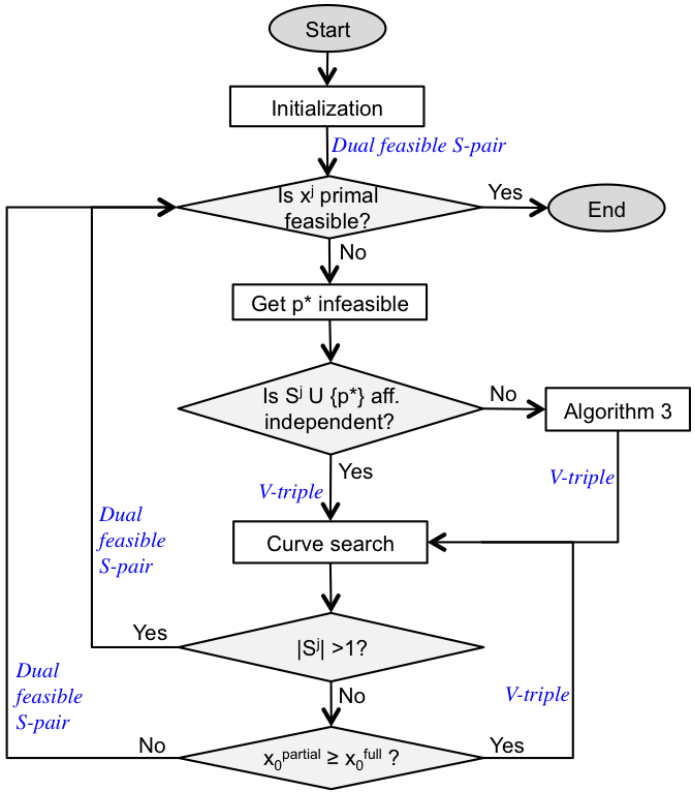


Figure 4.11: Flowchart of the algorithm.

dual feasible S-pair. The diagram from Figure 4.11 clarifies these statements statements. The main consequence of the lemmas is that, starting from a V-triple $(x^j, \mathcal{S}^j, p^*)$ for which $\left\|\overline{p}^* - \overline{x}^j\right\| > (p_0^* - x_0^j)$, one can obtain a new dual feasible S-pair, $(\mathcal{S}^{j+1}, x^{j+1})$, in at most $|\mathcal{S}^j| - |\mathcal{S}^{j+1}| \leq n$ partial steps and one full step. This new feasible S-pair is such that $\mathcal{S}^{j+1} \subseteq \mathcal{S}^j$, and $x_0^{j+1} < x_0^j$, since, even though the value of x_0 may be maintained when taking a partial step or a full step, we know that the value of x_0 must decrease either in one of the partial steps or in the full step. That is because, otherwise, we would have $x^{j+1} = x^j$, contradicting the fact that p^* is infeasible at x^j. Therefore, since the value of x_0 strictly decreases at each loop iteration, the same dual feasible S-pair can never reoccur. Since the number of dual feasible S-pairs is finite, we conclude the main result of this section:

Theorem 4.32. *The proposed dual algorithm solves problem* $\mathrm{Inf}_Q(\mathcal{P})$ *in a finite number of iterations.*

4.3.5 A note on degeneracy

As mentioned previously, as far as degeneracy is concerned, no special procedure is required to prevent *cycling*, since, at each major iteration, the value of the objective function strictly improves, so a basis cannot occur twice.

However, in a primal algorithm cycling could occur. As a consequence of the lack of strict complementarity, the current primal feasible solution may be active for constraints corresponding to points other than the ones on the current basis. That is, there may be multiple dual solutions that correspond to the same primal feasible solution. Thus, it can happen that, after a sequence of adding/dropping points from the basis, the algorithm does not move in the primal space and ends up in a basis that had been previously visited. That is due to the fact that a primal algorithm cannot choose which point to enter the basis: it has to be one corresponding to a primal constraint that becomes active. Instead, in the dual algorithm, the equivalent situation of when a movement is not possible because a dual variable is zero at the current iterate is easily dealt by removing the point corresponding to that dual variable from $\mathcal{S}^j$ (either at a partial step or when $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely dependent). This is done as many times as the number of dual variables that are zero, after which a movement will then be possible in the next iteration.

4.4 Implementation details

The main computational work that is required by the algorithm happens before each curve search when three linear systems need to be solved to get vectors u, v, and w:

$$
\begin{aligned}
(M^T M)u &= b - M^T \overline{p}_{j_1} \\
(M^T M)v &= c \\
(M^T M)w &= -M(\overline{p}^* - \overline{p}_{j_1})
\end{aligned}
$$

Our implementation is based on the Cholesky factorization of matrix $M^T M$.

Let $\mathcal{S}^j = \{p_{j_1}, p_{j_2}, ..., p_{j_s}\}$ be the current basis at iteration j, and let $|\mathcal{S}^j| = s$. Recall that

$\mathcal{S}^j$ may change throughout the iteration but $\overline{\mathcal{S}}^j$ is always kept affinely independent. Thus, matrix $M \in \mathbb{R}^{(n-1)\times(s-1)}$, given by

$$M = \begin{bmatrix} \overline{p}_{j_2} - \overline{p}_{j_1} & \overline{p}_{j_3} - \overline{p}_{j_1} & \cdots & \overline{p}_{j_s} - \overline{p}_{j_1} \end{bmatrix}$$

is always full column rank, and M^TM symmetric positive definite. Let LL^T be the Cholesky factorization of M^TM, with $L \in \mathbb{R}^{(s-1)\times(s-1)}$ lower triangular. Recall that the Cholesky factor L is unique since M^TM is symmetric, e.g. [52, §4]. Knowing the Cholesky factorization of M^TM allows the linear systems on u, v, and w, to, each, be reduced to a linear system with an upper triangular matrix and another with a lower triangular matrix. These linear systems can be solved using *back* and *forward substitution*, respectively, requiring $\mathcal{O}(s^2)$ flops each [52, §3.1]. Together with the matrix-vector products, finding u, v, and w requires $\mathcal{O}(ns + s^2)$ basic operations.

4.4.1 Updating the Cholesky factorization

We now show how the Cholesky factorization M^TM can be updated whenever $\mathcal{S}^j$ changes, using $\mathcal{O}(s^2)$ basic operations and in a numerically stable way. Note that a Cholesky factorization is never calculated from scratch at any time of the algorithm unless a dual feasible S-pair with more than one point is given as input, in which case a Cholesky factorization is calculated only once during the initialization phase.

A point is added to $\mathcal{S}^j$

After a full step is taken, point p^* is added to $\mathcal{S}^j$ to get $\mathcal{S}^{j+1}$. That corresponds to appending column $\overline{p}^* - \overline{p}_{j_1}$ to matrix M, creating $\hat{M}$. Since

$$\hat{M}^T\hat{M} = \left[\begin{array}{c|c} M^TM & M^T(\overline{p}^* - \overline{p}_{j_1}) \\ \hline (\overline{p}^* - \overline{p}_{j_1})^TM & (\overline{p}^* - \overline{p}_{j_1})^T(\overline{p}^* - \overline{p}_{j_1}) \end{array}\right],$$

the Cholesky factor, $\hat{L}$, of $\hat{M}^T\hat{M}$ is such that

$$\hat{L} = \left[\begin{array}{c|c} L & 0 \\ \hline a^T & \sqrt{(\overline{p}^* - \overline{p}_{j_1})^T z} \end{array}\right],$$

for $a \in \mathbb{R}^{s-1}$ such that $La = M^T(\overline{p}^* - \overline{p}_{j_1})$ and $z = (I - MM^+)(\overline{p}^* - \overline{p}_{j_1})$ (z is already available since it is calculated during the curve search). In conclusion, when a new point is added to $\mathcal{S}^j$, updating the Cholesky factor of M^TM boils down to the solution of a lower triangular linear system involving a $(s-1) \times (s-1)$ matrix.

A point is removed from $\mathcal{S}^j$

Whenever a partial step is taken or $\overline{\mathcal{S}}^j \cup \{\overline{p}^*\}$ is affinely dependent, a point p_{j_k} from $\mathcal{S}^j$ is removed. That corresponds to removing a column from M. Let $\hat{M}$ denote the matrix corresponding to $\mathcal{S}^j \setminus \{p_{j_k}\}$ and $\hat{L}$ the Cholesky factor of $\hat{M}^T\hat{M}$.

Case 1: p_{j_1} *is the point to be removed.* This is the trickiest case, as it involves a rank-2 update of matrix M^TM. Define $\hat{M}$, the matrix corresponding to $\mathcal{S}^j \setminus \{p_{j_1}\}$, as:

$$\hat{M} = \left[\begin{array}{cccc} \overline{p}_{j_2} - \overline{p}_{j_s} & \overline{p}_{j_3} - \overline{p}_{j_s} & \cdots & \overline{p}_{j_{s-1}} - \overline{p}_{j_s} \end{array}\right].$$

Basically, we now think of the points of $\mathcal{S}^j$ ordered as $\{p_{j_s}, p_{j_2}, ..., p_{j_{s-1}}\}$. To obtain the Cholesky factor $\hat{L}$ of $\hat{M}^T\hat{M}$, first note that,

$$N := M + (\overline{p}_{j_s} - \overline{p}_{j_1})1_{s-1}^T = \left[\begin{array}{c|c} \hat{M} & 0 \end{array}\right],$$

with $1_{s-1} \in \mathbb{R}^{s-1}$ a vector with entries all 1. Let L_N be the Cholesky factor of N^TN. It is easy to see that

$$L_N = \left[\begin{array}{c|c} \hat{L} & 0 \\ \hline 0 & 0 \end{array}\right].$$

Now we explain how to calculate L_N. Consider $I_{s-1} \in \mathbb{R}^{(s-1)\times(s-1)}$ the identity matrix, and

$e_{s-1} \in \mathbb{R}^{s-1}$ the $(s-1)$-th column of I_{s-1}. We have that

$$\begin{aligned} N^T N &= (M - Me_{s-1}1_{s-1}^T)^T(M - Me_{s-1}1_{s-1}^T) \\ &= (I_{s-1} - e_{s-1}1_{s-1}^T)^T M^T M(I_{s-1} - e_{s-1}1_{s-1}^T). \end{aligned} \tag{4.26}$$

In [50], it is shown how one can solve the following problem: given the Cholesky factor L_A of the positive definite matrix $A \in \mathbb{R}^{n\times n}$ and given vectors $x, y \in \mathbb{R}^n$, find the Cholesky factor L_B of matrix

$$B = (I + yx^T)^T A(I + yx^T) \in \mathbb{R}^{n\times n}. \tag{4.27}$$

The method solves the problem in a numerically stable way using $\mathcal{O}(n^2)$ operations. It consists on finding a lower triangular matrix $\tilde{L}$ for which there exists an orthonormal matrix Q that satisfies $(I_d + zw^T)Q = \tilde{L}$, with vectors z and w given by $Lz = v$ and $w = L^T u$. It is easy to see that $B = L_A\tilde{L}Q^TQ\tilde{L}^TL_A^T$, and so $L_B = L_A\tilde{L}$. The author proposes two different methods to calculate $\tilde{L}$, one using Givens' plane rotations and the other Householder transformations. Since (4.26) has the same form as (4.27), we can use one of the methods mentioned above to calculate L_N, which yields $\hat{L}$ after removing its last row and column. The total amount of computational work to calculate L_N is $\mathcal{O}(s^2)$.

Case 2: p_{j_k}, $k = 2, ..., s-1$, is the point to be removed.

In this case $\hat{M}^T\hat{M}$ is obtained from M^TM by deleting the $(k-1)$-th row and column. Updating the Cholesky factor in this case is rather simple. Let

$$L = \left[\begin{array}{c|c|c} L_{11} & 0 & 0 \\ \hline l_{k-1,1}^T & l_{k-1,k-1} & 0 \\ \hline L_{13} & l_{3,k-1} & L_{33} \end{array}\right]$$

where $l_{k-1,k-1}$ is the $(k-1, k-1)$-entry of L. It is easy to see that $\hat{L}$ is such that

$$\hat{L} = \left[\begin{array}{c|c} L_{11} & 0 \\ \hline L_{13} & \hat{L}_{33} \end{array}\right]$$

where $\hat{L}_{33} \in \mathbb{R}^{(s-k)\times(s-k)}$ is the Cholesky factor of matrix $L_{33}L_{33}^T + l_{3,k-1}l_{3,k-1}^T$. This corresponds to the *updating problem* of a Cholesky factorization and can be accomplished in $\mathcal{O}((s-k)^2)$ computational time. There are several methods to do so [48], in particular this problem is analogous to updating the QR factorization after appending a row [97, pp. 340]. In our implementation we use the `cholupdate` function Matlab provides for the *updating* and *downdating* problems.

Case 3: p_{j_s} is the point to be removed.

In this case, the last column of M needs to be deleted. That corresponds to deleting the last row and last column of M^TM. Thus, $\hat{L}$ can be obtained from L by simply deleting the last row and the last column of L.

4.4.2 Iteration complexity

Each iteration of the algorithm, that is, each time an infeasible point p^* is selected to enter the basis, consists on the following work:

- the optimality check ($\mathcal{O}(mn)$ computational work);
- at most $\hat{s}$ curve searches, where $\hat{s}$ is the size of the basis corresponding to the dual feasible S-pair at the beginning of an iteration.

From the analysis made in the previous section, it is possible to conclude that each curve search has a computational complexity of $\mathcal{O}(ns + s^2)$, where $s = |\mathcal{S}^j|$ at the beginning of the curve search. Within an iteration, every time a curve search is performed a point is removed from the set $\mathcal{S}^j$, and at most $\hat{s}$ curve searches can be performed. Therefore, we conclude that each iteration of the algorithm has complexity $\mathcal{O}(mn+n\hat{s}^2+\hat{s}^3)$. In the worst case scenario, $\hat{s} = n$ in which case the iteration has complexity $\mathcal{O}(mn + n^3)$. However, as we will see in Section 4.5, in practice, we often observe that the size of $\mathcal{S}^j$ is much smaller than n.

Finally, we stress that there is no polynomial number of iterations guarantee.

4.4.3 Discussion of other options

The nature of the matrix $M^T M$ makes the usage of the Cholesky factorization a natural choice. However, other factorizations that would yield an analogous computational work could also be considered. One possibility would be to keep the QR factorization of $M \in \mathbb{R}^{(n-1)\times(s-1)}$, that is $M = QR$, such that $Q \in \mathbb{R}^{(s-1)\times(s-1)}$ is orthogonal and $R \in \mathbb{R}^{(n-1)\times(s-1)}$ upper triangular. Updating such QR factorization, however, is done in $\mathcal{O}(n^2 + ns)$ computational time, which becomes much slower than updating the Cholesky factorization, specially when $s \ll n$. In particular, when a point p^* is added to the basis, meaning a new column c is appended to M, we need $n - s - 2$ Givens' rotation matrices to zero the $n - s - 2$ non-zero entries of the new column $Q^T c$ of R.

Another option would be to keep the QR factorization of $M^T M$, which can be updated in $\mathcal{O}(s^2)$ time, similarly to the Cholesky factorization. An immediate advantage of this option over the Cholesky factorization is that only three triangular systems are required to be solved to calculate vectors u, v, and w, instead of six. However, maintaining the factorization whenever a point is added/removed from $\mathcal{S}^j$ requires a larger constant number operations. For instance, every time a point is deleted, that corresponds to deleting a row and a column from $M^T M$, so the factorization needs two updates.

With the goal of seeing in practice the impact of the above observations, besides implementing the algorithm using the Cholesky factorization of $M^T M$, we have also created an implementation using the QR factorization of M and another using the QR factorization of $M^T M$. We ran a few tests to compare them, and the running times are presented in Tables 4.1 and 4.2. The average of the maximum sizes of a basis observed in each run is also reported. Our experiments were conducted using MATLAB R2018b (version 9.5.0) on a Mac with an Intel Core i5 1.6 GHz processor, with 8GB RAM, running Mac OS X version 10.11.6.

We observe that the implementation using the Cholesky factorization is consistently faster than the other two studied alternatives. As observed above, when the maximum size of a basis is much smaller than the dimension, the implementation with $M = QR$ is

	n	100	500	1000	1000	2500	5000
	m	1000	1000	1000	5000	5000	10000
	Max size basis	19.4	56.2	71.8	105.3	52.9	81
Running time (secs.)	$M^TM = LL^T$	0.01	0.16	0.41	1.29	0.28	0.80
	$M^TM = QR$	0.02	0.18	0.46	1.44	0.31	0.92
	$M = QR$	0.04	1.26	1.71	10.51	0.67	2.34

Table 4.1: Average running times (in seconds) corresponding to 10 runs of Algorithm 4.4 implemented with a different matrix factorization. The input points are standard normally distributed, with each coordinate chosen independently. The average maximum size of a basis observed during the algorithm runs is also reported.

	n	100	500	1000	1000	2500	5000
	m	1000	1000	1000	5000	5000	10000
	Max size basis	97.1	597.1	924.0	1943.9	494.2	966.4
Running time (secs.)	$M^TM = LL^T$	0.29	7.83	48.38	235.92	19.55	86.20
	$M^TM = QR$	0.63	15.34	100.47	494.54	42.92	177.32
	$M = QR$	0.43	25.99	111.14	935.30	35.19	183.28

Table 4.2: Average running times (in seconds) corresponding to 10 runs of Algorithm 4.4 implemented with a different matrix factorization. The input points are uniformly distributed within the set $\{x \in \mathbb{R}^n : \|x\| \leq x_0 \ \wedge \ x_0 \leq 1\}$ (a portion of a second-order cone). The average maximum size of a basis observed during the algorithm runs is also reported.

slower than the other two. However, when the bases sizes are closer to n, (Table 4.2, e.g. $m = 10000$) we can see that the running times of $M = QR$ and $M^TM = QR$ become closer.

In Table 4.1, we can see that, when the bases sizes are much smaller than n, the implementation that uses $M^TM = QR$, though slightly slower, has comparable running times to the implementation with the Cholesky factorization. When the bases sizes are larger, since the algorithm starts with a single point on the basis and adds a point at each iteration, the number of iterations is at least as large as the maximum size of a basis. It is in this case, that we see the effect of the extra work required by the implementation with $M^TM = QR$ resulting in running times about twice as large as the implementation using the Cholesky factorization.

4.5 Computational results

Our computational experiments were conducted using MATLAB R2018b (version 9.5.0) on a Mac with an Intel Core i5 1.6 GHz processor, with 8GB RAM, running Mac OS X version 10.11.6. All of our experiments were conducted on randomly generated data sets, according to various distributions. Specifically, we considered the following three classes of point datasets:

- normally distributed, with each coordinate chosen independently according to a normal distribution with mean 0 and standard deviation 1;
- uniformly distributed within a unit cube;
- uniformly distributed within a within the set $\{x \in \mathbb{R}^n : \|\bar{x}\| \leq x_0 \wedge x_0 \leq 1\}$, that is, a portion of a second-order cone (s.o.c.).

We have also considered datasets with points uniformly distributed within a hypersphere with radius 1. However, since the studied performance parameters corresponding to this dataset, such as time and number of iterations, were comparable in magnitude to the ones of the uniformly distributed within a unit cube and normally distributed, we decided to omit this dataset for succinctness.

There are several ways to choose which point p^*, corresponding to an infeasible constraint, enters the basis at each iteration. In our experiments, the point p^* is chosen to be the one corresponding to the largest infeasibility gap at the current iterate x^j, that is,

$$p^* = \arg\max_{p_i \in \mathcal{P}} \left\{ \left\| \bar{p}_i - \bar{x}^j \right\| - \left(p_{i0} - x_0^j \right) \right\}.$$

This approach showed to result in fewer iterations and shorter running times over other options such as the first infeasible point found. Unless mentioned otherwise, in our experiments, we consider a feasibility tolerance of 10^{-8}.

In all our experiments, the algorithm was initialized with a dual feasible S-pair containing a single point from the input dataset.

4.5.1 Performance of the algorithm

We begin by showing how the algorithm performs for different datasets. Figures 4.12 and 4.13 show the number of basis updates - number of times a point is added or removed from the basis, that is, number of dual feasible S-pair updates, and the maximum size of a basis, for datasets with points normally distributed and uniformly distributed in a cube and variable dimension n. Figure 4.12 shows the results for 1000-point datasets, and Figure 4.13 shows the results for $10,000$-point datasets and $100,000$-point datasets side by side. From these two figures we can observe that, for the datasets considered, the number of points on the optimal bases is much smaller than the dimension (the maximum possible size of a basis). An exception is the datasets with $n = 10$, which is to be expected since $n \ll m$. Consider, for instance, the 1000-point normal datasets in Table 4.12: for $n = 100$, the optimal basis size is about $\frac{1}{5}n$, and for $n = 1000$ it is about $\frac{6}{100}n$. This phenomenon does not seem to change considerably when the number of input points increases. For instance, considering $n = 1000$ and the normal datasets, we can see that the average number of points on the basis only slightly increases with the number of input points, being observed to be always between 50 and 100 for the values of m studied.

The number of iterations - whenever a point is selected to enter the basis, is always bounded below by the optimal basis size, since the algorithm is initialized with a basis having a single point. Additionally, it is bounded above by the number of basis updates, since, in an iteration, there is at least one curve search, that is, a basis update. Figures 4.12 and 4.13 show that, for the normally distributed and uniformly distributed on a cube datasets, the number of basis updates is just slightly larger than the size of the optimal bases (we observe that the difference is usually no more than 5). This means that most of the time when a point is selected to enter the basis, it will end up being in the optimal basis returned by the algorithm. As a consequence, for each instance, the number of iterations is only slightly larger than the lower bound for that instance. This behavior does not seem to change when the number of points increases, as we see when we compare the data for $m = 10,000$ with $m = 100,000$ in Figure 4.13.

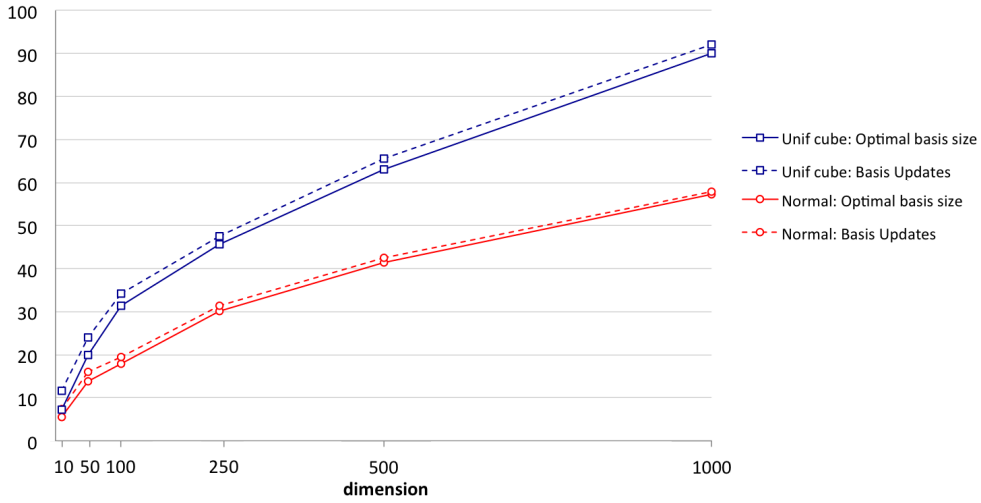


Figure 4.12: Number of points on the optimal basis and number of basis updates as a function of the dimension, for 1000-point standard normally distributed and uniformly distributed un a unit cube datasets. The reported numbers correspond to the averages over 10 runs.

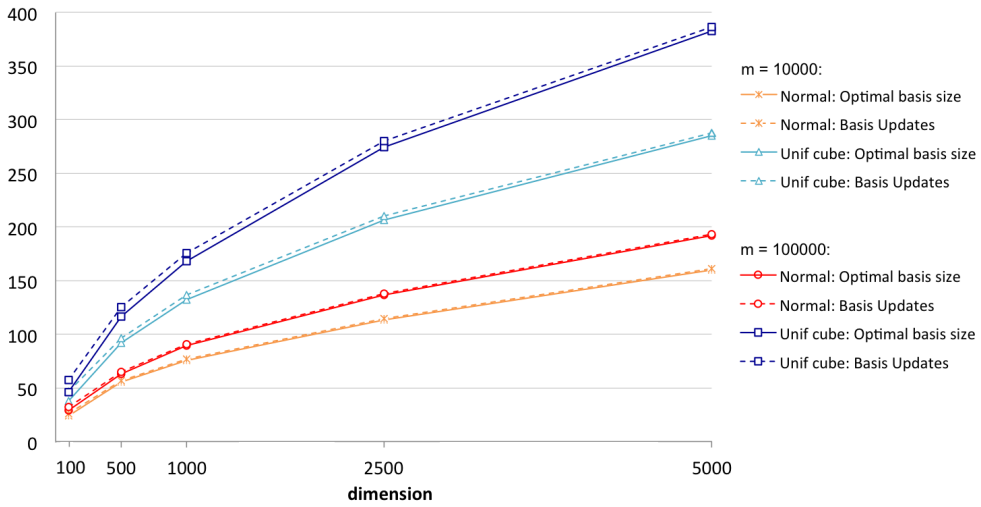


Figure 4.13: Number of points on the optimal basis and number of basis updates as a function of the dimension, for $10,000$-point and $100,000$-point standard normally distributed and uniformly distributed on a unit cube datasets. The reported numbers correspond to the averages over 10 runs.

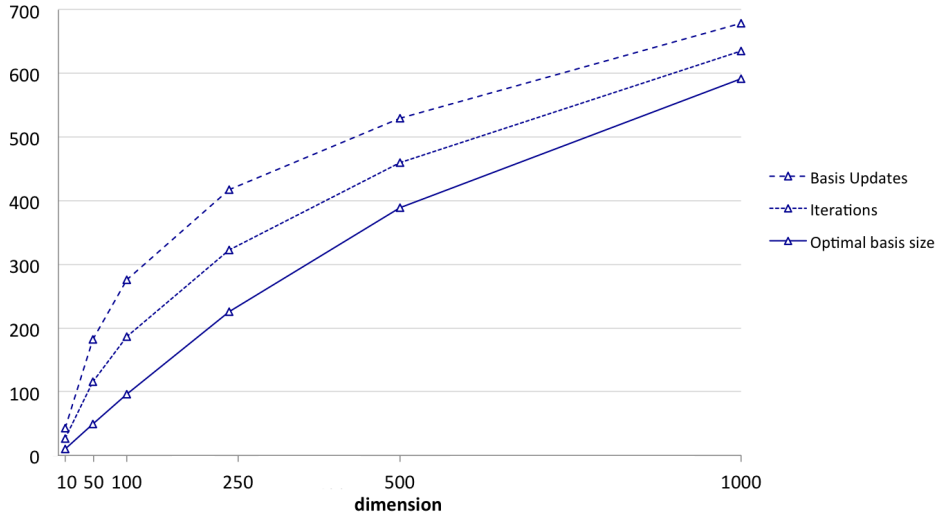


Figure 4.14: Number of points on the optimal basis, number of basis updates, and number of iterations as a function of the dimension, for 1000-point uniformly distributed on a portion of a s.o.c. datasets. The reported numbers correspond to the averages over 10 runs.

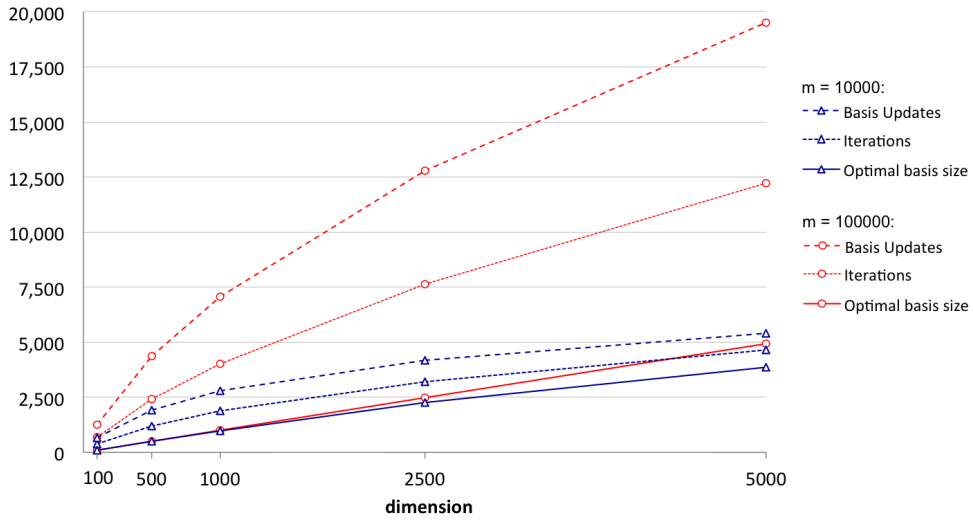


Figure 4.15: Number of points on the optimal basis, number of basis updates, and number of iterations as a function of the dimension, for $10,000$-point and $100,000$-point uniformly distributed on a portion of a s.o.c. datasets. The reported numbers correspond to the averages over to 10 runs.

Iter j	Max Infeas Gap	Infeas Constr Count	$\|x^j - x^{j-1}\|$	$\|\mathcal{S}\|^j$
1	2.54781806	99999	1.80157942	2
2	0.90380296	30592	0.75659132	2
3	0.68942045	11868	0.59988039	3
4	0.23478991	462	0.22469704	4
5	0.09796021	89	0.09835393	5
6	0.10740123	61	0.13783676	6
7	0.09703347	28	0.13499837	6
8	0.12438402	30	0.14568556	6
9	0.05559249	8	0.06385760	7
10	0.03883349	11	0.06015916	8
11	0.03273477	6	0.04292090	9
12	0.03867943	4	0.06186058	8
13	0.00539842	2	0.01093005	9
14	0.00000000	0	0.00000000	9

Table 4.3: Iteration log for a $100,000$-point dataset with $n = 10$ and points uniformly distributed on a unit cube. "Max Infeas Gap" is the maximum infeasibility gap at x^j, and "Infeas Constr Count" is the number of infeasible constraints at iteration j (considering an infeasibility tolerance of 10^{-8}).

Iter j	Max Infeas Gap	Infeas Constr Count	$\left\|x^j - x^{j-1}\right\|$	$\|\mathcal{S}\|^j$
1	1.79518774	99999	1.26938943	2
2	0.13388237	41567	0.12610273	3
3	0.04240208	26781	0.04162795	4
4	0.02192067	26127	0.04519436	5
5	0.04996865	25121	0.06077760	5
6	0.04628315	18728	0.04790556	6
7	0.01817180	10542	0.01896393	7
8	0.01022663	7951	0.02670843	8
9	0.02520613	11957	0.02658685	8
10	0.01203876	7067	0.01332468	9
11	0.00428870	3866	0.01531832	9
12	0.01337036	7058	0.01460064	10
13	0.00082727	1494	0.00206432	10
14	0.00190522	1947	0.00292080	10
15	0.00179799	1832	0.00257150	10
16	0.00153712	1621	0.00180644	10
17	0.00079142	931	0.00095881	10
18	0.00014300	446	0.00021238	10
19	0.00019104	317	0.00031522	10
20	0.00017591	301	0.00020607	10
21	0.00006997	201	0.00008872	10
22	0.00006014	164	0.00007220	10
23	0.00006983	158	0.00012879	10
24	0.00007081	137	0.00013906	10
25	0.00005353	96	0.00007079	10
26	0.00002374	64	0.00004787	10
27	0.00002558	55	0.00003901	10
28	0.00002422	47	0.00004088	10
29	0.00001287	32	0.00003600	10
30	0.00002371	37	0.00002757	10
31	0.00000688	22	0.00002178	10
32	0.00001344	25	0.00002155	10
33	0.00000391	13	0.00002252	10
34	0.00000937	24	0.00001859	10
35	0.00000378	9	0.00001058	10
36	0.00000336	11	0.00000600	10
37	0.00000553	12	0.00000985	10
38	0.00000217	8	0.00001028	10
39	0.00000528	13	0.00000743	10
40	0.00000156	5	0.00000409	10
41	0.00000125	4	0.00000314	10
42	0.00000046	2	0.00000095	10
43	0.00000026	1	0.00000047	10
44	0.00000000	0	0.00000000	10

Table 4.4: Iteration log for a $100,000$-point dataset with $n = 10$ and points uniformly distributed on a portion of a s.o.c. "Max Infeas Gap" is the maximum infeasibility gap at x^j, and "Infeas Constr Count" is the number of infeasible constraints at iteration j (considering an infeasibility tolerance of 10^{-8}).

		n					
m	*Dataset*	10	50	100	250	500	1000
1000	Normal	0.005	0.01	0.02	0.04	0.10	0.16
	Uniform cube	0.01	0.02	0.03	0.07	0.10	0.23
	Uniform cone	0.05	0.18	0.34	1.35	3.15	7.78

Table 4.5: Running time (in seconds) of the algorithm as a function of the dimension n, for different 1000-point datasets: standard normally distributed, uniformly distributed on a unit cube, and uniformly distributed on a portion of a s.o.c. The reported numbers correspond to the averages corresponding to 10 runs of the algorithm for each dataset type.

		n				
m	*Dataset*	100	500	1000	2500	5000
$10,000$	Normal	0.04	0.28	0.78	2.51	7.07
	Uniform cube	0.07	0.52	1.27	4.84	13.94
	Uniform cone	1.62	21.66	85.56	501.50	2117.70
$100,000$	Normal	0.30	2.59	10.16	46.89	232.40
	Uniform cube	0.53	4.43	12.07	58.47	287.95
	Uniform cone	6.27	116.24	494.86	3302.13	20015.40

Table 4.6: Running time (in seconds) of the algorithm as a function of the dimension n, for different $10,000$-point and $100,000$-point datasets: standard normally distributed, uniformly distributed on a unit cube, and uniformly distributed on a portion of a s.o.c. The reported numbers correspond to the averages corresponding to 10 runs of the algorithm for each dataset type.

Now, we turn our attention to datasets that cause the algorithm's worst behavior: datasets shaped like a second-order cone. Figures 4.14 and 4.15 show the computational results corresponding to the datasets with points uniformly distributed on a portion of a second-order cone. Like before, we consider 1000, $10,000$, and $100,000$ input points. These figures show the number of basis updates, iterations, and size of the optimal bases, for different dimensions n. The first thing we notice is that the sizes of the optimal bases are larger than what we observe for other datasets, being often very close to or equal to n. This fact is not surprising when we think of the geometric interpretation of the Inf_Q problem as moving a second-order cone to enclose all the input points. So, it is expected that a large number of points of the input set end up on the boundary of the cone translated to the optimal solution, and are part of an optimal basis. We also observe that the number of basis updates is larger than the number of iterations, meaning that, often some points are dropped from the current basis during the curve search.

The larger sizes of the bases are only one of the factors that contribute to the larger number of iterations. The other has to do with the fact that, when the input is shaped like a cone, and especially when the points are distributed uniformly, when x^j moves on the curve to enclose p^* at each iteration, several points that were feasible often lose their feasibility. Tables 4.3 and 4.4 show the iteration logs outputted by the algorithm for a 10-dimensional dataset with $100,000$ points distributed uniformly on a unit cube and distributed uniformly on a portion of a s.o.c., respectively. In these tables "Max Infeas Gap" is the maximum infeasibility gap at x^j, and "Infeas Constr Count" is the number of infeasible constraints at iteration j (considering an infeasibility tolerance of 10^{-8}). We can observe how the maximum infeasibility gap decreases more slowly for the dataset with points uniformly drawn from a s.o.c. than for the dataset with points uniformly drawn from a unit cube. We can also observe in Table 4.4 that there are many iterations where the number of infeasible constraints increases considerably, and we can also see how x^j suffers small changes in the last iterations in order to get all constraints feasible. Recall that we are using a feasibility tolerance of 10^{-8}. Increasing this tolerance can result in a significant decrease of the number of iterations for the datasets with points uniformly drawn from a portion of a s.o.c. For instance, for the dataset corresponding to the log shown on Table 4.4, if we had chosen an

infeasibility tolerance of 10^{-5} instead of 10^{-8}, optimality would have been declared after 31 iterations, instead of 44.

The running times corresponding to the experiments reported in Figures 4.12 to 4.15 are reported in Tables 4.5 and 4.6. We observe that the running times corresponding to the normal and uniform on a cube datasets are quite small. For 1000-point datasets, the average running times are consistently under 1 second; under 15 seconds for $m = 10,000$; and under 300 seconds (5 minutes) for $m = 100,000$. On the other hand, the datasets with points uniformly distributed on a s.o.c. are the ones where the algorithm shows the poorer computational times, as expected.

When we look at the results of Figure 4.13 corresponding to the uniformly distributed datasets with $n = 5000$, we see that the average number of basis updates is close to 300 for $m = 10,000$, and close to 400 for $m = 100,000$. So, we expect that the amount of time spent on the curve searches on these cases does not differ by a large amount. However, the average running times are very different: for $m = 10,000$ we have 14 seconds, and for $m = 100,000$ we have 288. This is a reflection of the fact that the largest contributor to the running times of the algorithm is the feasibility check performed at the beginning of each iteration. It is possible to skip part of the required queries in this phase by employing some distance filtering techniques, using, for instance, the triangle inequality or Cauchy-Schwarz. See [62] for a discussion on this.

4.5.2 Comparison with other methods

For comparison purposes we chose two SOCP solvers based on interior-point-methods: MOSEK version 8.1.0 [77], and SDPT3 version 4.0 [101]. We used their Matlab interfaces to solve the dual problem (4.4) since its size is smaller than the primal problem (4.1). All MOSEK and SDPT3 parameters were kept at their default values. We also tried to use Gurobi (version 8.0.0) [53], but experimentation showed that solving the dual problem would often cause Matlab to freeze, and solving the primal would result in running out of memory, even for small instances.

Tables 4.7, 4.8, and 4.9 show the average running times in seconds corresponding to our dual simplex-like algorithm, MOSEK, and SDPT3 for different datasets. Table 4.7 reports the times corresponding to datasets with points normally distributed, Table 4.8 to datasets with points uniformly distributed on a unit cube, and Table 4.9 to datasets with points uniformly distributed on a portion of a second-order cone.

In general, these tables show that our algorithm performs very well when compared to MOSEK and SDPT3. From Table 4.7, we can see that, for $\mathfrak{m} = 1000$, our algorithm is between 15 to 50 times faster than MOSEK, and between 50 to 120 times faster than SDPT3. These rates considerably increase for $\mathfrak{m} = 10,000$, where, for instance, for $\mathfrak{n} = 5000$, our algorithm is about 850 times faster than MOSEK and about 500 times faster than SDPT3. Similar results are observed in Table 4.8 for datasets with points uniformly distributed on a unit cube. For instance, for $\mathfrak{m} = 10,000$, our algorithm is between 30 and 390 times faster than MOSEK, and between 100 and 380 times faster than SDPT3. As mentioned before, our algorithm shows its worst performance for the datasets with points uniformly distributed on a portion of a s.o.c.. For these datasets, in Table 4.9, we can see an approximation of the algorithm's running times to the running times of MOSEK and SDPT3, though most of the time our algorithm is about 2-times faster.

We omitted the number of interior-point iterations of both MOSEK and SDPT3, as they are not relevant for our analysis. However, for completeness, we want to mention that the number of iterations was somewhat independent on the point distribution, $\mathfrak{d}$, and $\mathfrak{m}$. In our experiments, MOSEK usually took between 15 and 20 iterations, and SDPT3 usually took between 20 and 25 iterations.

It is not surprising that our algorithm, built to solve a specific problem, has better running times than general-purpose solvers. Although the performance of MOSEK or SDPT3 may be improvable by adjusting their respective parameters, such a task is typically time consuming and requires an in-depth understanding of the underlying methods.

Finally, we did not have the chance of comparing our algorithm with the ones of Kumar et al. [65] and Zhou et al. [110]. The former is available online to the public, but we did not manage to run it due to an error returned by Matlab, and the latter was not available.

$\mathfrak{n}$	$\mathfrak{m}$	Alg. 4.4	MOSEK	SDPT3
10	1000	0.00	0.08	0.28
50	1000	0.01	0.28	0.70
100	1000	0.02	0.56	1.53
250	1000	0.04	1.43	4.16
500	1000	0.10	3.40	10.09
1000	1000	0.16	8.19	19.64
10	10,000	0.01	0.74	2.12
100	10,000	0.04	8.23	24.77
500	10,000	0.28	44.16	145.10
1000	10,000	0.78	93.93	312.61
2500	10,000	2.51	430.62	1134.25
5000	10,000	7.07	6082.79	3675.33

Table 4.7: Running time (in seconds) of Algorithm 4.4, MOSEK and SDPT3 solvers, for datasets with points standard normally distributed, with different dimension $\mathfrak{n}$ and number of points $\mathfrak{m}$. The reported numbers correspond to the averages corresponding to 10 runs for each $\mathfrak{m}$ and $\mathfrak{n}$ combination.

$\mathfrak{n}$	$\mathfrak{m}$	Alg. 4.4	MOSEK	SDPT3
10	1000	0.01	0.06	0.20
50	1000	0.02	0.25	0.72
100	1000	0.03	0.55	1.33
250	1000	0.07	1.57	4.26
500	1000	0.10	3.38	9.39
1000	1000	0.23	7.27	18.44
10	10,000	0.02	0.69	1.84
100	10,000	0.07	7.52	24.82
500	10,000	0.52	43.71	146.21
1000	10,000	1.27	80.63	268.02
2500	10,000	4.84	415.26	1038.64
5000	10,000	13.94	5521.37	3855.16

Table 4.8: Running time (in seconds) of Algorithm 4.4, MOSEK and SDPT3 solvers, for datasets with points uniformly distributed in a unit cube, with different dimension $\mathfrak{n}$ and number of points $\mathfrak{m}$. The reported numbers correspond to the averages corresponding to 10 runs for each $\mathfrak{m}$ and $\mathfrak{n}$ combination.

$\mathfrak{n}$	$\mathfrak{m}$	Alg. 4.4	MOSEK	SDPT3
10	1000	0.05	0.08	0.35
50	1000	0.18	0.30	0.77
100	1000	0.34	0.57	1.36
250	1000	1.35	1.58	4.51
500	1000	3.15	3.21	9.88
1000	1000	7.78	6.70	17.05
10	10,000	0.02	0.55	1.53
100	10,000	1.62	7.95	26.36
500	10,000	21.66	47.86	149.44
1000	10,000	85.56	94.00	297.03
2500	10,000	501.50	421.46	999.27
5000	10,000	2035.77	5702.86	3390.80

Table 4.9: Running time (in seconds) of Algorithm 4.4, MOSEK and SDPT3 solvers, for datasets with points uniformly distributed in a portion of a s.o.c., with different dimension $\mathfrak{n}$ and number of points $\mathfrak{m}$. The reported numbers correspond to the averages corresponding to 10 runs for each $\mathfrak{m}$ and $\mathfrak{n}$ combination.

4.6 Conclusion

In this chapter, we proposed a dual simplex-like algorithm to solve the $\text{Inf}_{\mathbb{Q}}$ problem, and consequently, the problem of enclosing a given set of balls with a ball of smallest radius. Our method solves the problem exactly, contrarily to the latest approaches that focus on calculating approximations, such as [65, 110].

Our algorithm employs a pivoting scheme resembling the simplex method for LP. Each iteration of our algorithm starts with a dual feasible S-pair and a point corresponding to an infeasible constraint. Then, it performs a sequence of exact curve searches until it arrives at a new dual feasible S-pair with a strictly better objective function value. In each one of those curve searches, a point is removed from the basis, and, at the end of the iteration, the point corresponding to the infeasible constraint is added to the basis. When the algorithm terminates, the basis will give which at most $\mathfrak{n}$ constraints (balls) determine the solution (the minimum enclosing ball). Our algorithm does not suffer from degeneracy and is not susceptible to cycling.

The computational work of each curve search consists mostly on the solution of three linear systems with a symmetric positive definite matrix. In our implementation, we use the Cholesky factorization of such matrix, which is updated every time a point is added or removed from the basis. This yields a curve search that has a $\mathcal{O}(\mathfrak{n}\mathfrak{s} + \mathfrak{s}^2)$ computational complexity, where $\mathfrak{s} \leq \mathfrak{n}$ is the size of the current basis. Though in the worst-case scenario $\mathfrak{s} = \mathfrak{n}$, we observed that, in general, $\mathfrak{s} \ll \mathfrak{n}$.

Our experiments using Matlab show that the algorithm can efficiently handle datasets with dimensions up to 5000, and it solves instances with 100,000 points within a few minutes, with the only exception being when the dataset has a "conic shape".

Given the work developed to produce the dual simplex-like algorithm presented in this chapter, most of the ingredients necessary to develop a primal version of the algorithm are now available. It would also be interesting to investigate, how far one can generalize the formulation of the $\text{Inf}_{\mathbb{Q}}$ problem, such that a simplex-like algorithm following the same mechanics of our algorithm would be possible to be applied to.

Bibliography

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, *Geometric approximation via coresets*, in Combinatorial and Computational Geometry, MSRI, University Press, 2005, pp. 1–30.

[2] P. K. Agarwal, S. Har-peled, and H. Yu, *Robust shape fitting via peeling and grating coresets*, in Proc. 17th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA '06, Philadelphia, PA, USA, 2006, SIAM, pp. 182–191.

[3] P. K. Agarwal and R. Sharathkumar, *Streaming algorithms for extent problems in high dimensions*, Algorithmica, 72 (2015), pp. 83–98.

[4] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, *Finding k points with minimum diameter and related problems*, J. Algorithms, 12 (1991), pp. 38 – 56.

[5] S. D. Ahipaşaoğlu and E. A. Yildirim, *Identification and elimination of interior points for the minimum enclosing ball problem*, SIAM J. Optim., 19 (2008), pp. 1392–1396.

[6] S. D. Ahipaşaoğlu, *Solving Ellipsoidal Inclusion and Optimal Experimental Design Problems: Theory and Algorithms*, PhD Thesis, Cornell University, 2009.

[7] ——, *Fast algorithms for the minimum volume estimator*, J. Global Optim., 62 (2015), pp. 351–370.

[8] F. Alizadeh and D. Goldfarb, *Second-order cone programming*, Math. Program., 95 (2003), pp. 3–51.

[9] Z. Allen-Zhu, Z. Liao, and Y. Yuan, *Optimization algorithms for faster computational geometry*, in Proc. 43rd International Colloquium on Automata, Languages, and Programming, ICALP '16, 2016.

[10] L. Barba, S. Durocher, R. Fraser, F. Hurtado, S. Mehrabi, D. Mondal, J. Morrison, M. Skala, and M. A. Wahid, *On k-enclosing objects in a coloured point set*, in Proc. 25th Canadian Conf. on Computational Geometry, CCCG2014, 2014, pp. 229–234.

[11] L. J. Bass and S. R. Schubert, *On finding the disc of minimum radius containing a given set of points*, Math. Comp., 21 (1967), pp. 712–714.

[12] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, *Support vector clustering*, J. Mach. Learn. Res., 2 (2002), pp. 125–137.

[13] B. K. Bhattacharya and G. T. Toussaint, *On geometric algorithms that use the furthest-point voronoi diagram*, in Comput. Geom., vol. 2 of Machine Intelligence and Pattern Recognition, North-Holland, 1985, pp. 43 – 61.

[14] L. M. Blumenthal and G. E. Wahlin, *On the spherical surface of smallest radius enclosing a bounded subset of* $\mathfrak{n}$*-dimensional euclidean space*, Bull. Amer. Math. Soc., 47 (1941), pp. 771–777.

[15] M. Bădoiu and K. L. Clarkson, *Smaller core-sets for balls*, in Proc. 14th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA '03, Philadelphia, PA, USA, 2003, SIAM, pp. 801–802.

[16] ——, *Optimal core-sets for balls*, Comput. Geom., 40 (2008), pp. 14 – 22.

[17] M. Bădoiu, S. Har-Peled, and P. Indyk, *Approximate clustering via core-sets*, in Proc. 34th Annual ACM Symp. on Theory of Computing, STOC '02, New York, NY, USA, 2002, ACM, pp. 250–257.

[18] Y. Bulatov, S. Jambawalikar, P. Kumar, and S. Sethia, *Hand recognition using geometric classifiers*, in Biometric Authentication, D. Zhang and A. K. Jain, eds., Berlin, Heidelberg, 2004, Springer, pp. 753–759.

[19] J. A. Candela, *Exact iterative computation of the multivariate minimum volume ellipsoid estimator with a branch and bound algorithm*, in Proc. 12th Symp. Computational Statistics, COMPSTAT 1996, Heidelberg, 1996, Physica-Verlag HD, pp. 175–180.

[20] M. Cavaleiro and F. Alizadeh, *A faster dual algorithm for the Euclidean minimum covering ball problem*, Ann. Oper. Res., (2018).

[21] T. M. Chan and V. Pathak, *Streaming and Dynamic Algorithms for Minimum Enclosing Balls in High Dimensions*, WADS 2011, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 195–206.

[22] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, *Choosing multiple parameters for support vector machines*, Mach. Learn., 46 (2002), pp. 131–159.

[23] B. Chazelle and J. Matoušek, *On linear-time deterministic algorithms for optimization problems in fixed dimension*, J. Algorithms, 21 (1996), pp. 579 – 597.

[24] X. W. Chen, *An improved branch and bound algorithm for feature selection*, Pattern Recognit. Lett., 24 (2003), pp. 1925 – 1933.

[25] V. Chvátal, *Linear Programming*, W. H. Freeman, New York, NY, USA, 1983.

[26] K. L. Clarkson, *Las vegas algorithms for linear and integer programming when the dimension is small*, J. ACM, 42 (1995), pp. 488–499.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, The MIT Press, Cambridge, MA, 3rd ed., 2009.

[28] C. Croux, G. Haesbroeck, and P. J. Rousseeuw, *Location adjustment for the minimum volume ellipsoid estimator*, Stat. Comput., 12 (2002), pp. 191–200.

[29] G. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, USA, 1963.

[30] A. Datta, H. Lenhof, C. Schwarz, and M. Smid, *Static and dynamic algorithms for k-point clustering problems*, J. Algorithms, 19 (1995), pp. 474 – 503.

[31] P. M. Dearing, P. Belotti, and A. M. Smith, *A primal algorithm for the weighted minimum covering ball problem in* $\mathbb{R}^n$, TOP, 24 (2016), pp. 466–492.

[32] P. M. Dearing and A. Smith, *A dual algorithm for the minimum covering weighted ball problem in* $\mathbb{R}^n$, J. Global Optim., 55 (2013), pp. 261–278.

[33] P. M. Dearing and C. R. Zeck, *A dual algorithm for the minimum covering ball problem in* $\mathbb{R}^n$, Oper. Res. Lett., 37 (2009), pp. 171–175.

[34] M. Dyer, *On a multidimensional search technique and its application to the euclidean one-centre problem*, SIAM J. Comput., 15 (1986), pp. 725–738.

[35] M. Dyer, B. Gärtner, N. Megiddo, and E. Welzl, *Linear programming*, Chapman and Hall/CRC, Boca Raton, FL, 3rd ed., 2017, ch. 49.

[36] A. Efrat, M. Sharir, and A. Ziv, *Computing the smallest k-enclosing circle and related problems*, in Algorithms and Data Structures, F. Dehne, J.-R. Sack, N. Santoro, and S. Whitesides, eds., Berlin, Heidelberg, 1993, Springer, pp. 325–336.

[37] D. J. Elzinga and D. W. Hearn, *The minimum covering sphere problem*, Manag. Sci., 19 (1972), pp. 96–104.

[38] D. Eppstein and J. G. Erickson, *Iterated nearest neighbors and finding minimal polytopes*, in Proc. 4th ACM-SIAM Symp. on Discrete Algorithms, SODA '13, Philadelphia, PA, USA, 1993, SIAM, pp. 64–73.

[39] H.-Y. Feng, D. H. Endrias, M. A. Taher, and H. Song, *An accurate and efficient algorithm for determining minimum circumscribed circles and spheres from discrete data points*, Comput.-Aided Des., 45 (2013), pp. 105 – 112. Solid and Physical Modeling 2012.

[40] K. Fischer, *Smallest enclosing balls of balls*, PhD Thesis, ETH Zürich, 2005.

[41] K. Fischer and B. Gärtner, *The smallest enclosing ball of balls: Combinatorial structure and algorithms*, Internat. J. Comput. Geom. Appl., 14 (2004), pp. 341–387.

[42] K. Fischer, B. Gärtner, and M. Kutz, *Fast smallest-enclosing-ball computation in high dimensions*, in Algorithms - ESA 2003. Lecture Notes in Computer Science, G. Di Battista and U. Zwick, eds., vol. 2832, Springer, 2003, pp. 630–641.

[43] B. Gärtner, *A subexponential algorithm for abstract optimization problems*, SIAM J. Comput., 24 (1995), pp. 1018–1035.

[44] ——, *Fast and Robust Smallest Enclosing Balls*, ESA '99, Springer, Berlin, Heidelberg, 1999, pp. 325–338.

[45] B. Gärtner and S. Schönherr, *An efficient, exact, and generic quadratic programming solver for geometric optimization*, in Proc. 16th annual ACM Symp. on Computational Geometry, SCG, 2000, pp. 110–118.

[46] B. Gärtner and E. Welzl, *Linear programming — randomization and abstract frameworks*, in STACS 96, C. Puech and R. Reischuk, eds., Berlin, Heidelberg, 1996, Springer, pp. 667–687.

[47] ——, *Explicit and Implicit Enforcing - Randomized Optimization*, Springer, Berlin, Heidelberg, 2001, pp. 25–46.

[48] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.

[49] A. Goel, P. Indyk, and K. Varadarajan, *Reductions among high dimensional proximity problems*, in Proc. 12th ACM-SIAM Symp. Discrete Algorithms, SODA '01, Philadelphia, PA, USA, 2001, SIAM, pp. 769–778.

[50] D. Goldfarb, *Factorized variable metric methods for unconstrained optimization*, Math. Comp., 30 (1976), pp. 796–811.

[51] D. Goldfarb and A. Idnani, *A numerically stable dual method for solving strictly convex quadratic programs*, Math. Program., 27 (1983), pp. 1–33.

[52] G. H. Golub and C. Van Loan, *Matrix Computations (3rd Ed.)*, Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[53] Gurobi Optimization Inc., *Gurobi Optimizer Reference Manual*, 2016.

[54] T. S. Hale and C. R. Moberg, *Location science research: A review*, Ann. Oper. Res., 123 (2003), pp. 21–35.

[55] S. Har-Peled and S. Mazumdar, *Fast algorithms for computing the smallest k-enclosing circle*, Algorithmica, 41 (2005), pp. 147–157.

[56] S. Har-Peled and B. Raichel, *Net and prune: A linear time algorithm for euclidean distance problems*, J. ACM, 62 (2015), pp. 44:1–44:35.

[57] S. Har-Peled and Y. Wang, *Shape fitting with outliers*, SIAM J. Comput., 33 (2004), pp. 269–285.

[58] T. H. Hopp and C. P. Reeve, *An algorithm for computing the minimum covering sphere in any dimension*, Technical Report NISTIR 5831, National Institute of Standards and Technology, Gaithersburg, MD, USA, (1996).

[59] P. M. Hubbard, *Approximating polyhedra with spheres for time-critical collision detection*, ACM Trans. Graph., 15 (1996), pp. 179–210.

[60] S. K. Jacobsen, *An algorithm for the minimax weber problem*, European J. Oper. Res., 6 (1981), pp. 144 – 148. Location Decisions.

[61] B. Kalantari, *A characterization theorem and an algorithm for a convex hull problem*, Ann. Oper. Res., 226 (2015), pp. 301–349.

[62] L. Källberg and T. Larsson, *Faster approximation of minimum enclosing balls by distance filtering and gpu parallelization*, J. Graph. Tools, 17 (2013), pp. 67–84.

[63] ——, *Improved pruning of large data sets for the minimum enclosing ball problem*, Graph. Models, 76 (2014), pp. 609 – 619.

[64] A. Karmakar, S. Das, S. C. Nandy, and B. K. Bhattacharya, *Some variations on constrained minimum enclosing circle problem*, in Combinatorial Optimization and Applications, W. Wu and O. Daescu, eds., Berlin, Heidelberg, 2010, Springer, pp. 354–368.

[65] P. Kumar, J. S. B. Mitchell, and E. A. Yildirim, *Approximate minimum enclosing balls in high dimensions using core-sets*, J. Exp. Algorithmics, 8 (2003).

[66] T. Larsson, G. Capannini, and L. Källberg, *Parallel computation of optimal enclosing balls by iterative orthant scan*, Comput. & Graphics, 56 (2016), pp. 1 – 10.

[67] T. Larsson and L. Källberg, *Fast and robust approximation of smallest enclosing balls in arbitrary dimensions*, in Proc. 11th Eurographics/ACMSIGGRAPH Symp. on Geometry Processing, SGP '13, Aire-la-Ville, Switzerland, 2013, Eurographics Association, pp. 93–101.

[68] C. Lawson, *The smallest covering cone or sphere (c. groenewod and l. eusanio)*, SIAM Rev., 7 (1965), pp. 415–416.

[69] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, Algorithmica, 16 (1996), pp. 498–516.

[70] J. Matoušek, *On enclosing k points by a circle*, Inform. Process. Lett., 53 (1995), pp. 217 – 221.

[71] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, in Proc. 8th Annual Symp. on Computational Geometry, SCG '92, New York, NY, USA, 1992, ACM, pp. 1–8.

[72] N. Megiddo, *Applying parallel computation algorithms in the design of serial algorithms*, in Proc. 22nd Annual Symp. on Foundations of Computer Science, SFCS 1981, 1981, pp. 399–408.

[73] ——, *Linear programming in linear time when the dimension is fixed*, J. ACM, 31 (1984), pp. 114–127.

[74] ——, *On the ball spanned by balls*, Discrete Comput. Geom., 4 (1989), pp. 605–610.

[75] E. Moradi and M. Bidkhori, *Single facility location problem*, in Facility Location: Concepts, Models, Algorithms and Case Studies, R. Zanjirani Farahani and M. Hekmatfar, eds., Heidelberg, 2009, Physica-Verlag HD, pp. 37–68.

[76] B. Mordukhovich, N. M. Nam, and C. Villalobos, *The smallest enclosing ball problem and the smallest intersecting ball problem: existence and uniqueness of solutions*, Optim. Lett., 7 (2013), pp. 839–853.

[77] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 8.1*, 2017.

[78] D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, *Quantile approximation for robust statistical estimation and k-enclosing problems*, Internat. J. Comput. Geom. Appli., 10 (2000), pp. 593–608.

[79] N. M. Nam, N. Hoang, and N. T. An, *Constructions of solutions to generalized sylvester and fermat–torricelli problems for euclidean balls*, J. Optimi. Theory Appl., 160 (2014), pp. 483–509.

[80] N. M. Nam, T. A. Nguyen, and J. Salinas, *Applications of convex analysis to the smallest intersecting ball problem*, J. Convex Anal., 19 (2012), pp. 497–518.

[81] P. M. Narendra and K. Fukunaga, *A branch and bound algorithm for feature subset selection*, IEEE Trans. Comput., 26 (1977), pp. 917–922.

[82] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM, Philadelphia, PA, USA, 1994.

[83] Y. Nesterov and M. Todd, *Self-scaled barriers and interior-point methods for convex programming*, Math. Oper. Res., 22 (1997), pp. 1–42.

[84] ——, *Primal-dual interior-point methods for self-scaled cones*, SIAM J. Optim., 8 (1998), pp. 324–364.

[85] F. Nielsen and R. Nock, *Approximating smallest enclosing balls with applications to machine learning*, Internat. J. Computat. Geom. Appl., 19 (2009), pp. 389–414.

[86] R. Panigrahy, *Minimum enclosing polytope in high dimensions*, CoRR, cs.CG/0407020 (2004).

[87] F. Plastria, *Continuous covering location problems*, in Facility Location: Applications and Theory, Z. Drezner and H. W. Hamacher, eds., Berlin, 2002, Springer, pp. 37–79.

[88] L. Pronzato, *On the elimination of inessential points in the smallest enclosing ball problem*, Optim. Methods Softw., 34 (2019), pp. 225–247.

[89] J. Ritter, *An efficient bounding sphere*, in Graph. Gems Ser., A. S. Glassner, ed., Academic Press Professional, Inc., San Diego, CA, USA, 1990, pp. 301–303.

[90] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*, John Wiley & Sons, Inc., New York, NY, USA, 1987.

[91] A. Ruszczynski, *Nonlinear Optimization*, Princeton University Press, Princeton, NJ, USA, 2006.

[92] A. Saha, S. V. N. Vishwanathan, and X. Zhang, *New approximation algorithms for minimum enclosing convex shapes*, in Proc. 22nd Annual ACM-SIAM Symp. on Discrete Algorithms, SODA '11, Philadelphia, PA, USA, 2011, SIAM, pp. 1146–1160.

[93] M. I. Shamos and D. Hoey, *Closest-point problems*, in Proc. 16th Annual Symp. on Foundations of Computer Science, SFCS 1975, 1975, pp. 151–162.

[94] M. Sharir and E. Welzl, *A combinatorial bound for linear programming and related problems*, in STACS 92, A. Finkel and M. Jantzen, eds., Berlin, Heidelberg, 1992, Springer, pp. 567–579.

[95] V. V. Shenmaier, *The problem of a minimal ball enclosing k points*, J. Appl. Ind. Math., 7 (2013), pp. 444–448.

[96] S. Skyum, *A simple algorithm for computing the smallest enclosing circle*, Inform. Process. Lett., 37 (1991), pp. 121 – 125.

[97] G. Stewart, *Matrix Algorithms*, SIAM, Philadelphia, PA, USA, 1998.

[98] J. J. Sylvester, *A question in the geometry of situation*, Quaterly J. Pure and Applied Math., (1857), pp. 1–79.

[99] T. Szabo and E. Welzl, *Unique sink orientations of cubes*, in Proc. 42nd IEEE Symp. on Foundations of Computer Science, SFCS 2001, 2001, pp. 547–555.

[100] M. Todd, *Minimum-Volume Ellipsoids*, SIAM, Philadelphia, PA, USA, 2016.

[101] K. C. Toh, M. Todd, and R. H. Tütüncü, *Sdpt3 – a matlab software package for semidefinite programming*, Optim. Methods Softw., 11 (1999), pp. 545–581.

[102] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, *Core vector machines: Fast svm training on very large data sets*, J. Mach. Learn. Res., 6 (2005), pp. 363–392.

[103] C. Van de Panne and A. Whinston, *Simplicial methods for quadratic programming*, Naval Res. Logist. Quart., 11 (1964), pp. 273–302.

[104] E. Welzl, *Smallest enclosing disks (balls and ellipsoids)*, in New Results and New Trends in Computer Science Proc., H. Maurer, ed., Berlin, Heidelberg, 1991, Springer, pp. 359–370.

[105] P. Wolfe, *The simplex method for quadratic programming*, Econometrica, 27 (1959), pp. 382–398.

[106] X. Wu, *A linear-time simple bounding volume algorithm*, in Graph. Gems Ser. III (IBM Version), D. Kirk, ed., Morgan Kaufmann, San Francisco, CA, USA, 1992, pp. 301 – 306.

[107] E. A. Yildirim, *Two algorithms for the minimum enclosing ball problem*, SIAM J. Optim., 19 (2008), pp. 1368–1391.

[108] B. Yu and B. Yuan, *A more efficient branch and bound algorithm for feature selection*, Pattern Recognit., 26 (1993), pp. 883 – 889.

[109] H. Zarrabi-Zadeh and T. M. Chan, *A simple streaming algorithm for minimum enclosing balls*, in Proc. 18th Canadian Conf. on Computational Geometry, CCCG2006, 2006, pp. 139–142.

[110] G. Zhou, K.-C. Tohemail, and J. Sun, *Efficient algorithms for the smallest enclosing ball problem*, Comput. Optim. Appl., 30 (2005), pp. 147–160.