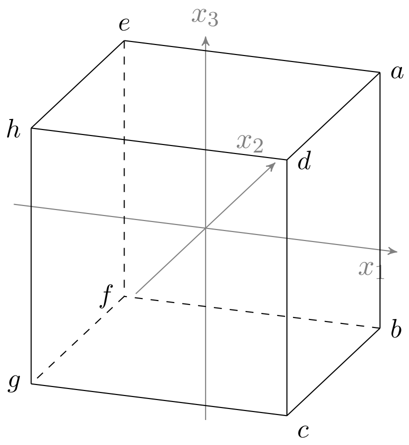


The Convex Hull Problem in Practice

Improving the Running Time of the Double Description Method

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
- Dr.-Ing. -

vorgelegt von

Blagoy Genov

Universität Bremen

Fachbereich 3: Mathematik und Informatik

The Convex Hull Problem in Practice

Improving the Running Time of the Double Description Method

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
- Dr.-Ing. -

vorgelegt von

Blagoy Genov

im Fachbereich 3 (Informatik/Mathematik)
der Universität Bremen

im Juli 2014

Tag der mündlichen Prüfung: 28. Januar 2015

Gutachter:
Prof. Dr. Jan Peleska (Universität Bremen)
Prof. Dr. Udo Frese (Universität Bremen)

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig und ohne unerlaubte fremde Hilfe angefertigt habe, dass ich alle benutzen Quellen und Hilfsmittel vollständig angegeben habe und dass alle Stellen, die ich wörtlich oder dem Sinne nach aus anderen Arbeiten entnommen habe, kenntlich gemacht worden sind.

Ort, Datum

Unterschrift

Zusammenfassung

Die Double-Description-Methode ist ein weit verbreiteter Algorithmus zur Berechnung von konvexen Hüllen und anderen verwandten Problemen. Wegen seiner unkomplizierten Umsetzung wird er oft in der Praxis bevorzugt, obwohl seine Laufzeitkomplexität nicht durch die Größe der Ausgabe beeinflusst wird. Aufgrund seiner zunehmenden Bedeutung in der Programmverifikation und der Analyse von metabolischen Netzwerken ist der Bedarf nach schnellen und zuverlässigen Implementierungen in den letzten Jahren rasant gestiegen. Bei den aktuellen Anwendungen besteht erhebliches Potenzial zur Verkürzung der Rechenzeit.

Die Aufzählung von benachbarten extremalen Halbgeraden gehört zu den rechenintensivsten Aktivitäten innerhalb der Double-Description-Methode und ist damit ein potentieller Optimierungskandidat. Zudem ist die praktische Seite dieses kombinatorischen Problems nicht ausgiebig erforscht worden. In dieser Dissertation werden zwei bedeutsame Beiträge zur Beschleunigung des Aufzählungsprozesses präsentiert. Zum einen werden die gängigen Datenstrukturen evaluiert und daraufhin diverse Optimierungen vorgeschlagen. Ihre Wirksamkeit wird durch empirische Daten demonstriert. Zum anderen wird ein neuer Nachbarschaftstest für extremale Halbgeraden vorgestellt. Dieser ist eine Weiterentwicklung des bekannten algebraischen Tests mit dem Vorteil, dass redundante Rechenoperationen eliminiert werden. Die Korrektheit des neuen Ansatzes wird formal bewiesen. Seine Wirksamkeit bei ausgearteten Problemen wird ebenfalls durch empirische Daten belegt.

Nebenläufigkeit ist ein weiterer Aspekt der Double-Description-Methode, der zusätzliche Untersuchung verlangt. Ein neulich vorgestellter Ansatz, aufgebaut nach dem teile-und-herrsche Prinzip, hat vielversprechende Ergebnisse in der Bioinformatik gezeigt. In dieser Dissertation werden allerdings massive praktische Einschränkungen dargelegt. Für eine Klasse von Schnitt-Polytopen hat die vorgestellte Technik beträchtliche Erhöhung der Rechenzeit verursacht. Dieses Problem wurde umgangen, indem die Methode weiterentwickelt wurde, um partielle Nebenläufigkeit zu ermöglichen. Diese hat sich in praktischen Bedingungen als stabil genug erwiesen, obwohl nur moderate Beschleunigung erzielt werden konnte.

Im Laufe dieser Arbeit wurde eine neue Implementierung der Double-Description-Methode entwickelt. Diese enthält nicht nur die oben angesprochenen konzeptionellen sondern auch viele technische Verbesserungen. Sie ist besonders auf die neuen Generationen von Prozessoren optimiert worden und ist in vielen Aspekten den gängigen frei verfügbaren Implementierungen überlegen. Diese Schlussfolgerung wird durch empirische Daten unterstützt.

Abstract

The double description method is a widely spread algorithm for computation of convex hulls and other related problems. Even though not being output sensitive, it is often preferred in practical applications due to its general simplicity. In recent years, the demand for faster and more reliable implementations has rapidly increased in view of its growing application in program verification, analysis of metabolic networks, etc. Although several implementations are available, there is a considerable potential for further performance related improvements.

The enumeration of adjacent extreme rays is arguably the most performance critical part of the double description method and hence a natural candidate for optimization. Moreover, the practical side of this combinatorial problem has not been extensively studied yet. In this dissertation, two significant contributions related to adjacency testing are presented. First, the currently used data structures are revisited and various optimizations are proposed. Empirical evidence is provided to demonstrate their competitiveness. Second, a new adjacency test is introduced. It is a refinement of the well known algebraic test featuring a technique for avoiding redundant computations. Its correctness is formally proven. Its superiority in multiple degenerate scenarios is demonstrated through experimental results.

Parallel computation is one further aspect of the double description method which deserves particular attention. A recently introduced divide-and-conquer technique showed promising results in bioinformatic applications. In this dissertation, however, some severe practical limitations are demonstrated. For a class of cut polytopes, the proposed technique caused a serious computational explosion on at least one of the resulting threads. As an alternative, the technique was refined to perform a partial parallel computation which eliminated the described explosion and brought a moderate speed-up.

In the course of this work, a new double description implementation was developed. It embodies not only the conceptional improvements presented in this work but also technical ones. It is particularly optimized for the latest generation of processors and is highly competitive with other freely available double description implementations. Empirical evidence is provided to back up that conclusion.

Acknowledgements

Starting my dissertation four years ago in the working group for operating and distributed systems, I knew very little about convex hulls and the problems related to them. To be honest, I had never planned to write a thesis on that subject. But as time went by, one open problem led to another, until I found myself working on that challenging but very interesting subject.

I owe my deepest gratitude to my advisor Professor Jan Peleska for his patience and the freedom he gave me to put my ideas into realization. Without his support, this work would not have been possible.

I wish to extend special thanks to Professor Wen-ling Huang for helping me polish the formal proofs. I am also very grateful to my colleges Florian Lapschies and Elena Gorbachuk for their readiness to proofread my work. Florian had to endure countless discussions on software efficacy, performance and optimization. I deeply appreciate that.

Last but not least, I wish to thank my family for all their support and encouragement. I could hardly imagine finishing this work without having my wife Genka by my side.

Hamburg, Juni 2014

Contents

List of Figures

List of Tables

1 Introduction

1.1 Objectives and Motivation

The convex hull problem is one of the oldest and most intensively studied problems in computational geometry. A great amount of algorithms have been developed, each of them claiming some small or substantial advantage towards the previous ones. Still, none of them is able to solve the problem in time polynomial in the size of the input and the output. On the other hand, there exists no formal proof ruling out the existence of such an algorithm; hence, the question of whether the convex hull problem is an NP-hard one is so far open.

The theoretical aspects of the convex hull problem did not lie within the scope of this work; hence, no new insights in that regard are presented. Instead, efforts were directed towards the practical application of one particular algorithm called the *double description method* [116]. A primary objective was to improve its running time by applying a series of optimizations related to data structures, elimination of redundant computations and parallelization. In particular, the focus was laid on degenerate problems and rational arithmetic. For nondegenerate problems, the *reverse search method* of Avis and Fukuda [13] is known to be superior and thus a primary choice for this particular problem group.

One particular application field of the double description method is the automated test case generation where efficient libraries for polyhedral computations become more and more important. This dissertation can be seen as a complementary work to the test case generator of Peleska et al. [121] which employs an SMT-solver (Satisfiability Modulo Theories) [20] to calculate the inputs necessary to reach a certain state of the system under test. In addition, *abstract interpretation* [55, 56] serves as a guiding tool for the solver, assisting it in the calculation of the input values and supplying it with information about potentially unsolvable constraints. The quality of assistance depends highly on the precision which the abstract interpretation can offer. In this respect, Cousot and Halbwachs [57] demonstrated the application of abstract interpretation on the polyhedral domain to determine linear constraints among program variables. This particular technique proved to be far more powerful in comparison to other non-relational domains such as interval arithmetic where the relationship between program variables is not explicitly considered. The higher precision, however, comes at the cost of some highly intensive polyhedral computations related to the convex hull problem. That was one of the reasons for the initially limited applicability of Cousot and Halbwachs' approach. As explained by Bagnara et al. [16], no library for polyhedral computations could meet the performance and robustness demands of the new technique by the time of its introduction. During that period, the geometric community was mainly debating on the theoretical nature of the convex hull problem; thus, it was not until the nineties that such libraries were developed. Some of them were particularly designed for program analysis and verification. Such are NEW POLKA, as part of Jeannet and Miné's tool APRON [91], and the more recent PPL of Bagnara et al. [15], both using the double description method. Other libraries such as Fukuda's CDDLIB [75] and Avis' LRSLIB [9] were designed with focus on the general nature of the problem, but were also successfully applied in the context of program analysis and verification.

Apart from program verification, the double description method has been applied in bioinformatics to compute *elementary flux modes* and *extreme pathways* in metabolic networks [132, 151, 79]. Multiple implementations were introduced as part of different analysis tools [21, 150, 141]. Moreover, noteworthy efforts were spent to obtain an applicable divide-and-conquer parallelization technique [100, 93, 92].

Despite the considerably large number of implementations, the evolution of the double description method can hardly be declared as finished. The new ideas proposed in this work were successfully employed to obtain a meaningful performance gain. Those optimizations are not related to any particular application area. The scope of this work is to bring together the strengths of the available implementations into one application.

1.2 Contributions

This work involves three major contributions. First, analysis, refinement and enhancement of the currently used data structures. Second, elimination of redundant calculations by caching intermediate results. Third, running the double description method on a distributed system.

Data Structures. Testing extreme rays for adjacency is arguably the most time consuming part of the double description method and requires the execution of a huge amount of search queries. Depending on the problem size, the number of those queries may easily reach tens or even hundreds of millions. The choice of appropriate data structures is therefore essential for achieving an efficient implementation. Even a minimal performance gain per query may deliver a substantial speed-up in view of the overall calculation time. With the *bit pattern tree* [143, 144], Terzer and Stelling introduced a new data structure which allows to effectively narrow down the search region. Here, its performance is revisited and several strategies for more efficient application are proposed. Furthermore, two additional data structures are introduced,

(1) an *extended bit pattern tree*, which narrows down the search region even more aggressively, and

(2) a *population tree*, which is an abstraction of the bit pattern tree and hence a more powerful solution from a theoretical point of view.

Finally, the application of *vantage point trees* [155] (aka *metric trees* [147]) within the double description method is demonstrated.

The given data structures were investigated in the context of a benchmark consisting of random problems with different level of degeneracy. First results had been published in [83]. The performed experiments clearly indicated the general superiority of bit pattern trees when applied in conjunction with the proposed heuristics. The extended bit pattern trees did not bring any additional improvement when used as a standalone solution; however, they brought a small speed-up for low degenerate problems when used as a complement to bit pattern trees. Population and vantage point trees were unable to deliver a general improvement. Yet, they showed superiority in some specific areas.

Redundancy Elimination. In an alternative form of adjacency testing, known as *algebraic test*, search queries are substituted by matrix rank computations; that is, the combinatorial problem is transformed to an algebraic one. That algebraic test, however, performs a great amount of redundant computations as many of the processed matrices differ in several rows only. Identical algebraic operations are thus repeated over and over again on each rank computation.

In order to overcome the above problem, the matrix rank algorithm was redesigned by integrating a global cache which holds information about linear dependencies of matrix row vectors. The underlying idea is to use the available cache information at the start of each new matrix rank computation in order to reduce the initial problem as much as possible. Consequently, the algorithm deals with much smaller matrices and in some cases, the problem can be reduced to a trivially solvable one.

The new technique was tested against different degenerate problems to compare its performance to the classic algebraic test and the combinatorial one. It performed best for a broad spectrum of problems with various degeneracy. The combinatorial test remained superior for problems featuring very high degeneracy.

Parallelization. Modern processors offer new parallelization possibilities in view of their multi-core architecture. Designing a multi-core version of the double description method can substantially improve its running time and is mostly a matter of technical realization. A much more complicated issue, however, is running the algorithm on a distributed system. In that respect, a recent contribution of Jevremović et al. [93] pointed out an easy to implement divide-and-conquer technique. The idea has been tried out in practice, achieving a substantial performance gain in the context of metabolic networks.

The complementary computational experience provided in this work demonstrates that for certain problems, a family of cut polytopes [18], the divide-and-conquer approach results in a serious computational explosion. After experimenting with different load balancing techniques, the explosion could be reduced but still not avoided. In the course of those experiments, however, another way of performing a partial divide-and-conquer computation emerged. In contrast to the approach of Jevremović et al., the distribution of the computation on all available processors is not done until the last steps of the algorithm. Despite the reduced achievable performance gain, this strategy performed robustly enough in practical conditions.

1.3 Thesis Overview

The thesis is organized as follows:

In Chapter 2, the necessary mathematical foundations are presented. The book of Schrijver [131] served as reference work during the preparation of this chapter.

In Chapter 3, the convex hull problem is introduced. Related problems are presented and the computational complexity is discussed. Furthermore, an overview of existing algorithms is given.

In Chapter 4, the double description method is presented. Implementation techniques and data structures resulting from the work of Fukuda and Prodon [77] and Terzer [141] are described.

In Chapter 5, several weaknesses of the bit pattern trees are pointed out and optimizations to deal with them are proposed. Furthermore, new data structures suitable for adjacency testing are introduced.

In Chapter 6, the new algebraic test is described and its correctness is proven.

In Chapter 7, the divide-and-conquer technique is revisited and its general applicability is discussed.

In Chapter 8, the developed double description implementation is compared with other noncommercial programs in terms of performance.

Chapter 9 is devoted to conclusions.

1.4 Notation and Conventions

Throughout this work, certain symbols are used with exclusively predefined meaning. If not stated otherwise,

A denotes an $n \times d$ matrix with $n > d$ which has full column rank,

a_i denotes the i-th row vector of A,

A_Q with $Q \subset \mathbb{N}$ denotes an $m \times d$ submatrix of A in the sense that each row vector a_i of A is also a row vector of A_Q if and only if $i \in Q$,

$\mathcal{C}$ denotes a polyhedral cone,

$\mathbb{N}_{\leq k}$ denotes the set of successive natural numbers $\{1, \ldots, k\}$ for $k \geq 1$,

$\mathcal{P}$ denotes a (possibly bounded) polyhedron,

r denotes an extreme ray,

$\mathcal{R}$ denotes a set of extreme rays, and

$\mathcal{V}$ denotes a set of vertices.

2 Preliminaries: Linear Inequalities and Polyhedra

The convex hull problem is closely related to systems of linear inequalities and the therefrom resulting convex polyhedra (see Section 2.1). Polyhedra are structurally decomposable into finitely many faces (see Section 2.2) which have important isomorphic properties emerging from the concept of duality (see Section 2.3). Those aspects play a central role in convex hull computations.

Convex polyhedra do not always come up in general form. Such non-general problems are said to be degenerate and pose a particular challenge for algorithmic computations (see Section 2.4).

2.1 Polyhedra, Polyhedral Cones and Polytopes

Let $C \subseteq \mathbb{R}^d$ be an arbitrary set of vectors. C is a *convex* set if

$$\lambda x + (1 - \lambda) y \in C$$

holds for every $x, y \in C$ and $\lambda \in \mathbb{R}$, $0 \leq \lambda \leq 1$. If λ is not explicitly bounded, then C is *affine*. By definition, every affine set is automatically convex. The *convex hull* of a vector set $X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^d$ with $m \in \mathbb{N}$

$$\texttt{conv.hull}(X) = \left\{ \sum_{i=1}^{m} \lambda_i x_i \mid \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \in \mathbb{R}, \lambda_i \geq 0 \right\} \tag{2.1}$$

is the smallest convex set containing X. Analogously, the *affine hull* of X

$$\texttt{aff.hull}(X) = \left\{ \sum_{i=1}^{m} \lambda_i x_i \mid \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \in \mathbb{R} \right\} \tag{2.2}$$

is the smallest affine set containing X. Finally, the *linear hull* (aka *linear span*) of X

$$\texttt{lin.hull}(X) = \left\{ \sum_{i=1}^{m} \lambda_i x_i \mid \lambda_i \in \mathbb{R} \right\} \tag{2.3}$$

is the smallest subspace of $\mathbb{R}^d$ containing X.

The set $C \subseteq \mathbb{R}^d$ is a *cone* if

$$\lambda x + \mu y \in C$$

holds for every $x, y \in C$ and $\lambda, \mu \in \mathbb{R}$, $\lambda \geq 0$, $\mu \geq 0$. The cone *generated* by the vector set $X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^d$

$$\texttt{cone}\,(X) = \left\{ \sum_{i=1}^{m} \lambda_i x_i \mid \lambda_i \geq 0 \right\} \tag{2.4}$$

is the smallest cone containing X. X is said to be *minimal* if there is no proper subset $X' \subset X$ generating the same cone. Another way of showing that X is minimal is by showing the nonexistence of a vector $x_j \in X$ which can be represented as a nonnegative combination of vectors in $X - \{x_j\}$; that is,

$$x_j \neq \sum_{i=1}^{j-1} \lambda_i x_i + \sum_{i=j+1}^{m} \lambda_i x_i \quad \text{for all } \lambda \in \mathbb{R}^m \text{ with } \lambda_i \geq 0, i \in [1, m] \ . \tag{2.5}$$

Definition 2.1 [Polyhedral Cone]**.** A nonempty set of vectors $\mathcal{C} \subseteq \mathbb{R}^d$ is called a *polyhedral cone* if for some matrix $A \in \mathbb{R}^{n \times d}$,

$$\mathcal{C} = \{x \mid Ax \leq 0\} \ . \tag{2.6}$$

A polyhedral cone defined by a matrix A is denoted as $\mathcal{C}\,(A)$.

In geometric terms, the polyhedral cone $\mathcal{C}$ is the intersection of finitely many *linear halfspaces* $\{x \mid a_i x \leq 0\}$ defined by the row vectors a_i of the matrix A. As each such halfspace contains the origin, it is guaranteed that $\mathcal{C}$ is never empty. The linear subspace which is shared among all halfspaces is called *lineality space* of $\mathcal{C}$. It is the linear span of those solutions $y \in \mathcal{C}$ which satisfy each inequality in (2.6) with equality, i.e.

$$\texttt{lin.space}(\mathcal{C}) = \mathcal{C} \cap \bar{\mathcal{C}} = \{y \mid Ay = 0\} \ .$$

The cone $\mathcal{C}$ is said to be *pointed* if its lineality space contains only the origin. Consequently, $\mathcal{C}$ is pointed if and only if $\mathit{rank}[A] = d$.

The dimension of $\mathcal{C}$ is equivalent to the dimension of the linear subspace in which it resides, thus

$$\mathbf{dim}\,(\mathcal{C}) = \mathbf{dim}\,(\texttt{lin.hull}(\mathcal{C})) \ .$$

The cone dimension can be easily obtained from the matrix A. For the sake of simplicity, assume that A has a strictly predefined form such that

$$A = \begin{bmatrix} A^{=} \\ A^{+} \end{bmatrix} \quad \text{with } A^{=}y = 0 \text{ and } A^{+}y \neq 0 \text{ for all } y \in \mathcal{C} - \{0\} \ . \tag{2.7}$$

Reaching (2.7) is obviously a matter of preprocessing. This decomposition of A allows another representation of the cone

$$\mathcal{C} = \left\{x \mid A^{=}x = 0, A^{+}x \leq 0\right\}$$

with tighter constraints. In geometric terms, $\mathcal{C}$ is now the intersection of both linear halfspaces defined by A^+ and linear hyperplanes $\{x \mid a_i x = 0\}$ defined by $A^=$. Each such hyperplane has a dimension $d-1$, the intersection of two hyperplanes, unless not parallel, has a dimension $d-2$, etc. Consequently, the intersection of all hyperplanes defined by $A^=$ defines a linear subspace with dimension

$$d - \mathit{rank}[A^=]$$

which is equivalent to the linear hull of $\mathcal{C}$. $\mathcal{C}$ is said to be *full-dimensional* if $A^=$ is empty and thus $\mathbf{dim}(\mathcal{C}) = d$.

Theorem 2.1 [Minkowski-Weyl]. *Let $\mathcal{C} \subseteq \mathbb{R}^d$ be a cone. Then $\mathcal{C}$ is polyhedral if and only if it is generated by finitely many vectors.*

As a consequence of Theorem 2.1, each polyhedral cone can be specified either explicitly by a set of generators as given in (2.4), or implicitly by a homogeneous system of linear inequalities as given in (2.6). Those two specifications are known as a $\mathcal{V}$- and an $\mathcal{H}$-representation respectively.

Definition 2.2 [Polyhedron]. A set of vectors $\mathcal{P} \subseteq \mathbb{R}^d$ is called a *polyhedron* if for a matrix $A \in \mathbb{R}^{n \times d}$ and a vector $b \in \mathbb{R}^n$

$$\mathcal{P} = \{x \mid Ax \leq b\} \quad . \tag{2.8}$$

A polyhedron defined by a matrix A and a vector b is denoted as $\mathcal{P}(A, b)$.

Obviously, the polyhedron is a more general form of a polyhedral cone, as it emerges as an intersection of *affine halfspaces* in the form $\{x \mid a_i x \leq b_i\}$. Each polyhedron has a *recession cone* which is obtained by substituting the vector b with the zero vector. In other words, $\mathcal{C}(A)$ is the recession cone of $\mathcal{P}(A, b)$. A polyhedron is pointed if its recession cone is pointed.

The dimension of a polyhedron $\mathcal{P}(A, b)$ is the dimension of its affine hull. In order to determine it, the same technique as on polyhedral cones can be applied. First, assume the decomposition

$$\begin{bmatrix} A & b \end{bmatrix} = \begin{bmatrix} A^= & b^= \\ A^+ & b^+ \end{bmatrix} \quad \text{with } A^= y = b^= \text{ and } A^+ y \neq b^+ \text{ for all } y \in \mathcal{P} \quad .$$

This allows the refinement of (2.8) into

$$\mathcal{P} = \left\{x \mid A^= x = b^=, A^+ x \leq b^+\right\} \quad .$$

Next, it is easy to verify that

$$\texttt{aff.hull}(\mathcal{P}) = \{y \mid A^= y = b^=\}$$

and hence

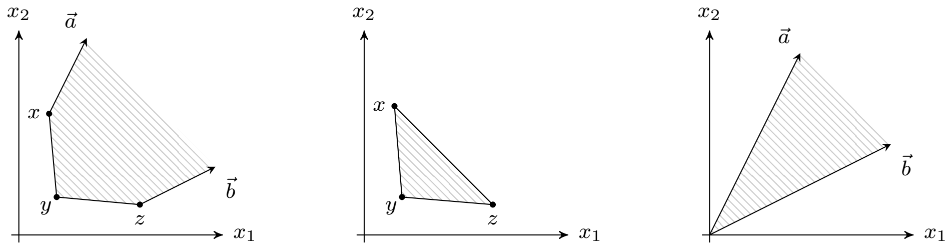
$$\mathbf{dim}(\mathcal{P}) = \mathbf{dim}(\texttt{aff.hull}(\mathcal{P})) = d - \mathit{rank}[A^=] \quad . \tag{2.9}$$

Theorem 2.2 [Decomposition Theorem for Polyhedra]**.** *$\mathcal{P} \subseteq \mathbb{R}^d$ is a polyhedron if and only if*

$$\mathcal{P} = \texttt{conv.hull}(X) + \texttt{cone}\,(Y) \quad \text{for two finite sets } X, Y \subset \mathbb{R}^d \ .$$

Following Schrijver [131], we can think of a polyhedron as a set of points X and a set of directions Y. In Figure 2.1, the consequences of Theorem 2.2 are illustrated for a two-dimensional polyhedron. If Y contains merely the origin then the polyhedron has no directions and is said to be *bounded*. A bounded polyhedron is the convex hull of finitely many points and hence a polytope (see Definition 2.3). On the other hand, if X is empty, then the polyhedron is defined only by a set of directions and thus reduces to a polyhedral cone.

Definition 2.3 [Polytope]**.** The convex hull of a finite set $X \subset \mathbb{R}^d$ is called a *polytope*.



(a) Polyhedron (b) Polytope (c) Polyhedral Cone

Figure 2.1: Decomposition of a polyhedron into a polytope and a cone

2.2 Faces, Extreme Rays and Vertices

Let $X \subset \mathbb{R}^d$ be an arbitrary vector set. A hyperplane $\mathcal{H} = \{x \mid cx = z\}$ is said to *support* X if $\mathcal{H} \cap X \neq \emptyset$ and $X \subseteq \{x \mid cx \leq z\}$. In geometric terms, a supporting hyperplane touches the convex hull of X without dividing it.

Definition 2.4 [Face]**.** For a polyhedron $\mathcal{P}$, a nonempty set $\mathcal{F} \subseteq \mathcal{P}$ is called a *face* of $\mathcal{P}$ if either $\mathcal{F} = \mathcal{P}$ or $\mathcal{F} = \mathcal{P} \cap \mathcal{H}$ where $\mathcal{H}$ is a supporting hyperplane of $\mathcal{P}$.

Each face defined by a supporting hyperplane, that is each face up to $\mathcal{P}$, is a structural part of the polyhedron's surface. Assuming that the $\mathcal{H}$-representation of $\mathcal{P}$ is free of redundancies, it is evident that each affine hyperplane

$$\mathcal{H}_i = \{x \mid a_i x = b_i\}$$

is a supporting one. Consequently, each $\mathcal{H}_i$ defines a face

$$\mathcal{F}_i = \{x \in \mathcal{P} : a_i x = b_i\} \tag{2.10}$$

which is maximal in the sense that no other face defined by a supporting hyperplane contains $\mathcal{F}_i$. Such faces which are distinct from $\mathcal{P}$ and not included in any other face than $\mathcal{P}$ are called *facets*.

Let $\mathcal{F}_i$ and $\mathcal{F}_j$ be two adjacent facets defined by $\mathcal{H}_i$ and $\mathcal{H}_j$, and let $\mathcal{F}_{ij}$ be their intersection. We can easily define a supporting hyperplane which includes $\mathcal{F}_{ij}$ but is distinct from both $\mathcal{H}_i$ and $\mathcal{H}_j$, and thus show that $\mathcal{F}_{ij}$ is another valid face of $\mathcal{P}$. For example, such a hyperplane can be constructed by rotating either $\mathcal{H}_i$ or $\mathcal{H}_j$ along their intersection $\mathcal{F}_{ij}$. We can proceed exploring new faces by intersecting already known ones until we reach the point where no new faces can be found. In the end, a hierarchical structure of faces emerges where each face is either a facet or is included in some other face. Faces which do not contain any other faces are called *minimal*.

As a consequence of the described face exploration procedure, each face $\mathcal{F}_{\mathcal{Z}}$ of $\mathcal{P}$ can be implicitly defined by a subset $\mathcal{Z} \subset \mathbb{N}_{\leq n}$ indexing those supporting hyperplanes $\mathcal{H}_i$ that $\mathcal{F}_{\mathcal{Z}}$ lies on in the sense that

$$\mathcal{F}_{\mathcal{Z}} = \{ x \in \mathcal{P} : \forall i \in \mathcal{Z} \, (a_i x = b_i) \} \quad . \tag{2.11}$$

According to (2.11), each $\mathcal{F}_{\mathcal{Z}}$ is again a nonempty polyhedron. The dimension of $\mathcal{F}_{\mathcal{Z}}$ is the dimension of its affine hull as given in (2.9); hence,

$$\mathbf{dim} \, (\mathcal{F}_{\mathcal{Z}}) = d - \mathit{rank} [A_{\mathcal{Z}}] \quad .$$

Obviously, a face of dimension zero is a minimal one. Such faces are also called *vertices*. A face of dimension one corresponds to a fragment of a line. If the fragment is bounded by two vertices, then the face is called an *edge*. The bounding vertices are called *adjacent*. If the fragment is bounded by only one vertex and hence a half-line, then the face is called an *extreme ray*.

Faces of Cones. Let $\mathcal{C}\,(A)$ be a polyhedral cone generated by a minimal set X. As a consequence of (2.11), each face of $\mathcal{C}\,(A)$

$$\mathcal{F}_{\mathcal{Z}} = \{ x \in \mathcal{C}\,(A) : \forall i \in \mathcal{Z} \, (a_i x = 0) \} \tag{2.12}$$

is again a polyhedral cone. Moreover, if $X' \subseteq X$ is the maximal subset of generators lying on $\mathcal{F}_{\mathcal{Z}}$, then $\mathcal{F}_{\mathcal{Z}}$ is generated by X'; that is,

$$X' = X \cap \mathcal{F}_{\mathcal{Z}} \Leftrightarrow \mathcal{F}_{\mathcal{Z}} = \mathtt{cone} \left(X' \right) \quad . \tag{2.13}$$

Each extreme ray of $\mathcal{C}\,(A)$

$$r_i = \mathtt{cone} \, (\{ x_i \})$$

corresponds to a single generator in $X = \{ x_1, \ldots, x_m \}$. Furthermore, each extreme ray is bounded by the origin, which is a trivial vertex of the cone. Although all extreme rays share that trivial vertex, we shall view them as the minimal faces of $\mathcal{C}\,(A)$. As a consequence of (2.12) and (2.13), each face of $\mathcal{C}\,(A)$ has a unique representation either as a set of generators which it contains or as a set of facets in which it is contained.

Faces of Polytopes. Let $\mathcal{P}\left(A, b\right)$ be the convex hull of X and hence a polytope. Then, each face $\mathcal{F}_{\mathcal{Z}}$ of $\mathcal{P}$ as given in (2.11) is a again polytope. Moreover, $\mathcal{F}_{\mathcal{Z}}$ is the convex hull of some subset $X' \subseteq X$ of the initial vector set, i.e.

$$X' = X \cap \mathcal{F}_{\mathcal{Z}} \Leftrightarrow \mathcal{F}_{\mathcal{Z}} = \texttt{conv.hull}(X') \ .$$

Assuming that X is minimal, each $x \in X$ is a vertex of $\mathcal{P}$. Consequently, each face of $\mathcal{P}$ has a unique representation either as the set of vertices which it contains or the set of facets in which it is contained.

Face Lattice. The set of all faces of a polyhedron $\mathcal{P}$ together with the empty set

$$L_{\mathcal{F}} = \{\mathcal{F} \subseteq \mathcal{P} : \mathcal{F} \text{ is a face of } \mathcal{P} \text{ or } \mathcal{F} = \emptyset\}$$

forms a partially ordered set $(L_{\mathcal{F}}, \subseteq)$ relative to inclusion. This fact is a direct implication from the face definition given in (2.11). Furthermore, for each two faces $\mathcal{F}'$ and $\mathcal{F}''$ there exists

an *infimum* $\mathcal{F}' \wedge \mathcal{F}''$ which is either the largest face contained in both $\mathcal{F}'$ and $\mathcal{F}''$, or the empty set if no such face exists, and

a *supremum* $\mathcal{F}' \vee \mathcal{F}''$ which is the smallest face containing both $\mathcal{F}'$ and $\mathcal{F}''$.

Consequently, $(L_{\mathcal{F}}, \vee, \wedge)$ is a lattice, also called the *face lattice* of $\mathcal{P}$. Moreover, each face is the supremum of the minimal faces which it includes and the infimum of the facets in which it resides. Those two properties make $(L_{\mathcal{F}}, \vee, \wedge)$ *atomic* and *coatomic* respectively [107].

Each lattice has an isomorphic graphical representation known as an *Hasse diagram.* In Figure 2.2, a three-dimensional simplex (see Definition 2.5) is given together with the Hasse diagram of its face lattice. Lattices and their graphical representations are described in greater detail by Birkhoff [26] and Davey and Priestley [59].

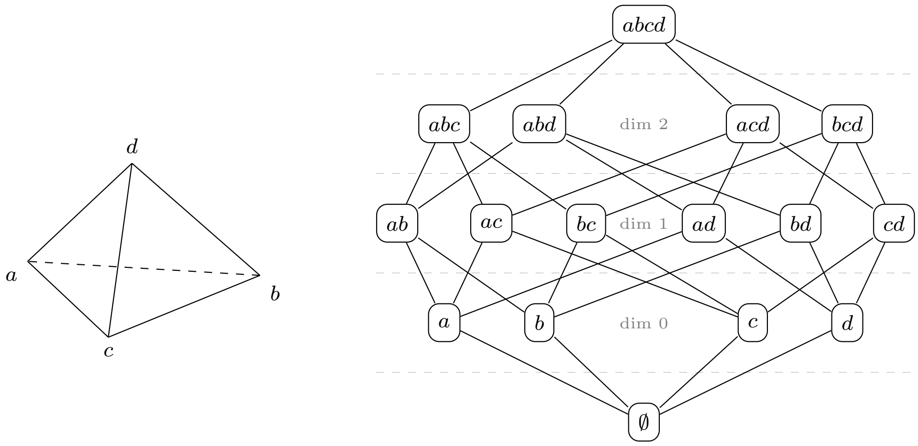
Definition 2.5 [Simplex]**.** A d-dimensional polytope with $d + 1$ vertices is called a *simplex.*

Incidence Graph. The *incidence graph* (aka *vertex* or *edge graph*) of the polytope $\mathcal{P}$ is the undirected graph $\mathit{\Gamma}\left(\mathcal{P}\right) = \left(V, E\right)$ with

V containing all vertices of $\mathcal{P}$ and

$E = \{\left(v', v''\right) \mid \mathbf{dim}\left(v' \vee v''\right) = 1\}$ containing all edges.

Consequently, the incidence graph introduces another possible representation of $\mathcal{P}$ which extends its $\mathcal{V}$-representation with additional information about adjacency of the vertices.



(a) 3-Simplex

(b) Face Lattice

Figure 2.2: The face lattice of a 3-Simplex

2.3 Duality

Let $X \subset \mathbb{R}^d$ be a finite vector set. The set

$$X^\star = \Big\{ y \in \mathbb{R}^d : \forall x \in X \left(x^T y \leq 1 \right) \Big\} \tag{2.14}$$

is called *dual* to X. The concept of the duality has immediate consequences for polyhedra, assigning to each polyhedron a dual one (see Section 2.3.1). It also implies an isomorphic mapping between the faces of dual polytopes (see Section 2.3.2) and dual polyhedral cones (see Section 2.3.3).

2.3.1 Properties of Duality

Each vector $x \in X - \{0\}$ is mapped to an affine halfspace

$$\mathscr{hs}(x) = \big\{ y \mid x^T y \leq 1 \big\} \qquad \text{with } X^\star = \bigcap_{x \in X} \mathscr{hs}(x) \tag{2.15}$$

the closure of all affine halfspaces and hence a convex polyhedron. Let

$$\mathscr{hp}(x) = \big\{ y \mid x^T y = 1 \big\}$$

be a further mapping of x to the hyperplane bounding $\mathscr{hs}(x)$. The following geometric properties apply to each vector $x \in X - \{0\}$.

(i) The line $L = \texttt{lin.hull}(\{x\})$, which crosses the origin and x, is orthogonal to $\mathscr{hp}(x)$.

(ii) If $\|x\|$ is the distance from x to the origin, then $\|x\|^{-1}$ is the distance from $\mathit{hp}\,(x)$ to the origin. Consequently, $\mathit{hs}\,(x)$ encloses x if and only if $\|x\| \leq 1$. Furthermore, for all $\lambda \geq 1$,

$$x' = \lambda x \Rightarrow \mathit{hs}\,\big(x'\big) \subseteq \mathit{hs}\,(x) \quad .$$

(iii) The halfspace $\mathit{hs}\,(x)$ always encloses the origin as $y = 0$ is a trivial solution in (2.14) and hence an element of $X^\star$.

In Figure 2.3, the above properties of duality are illustrated for $\mathbb{R}^3$.

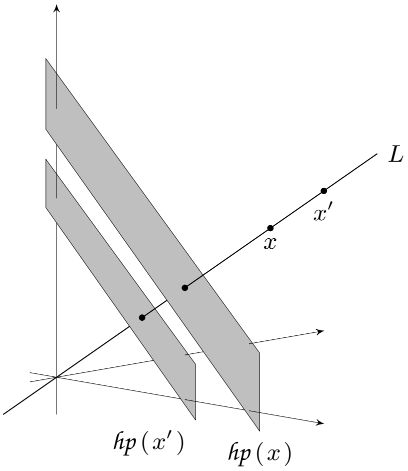


Figure 2.3: Properties of duality

Let $\mathcal{P}$ be a polyhedron containing the origin. As a consequence of (2.15), $\mathcal{P}^\star$ is a polyhedron as well. An important phenomenon arises when building $\mathcal{P}^{\star\star}$, the dual of $\mathcal{P}^\star$. It is easy to show that $\mathcal{P}$ and $\mathcal{P}^{\star\star}$ are identical (see Theorem 2.3).

Theorem 2.3 [Duality of Polyhedra]**.** *Let $\mathcal{P}$ be a polyhedron such that $\mathcal{P} = \mathcal{P} \cup \{0\}$. Then $\mathcal{P}^{\star\star} \; = \; \mathcal{P}$.*

Proof. Following Schrijver [131], it is easy to show that $\mathcal{P} \subseteq \mathcal{P}^{\star\star}$ (i) and $\mathcal{P}^{\star\star} \subseteq \mathcal{P}$ (ii).

(i) $\forall x \in \mathcal{P}\, \forall y \in \mathcal{P}^\star \left(x^T y \leq 1\right) \Rightarrow \forall x \in \mathcal{P}\, \forall y \in \mathcal{P}^\star \left(y^T x \leq 1\right)$
$\Rightarrow \forall x \in \mathcal{P}\, (x \in \mathcal{P}^{\star\star}) \Rightarrow \mathcal{P} \subseteq \mathcal{P}^{\star\star}$.

(ii) First, assume that $\mathcal{P}^{\star\star} \not\subseteq \mathcal{P}$. As a consequence, there exists a constraint $\alpha x \leq \beta$ which holds on $\mathcal{P}$ but not on $\mathcal{P}^{\star\star}$; hence,

$$\exists x' \in \mathcal{P}^{\star\star} \left(\alpha x' > \beta\right) \quad . \tag{2.16}$$

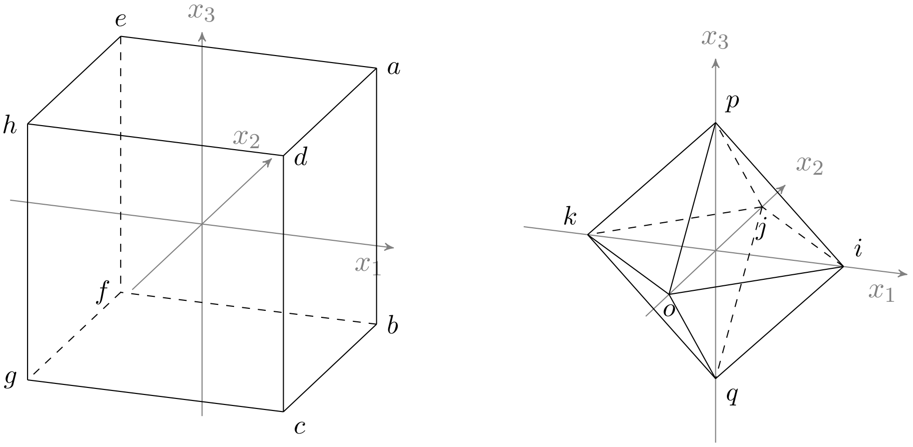
Since $\mathcal{P}$ contains the origin, it is guaranteed that $\beta \geq 0$. Next, the following two cases are to be considered.

If $\beta > 0$, then $\forall x \in \mathcal{P} \left(\beta^{-1}(\alpha x) \leq 1\right) \Rightarrow \forall x \in \mathcal{P} \left(x^T(\beta^{-1}\alpha^T) \leq 1\right)$
$\Rightarrow (\beta^{-1}\alpha^T) \in \mathcal{P}^\star \Rightarrow (\beta^{-1}\alpha)x' \leq 1$ which contradicts with (2.16).

If $\beta = 0$, then $\forall x \in \mathcal{P}\left(\alpha x \leq 0\right) \Rightarrow \forall x \in \mathcal{P}\left(x^T \alpha^T \leq 0\right)$
$\Rightarrow \forall \lambda \geq 0\left(\lambda \alpha^T \in \mathcal{P}^\star\right) \Rightarrow \forall \lambda \geq 0\left((\lambda\alpha)x' \leq 1\right) \Rightarrow \forall \lambda \geq 0\left(\alpha x' \leq \lambda^{-1}\right)$ which again contradicts with (2.16) as $\lim\limits_{\lambda \to \infty} \lambda^{-1} = 0$ and hence $\alpha x' \leq 0$.

□

In Figure 2.4, two dual polyhedra are illustrated.



(a) Cube

(b) Regular Octahedron

Figure 2.4: Pair of dual polyhedra

2.3.2 Duality of Polytopes

Let $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$ be a finite set of vectors with a convex hull equivalent to the d-dimensional polytope $\mathcal{P}\left(A, b\right)$. Furthermore, assume the following preconditions. First, X is minimal in the sense that each vector in X is a vertex of $\mathcal{P}$. Second, $\mathcal{P}$ encloses the origin. Third, the origin does not lie on any facet of $\mathcal{P}$; that is, each ray starting at the origin and shooting in any direction intersects a facet of $\mathcal{P}$ in a point distinct from the origin.

Each x_i is a zero-dimensional polyhedron which is dual to the affine halfspace $\mathit{hs}\left(x_i\right)$. The intersection of all those affine halfspaces

$$\mathcal{P}' = \bigcap_{i=1}^{m} \mathit{hs}\left(x_i\right) = \bigcap_{i=1}^{m} \left\{y \mid x_i^T y \leq 1\right\} \tag{2.17}$$

is obviously another polyhedron and has the following properties:

(i) $\mathcal{P}' = \mathcal{P}^\star$ (see Lemma 2.4),

(ii) $\mathcal{P}'$ is a polytope (see Lemma 2.5),

(iii) the facets of $\mathcal{P}$ are isomorphic to the vertices of $\mathcal{P}'$ (see Theorem 2.7), and

(iv) the vertices of $\mathcal{P}$ are isomorphic to the facets of $\mathcal{P}'$ (see Corollary 2.8).

Lemma 2.4. *The dual set of a nonempty finite vector set $X \subset \mathbb{R}^d$ is equivalent to the dual set of its convex hull.*

Proof. Let $X = \{x_1, \dots, x_m\}$. From (2.14), it follows that

$$\begin{aligned}
&\forall y \in X^\star \left(\bigwedge_{i=1}^{m} x_i^T y \le 1 \right) \\
\Rightarrow\ &\forall y \in X^\star \left(\bigwedge_{i=1}^{m} \lambda_i x_i^T y \le \lambda_i \right) \quad \text{with } \lambda_i \in \mathbb{R}, \lambda_i \ge 0 \text{ and } \sum_{i=1}^{m} \lambda_i = 1 \\
\Rightarrow\ &\forall y \in X^\star \left((\sum_{i=1}^{m} \lambda_i x_i)^T y \le 1 \right) \quad \text{with } \lambda_i \in \mathbb{R}, \lambda_i \ge 0 \text{ and } \sum_{i=1}^{m} \lambda_i = 1 \\
\Rightarrow\ &\forall y \in X^\star \left(x^T y \le 1 \right) \quad \text{with } x \in \texttt{conv.hull}(X) \\
\Rightarrow\ &X^\star \subseteq \texttt{conv.hull}(X)^\star \ .
\end{aligned}$$

It is easy to show that $X^\star \supseteq \texttt{conv.hull}(X)^\star$ as $\texttt{conv.hull}(X)^\star$ is bounded by all constraints which are also valid for $X^\star$. □

Lemma 2.5. *The dual polyhedron of a polytope enclosing the origin is another polytope.*

Proof. First, it is easy to verify the following equivalence:

$$\mathcal{P} \text{ is bounded} \Leftrightarrow \exists q \in \mathbb{R} \left(q \ge 0 \land \forall x, x' \in \mathcal{P} \ \left(x^T x' \le q \right) \right) \ . \tag{2.18}$$

In other words, if $\mathcal{P}$ is bounded, then for each $x \in \mathcal{P}$, we can find a hyperplane which

(i) is orthogonal to the ray starting at the origin and crossing x, and

(ii) defines a halfspace containing the whole polytope $\mathcal{P}$.

Next, as the origin is enclosed by $\mathcal{P}$, we can define a ray in any possible direction by using the origin as a starting point and some vector from $\mathcal{P}$ as a direction. This allows us to express each vector of $\mathcal{P}^\star$ as a vector of $\mathcal{P}$ multiplied with some positive scalar; that is, for each $y \in \mathcal{P}^\star$,

$$\exists x \in \mathcal{P}\, \exists \lambda \ge 0 \, (y = \lambda x) \ .$$

Consequently, for all $y, y' \in \mathcal{P}^\star$,

$$y^T y' = (\lambda x^T)(\lambda' x') = \lambda\lambda' x^T x' \le q' \quad \text{for some sufficiently large } q' \in \mathbb{R} \ .$$

According to (2.18), $\mathcal{P}^\star$ is then bounded as well. □

Lemma 2.6. *There is a homomorphic mapping from $\mathcal{P}$'s facets to $\mathcal{P}^\star$'s vertices.*

Proof. Each supporting hyperplane of $\mathcal{P}$

$$\mathcal{H}_i = \{x \mid a_i x = b_i\} \qquad \text{with } i \in [1, n]$$

corresponds to a vector $y_i = b_i^{-1} a_i^T$. Furthermore, for each $x \in \mathcal{P}$,

$$x^T y_i = x^T (b_i^{-1} a_i^T) = b_i^{-1} a_i x \leq 1$$

and thus $y_i \in \mathcal{P}^\star$. The vertices of $\mathcal{P}$ lying on each $\mathcal{H}_i$ define a polytope of dimension $d-1$; hence, for each $\mathcal{H}_i$, there exists a unique maximal subset $X_i \subset X$ with at least d linear independent elements and

$$\bigwedge_{x \in X_i} a_i x = b_i \Leftrightarrow \bigwedge_{x \in X_i} x^T (b_i^{-1} a_i^T) = 1 \Leftrightarrow \bigwedge_{x \in X_i} x^T y_i = 1 \ .$$

It is evident that y_i is a vertex of $\mathcal{P}^\star$ in that case. □

Theorem 2.7. *There is an isomorphism between the facets of $\mathcal{P}$ and the vertices of $\mathcal{P}^\star$.*

Proof. The homomorphism from facets to vertices was demonstrated in Lemma 2.6. By reversing the proof given there, it can be easily shown that the mapping is isomorphic. Let y' be a vertex of $\mathcal{P}^\star$. Then according to the construction of $\mathcal{P}^\star$ given in (2.17), y' emerges from the intersection of at least d hyperplanes

$$\big\{ y \mid x^T y = 1 \text{ for some } x \in X \big\} \ .$$

Hence, there exists a subset $X' \subset X$ such that

$$\mathbf{dim}\big(\texttt{aff.hull}(X')\big) = d - 1$$

and

$$\bigwedge_{x' \in X'} x'^T y' = 1 \text{ and } \bigwedge_{x \in X - X'} x^T y' \leq 1 \ .$$

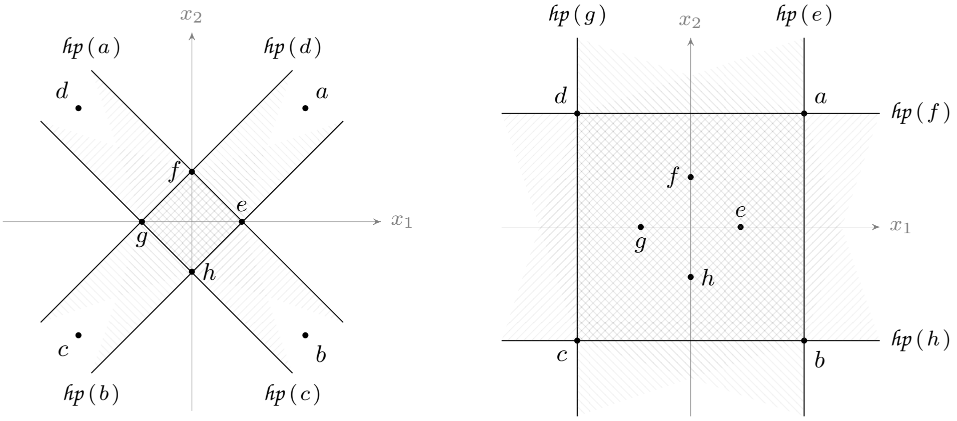
The latter can be strengthened to

$$\bigwedge_{x \in X - X'} x^T y' < 1$$

by maximizing X'; that is, by transferring to X' all vectors from $X - X'$ which lie within the affine hull of X'. The convex hull of X' is then a $d-1$ dimensional polytope which is a facet of $\mathcal{P}$. Consequently, there exists a supporting hyperplane $\mathcal{H}_i$ which defines it. □

Corollary 2.8. *There is an isomorphism between the vertices of $\mathcal{P}$ and the facets of $\mathcal{P}^\star$.*

Proof. Follows from Theorems 2.3 and 2.7. □



(a) Vertices of $\mathcal{P}$ and facets of $\mathcal{P}^\star$

(b) Vertices of $\mathcal{P}^\star$ and facets of $\mathcal{P}$

Figure 2.5: Isomorphic vertices and facets of dual polytopes

Example 2.1. Consider the set $X = \{a, b, c, d\} \subset \mathbb{R}^2$ where

$$a = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 2 \\ -2 \end{bmatrix}, c = \begin{bmatrix} -2 \\ -2 \end{bmatrix} \text{ and } d = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

as shown in Figure 2.5a. Let $\mathcal{P}$ denote the convex hull of X. The polytope $\mathcal{P}^\star$ is then the intersection of four halfspaces, i.e.

$$\mathcal{P}^\star = X^\star = \mathit{hs}(a) \cap \mathit{hs}(b) \cap \mathit{hs}(c) \cap \mathit{hs}(d) = \left\{ y \mid \begin{bmatrix} 2 & 2 \\ 2 & -2 \\ -2 & -2 \\ -2 & 2 \end{bmatrix} y \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\}$$

with

$$e = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}, f = \begin{bmatrix} 0 \\ 1/2 \end{bmatrix}, g = \begin{bmatrix} -1/2 \\ 0 \end{bmatrix} \text{ and } h = \begin{bmatrix} 0 \\ -1/2 \end{bmatrix}$$

the vertices of $\mathcal{P}^\star$. As a consequence, each vertex of $\mathcal{P}$ can be mapped to a facet of $\mathcal{P}^\star$ as follows:

$$a \mapsto \texttt{conv.hull}(\{e, f\}),\ b \mapsto \texttt{conv.hull}(\{h, e\}),$$
$$c \mapsto \texttt{conv.hull}(\{g, h\}),\ d \mapsto \texttt{conv.hull}(\{g, f\})\ .$$

A similar situation arises when building $\mathcal{P}$ out of $\mathcal{P}^\star$'s vertices (see Figure 2.5b). According to (2.17),

$$\mathcal{P} = \mathit{hs}(e) \cap \mathit{hs}(f) \cap \mathit{hs}(g) \cap \mathit{hs}(h) = \left\{ x \mid \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \\ -1/2 & 0 \\ 0 & -1/2 \end{bmatrix} x \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\}$$

where each vertex of $\mathcal{P}^\star$ defines a hyperplane which supports $\mathcal{P}$. For example, e defines $\mathit{hp}\,(e) = \left\{x \mid \begin{bmatrix}1 & 0\end{bmatrix} x = 2\right\}$, f defines $\mathit{hp}\,(f) = \left\{x \mid \begin{bmatrix}0 & 1\end{bmatrix} x = 2\right\}$, etc. Moreover, each such hyperplane contains a unique adjacent pair of $\mathcal{P}$'s vertices which yields the following isomorphic mapping:

$$e \mapsto \texttt{conv.hull}(\{a, b\}),\ f \mapsto \texttt{conv.hull}(\{d, a\}),$$
$$g \mapsto \texttt{conv.hull}(\{c, d\}),\ h \mapsto \texttt{conv.hull}(\{b, c\})\ .$$

2.3.3 Duality of Cones

Consider again the finite vector set $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$, but this time from another perspective. Let $\mathcal{C}\,(A)$ be the cone generated by X. Without loss of generality, assume that $\mathcal{C}$ is both full-dimensional and pointed, and that X represents a minimal set of generators. Thus, each x_i defines an extreme ray

$$r_i = \texttt{cone}\,(\{x_i\}) = \{\lambda x_i \mid \lambda \in \mathbb{R}, \lambda \geq 0\} \tag{2.19}$$

starting at the origin and crossing x_i. By definition, each such extreme ray is a polyhedron which contains the origin and has the halfspace

$$r_i^\star = \texttt{cone}\,(\{x_i\})^\star = \left\{y \mid x_i^T y \leq \lambda^{-1}, \lambda \in \mathbb{R}, \lambda \geq 0\right\} \tag{2.20}$$

as a dual set. The intersection of all such halfspaces

$$\mathcal{C}' = \bigcap_{i=1}^{m} r_i^\star \tag{2.21}$$

yields another polyhedron with the following properties:

(i) $\mathcal{C}'$ is a polyhedral cone as each $r_i^\star$ is a linear halfspace (see Lemma 2.9),

(ii) $\mathcal{C}' = \mathcal{C}^\star$ (see Lemma 2.10),

(iii) the facets of $\mathcal{C}$ are isomorphic to the extreme rays of $\mathcal{C}'$ (see Theorem 2.12) and

(iv) the extreme rays of $\mathcal{C}$ are isomorphic to the facets of $\mathcal{C}'$ (see Corollary 2.13).

In Example 2.2, the above properties are illustrated for a pair of dual polyhedral cones in $\mathbb{R}^3$.

Lemma 2.9. *The dual polyhedron of an extreme ray is a linear halfspace.*

Proof. Consider the dual polyhedron $r_i^\star$ as given in (2.20). It is evident that

$$\lim_{\lambda \to \infty} \left(\lambda^{-1}\right) = 0$$

which allows the elimination of λ in (2.20) and hence an alternative representation

$$r_i^\star = \left\{y \mid x_i^T y \leq 0\right\}\ .$$

In other words, for each λ' and y' with $x_i^T y' \leq \lambda'^{-1}$ and $x_i^T y' > 0$, we can always find another $\lambda'' > \lambda'$ such that $x_i^T y' > \lambda''^{-1}$ and thereby prove that $y' \notin r_i^\star$. □

Lemma 2.10. *Let $X \subset \mathbb{R}^d$ be a finite set of vectors. Then*

$$\mathtt{cone}\,(X)^\star = \bigcap_{x \in X} \mathtt{cone}\,(\{x\})^\star \quad .$$

Proof. Let $X = \{x_1, \ldots, x_m\}$. Then

$$\begin{aligned}\bigcap_{i=1}^{m} \mathtt{cone}\,(\{x_i\})^\star &= \bigcap_{i=1}^{m} \{\lambda_i x_i \mid \lambda_i \geq 0\}^\star = \bigcap_{i=1}^{m} \{y \mid \lambda_i x_i^T y \leq 1, \lambda_i \geq 0\} \\ &= \left\{ y \mid \bigwedge_{i=1}^{m} \lambda_i x_i^T y \leq 1, \lambda_i \geq 0 \right\} \\ &\subseteq \left\{ y \mid \sum_{i=1}^{m} \lambda_i x_i^T y \leq m, \lambda_i \geq 0 \right\} \\ &= \left\{ y \mid \sum_{i=1}^{m} \lambda_i' x_i^T y \leq 1, \lambda_i' = \lambda_i / m, \lambda_i \geq 0 \right\} \\ &= \mathtt{cone}\,(X)^\star \quad .\end{aligned}$$

It is also evident that

$$\mathtt{cone}\,(X)^\star \subseteq \bigcap_{i=1}^{m} \mathtt{cone}\,(\{x_i\})^\star$$

as for each $x_i \in X$, $\mathtt{cone}\,(X)^\star \subseteq \mathtt{cone}\,(\{x_i\})^\star$. □

Lemma 2.11. *There is a homomorphism which maps each facet of $\mathcal{C}$ to an extreme ray of $\mathcal{C}^\star$.*

Proof. Each supporting hyperplane of $\mathcal{C}$

$$\mathcal{H}_i = \{x \mid a_i x = 0\} \tag{2.22}$$

defines a facet. Thus, $\mathcal{H}_i$ contains at least $d-1$ extreme rays of $\mathcal{C}$ which generate a $d-1$ dimensional polyhedral cone; hence, for each $i \in [1, n]$ there exists a unique $X_i \subset X$ such that

$$\forall x \in X_i\,(a_i x = 0) \wedge \forall x \in X - X_i\,(a_i x < 0)$$

and

$$\mathbf{dim}\,(\mathtt{cone}\,(X_i)) = \mathbf{dim}\,(\mathtt{lin.hull}(X_i)) = d - 1 \quad .$$

Consequently, it is easy to verify that $\mathtt{cone}\left(\left\{a_i^T\right\}\right)$ is an extreme ray of $\mathcal{C}^\star$ as for each $y \in \mathtt{cone}\left(\left\{a_i^T\right\}\right)$,

$$\forall x \in X_i\left(x^T y = 0\right) \wedge \forall x \in X - X_i\left(x^T y < 0\right)$$

and

$$\mathbf{dim}\left(\mathtt{cone}\left(\left\{a_i^T\right\}\right)\right) = d - \mathbf{dim}\,(\mathtt{lin.hull}(X_i)) = 1 \quad .$$

□

Theorem 2.12. *There is an isomorphic mapping from $\mathcal{C}$'s facets to $\mathcal{C}^\star$'s extreme rays.*

Proof. In what follows, it is shown that the homomorphism from Lemma 2.11 is indeed an isomorphism. Each extreme ray of $\mathcal{C}^\star$ emerges from the intersection of $d-1$ facets, thus for some $X' \subset X$,

$$r' = \bigcap_{x \in X'} \{ y \mid x^T y = 0 \} = \left\{ y \mid \bigwedge_{x \in X'} x^T y = 0 \right\}$$

is an extreme ray of $\mathcal{C}^\star$ if for all $y \in r'$,

$$\forall x \in X - X' \left(x^T y \leq 0 \right) \text{ and } \mathbf{dim}\left(\texttt{lin.hull}(X') \right) = d - 1 \ .$$

Moreover, we can enforce $x^T y < 0$ for all $x \in X - X'$ by maximizing X' without changing the dimension of its linear hull. Consequently, $\texttt{cone}\,(X')$ is a facet of $\mathcal{C}$ and thus there exists a hyperplane $\mathcal{H}_i$ as given in (2.22) which defines it. □

Corollary 2.13. *There is an isomorphic mapping from $\mathcal{C}$'s extreme rays to $\mathcal{C}^\star$'s facets.*

Proof. Follows directly from Theorems 2.3 and 2.12. □

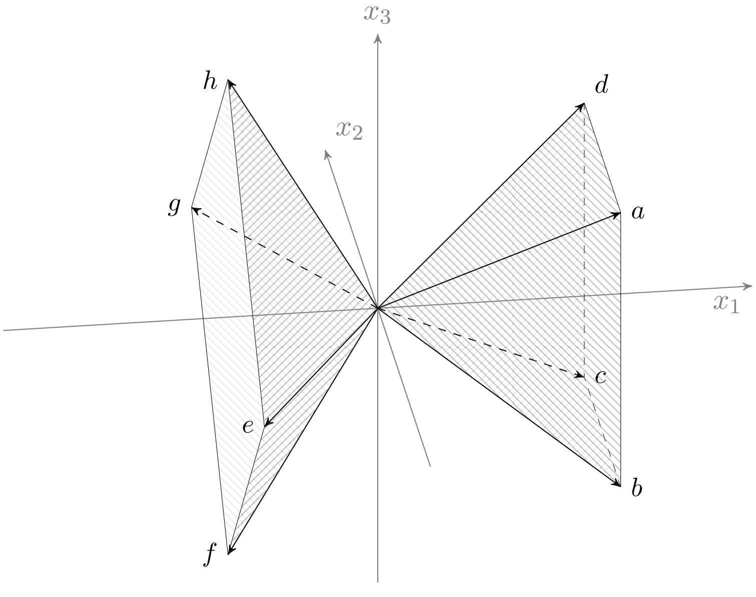


Figure 2.6: Pair of dual polyhedral cones

Example 2.2. In Figure 2.6, the polyhedral cone

$$\mathcal{C} = \left\{ y \mid \begin{bmatrix} -1 & -2 & 0 \\ -1 & 0 & -2 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{bmatrix} y \leq 0 \right\}$$

is illustrated. It is generated by the vectors

$$a = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}, b = \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix}, c = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} \text{ and } d = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} \quad .$$

The dual cone of $\mathcal{C}$ is then

$$\mathcal{C}^\star = \left\{ y \mid \begin{bmatrix} 2 & -1 & 1 \\ 2 & -1 & -1 \\ 2 & 1 & -1 \\ 2 & 1 & 1 \end{bmatrix} y \leq 0 \right\}$$

with generators

$$e = \begin{bmatrix} -1 \\ -2 \\ 0 \end{bmatrix}, f = \begin{bmatrix} -1 \\ 0 \\ -2 \end{bmatrix}, g = \begin{bmatrix} -1 \\ 2 \\ 0 \end{bmatrix} \text{ and } h = \begin{bmatrix} -1 \\ 0 \\ 2 \end{bmatrix} \quad .$$

The isomorphic mapping between $\mathcal{C}$'s facets and $\mathcal{C}^\star$'s extreme rays is defined by

$$\texttt{cone}\left(\{a, b\}\right) \mapsto \texttt{cone}\left(\{e\}\right), \ \texttt{cone}\left(\{b, c\}\right) \mapsto \texttt{cone}\left(\{f\}\right),$$
$$\texttt{cone}\left(\{c, d\}\right) \mapsto \texttt{cone}\left(\{g\}\right), \ \texttt{cone}\left(\{d, a\}\right) \mapsto \texttt{cone}\left(\{h\}\right) \quad .$$

Analogously, we obtain the isomorphic mapping between $\mathcal{C}$'s extreme rays and $\mathcal{C}^\star$'s facets:

$$\texttt{cone}\left(\{a\}\right) \mapsto \texttt{cone}\left(\{h, e\}\right), \ \texttt{cone}\left(\{b\}\right) \mapsto \texttt{cone}\left(\{e, f\}\right),$$
$$\texttt{cone}\left(\{c\}\right) \mapsto \texttt{cone}\left(\{f, g\}\right), \ \texttt{cone}\left(\{d\}\right) \mapsto \texttt{cone}\left(\{g, h\}\right) \quad .$$

2.4 Degeneracy

Let $X \subset \mathbb{R}^d$ be a finite set of points. In computational geometry, assuming that all points in X are in *general position* means that *they avoid the troublesome configurations, known as degenerate situations* [60, p. 40]. In other words, there are no more than two points in X lying on the same line, no more than three lying on the same plane, etc. A lot of algorithms are based on this assumption as it effectively allows authors to focus on the general problem without the necessity of investigating numerous side cases. However, those degenerate cases cannot be neglected as otherwise the affected algorithms would remain incomplete and hence hardly applicable in practice. Moreover, many important geometric problems turn out to be degenerate. In that regard, a bibliography referring to all kinds of such problems appearing in various fields was compiled by Gal [80].

In the context of convex polyhedra, degeneracy is related to the number of facets on which each vertex lies. The following definition originates (in a slightly different form) from the work of Tijssen and Sierksma [146].

Definition 2.6 [Degeneracy Degree]**.** Given a polyhedron $\mathcal{P}(A,b)$ with $A \in \mathbb{R}^{n \times d}$, let $Z \subset \mathbb{N}_{\leq n}$ be maximal with respect to $A_Z v = b_Z$ for some vertex v. The *degeneracy degree* of v,

$$\sigma(v) = \|Z\| - d \tag{2.23}$$

is the number of redundant hyperplanes defining v.

As a consequence of Definition 2.6, the vertex v is called degenerate if $\sigma(v) > 0$ and nondegenerate otherwise. Furthermore, a definition equivalent to (2.23) can be given for each extreme ray r of $\mathcal{P}(A,b)$. The degeneracy degree of r is then

$$\sigma(r) = \|Z'\| - (d-1)$$

with $Z' \subset \mathbb{N}_{\leq n}$ maximal with respect to $A_{Z'}x = 0$ for all $x \in r$. Again, the extreme ray r is degenerate if $\sigma(r) > 0$ and nondegenerate otherwise. Consequently, the polyhedron $\mathcal{P}(A,b)$ is nondegenerate if and only if all of its vertices and extreme rays are nondegenerate.

Nondegerate polytopes, also called *simple* (see Definition 2.7), have some very distinguishing properties. Each vertex x lies on exactly d edges. Moreover, intersecting any $d-1$ facets on which x lies automatically yields one of those edges. As we shall see later, those properties are of great importance when dealing with the convex hull problem. It should be pointed out that the dual of a simple polytope is not necessarily simple as well. Such polytopes have other properties and are called *simplicial* (see Definition 2.8).

Definition 2.7 [Simple Polytope]**.** A d-dimensional polytope is called simple if and only if each vertex lies on exactly d facets.

Definition 2.8 [Simplicial Polytope]**.** A d-dimensional polytope is called simplicial if and only if each facet contains exactly d vertices.

Perturbation. As already stated, many geometric algorithms assume general position of the input. Then an immediately arising question is how the degenerate cases are being solved. We can think of each degenerate vertex as a set of nondegenerate ones which have the same coordinates and hence zero distance between each other. In order to distinguish them as independent points, we can require that the distance between each two vertices is nonzero. Such a technique for transforming a degenerate polyhedron into a nondegenerate one is called *perturbation*.

Let v_0 be some arbitrary vertex such that $A_Z v_0 = b_Z$ where A_Z is a rectangular matrix. If there exists another row vector a_j of A such that $a_j v_0 = b_j$ and $j \notin Z$, then v_0 is obviously degenerate. Consequently, v_0 is degenerate if and only if there exists a vector $\beta \in \mathbb{R}^n$ satisfying

$$\sum_{i \in Z} \beta_i \begin{bmatrix} a_i^T \\ b_i \end{bmatrix} = \begin{bmatrix} a_j^T \\ b_j \end{bmatrix} \qquad \text{for some } j \in \mathbb{N}_{\leq n} - Z \ . \tag{2.24}$$

In the context of linear programming, Charnes [46] proposed a technique for ruling out the existence of linear dependencies as those given in (2.24) by perturbing the vector b. His approach involved replacing b by $b(\epsilon) = b + \epsilon$ where

$$\epsilon = \begin{bmatrix} \epsilon_1 & \epsilon_2 & \dots & \epsilon_n \end{bmatrix}^T$$

is a vector consisting of very small real numbers. Furthermore, each ϵ_i is significantly smaller than ϵ_{i-1} and ϵ_1 is significantly smaller than any other number coming up in the original problem. *The idea is that each ϵ_i acts on an entirely different scale from all the other ϵ_i´s and the data for the problem* [148, p. 33]. Consequently, it can be shown that for some sufficiently small ϵ_{max} and

$$\epsilon_{max} \gg \epsilon_1 \gg \epsilon_2 \gg \dots \gg \epsilon_n$$

the perturbed polyhedron $\mathcal{P}(A, b(\epsilon))$ contains no linear dependencies as given in (2.24) and hence no degeneracies. It should be noted, however, that each degenerate vertex v_0 yields up to $\binom{n+\sigma(v_0)}{\sigma(v_0)}$ nondegenerate ones; hence, the number of vertices may grow exponentially after perturbation. In Figure 2.7, the geometric consequences of perturbation are illustrated. A detailed explanation of Charnes' method was given by Hadley [85].

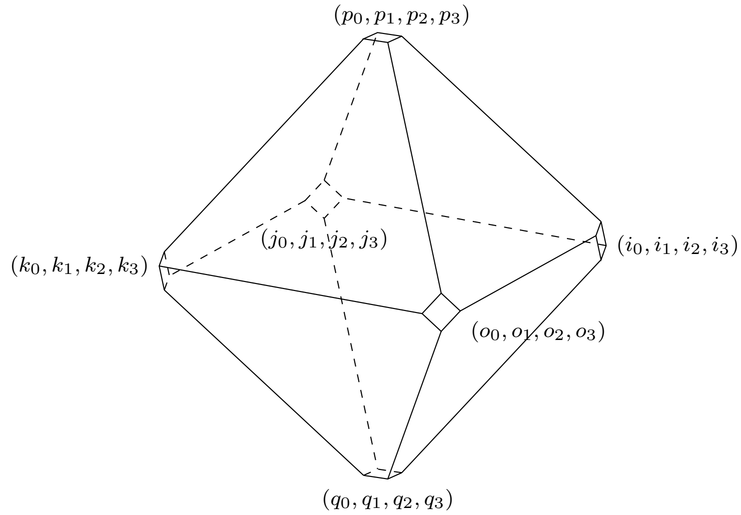


Figure 2.7: Perturbed octahedron

Several general perturbation schemes dealing with degeneracy in all kinds of geometric algorithms were developed. Such are the *simulation of simplicity* of Edelsbrunner and Mücke [66], the *symbolic scheme* of Yap [154] and the *linear scheme* of Emiris and Canny [67, 68], the latter being the most efficient one. In this context, Seidel [138] and Burnikel et al. [42] gave interesting discussions about the limitations of perturbation and to what extent it poses a general solution for degeneracy.

3 The Convex Hull Problem: Theory and Practice

The convex hull problem is one of the oldest and most intensively studied problems in computational geometry. It is related to finding a minimal set of halfspaces whose intersection is equivalent to the convex hull of finitely many vectors residing in the d-dimensional space (see Section 3.1). Despite the huge amount of work being done over the years, only the computational complexity of nondegenerate problems is known so far (see Section 3.2). A great number of algorithms have been proposed; however, none of them is able to solve degenerate problems in polynomial time. For nondegenerate ones, pivoting algorithms are able to deliver such a solution (see Section 3.3). Incremental algorithms, to which the double description method belongs, are characterized by an exponential complexity for both degenerate and nondegenerate problems (see Section 3.4).

3.1 Definition and Related Problems

Problem 3.1 [Convex Hull]. *Given a finite set of vectors $X \subset \mathbb{R}^d$, find a polytope in $\mathcal{H}$-representation equivalent to the convex hull of X.*

If X is minimal in the sense that excluding any x from X also changes the convex hull, then each vector in X is actually a vertex of the polytope. Consequently, the convex hull problem can be viewed as a more general form of *facet enumeration*.

Problem 3.2 [Facet Enumeration]. *Given a polytope $\mathcal{P}$ as a finite set of vertices $\mathcal{V}$, find a minimal set of supporting hyperplanes defining its facets.*

Suppose that we are given a polytope $\mathcal{P}$ in $\mathcal{H}$-representation. By means of duality, the $\mathcal{V}$-representation of the dual polytope $\mathcal{P}^\star$ is easily obtained. Solving the facet enumeration for $\mathcal{P}^\star$ automatically gives a $\mathcal{V}$-representation of $\mathcal{P}$; hence, we can apply duality to perform an implicit *vertex enumeration*. In that regard, we shall consider Problems 3.2 and 3.3 to be equivalent.

Problem 3.3 [Vertex Enumeration]. *Given a polytope $\mathcal{P}$ as an intersection of finitely many affine halfspaces, find all vertices of $\mathcal{P}$.*

Suppose now that $\mathcal{C}$ is a polyhedral cone given in $\mathcal{H}$-representation and that an algorithm to solve Problem 3.3 is available. Then, we can use that algorithm to perform an *extreme ray enumeration* for $\mathcal{C}$ (see Problem 3.4) by defining an additional affine halfspace $\{x \mid cx \leq 1\}$ which bounds $\mathcal{C}$ to a polytope (see Figure 3.1). Except the zero vector, each vertex of the so defined polytope corresponds to a single extreme ray of $\mathcal{C}$. It is also easy to show that the mapping is isomorphic.

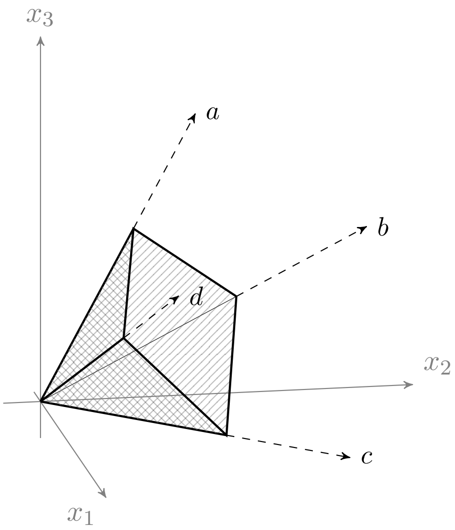


Figure 3.1: Transformation of a polyhedral cone into a polytope

If $\mathcal{C}$ is given in $\mathcal{V}$-representation instead, a facet enumeration (see Problem 3.5) is easily performed by first obtaining the dual cone $\mathcal{C}^\star$ and then enumerating its extreme rays as already described. Consequently, any algorithm to run on a polytope is easily extended to run on a polyhedral cone as well. The reverse implication is also valid. To each polytope $\mathcal{P}'$ with a vertex set $\mathcal{V}'$, we can assign a pointed polyhedral cone

$$\mathcal{C}' = \mathtt{cone}\left(\left\{\begin{bmatrix}1\\v\end{bmatrix} \mid v \in \mathcal{V}'\right\}\right)$$

by increasing the dimension of the problem. It is easy to verify that if

$$\left\{\begin{bmatrix}y\\x\end{bmatrix} \mid by - cx = 0\right\} \qquad \text{with } y \in \mathbb{R} \text{ and } x \in \mathbb{R}^d$$

defines a facet of $\mathcal{C}$, then $\{x \mid cx = b\}$ defines a facet of $\mathcal{P}'$. Thus, the facet enumeration of a polytope can be reduced to a facet enumeration of a polyhedral cone. By means of duality, the vertex enumeration is also reducible to an extreme ray enumeration.

Problem 3.4 [Extreme Ray Enumeration]**.** *Given a polyhedral cone $\mathcal{C}$ as an intersection of finitely many linear halfspaces, find a minimal set of generators X such that $\mathcal{C} = \mathtt{cone}(X)$.*

Problem 3.5 [Facet Enumeration of Cones]**.** *Given a finite set of vectors $X \subset \mathbb{R}^d$, find a minimal set of supporting hyperplanes defining the facets of $\mathtt{cone}(X)$.*

In conclusion, we shall say that Problems 3.2, 3.3, 3.4 and 3.5 are equivalent in the sense that an algorithm able to solve one of them is easily modifiable to solve any of them. This equivalence is illustrated by the problem cube in Figure 3.2. The availability of an algorithm to solve one of the four problems is sufficient to explore a path between nodes corresponding to $\mathcal{H}$- and $\mathcal{V}$-representations of a polytope or a cone. Such an algorithm can also be extended to solve the general convex hull problem; that is, to accept an input set $X \subset \mathbb{R}^d$ which is not necessarily minimal and thus contains not only vertices, but also other redundant vectors. The double description method, which is considered in this work, can handle such redundancies implicitly without additional modifications. Indeed, the same is true for the majority of available algorithms. In that regard, we can consider Problem 3.1 to have the same solution as the other four.

$$\begin{tikzcd}
 & \mathcal{H}(\mathcal{C}) \arrow[rr, leftrightarrow] \arrow[dl, leftrightarrow, "\text{Duality}"'] \arrow[dd, shift left, "\text{Extreme Ray Enumeration}"] & & \mathcal{H}(\mathcal{P}) \arrow[dl, leftrightarrow, "\text{Duality}"'] \arrow[dd, shift right, "\text{Vertex Enumeration}"'] \\
\mathcal{V}(\mathcal{C}^{\star}) \arrow[rr, leftrightarrow] \arrow[dd, shift left, "\text{Facet Enumeration}"] & & \mathcal{V}(\mathcal{P}^{\star}) \arrow[dd, shift left, "\text{Facet Enumeration}"] & \\
 & \mathcal{V}(\mathcal{C}) \arrow[rr, leftrightarrow] \arrow[dl, leftrightarrow, "\text{Duality}"] \arrow[uu, shift left, "\text{Facet Enumeration}"] & & \mathcal{V}(\mathcal{P}) \arrow[dl, leftrightarrow, "\text{Duality}"] \arrow[uu, shift right, "\text{Facet Enumeration}"'] \\
\mathcal{H}(\mathcal{C}^{\star}) \arrow[rr, leftrightarrow] \arrow[uu, shift left, "\text{Extreme Ray Enumeration}"] & & \mathcal{H}(\mathcal{P}^{\star}) \arrow[uu, shift left, "\text{Vertex Enumeration}"] &
\end{tikzcd}$$

Figure 3.2: Variations of the convex hull problem

In this work, a general differentiation between *primal* and *dual space* algorithms is made depending on the problem which they are designed to solve. A primal space algorithm solves Problems 3.1, 3.2 and 3.5; that is, it solves the convex hull problem directly. A dual space algorithm solves Problems 3.3 and 3.4. It delivers a solution of the convex hull problem implicitly while operating on the dual polytope/cone.

3.2 Complexity

Given a finite set of vectors $X \subset \mathbb{R}^d$, the first question which arises when considering the complexity of the convex hull problem is related to the maximal number of facets which $\texttt{conv.hull}(X)$ may have. Motzkin [115] was the first one to state a conjecture that a specific family of *cyclic polytopes* (see Definition 3.1) sets an upper bound in that respect. Given a d-dimensional cyclic polytope $\mathcal{P}_{\mathcal{C}}$ with n vertices, there is no other d-dimensional polytope with n vertices which has more facets than $\mathcal{P}_{\mathcal{C}}$. A substantial work on the proof of that conjecture was done by Fieldhouse [71], Klee [101] and Gale [81] until the matter was finally settled by McMullen [112, 113] who delivered a complete proof. Since then it is known as the *upper bound theorem*.

Definition 3.1 [CYCLIC POLYTOPE]**.** A d-dimensional cyclic polytope is the convex hull of the vector set $\{x(t_1), \ldots, x(t_n)\} \subset \mathbb{R}^d, n > d$ where

$$x(t_i) = \begin{bmatrix} t_i^1 & t_i^2 & \ldots & t_i^d \end{bmatrix}^T \qquad \text{for } t_i \in \mathbb{R} \ .$$

Assume now that we want to determine the complexity of an algorithm which solves the convex hull problem. According to the upper bound theorem, the number of facets is maximal for cyclic polytope. For n vertices the algorithm has to compute

$$\binom{n - \lceil d/2 \rceil}{\lfloor d/2 \rfloor} + \binom{n - 1 - \lceil (d-1)/2 \rceil}{\lfloor (d-1)/2 \rfloor}$$

facets [88]. In order to emphasize the consequences of that fact more clearly, we may refer to the asymptotic version of the upper bound theorem [137]. It states that a polytope with n vertices (facets) has $\mathcal{O}\left(n^{\lfloor d/2 \rfloor}\right)$ facets (vertices) for some fixed dimension d. Consequently, if we measure the complexity by considering the input only, then any algorithm has an exponential running time; hence, a more appropriate approach is to take both the input and the output into account.

For the moment, the complexity of the convex hull problem is an open issue, meaning that there is no known algorithm which solves it in time polynomial in the number of vertices and facets. On the other hand, there is no complete proof yet that such an algorithm does not exist. Thus far, it has been shown by Khachiyan et al. [98] that the vertex enumeration of an unbounded polyhedron is generally NP-hard. However, their complexity proof does not rule out the possibility of a polynomial solution for bounded polyhedra.

There are several classes of unbounded polyhedra which have been shown to have polynomial solutions [82, 125, 2, 95]. For bounded polyhedra, such solutions are known for simple [14], simplicial [36] and 0/1 polytopes [44]. With respect to the latter, the vertex enumeration was shown to be NP-hard on unbounded 0/1 polyhedra [32].

It is worth mentioning that another closely related problem, the general face enumeration, is solvable in polynomial time for polytopes [134, 78, 76]. Murty [118] has discussed the transferability of one such algorithm, introduced by Murty and Chung [119], on the vertex enumeration and has given an efficient solution for one class of polytopes.

3.3 Pivoting Algorithms

A whole family of convex hull algorithms emerged from the field of linear programming and optimization (see Section 3.3.1). Those are referred to as *pivoting algorithms* and are closely related to the *simplex method* of Dantzig [58] (see Section 3.3.2). For instance, the *reverse search method* of Avis and Fukuda [12, 14] is an application of the simplex method in reverse and arguably the most efficient pivoting algorithm so far (see Section 3.3.3). Their method completed the long lasting development of algorithmic solutions both in the primal (see Section 3.3.5) and the dual space (see Section 3.3.4).

3.3.1 The Linear Programming Problem

Problem 3.6 [Linear Programming]**.** *For $c \in \mathbb{R}^d$, $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times d}$ maximize the value of $c^T x$ subject to $Ax \leq b$.*

Definition 3.2 [Basis]**.** A set $B \subseteq \mathbb{N}_{\leq n}$ with $|B| = d$ is called a *basis* if $rank[A_B] = d$.

Definition 3.3 [Feasible Basis]**.** Each basis B with $x \in \mathbb{R}^d$ satisfying $A_B x = b_B$ is called *feasible* if x also satisfies $Ax \leq b$.

The solution of Problem 3.6 involves finding a feasible basis for which x is optimal in terms of the given condition. Problem 3.6 has also a geometric solution which corresponds to finding an optimal vertex within the underlying polyhedron $\mathcal{P}(A, b)$. This is an obvious consequence of the fact that each feasible basis is bound to a vertex of $\mathcal{P}(A, b)$. This mapping between feasible bases and vertices, however, is not necessarily isomorphic. In case of degeneracy, several feasible bases may yield the same vertex.

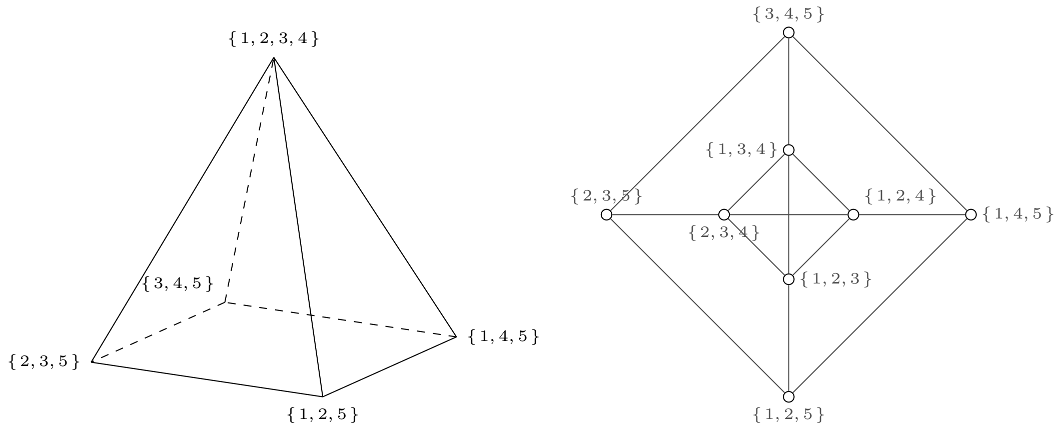
Let $\Gamma_{LP}(A, b) = (V_{LP}, E_{LP})$ be an undirected graph with

V_{LP} denoting the set of all feasible bases corresponding to $Ax \leq b$ and

$E_{LP} = \{(B', B'') || B' \cap B''| = d - 1\}$ defining an edge between each two bases differing in only one element; that is, B' is obtainable out of B'' and vice verse by exchanging only one element.

Γ_{LP} is called a *linear programming graph.* Compare now $\Gamma_{LP}(A, b)$ with the incidence graph $\Gamma(\mathcal{P}')$ where $\mathcal{P}'$ is the polytope emerging from the decomposition of $\mathcal{P}(A, b)$ according to Theorem 2.2. It is easy to show that both graphs are equivalent if $\mathcal{P}'$ is nondegenerate. In the degenerate case, however, multiple elements of V_{LP} correspond to a single polytope vertex, thus $\Gamma_{LP}(A, b)$ is reducible to $\Gamma(\mathcal{P}')$, but its calculation requires potentially more effort.

The general idea behind pivoting algorithms is to enumerate all feasible bases by visiting each node of $\Gamma_{LP}(A, b)$ and hence obtain all vertices in the process. For nondegenerate problems, this approach allows to carry over techniques from the linear programming without fearing any negative consequences in terms of higher computational complexity. In case of degeneracy, however, a typical case of state explosion occurs as the size of $\Gamma_{LP}(A, b)$ may grow exponentially compared to the size of the incidence graph. In that regard, consider the polytope $\mathcal{P}(A, b)$ given in Figure 3.3, which has merely one degenerate vertex. When building the graph $\Gamma_{LP}(A, b)$, this vertex yields four separate feasible bases and thus nearly doubles the overall number of nodes compared to the incidence graph. Clearly, enumerating all vertices does not require to visit each node of $\Gamma_{LP}(A, b)$; hence, spanning a tree over five of $\Gamma_{LP}(A, b)$'s nodes is both necessary and sufficient. Finding such an optimal algorithm for the general case is a point where current theory is struggling. Among all developed algorithms, no one is able to span a tree for which the number of nodes is polynomially bounded by the number vertices. On the other hand, there is no formal proof ruling out the existence of such an algorithm.



(a) Pyramid $\mathcal{P}(A, b)$ (b) $\Gamma_{LP}(A, b)$

Figure 3.3: The impact of degeneracy on the number of feasible bases

3.3.2 The Simplex Method

In geometric terms, Problem 3.6 corresponds to finding a vertex of the polyhedron

$$\mathcal{P} = \{x \mid Ax \leq b\}$$

which yields the maximal dot product $c \cdot x$. Assuming that at least one feasible basis B_0 corresponding to a vertex x_0 is known, the simplex method takes (x_0, B_0) as a starting point and traverses $\mathcal{P}$ along the edges until it finds an optimal vertex. Let

$$\Gamma_{LP}(A, b) = (V_{LP}, E_{LP})$$

be the underlying linear programming graph. Each run of the simplex method corresponds to exploring a single path along $\Gamma_{LP}(A, b)$. Schrijver [131] defined that process by means of two geometric operations which are summarized in this section. First, moving from one graph node to another and second, checking whether a certain vertex is an optimal one. Regarding the latter, it is easy to show that for any $u \in \mathbb{R}^n, u \geq 0$ satisfying $u^T A = c$,

$$u^T Ax \leq u^T b \Rightarrow c^T x \leq u^T b \Rightarrow \mathbf{max}\left\{c^T x \mid Ax \leq b\right\} \leq \mathbf{min}\left\{u^T b \mid u \geq 0, u^T A = c\right\} \quad .$$

Consequently, for some vertex x', $c^T x'$ is guaranteed to be maximal if there exists such u' that $c = u'^T A$ and $c^T x' = u'^T b$. This settles the definition of the second operation. It remains to be shown how to move along the nodes of $\Gamma_{LP}(A, b)$.

Let (x_0, B_0) be a known pair of a vertex and a corresponding feasible basis. It is worth mentioning that B_0 is unique only if x_0 is nondegenerate. Otherwise, several feasible bases can be found. Depending on the value of $u_0 = c^T A_{B_0}^{-1}$, one of the following two cases occurs:

(i) u_0 is a nonnegative vector. It this case, x_0 is already optimal due to the existence of a vector $u \in \mathbb{R}^n$ with

$$u_{B_0} = u_0 \text{ and } u_{\bar{B}_0} = 0 \ .$$

It is evident that $u \geq 0$ and

$$u^T b = u_{B_0}^T b_{B_0} = c^T A_{B_0}^{-1} b_{B_0} = c^T x_0 \ .$$

(ii) u_0 has at least one negative coefficient. Let j be the minimal index for such a coefficient and $e_j \in \mathbb{R}^n$ the identity vector with a positive value at position j and zero at all other positions. Consider now the ray

$$r_0 = \{x_0 + \lambda y \mid A_{B_0} y = -e_j, \lambda \geq 0\}$$

which starts at the vertex x_0 and points either in the direction of an outgoing edge or outward from $\mathcal{P}$ in the sense that $r_0 \cap \mathcal{P} = \{x_0\}$. It is important to note that the latter case occurs only if x_0 is degenerate. Depending on the value of y two cases are possible.

If $Ay \leq 0$, then r_0 is an extreme ray of the polyhedron $\mathcal{P}$ as it is a face of dimension one and is not bounded by any halfspace. In this case, $c^T x$ has no maximum.

If there exists a row a_l of A such that $a_l y > 0$, then r_0 is bounded by at least one facet of $\mathcal{P}$. Consequently, by gliding along the ray r_0, at some point a supporting hyperplane is reached. The intersection of r_0 with the first such hyperplane is again a vertex

$$x_1 = x_0 + \lambda_0 y$$

where λ_0 satisfies

$$\exists l \in \bar{B}_0 \left(a_l (x_0 + \lambda_0 y) = b_l \right) \text{ and } \nexists k \in \bar{B}_0 \left(a_k (x_0 + \lambda_0 y) > b_k \right) \ .$$

In other words, a_l defines the first hyperplane to cross and a_k any other hyperplane beyond that. Next, a feasible basis

$$B_1 = (B_0 - j) + l$$

is constructed out of B_0 by swapping the elements j and l. The pair (j, l) is called a *pivot*. An important special case occurs when $\lambda_0 = 0$ and thus $x_1 = x_0$. The simplex method does not move to another vertex, but simply changes the feasible basis of x_0 instead. Furthermore, a_l is not necessarily unique, as there may be another $l' \in \bar{B}_0, l' \neq l$ such that $a_{l'}(x_0 + \lambda_0 y) = b_{l'}$, meaning that r_0 crosses multiple supporting hyperplanes at the same point. According to Bland's rule [27], the row with the minimal index is preferred in that case. The procedure then checks (x_1, B_1) for optimality and either terminates or proceeds by examining further pairs until an optimal vertex is found or it is proven that no maximum exists.

In conclusion, we shall say that each execution of the simplex method produces a trace

$$\nabla\left((x_0, B_0), \mathcal{P}\right) = \langle (x_0, B_0), \ldots, (x_m, B_m) \rangle \tag{3.1}$$

consisting of feasible bases and their corresponding vertices. All feasible bases are distinct; that is, the simplex method avoids loops. If $\mathcal{P}$ is nondegenerate then all vertices are also distinct.

3.3.3 Reverse Search

The idea behind reverse search is to apply the simplex method in reverse; that is, to start with an optimal pair (x_0, B_0) and then span a tree over the linear programming graph $\mathit{\Gamma}_{LP}(A, b)$ by traversing backwards all possible paths leading to (x_0, B_0). For the moment, assume that x_0 is nondegenerate and B_0 is a unique feasible basis. It is easy to construct a vector $c \in \mathbb{R}^d$ such that $c^T x$ is optimal with respect to Problem 3.6 and thus

$$\nabla\left((x_0, B_0), \mathcal{P}\right) = \langle (x_0, B_0) \rangle$$

is a trivial valid trace of the simplex method. Next, assume that

$$\textbf{simplex} : V_{LP} \nrightarrow V_{LP} \tag{3.2}$$

is a partial mapping which corresponds to a single step of the simplex method. For any feasible basis B_{i+1}, it returns another one B_i such that

$$\nabla\left((x_{i+1}, B_{i+1}), \mathcal{P}\right) = \langle (x_{i+1}, B_{i+1}), (x_i, B_i), \ldots, (x_0, B_0) \rangle$$

is a valid trace or is undefined if B_{i+1} is already optimal. Consequently, the execution of the simplex method corresponds to the repeated application of the mapping given in (3.2) as long as it is defined.

As already stated, in order to solve the vertex enumeration problem, it is sufficient to visit each node of $\mathit{\Gamma}_{LP}(A, b)$ at least once. By using (x_0, B_0) as a starting point and the mapping from (3.2) as an oracle, we can span a tree $T_\nabla = (V_{LP}, E'_{LP})$ over $\mathit{\Gamma}_{LP}(A, b)$ by preserving only those edges which are valid simplex transitions; hence,

$$E'_{LP} = \left\{ \left(B', B''\right) \in E_{LP} : \textbf{simplex}\left(B'\right) = B'' \right\} \ .$$

Obviously, constructing T_∇ is sufficient to enumerate all vertices.

Algorithmic Step. The construction of T_∇ depends on the existence of a procedure **rsimplex** $(\ldots)$ which given the head of some valid trace

$$\nabla\left((x_i, B_i), \mathcal{P}\right) = \langle (x_i, B_i), \ldots, (x_0, B_0) \rangle$$

enumerates all pairs $\left(x_{i+1}, A_{B_{i+1}}\right)$ such that each

$$\nabla\left((x_{i+1}, B_{i+1}), \mathcal{P}\right) = \langle (x_{i+1}, B_{i+1}), (x_i, B_i), \ldots, (x_0, B_0) \rangle \tag{3.3}$$

is again a valid trace. Indeed, such a procedure is easily obtained by reversing the step of the simplex method. We can find all neighbors of (x_i, B_i) in $\Gamma_{LP}(A, b)$ by constructing a ray

$$r_i = \{x_i + \lambda y \mid \lambda \geq 0, A_{B_l} y = 0, a_l y = -1\} \qquad \text{for each } l \in B_i \text{ with } B_l = B_i - l \quad (3.4)$$

and then intersect it with the first supporting hyperplane of $\mathcal{P}$ on the way. As a result, for each r_i, we reach

a vertex $x_{i+1} = x_i + \lambda_i y$ with λ_i satisfying

$$\exists j \in \bar{B}_i \left(a_j(x_i + \lambda_i y) = b_j \right) \text{ and } \nexists k \in \bar{B}_i \left(a_k(x_i + \lambda_i y) > b_k \right), \text{ and}$$

a feasible basis $B_{i+1} = (B_i - l) + j$.

The newly obtained pair (x_{i+1}, B_{i+1}) is a valid output of **rsimplex** (x_i, B_i) if and only if **simplex** $(B_{i+1}) = B_i$. The so defined reverse step has two special cases to consider. First, $\lambda_0 = 0$ implies $x_{i+1} = x_i$, thus B_{i+1} is just another feasible basis yielding x_i. Second, if x_{i+1} is degenerate, then l is not necessarily unique resulting in multiple valid feasible bases and thus multiple pairs

$$\left(x_{i+1}, B'_{i+1}\right), \left(x_{i+1}, B''_{i+1}\right), \ldots$$

which require examination.

Thus far, we have assumed the existence of a nondegenerate starting vertex x_0 with a unique feasible basis B_0. The problem with degeneracy and hence the existence of multiple feasible bases B'_0, B''_0, etc. is that the simplex method may terminate at any of them. Therefore, selecting only one as a starting point for the reverse search, does not guarantee the enumeration of all nodes in V_{LP} as some paths may lead to another feasible basis, which corresponds to an optimal solution as well. In order to bypass that problem, the reverse search has to be applied on each feasible basis corresponding to the initial vertex x_0. In the end, we obtain a forest of reverse search trees.

Example 3.1. Consider the octahedron $\mathcal{P} = \{x \mid Ax \leq b\}$ defined by

$$A \in \mathbb{R}^{8\times 3} \text{ and } b \in \mathbb{R}^8$$

as shown in Figure 3.4. Let x_0 be a known vertex and B_0 a corresponding feasible basis such that

$$A_{B_0} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad .$$

In order to obtain the output of **rsimplex** (x_0, B_0), we have to construct three rays r'_0, r''_0 and r'''_0 as defined in (3.4) and then intersect each of them with the first supporting hyperplane which they cross; that is,

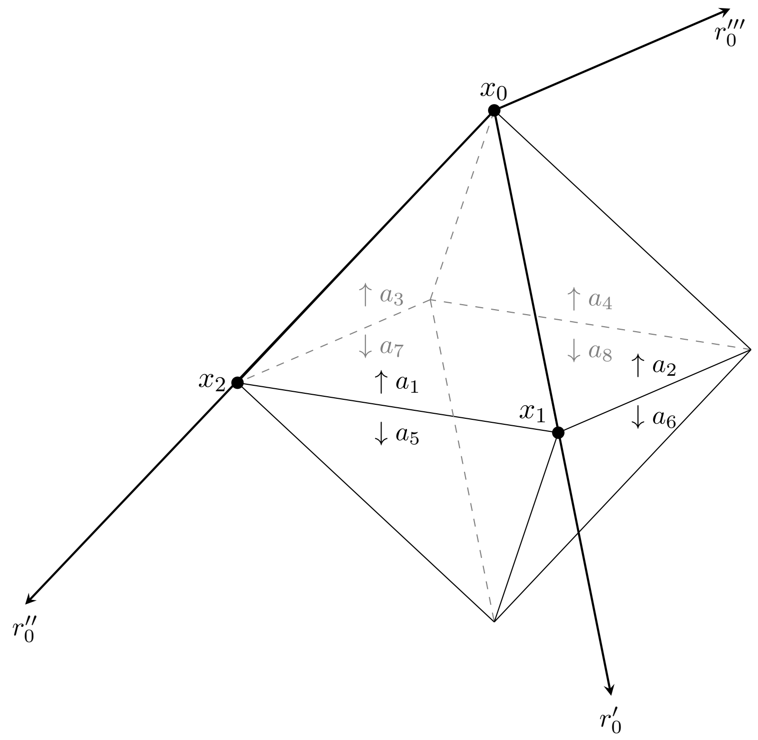


Figure 3.4: Neighbors exploration in the reverse search method

r_0' crosses a_5 and a_6 simultaneously in x_1 yielding two feasible bases B_1' and B_1'' with

$$A_{B_1'} = \begin{bmatrix} a_1 \\ a_2 \\ a_5 \end{bmatrix} \text{ and } A_{B_1''} = \begin{bmatrix} a_1 \\ a_2 \\ a_6 \end{bmatrix} \quad ,$$

r_0'' crosses a_5 and a_7 simultaneously in x_2 yielding the feasible bases B_2' and B_2'' with

$$A_{B_2'} = \begin{bmatrix} a_1 \\ a_3 \\ a_5 \end{bmatrix} \text{ and } A_{B_2''} = \begin{bmatrix} a_1 \\ a_3 \\ a_7 \end{bmatrix} \quad , \text{ and}$$

r_0''' crosses a_4 in x_0 yielding another feasible basis B_0' with

$$A_{B_0'} = \begin{bmatrix} a_2 \\ a_3 \\ a_4 \end{bmatrix} \quad ,$$

which is also optimal.

Consequently, we can build a fragment of the linear programming graph as shown in Figure 3.5. Those edges which correspond to valid simplex transitions are the elements returned by **rsimplex** (x_0, B_0). In the next step, the reverse search method proceeds by examining their neighbors.

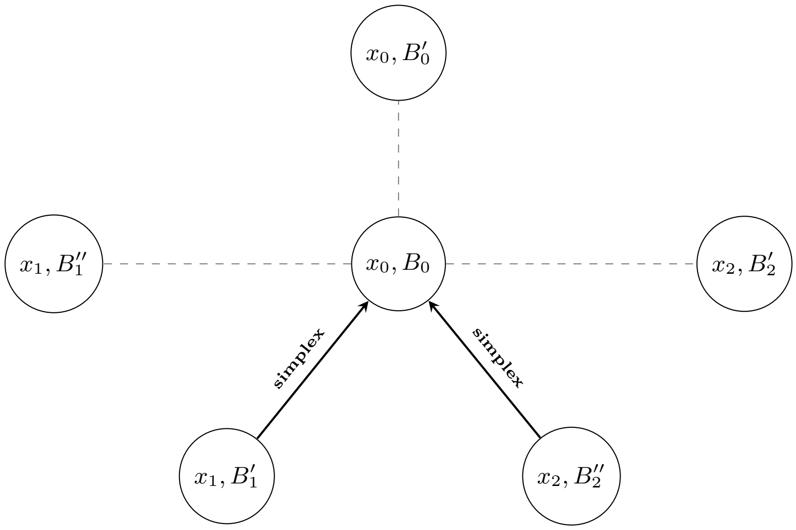


Figure 3.5: Spanning a reverse search tree over the linear programming graph

Concluding Remarks. The reverse search method given here is a short summary of [14] and serves only as an illustration of the basic idea. For a more technical view on the algorithm, see [5]. Avis further revised and improved the method by adopting a lexicographic pivoting rule [6, 8] forming the current version of the algorithm known as *lrs* (*lexicographic reverse search*). The evolution of the reverse search method has been discussed by Avis in [7]. A parallel version of the algorithm has been incorporated into the ZRAM library [105, 40].

3.3.4 Dual Space Algorithms

Balinski (1961). The first pivoting algorithm to solve the vertex enumeration problem for a polyhedron in any dimension is due to Balinski [17]. The author suggested enumerating the vertices facet by facet; that is, pick a facet, find all vertices on it, then drop the corresponding constraint from the linear system and proceed with another facet. In order to enumerate all vertices on a certain facet F, the algorithm either

(i) makes a recursive invocation on F if $\mathbf{dim}(F) \geq 3$ or

(ii) traverses the Hamiltonian path on which all vertices lie if $\mathbf{dim}(F) = 2$.

The exploration of the Hamiltonian path is accomplished by using the simplex method in order to move from one vertex to the next one. It is worth mentioning that by dropping an inequality from the linear system, the polyhedron is being modified and thus not all vertices of the subsequently investigated polyhedra are also vertices of the initial one. This problem is evident in Figure 3.6 where two subsequent steps of the algorithm are illustrated. The point m which emerges in the $i + 1$-th step is clearly not a vertex of the polyhedron from the previous one. This example also points out another drawback of the algorithm, namely that degenerate vertices are visited multiple times.

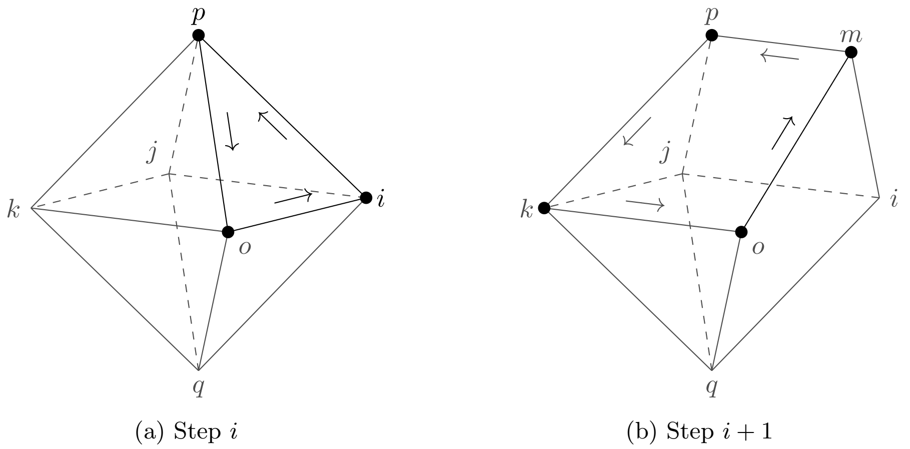


Figure 3.6: Geometric interpretation of Balinski's algorithm

Maňas and Nedoma (1968). A more direct approach to solve the vertex enumeration has been proposed several years later by Maňas and Nedoma [111] and then revisited and improved by Altherr [3]. The idea of their method has been to span a tree directly over the linear programming graph. The algorithm uses a standard technique of differentiation between *explored* nodes for which all neighbors are known and *boundary* nodes which are pending for neighborhood exploration. Assuming that a starting feasible basis B_0 together with a standard simplex *tableaux* is known, the algorithm calculates the corresponding vertex v_0 and marks B_0 as a boundary node. Next, it explores the neighborhood of B_0 and marks all newly found nodes as boundary ones. Finally, B_0 is marked as explored, the tableaux is transformed to match one of the neighboring boundary nodes and the process is repeated once again. That last step requires some additional attention in the general case. Let v_i be the last calculated vertex with a feasible basis B_i. The algorithm then proceeds with the first neighbor of B_i which is marked as a boundary node. If such a node does not exist, backtracking is performed until the nearest boundary node with respect to B_i is found. The algorithm terminates when all boundary nodes have been processed.

Mattheiss (1973). The subsequent efforts to solve convex hull problem more efficiently resulted in rather complex algorithms. In 1973, Mattheiss [109, 108] proposed a fundamentally different approach by embedding $\mathcal{P}$ into another polyhedron

$$\mathcal{P}' = \left\{ \begin{bmatrix} x \\ y \end{bmatrix} \mid Ax + ty \leq b \right\}, y \in \mathbb{R}, t \in \mathbb{R}^n$$

which resides one dimension higher. $\mathcal{P}'$ is constructed in such a way that each face of $\mathcal{P}'$ has a nonempty intersection with $\mathcal{P}$. In other words, each vertex of $\mathcal{P}$ has an adjacent one in $\mathcal{P}' - \mathcal{P}$. The algorithm then spans a tree over the linear programming graph of $\mathcal{P}' - \mathcal{P}$ and uses it to enumerate the vertices of $\mathcal{P}$. The algorithm of Mattheiss rests on an unproven conjecture that the linear programming graph of $\mathcal{P}' - \mathcal{P}$ has less nodes than the one obtained directly from $\mathcal{P}$ (see also [102]).

Burdet (1974). A rather complex algorithm generating all faces of a polyhedron has been proposed by Burdet [41]. The faces are enumerated by constructing a tree (referred to as an *arborescence*) where the root is the polyhedron itself and each subsequent level contains faces which reside one dimension lower than the previous one. Consequently, the vertices are the leaves of the tree. Mattheiss and Rubin [110] assessed the computational demands of the algorithm as *enormous* because the calculation of each face involves solving the linear programming problem multiple times.

Dyer and Proll (1977). The pivoting algorithm of Dyer and Proll [62, 63, 64] rests on the idea of spanning a tree over the linear programming graph by applying breadth-first search. Additionally, a list identifying known neighbors is attached to each node and continuously updated. Redundant operations are thus avoided when exploring the neighborhood of each node. Let B_0 be the feasible basis constituting the root of the tree. Then, at some point of the calculation, all nodes $B'_g, B''_g, \ldots$ at distance g are known. In order to find all nodes at distance $g+1$, all neighbors of $B'_g, B''_g, \ldots$ have to be found. By keeping track which of them are already known that effort can be considerably reduced. This bookkeeping strategy, however, requires to compile an adjacency list for each newly encountered node.

Concluding Remarks. For the preparation of this section, the survey of Mattheiss and Rubin [110] has also been consulted. It delivers an excellent overview and an empirical comparison of the algorithmic solutions presented here.

Another approach to attack the vertex enumeration problem with linear optimization techniques was presented by Khang and Fujiwara [99]. The authors proposed an algorithm based on *polyhedral annexation* [89] and compared their experimental results with the algorithms from the survey of Mattheiss and Rubin. Khang and Fujiwara concluded that in terms of efficiency, their algorithm is unlikely to beat the existing solutions.

3.3.5 Primal Space Algorithms

Chand and Kapur (1970). Parallel to the vertex enumeration methods, an alternative class of algorithms has emerged from the idea of exploring neighboring facets. The algorithm of Chand and Kapur [45] was the first one to implement this idea for $d > 3$. Their method uses the fact that each two neighboring facets share a face of dimension $d - 2$. Assuming that such a face $\mathcal{F}$ together with the supporting hyperplane $\mathcal{H}$ of one of the facets is known, $\mathcal{H}$ can be rotated along $\mathcal{F}$ until it coincides with the supporting hyperplane of the other facet. As each facet is again a polytope, the same procedure can be applied to enumerate all $d - 2$ faces within a facet. Then each such face is used to visit another facet and so on until all facets have been enumerated. This approach is mostly referred to as *gift wrapping*. Basically the same idea has been suggested three years later by McRae and Davidson [114]. Their algorithm, however, works on polyhedral cones instead of polytopes. Analysis and improvements of the algorithm of Chand and Kapur can be found in the work of Swart [140] who reached a complexity of $\mathcal{O}(nf + f\log(f))$ where f denotes the overall number of faces.

Seidel (1986). Seidel [134, 135] proposed another idea to solve the convex hull problem by enumerating the facets in a *shelling* order. According to Ziegler [156], a linear ordering of the facets $F_1, \ldots, F_m$ of $\mathcal{P}$ is called shelling if either

$$\mathbf{dim}(\mathcal{P}) = 1 \quad \text{or}$$

(i) the facets of F_1, which are $d-2$ dimensional faces of $\mathcal{P}$, have a shelling order and

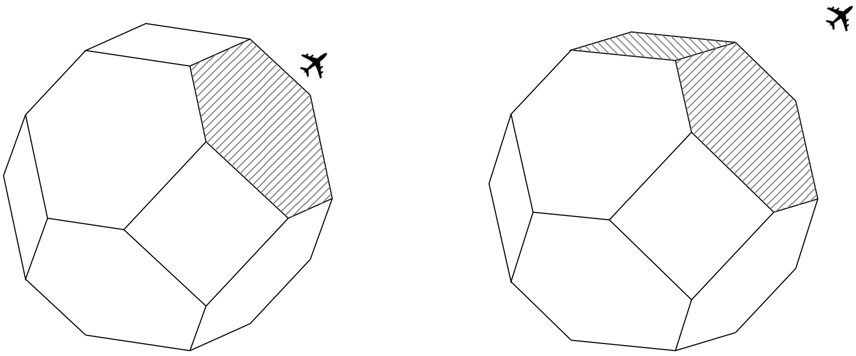
(ii) for $1 < j \leq m$,

$$F_j \cap \left(\bigcup_{i=1}^{j-1} F_i \right) = G_1 \cup G_2 \cup \ldots \cup G_r$$

where $G_1, G_2, \ldots, G_r, \ldots, G_t$ is a valid shelling order of the facets of F_j.

The algorithm of Seidel rests on a theorem proven by Mani and Bruggesser [104] that the facets of each polytope have a shelling order. The proof coincides with the definition of a strategy to provide that order for any arbitrary polytope. Naturally, this solves the facet enumeration problem as well.

Imagine an airplane p taking off from some facet of the polytope $\mathcal{P}$ and then following a course defined by a line l crossing $\mathcal{P}$'s center and the takeoff position. The more the distance between p and $\mathcal{P}$ grows the more of $\mathcal{P}$'s facets become visible from the plane (see Figure 3.7). An observer sitting on the plane writes down the facets in that order.



(a) Step i

(b) Step $i+1$

Figure 3.7: Geometric interpretation of the shelling algorithm

At some point this process ceases with no further facets becoming visible no matter how far the plane flies. The plane is then repositioned on the other side of $\mathcal{P}$, again on the line l, but at an infinite distance to $\mathcal{P}$. The observer sees now all facets which have been out of sight up to that point. The plane continues following the course defined by l, but this time towards $\mathcal{P}$. As it approaches the polytope, more and more facets become invisible. The observer now appends each facet which becomes invisible to his list. Finally, the plane reaches $\mathcal{P}$ landing on some facet. This is the last entry in the observer's list. Consequently, it can be proven that for each polytope there exists such a takeoff position that the observer has seen all facets and that the order in which he has written them down is a shelling one.

The shelling algorithm of Seidel can be viewed as a formalization of the described strategy. The algorithm itself runs in time $\mathcal{O}\left(n^2 + f\log(n)\right)$ where f is the overall number of faces in the polytope and is therefore output sensitive. It should be pointed out that the algorithm computes the complete facial lattice and not merely the facets of the polytope. A subsequent improvement by Matoušek [106] cut the complexity down to $\mathcal{O}\left(n^{2-2/(1+\lfloor d/2 \rfloor)+\delta} + f\log(n)\right)$ for some small $\delta > 0$.

Rote (1992). Rote [128] proposed an algorithm for enumerating all faces of a polytope which rests on the same idea as the reverse search method of Avis and Fukuda, but instead of visiting feasible bases, it visits the facets of the polytope in a way analogous to that of Chand and Kapur's method. The algorithm handles degeneracies explicitly and has small storage demands.

3.4 Incremental Algorithms

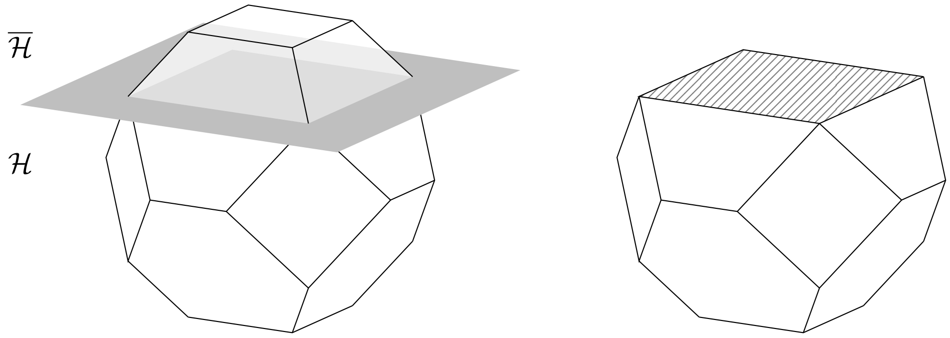
An alternative approach to solve the convex hull problem is to process the input set step by step; that is, the convex hull is constructed incrementally by inserting one element at a time. Accordingly, the resulting algorithms are called *incremental*. They are mostly easy to implement and have a very illustrative geometric interpretation. Similar to pivoting algorithms, they operate either in the primal (see Section 3.4.2) or in the dual space (see Section 3.4.1).

3.4.1 Dual Space Algorithms

Motzkin (1953). The double description method (aka dd method) of Motzkin et al. [116] was the first incremental algorithm to operate in the dual space. It runs on a homogeneous system of linear inequalities defined by the input matrix $A \in \mathbb{R}^{n \times d}$ and thus solves the extreme ray enumeration problem. The dd method starts by computing d extreme rays corresponding to the first d linear inequalities. Following this, the remaining constraints $a_{d+1}, \ldots, a_n$ are inserted and processed in a strict sequential order. At each step, the set of known extreme rays is being refined by removing those which are not valid for the new constraint. Furthermore, new extreme rays are being computed. Each of them emerges from the linear combination of one valid and one invalid adjacent extreme rays. The differentiation between adjacent and nonadjacent rays is thus a key aspect for the computation. The dd method has a very illustrative geometric interpretation of a convex hull $\mathcal{P}$ being cut with a halfspace $\mathcal{H}$ at each step resulting in

$$\mathcal{P}' = \mathcal{P} \cap \mathcal{H}$$

as shown in Figure 3.8. An example for a computational run can be found in Chapter 4 where the method is discussed in greater detail.



(a) Polytope $\mathcal{P}$ and a halfspace $\mathcal{H}$ (b) Polytope $\mathcal{P}' = \mathcal{P} \cap \mathcal{H}$

Figure 3.8: Geometric interpretation of cutting plane algorithms

Fukuda (1996). In terms of complexity, the dd method has been considered inferior to many of the other algorithms for a long time as it effectively does not guarantee any complexity bounds in the worst case. In the nineties, however, it gained popularity after being revisited by Fukuda and Prodon [77]. The authors provided a practical implementation together with a survey on degenerate problems and showed that the dd method can deal with certain classes of polyhedra quite effectively. Thereby, the insertion order of the constraints $a_{d+1}, \ldots, a_n$ played a key role for obtaining an efficient computational run. Shortly after, Avis and Bremner [10] introduced polytope families which are hard to solve for various pivoting and incremental algorithms (see also [11, 34]). Those results were extended by Bremner to reach a final verdict concerning the complexity of the dd method. Bremner [33] showed that incremental algorithms are not output sensitive in general (see also [35]). He provided counter-examples for which any insertion order yields a superpolynomial running time. For those problems, the intermediate results grow exponentially in the size of the final output no matter what insertion order is being chosen.

Despite the theoretical results, the dd method has remained a viable alternative in practical applications. The algorithm is easy to implement and allows a direct mapping of algorithmic operations to processor instructions. Moreover, the computational experience provided by Avis et al. [11] confirmed its advantages over other algorithms in certain domains. Naturally, pivoting algorithms are to be preferred when dealing with purely nondegenerate problems. Borgwardt [29, 30] analyzed the average-case complexity of both the gift-wrapping algorithm and the dd method for randomly generated nondegenerate input. The results of his study confirmed the superiority of pivoting algorithms for that kind of problems.

Chernikova (1964). The *algorithm of Chernikova* [48, 49] is virtually an independent reinvention of the dd method based on the same idea of incremental constraint processing. Initially, it was defined to work on a nonnegative domain, meaning that the $\mathcal{H}$-representation of the cone is given as

$$\mathcal{C} = \{x \mid Ax \geq 0, x \geq 0\} \ .$$

The algorithm was extended by Fernández and Quinton [70] to solve mixed linear systems containing both equalities and inequalities and then once more revisited by Verge [149] who improved the criterion used to verify the adjacency of extreme rays.

Clarkson and Shor (1991)/Chazelle (1993). During the years, a reasonable effort was devoted to find an algorithm which is optimal when the complexity is measured only with respect to the input. Given a polytope with n vertices, the number of possible facets is bounded by $\mathcal{O}\left(n^{\lfloor d/2 \rfloor}\right)$. An algorithm is thus considered optimal if it has the same computational complexity in the worst case. A partially optimal algorithm was proposed by Seidel [133] (see Section 3.4.2) achieving $\mathcal{O}\left(n \log\left(n\right) + n^{\lfloor d/2 \rfloor}\right)$ for any even d. About a decade later, as *randomized algorithms* (e.g. see [117]) gained momentum, Clarkson and Shor [51] proposed a *Las Vegas* algorithm with optimal expected running time (see also [54, 52, 53]). The randomized aspect is related to the choice of a random order to process the halfspaces. The algorithm itself is a rather simple one and can be viewed as a refined form of the dd method which additionally maintains an incidence graph to keep track of adjacent vertices. The complexity proof, however, is a completely different story and requires some effort to understand. With regard to that, Seidel [136] introduced a randomized primal space algorithm which achieves the same expected complexity, but has a proof which is much easier to understand. The complexity matter was finally settled by Chazelle [47] who gave a derandomized version of Clarkson and Shor's algorithm which is optimal in any dimension.

Another closely related algorithm was introduced by Boissonnat et al. [28]. It is an *online* version of Clarkson and Shor's algorithm featuring the same running time complexity. A couple of years later, the algorithm was revisited once more by Brönnimann [38, 39] who extended it to accept degenerate problems.

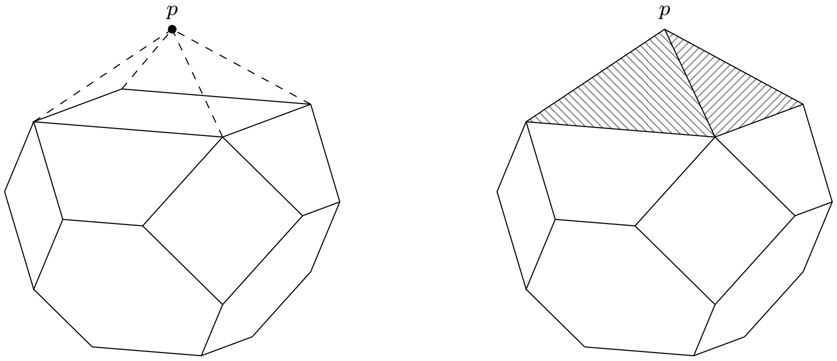
3.4.2 Primal Space Algorithms

Seidel/Kallay (1981). The problem of intersecting a polytope with a halfspace was once again revisited by Seidel who in his master thesis [133] proposed an algorithm widely known as the *beneath-beyond method*. It was conceived as a primal space algorithm, but operated internally in the dual space. Later, Seidel provided a pure primal space version which was published by Edelsbrunner [65]. In some sources, the authorship of the beneath-beyond method also goes to Kallay referring to an unpublished manuscript [96]. His algorithm appeared later in the book of Preparata and Shamos [124].

The beneath-beyond method incrementally builds the convex hull by processing one point at a time; that is, if $\mathcal{P}$ is a polytope and p some point outside of $\mathcal{P}$, the algorithm computes

$$\mathcal{P}' = \texttt{conv.hull}(\mathcal{P} \cup \{p\})$$

as illustrated in Figure 3.9. The algorithmic step rests on a theorem given by McMullen and Shephard [113] stating that each face of $\mathcal{P}'$ is obtained in one of the following two ways.



(a) Polytope $\mathcal{P}$ and point p (b) $\mathcal{P}' = \texttt{conv.hull}(\mathcal{P} \cup \{p\})$

Figure 3.9: Geometric interpretation of the beneath-beyond method

(i) A face $\mathcal{F}$ of $\mathcal{P}$ is also a face of $\mathcal{P}'$ if and only if there is a facet F of $\mathcal{P}$ such that $\mathcal{F} \subset F$ and p is beneath F.

(ii) If $\mathcal{F}$ is a face of $\mathcal{P}$, then $\mathcal{F}' = \texttt{conv.hull}(\mathcal{F} \cup \{p\})$ is a face of $\mathcal{P}'$ if and only if either

   (a) $p \in \texttt{aff.hull}(\mathcal{F})$ or

   (b) p is beyond at least one facet of $\mathcal{P}$ containing $\mathcal{F}$ and beneath another one.

Compared to the incremental algorithms in dual space, the beneath-beyond method maintains a lot more intermediate data during the computation. In order to update the convex hull, it requires a nearly complete description of $\mathcal{P}$ in terms of a facial graph. This information is updated at each step and then passed to the next one. In contrast, the dd method maintains merely a list of $\mathcal{P}$'s vertices. However, the latter has to perform explicit tests for adjacency of vertices which is not necessary in the beneath-beyond method as the adjacency information is obtainable from the facial graph.

Joswig [94] compared the beneath-beyond method with the double description method of Fukuda [75] and the reverse search method of Avis [8] in practical conditions. The author demonstrated that the beneath-beyond method is indeed superior over the other two for some types of polyhedra.

Barber et al. (1996). The *quickhull algorithm* of Barber et al. [19] is a variation of the beneath-beyond method which operates on strictly nondegenerate input. The authors introduced the concept of *partitioning* which defines the order in which the points are being processed. After constructing an initial convex hull, each remaining point is assigned to exactly one facet visible from the point. Next, an arbitrary facet is selected and all assigned points are added to the convex hull by starting with the most distant one. The reasoning behind this strategy is to achieve maximal widening of the convex hull on each step and thus enclose as many of the remaining points within the partition as possible. As a consequence, all enclosed points do not require further treatment and can be discarded.

Barber et al. provided computational evidence that their partitioning strategy achieves better results than processing the points in a random order. However, the benchmark covered only three dimensional problems.

4 The Double Description Method: Definition and Implementation

The double description method is arguably the convex hull algorithm which has received most attention in practice. Due to its general simplicity (see Section 4.1), it has been integrated into various tools resulting in numerous implementations. The development of the algorithm is still an ongoing process as time consuming parts like the *adjacency test* (see Section 4.2) are a subject of improvement. Over the years, several implementation techniques have been particularly effective and are considered *best practice* when developing new double description applications (see Section 4.3).

4.1 Algorithm

The dd method solves the extreme ray enumeration problem (see Problem 3.4). The general idea behind the algorithm is to identify an over-approximation cone $\mathcal{C}_0 \supseteq \mathcal{C}\,(A)$ for which the set of all extreme rays $\mathcal{R}_0$ is known (see Initial Step) and then iteratively intersect it with the halfspaces defined in A (see Iteration Step). At each step, the cone is being reshaped by cutting away those parts which do not belong to the current halfspace. The cut operation effectively corresponds to updating the set of known extreme rays. The iterative repetition of this process yields a trace of double description pairs

$$(A_0, \mathcal{R}_0) \longrightarrow \ldots \longrightarrow (A_i, \mathcal{R}_i) \longrightarrow \ldots \longrightarrow (A, \mathcal{R})$$

which mirror $\mathcal{H}$- and $\mathcal{V}$-representations of intermediate cones during the refinement process. The final set of extreme rays $\mathcal{R}$ is the $\mathcal{V}$-representation of $\mathcal{C}\,(A)$.

In what follows, a specification of the dd method is given. For the most part, the notation of Fukuda and Prodon [77] is adopted. Without loss of generality, assume that $\mathcal{C}\,(A)$ is pointed and that the first d rows of A form a nonsingular matrix A_0 such that $rank[A_0] = d$. Note that the latter is easily satisfied by reordering the row vectors in A. It is also worth mentioning that each set of extreme rays $\mathcal{R}_i$ is defined if there is a known minimal set of generators X_i such that $\mathcal{C}\,(A_i) = \texttt{cone}\,(X_i)$.

Initial Step: The intersection of the first d halfspaces produces an initial cone $\mathcal{C}_0$ with exactly d extreme rays. $\mathcal{C}_0$ is obviously nondegenerate and thus each ray lies on exactly $d-1$ facets. A set of generators $X_0 = \{x_1, \ldots, x_d\}$ can be obtained by solving d systems of linear equations. Each generator x_i satisfies the i-th row of A_0 with inequality and all other with equality; hence,

$$\mathcal{C}_0 = \texttt{cone}\left(\left\{ x \mid a_i x < 0 \wedge A_{B_i} x = 0 \text{ for } i \in \mathbb{N}_{\leq d}, B_i = \mathbb{N}_{\leq d} - i \right\}\right) \ .$$

Consequently, X_0 defines $\mathcal{R}_0$.

Iteration Step: Let X_i be a minimal set of generators of the cone $\mathcal{C}\,(A_i)$ where

$$A_i = A_{\mathbb{N}_{\leq\{d+i\}}}$$

is the matrix containing the first $d+i$ rows of the initial matrix A. Furthermore, let

$$\mathcal{H}_j = \{\, x \mid a_j x \leq 0 \,\} \qquad \text{with } j = d+i+1$$

be the next halfspace to process. The intersection of $\mathcal{C}\,(A_i)$ with $\mathcal{H}_j$ leads to the creation of a new cone $\mathcal{C}\,(A_{i+1})$ whose generators are obtained out of X_i and $\mathcal{H}_j$. This involves the separation of X_i into three subsets

(i) $X_i^+ = \{x \in X_i : a_j x > 0\}$,

(ii) $X_i^0 = \{x \in X_i : a_j x = 0\}$ and

(iii) $X_i^- = \{x \in X_i : a_j x < 0\}$.

All elements in X_i^- and X_i^0 are automatically generators of $\mathcal{C}\,(A_{i+1})$. Additionally, the intersection of the hyperplane $\{\, x \mid a_j x = 0 \,\}$ with each two-dimensional face $\mathcal{F}$ of $\mathcal{C}\,(A_i)$ yields a new generator if there exists a pair $(x', x'') \in (X_i^+ \times X_i^-)$ such that $x', x'' \in \mathcal{F}$. After collecting all pairs (x', x''), the set of new generators is given by

$$X_i^{\triangledown} = \left\{\, x \mid x = (a_j x')x'' - (a_j x'')x' \,\right\} \quad . \tag{4.1}$$

Consequently, the set

$$X_{i+1} = X_i^- \cup X_i^0 \cup X_i^{\triangledown}$$

generates the cone $\mathcal{C}\,(A_{i+1})$. Since X_{i+1} is minimal, it defines $\mathcal{R}_{i+1}$. The algorithm proceeds with the next iteration step $i+1$ or terminates if $j = n$.

Example 4.1. Consider the cone $\mathcal{C}\,(A)$ with

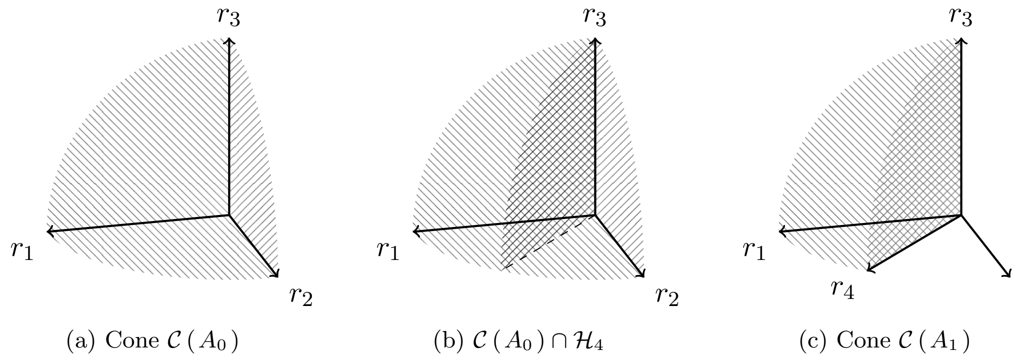
$$A = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

for which the extreme rays should be computed. In the initial step, a cone $\mathcal{C}\,(A_0)$ with generators

$$x_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, x_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ and } x_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

defining $\mathcal{R}_0 = \{r_0, r_1, r_2\}$ is constructed out of the first three rows (see Figure 4.1a). The algorithm then proceeds with the intersection of $\mathcal{C}\,(A_0)$ where the halfspace $\mathcal{H}_4$ is defined by the forth row in A (see Figure 4.1b). This splits the generators of $\mathcal{C}\,(A_0)$ into

$$X_0^+ = \{\, x_2 \,\}, X_0^0 = \{\, x_3 \,\} \text{ and } X_0^- = \{\, x_1 \,\} \quad .$$



(a) Cone $\mathcal{C}(A_0)$ (b) $\mathcal{C}(A_0) \cap \mathcal{H}_4$ (c) Cone $\mathcal{C}(A_1)$

Figure 4.1: Geometric interpretation of the double description method

According to the rule from (4.1), the pair (x_2, x_1) yields a new vector

$$x_4 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad \text{with } X_0^{\triangledown} = \{x_4\} \quad .$$

Consequently, the extreme rays of $\mathcal{C}(A_1)$ are r_1, r_3 and r_4 emerging from the minimal set of generators $X_1 = \{x_1, x_3, x_4\}$ (see Figure 4.1c).

4.2 Adjacency Test

One aspect of the dd method deserving particular attention is the enumeration of all two-dimensional faces needed to compute $X_i^{\triangledown}$. This process involves the identification of all pairs $(x', x'') \in (X_i^+ \times X_i^-)$ for which $\mathcal{F} = \texttt{cone}(\{x', x''\})$ is a valid face of $\mathcal{C}(A_i)$. The problem of deciding whether a pair of generators defines a two-dimensional face is equivalent to deciding whether the supremum of two extreme rays is a two-dimensional face. In geometric terms, this is the problem of whether two extreme rays are adjacent. An adjacency test can be performed in two different ways which are specified in Lemma 4.1. Those are known as a *combinatorial* (i) and an *algebraic test* (ii) (see also [77, Proposition 7]).

Lemma 4.1 (Adjacency Test)**.** *For each pair of extreme rays r', r'' of the polyhedral cone $\mathcal{C}$, showing that $\mathcal{F} = r' \vee r''$ is a two-dimensional face of $\mathcal{C}$ is equivalent to showing that*

(i) there exists no other extreme ray r''' of $\mathcal{C}$ such that $r''' \subset \mathcal{F}$ or

(ii) $\operatorname{rank}[A_{\mathcal{F}}] = d - 2$ with $A_{\mathcal{F}}$ the maximal submatrix of A such that $A_{\mathcal{F}} x' = 0$ and $A_{\mathcal{F}} x'' = 0$ for all $x' \in r'$ and $x'' \in r''$.

Proof. The face $\mathcal{F}$ is guaranteed to be a polyhedral cone itself. Therefore,

(i) if there is no other ray $r''' \subset \mathcal{F}$, then $\mathcal{F} = \texttt{cone}\,(r' \cup r'')$ and hence $\mathbf{dim}\,(\mathcal{F}) = 2$, and

(ii) $\mathbf{dim}\,(\mathcal{F}) = d - \mathit{rank}[A_{\mathcal{F}}] = 2$ which is a direct implication of (2.9).

□

Complexity. With regard to the combinatorial test, the existence (resp. nonexistence) of r''' can be trivially proven in linear time. By using multidimensional search trees [23], the combinatorial test can be reduced to a *partial match query* [127] which is a problem solvable in $\mathcal{O}(|\mathcal{R}_i|^{\alpha})$ with $\alpha < 1$ [73]. A general analysis on the lower bounds of this problem, referred to as a no partial match query, was done by Borodin et al. [31]. Terzer and Stelling [143, 144] incorporated this approach into the dd method achieving a substantial speed-up for some problems.

With regard to the algebraic test, the rank of $A_{\mathcal{F}}$ can be easily calculated by transforming $A_{\mathcal{F}}$ into an Hermite normal form. A natural candidate for performing that transformation is the Gauss elimination algorithm. An alternative solution is given by the *LSP matrix decomposition algorithm* of Ibarra et al. [90] which has a running time in $\mathcal{O}(n^{\omega-1}d)$ with $\mathcal{O}(n^{\omega})$ the complexity of the matrix multiplication used. For the moment, the lowest multiplication exponent $\omega = 2.3727$ has been given by Williams [153].

An alternative to the standard adjacency test is given by a recent contribution of Burton [43]. The author demonstrated an adjacency test in $\mathcal{O}\,(d)$ with preprocessing which requires $\mathcal{O}\left(d\,|\mathcal{R}_i|^2 + d^2\,|\mathcal{R}_i|\right)$ time. However, the test is incomplete when applied on degenerate problems.

Combinatorial vs Algebraic Test. The algebraic test has two major advantages when compared to the combinatorial one. First, it does not depend on the overall number of extreme rays in $\mathcal{R}_i$ and second, the tests can be easily distributed on different machines. These two advantages, however, do not necessarily make the algebraic test first choice when it comes to practical implementations. Current processors do not offer enough precision to natively cover the demands of large rank computations. The Gauss elimination, for example, is an algorithm bounded by a polynomial number of arithmetic operations. Due to the *intermediate coefficient explosion*, however, it was shown by Fang and Havas [69] to have an exponential complexity when counting the bit operations. This coefficient explosion problem has been extensively studied over the years with several polynomial time solutions being developed [74, 97, 139, 122]. Nevertheless, the algebraic test still suffers from increased computational time and memory demands.

Another problem related to the algebraic test is the high computational redundancy which emerges from the independent conduction of each adjacency test. Consider, for instance, two matrices A' and A'' for which the majority of rows are identical. $\mathit{rank}[A']$ and $\mathit{rank}[A'']$ can be calculated far more efficiently together rather than independently. In this respect, Terzer and Stelling [144] introduced a technique called *lazy rank updating* in which matrix triangulations use previous results and do not have to start every time from scratch. Still, for the majority of problems, the combinatorial test remained the more efficient one.

4.3 Implementation Details

In practical applications, the dd method often spends most of the computational time in performing adjacency tests. The efficiency of the test implementation is thus critical for the performance of the whole application. For one thing, this involves a computer-friendly representation of extreme rays (see Section 4.3.1). Furthermore, several heuristics related to both the general structure of the test process (see Section 4.3.2) and the applied data structures (see Section 4.3.3) are capable of delivering a significant performance gain.

4.3.1 Extreme Ray Representation

Active Sets. Consider the iteration step of the dd method with $\mathcal{R}_i$ the extreme rays of the intermediate cone $\mathcal{C}(A_i)$. According to the face representation given in (2.11) on page 25, each proper face of $\mathcal{C}(A_i)$ emerges as an intersection of a certain number of facets. Thus, an extreme ray r, which is a minimal face, can be represented by the facets which it lies on. The set of those defining facets is mostly referred to as an *active set* (see [77]). It can be obtained by means of an injective mapping

$$z : \mathcal{R}_i \to \wp(\mathbb{N})$$

from extreme rays to a set of natural numbers indexing the rows of A_i which each $r \in \mathcal{R}_i$ satisfies with equality; that is,

$$l \in z(r) \Leftrightarrow \forall x \in r\,(a_l x = 0) \ .$$

With regard to the combinatorial test, active sets can be used to prove adjacency (resp. non-adjacency) for two extreme rays by applying the following rule:

$$r''' \subset r' \vee r'' \Leftrightarrow z(r') \cap z(r'') \subset z(r''') \qquad \text{for } r', r'', r''' \in \mathcal{R}_i \ . \tag{4.2}$$

In other words, the supremum of two extreme rays can be obtained by intersecting their active sets and most important, the combinatorial test can be performed by merely using set operations.

Binary Vectors. Each active set $z(r)$ has an equivalent representation as a binary vector $\omega \in \{0,1\}^m$ with $m = d + i$ the number of rows in A_i. The l-th bit of ω is set if and only if $l \in z(r)$. Consequently, we can define a direct mapping

$$b : \mathcal{R}_i \to \{0,1\}^m \tag{4.3}$$

from extreme rays to binary vectors and hence obtain a native computer representation for each $r \in \mathcal{R}_i$. This simple mapping enables the application of processor instructions directly on extreme rays. Consider the logical operations $\wedge$ (conjunction), $\vee$

(disjunction), $\oplus$ (exclusive disjunction) and complement. Those can be easily defined on binary vectors as done by Posthoff and Steinbach [123]. For $\omega, \omega' \in \{0,1\}^m$ with $\omega = (\omega_1, \dots, \omega_m)$ and $\omega' = (\omega'_1, \dots, \omega'_m)$,

$$\omega \wedge \omega' = (\omega_1 \wedge \omega'_1, \dots, \omega_m \wedge \omega'_m), \tag{4.4}$$

$$\omega \vee \omega' = (\omega_1 \vee \omega'_1, \dots, \omega_m \vee \omega'_m), \tag{4.5}$$

$$\omega \oplus \omega' = (\omega_1 \oplus \omega'_1, \dots, \omega_m \oplus \omega'_m) \text{ and} \tag{4.6}$$

$$\bar{\omega} = (\bar{\omega}_1, \dots, \bar{\omega}_m) \ . \tag{4.7}$$

An analogous definition for the relations $\leq$ and $<$ states that

$$\omega \leq \omega' \Leftrightarrow \omega_1 \leq \omega'_1, \dots, \omega_m \leq \omega'_m \text{ and} \tag{4.8}$$

$$\omega < \omega' \Leftrightarrow \omega \leq \omega' \text{ and } \omega \neq \omega' \ . \tag{4.9}$$

The number of facets on which an extreme ray lies is defined by the *population* of the binary vector

$$\|\omega\| : \{0,1\}^m \to \mathbb{N}^0 \ , \tag{4.10}$$

that is the number of its 1 values. The test whether an extreme ray belongs to a certain facet is performed by the function

$$\textbf{value} : \{0,1\}^m \times \mathbb{N}_{\leq m} \to \{0,1\} \tag{4.11}$$

which returns the vector's value at a given position. It is easy to see that for $r', r'' \in \mathcal{R}_i$,

$$\begin{aligned} z\left(r'\right) \cap z\left(r''\right) &\Leftrightarrow b\left(r'\right) \wedge b\left(r''\right), \\ z\left(r'\right) \cup z\left(r''\right) &\Leftrightarrow b\left(r'\right) \vee b\left(r''\right), \\ z\left(r'\right) - z\left(r''\right) &\Leftrightarrow b\left(r'\right) \oplus \bar{b}\left(r''\right), \\ z\left(r'\right) \subset z\left(r''\right) &\Leftrightarrow b\left(r'\right) < b\left(r''\right), \\ z\left(r'\right) \subseteq z\left(r''\right) &\Leftrightarrow b\left(r'\right) \leq b\left(r''\right) \text{ and} \\ |z\left(r\right)| &= \|b\left(r\right)\| \ . \end{aligned}$$

Consequently, on binary level, the combinatorial test from (4.2) transforms to

$$b\left(r'\right) \wedge b\left(r''\right) < b\left(r'''\right) \Leftrightarrow r''' \subset r' \vee r'' \ . \tag{4.12}$$

Assuming that m is a constant, each operation on binary vectors can be mapped to a constant number of processor instructions and hence executed in constant time. It is worth mentioning that the population of binary vectors can be calculated by using the instruction `POP_CNT` [126]. This optimal realization shows the general superiority of the combinatorial test over the algebraic one.

4.3.2 Narrowing and Verification

Let $\mathcal{R}_i^+$, $\mathcal{R}_i^-$ and $\mathcal{R}_i^0$ be the extreme ray sets corresponding to X_i^+, X_i^- and X_i^0 with respect to the iteration step of the dd method (see Section 4.1). Then, a simple implementation of the combinatorial test can be achieved as follows.

(i) Iterate over the set $\mathcal{R}_i^+ \times \mathcal{R}_i^-$.

(ii) For each $(r', r'') \in \left(\mathcal{R}_i^+ \times \mathcal{R}_i^-\right)$, search for $r''' \in \mathcal{R}_i$ satisfying (4.12). If there is no match, then r' and r'' are adjacent.

This procedure can be further optimized by taking into account the following implications from Lemma 4.1.

Corollary 4.2. *The extreme rays r' and r'' are nonadjacent if*

$$\left\|b\left(r'\right) \wedge b\left(r''\right)\right\| < d - 2 \ .$$

Proof. From $\|b(r') \wedge b(r'')\| < d - 2$, it follows that the face $\mathcal{F} = r' \vee r''$ lies on utmost $d - 3$ facets and thus has a dimension no less than three. □

Corollary 4.3. *If $\|b(r')\| = \|b(r') \wedge b(r'')\| + 1$, then r' and r'' are adjacent.*

Proof. This implication is easily shown by applying the algebraic test on r' and r''. Let $B' = z(r')$. Then, $rank[A_{B'}] = d - 1$. The matrix $A_{\mathcal{F}}$ needed for the algebraic test is obtained out of $A_{B'}$ by removing the row which r'' does not satisfy with equality. Consequently, it is evident that $rank[A_{\mathcal{F}}] = d - 2$. □

Corollary 4.4. *If $\|b(r') \wedge b(r'')\| \geq \|b(r''')\| - 1$, then $r' \vee r'' \not\subset r'''$.*

Proof. Assume that both

$$\left\|b\left(r'\right) \wedge b\left(r''\right)\right\| \geq \left\|b\left(r'''\right)\right\| - 1 \text{ and } r' \vee r'' \subset r'''$$

hold. Thus, r' and r'' are nonadjacent and according to the algebraic test,

$$rank[A_{\mathcal{F}}] < d - 2 \ .$$

Furthermore, the matrix $A_{B'''}$ with $B''' = z(r''')$ has exactly one row which does not come up in $A_{\mathcal{F}}$; hence,

$$rank[A_{B'''}] \leq rank[A_{\mathcal{F}}] + 1$$

which contradicts with $rank[A_{B'''}] = d - 1$ as $rank[A_{\mathcal{F}}] < d - 2$. □

Following the work of Terzer [141], the enumeration of adjacent extreme rays can be divided into a *narrowing* and a *verification* phase. In the narrowing phase, Corollary 4.2 is applied in order to exclude from further processing those pairs (r', r'') for which nonadjacency can be shown right away; that is, for those pairs the existence of r''' according to (4.12) is implicitly proven. The remaining pairs, called *adjacency candidates*, are checked once again in the verification phase. Those which satisfy Corollary 4.3

are marked as adjacent. The status of all the others is subject to the combinatorial test; that is, the existence (resp. nonexistence) of r''' has to be proven explicitly. As a result of Corollary 4.4, the search region for r''' can be narrowed to some extent. If r' and r'' are shown to be nonadjacent by finding a matching r''', then r''' is guaranteed to be degenerate. The search region for r''' is thus reduced to the set

$$\mathcal{R}_i^{deg} = \left\{ r \in \left(\mathcal{R}_i^+ \cup \mathcal{R}_i^- \cup \mathcal{R}_i^0 \right) : \| b\left(r \right) \| \geq d \right\}$$

containing only degenerate extreme rays.

In Procedure 1, an implementation covering the enumeration of adjacent extreme rays is given. In that regard, $\mathcal{K}\left(\mathcal{R}\right)$ denotes a container which holds the extreme rays from $\mathcal{R}$. We can think of $\mathcal{K}\left(\mathcal{R}\right)$ as a data structure which encapsulates the elements of $\mathcal{R}$ without revealing the exact internal organization. In order to be applicable within Procedure 1, a container has to meet the following requirements. First, there has to be a function

$$\textbf{create} : \mathcal{R} \to \mathcal{K}\left(\mathcal{R}\right) \tag{4.13}$$

responsible for its initialization. Second, it has to answer two types of queries, a *narrowing query*

$$\textbf{narrow}\left(r', \mathcal{K}\left(\mathcal{R}\right)\right) := \left\{ \left(r', r''\right) \;\middle|\; \begin{array}{l} r'' \in \mathcal{R}, \\ \| b\left(r'\right) \wedge b\left(r''\right) \| \geq d - 2 \end{array} \right\} , \tag{4.14}$$

and a *verification query*

$$\textbf{verify}\left(r', r'', \mathcal{K}\left(\mathcal{R}\right)\right) := \begin{cases} \textbf{false}, & \text{if } \exists r''' \in \mathcal{R} - \left\{r', r''\right\} \\ & \quad b\left(r'\right) \wedge b\left(r''\right) < b\left(r'''\right) \\ & \\ \textbf{true}, & \text{otherwise} . \end{cases} \tag{4.15}$$

It is evident that the efficiency of Procedure 1 depends highly on the performance of the two given queries; hence, a container with favorable data organization is important for obtaining a fast implementation.

4.3.3 Bit Pattern Trees

A container satisfying the requirements of Procedure 1 is easily implemented by using an array to store all elements in a sequential order. Let $\mathcal{A}\left(R\right)$ denote such a container. At first, the running time of $\mathcal{A}\left(R\right)$ may be considered unsatisfactory or even poor. In terms of practical efficacy, however, this container is indeed a feasible alternative and not necessarily a bad choice. When implemented properly, it utilizes both maximal cache efficiency and minimal amount of conditional branches, and thus enables the processor to prefetch data blocks and instructions well ahead of time. A container featuring a better running time in theory is not guaranteed to outperform $\mathcal{A}\left(R\right)$ in practice. On the contrary, the more complicated the implementation gets, the more difficult it is to transform a theoretical superiority into practical results.

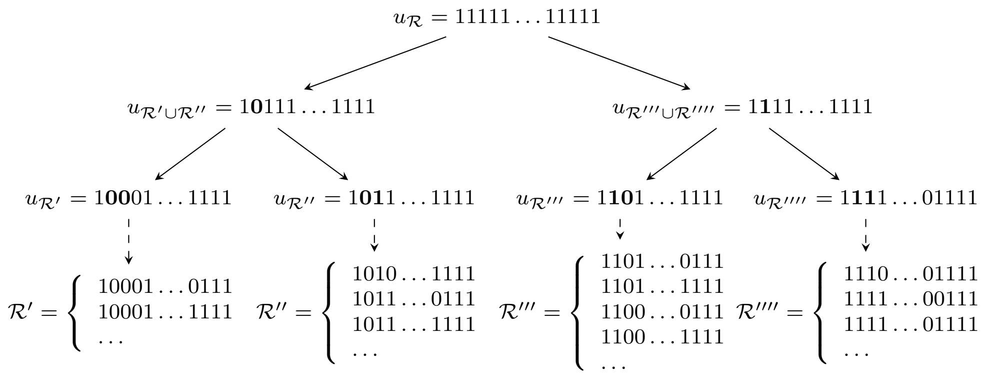
Procedure 1 enumerate: adjacent extreme ray enumeration

```
Input: $\begin{cases} a_j\text{, the matrix row to be processed} \\ \mathcal{R}_i^+\text{, extreme rays lying within the halfspace } \{x \mid a_j x > 0\} \\ \mathcal{R}_i^0\text{, extreme rays lying on the hyperplane } \{x \mid a_j x = 0\} \\ \mathcal{R}_i^-\text{, extreme rays lying within the halfspace } \{x \mid a_j x < 0\} \end{cases}$
Output: $E_i \subseteq \left(\mathcal{R}_i^+ \times \mathcal{R}_i^-\right)$ containing all adjacent extreme rays
$E_i \leftarrow \emptyset$; $E_i^\circ \leftarrow \emptyset$
{⋙ Narrowing Phase ⋘}
$\mathcal{K}\left(\mathcal{R}_i^-\right) \leftarrow$ create $\left(\mathcal{R}_i^-\right)$
for all $r' \in \mathcal{R}_i^+$ do
    $E_i^\circ \leftarrow E_i^\circ \cup$ narrow $\left(r', \mathcal{K}\left(\mathcal{R}_i^-\right)\right)$
end for
{⋙ Verification Phase ⋘}
$\mathcal{R}_i^{deg} \leftarrow \left\{r \in \mathcal{R}_i^+ \cup \mathcal{R}_i^0 \cup \mathcal{R}_i^- : \|b(r)\| \geq d\right\}$
$\mathcal{K}\left(\mathcal{R}_i^{deg}\right) \leftarrow$ create $\left(\mathcal{R}_i^{deg}\right)$
for all $(r', r'') \in E_i^\circ$ do
    if min $\{\|b(r')\|, \|b(r'')\|\} = \|b(r') \wedge b(r'')\| - 1$ then
        $adj \leftarrow$ true
    else
        $adj \leftarrow$ verify $\left(r', r'', \mathcal{K}\left(\mathcal{R}_i^{deg}\right)\right)$
    end if
    if $adj =$ true then
        $E_i \leftarrow E_i \cup \{(r', r'')\}$
    end if
end for
return $E_i$
```

Obtaining a container which operates faster than $\mathcal{A}(R)$ in the general case is a subject which was recently brought up by Terzer and Stelling [143, 144, 141]. The authors introduced a branch-and-bound solution involving *bit pattern trees*, a variation of the well known *k-d* trees [23] adapted for the dd method. Generally, those are binary trees designed to store binary vectors partitioned into finitely many disjoint subsets which are attached to the leaf nodes. Each subset groups together vectors with equal values at predefined positions. Each tree node p is labeled with an *union map* $u_{\mathcal{R}}$ which is an over-approximation vector obtained by building the disjunction of all binary vectors which are reachable from p (see Figure 4.2).

We can think of the binary vectors within a bit pattern tree as extreme rays which are obtainable by reversing the mapping given in (4.3). Let for some tree node p, $\mathcal{R}'$ be the set of extreme rays distributed over all leaf nodes reachable from p. Then, the union map at p

$$u_{\mathcal{R}'} = (u_1, \ldots, u_{d+i})$$



$u_{\mathcal{R}} = 11111\ldots11111$
$u_{\mathcal{R}'\cup\mathcal{R}''} = 10111\ldots1111$
$u_{\mathcal{R}'''\cup\mathcal{R}''''} = 1111\ldots1111$
$u_{\mathcal{R}'} = 10001\ldots1111$
$u_{\mathcal{R}''} = 1011\ldots1111$
$u_{\mathcal{R}'''} = 1101\ldots1111$
$u_{\mathcal{R}''''} = 1111\ldots01111$
$\mathcal{R}' = \{ 10001\ldots0111, 10001\ldots1111, \ldots$
$\mathcal{R}'' = \{ 1010\ldots1111, 1011\ldots0111, 1011\ldots1111, \ldots$
$\mathcal{R}''' = \{ 1101\ldots0111, 1101\ldots1111, 1100\ldots0111, 1100\ldots1111, \ldots$
$\mathcal{R}'''' = \{ 1110\ldots01111, 1111\ldots00111, 1111\ldots01111, \ldots$

Figure 4.2: Structure of bit pattern trees

is a binary vector indicating those rows of A_i which are satisfied with a strict inequality by all rays in $\mathcal{R}'$. They are indexed with zero in $u_{\mathcal{R}'}$ in the sense that

$$u_l = 0 \Leftrightarrow \nexists r \in \mathcal{R}' \left(a_l x = 0 \right) \qquad \text{for all } x \in r \enspace .$$

Note that for each row indexed with one in $u_{\mathcal{R}'}$, we can merely conclude the existence of an extreme ray in $\mathcal{R}'$ satisfying it with equality.

The idea of Terzer and Stelling is to apply bit pattern trees both in the narrowing and the verification phase. First, Corollary 4.2 can be applied on block for multiple pairs $(r', r'') \in \mathcal{R}_i^+ \times \mathcal{R}_i^-$. Thereby, the union maps are used to determine whether certain branches can be eliminated. For any $r' \in \mathcal{R}_i^+$ and $\mathcal{R}_\subseteq^- \subseteq \mathcal{R}_i^-$, it is evident that

$$b\left(r'\right) \wedge u_{\mathcal{R}_\subseteq^-} < d - 2 \Rightarrow b\left(r'\right) \wedge b\left(r''\right) < d - 2 \qquad \text{for all } r'' \in \mathcal{R}_\subseteq^- \enspace .$$

Hence, each branch within the tree can be excluded from the search region if the union map at the corresponding node suggests so.

In the verification phase, a query is performed for all adjacency candidates

$$\left(r', r''\right) \qquad \text{with } e = b\left(r'\right) \wedge b\left(r''\right) \enspace .$$

For all $\mathcal{R}_\subseteq^{deg} \subseteq \mathcal{R}_i^{deg}$, branch elimination rests on the fact that

$$e \not< u_{\mathcal{R}_\subseteq^{deg}} \Rightarrow e \not< b\left(r\right) \qquad \text{for all } r \in \mathcal{R}_\subseteq^{deg} \enspace .$$

5 Combinatorial Test: Comparison of Data Structures

Over the years, multiple tree-based data structures were developed in conjunction with metric space search techniques. Their integration in the adjacency testing process promises a considerable performance gain in practical computations (see Section 5.1). The bit pattern trees of Terzer and Stelling were so far the only representative of that idea. While being already an effective solution, their performance can be further improved by means of several heuristics (see Section 5.2). Yet, there are particular scenarios which bit pattern trees cannot handle well. As a consequence, three alternative data structures were developed: extended bit pattern trees (see Section 5.3), population trees (see Section 5.4) and vantage point trees (see Section 5.5). In practice, however, they turned out to be weaker competitors for general problems (see Section 5.6).

5.1 Generic Binary Tree Container

The efficiency of both narrowing and verification queries is the most performance critical issue concerning adjacency tests (see Section 5.1.1). In that regard, binary search trees are a natural candidate to employ a branch-and-bound technique and thus obtain an expected running time for both queries which is close to logarithmic. The broad variety of partitioning criteria can be handled by a generic framework which both keeps the complexity of the resulting implementation low and ensures its modular expandability (see Section 5.1.2).

5.1.1 Requirements and Complexity

Consider the array based container $\mathcal{A}(R)$ with $|R| = k$. In terms of complexity, the narrowing query requires exactly k operations on each run. The verification query requires maximal k operations if the return value is **false** and exactly k if it is **true**.

Narrowing. In the general case, we can expect that Corollary 4.2 delivers a conclusive result for the majority of elements in $\mathcal{R}$; that is,

$$k \gg |\mathbf{narrow}(r, \mathcal{A}(R))| \quad .$$

A natural optimization of the narrowing query is thus given by applying a branch-and-bound technique on $\mathcal{R}$. The set $\mathcal{R}$ is partitioned by grouping together similar elements and labeling each partition with the criterion unifying the included elements. Consequently, the narrowing query iterates over all generated partitions, but instead of inspecting the elements separately, it first consults the partition label whether Corollary 4.2 is applicable *on block* for all included elements. If so, the whole partition can be skipped right away. Basically, this is the same idea which was already implemented by Terzer and Stelling in the context of the bit pattern trees (see Section 4.3.3). Here, however, the problem is viewed from a more abstract perspective. The goal is to define a framework which allows branch-and-bound to be performed upon multiple criteria; that is, to be able to define and employ different partitioning techniques without reimplementing the whole enumeration process of adjacent extreme rays.

The application of the described branch-and-bound technique targets the execution of narrowing queries in expected logarithmic time. It is important to note, however, that the general problem complexity is linear. Consider, for example, a simplex polytope of any dimension. As each two vertices are adjacent, running a narrowing query for any arbitrary vertex will return all other vertices. This simple example outlines one major theoretical drawback of the branch-and-bound approach; namely, the complexity related to the narrowing query may grow up to $\mathcal{O}\left(k \times \log\left(k\right)\right)$ in the worst case. This unpleasant scenario occurs when no partitions can be eliminated during the search process and hence the query evaluates not only all stored extreme rays but also all partition labels. In other words, in order to achieve a speed-up, we actually risk to concede a slowdown.

It should be noted that the narrowing query is equivalent to a *fixed-radius near neighbor search* [24] if all extreme rays have the same degeneracy degree; that is, if all binary vectors have the same population. This additional constraint allows embedding all binary vectors into a metric space as there is a distance (aka *Hamming distance*) between each two binary vectors which is defined by the $\oplus$-operator as specified in (4.6). Consequently, the distance can be used as a partitioning criterion. The condition from Corollary 4.2 translates to a distance related problem as

$$\left\|b\left(r'\right) \wedge b\left(r''\right)\right\| \geq d-2 \Leftrightarrow \left\|b\left(r'\right)\right\| + \left\|b\left(r''\right)\right\| - \left\|b\left(r'\right) \oplus b\left(r''\right)\right\| \geq 2\left(d-2\right)$$

where $\left\|b\left(r'\right)\right\|$ and $\left\|b\left(r''\right)\right\|$ are constant values. Depending on the density of the elements, a fixed-radius near neighbor search can be performed in time $\mathcal{O}\left(\log\left(k\right)\right)$ [25].

Verification. As already discussed in Section 4.2, the verification query is a simplified form of a partial match query. Assuming that all potential matches are only a tiny fraction of all elements pending for investigation, the search process can be sped up by applying a similar branch-and-bound technique as defined for the narrowing query. By partitioning $\mathcal{R}$, the verification query can be designed to skip entire partitions which are guaranteed to contain no extreme ray r''' satisfying (4.12). Again, we aim for logarithmic running time in the average case.

5.1.2 Implementation

Following the considerations outlined in the previous section, a generic binary tree container was developed to store an arbitrary extreme ray set $\mathcal{R}$. It is called generic in the sense that it allows the definition of different partitioning strategies by providing abstract functions. Depending on their particular definition, $\mathcal{R}$ can be structured in a different way allowing the application of different criteria for reducing the search region.

Definition. The backbone of the generic container is a standard binary tree featuring two major enhancements. First, the set $\mathcal{R}$ is partitioned into finitely many subsets which are then assigned to the tree nodes. Assigning a subset to each node is not mandatory. Second, each node has a label encapsulating certain properties of all extreme rays attached to it or to any subsequent node further down the tree. As a consequence, the label of each node is at most as strict as the labels of its successors. The basic structure of the generic binary tree container is shown in Figure 5.1 where $s_1, \ldots, s_5$ denote labels and $\mathcal{R}'$ and $\mathcal{R}''$ subsets of $\mathcal{R}$.

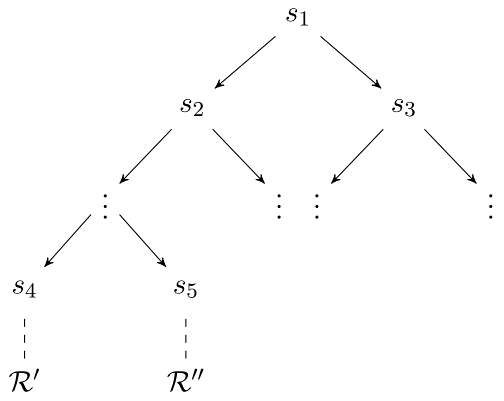


Figure 5.1: Generic binary tree container

Formally, the generic binary tree container is defined as a tuple

$$\mathcal{T}(\mathcal{R}, \Sigma_R) = (V, l, c, E) \qquad \text{where}$$

V is the set of tree nodes,

Σ_R is a set of valid node labels,

$l : V \to \Sigma_R$ is a node labeling function,

$c : V \nrightarrow \wp(\mathcal{R})$ is a partial mapping assigning subsets of $\mathcal{R}$ to the tree nodes, and

$E \subset V \times V$ is the set of all edges.

The initialization of $\mathcal{T}(\mathcal{R}, \Sigma_R)$ for some set of extreme rays $\mathcal{R}$ is given in Procedure 2. Starting with a root node v_{root} and a label s_{root}, the procedure creates two subnodes and makes a recursive call on each of them until some final condition is reached. This can be, for example, a minimal subset size or a maximal tree depth. Procedure 2 assumes the existence of a function

$$\textbf{partition} : \wp(\mathcal{R}) \to \wp(\mathcal{R})^3 \times \Sigma_R^2$$

which splits a subset of $\mathcal{R}$ into three parts, one to be assigned to the node v, and one to proceed the partitioning on each branch with. Furthermore, **partition** defines the labels for both successors of v. Consequently, obtaining a specialization of the generic binary tree container requires the specification of Σ_R and the definition of **partition**.

`Procedure` **2 create**: initialization of a generic binary tree container

```
Input: { R, extreme ray set
       { v, tree node
       { s, characteristic label for R
Output: $\mathcal{T}(\mathcal{R}, \Sigma_R)$, a binary tree container
  if leaf_condition $(\mathcal{R}, s)$ then
    return $(V := \{v\}, l := \{(v, s)\}, c := \{(v, \mathcal{R})\}, E := \emptyset)$
  else
    Create a pair of tree nodes $v'$ and $v''$
    $(\mathcal{R}', \mathcal{R}'', \mathcal{R}''', s', s'') \leftarrow$ partition $(\mathcal{R})$
    $(V', l', c', E') \leftarrow$ create $(\mathcal{R}', v', s')$
    $(V'', l'', c'', E'') \leftarrow$ create $(\mathcal{R}'', v'', s'')$
    if $\mathcal{R}''' \neq \emptyset$ then
      $c''' \leftarrow \{(v, \mathcal{R}''')\}$
    else
      $c''' \leftarrow \emptyset$
    end if
    $V \leftarrow V' \cup V'' \cup \{v\}$
    $l \leftarrow l' \cup l'' \cup \{(v, s)\}$
    $c \leftarrow c' \cup c'' \cup c'''$
    $E \leftarrow E' \cup E'' \cup \{(v, v')\} \cup \{(v, v'')\}$
    return $(V, l, c, E)$
  end if
```

Narrowing. The realization of the narrowing query is based on a depth-first search featuring conditional branch elimination (see Procedure 3). Processing a tree node v begins with the examination of its label $l(v)$. The extreme rays which $l(v)$ applies to are defined by means of a mapping

$$\textbf{reach} : V \rightarrow \wp(\mathcal{R})$$

from tree nodes to subsets of $\mathcal{R}$. Should $l(v)$ reveal that nonadjacency is guaranteed for r' and any $r'' \in \textbf{reach}\,(v)$, then further searching within the current branch is unnecessary and hence omitted. In the opposite case, the query examines the assigned extreme ray subset $c(v)$ and collects all adjacency candidates. Unless v is a leaf node, it proceeds by visiting each direct successor of v. The evaluation of the node label is performed by a forecast function

$$\textbf{feasible} : \mathcal{R} \times \Sigma_R \rightarrow \mathbb{B}$$

which is abstract with regard to the generic binary tree container. It returns **false** when a branch is guaranteed to contain no matches, and **true** otherwise. Obtaining a container specialization requires its concrete definition.

```
Procedure 3 narrow: narrowing query on a generic binary tree container
Input: $\begin{cases} r', \text{ extreme ray} \\ \mathcal{T}(\mathcal{R}, \Sigma_R) = (V, l, c, E), \text{ binary tree container} \\ v \in V, \text{ starting node} \end{cases}$
Output: $C = \{(r', r'') \mid r'' \in \mathbf{reach}(v), \|b(r') \wedge b(r'')\| \geq d - 2\}$
  $C \leftarrow \emptyset$
  if feasible $(r', l(v))$ then
    if $c(v)$ is defined then
      $C \leftarrow \{(r', r'') \mid r'' \in c(v)$ and $\|b(r') \wedge b(r'')\| \geq d - 2\}$
    end if
    for all $v' \in V : (v, v') \in E$ do
      $C \leftarrow C \cup \mathbf{narrow}(r', \mathcal{T}(\mathcal{R}, \Sigma_R), v')$
    end for
  end if
  return $C$
```

Verification. The verification query (see Procedure 4) is structured in a similar manner as the narrowing one. Again, a depth-first search is performed by examining the assigned subset $c(v)$ on each node v and delegating the query onto the successor nodes unless a match has been encountered. The procedure is additionally sped up by omitting certain branches which are guaranteed to produce no match. This is accomplished by applying a forecast function

$$\mathbf{feasible} : \mathcal{R} \times \mathcal{R} \times \Sigma_R \rightarrow \mathbb{B}$$

which evaluates the node label $l(v)$ and predicts the outcome of the query for the node v and its successors. Again, **false** claims the nonexistence of a match within the current branch and **true** suggests the opposite. As in the narrowing query, the forecast function needs to be defined in order to obtain a container specialization.

5.2 Bit Pattern Tree Container

The bit pattern trees of Terzer and Stelling are easily applicable in the context of the generic binary tree container (see Section 5.2.1). A deeper analysis of their performance revealed several structural weaknesses (see Section 5.2.2) which were addressed with three major optimizations:

query bits neutralization (see Section 5.2.3),

cross-narrowing (see Section 5.2.4) and

```
Procedure 4 verify: verification query on a generic binary tree container
Input: $\begin{cases} (r', r''), \text{ adjacency candidate} \\ \mathcal{T}(\mathcal{R}, \Sigma_R) = (V, l, c, E), \text{ binary tree container} \\ v \in V, \text{ starting node} \end{cases}$
Output: $\begin{cases} \textbf{false}, & \text{if } \exists r''' \in \textbf{reach}(v) - \{r', r''\} \, (b(r') \wedge b(r'') < b(r''')) \\ \textbf{true}, & \text{otherwise} \end{cases}$
  if feasible $(r', r'', l(v))$ then
    if $c(v)$ is defined then
      if $\exists r''' \in c(v) - \{r', r''\} \, (b(r') \wedge b(r'') < b(r'''))$ then
        return false
      end if
    end if
    for all $v' \in V : (v, v') \in E$ do
      if not verify $(r', r'', \mathcal{T}(\mathcal{R}, \Sigma_R), v')$ then
        return false
      end if
    end for
  end if
  return true
```

highly degenerate first verification (see Section 5.2.5).

Finally, the process of adjacent rays enumeration was enhanced by the new optimizations (see Section 5.2.6).

5.2.1 Implementation

A *bit pattern tree (bp-tree) container*

$$\mathcal{T}_{bpt}(\mathcal{R}) = \mathcal{T}(\mathcal{R}, \Sigma_R := \{0,1\}^m)$$

is a specialization of the generic binary tree container with each node label being a single binary vector. The length of the binary vectors m corresponds to the overall number of facets. The partitioning process is covered in Procedure 5. It selects some arbitrary bit position $l \in [1, m]$ and splits the set $\mathcal{R}_\circ \subseteq \mathcal{R}$ into two disjoint subsets $\mathcal{R}'_\circ$ and $\mathcal{R}''_\circ$ depending on the value at position l in each $b(r), r \in \mathcal{R}_\circ$. For each of the resulting subsets, a union map is generated and then assigned as a label to the corresponding node.

Let $u_{\mathcal{R}_\circ} = l(v)$ be the label at some tree node v. The forecast function related to the narrowing query

$$\textbf{feasible}\left(r', u_{\mathcal{R}_\circ}\right) := \begin{cases} \textbf{false}, & \text{if } \|b(r') \wedge u_{\mathcal{R}_\circ}\| < d - 2 \\ \textbf{true}, & \text{otherwise} \end{cases}$$

Procedure 5 partition: bp-tree container

```
Input: $\mathcal{R}_\circ$, set of extreme rays
    Select $l \in [1, m]$ with $\exists r', r'' \in \mathcal{R}_\circ\,(\mathbf{value}\,(b\,(r')\,, l) \neq \mathbf{value}\,(b\,(r'')\,, l))$
    $\mathcal{R}'_\circ \leftarrow \{r \in \mathcal{R}_\circ : \mathbf{value}\,(b\,(r)\,, l) = 0\}$
    $\mathcal{R}''_\circ \leftarrow \mathcal{R}_\circ - \mathcal{R}'_\circ$
    $u_{\mathcal{R}'_\circ} \leftarrow \bigvee_{r' \in \mathcal{R}'_\circ} b\,(r');\ u_{\mathcal{R}''_\circ} \leftarrow \bigvee_{r'' \in \mathcal{R}''_\circ} b\,(r'')$
    return $(\mathcal{R}'_\circ, \mathcal{R}''_\circ, \emptyset, u_{\mathcal{R}'_\circ}, u_{\mathcal{R}''_\circ})$
```

interprets $u_{\mathcal{R}_\circ}$ as an over-approximation vector for all extreme rays reachable from v. If the narrowing condition does not hold for $u_{\mathcal{R}_\circ}$, then it also does not hold for any of those extreme rays.

The above concept remains valid for the verification query as well. Consequently, the corresponding forecast function is defined as

$$\mathbf{feasible}\,(r', r'', u_{\mathcal{R}_\circ}) := \begin{cases} \mathbf{false}, & \text{if } b\,(r') \wedge b\,(r'') \not< u_{\mathcal{R}_\circ} \\ \mathbf{true}, & \text{otherwise} \quad . \end{cases}$$

5.2.2 Efficacy and Limitations

Narrowing. The bp-tree container does not automatically guarantee a speed-up in all cases. One point on which the narrowing query's efficacy depends is the population of the union maps. The lower the population the more likely it is to eliminate a branch. The query input r' has also a significant impact on the performance. If $b\,(r')$ is sparsely populated, the branch elimination rate is likely to increase. Those two conditions, however, may still not be sufficient to performing an efficient query. Consider the subset $\mathcal{R}_\circ \subset \mathcal{R}$ and the expression

$$\|b\,(r')\| + \|u_{\mathcal{R}_\circ}\| - \|b\,(r') \oplus u_{\mathcal{R}_\circ}\| < 2\,(d-2) \tag{5.1}$$

which is an alternative form of the narrowing condition

$$\|b\,(r') \wedge u_{\mathcal{R}_\circ}\| < d - 2$$

embodying the Hamming distance. The probability that (5.1) evaluates to **true** decreases when reducing the Hamming distance between $b\,(r')$ and $u_{\mathcal{R}_\circ}$. The distance is minimal if $b\,(r') < u_{\mathcal{R}_\circ}$ with the expression from (5.1) being constantly **false** in that case.

Consider now the partitioning function related to the bp-tree container (see Procedure 5). If for each chosen l, the statistical probability of

$$\mathbf{value}\,(b\,(r')\,, l) = 0$$

turns out to be very high, then we may run into the pitfall of obtaining minimal or close to minimal distances between $b\,(r')$ and the union maps within the tree. Consequently, the choice of l may have a great impact on the overall query performance. In the next section, a strategy for selecting favorable bit positions is discussed.

Verification. One important difference between narrowing and verification queries is the behavior when a match is encountered; namely, the verification query terminates immediately in that case. Therefore, when analyzing the requirements for an efficient partitioning, two cases related to the query result should be differentiated.

On **true**, each element within the container is either explicitly or implicitly checked. The partitioning requirements are thus very similar to those concerning the narrowing query. Let (r', r'') be some arbitrary input. Selecting values for l which are likely to induce

$$\mathbf{value}\,(e, l) = 1 \quad \text{with } e = b\left(r'\right) \wedge b\left(r''\right)$$

are preferable in order to achieve a higher branch elimination.

On **false**, the efficacy considerations should be extended to cover one additional aspect. There is at least one match within the container, thus finding it terminates the query immediately. In view of Corollary 4.4, a match is more likely to be found among highly degenerate extreme rays. Consequently, it is important to organize the extreme rays in a way allowing most prosperous candidates to be checked first. Branch elimination has a lower priority in that case.

5.2.3 Query Bits Neutralization

By making use of the fact that all potential query inputs are actually known before the queries are executed, we can define a forecast function

$$\tau : \wp\,(\mathcal{R}) \times \mathbb{N}_{\leq m} \rightarrow [0, 1]$$

which returns the percentage of rays in some $\mathcal{R}_\circ \subseteq \mathcal{R}$ lying on the facet indexed by some $l \in \mathbb{N}_{\leq m}$. In the context of the binary vector representation

$$b\,(r) = (b_1, \ldots, b_m) \quad ,$$

the function $\tau\,(\mathcal{R}_\circ, l)$ returns the probability of $b_l = 1$ for an arbitrary $r \in \mathcal{R}_\circ$. The return value is a real number in the range between zero and one where a greater value indicates greater probability.

Narrowing. Consider the example given in Figure 5.2a which illustrates the execution of a narrowing query for each element in

$$\mathcal{R}' = \left\{ r'_1, r'_2, r'_3 \right\} \quad .$$

Let $\mathcal{T}_{bpt}\,(\mathcal{R}'')$ be the container on which those queries are executed and let

$$\left\| b\left(r'\right) \wedge b\left(r''\right) \right\| \geq d - 2 \quad \text{with } r' \in \mathcal{R}',\ r'' \in \mathcal{R}'' \text{ and } d = 7$$

be the narrowing condition. Assume also that $\mathcal{T}_{bpt}\left(\mathcal{R}''\right)$ is constructed by always picking the first viable bit position at each partitioning step; that is, the minimal possible value for l is selected on each invocation of Procedure 5. Consequently, for all $r' \in \mathcal{R}'$,

$$\textbf{feasible}\left(r', u_{\mathcal{R}''_{\circ}}\right) = \textbf{true} \text{ and } \textbf{feasible}\left(r', u_{\mathcal{R}''_{\triangleright}}\right) = \textbf{true} \ .$$

Hence, Procedure 3 will force the search in both subsequent branches after evaluating the node labels $u_{\mathcal{R}''_{\circ}}$ and $u_{\mathcal{R}''_{\triangleright}}$. The outcome of the evaluation for $u_{\mathcal{R}''_{\triangleright}}$, however, can be changed by selecting different values for l during the partitioning process. It is easy to see that

$$\tau\left(\mathcal{R}', 2\right) = 0 \quad \text{while } \tau\left(\mathcal{R}', 3\right) = 1 \ ,$$

suggesting that each input has the value 0 at position 2 and 1 at position 3. Consequently, it can be foreseen that defining $l = 2$ in Procedure 5 is a bad choice when aiming for high branch elimination rate. In contrast, $l = 3$ is a much more promising candidate. Figure 5.2b illustrates the reduction of the search region in the latter case.

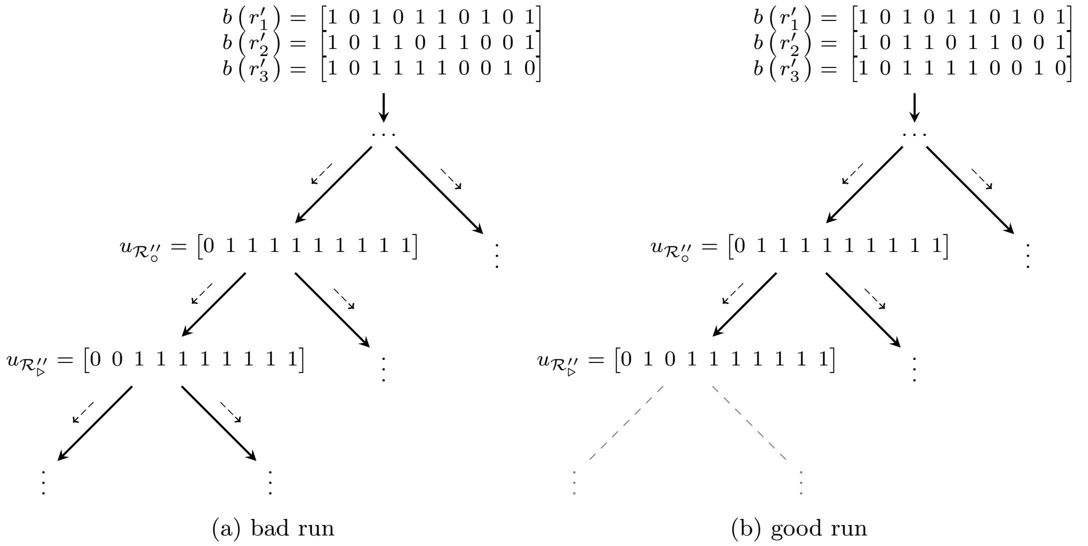

$b\left(r'_1\right) = \left[1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\right]$
$b\left(r'_2\right) = \left[1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\right]$
$b\left(r'_3\right) = \left[1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\right]$
$u_{\mathcal{R}''_{\circ}} = \left[0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\right]$
$u_{\mathcal{R}''_{\triangleright}} = \left[0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\right]$
(a) bad run
$b\left(r'_1\right) = \left[1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\right]$
$b\left(r'_2\right) = \left[1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\right]$
$b\left(r'_3\right) = \left[1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\right]$
$u_{\mathcal{R}''_{\circ}} = \left[0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\right]$
$u_{\mathcal{R}''_{\triangleright}} = \left[0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\right]$
(b) good run

Figure 5.2: Query bits neutralization (narrowing query)

Considering merely the probability values emerging from the input elements in $\mathcal{R}'$ does not always deliver the desired effect. Another factor to be taken into account is the statistical data emerging from the container elements $\mathcal{R}''$. Let for some l, $\tau\left(\mathcal{R}', l\right)$ return a value close to 1. The position l may still be a bad candidate if $\tau\left(\mathcal{R}'', l\right)$ is another close to 1 value. In general, we may expect a satisfactory result by selecting l such that

$$\tau\left(\mathcal{R}', l\right) - \tau\left(\mathcal{R}'', l\right)$$

is maximal. This strategy is likely to create an unbalanced tree with long paths to the leaves on the left side and short ones on the right. In view of the narrowing query, the short paths are likely to be visited and the long ones to be skipped.

Verification. The considerations from the previous paragraph related to the choice of l remain mostly valid when dealing with verification queries as well. Consider again the bp-tree containers given in Figure 5.2, but this time from the perspective of three verification queries executed with

$$b\left(r'\right) \wedge b\left(r''\right) \in \left\{ \begin{array}{l} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}, \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}, \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \end{array} \right\}$$

as input. With regard to the evaluation of $u_{\mathcal{R}''_{\circ}}$ and $u_{\mathcal{R}''_{\triangleright}}$, the behavior of both containers is identical to that observed in the narrowing query. The first container (see Figure 5.2a) yields

$$\textbf{feasible}\left(r', r'', u_{\mathcal{R}''_{\triangleright}}\right) = \textbf{true}$$

for all inputs and thus forces the examination of the subsequent branches, while the second one (see Figure 5.2b) reduces the search region as a result of

$$\textbf{feasible}\left(r', r'', u_{\mathcal{R}''_{\triangleright}}\right) = \textbf{false} \ .$$

When dealing with degenerate problems, the majority of narrowing queries would produce several candidates for verification each. In the end, the number of executed verification queries is likely to be much greater than the number of narrowing ones. Consequently, performing a statistical evaluation on the verification input data may easily generate a huge bottleneck and thus eliminate any potential performance gain. As given in Procedure 1, the verification query input is an element of $\mathcal{R}_i^+ \times \mathcal{R}_i^-$ coupled with a container $\mathcal{K}\left(\mathcal{R}_i^{deg}\right)$, which holds all degenerate rays from $\mathcal{R}_i^+$, $\mathcal{R}_i^-$ and $\mathcal{R}_i^0$. Assuming that an independent statistical evaluation is available for each of those three sets, the following facts are to be taken into consideration when defining a partitioning strategy for $\mathcal{T}_{bpt}\left(\mathcal{R}_i^{deg}\right)$.

(i) If for some position l, $\tau\left(\mathcal{R}_i^-, l\right) \ll 1$ or $\tau\left(\mathcal{R}_i^+, l\right) \ll 1$, then the probability of $\textbf{value}\left(e, l\right) = 1$ for any

$$e = b\left(r'\right) \wedge b\left(r''\right) \quad \text{with } \left(r', r''\right) \in \mathcal{R}_i^+ \times \mathcal{R}_i^-$$

is minimal. Consequently, partitioning upon the position l is likely to cause a query behavior close to that given in Figure 5.2a.

(ii) If $\tau\left(\mathcal{R}_i^-, l\right) \gg 0$, $\tau\left(\mathcal{R}_i^+, l\right) \gg 0$ and $\tau\left(\mathcal{R}_i^0, l\right) \gg 0$, then the probability of

$$\textbf{value}\left(e, l\right) = 1$$

is considerably high, but so is also $\tau\left(\mathcal{R}_i^{deg}, l\right)$. An example of how the resulting tree may look like is given in Figure 5.3 where $|\mathcal{R}'| \approx |\mathcal{R}''| \approx |\mathcal{R}'''| \ll |\mathcal{R}''''|$. In the best case, a verification query would skip examining $\mathcal{R}'$, $\mathcal{R}''$ and $\mathcal{R}'''$, but as those contain only a small portion of all container elements, any potential speed-up can be considered negligible.

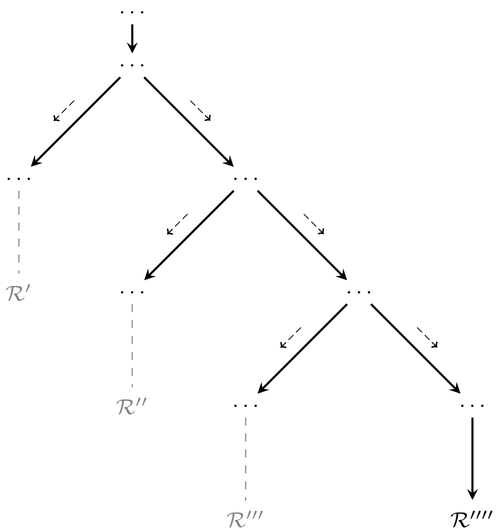


Figure 5.3: Example for an ineffective verification container

As a consequence of (i) and (ii), it is clear that only one potential scenario is likely to deliver any performance speed-up, namely when

$$\tau\left(\mathcal{R}_i^-, l\right) \gg 0,\ \tau\left(\mathcal{R}_i^+, l\right) \gg 0 \text{ and } \tau\left(\mathcal{R}_i^0, l\right) \ll 1 \enspace .$$

In that case, extreme rays from $\mathcal{R}_i^0$ which cannot produce a match would be effectively eliminated from the search. Consequently, defining l such that

$$\mathbf{min}\left\{\tau\left(\mathcal{R}_i^-, l\right), \tau\left(\mathcal{R}_i^+, l\right)\right\} \times \left(\tau\left(\mathcal{R}_i^0, l\right) - 1\right)$$

is maximal appears to be the most promising strategy in the context of the verification query.

5.2.4 Cross-Narrowing

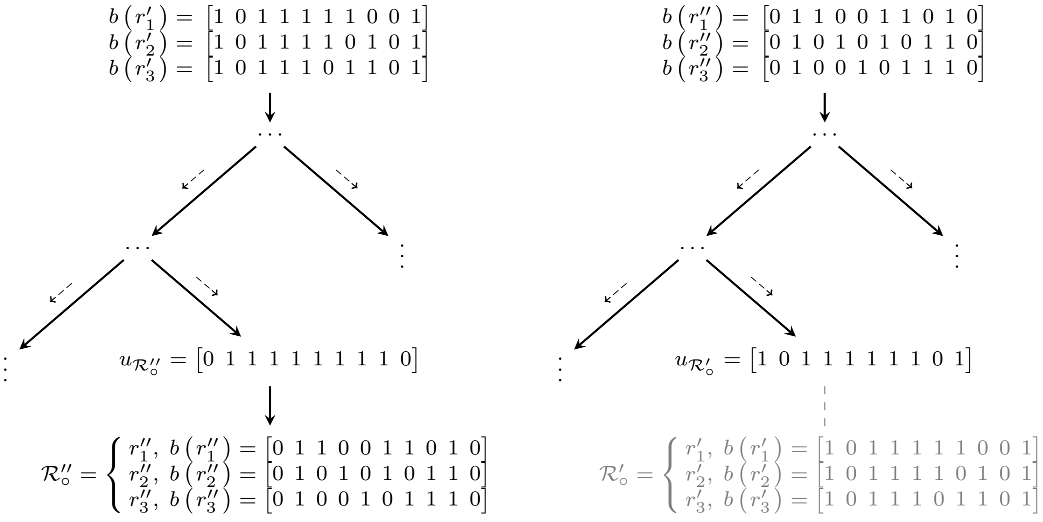
Generally, a union map $u_{\mathcal{R}''_\circ}$ assigned to some tree node v is likely to have a considerably higher population than any $b\left(r''\right)$ with $r'' \in \mathcal{R}''_\circ$ an extreme ray reachable from v. This fact emerges from the definition of $u_{\mathcal{R}''_\circ}$ as a disjunction of multiple binary vectors. An unfavorable situation occurs when a narrowing query is invoked with a highly degenerate extreme ray r' as an input; that is, the binary vector $b\left(r'\right)$ is highly populated as well. Consider the example given in Figure 5.4a. It illustrates a bp-tree container together with three narrowing query inputs $r'_1, r'_2, r'_3 \in \mathcal{R}'_\circ \subset \mathbb{R}^6$. The corresponding search queries involve the enumeration of all extreme rays r'' satisfying

$$\left\|b\left(r'\right) \wedge b\left(r''\right)\right\| \geq d - 2 \quad \text{with } r' \in \mathcal{R}'_\circ \text{ and } d = 6 \enspace .$$

Even though the above condition does not hold for any of the ray combinations, an explicit examination of $\mathcal{R}''_\circ$ is performed in all three query runs as

$$\mathbf{feasible}\left(r', u_{\mathcal{R}''_\circ}\right) = \mathbf{true} \quad \text{for all } r' \in \mathcal{R}'_\circ \enspace .$$

In other words, the label falsely claims the possibility of a match in $\mathcal{R}''_{\circ}$.



$b(r'_1) = [1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1]$
$b(r'_2) = [1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1]$
$b(r'_3) = [1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1]$
$u_{\mathcal{R}''_{\circ}} = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0]$
$\mathcal{R}''_{\circ} = \{ r''_1,\ b(r''_1) = [0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0]; r''_2,\ b(r''_2) = [0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0]; r''_3,\ b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0]$
$b(r''_1) = [0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0]$
$b(r''_2) = [0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0]$
$b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0]$
$u_{\mathcal{R}'_{\circ}} = [1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1]$
$\mathcal{R}'_{\circ} = \{ r'_1,\ b(r'_1) = [1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1]; r'_2,\ b(r'_2) = [1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1]; r'_3,\ b(r'_3) = [1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1]$

(a) bad run

(b) good run

Figure 5.4: Cross-Narrowing on bit pattern tree containers

In order to reach an optimal behavior, the roles of $\mathcal{R}'_{\circ}$ and $\mathcal{R}''_{\circ}$ have to be switched. The search region can be effectively reduced by embedding $\mathcal{R}'_{\circ}$ in a tree and using $\mathcal{R}''_{\circ}$ as a source for query inputs (see Figure 5.4b). Due to

$$\left\| b\left(r''\right) \wedge u_{\mathcal{R}'_{\circ}} \right\| < 4 \quad \text{with } r'' \in \mathcal{R}''_{\circ} \ ,$$

all three queries skip the explicit examination of $\mathcal{R}'_{\circ}$. This difference of the query behavior is easily explained when looking at the population of the individual extreme rays where

$$\left\| b\left(r'_1\right) \right\| = \left\| b\left(r'_2\right) \right\| = \left\| b\left(r'_3\right) \right\| = 7 \text{ and } \left\| b\left(r''_1\right) \right\| = \left\| b\left(r''_2\right) \right\| = \left\| b\left(r''_3\right) \right\| = 5 \ .$$

The lower the population of two binary vectors, the lower the expected population of their conjunction. If we assume that the union maps are highly populated anyway, the only way of achieving a higher branch elimination rate is by constructing queries with inputs inducing minimally populated binary vectors.

5.2.5 Highly Degenerate First Verification

The degeneracy degree poses an important criterion to find matches quickly within verification queries. Performing a query for two extreme rays r' and r'' is pointless if the search region contains solely rays r satisfying

$$\left\| b\left(r\right) \right\| \leq \left\| b\left(r'\right) \wedge b\left(r''\right) \right\| - 1 \ .$$

The nonexistence of a match is provable right away in that case (see Corollary 4.4). Unfortunately, the bp-tree container does not consider the degeneracy degree when grouping together the individual rays. The only criterion involved in the partitioning process is their position to a certain facet. Consequently, low degenerate rays may be stored together with highly degenerate ones which causes unnecessary checks. Consider the example given in Figure 5.5 featuring a bp-tree container $\mathcal{T}_{bpt}(\mathcal{R}')$. It illustrates two verification queries defined by the input pairs (r_1, r_2) and (r_3, r_4).

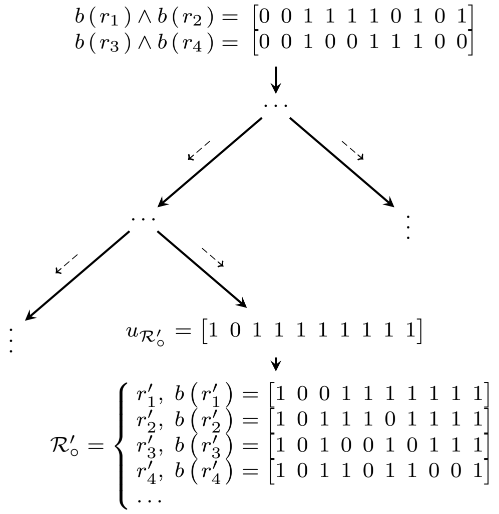

$b(r_1) \wedge b(r_2) = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1]$
$b(r_3) \wedge b(r_4) = [0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0]$
...
...
$u_{\mathcal{R}'_\circ} = [1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$\mathcal{R}'_\circ =$
$r'_1,\ b(r'_1) = [1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$r'_2,\ b(r'_2) = [1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1]$
$r'_3,\ b(r'_3) = [1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1]$
$r'_4,\ b(r'_4) = [1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1]$
...

Figure 5.5: Verification queries on bit pattern tree containers

According to Corollary 4.4, the rays $r'_3, r'_4 \in \mathcal{R}'_\circ \subset \mathcal{R}'$ cannot produce a match for (r_1, r_2), but are examined anyway as other elements in $\mathcal{R}'_\circ$ are indeed viable candidates. In order to reduce those unnecessary checks, one can define a threshold $t_v \geq d$ and split the bp-tree container into two separate ones

$\mathcal{T}_{bpt}(\mathcal{R}'_{hd})$ with $\mathcal{R}'_{hd} = \{r \in \mathcal{R}' \mid \|b(r)\| > t_v\}$ containing highly degenerate rays (see Figure 5.6a), and

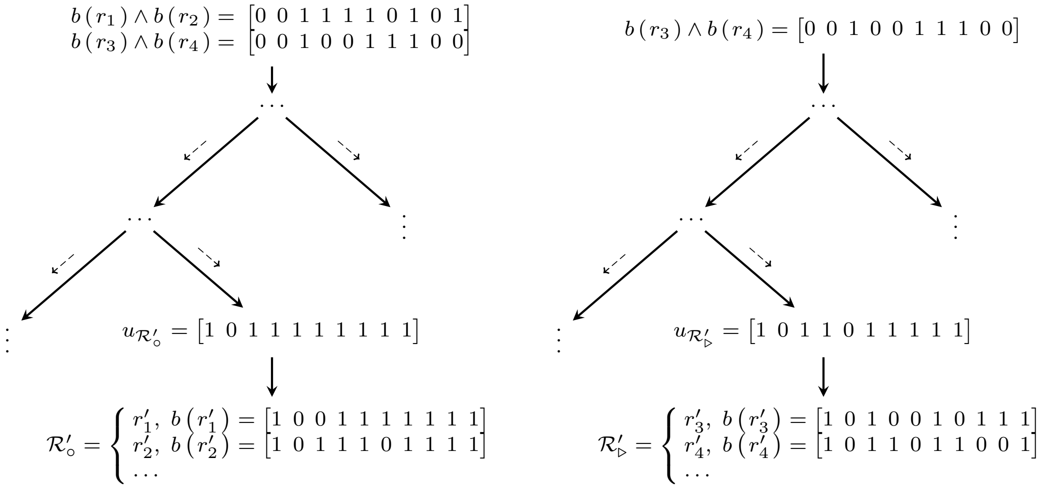
$\mathcal{T}_{bpt}(\mathcal{R}'_{ld})$ with $\mathcal{R}'_{ld} = \{r \in \mathcal{R}' \mid d \leq \|b(r)\| \leq t_v\}$ containing low degenerate ones (see Figure 5.6b).

As a consequence, examining $\mathcal{T}_{bpt}(\mathcal{R}'_{ld})$ is necessary only for the input pair (r_3, r_4).

In general, highly degenerate rays are more likely to produce a match than low degenerate ones; hence, the higher the population of $r' \in \mathcal{R}'$, the greater the chance that

$$b(r_3) \wedge b(r_4) < b(r') \quad .$$

Therefore, performing the query first on $\mathcal{T}_{bpt}(\mathcal{R}'_{hd})$ seems to be a logical choice in view of the probability to encounter a match. If nonadjacency is detected on that first run, the examination of $\mathcal{T}_{bpt}(\mathcal{R}'_{ld})$ is skipped right away.


$b(r_1) \wedge b(r_2) = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1]$
$b(r_3) \wedge b(r_4) = [0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0]$
$u_{\mathcal{R}'_\circ} = [1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$\mathcal{R}'_\circ = \{ r'_1,\ b(r'_1) = [1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1]; r'_2,\ b(r'_2) = [1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1]; \ldots$
$b(r_3) \wedge b(r_4) = [0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0]$
$u_{\mathcal{R}'_\triangleright} = [1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1]$
$\mathcal{R}'_\triangleright = \{ r'_3,\ b(r'_3) = [1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1]; r'_4,\ b(r'_4) = [1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1]; \ldots$

(a) $\mathcal{T}_{bpt}(\mathcal{R}'_{hd})$

(b) $\mathcal{T}_{bpt}(\mathcal{R}'_{ld})$

Figure 5.6: Highly degenerate first verification

5.2.6 The Adjacency Test Revisited

In Procedure 6, the enumeration of adjacent extreme rays is enhanced by cross-narrowing and highly degenerate first verification. In contrast to the initial version in Procedure 1, a threshold t_n is introduced to differentiate between low degenerate and highly degenerate extreme rays. Thereon, $\mathcal{R}_i^+$ and $\mathcal{R}_i^-$ are split into two subsets each, separating extreme rays with low and high degeneracy degree. Consequently, the generation of all adjacency candidates

$$\bigcup_{r \in \mathcal{R}_i^+} \textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_i^-\right)\right)$$

can be redefined as

$$\begin{aligned} &\bigcup_{r \in \mathcal{R}_{ld}^-} \left(\textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_{ld}^+\right)\right) \cup \textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^+\right)\right)\right) \\ \cup &\bigcup_{r \in \mathcal{R}_{ld}^+} \textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^-\right)\right) \cup \bigcup_{r \in \mathcal{R}_{hd}^+} \textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^-\right)\right) \end{aligned}$$

which maximizes the number of narrowing queries executed with low degenerate input.

A similar differentiation is applied when dealing with verification queries. Depending on a degeneracy degree threshold t_v, the set $\mathcal{R}_i^{deg}$ is split into two subsets $\mathcal{R}_{ld}^{deg}$ and $\mathcal{R}_{hd}^{deg}$ holding low and highly degenerate extreme rays. The execution of

$$\textbf{verify}\left(r', r'', \mathcal{T}_{bpt}\left(\mathcal{R}_i^{deg}\right)\right)$$

is thus equivalent to the conjunction of two separate queries

$$\textbf{verify}\left(r', r'', \mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^{deg}\right)\right) \textbf{ and verify}\left(r', r'', \mathcal{T}_{bpt}\left(\mathcal{R}_{ld}^{deg}\right)\right) \quad .$$

It is worth mentioning that both optimizations can be disabled by setting the corresponding thresholds t_n and t_v to zero. In that case, Procedure 6 effectively reduces to Procedure 1.

```
Procedure 6 enumerate_fast: optimized adjacent extreme ray enumeration
Input: $\begin{cases} \mathcal{R}_i^+, \text{ extreme rays lying within the halfspace } \{x \mid a_j x > 0\} \\ \mathcal{R}_i^0, \text{ extreme rays lying on the hyperplane } \{x \mid a_j x = 0\} \\ \mathcal{R}_i^-, \text{ extreme rays lying within the halfspace } \{x \mid a_j x < 0\} \\ t_n, \text{ narrowing threshold} \\ t_v, \text{ verification threshold} \end{cases}$
Output: $E_i \subseteq \left(\mathcal{R}_i^+ \times \mathcal{R}_i^-\right)$ containing all adjacent extreme rays
  $E_i \leftarrow \emptyset$; $E_i^\circ \leftarrow \emptyset$
  {⋙ Narrowing Phase feat. Cross-Narrowing ⋘}
  $\mathcal{R}_{ld}^+ \leftarrow \left\{r \in \mathcal{R}_i^+ : \|b(r)\| \leq t_n\right\}$; $\mathcal{R}_{hd}^+ \leftarrow \mathcal{R}_i^+ - \mathcal{R}_{ld}^+$
  $\mathcal{T}_{bpt}\left(\mathcal{R}_{ld}^+\right) \leftarrow \textbf{create}\left(\mathcal{R}_{ld}^+\right)$; $\mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^+\right) \leftarrow \textbf{create}\left(\mathcal{R}_{hd}^+\right)$
  $\mathcal{R}_{ld}^- \leftarrow \left\{r \in \mathcal{R}_i^- : \|b(r)\| \leq t_n\right\}$; $\mathcal{R}_{hd}^- \leftarrow \mathcal{R}_i^- - \mathcal{R}_{ld}^-$
  $\mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^-\right) \leftarrow \textbf{create}\left(\mathcal{R}_{hd}^-\right)$
  for all $r \in \mathcal{R}_{ld}^-$ do
    $E_i^\circ \leftarrow E_i^\circ \cup \textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_{ld}^+\right)\right) \cup \textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^+\right)\right)$
  end for
  for all $r \in \mathcal{R}_i^+$ do
    $E_i^\circ \leftarrow E_i^\circ \cup \textbf{narrow}\left(r, \mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^-\right)\right)$
  end for
  {⋙ Verification Phase feat. Highly Degenerate First ⋘}
  $\mathcal{R}_i^{deg} \leftarrow \left\{r \in \mathcal{R}_i^+ \cup \mathcal{R}_i^0 \cup \mathcal{R}_i^- : \|b(r)\| \geq d\right\}$
  $\mathcal{R}_{ld}^{deg} \leftarrow \left\{r \in \mathcal{R}_i^{deg} : \|b(r)\| \leq t_v\right\}$; $\mathcal{R}_{hd}^{deg} \leftarrow \mathcal{R}_i^{deg} - \mathcal{R}_{ld}^{deg}$
  $\mathcal{T}_{bpt}\left(\mathcal{R}_{ld}^{deg}\right) \leftarrow \textbf{create}\left(\mathcal{R}_{ld}^{deg}\right)$; $\mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^{deg}\right) \leftarrow \textbf{create}\left(\mathcal{R}_{hd}^{deg}\right)$
  for all $(r', r'') \in E_i^\circ$ do
    if $\textbf{min}\{\|b(r')\|, \|b(r'')\|\} = \|b(r') \wedge b(r'')\| + 1$ then
      $adj \leftarrow$ true
    else
      $adj \leftarrow \textbf{verify}\left(r', r'', \mathcal{T}_{bpt}\left(\mathcal{R}_{hd}^{deg}\right)\right)$
      if $adj =$ true and $\|b(r') \wedge b(r'')\| < t_v - 1$ then
        $adj \leftarrow \textbf{verify}\left(r', r'', \mathcal{T}_{bpt}\left(\mathcal{R}_{ld}^{deg}\right)\right)$
      end if
    end if
    if $adj =$ true then
      $E_i \leftarrow E_i \cup \{(r', r'')\}$
    end if
  end for
  return $E_i$
```

5.3 Extended Bit Pattern Tree Container

Motivation. Despite the proposed optimizations, the bp-tree container still has several weaknesses which arise in particular scenarios. Consider, for example, the narrowing query illustrated in Figure 5.7a where

$$\left\|b\left(r'\right) \wedge b\left(r''\right)\right\| \geq d-2 \quad \text{with } d=6$$

is the corresponding narrowing condition. An explicit examination of the set $\mathcal{R}''_\circ$ is performed although no extreme ray $r'' \in \mathcal{R}''_\circ$ satisfies the narrowing condition. The reason for this unsatisfactory behavior roots in the abstract characterization of $\mathcal{R}''_\circ$ which $u_{\mathcal{R}''_\circ}$ delivers. The only information which can be extracted from $u_{\mathcal{R}''_\circ}$ is the existence of one facet not containing any extreme rays from $\mathcal{R}''_\circ$. With a minimal effort, however, the node label can be extended with additional data which is sufficient to forecast the nonexistence of an adjacency candidate in $\mathcal{R}''_\circ$. This involves the definition of

a *cut map* $c_{\mathcal{R}''_\circ} = \bigwedge\limits_{r'' \in \mathcal{R}''_\circ} b\left(r''\right)$ obtained by building the conjunction of all corresponding binary vectors and

a *population coefficient* $p_{\mathcal{R}''_\circ} = \mathbf{max}\left\{\left\|b\left(r''\right)\right\| \mid r'' \in \mathcal{R}''_\circ\right\}$ which is the maximal binary vector population emerging from the extreme rays in $\mathcal{R}''_\circ$.

Now, it is easy to verify that

$$\left\|b\left(r''\right)\right\| - \left\|b\left(r''\right) \oplus \bar{b}\left(r'\right)\right\| < d-2 \Rightarrow \left\|b\left(r'\right) \wedge b\left(r''\right)\right\| < d-2 \quad \text{for all } r'' \in \mathcal{R}''_\circ$$

which can be further generalized to

$$p_{\mathcal{R}''_\circ} - \left\|c_{\mathcal{R}''_\circ} \oplus \bar{b}\left(r'\right)\right\| < d-2 \Rightarrow \left\|b\left(r'\right) \wedge b\left(r''\right)\right\| < d-2 \tag{5.2}$$

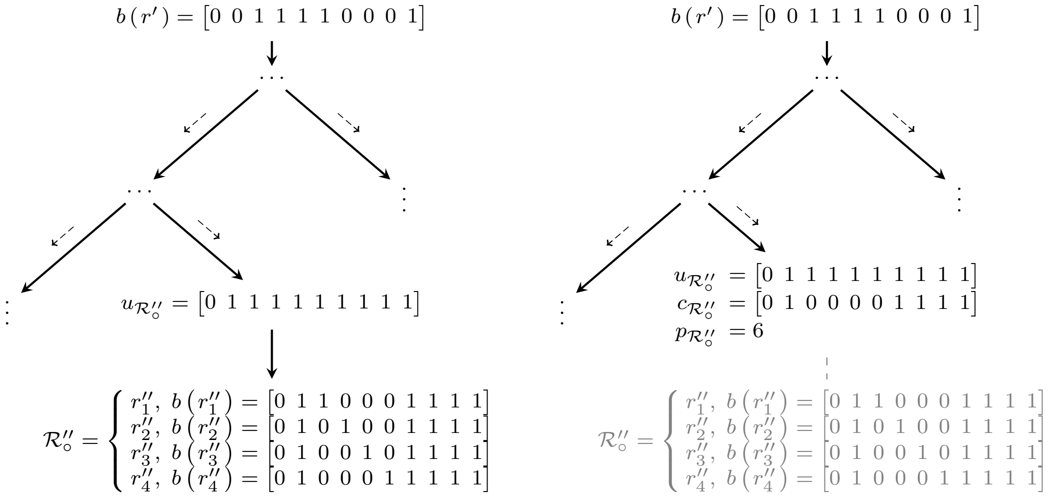
by substituting $b\left(r''\right)$ with $c_{\mathcal{R}''_\circ}$ and $\left\|b\left(r''\right)\right\|$ with $p_{\mathcal{R}''_\circ}$. Consider now the container in Figure 5.7b which embodies the proposed extensions. The inequation from (5.2) holds and hence implies the nonexistence of an adjacency candidate $\left(r', r''\right)$ with $r'' \in \mathcal{R}''_\circ$.

The proposed node label extension can be used in a similar manner to perform an additional branch elimination within verification queries. Consider again the container given in Figure 5.7a. Let

$$\left(r_1, r_2\right) \quad \text{with } e = b\left(r_1\right) \wedge b\left(r_2\right) = \begin{bmatrix}0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\end{bmatrix}$$

act as a verification query input. The container behavior is again unsatisfactory as $\mathcal{R}''_\circ$ is explicitly examined despite the obvious nonexistence of a match. However, by employing a cut map and a population coefficient, the outcome of the search procedure on $\mathcal{R}''_\circ$ can be predicted. It is easy to show that

$$\left\|b\left(r''\right)\right\| - \left\|b\left(r''\right) \oplus \bar{e}\right\| < \|e\| \Rightarrow e \not\leq b\left(r''\right) \quad \text{for all } r'' \in \mathcal{R}''_\circ \ .$$


$b(r') = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1]$
$u_{\mathcal{R}''_\circ} = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$\mathcal{R}''_\circ = \{ r''_1, b(r''_1) = [0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1]$
$r''_2, b(r''_2) = [0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1]$
$r''_3, b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1]$
$r''_4, b(r''_4) = [0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1]$
$b(r') = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1]$
$u_{\mathcal{R}''_\circ} = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$c_{\mathcal{R}''_\circ} = [0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1]$
$p_{\mathcal{R}''_\circ} = 6$
$\mathcal{R}''_\circ = \{ r''_1, b(r''_1) = [0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1]$
$r''_2, b(r''_2) = [0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1]$
$r''_3, b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1]$
$r''_4, b(r''_4) = [0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1]$

(a) bp-tree container

(b) ebp-tree container

Figure 5.7: Regular vs. extended bit pattern tree containers (narrowing query)

Again, substituting $b(r'')$ with $c_{\mathcal{R}''_\circ}$ and $\|b(r'')\|$ with $p_{\mathcal{R}''_\circ}$ maximizes the left side of the inequation, thus

$$p_{\mathcal{R}''_\circ} - \left\| c_{\mathcal{R}''_\circ} \oplus \bar{e} \right\| < \|e\| \Rightarrow e \not\leq b\left(r''\right) \qquad \text{for all } r'' \in \mathcal{R}''_\circ \tag{5.3}$$

remains valid. Consequently, by running the query on the extended container from Figure 5.7b, the nonexistence of a ray $r'' \in \mathcal{R}''_\circ$ satisfying the verification condition is predicted and the corresponding branch eliminated.

Implementation. The considerations from the previous paragraph can be formalized into an *extended bit pattern tree (ebp-tree) container*

$$\mathcal{T}_{ebpt}(\mathcal{R}) = \mathcal{T}\left(\mathcal{R}, \Sigma_R := \left(\{0,1\}^m \times \{0,1\}^m \times \mathbb{N}\right)\right) \ .$$

It is an enhancement of the regular bp-tree container in the sense that additional data is being assigned to each node label. Accordingly, its partitioning function (see Procedure 7) is an enhanced form of Procedure 5. The same applies to the narrowing forecast function

$$\textbf{feasible}\left(r', \left(u_{\mathcal{R}_\circ}, c_{\mathcal{R}_\circ}, p_{\mathcal{R}_\circ}\right)\right) := \begin{cases} \textbf{false}, & \text{if } \|b(r') \wedge u_{\mathcal{R}_\circ}\| < d-2 \textbf{ or} \\ & \left\| c_{\mathcal{R}_\circ} \oplus \bar{b}(r') \right\| + d - 2 > p_{\mathcal{R}_\circ} \\ \textbf{true}, & \text{otherwise} \end{cases}$$

which incorporates the implication given in (5.2). Correspondingly, the verification forecast function is extended by the implication given in (5.3); hence,

$$\textbf{feasible}\left(r', r'', \left(u_{\mathcal{R}_\circ}, c_{\mathcal{R}_\circ}, p_{\mathcal{R}_\circ}\right)\right) := \begin{cases} \textbf{false}, & \text{if } e \not\leq u_{\mathcal{R}_\circ} \textbf{ or} \\ & \|c_{\mathcal{R}_\circ} \oplus \bar{e}\| + \|e\| > p_{\mathcal{R}_\circ} \\ \textbf{true}, & \text{otherwise} \end{cases}$$

with e denoting $b(r') \wedge b(r'')$.

```
Procedure 7 partition: ebp-tree container
Input: $\mathcal{R}_\circ$, set of extreme rays
  Select $l \in [1, m]$ with $\exists r', r'' \in \mathcal{R}_\circ \left(\mathbf{value}(b(r'), l) \neq \mathbf{value}(b(r''), l)\right)$
  $\mathcal{R}'_\circ \leftarrow \{r \in \mathcal{R}_\circ : \mathbf{value}(b(r), l) = 0\}$
  $\mathcal{R}''_\circ \leftarrow \mathcal{R}_\circ - \mathcal{R}'_\circ$
  $u_{\mathcal{R}'_\circ} \leftarrow \bigvee_{r' \in \mathcal{R}'_\circ} b(r')$; $c_{\mathcal{R}'_\circ} \leftarrow \bigwedge_{r' \in \mathcal{R}'_\circ} b(r')$; $p_{\mathcal{R}'_\circ} = \mathbf{max}\{\|b(r')\| \mid r' \in \mathcal{R}'_\circ\}$
  $u_{\mathcal{R}''_\circ} \leftarrow \bigvee_{r'' \in \mathcal{R}''_\circ} b(r'')$; $c_{\mathcal{R}''_\circ} \leftarrow \bigwedge_{r'' \in \mathcal{R}''_\circ} b(r'')$; $p_{\mathcal{R}''_\circ} = \mathbf{max}\{\|b(r'')\| \mid r'' \in \mathcal{R}''_\circ\}$
  return $\left(\mathcal{R}'_\circ, \mathcal{R}''_\circ, \emptyset, \left(u_{\mathcal{R}'_\circ}, c_{\mathcal{R}'_\circ}, p_{\mathcal{R}'_\circ}\right), \left(u_{\mathcal{R}''_\circ}, c_{\mathcal{R}''_\circ}, p_{\mathcal{R}''_\circ}\right)\right)$
```

5.4 Population Tree Container

Motivation. Unfortunately, the enhancements presented in the previous section are not always sufficient to close the weaknesses of the bp-tree container. Consider, for example, the container $\mathcal{T}_{bpt}(\mathcal{R}'')$ given in Figure 5.8a where r' is a narrowing query input. The query involves the enumeration of all rays $r'' \in \mathcal{R}''_\circ \subset \mathcal{R}''$ which satisfy

$$\left\|b\left(r'\right) \wedge b\left(r''\right)\right\| \geq d - 2 \quad \text{with } d = 6 \ .$$

It is evident that no extreme ray in $\mathcal{R}''_\circ$ satisfies the above condition; hence, the desired action would be to simply eliminate the whole branch from the search region. Yet, the union map $u_{\mathcal{R}''_\circ}$ suggests the potential presence of a viable candidate in $\mathcal{R}''_\circ$ and thus forces its explicit examination.

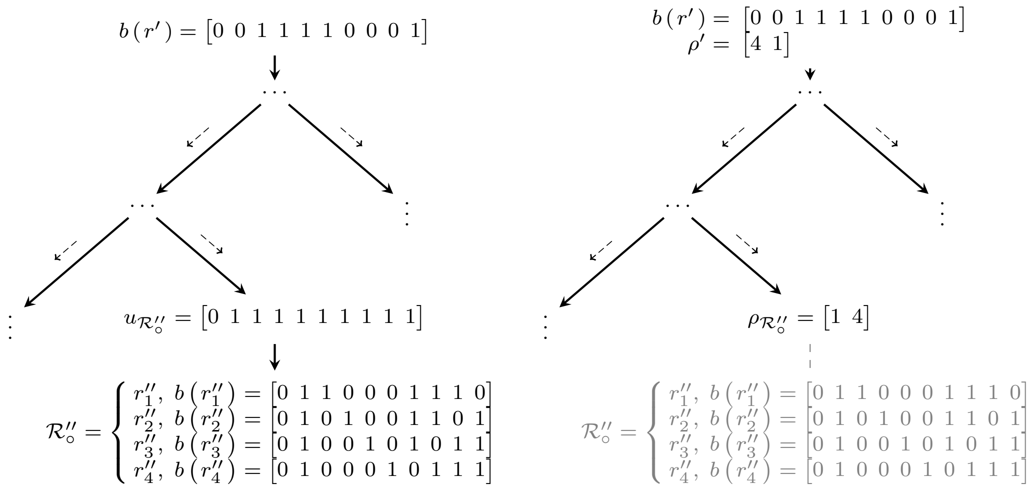
The query performance hardly improves when an ebp-tree container is employed. Attaching a cut map

$$c_{\mathcal{R}''_\circ} = \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

to the terminal node does not introduce a sufficiently strong characterization of $\mathcal{R}''_\circ$ and thus does not change the outcome of the forecast. Bit patterns do not seem to be suitable for that particular scenario. A closer look at the binary vectors emerging from $\mathcal{R}''_\circ$ reveals that those are equally populated, but have only two positions with matching values. This constellation obstructs the extraction of a bit pattern which is strong enough to deliver meaningful information about the extreme rays in $\mathcal{R}''_\circ$. It is evident, however, that those vectors are similar in a way; hence, another criterion to extract that similarity should be found.

Consider the following two observations. First, reducing each binary vector $b(r'')$, $r'' \in \mathcal{R}''_\circ$ to a four-dimensional subvector containing only positions 3, 4, 5 and 6 implies an equal population of 1. Second, applying the same reduction to positions 7, 8, 9 and 10 implies an equal again population of 3. Those observations lead to the definition of two *fragmentation masks*

$$f_1 = \begin{bmatrix}1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\end{bmatrix} \text{ and } f_2 = \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1\end{bmatrix}$$


$b(r') = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1]$
$u_{\mathcal{R}''_\circ} = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$\mathcal{R}''_\circ = \begin{cases} r''_1,\ b(r''_1) = [0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0] \\ r''_2,\ b(r''_2) = [0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1] \\ r''_3,\ b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1] \\ r''_4,\ b(r''_4) = [0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1] \end{cases}$
$b(r') = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1]$
$\rho' = [4\ 1]$
$\rho_{\mathcal{R}''_\circ} = [1\ 4]$
$\mathcal{R}''_\circ = \begin{cases} r''_1,\ b(r''_1) = [0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0] \\ r''_2,\ b(r''_2) = [0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1] \\ r''_3,\ b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1] \\ r''_4,\ b(r''_4) = [0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1] \end{cases}$

(a) bp-tree container

(b) pop-tree container

Figure 5.8: Bit pattern vs. population tree containers (narrowing query)

which indicate regions of the binary vectors featuring similar population. Thereon, the union map $u_{\mathcal{R}''_\circ}$ is substituted by a vector

$$\rho_{\mathcal{R}''_\circ} = \begin{bmatrix} \rho''_1 := \mathbf{max}\left\{ \|f_1 \wedge b(r'')\| \mid r'' \in \mathcal{R}''_\circ \right\} \\ \rho''_2 := \mathbf{max}\left\{ \|f_2 \wedge b(r'')\| \mid r'' \in \mathcal{R}''_\circ \right\} \end{bmatrix}^T \tag{5.4}$$

storing the maximal population which emerges from the conjunction of the two masks with each $b(r''), r'' \in \mathcal{R}''_\circ$. Such a vector

$$\rho' = \begin{bmatrix} \rho'_1 := \|f_1 \wedge b(r')\| \\ \rho'_2 := \|f_2 \wedge b(r')\| \end{bmatrix}^T \tag{5.5}$$

is also generated for the query input r' by applying both masks on $b(r')$. In Figure 5.8b, the so modified container is illustrated. It is easy to verify that

$$\mathbf{min}\left\{\rho'_1, \rho''_1\right\} + \mathbf{min}\left\{\rho'_2, \rho''_2\right\} < d - 2 \ ,$$

which is sufficient to show the nonexistence of an extreme ray $r'' \in \mathcal{R}''_\circ$ satisfying the narrowing condition.

Consider now the example given in Figure 5.9a, which illustrates a verification query on a bp-tree container $\mathcal{T}_{bpt}(\mathcal{R})$. The input (r', r'') does not require the explicit examination of $\mathcal{R}_\circ \subset \mathcal{R}$ as no match with respect to the verification condition exists. Again, with the extracted union map $u_{\mathcal{R}_\circ}$, this fact cannot be foreseen and thus the search region is not being reduced. However, with the technique presented in the previous example, the problematic tree branch can be effectively eliminated. Using again the fragment masks f_1 and f_2, two vectors

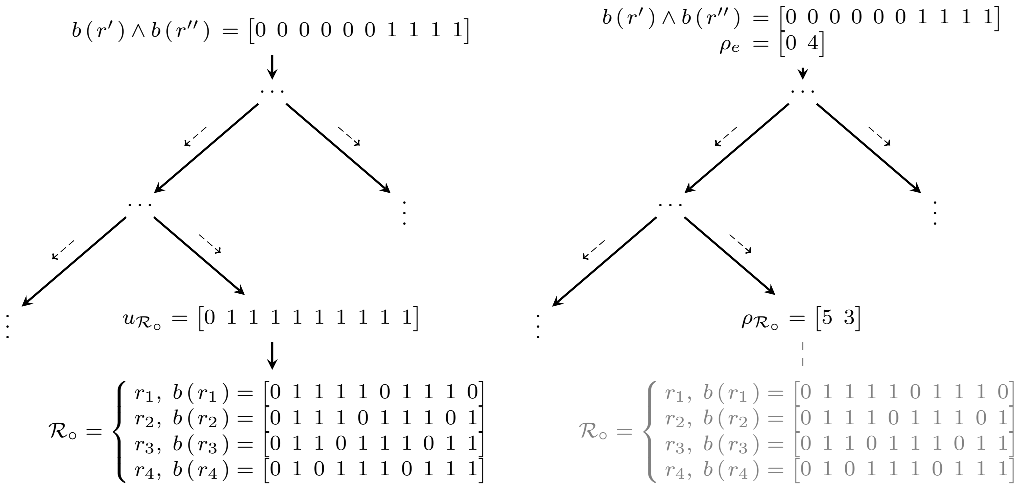
$$\rho_{\mathcal{R}_\circ} = \begin{bmatrix}\rho_1\ \rho_2\end{bmatrix} \text{ and } \rho_e = \begin{bmatrix}\rho'_1\ \rho'_2\end{bmatrix}$$

are extracted. Their specification is identical with the one given in (5.4) and (5.5). For the latter, the vector

$$e = b\left(r'\right) \wedge b\left(r''\right)$$

is used instead of $b\left(r'\right)$. The nonexistence of a match becomes now obvious as

$$\begin{aligned} \rho_2 < \rho_2' &\Rightarrow \left\| b\left(r\right) \wedge f_2 \right\| < \left\| e \wedge f_2 \right\| \quad \text{for all } r \in \mathcal{R}_\circ \\ &\Rightarrow e \not< b\left(r\right) \quad \text{for all } r \in \mathcal{R}_\circ \ . \end{aligned}$$


$b(r') \wedge b(r'') = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1]$
$u_{\mathcal{R}_\circ} = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$\mathcal{R}_\circ =$
$r_1,\ b(r_1) = [0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0]$
$r_2,\ b(r_2) = [0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1]$
$r_3,\ b(r_3) = [0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1]$
$r_4,\ b(r_4) = [0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1]$
$b(r') \wedge b(r'') = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1]$
$\rho_e = [0\ 4]$
$\rho_{\mathcal{R}_\circ} = [5\ 3]$
$\mathcal{R}_\circ =$
$r_1,\ b(r_1) = [0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0]$
$r_2,\ b(r_2) = [0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1]$
$r_3,\ b(r_3) = [0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1]$
$r_4,\ b(r_4) = [0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1]$

(a) bp-tree container

(b) pop-tree container

Figure 5.9: Bit pattern vs. population tree containers (verification query)

Implementation. The branch elimination technique from the previous paragraph is now formalized to a *population tree (pop-tree) container*. Its creation requires some additional effort in comparison to the solutions involving bit patterns. For one thing, a tuple of non-overlapping fragmentation masks

$$f = \left(f_1, \ldots, f_k\right) \in \{0,1\}^m \times \ldots \times \{0,1\}^m$$

to cover all bit positions is needed; that is,

$$\sum_{i=1}^{k} \left\| f_i \right\| = m \text{ and } \forall i,j \in \mathbb{N}_{\leq k} \left(i \neq j \Rightarrow \left\| f_i \wedge f_j \right\| = 0\right) \ .$$

Furthermore, each extreme ray r to be stored into the pop-tree container is mapped to a k-dimensional *population vector*

$$\rho\left(r, f\right) = \left[\rho_1 \ \ldots \ \rho_k\right]$$

by applying f on $b(r)$ in the sense that

$$\rho_i = \|b(r) \wedge f_i\| \qquad \text{for all } i \in [1,k] \quad .$$

Each node v of the resulting tree is labeled with a population vector maximized over all $\rho(r,f)$ with r some arbitrary extreme ray reachable from v. Consequently, the pop-tree container is formally defined as

$$\mathcal{T}_{pt}(\mathcal{R}) := \mathcal{T}\left(\mathcal{R}, \Sigma_R := \mathbb{N}^k\right) \quad .$$

The extreme rays are partitioned upon the values of their population vectors. The partitioning technique itself is equivalent to that of a *k-d* tree. Splitting an extreme ray set $\mathcal{R}_\circ$ into two subsets involves selecting a target mask $f_j, j \in [1,k]$ together with some population threshold $p_{\mathcal{R}_\circ}$, and then separating from the rest all extreme rays with population vectors satisfying $\rho_j \leq p_{\mathcal{R}_\circ}$ (see Procedure 8).

```
Procedure 8 partition: pop-tree container
Input: $\mathcal{R}_\circ$, set of extreme rays
  Select $f_j$ such that $\exists r', r'' \in \mathcal{R}_\circ \left(\|b(r') \wedge f_j\| \neq \|b(r'') \wedge f_j\|\right)$
  $P \leftarrow \{\|b(r) \wedge f_j\| \mid r \in \mathcal{R}_\circ\}$
  Select $p_{\mathcal{R}'_\circ}$ such that $\mathbf{min}\{P\} \leq p_{\mathcal{R}'_\circ} < \mathbf{max}\{P\}$
  $p_{\mathcal{R}''_\circ} \leftarrow \mathbf{max}\{P\}$
  $\mathcal{R}'_\circ \leftarrow \left\{r \in \mathcal{R}_\circ : \|b(r) \wedge f_j\| \leq p_{\mathcal{R}'_\circ}\right\}$
  $\mathcal{R}''_\circ \leftarrow \mathcal{R}_\circ - \mathcal{R}'_\circ$
  $\rho_{\mathcal{R}'_\circ} \leftarrow \left[\rho'_1 \dots \rho'_k\right]$ with $\rho'_i = \mathbf{max}\{\|b(r') \wedge f_i\| \mid r' \in \mathcal{R}'_\circ\}, i \in [1,k]$
  $\rho_{\mathcal{R}''_\circ} \leftarrow \left[\rho''_1 \dots \rho''_k\right]$ with $\rho''_i = \mathbf{max}\{\|b(r'') \wedge f_i\| \mid r'' \in \mathcal{R}''_\circ\}, i \in [1,k]$
  return $\left(\mathcal{R}'_\circ, \mathcal{R}''_\circ, \emptyset, \rho_{\mathcal{R}'_\circ}, \rho_{\mathcal{R}''_\circ}\right)$
```

With regard to narrowing queries, a decision whether a particular branch requires further processing is taken after building the population vector ρ' of the input r' and then minimizing it by using the corresponding node label $\rho_{\mathcal{R}_\circ}$; that is, both vectors are merged by selecting the minimal value at each component. The so defined minimized vector has a sum of its components not greater than

$$\left\|b(r') \wedge b(r'')\right\|$$

for any r'' stored in the underlying branch. That defines the narrowing forecast function

$$\mathbf{feasible}\left(r', \rho_{\mathcal{R}_\circ}\right) := \begin{cases} \mathbf{false}, & \text{if } \sum\limits_{i=1}^{k} \mathbf{min}\{\rho_i, \|b(r') \wedge f_i\|\} < d-2 \\ \mathbf{true}, & \text{otherwise} \end{cases}$$

with $\rho_{\mathcal{R}_\circ} = \left[\rho_1 \dots \rho_k\right]$.

With regard to verification queries, a particular branch requires further inspection only if the population vector in the label is at least equal in all components to the population vector built from the query input (r', r''). Otherwise, there exists a fragmentation mask f_i such that

$$b(r') \wedge b(r'') \wedge f_i \not< b(r) \wedge f_i \Rightarrow b(r') \wedge b(r') \not< b(r)$$

with r an extreme ray lying within the inspected branch. The above implication defines the verification forecast function

$$\mathbf{feasible}\left(r', r'', \rho_{\mathcal{R}_\circ}\right) := \begin{cases} \mathbf{false}, & \text{if } \exists i \in [1,k]\,(\|e \wedge f_i\| > \rho_i) \\ \mathbf{true}, & \text{otherwise} \end{cases}$$

with $e = b(r') \wedge b(r'')$ and $\rho_{\mathcal{R}_\circ} = \left[\rho_1 \ \ldots \ \rho_k\right]$.

5.5 Vantage Point Tree Container

Motivation. Population trees have a noteworthy relation to vantage point trees (vp-trees), a well known data structure applicable in metric spaces [155]. Consider the narrowing query example in Figure 5.10a. The problem has already been dealt with in the previous section; the given bp-tree container $\mathcal{T}_{bpt}(\mathcal{R}'')$ fails to eliminate the branch holding $\mathcal{R}''_\circ \subset \mathcal{R}''$ for the narrowing query input r' with respect to the narrowing condition

$$\left\|b(r') \wedge b(r'')\right\| \geq d - 2 \qquad \text{with } r'' \in \mathcal{R}''_\circ \quad \text{and } d = 6.$$

The idea behind vp-trees is to mark some of the extreme rays as vantage points and characterize all other extreme rays by determining their similarity to the vantage points. In Figure 5.10b, this technique is applied in the context of the current example. First, the extreme ray r''_1 is marked as a vantage point and moved into the node label. Next, the similarity degree of all other rays towards the vantage point is determined and attached to the node label as well. The latter involves finding a *characteristic vector*

$$p_{\mathcal{R}''_\circ} = \left[p'' \ \bar{p}''\right] \qquad \text{where}$$

p'' is the maximal population emerging from the conjunction of $b(r''_1)$ with any other binary vector $b(r''_i)$ with $i \in [2,4]$ and

$\bar{p}''$ is the maximal population emerging from the conjunction of the inversed vantage point vector $\bar{b}(r''_1)$ with any other binary vector $b(r''_i)$.

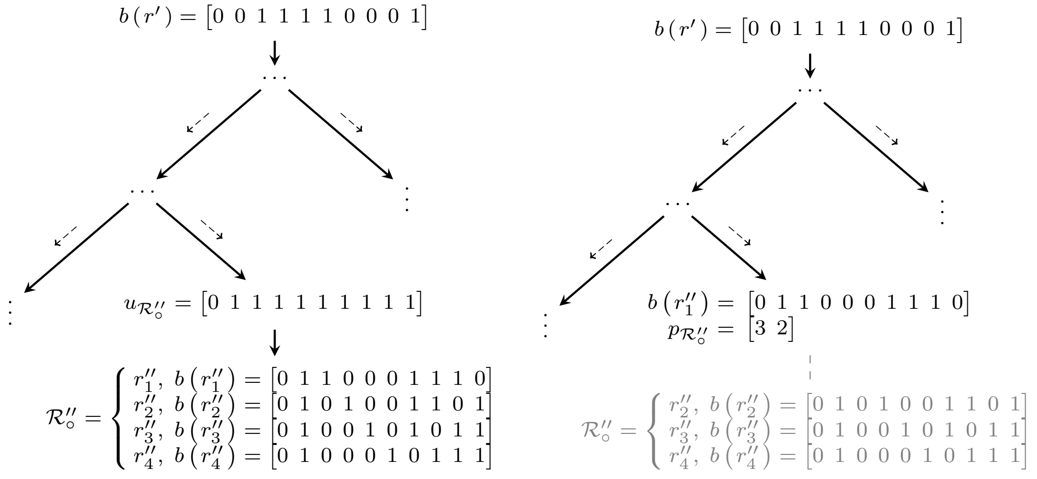
It is now easy to show that for the query input r' and any $r'' \in \mathcal{R}''_\circ$, the value of

$$\left\|b(r') \wedge b(r'')\right\|$$

is utmost

$$\mathbf{min}\left\{p'', \left\|b(r''_1) \wedge b(r')\right\|\right\} + \mathbf{min}\left\{\bar{p}'', \left\|\bar{b}(r''_1) \wedge b(r')\right\|\right\} \ . \tag{5.6}$$

Consequently, the nonexistence of an extreme ray in $\mathcal{R}''_\circ$ adjacent to r' is guaranteed as the over-approximation given in (5.6) delivers a value of three.


$b(r') = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1]$
$u_{\mathcal{R}''_\circ} = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$
$\mathcal{R}''_\circ = \begin{cases} r''_1,\ b(r''_1) = [0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0] \\ r''_2,\ b(r''_2) = [0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1] \\ r''_3,\ b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1] \\ r''_4,\ b(r''_4) = [0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1] \end{cases}$
$b(r') = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1]$
$b(r''_1) = [0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0]$
$p_{\mathcal{R}''_\circ} = [3\ 2]$
$\mathcal{R}''_\circ = \begin{cases} r''_2,\ b(r''_2) = [0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1] \\ r''_3,\ b(r''_3) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1] \\ r''_4,\ b(r''_4) = [0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1] \end{cases}$

(a) bp-tree container

(b) vp-tree container

Figure 5.10: Bit pattern vs. vantage point tree containers (narrowing query)

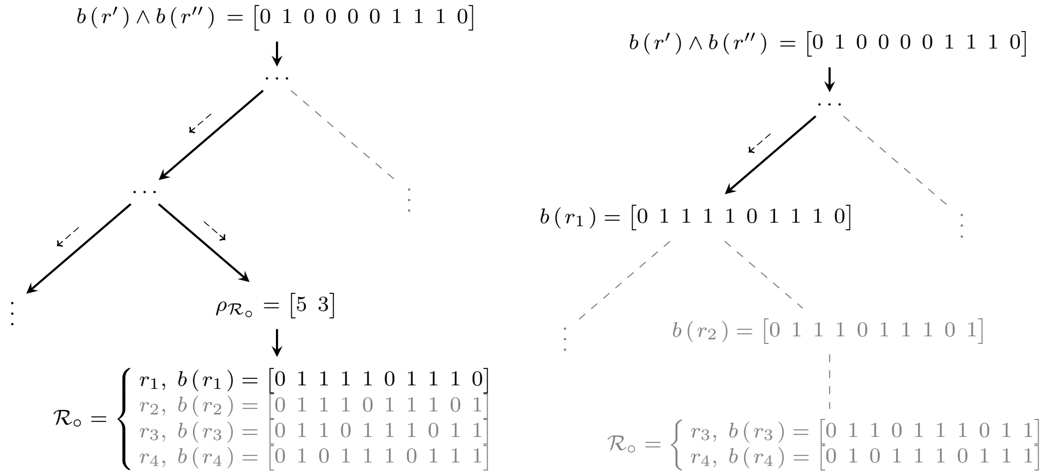
The construction of the characteristic vector $p_{\mathcal{R}''_\circ}$ is a process equivalent to finding the ray in $\mathcal{R}''_\circ$ with the highest similarity to the vantage point and the one with the highest deviation from it. In a metric space, high deviation would automatically imply low similarity and vice versa, meaning that p'' and $\bar{p}''$ could not emerge from the same element, unless there is only one element in the set. It should be pointed out, however, that in the general case, a single extreme ray may very well define both p'' and $\bar{p}''$ as there is no guarantee that all extreme rays are equally populated.

Applying the vantage point technique within verification queries has one major advantage compared to the other container types presented so far, namely that nonadjacency can be detected more quickly. This conjecture rests on the fact that extreme rays are also planted within intermediate nodes and thus a verification query can terminate with a match without even reaching any terminal node. In that regard, consider the verification query example given in Figure 5.11a which features a pair of extreme rays (r', r'') the nonadjacency of which is being proven after encountering r_1. By using the vantage point technique illustrated in Figure 5.11b, certain elements are effectively pulled up the tree and therefore examined earlier than in other container types. If those elements, such as r_1 in the given example, deliver a match, then the whole verification procedure terminates more quickly.

Implementation. The vp-tree container can be viewed as a pop-tree container where the tuple of fragmentation masks f is not static for the whole tree, but dynamically selected for each node. At each partitioning step, an element $r_{vp} \in \mathcal{R}$ is selected defining

$$f = \left(b\left(r_{vp}\right), \bar{b}\left(r_{vp}\right)\right) \ .$$

This guarantees a unique mask tuple for each node, because r_{vp} remains attached to that node and is not passed further down the tree (see Procedure 9); hence, it cannot



b(r′) ∧ b(r″) = [0 1 0 0 0 0 1 1 1 0]
ρ_{R∘} = [5 3]
R∘ =
r1, b(r1) = [0 1 1 1 1 0 1 1 1 0]
r2, b(r2) = [0 1 1 1 0 1 1 1 0 1]
r3, b(r3) = [0 1 1 0 1 1 1 0 1 1]
r4, b(r4) = [0 1 0 1 1 1 0 1 1 1]
b(r′) ∧ b(r″) = [0 1 0 0 0 0 1 1 1 0]
b(r1) = [0 1 1 1 1 0 1 1 1 0]
b(r2) = [0 1 1 1 0 1 1 1 0 1]
R∘ =
r3, b(r3) = [0 1 1 0 1 1 1 0 1 1]
r4, b(r4) = [0 1 0 1 1 1 0 1 1 1]

(a) pop-tree container

(b) vp-tree container

Figure 5.11: Population vs. vantage point tree containers (verification query)

be used as a vantage point again. Formally, the vp-tree container is defined as

$$\mathcal{T}_{vpt}(\mathcal{R}) = \mathcal{T}\left(\mathcal{R}, \Sigma_R := \left(\{0,1\}^m \times \mathbb{N}^2\right)\right)$$

with each label being a combination of a vantage point vector and a characteristic one.

`Procedure` **9 partition**: vp-tree container

```
Input: $\mathcal{R}_\circ$, set of extreme rays
  Select some $r_{vp} \in \mathcal{R}_\circ$
  $f_{vp} \leftarrow b(r_{vp})$
  $P \leftarrow \{\|b(r) \wedge f_{vp}\| \mid r \in \mathcal{R}_\circ \textbf{ and } r \neq r_{vp}\}$
  Select $p_{\mathcal{R}'_\circ}$ such that $\textbf{min}\{P\} \leq p_{\mathcal{R}'_\circ} < \textbf{max}\{P\}$
  $p_{\mathcal{R}''_\circ} \leftarrow \textbf{max}\{P\}$
  $\mathcal{R}'_\circ \leftarrow \left\{r \in \mathcal{R}_\circ : \|b(r) \wedge f_{vp}\| \leq p_{\mathcal{R}'_\circ}\right\}$
  $\mathcal{R}''_\circ \leftarrow \mathcal{R}_\circ - (\mathcal{R}'_\circ + r_{vp})$
  $\bar{p}_{\mathcal{R}'_\circ} = \textbf{max}\left\{\left\|b(r') \wedge \bar{f}_{vp}\right\| \mid r' \in \mathcal{R}'_\circ\right\}$
  $\bar{p}_{\mathcal{R}''_\circ} = \textbf{max}\left\{\left\|b(r'') \wedge \bar{f}_{vp}\right\| \mid r'' \in \mathcal{R}''_\circ\right\}$
  return $\left(\mathcal{R}'_\circ, \mathcal{R}''_\circ, \{r_{vp}\}, \left(f_{vp}, p_{\mathcal{R}'_\circ}, \bar{p}_{\mathcal{R}'_\circ}\right), \left(f_{vp}, p_{\mathcal{R}''_\circ}, \bar{p}_{\mathcal{R}''_\circ}\right)\right)$
```

The narrowing forecast function

$$\textbf{feasible}\left(r', f_{vp}, p_{\mathcal{R}_\circ}, \bar{p}_{\mathcal{R}_\circ}\right) := \begin{cases} \textbf{false}, & \text{if } p_{\mathcal{R}_\circ} + \left\|b(r') \wedge \bar{f}_{vp}\right\| < d-2 \textbf{ or} \\ & \|b(r') \wedge f_{vp}\| + \bar{p}_{\mathcal{R}_\circ} < d-2 \\ \textbf{true}, & \text{otherwise} \end{cases}$$

is a simplified form of the function related to pop-tree containers which admits only two fragments. The same applies to the verification one

$$\textbf{feasible}\left(r', r'', f_{vp}, p_{\mathcal{R}_\circ}, \bar{p}_{\mathcal{R}_\circ}\right) := \begin{cases} \textbf{true}, & \text{if } p_{\mathcal{R}_\circ} < \left\|e \wedge f_{vp}\right\| \textbf{ and} \\ & \bar{p}_{\mathcal{R}_\circ} < \left\|e \wedge \bar{f}_{vp}\right\| \\ \textbf{false}, & \text{otherwise} \end{cases}$$

with e denoting $b\,(r') \wedge b\,(r'')$.

5.6 Experimental Results

In order to compare all container types in practical conditions, a benchmark consisting of problems in various dimensions and featuring a different level of degeneracy was compiled. All problems have been randomly generated with truncation being used in order to enforce high degeneracy levels. They are given in Table 5.1 together with the achieved speed-up when applying the presented optimizations. It should be noted that all execution times throughout this section are given in seconds and were obtained by merely counting the number of the resulting extreme rays; that is, the exact generators were not calculated.

	degeneracy	A	$\lvert\mathcal{R}\rvert$	avg. deg. degree	speed-up
random_4_16_64	low	64 × 16	1 914 024	0.57	54%
random_4_16_96		96 × 16	450 714	0.75	18%
random_6_64_96		96 × 64	3 023 180	0.43	63%
random_8_128_157		157 × 128	1 107 211	0.53	77%
random_6_16_104	medium	104 × 16	2 779 391	1.1	66%
random_4_64_112		112 × 64	1 790 042	1.4	64%
random_4_128_172		172 × 128	1 520 534	1.8	67%
random_4_16_112	high	112 × 16	448 658	3.3	55%
random_16_68_115		115 × 68	185 834	5.8	67%
random_12_128_174		174 × 128	134 994	6.0	58%
random_8_20_160	very high	160 × 20	199 080	9.7	31%
random_16_64_179		179 × 64	271 255	19	37%
random_24_128_277		277 × 128	174 939	32	28%

Table 5.1: Benchmark for adjacency tests

Bit Pattern Trees. The three presented optimizations

(i) cross-narrowing,

(ii) highly degenerate first verification and

(iii) query bits neutralization

were experimentally tested, both independently and in collaboration with each other (see Table 5.2). While the first two delivered a small to moderate performance gain, query bits neutralization turned out to be a very strong heuristic. Moreover, it covers both narrowing and verification, and does not require any particular parameterization. For cross-narrowing and highly degenerate first, the thresholds t_n and t_v have a significant impact on the final result and should be selected carefully.

	bp-trees	heuristics			
		cross-nar.	h.deg.first	q.b.n.	all in one
RANDOM_4_16_64	157.106	130.596	156.579	76.484	**71.878**
RANDOM_4_16_96	27.312	25.266	26.600	**22.329**	23.943
RANDOM_6_64_96	100.676	74.519	100.689	41.340	**37.088**
RANDOM_8_128_157	82.033	50.846	78.657	23.110	**18.840**
RANDOM_6_16_104	603.381	464.026	532.831	337.051	**207.898**
RANDOM_4_64_112	242.210	213.363	184.510	118.912	**86.396**
RANDOM_4_128_172	325.128	304.534	236.182	149.108	**106.803**
RANDOM_4_16_112	90.146	77.344	83.071	48.124	**40.395**
RANDOM_16_68_115	152.866	154.935	103.005	52.878	**50.702**
RANDOM_12_128_174	118.506	119.588	93.662	64.728	**50.332**
RANDOM_8_20_160	179.535	148.894	176.147	128.562	**123.437**
RANDOM_16_64_179	1 309.003	1 238.839	1 242.471	1 223.103	**828.286**
RANDOM_24_128_277	1 637.366	1 581.302	1 592.305	1 384.321	**1185.609**

Table 5.2: Computational results for bp-tree containers

Extended Bit Pattern Trees. The ebp-tree container promises a potentially better branch elimination rate at the cost of additional operations due to the more complex forecast functions. In practice, the direct application of those containers did not deliver any performance improvement neither in the narrowing nor in the verification phase (see Table 5.3). On the contrary, it even caused a slowdown which became significant with growing degeneracy. Still, ebp-tree containers showed some advantages in the low degenerate domain. For example, a beneficial scenario emerged when they were used as complementary containers to handle solely low degenerate rays in the context of cross-narrowing and highly degenerate first verification. In the last two columns of Table 5.3, the consequences of that strategy are illustrated. It should be noted, however, that its advantages vanished for the most part when query bits neutralization was additionally applied.

Vantage Point/Population Trees. In terms of performance, vp-trees remained well behind bp-trees (see Table 5.4). This fact is partially related to the more complex

	bp-trees	ebp-trees		cross-nar. & h.deg.first	
		narrowing	verification	bp-trees	ebp/bp-trees
random_4_16_64	157.106	157.565	158.142	127.015	**123.278**
random_4_16_96	27.312	27.611	27.985	**25.010**	25.124
random_6_64_96	100.676	104.933	104.146	72.384	**66.761**
random_8_128_157	82.033	82.489	83.517	44.529	**42.942**
random_6_16_104	603.381	606.589	605.168	381.232	**379.219**
random_4_64_112	242.210	244.354	254.940	**144.845**	145.054
random_4_128_172	325.128	326.959	339.305	**202.322**	203.121
random_4_16_112	90.146	91.323	99.107	**70.392**	70.415
random_16_68_115	152.866	153.261	176.784	145.510	**145.461**
random_12_128_174	118.506	118.617	140.984	**113.908**	114.005
random_8_20_160	179.535	180.774	227.956	**152.290**	152.730
random_16_64_179	1 309.003	1 313.670	1 710.538	1 261.136	**1 255.095**
random_24_128_277	1 637.366	1 609.675	2 008.565	1 606.097	**1 605.419**

Table 5.3: Computational results for ebp-tree containers

implementation which they require. With growing dimensionality, vp-trees gained some momentum at least in the narrowing phase, but could hardly be viewed as a general alternative. Verification queries turned out to be a particular weakness of vp-trees with some of the computations being interrupted after reaching the preset timeout limit. Nevertheless, it can be demonstrated that vp-trees have a domain of superiority too. The product of a hypercube and a random 0/1 polytope, rnd_30_cube_5 and the truncated polytope trunc_50 are two particular instances in that respect.

In many aspects, the performance of pop-trees can be viewed similar to that of vp-trees. They turned out to be practically unsuitable for verification queries and showed an increasingly better performance with growing degeneracy for narrowing queries. For problems featuring high degeneracy, their performance became comparable with that of bp-trees. It should be noted that population trees are more general data structures than bit pattern trees and thus have broader parameterization capabilities. In this respect, the execution times may be improved by selecting a more suitable tuple of fragments f. The results presented here were obtained with fragments having a fixed size of 4 bits. The bits at each fragment were selected by using the query bits neutralization technique; that is, the first fragment had the four bits with greatest hit probability, the second one the next four, etc.

	bp-trees	pop-trees		vp-trees	
		narrowing	verification	narrowing	verification
random_4_16_64	**157.106**	160.704	278.316	468.938	639.791
random_4_16_96	**27.312**	28.057	41.825	69.935	48.280
random_6_64_96	**100.676**	205.962	291.643	248.142	240.326
random_8_128_157	**82.033**	102.914	210.508	95.232	174.988
random_6_16_104	**603.381**	643.892	🕒	1 253.839	2 346.667
random_4_64_112	**242.210**	324.381	842.792	504.753	1 228.793
random_4_128_172	**325.128**	470.149	1 541.379	704.531	🕒
random_4_16_112	**90.538**	95.138	377.493	137.877	264.268
random_16_68_115	**152.944**	165.118	217.402	188.090	205.719
random_12_128_174	**118.596**	122.193	219.954	148.295	766.633
random_8_20_160	**179.535**	186.350	1 650.890	202.719	1 256.544
random_16_64_179	**1 309.003**	1 337.564	🕒	1 480.072	🕒
random_24_128_277	**1 637.366**	1 642.699	🕒	1 728.037	2 921.498
rnd_30_cube_5	129.870	-	204.124	-	**33.817**
trunc_50	299.860	-	232.326	-	**216.486**

Table 5.4: Computational results for pop/vp-tree containers

6 Algebraic Test: Redundancy Elimination

The algebraic test, once considered a viable alternative to the combinatorial one, has consistently lost significance due to its performance limitations. This tendency, however, can be reversed by minimizing the number of redundant operations emerging from the independent conduct of each test. Consider the extreme ray r' of the polyhedral cone $\mathcal{C}(A)$. In the narrowing phase, all adjacency candidates

$$(r', r''_1), (r', r''_2), \ldots, (r', r''_k)$$

in which r' participates are collected. Subsequently, an algebraic test is performed for each of them. At each test, a different submatrix of A is being examined in terms of rank calculation. It is easy to see, however, that those matrices are possibly similar to each other and have a lot of common rows. Consequently, by performing each test independently, equivalent transformations are performed multiple times.

A natural solution of the above problem is to cache those intermediate results which may be of interest for other adjacency tests (see Section 6.1). Formally, this requires the definition of data structures capable of holding additional structural information for each extreme ray (see Section 6.2). When performing an adjacency test, the available information related to both extreme rays is combined to reduce the complexity of the problem as much as possible (see Section 6.3). After integrating the described technique into the dd method (see Section 6.4), a considerable performance gain could be reported for problems featuring small to moderate degeneracy (see Section 6.5).

6.1 Active Set Partitioning

Let $\mathcal{C}$ be a polyhedral cone with a representation matrix $A \in \mathbb{R}^{n \times d}$. Suppose we knew all combinations of matrix row vectors which are linear independent. Then we could conduct the adjacency test by checking whether the intersection of the active sets is a superset of some known $d-2$ independent row vectors. The classification of different subsets of matrix column vectors is a well known topic in mathematics and is associated with the term *matroid* [152] (see Section 6.1.1). By adapting its initial definition to consider row vectors instead of column ones, a theoretical basis emerges to derive a new adjacency test (see Section 6.1.2). Its general idea rests on the identification of an independent set within each active set; that is, among the facets defining an extreme ray, the redundant ones are explicitly marked (see Section 6.1.3). Consequently, when performing an adjacency test for two rays r' and r'', particular information about linear dependence of A's row vectors is available prior to the test. In some cases, this is already sufficient to deduce the dimension of $r' \vee r''$. Otherwise, complementary computations are to be performed, but those are less complex than an independent algebraic test (see Section 6.1.4).

6.1.1 Matrix Matroid

Given a $n \times d$ matrix A, let

$$\mathcal{I} = \{ I \subseteq E : rank[A_I] = |I| \}$$

be a collection of subsets of $E = \mathbb{N}_{\leq n}$ labeling only linear independent rows. The ordered pair $(E, \mathcal{I})$ is called a *matrix matroid* of A and is denoted as $M[A]$. Following Oxley [120], two fundamental rules hold for E and $\mathcal{I}$:

(a) $I \in \mathcal{I} \wedge I' \subset I \Rightarrow I' \in \mathcal{I}$ and

(b) $I, I' \in \mathcal{I} \wedge |I| > |I'| \Rightarrow \exists e \in I - I' \, (I' \cup \{e\} \in \mathcal{I})$.

Each subset $K \subseteq E$ has a rank defined by the rank of A_K. Each element of $\mathcal{I}$ is called an *independent set* of $M[A]$. An independent set I is *maximal* if there is no other independent set I' such that $I \subset I'$. A subset which is not a member of $\mathcal{I}$ is called *dependent*. A dependent set D is *minimal* if each proper subset of D is independent. A minimal dependent set is also called a *circuit* of $M[A]$. The set containing all circuits of $M[A]$ is denoted as $C[A]$. It is obvious that $C[A]$ can be constructed out of $\mathcal{I}$ and vice versa, which makes those two representations fully interchangeable. Proposition 6.1 describes one of the important relations between independent sets and circuits. Another meaningful property of circuits is given by the *strong circuit elimination axiom* (see Proposition 6.2).

Proposition 6.1. *Let I be an independent set of $M[A]$ and $I + e$ a dependent one for some $e \in E$. Then there exists a unique circuit $C \subseteq I + e$ which contains e.*

Proof. See [120, Proposition 1.1.6]. □

Proposition 6.2. *Let C_1 and C_2 be two distinct members of $C[A]$ such that $e \in C_1 \cap C_2$ and $f \in C_1 - C_2$. Then there exists another circuit $C_3 \in C[A]$ such that*

$$C_3 \subseteq (C_1 \cup C_2) - e \text{ and } f \in C_3 \ .$$

Proof. See [120, Proposition 1.4.11]. □

6.1.2 Observations

Let r be an extreme ray of $\mathcal{C}(A)$ with $A \in \mathbb{R}^{n \times d}$ and $M[A] = (E, \mathcal{I})$. The active set $z(r)$ has an alternative representation as a pair of disjoint subsets $z_p(r) \in \mathcal{I}$ and $z_s(r) \in \wp(E)$ defined as follows:

$z_p(r)$ is an independent set of $M[A]$ containing exactly $d - 1$ elements, and

$z_s(r)$ is a member of the power set of E which contains all other elements from $z(r)$.

For the moment, assume that for each cone generated by $\mathcal{R}$, there exists a partitioning function

$$\textbf{split} : \mathcal{R} \rightarrow \mathcal{I} \times \wp\left(E\right)$$

which produces a valid pair of subsets $\left(z_p\left(r\right), z_s\left(r\right)\right)$ according to the given definition.

Consider now a pair of extreme rays $\left(r', r''\right)$ for which an adjacency test should be performed. The standard algebraic test involves extracting the submatrix A_D with $D = z\left(r'\right) \cap z\left(r''\right)$ and then calculating $rank\left[A_D\right]$. Here, an alternative route to check the rank of A_D is proposed. Applying **split** on both rays produces two pairs of subsets

$$\left(z_p\left(r'\right), z_s\left(r'\right)\right) \text{ and } \left(z_p\left(r''\right), z_s\left(r''\right)\right)$$

which together yield an alternative representation of the set D as

$$\left(z_p\left(r'\right) \cap z_p\left(r''\right)\right) \cup \left(z_p\left(r'\right) \cap z_s\left(r''\right)\right) \cup \left(z_s\left(r'\right) \cap z_p\left(r''\right)\right) \cup \left(z_s\left(r'\right) \cap z_s\left(r''\right)\right) \ .$$

The point behind this new representation is given in the following Observations 6.1 and 6.2.

Observation 6.1. *The set*

$$\mathcal{L} = \left(z_p\left(r'\right) \cap z_p\left(r''\right)\right) \cup \left(z_p\left(r'\right) \cap z_s\left(r''\right)\right) \cup \left(z_s\left(r'\right) \cap z_p\left(r''\right)\right)$$

is an independent one (with respect to $M\left[A\right]$) if the following two constraints are satisfied:

1. $\forall i \in \left(z_s\left(r'\right) \cap z_p\left(r''\right)\right) \left(\left(C'_i \in C\left[A\right] \wedge C'_i \subseteq z_p\left(r'\right) + i\right) \Rightarrow C'_i \subseteq \mathbb{N}_{\leq i}\right)$ *and*
2. $\forall i \in \left(z_s\left(r''\right) \cap z_p\left(r'\right)\right) \left(\left(C''_i \in C\left[A\right] \wedge C''_i \subseteq z_p\left(r''\right) + i\right) \Rightarrow C''_i \subseteq \mathbb{N}_{\leq i}\right)$.

Corollary 6.3. *If the active sets of r' and r'' are partitioned according to Observation* 6.1 *and $\left|\mathcal{L}\right| = d - 2$, then r' and r'' are adjacent.*

Observation 6.2. *For each two adjacent extreme rays r' and r'', there always exists an active set partitioning that satisfies Corollary* 6.3.

Consequently, an adjacency test can be performed by finding a partitioning which satisfies Corollary 6.3 or proving that it does not exist. According to Observation 6.2, the test is complete in the sense that a conclusive result is guaranteed for each pair of extreme rays. The idea is illustrated in the following Examples 6.1 and 6.2.

Example 6.1. Consider the matrix A' as shown in Figure 6.1 where

$$x_1 = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \text{ and } x_2 = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

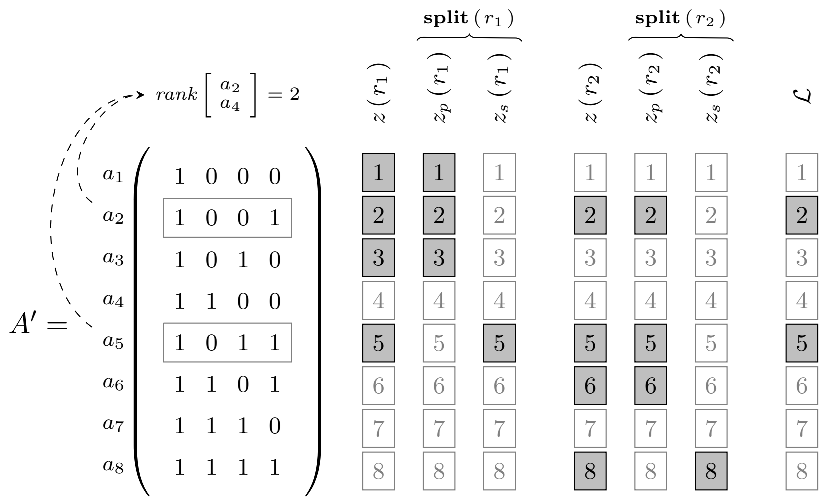


Figure 6.1: Adjacency proof by means of active set partitioning

are generators of the polyhedral cone $\mathcal{C}(A')$. Let x_1 and x_2 define the extreme rays r_1 and r_2 with active sets

$$z(r_1) = \{1,2,3,5\} \text{ and } z(r_2) = \{2,5,6,8\} \quad .$$

Suppose that applying **split** on both extreme rays delivers

$$\textbf{split}(r_1) = (\{1,2,3\},\{5\}) \text{ and } \textbf{split}(r_2) = (\{2,5,6\},\{8\}) .$$

The adjacency of r_1 and r_2 is easily proven by applying the algebraic test. Alternatively, the same result is obtained by building the set $\mathcal{L}$ according to Corollary 6.3.

Example 6.2. Consider now another matrix A'' (see Figure 6.2) for which the generators

$$x_1 = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \text{ and } x_2 = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

of the polyhedral cone $\mathcal{C}(A'')$ are known. Let r_1 and r_2 be the corresponding extreme rays and **isplit** a particular implementation of **split** delivering the output

$$\textbf{isplit}(r_1) = (\{2,3,4,7\},\{6\}) \text{ and} \tag{6.1}$$

$$\textbf{isplit}(r_2) = (\{4,5,6,8\},\{3\}) \ . \tag{6.2}$$

The given partitioning yields $|\mathcal{L}| = d - 2$, but does not match the constraints from Observation 6.1. It should be noticed that both active sets, $z(r_1)$ and $z(r_2)$, contain the circuit $C = \{3,4,6\}$. C, however, is partitioned differently in (6.1) and (6.2). This leads to the problem that $\mathcal{L}$ cannot be independent as C becomes a subset of it.

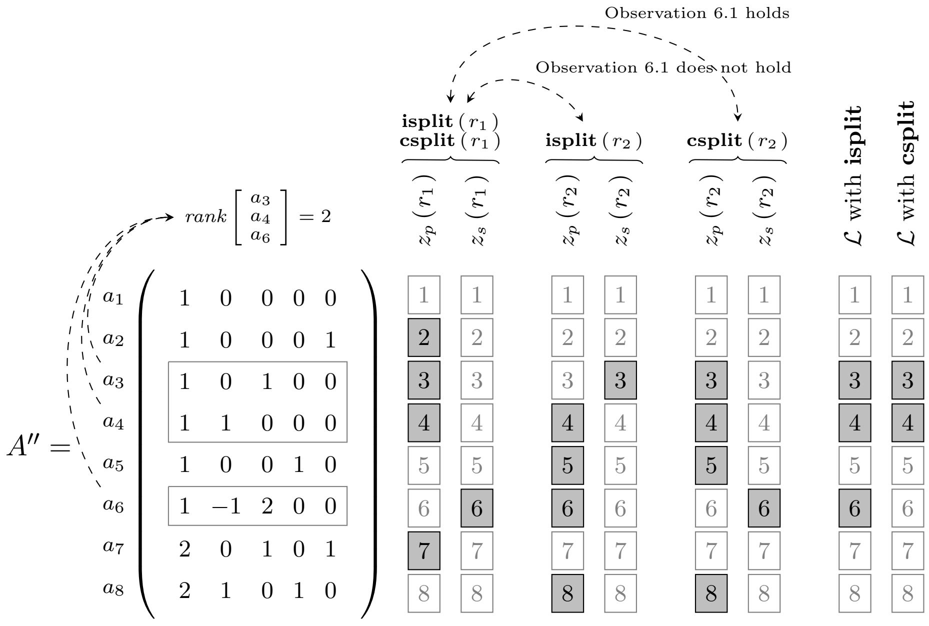


Figure 6.2: Active set partitioning for nonadjacent rays

The reasoning behind the constraints in Observation 6.1 becomes evident now. They guarantee that $C \nsubseteq \mathcal{L}$.

Suppose that we could apply another partitioning function **csplit**, which is again an implementation of **split**, but behaves differently from **isplit**. Let **csplit** deliver the same result for r_1 as in (6.1), but a different one for r_2:

$$\textbf{csplit}\,(r_2) = (\{3,4,5,8\},\{6\}) \quad . \tag{6.3}$$

Although Observation 6.1 holds with respect to (6.1) and (6.3), the result remains inconclusive as now $|\mathcal{L}| < d-2$. This problem is resolved by Observation 6.2. It is easy to verify that another partitioning satisfying the constraints from Observation 6.1 is impossible; thus, r_1 and r_2 cannot be adjacent.

6.1.3 Partitioning Function

It is evident that the application of Observation 6.1 requires a great amount of effort as it demands the calculation of certain circuits within the active sets. This calculation effort is likely to quickly even out the benefits of the proposed adjacency proof method. With regard to that, a simple implementation of the partitioning function, called **nsplit**, is proposed in Procedure 10. It possesses the important property of partitioning the active sets in way that they fulfill the constraints in Observation 6.1 by default. At first, this seems to be just a calculation shift as Procedure 10 has to be invoked on each new ray. This invocation, however, is not always necessary. Consider the iteration step of the dd method (see Section 4.1). Assume that for two adjacent extreme rays r', r'' emerging

Procedure 10 nsplit: active set partitioning

```
Input: $r$, an extreme ray of $\mathcal{C}(A)$ with $A \in \mathbb{R}^{n \times d}$
Output: $(z_p(r), z_s(r)) \in \mathcal{I} \times \wp(E)$ such that
  $z(r) = z_p(r) \cup z_s(r)$ and $z_p(r) \cap z_s(r) = \emptyset$ for $M[A] = (E, \mathcal{I})$
  if $|z(r)| = d - 1$ then
    $z_p(r) \leftarrow z(r)$; $z_s(r) \leftarrow \emptyset$
  else
    $z_p(r) \leftarrow \emptyset$; $z_s(r) \leftarrow \emptyset$
    for all $l \in z(r)$ do
      if $|z_p(r)| = d - 1$ or $rank\left[A_{z_p(r)}\right] = rank\left[A_{z_p(r)+l}\right]$ then
        $z_s(r) \leftarrow z_s(r) + l$
      else
        $z_p(r) \leftarrow z_p(r) + l$
      end if
    end for
  end if
  return $(z_p(r), z_s(r))$
```

from the generators $x' \in X_i^+$ and $x'' \in X_i^-$, $|\mathcal{L}| = d - 2$ holds with respect to the partitioning

$$\mathbf{nsplit}\left(r'\right) = \left(z_p\left(r'\right), z_s\left(r'\right)\right) \text{ and } \mathbf{nsplit}\left(r''\right) = \left(z_p\left(r''\right), z_s\left(r''\right)\right) \quad .$$

After building the new generator $x \in X_i^{\triangledown}$, the corresponding extreme ray r gains a valid active set partitioning

$$\left(\mathcal{L} + j, z_s\left(r'\right) \cap z_s\left(r''\right)\right) \quad .$$

Most important, the above result always matches the result of Procedure 10 and thus makes its explicit invocation unnecessary.

6.1.4 Completeness

So far, it has been demonstrated how to partition active sets and that a favorable partitioning can immediately indicate adjacency. It is, however, unclear how to proceed when the partitioning delivers an initially inconclusive result as a result of $|\mathcal{L}| < d - 2$. Those cases, of course, could be covered by the classic rank computation, but as the general focus is laid on its complete elimination from the dd method, a supplemental strategy is introduced to provide completeness. The idea is best illustrated by the following Example 6.3.

Example 6.3. Consider the matrix A''' (see Figure 6.3) and the generators

$$x_1 = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \text{ and } x_2 = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

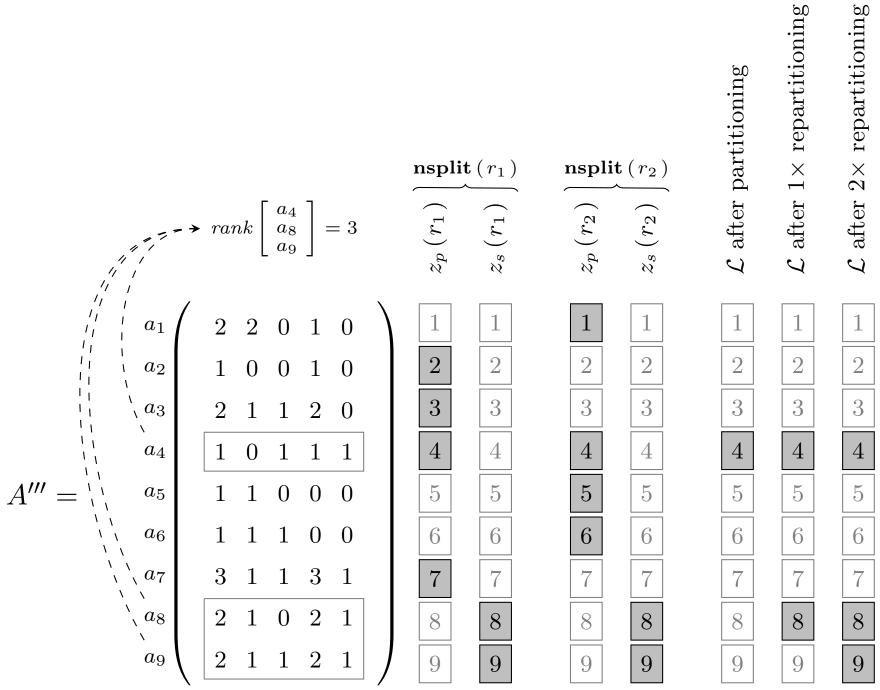


Figure 6.3: Repartitioning of active sets

which define the adjacent extreme rays r_1 and r_2 of the polyhedral cone $\mathcal{C}\,(A''')$. The partitioned active sets

$$\mathbf{nsplit}\,(r_1) = (\{2,3,4,7\},\{8,9\}) \text{ and} \tag{6.4}$$

$$\mathbf{nsplit}\,(r_2) = (\{1,4,5,6\},\{8,9\}) \tag{6.5}$$

are insufficient for an adjacency proof as the cardinality of $\mathcal{L}$ is 1 where 3 is needed. Suppose that the equations

$$a_8 = -a_2 - a_3 - a_4 + 2a_7 \text{ and} \tag{6.6}$$

$$a_9 = -a_2 + a_7 \tag{6.7}$$

have been additionally attached to the active set of r_1, giving particular information of how exactly each element in $z_s\,(r_1)$ can be represented as a linear combination of elements from $z_p\,(r_1)$. As a consequence, we can repartition (6.4) by moving the elements 8 and 9 from $z_s\,(r_1)$ into $z_p\,(r_1)$ so that $|\mathcal{L}| = 3$ is reached in the end. This process consists of two transformations

$$(\{2,3,4,7\},\{8,9\}) \rightarrow (\{x,x,x,8\},\{x,9\}) \rightarrow (\{x,x,8,9\},\{x,x\})$$

swapping each of the elements 8 and 9 with some element from $z_p\,(r_1)$.

First, the element 8 is processed. From (6.6), we already know that $C_8 = \{2, 3, 4, 7, 8\}$ is a circuit. We can therefore deduce that

$$(\{3, 4, 7, 8\}, \{2, 9\}) \tag{6.8}$$

represents a legitimate partitioning as well. In addition, (6.5) and (6.8) comply with Observation 6.1.

Second, the element 9 is swapped. Before executing this transition, however, one last problem has to be addressed. In contrast to the first step, here we have no particular information about the circuit $C_9 \subseteq \{3, 4, 7, 8, 9\}$. C_9 is obviously needed in order to select an appropriate element to swap 9 with. Selecting an element outside of it would result in an invalid partitioning. The computation of C_9 is achieved by building the sum of the equations (6.6) and (6.7) which leads to

$$a_9 = a_3 + a_4 - a_7 + a_8$$

and hence $C_9 = \{3, 4, 7, 8, 9\}$. As a result, we can now derive that

$$(\{4, 7, 8, 9\}, \{2, 3\}) \tag{6.9}$$

represents a valid partitioning of $z(r_1)$.

Consequently, the active set partitioning given in (6.5) and (6.9) delivers an adjacency proof according to Corollary 6.3.

Based on Example 6.3, a general strategy for performing the adjacency test can be outlined. To begin with, assume adjacency for the extreme rays under test. Then, try to prove it by applying repartitioning transformations which increase the size of $\mathcal{L}$. At the same time, keep the repartitioned active sets consistent with the constraints from Observation 6.1. At some point, a maximum is reached where no further transformation leads to a size increase of $\mathcal{L}$. If Corollary 6.3 holds at this point, then the extreme rays are adjacent. Otherwise, nonadjacency is the case.

6.2 Data Structures

Definition 6.1 [Active Set Pair]. Let r be an extreme ray of the polyhedral cone $\mathcal{C}(A)$. The ordered pair (z_p, z_s) is called an *active set pair* of r if and only if

$z_p \subseteq z(r)$ is an independent set of $M[A]$ with $|z_p| = d - 1$, and

$z_s = z(r) - z_p$ contains elements which build a dependent set with z_p.

The set containing all valid active set pairs of $r \in \mathcal{R}$ is obtainable by means of a mapping

$$\phi : \mathcal{R} \to \{\omega \mid \omega = (z_p, z_s) \text{ is an active set pair of } r\} \quad .$$

Definition 6.2 [Dependence Map]**.** Let r be an extreme ray of $\mathcal{C}(A)$ with

$$\omega = (z_p, z_s) \in \phi(r)$$

some active set pair of r. The set

$$\begin{aligned}\chi(\omega) = \{(l, \mathcal{M}, \rho) \mid\ & l \in z_s,\\ & \mathcal{M} \subseteq z_p \text{ and } \mathcal{M} + l \in C[A],\\ & \rho \in \mathbb{R}^n - \{0\} : \sum_{i=1}^{n} \rho_i a_i = 0 \text{ where } \rho_i = 0 \Leftrightarrow i \notin \mathcal{M} + l\}\end{aligned}$$

is called a *dependence map* of ω.

To each active set pair a dependence map is assigned to keep track of those independent sets of $M[A]$ which are both subsets of z_p and build a circuit with one of z_s's elements. Consider the matrix A given in Figure 6.4. Let $\omega = (\{2,3,4\}, \{5\})$ be a valid active set pair. The dependence map of ω stores the necessary information of how to express a_5 as a linear combination of a_2, a_3 and a_4. $\mathcal{M}_5$ indexes those rows which together with a_5 build a circuit of $M[A]$. Furthermore, a vector ρ is defined such that

$$\sum_{i=1}^{6} \rho_i a_i = 0 \wedge (\rho_i = 0 \Leftrightarrow i \notin \mathcal{M}_5 + 5) \ .$$

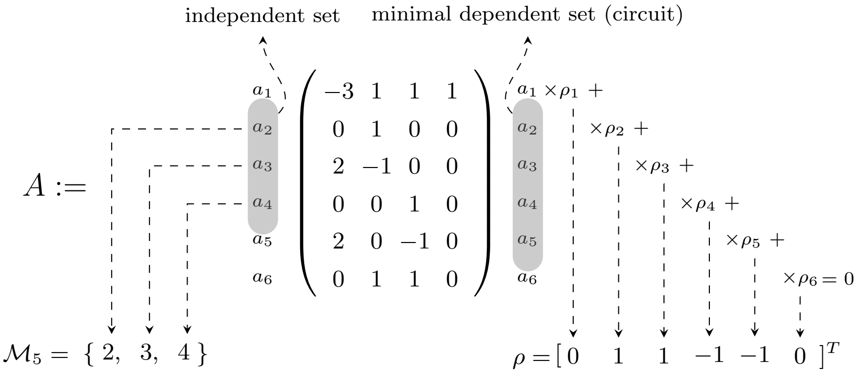

independent set
minimal dependent set (circuit)
$A :=$
a_1
a_2
a_3
a_4
a_5
a_6
$\begin{pmatrix} -3 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 2 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$
$a_1 \times\rho_1 +$
$a_2 \times\rho_2 +$
$a_3 \times\rho_3 +$
$a_4 \times\rho_4 +$
$a_5 \times\rho_5 +$
$a_6 \times\rho_6 = 0$
$\mathcal{M}_5 = \{\,2,\ 3,\ 4\,\}$
$\rho = [\,0 \quad 1 \quad 1 \quad -1 \quad -1 \quad 0\ ]^T$

Figure 6.4: Construction of dependence maps

It is clear that $\mathcal{M}$ and ρ are redundant structures in the dependence map as $\mathcal{M}$ could be extracted out of ρ at any time. The reasoning behind this redundancy is the suitability of $\mathcal{M}$ to be represented by a binary vector which allows the quick execution of set operations. The dependence map can be implemented as a hash table by using l as a key and the pair $(\mathcal{M}, \rho)$ as a value. As the following Corollary 6.4 states, each l is then associated with exactly one pair.

Corollary 6.4. *Let r be an extreme ray of $\mathcal{C}(A)$ with $(z_p, z_s) \in \phi(r)$ some active set pair of r. For each $l \in z_s$, there exists exactly one circuit $\mathcal{M} + l \in C[A]$ such that $\mathcal{M} \subseteq z_p$.*

Proof. Follows directly from Proposition 6.1 as z_p is an independent set and $z_p + l$ a dependent one. □

It is obvious that the construction of a single dependence map $\chi(\omega)$ requires $|z_s|$ systems of linear equations to be solved. At first, this seems to be highly expensive. It should be pointed out, however, that the dependence maps of many extreme rays are not disjoint and thus have common entries. Actually, each dependence map can be seen as a subset of $C[A]$; thus, we can define a global set $\mathcal{X}$ holding all known entries of $C[A]$. Each dependence map $\chi(\omega)$ contains links to those which are of any significance for ω. Consequently, building a new dependence map involves checking whether the corresponding entries are already in the global set $\mathcal{X}$. If so, they are merely linked to the active set. Otherwise, they are calculated, put into $\mathcal{X}$ and finally linked.

Consider the example given in Figure 6.5 where ω' and ω'' are two valid active set pairs to build the dependence maps for. Starting with ω', the tuples $(l_1, \mathcal{M}_5, \rho')$ and $(l_2, \mathcal{M}_6, \rho'')$ are calculated, put in the global set $\mathcal{X}$ and finally linked with ω'. As a consequence, the construction of $\chi(\omega'')$ requires merely linking $(l_2, \mathcal{M}_6, \rho'')$ as the necessary calculation has been already performed in the previous step.

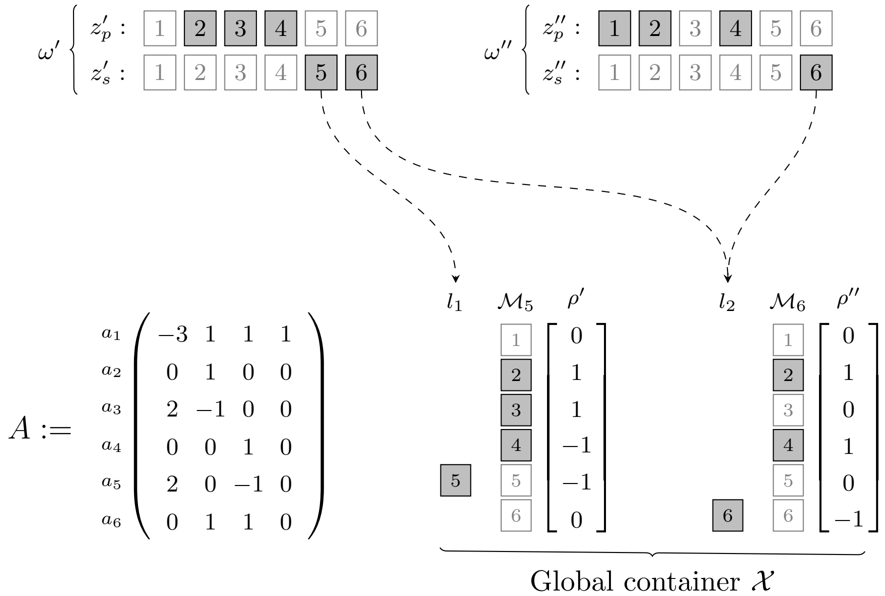


Figure 6.5: Redundancy-free storage of dependence maps

Definition 6.3 [Weak Combinability]. Two active set pairs

$$\omega' = (z'_p, z'_s) \text{ and } \omega'' = (z''_p, z''_s)$$

are weakly combinable if

$$\begin{aligned}\forall\left(l,\mathcal{M}_l,\rho\right)\in\chi\left(\omega'\right)\left(l\in z''_p\Rightarrow\mathcal{M}_l\subset\mathbb{N}_{\leq l}\right)\ \text{and}\\ \forall\left(m,\mathcal{M}_m,\rho\right)\in\chi\left(\omega''\right)\left(m\in z'_p\Rightarrow\mathcal{M}_m\subset\mathbb{N}_{\leq m}\right)\ .\end{aligned}$$

Definition 6.4 [Strong Combinability]**.** Two weakly combinable active set pairs

$$\omega'=\left(z'_p,z'_s\right)\ \text{and}\ \omega''=\left(z''_p,z''_s\right)$$

are also *strongly combinable* if

$$\begin{aligned}\forall\left(l,\mathcal{M}_l,\rho\right)\in\chi\left(\omega'\right)\left(l\in z''_s\Rightarrow\mathcal{M}_l\subset\mathbb{N}_{\leq l}\right)\ \text{and}\\ \forall\left(m,\mathcal{M}_m,\rho\right)\in\chi\left(\omega''\right)\left(m\in z'_s\Rightarrow\mathcal{M}_m\subset\mathbb{N}_{\leq m}\right)\ .\end{aligned}$$

Definition 6.5 [Normal Form]**.** The active set pair $\omega=\left(z_p,z_s\right)$ is in *normal form* (or normalized) if

$$\forall\left(l,\mathcal{M}_l,\rho\right)\in\chi\left(\omega\right)\left(\mathcal{M}_l\subset\mathbb{N}_{\leq l}\right)\ .$$

It is easy to verify that Procedure 10 creates normalized active set pairs and that two normalized active set pairs are always strongly combinable.

6.3 The Refined Algebraic Test

With the data structures introduced in the previous section, the necessary basis has been laid out to formalize the adjacency test from Section 6.1. In order to resolve any doubt concerning its correctness, a proof of Observations 6.1 and 6.2 is provided (see Section 6.3.1) before specifying the actual adjacency algorithm called *partitioned sets test* or shortly PST (see Section 6.3.2).

6.3.1 Correctness and Formal Proof

First, a formal proof is provided that Corollary 6.3 can be used as a sufficient (see Proposition 6.5) and a necessary condition for adjacency (see Lemma 6.7). In that context, a sufficient condition for nonadjacency is specified as well (see Proposition 6.8). Finally, the necessary rules needed for repartitioning of active set pairs are provided (see Lemma 6.9 and 6.10).

Proposition 6.5 (PST Correctness)**.** *Let r' and r'' be two distinct extreme rays of $\mathcal{C}\left(A\right)$, with active set pairs*

$$\omega'=\left(z'_p,z'_s\right)\ \textit{and}\ \omega''=\left(z''_p,z''_s\right)\ .$$

If ω' and ω'' are weakly combinable and $|\mathcal{L}| = d-2$ holds for

$$\mathcal{L} = \left(z'_p \cap z''_p\right) \cup \left(z'_p \cap z''_s\right) \cup \left(z''_p \cap z'_s\right) \ ,$$

then r' and r'' are adjacent.

Proof. Assume that $\mathcal{L}$ is a dependent set. Then for some $m \in \mathcal{L}$, there exists a circuit $D \subseteq \mathcal{L} \cap \mathbb{N}_{\leq m}$. From the definition of $\mathcal{L}$, it is evident that either $m \in z'_p$ or $m \in z''_p$. If $m \in z'_p$, then an application of Lemma 6.6 with $r = r'$ and $K = z'_s \cap z''_p$ guarantees that

$$D \nsubseteq (z'_p \cup (z'_s \cap z''_p)) \cap \mathbb{N}_{\leq m} \supseteq \mathcal{L} \cap \mathbb{N}_{\leq m} \ .$$

Analogously, if $m \in z''_p$, we can prove that

$$D \nsubseteq (z''_p \cup (z''_s \cap z'_p)) \cap \mathbb{N}_{\leq m} \supseteq \mathcal{L} \cap \mathbb{N}_{\leq m}$$

by applying Lemma 6.6 with $r = r''$ and $K = z''_s \cap z'_p$. Consequently, we have shown the nonexistence of the circuit D and hence the independence of $\mathcal{L}$. Therefore,

$$|\mathcal{L}| = d-2 \Rightarrow rank\,(\mathcal{L}) = d-2 \Rightarrow rank\,[A_{\mathcal{L}}] = d-2 \ .$$

□

Lemma 6.6. *Let r be an extreme ray of $\mathcal{C}\,(A)$ where $\omega = (z_p, z_s) \in \phi\,(r)$ is a valid active set pair of r and $K \subseteq z_s$. If*

$$\forall\,(l, \mathcal{M}_l, \rho) \in \chi\,(\omega)\,\left(l \in K \Rightarrow \mathcal{M}_l \subset \mathbb{N}_{\leq l}\right) \tag{6.10}$$

then for each $m \in z_p$ there is no circuit

$$D \subseteq z_p \cup K_m \qquad with\ K_m = K \cap \mathbb{N}_{\leq m}$$

which contains m.

Proof. If $K = \emptyset$ then the nonexistence of D is obvious. Let $K \neq \emptyset$ and let for some $m \in z_p$,

$$D \subseteq z_p \cup K_m \tag{6.11}$$

be a circuit which contains m. From the dependence map $\chi\,(\omega)$ we can obtain a unique circuit

$$D_l = \mathcal{M}_l + l \subseteq z_p + l \tag{6.12}$$

for each $l \in K_m$. According to (6.10), $m \notin D_l$ for each such circuit D_l as m is greater than any l. By applying Proposition 6.2 to D and D_l with maximal l, we can produce another circuit

$$D' \subseteq (D \cup D_l) - l$$

which still contains m, but avoids l. The iterative application of that last step on all other D_l leads finally to a circuit

$$D'' \subseteq (D \cup \bigcup_{l \in K_m} D_l) - K_m$$

which certainly contains m, but avoids the whole set K_m. Together with (6.11) and (6.12), the latter actually implies that

$$\begin{aligned} D'' &\subseteq \left(z_p \cup K_m \cup \bigcup_{l \in K_m} (z_p + l) \right) - K_m \\ &\subseteq (z_p \cup K_m \cup z_p \cup K_m) - K_m \\ &\subseteq z_p \end{aligned}$$

which is obviously a contradiction as z_p is an independent set by definition. Consequently, the nonexistence of the D is proven. □

Lemma 6.7 (PST Completeness). *Let r' and r'' be two distinct extreme rays of $\mathcal{C}(A)$. If r' and r'' are adjacent, then there exist at least two weakly combinable active set pairs*

$$\omega' = \left(z'_p, z'_s \right) \text{ and } \omega'' = \left(z''_p, z''_s \right)$$

such that $|\mathcal{L}| = d - 2$ holds for

$$\mathcal{L} = \left(z'_p \cap z''_p \right) \cup \left(z'_p \cap z''_s \right) \cup \left(z''_p \cap z'_s \right) \ .$$

Proof. The adjacency of r' and r'' implies that

$$rank \left(z\left(r'\right) \cap z\left(r''\right) \right) = d - 2$$

and thus the existence of an independent set

$$I \subseteq z\left(r'\right) \cap z\left(r''\right)$$

containing exactly $d - 2$ elements. Furthermore,

$$rank \left(z\left(r'\right) \right) = rank \left(z\left(r''\right) \right) = d - 1$$

which guarantees the existence of another two distinct independent sets

$$I' \subseteq z\left(r'\right) \text{ and } I'' \subseteq z\left(r''\right)$$

which are obtained by maximizing I; hence, $|I'| = |I''| = d - 1$ and $I = I' \cap I''$. As a consequence, we can build the active set pairs

$$\omega' = \left(I', z\left(r'\right) - I' \right) \text{ and } \omega'' = \left(I'', z\left(r''\right) - I'' \right)$$

with

$$\mathcal{L} = I = I' \cap I'' \text{ and } I' \cap \left(z\left(r''\right) - I'' \right) = I'' \cap \left(z\left(r'\right) - I' \right) = \emptyset \ .$$

Due to $z'_p \cap z''_s = z''_p \cap z'_s = \emptyset$, the weak combinability of ω' and ω'' is guaranteed as well. □

Proposition 6.8 (PST Nonadjacency Condition)**.** *Let r' and r'' be two extreme rays of $\mathcal{C}\left(A\right)$ with*

$$\omega' = \left(z'_p, z'_s\right) \text{ and } \omega'' = \left(z''_p, z''_s\right)$$

two weakly combinable active set pairs of r' and r''. If for each $l \in \left(z_s\left(r'\right) \cap z_s\left(r''\right)\right)$

$$\exists \left(l, \mathcal{M}_l, \rho\right) \in \left\{\chi\left(\omega'\right) \cup \chi\left(\omega''\right)\right\}\left(\mathcal{M}_l \subseteq \mathcal{L}\right) \tag{6.13}$$

and $|\mathcal{L}| < d - 2$ holds for

$$\mathcal{L} = \left(z'_p \cap z''_p\right) \cup \left(z'_p \cap z''_s\right) \cup \left(z''_p \cap z'_s\right) \ ,$$

then r' and r'' are nonadjacent.

Proof. Condition (6.13) implies that for each $l \in z'_s \cap z''_s$ there exists a circuit $D_l \subseteq \mathcal{L} + l$ which contains l and thus

$$\begin{aligned} &rank(\mathcal{L} \cup (z'_s \cap z''_s)) = rank\left(\mathcal{L}\right) \\ \Rightarrow &rank\left(z\left(r'\right) \cap z\left(r''\right)\right) = |\mathcal{L}| \\ \Rightarrow &rank\left(z\left(r'\right) \cap z\left(r''\right)\right) < d - 2 \end{aligned}$$

showing the nonadjacency of r' and r''. □

Lemma 6.9. *Let r be an extreme ray with an active set pair $\omega = \left(z_p, z_s\right)$ and $\left(l, \mathcal{M}, \rho\right)$ some element of its dependence map $\chi\left(\omega\right)$. Then for all $e \in \mathcal{M}$*

$$((z_p - e) + l, (z_s - l) + e)$$

is also a valid active set pair of r.

Proof. From the dependence map $\chi\left(\omega\right)$, we know that $\mathcal{M} + l \subseteq z_p + l$ is a unique circuit that definitely contains e. Therefore $(z_p - e) + l$ has to be an independent set as otherwise $z_p + l$ would contain at least one more circuit which contradicts with Proposition 6.1. □

Lemma 6.10. *Let r be an extreme ray of $\mathcal{C}\left(A\right), A \in \mathbb{R}^{n \times d}$ with*

$$\omega = \left(z_p, z_s\right) \text{ and } \omega' = \left(z'_p, z'_s\right)$$

two active set pairs of r such that $\left(l, \mathcal{M}_l, \psi\right) \in \chi\left(\omega\right)$, $e \in \mathcal{M}_l$ and $z'_p = (z_p - e) + l$. Then

$$\forall \left(p, \mathcal{M}_p, \rho\right) \in \chi\left(\omega\right)\left(p \neq l \wedge e \in \mathcal{M}_p \Rightarrow \left(p, \mathcal{M}'_p, \varphi\right) \in \chi\left(\omega'\right)\right)$$

where $\varphi = \psi_e \rho - \rho_e \psi$ and $\mathcal{M}'_p = \{m \mid \varphi_m \neq 0 \wedge m \neq p\}$.

Proof. From Definition 6.2 we obtain

$$\sum_{i=1}^{n} \psi_i a_i = 0 \Rightarrow \sum_{i=1}^{n} \rho_e \psi_i a_i = 0 \tag{6.14}$$

and

$$\sum_{i=1}^{n} \rho_i a_i = 0 \Rightarrow \sum_{i=1}^{n} \psi_e \rho_i a_i = 0. \tag{6.15}$$

Subtracting (6.14) from (6.15) leads to

$$\sum_{i=1}^{n} (\psi_e \rho_i - \rho_e \psi_i) a_i = 0 \ .$$

Let $\varphi = \psi_e \rho - \rho_e \psi$. Then

1. $\varphi_e = \psi_e \rho_e - \rho_e \psi_e = 0$,
2. $\varphi_l = \psi_e \rho_l - \rho_e \psi_l = \psi_e \rho_l \neq 0$,
3. $\varphi_p = \psi_e \rho_p - \rho_e \psi_p = -\rho_e \psi_p \neq 0$ and
4. $\varphi_i = 0$ for each $i \notin \mathcal{M}_l + l + \mathcal{M}_p + p$ (from Definition 6.2 for ρ and ψ).

Consequently, there exists an independent set

$$\mathcal{M}'_p := \{ m \mid \varphi_m \neq 0 \wedge m \neq p \} \subseteq \mathcal{M}_l + l + \mathcal{M}_p - e \subseteq z'_s$$

such that $(p, \mathcal{M}'_p, \varphi)$ is a valid element of $\chi(\omega)$. □

6.3.2 Adjacency Algorithm

Let $\omega' = (z'_p, z'_s)$ and $\omega'' = (z''_p, z''_s)$ be two weakly combinable active set pairs. In the context of the partitioned sets test, a conclusive result regarding the adjacency of r' and r'' requires the fulfillment of either Proposition 6.5 or Proposition 6.8. An initially inconclusive result, on the other hand, could be dealt with by repartitioning ω' or ω'' (or both) until one of the given propositions becomes applicable.

Repartitioning Strategy. Assume that ω' and ω'' deliver an inconclusive result. With respect to Proposition 6.8, at least one of the following three conditions is then unsatisfied:

(i) weak combinability of ω' and ω'',

(ii) $|\mathcal{L}| < d - 2$ or

(iii) fulfillment of (6.13) for all elements in $z'_s \cap z''_s$.

The first one is satisfied by default as weak combinability was assumed. The second one is also satisfied as otherwise Proposition 6.5 would have held. Consequently, the application of Proposition 6.8 fails because the third condition does not hold. Assume that it can be satisfied by repartitioning ω' while keeping the first one satisfied as well. If the second condition still holds afterwards, then so does Proposition 6.8; hence, r'

and r'' are nonadjacent. Otherwise, Proposition 6.5 becomes applicable and implies the adjacency of r' and r''.

The process of satisfying the third condition involves to iteratively traverse all

$$l \in K = z_s' \cap z_s'' \tag{6.16}$$

in ascending order and, if necessary, enforce condition (6.13). The latter is achieved by using the dependence map entry $(l, \mathcal{M}, \rho) \in \chi(\omega')$ to transform ω' into another active set pair

$$\omega_1' = ((z_p' - e) + l, (z_s' - l) + e) \tag{6.17}$$

according to Lemma 6.9. By selecting

$$e \in \mathcal{M} - z\left(r''\right)$$

to be an element outside of ω'', l is virtually moved from the set K into the set $\mathcal{L}$. Moreover, the existence of such e is guaranteed as otherwise condition (6.13) would have held for l. Consequently, K is step by step reduced to a set of elements all of which comply with condition (6.13).

After creating a new active set pair as shown in (6.17), we have to make sure that the corresponding dependence map $\chi(\omega_1')$ is also generated. The following three rules give an overview of how this is accomplished.

I. All elements $(p, \mathcal{M}_p, \rho) \in \chi(\omega')$ with $e \notin \mathcal{M}_p$ are also valid for ω_1'. They can be copied into $\chi(\omega_1')$ without modification.

II. The circuit $\mathcal{M} + l$, coded by the tuple $(l, \mathcal{M}, \psi)$ in $\chi(\omega')$, is also valid for ω_1'. In $\chi(\omega_1')$, it receives the representation $(e, (\mathcal{M} - e) + l, \psi)$.

III. All other elements $(p, \mathcal{M}_p, \rho) \in \chi(\omega')$ with $e \in \mathcal{M}_p$ are irrelevant for ω_1'. For each of them, a new tuple $\left(p, \mathcal{M}_p', \varphi\right)$ with $\mathcal{M}_p' \subseteq (\mathcal{M}_p - e) + l$ has to be built. The exact calculation of $\mathcal{M}_p'$ and φ is covered in Lemma 6.10.

In Figure 6.6, the creation of $\chi(\omega_1')$ is illustrated for the extreme ray r_1 from Example 6.3. In that context,

$\omega_1 = (\{2,3,4,7\}, \{8,9\})$ and $\omega_1' = (\{3,4,7,8\}, \{2,9\})$ are the active set pairs before and after the repartitioning, and

$\chi(\omega_1) = \{(l_1, \mathcal{M}_8, \psi), (l_2, \mathcal{M}_9, \rho)\}$ is the dependence map of ω_1.

Weak vs Strong Combinability. While the described repartitioning procedure achieves the fulfillment of (6.13) for all elements in $z_s' \cap z_s''$, it is unclear how the weak combinability of ω' and ω'' is preserved for ω_1' and ω''. Obviously, this property is not automatically satisfied as for some $(l, \mathcal{M}'', \rho'') \in \chi(\omega'')$,

$$\mathcal{M}'' \subset \mathbb{N}_{\leq l} \tag{6.18}$$

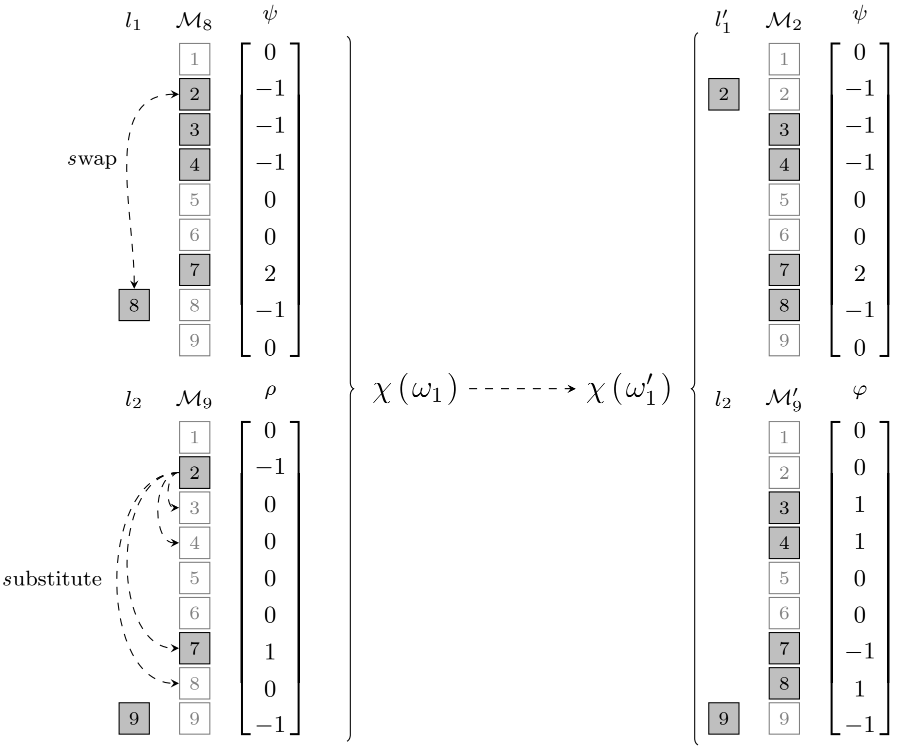


Figure 6.6: Update of dependence maps after repartitioning

may not hold. This obstacle can be easily cleared by requiring strong combinability for ω' and ω''. In this case, (6.18) is satisfied by definition. Still, in order to guarantee the strong combinability of ω_1' and ω'' as well, one last problem has to be solved. Let $p \in z_s' \cap z_p''$ be some element with associated dependence map entry $(p, \mathcal{M}_p, \rho) \in \chi(\omega')$ such that $p < l$ and $e \in \mathcal{M}_p$. After swapping l and e in the transformation step (6.17), we should construct the new dependence map entry $\left(p, \mathcal{M}_p', \varphi\right) \in \chi(\omega_1')$ according to rule III (see previous paragraph). $\mathcal{M}_p' + p$, however, is a circuit containing both l and p which puts $\omega_1' = \left(z_{p1}', z_{s1}'\right)$ and ω'' in a noncombinable form as $l \in z_{p1}' \cap z_s''$ and $p < l$ (see Definition 6.3). This issue is easily resolved by widening the range defined in (6.16) to

$$K' = z_s' \cap (z_p'' \cup z_s'') \ .$$

By doing so, the existence of a $p < l$ with $(p, \mathcal{M}_p, \rho) \in \chi(\omega')$ is effectively ruled out as

$$z_s' \cap z_p'' \cap \mathbb{N}_{\leq l} = \emptyset$$

for each transformation candidate l.

Implementation. In Procedures 11, 12, 13 and 14, one possible implementation of the partitioned sets test is demonstrated. The starting point of the algorithm is Procedure 11. It requires two strongly combinable active set pairs ω' and ω'' of the extreme rays which the adjacency test should be performed for. Its output is one of the boolean values **true**, in case of adjacency, or **false**, otherwise. It is worth mentioning that optimizations have been intendedly omitted in order the preserve the algorithm's compactness and simplicity.

```
Procedure 11 pst: partitioned sets test
Input: $\begin{cases} \omega' = \left(z'_p, z'_s\right) \in \phi\left(r'\right), & \text{an active set pair of } r' \\ \chi\left(\omega'\right), & \text{dependence map of } \omega' \\ \omega'' = \left(z''_p, z''_s\right) \in \phi\left(r''\right), & \text{an active set pair of } r'' \end{cases}$
Output: $\begin{cases} \textbf{true}, & \text{if } r' \text{ and } r'' \text{ are adjacent} \\ \textbf{false}, & \text{otherwise} \end{cases}$
  if nonadjacent $(\omega', \omega'')$ then
    return false
  else
    if adjacent $(\omega', \omega'')$ then
      return true
    else
      $\mathcal{L} \leftarrow (z'_p \cap z''_p) \cup (z'_p \cap z''_s) \cup (z''_p \cap z'_s)$
      $C \leftarrow \{(p, \mathcal{M}_p) \mid (p, \mathcal{M}_p, \rho) \in \chi(\omega') \wedge \mathcal{M}_p \nsubseteq \mathcal{L}\}$
      $(l, \mathcal{M}) \leftarrow \mathbf{min}(C)$ {Returns the tuple with minimal $l$}
      $e \leftarrow \mathbf{min}\left(\mathcal{M} - (z''_p \cup z''_s)\right)$
      return pst (repartition $(\omega', \chi(\omega'), e, l), \omega''$)
    end if
  end if
```

```
Procedure 12 adjacent: partial adjacency test
Input: $\omega' = \left(z'_p, z'_s\right) \in \phi\left(r'\right)$ and $\omega'' = \left(z''_p, z''_s\right) \in \phi\left(r''\right)$ are strongly combinable active
  set pairs of $r'$ and $r''$
Output: $\begin{cases} \textbf{true}, & \text{if } r' \text{ and } r'' \text{ are adjacent} \\ \textbf{false}, & \text{if the result is inconclusive} \end{cases}$
  $\mathcal{L} \leftarrow (z'_p \cap z''_p) \cup (z'_p \cap z''_s) \cup (z''_p \cap z'_s)$
  if $|\mathcal{L}| = d - 2$ then
    return true
  else
    return false
  end if
```

Procedure 13 **nonadjacent**: partial nonadjacency test

```
Input: $\omega' = \left(z'_p, z'_s\right) \in \phi\left(r'\right)$ and $\omega'' = \left(z''_p, z''_s\right) \in \phi\left(r''\right)$ are strongly combinable active
  set pairs of $r'$ and $r''$
Output: $\begin{cases} \textbf{true}, & \text{if } r' \text{ and } r'' \text{ are nonadjacent} \\ \textbf{false}, & \text{if the result is inconclusive} \end{cases}$
  $\mathcal{L} \leftarrow (z'_p \cap z''_p) \cup (z'_p \cap z''_s) \cup (z''_p \cap z'_s)$
  $indep \leftarrow |\mathcal{L}|$
  $dep \leftarrow |\{p \in z'_s \cap z''_s : (p, \mathcal{M}_p, \rho) \in \{\chi(\omega') \cup \chi(\omega'')\}$ and $\mathcal{M}_p \subseteq \mathcal{L}\}|$
  $inconc \leftarrow |z'_s \cap z''_s| - dep$
  if $indep + inconc \geq d - 2$ then
    return false
  else
    return true
  end if
```

Procedure 14 **repartition**: active set pair repartitioning

```
Input: $\begin{cases} \omega = (z_p, z_s) \in \phi(r), & \text{a valid active set pair of } r \\ \chi(\omega), & \text{the dependence map of } \omega \\ (e, l) \in z_p \times z_s, & \text{the elements to swap between } z_p \text{ and } z_s \end{cases}$
Output: $\begin{cases} \omega_1 \in \phi(r), & \text{an active set pair of } r \text{ distinct from } \omega \\ \chi(\omega_1), & \text{the dependence map of } \omega_1 \end{cases}$
  $\omega_1 \leftarrow ((z_p + l) - e, (z_s + e) - l)$
  $\chi(\omega_1) \leftarrow \emptyset$
  for all $(p, \mathcal{M}_p, \rho) \in \chi(\omega)$ do
    if $p \neq l$ then
      if $e \in \mathcal{M}_p$ then
        $\varphi \leftarrow \psi_e \rho - \rho_e \psi$ {with $(l, \mathcal{M}_l, \psi) \in \chi(\omega)$}
        $\mathcal{M}'_p \leftarrow \{m \mid m \neq p \wedge \varphi_m \neq 0\}$
        $\chi(\omega_1) \leftarrow \chi(\omega_1) + \left(p, \mathcal{M}'_p, \varphi\right)$
      else
        $\chi(\omega_1) \leftarrow \chi(\omega_1) + (p, \mathcal{M}_p, \rho)$
      end if
    else
      $\chi(\omega_1) \leftarrow \chi(\omega_1) + (e, (\mathcal{M}_p - e) + l, \rho)$
    end if
  end for
  return $(\omega_1, \chi(\omega_1))$
```

6.4 The Double Description Method Enhanced

Corollary 6.11. *In the context of the dd method's iteration step, consider the extreme rays r', r'' and $r^\triangledown$ emerging from the generators $x' \in X_i^+$, $x'' \in X_i^-$ and $x^\triangledown \in X_i^\triangledown$ where*

$$x^\triangledown = (a_j x')x'' - (a_j x'')x' \ .$$

Assuming that $\omega' = \left(z_p', z_s'\right)$ and $\omega'' = \left(z_p'', z_s''\right)$ are strongly combinable active set pairs which satisfy Proposition 6.5*, then*

$$\omega^\triangledown = \left(\mathcal{L} + j, z_s' \cap z_s''\right) \in \phi\left(r^\triangledown\right) \tag{6.19}$$

is a normalized active set pair of $r^\triangledown$.

Proof. Assume that $\omega^\triangledown$ is not normalized. Then for some $l \in z_s' \cap z_s''$ and $m \in \mathcal{L}$ with $l < m$ there exists a circuit $D \subseteq (\mathcal{L} + l) \cap \mathbb{N}_{\leq m}$ containing both m and l. It is evident that either $m \in z_p'$ or $m \in z_p''$. In the former case, applying Lemma 6.6 with $r = r'$ and $K = z_s' \cap (z_p'' \cup z_s'')$ implies that

$$D \nsubseteq (z_p' \cup (z_s' \cap (z_p'' \cup z_s''))) \cap \mathbb{N}_{\leq m} \supseteq (\mathcal{L} + l) \cap \mathbb{N}_{\leq m} \ .$$

Analogously, in the latter case, we can show that

$$D \nsubseteq (z_p'' \cup (z_s'' \cap (z_p' \cup z_s'))) \cap \mathbb{N}_{\leq m} \supseteq (\mathcal{L} + l) \cap \mathbb{N}_{\leq m}$$

by applying Lemma 6.6 with $r = r''$ and $K = z_s'' \cap (z_p' \cup z_s')$. Consequently, the nonexistence of D and thus the normalization of $\omega^\triangledown$ are proven. □

The integration of the partitioned sets test requires three major changes of the dd method:

(i) the substitution of active sets by normalized active set pairs,

(ii) the attachment of a dependence map to each such pair and

(iii) the application of Procedure 11 in place of the classic algebraic test.

It is important to note that an active set pair is valid only in the context of one iteration step. For some extreme ray $r \in \mathcal{R}_i$, the normalized active set pair $\omega_i^n(r)$ is not necessarily equal to $\omega_{i+1}^n(r)$ which is the normalized active set pair related to the same ray r but in the next iteration step. Analogously, the corresponding dependence maps $\chi\left(\omega_i^n(r)\right)$ and $\chi\left(\omega_{i+1}^n(r)\right)$ may differ as well.

The described changes lead to the following enhancements of dd method's steps.

Initial Step: As described in Section 4.1, $\mathcal{C}_0$ has exactly d extreme rays. The normalized active set pair of each ray r_i is then

$$\omega_0^n(r_i) = \left(z\left(r_i\right), \emptyset\right) \ .$$

The corresponding dependence map is empty; hence,

$$\chi\left(\omega_0^n(r_i)\right) = \emptyset \ .$$

Iteration Step: Assume that for each extreme ray $r \in \mathcal{R}_i$ the normalized active set pair $\omega_i^n(r)$ and its dependence map $\chi\left(\omega_i^n(r)\right)$ are known. The extreme rays

$$\left(r', r''\right) \in \mathcal{R}_i^+ \times \mathcal{R}_i^-$$

are adjacent if and only if

$$\mathbf{pst}\left(\omega_i^n(r'), \chi\left(\omega_i^n(r')\right), \omega_i^n(r'')\right) = \mathbf{true} \quad .$$

After all adjacent extreme rays have been collected, a set $\mathcal{R}_i^\triangledown$ containing all new extreme rays emerges; hence,

$$\mathcal{R}_{i+1} = \mathcal{R}_i^- \cup \mathcal{R}_i^0 \cup \mathcal{R}_i^\triangledown \quad .$$

This is the point where the traditional dd method proceeds with the next iteration step or terminates if all constraints have been already processed. When using the partitioned sets test, however, additional postprecessing actions are necessary in order to satisfy the preconditions of the next iteration step; that is, we have to make sure that $\omega_{i+1}^n(r)$ and $\chi\left(\omega_{i+1}^n(r)\right)$ are available for each extreme ray $r \in \mathcal{R}_{i+1}$. In this respect, the following three cases are to be considered.

If $r \in \mathcal{R}_i^-$, then the active set pair of r and the corresponding dependence map remain unchanged; that is,

$$\omega_{i+1}^n(r) = \omega_i^n(r) \text{ and } \chi\left(\omega_{i+1}^n(r)\right) = \chi\left(\omega_i^n(r)\right) \quad .$$

If $r \in \mathcal{R}_i^0$, then $z(r)$ is extended by the element j. Let $\omega_i^n(r) = (z_p, z_s)$. Then

$$\omega_{i+1}^n(r) = (z_p, z_s + j) \quad .$$

The extension of the active set pair also triggers an extension of the corresponding dependence map. The new element j is part of a circuit $D \subseteq z_p + j$ which has to be calculated and stored in the dependence map as a tuple $(j, D - j, \psi)$. Provided all known circuits are organized in a global structure as described in Section 6.2, we can check whether such D is already known. If not, D and ψ are obtained by solving the system of linear equations $\left[A_{z_p}^T x \mid a_j^T\right]$. Consequently,

$$\chi\left(\omega_{i+1}^n(r)\right) = \chi\left(\omega_i^n(r)\right) + (j, D - j, \psi) \quad .$$

If $r \in \mathcal{R}_i^\triangledown$, then r emerges as a linear combination of two adjacent extreme rays r' and r''. During the partitioned sets test two strongly combinable active set pairs

$$\omega' = \left(z_p', z_s'\right) \in \phi\left(r'\right) \text{ and } \omega'' = \left(z_p'', z_s''\right) \in \phi\left(r''\right)$$

satisfying Proposition 6.5 have been found. Provided those can be accessed after the adjacency test, the normalized active set pair of r is

$$\omega_{i+1}^n(r) = \left(\mathcal{L} + j, z_s' \cap z_s''\right)$$

according to Corollary 6.11. Its dependence map is then

$$\chi\left(\omega_{i+1}^n(r)\right) = \left\{(l, \mathcal{M}, \rho) \in \left\{\chi\left(\omega'\right) \cup \chi\left(\omega''\right)\right\} : l \in z_s' \cap z_s'' \text{ and } \mathcal{M} \subseteq \mathcal{L}\right\} \quad .$$

6.5 Experimental Results

The benchmark from Section 5.6 was used once again to assess the competitiveness of the partitioned sets test with regard to the classic algebraic test and the combinatorial one. The obtained results are summarized in Table 6.1. It should be said that each program configuration had a time limit of four hours to solve a single problem group; that is, sixteen hours for all problems. The classic algebraic test did not manage to finish the computation within the given time constraints; thus, exact computational times could not be obtained. The tendency, however, is clear: it is hardly competitive with the other two adjacency test variants. The opposite can be said for the partitioned sets test. It turned out to be a very promising alternative to the combinatorial test for a broad variety of problems. Its general performance depended very much on the size of the maintained cache. The smaller the size, the better the performance. Consequently, high dimensional problems featuring very high degeneracy remained a domain where the combinatorial test could still claim a superiority.

	combinatorial	algebraic	
		pst	classic
RANDOM_4_16_64	161.525	**136.384**	528.466
RANDOM_4_16_96	27.783	**23.207**	93.475
RANDOM_6_64_96	94.041	**83.535**	🕒
RANDOM_8_128_157	71.136	**68.019**	🕒
RANDOM_6_16_104	519.205	**414.150**	1 427.240
RANDOM_4_64_112	154.722	**113.594**	🕒
RANDOM_4_128_172	182.097	**132.198**	🕒
RANDOM_4_16_112	68.114	**63.281**	1 052.895
RANDOM_16_68_115	58.910	**16.883**	🕒
RANDOM_12_128_174	67.625	**18.554**	🕒
RANDOM_8_20_160	**150.944**	714.888	🕒
RANDOM_16_64_179	**1 296.747**	1 500.953	🕒
RANDOM_24_128_277	**1 470.488**	3 402.260	🕒

Table 6.1: Combinatorial vs. algebraic test

7 Divide and Conquer: Advantages and Limitations

One way of fighting the computational complexity of the convex hull problem is by parallelizing the computation. The dd method can be easily adapted for parallel computation by distributing its most time consuming part, the adjacency testing of extreme rays, on different processors. By applying the *master-and-slave* [129] paradigm, a central processor is given a master role of assigning adjacency tests to other available processors, the so called slaves, and then collecting and postprocessing the results at the end of each incremental step. Several implementations employing that approach have been introduced with corresponding experimental results provided and evaluated [130, 103, 145]. This technique, however, has some serious limitations with regard to the scalability of the parallelization. First, we cannot effectively break the sequentiality as synchronization is required at the end of each iteration step. If adjacency tests are to be distributed on several machines, this may result in potentially large amounts of data being shifted over the communication network. Moreover, another problem arises from the potentially exponential growth of intermediate results. At some point they may not fit into the operational memory of the master processor; hence, even having enough computational capacity does not guarantee to solve a particular problem as the algorithm may still fail due to lack of space.

In order to effectively deal with the above limitations, another parallel processing paradigm, the *divide-and-conquer* one [86], can be applied. The general idea rests on *backtracking* [76] and goes back as far as the algorithm of Balinski [17]. Let $\mathcal{C}_i$ be a polyhedral cone with known $\mathcal{H}$- and $\mathcal{V}$- representations and let $a_j, \dots, a_n$ be a sequence of row vectors defining linear halfspaces to cut $\mathcal{C}_i$ with. We can think of $\mathcal{C}_i$ as an intermediate cone within the dd method where $a_j, \dots, a_n$ are the unprocessed rows of A. Instead of performing the traditional sequential step with $\mathcal{C}_{i+1} = \mathcal{C}_i \cap \{x \mid a_j x \leq 0\}$, an alternative route can be followed.

(i) Split the computation into two threads, and pass $\mathcal{C}_i$ and $a_j, \dots, a_n$ to each of them.

(ii) Within the first thread, compute $\mathcal{C}'_{i+1} = \mathcal{C}_i \cap \{x \mid a_j x = 0\}$ and then proceed the computation with $\mathcal{C}'_{i+1}$.

(iii) Within the second thread, skip a_j for the moment and proceed directly with a_{j+1}. At the end of the computation, when a final cone $\mathcal{C}''$ is reached, extract only those extreme rays of $\mathcal{C}''$ which lie strictly within the halfspace $\{x \mid a_j x < 0\}$.

In Section 7.1, the above idea is presented in greater detail. Following that, a divide-and-conquer version of the dd method is given in Section 7.2.

The divide-and-conquer approach has been discussed on several occasions by the bioinformatics community in the context of extreme pathway enumeration [100, 93] and recently applied in practice by Jevremović et al. [92]. The authors reported a substantial performance gain which opened the question of its applicability for general problems. The computational experience gathered so far has been limited to the particular area of metabolic networks; hence, no general conclusions could be derived. Here, a complementary computational experience is provided to demonstrate that for certain problems, a class of cut polytopes [18], the divide-and-conquer approach leads to a serious computational explosion (see Section 7.3). The described behavior could not be avoided by changing the insertion order of the rows in A (see Section 7.4). However, in the course of experiments, another way of performing a partial divide-and-conquer computation emerged. Splitting the computation in its final stages turned out to be robust enough, despite achieving only a moderate performance gain.

7.1 From a Sequence to a Tree

Let $\mathcal{C}(A), A \in \mathbb{R}^{n \times d}$ be a polyhedral cone for which a minimal set of generators X should be computed. In this context, the application of the dd method corresponds to finding an initial cone $\mathcal{C}_0$ and then sequentially refining it by adding one halfspace constraint at a time. In total, we obtain $n-d$ refinement steps. At each step i, both $\mathcal{H}$- and $\mathcal{V}$-representations of the intermediate cone are available forming a double description pair

$$p_i = (A_i, X_i)$$

which consists of a $(d+i) \times d$ matrix and a minimal set of the generators for $\mathcal{C}(A_i)$. Consider now the tuple

$$t = (A^{\circ}, A^{\bullet}, X^{\circ}, A^{\triangleright}) \tag{7.1}$$

with A°, $A^{\bullet}$ and $A^{\triangleright}$ disjoint row submatrices of A_i such that

A° and $A^{\bullet}$ define an intermediate cone $\mathcal{C}^{\circ} = \{x \mid A^{\circ}x \leq 0, A^{\bullet}x = 0\}$,

$A^{\triangleright}$ contains all other rows of A_i not appearing in $A^{\bullet}$ or A°, and

X° contains a minimal set of generators corresponding to $\mathcal{C}^{\circ}$.

It is obvious that the tuple given in (7.1) is at least as powerful as a dd pair due to the trivial equivalence

$$(A_i, X_i) \sim (A_i, \perp, X_i, \perp) \tag{7.2}$$

with $\perp$ denoting the empty matrix. Additionally, it enables partial representations of the intermediate cone $\mathcal{C}(A_i)$ in the sense that

$$t' = \left(A_{i-1}, \left[a_l\right], X_i', \perp\right) \qquad \text{with } l = d + i$$

represents one facet of $\mathcal{C}(A_i)$ emerging from the intersection of $\mathcal{C}(A_{i-1})$ with the hyperplane $\mathcal{H}_l = \{x \mid a_l x = 0\}$, whereas

$$t'' = \left(A_{i-1}, \perp, X_{i-1}, \left[a_l\right]\right)$$

still corresponds to the cone $\mathcal{C}\left(A_{i-1}\right)$, but also suggests that an intersection with $\mathcal{H}_l$ has been performed elsewhere. It is easy to see that (t', t'') is an equivalent representation of p_i as

$$A_i = \begin{bmatrix} A_{i-1} \\ a_l \end{bmatrix} \text{ and } X_i = X_i' \cup \{x \in X_{i-1} : a_l x < 0\}$$

are trivially obtainable out of (t', t''). Consequently, we shall call the tuples defined in (7.1) *fragments* of a dd pair.

Each fragment can be further divided by selecting a facet and expressing separately those generators which lie on the facet and those which do not; that is, for some matrix row a_q of A°,

$$(A^\circ, A^\bullet, X^\circ, A^\triangleright) \sim \left(\left(A^{\circ\prime}, \begin{bmatrix} A^\bullet \\ a_q \end{bmatrix}, X^{\circ\prime}, A^\triangleright \right), \left(A^{\circ\prime}, A^\bullet, X^{\circ\prime\prime}, \begin{bmatrix} A^\triangleright \\ a_q \end{bmatrix} \right) \right) \tag{7.3}$$

where

$A^{\circ\prime}$ denotes the row submatrix of A° obtained by removing a_q,

$X^{\circ\prime} \subseteq X^\circ$ contains the generators lying on the hyperplane $\{x \mid a_q x = 0\}$, and

$X^{\circ\prime\prime}$ stands for a minimal set of generators corresponding to the cone

$$\left\{ x \mid A^{\circ\prime} x \leq 0, A^\bullet x = 0 \right\} \ .$$

Consequently, it is easily provable that

$$x \in X^\circ \Leftrightarrow x \in X^{\circ\prime} \vee \left(x \in X^{\circ\prime\prime} \wedge a_q x < 0 \right) \ ,$$

thus the recursive application of the equivalence rules given in (7.2) and (7.3) yields an alternative representation for each dd pair by finitely many fragments.

Let $T_i = \{t_1, \dots, t_g\}$ be a set of fragments representing the dd pair p_i with

$$t_k = (A_k^\circ, A_k^\bullet, X_k^\circ, A_k^\triangleright) \quad \text{for } 1 \leq k \leq g \ .$$

In context of the dd method, the transition to p_{i+1} corresponds to

$$T_i \longrightarrow \bigcup_{k=1}^{g} \left\{ \left(\begin{bmatrix} A_k^\circ \\ a_j \end{bmatrix}, A_k^\bullet, \hat{X}_k^\circ, A_k^\triangleright \right) \right\} \tag{7.4}$$

with $\hat{X}_k^\circ$ a minimal set of generators satisfying

$$\texttt{cone}\left(\hat{X}_k^\circ\right) = \texttt{cone}\left(X_k^\circ\right) \cap \{x \mid a_j x \leq 0\} \ .$$

By applying the equivalence given in (7.3), we obtain an alternative transition

$$T_i \longrightarrow \bigcup_{k=1}^{g} \left\{ \left(A_k^\circ, \begin{bmatrix} A_k^\bullet \\ a_j \end{bmatrix}, X_k^{\circ\prime}, A_k^\triangleright \right), \left(A_k^\circ, A_k^\bullet, X_k^\circ, \begin{bmatrix} A_k^\triangleright \\ a_j \end{bmatrix} \right) \right\} \tag{7.5}$$

which doubles the number of fragments. Consequently, the traditional dd method can be redefined from performing a sequential computation of dd pairs to a divide-and-conquer algorithm which operates on fragments instead (see Figure 7.1).

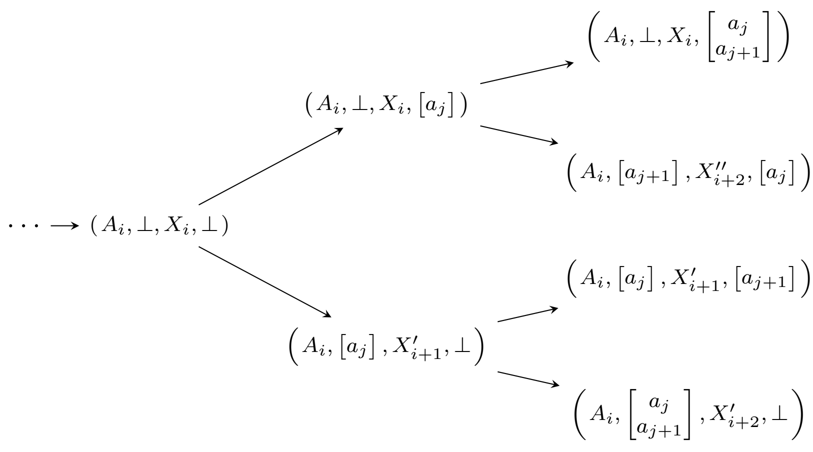

$\left(A_i, \bot, X_i, \begin{bmatrix} a_j \\ a_{j+1} \end{bmatrix}\right)$
$(A_i, \bot, X_i, [a_j])$
$\left(A_i, [a_{j+1}], X''_{i+2}, [a_j]\right)$
$\cdots \longrightarrow (A_i, \bot, X_i, \bot)$
$\left(A_i, [a_j], X'_{i+1}, [a_{j+1}]\right)$
$\left(A_i, [a_j], X'_{i+1}, \bot\right)$
$\left(A_i, \begin{bmatrix} a_j \\ a_{j+1} \end{bmatrix}, X'_{i+2}, \bot\right)$

Figure 7.1: Divide-and-Conquer

7.2 The Distributed Double Description Method

In Procedure 15, an extended version of the dd method is presented which is suitable for parallel computation. The proposed algorithm is parametrized by a list L indicating those steps at which the transition given in (7.5) is applied; that is, it indicates the steps at which the computation is being split. All other steps are performed according to the transition given in (7.4). Furthermore, it assumes the existence of the functions

dd_cut_plane $(X_i, a_j) = X_{i+1}$ which executes a stricter form of the iteration step in the sense that $\texttt{cone}\,(X_{i+1}) = \texttt{cone}\,(X_i) \cap \{x \mid a_j x = 0\}$, and

dd_cut_space $(X_i, a_j) =$ **dd_cut_plane** $(X_i, a_j) \cup \{x \in X_i : a_j x < 0\}$ which covers the traditional form of the iteration step where a_j is satisfied with inequality.

The reasoning behind the divide-and-conquer approach becomes evident when comparing the transitions given in (7.4) and (7.5) in the context of Procedure 15. For one thing, **dd_cut_plane** is a stricter form of **dd_cut_space** and thus produces less intermediate generators. This reduces the overall amount of data passed onto the successive steps. Consequently, in the main thread, less computational effort is expected when applying divide-and-conquer. The critical point, however, is the new thread which is opened with the invocation of **fork** $(\dots)$. The matrix passed to that thread is effectively the original one reduced by one constraint; hence, the assumption is that it requires less effort to compute as well. Assume now that an unlimited number of free processors is available and thus each invocation of **fork** guarantees the exclusive assignment of a free processor to the resulting new thread. Then, for $L_1 \supset L_2$, the execution time of **dd_comp** $(\dots, L_1)$ should not exceed that of **dd_comp** $(\dots, L_2)$.

In practice, we have seldom the luxury of possessing an unlimited number of processors. We thus have to decide when to split the computation and when to let it run sequentially. Moreover, the number of fork operations grows exponentially if divide-and-conquer is to be applied on each consecutive step. Let 2^t be the number of available processors. Then, Procedure 15 **dd_comp**: divide-and-conquer version of the dd method

```
Input: { $A$, $\mathcal{H}$-representation of the polyhedral cone
       { $l$, current row index
       { $t = (A^{\circ}, A^{\bullet}, X^{\circ}, A^{\triangleright})$, current dd pair fragment
       { $L$, list of divide-and-conquer steps
Output: A minimal set of generators corresponding to $\mathcal{C}(A)$
if $l > \mathbf{rows}(A)$ then
    return $\{x \in X^{\circ} : A^{\triangleright} x < 0\}$
else
    if $l \notin L$ then
        $A^{\circ\prime} \leftarrow \begin{bmatrix} A^{\circ} \\ a_l \end{bmatrix}$; $X^{\circ\prime} \leftarrow \mathbf{dd\_cut\_space}(X^{\circ}, a_l)$; $t' \leftarrow (A^{\circ\prime}, A^{\bullet}, X^{\circ\prime}, A^{\triangleright})$
        return $\mathbf{dd\_comp}(A, l+1, t', L)$
    else
        $A^{\bullet\prime} \leftarrow \begin{bmatrix} A^{\bullet} \\ a_l \end{bmatrix}$; $X^{\circ\prime} \leftarrow \mathbf{dd\_cut\_plane}(X^{\circ}, a_l)$; $t' \leftarrow (A^{\circ}, A^{\bullet\prime}, X^{\circ\prime}, A^{\triangleright})$
        $A^{\triangleright\prime\prime} \leftarrow \begin{bmatrix} A^{\triangleright} \\ a_l \end{bmatrix}$; $t'' \leftarrow (A^{\circ}, A^{\bullet}, X^{\circ}, A^{\triangleright\prime\prime})$
        return $\mathbf{dd\_comp}(A, l+1, t', L) \cup \mathbf{fork}(\mathbf{dd\_comp}(A, l+1, t'', L))$
    end if
end if
```

we can outline two major strategies for performing the parallelization. For one thing, we can split the computation from the beginning onto 2^t threads and then proceed with the classic dd method on each of them. This approach basically corresponds to the algorithm proposed by Jevremović et al. [92]. We shall call it an *early split*. Alternatively, we can compute the majority of steps on a single processor and then distribute the last t steps on the available processors. This approach has not been investigated so far. We shall call it a *late split*. With respect to the construction of L, an early split implies that $L = \{d+1, \ldots, d+t\}$, whereas a late split is defined by $L = \{n-t, \ldots, n-1\}$. It is evident that in both cases the preprocessing of the input matrix may have an impact on the subsequent performance as it implicitly defines the constraints upon which divide-and-conquer steps are executed.

7.3 Experimental Results

The facet enumeration of cut polytopes [18] has a long history of serving as a benchmark for convex hull algorithms and thus makes a good starting point to assess the divide-and-conquer technique. The test set considered here contains the cut polytope of a complete graph on seven vertices CCP_7 (available in the cdd package [75]) as well as three cut polytopes of incomplete graphs on eight vertices with 20 to 22 edges (CP_8_20,…,CP_8_22). The experimental results for early and late split are summarized in Tables 7.1 and 7.2 with all time values given in seconds. For each problem, the rows of the input matrix were rearranged by imposing four different variations of the lexicographic order, which is known to work well for cut polytopes (see [77]). All experiments were performed with `addibit` on CentOS 6.5 running 3 Intel Xeon processors with 4 cores each and 16 gigabytes of operating memory.

	order	1 thread	2 threads		4 threads			
CCP_7	lex	23.3	3.5	53.3	0.9	17.3	20.1	51.3
	colex	23.4	3.7	54.8	0.9	17.9	21.0	52.8
	rev-lex	23.2	3.8	52.7	0.9	16.8	19.5	50.9
	rev-colex	23.4	3.7	54.2	1.0	17.7	20.1	52.2
CP_8_20	lex	24.1	3.7	40.3	0.9	6.0	6.5	71.3
	colex	24.9	1.6	34.3	0.3	3.1	3.2	50.6
	rev-lex	24.2	3.7	40.4	0.9	6.0	6.5	71.1
	rev-colex	24.5	1.6	33.8	0.3	3.1	3.2	50.6
CP_8_21	lex	70.4	4.3	106.5	1.2	20.2	20.3	121.0
	colex	131.3	8.5	204.2	1.9	40.4	39.4	254.9
	rev-lex	71.2	4.3	106.5	1.2	20.2	20.3	121.3
	rev-colex	132.6	8.4	204.0	1.9	39.7	39.6	252.5
CP_8_22	lex	418.8	26.7	840.7	2.0	41.2	37.7	1202.8
	colex	1057.4	59.0	2045.6	2.9	91.1	74.0	2718.5
	rev-lex	416.9	26.9	840.3	1.9	41.3	37.6	1212.8
	rev-colex	1053.3	59.2	2043.2	2.9	90.9	74.0	2710.4

Table 7.1: Early split on cut polytopes (time per thread)

	$n \times d$	order	1 thread	2 threads	4 threads	8 threads	16 threads
CCP_7	64×22	lex	23.6	20.9	17.6	16.0	15.6
CP_8_20	128×21		24.1	23.7	23.0	22.4	22.1
CP_8_21	128×22		70.4	68.3	66.4	63.1	60.5
CP_8_22	128×23		418.8	398.9	386.8	360.6	349.0

Table 7.2: Late split on cut polytopes (overall time)

7.4 Discussion

The divide-and-conquer algorithm splits the computation by reducing the dimensionality of the problem on one of the resulting threads and the number of defining halfspaces on the other one. Another way to look at the algorithmic step is of a matrix which is reduced by one column on the first thread and by one row on the second one. Thus, the application of the early split strategy on two threads reduces the original $n \times d$ matrix to an $n \times d - 1$ one on thread 1 and an $n - 1 \times d$ one on thread 2. On four threads, we obtain an $n \times d - 2$ matrix on thread 1, $n - 1 \times d - 1$ matrices on threads 2 and 3, and an $n - 2 \times d$ matrix on thread 4. For the here investigated cut polytopes, removing merely rows from the input matrix has obviously an adverse effect on the problem's complexity. This fact may seem initially counterintuitive; however, the removal of those halfspaces from the input changes the computational run of the dd method. In geometric terms, the algorithm has to deal now with extreme rays which lie within the eliminated halfspaces and as such would have been discarded in the classic computation. Consequently, the algorithm has less steps but may still produce more intermediate rays due to the changed geometry of the polyhedral cone.

In the previous work (see [92]), the rows of the input matrix have been rearranged manually in order to obtain better results. Regarding the cut polytopes investigated here, however, it is not clear how to define an ordering which both makes early split scalable and remains competitive in the sequential scenario. Consider, for example, the computational run on two threads for CCP_7 resulting from the reversed lexicographic order (rev-lex). We can manually select another row a_{l_1} to be excluded from the computation on thread 2 by putting a_{l_1} at position $d + 1$ in the input matrix A_7. The objective is obviously to bring the running time on that thread below the mark set by the sequential computation. In total, there are five rows $a_{60}, \ldots, a_{64}$ satisfying that requirement (see Figure 7.2). In those cases, however, the performance of thread 1 takes a significant hit after rearranging the rows. In the end, the parallel algorithm outperforms the sequential one in two cases only, when either a_{60} or a_{61} is excluded. Let A_7' and A_7'' denote the rearranged matrices in those two cases. With both of them, the dd method has worse execution time on four threads. Yet, following the previous strategy, we can again manually select another row a_{l_2} to be excluded from A_7' or A_7'' on thread 4 in order to rebalance the computation. All rearrangements in A_7'' showed no improvement (see Figure 7.4). For A_7', a_{59} is the only row which brings some speedup (see Figure 7.3) On eight threads, however, the execution time drops once again and here no rearrangements could improve it anymore (see Figure 7.5). Similar experiments were performed with other rearrangement strategies eventually leading to the same result.

The removal of constraints from the input matrix does not have such a strong impact in the late split scenario as the dimensionality of the problem is reduced within each step as soon as the parallelization starts. That particular technique delivered only a minimal speedup for cut polytopes; however, it should be pointed out that the majority of the computation was still performed in a sequential manner. For example, a late split on eight threads means that one algorithmic step is distributed on eight threads, one on four, one on two and the rest is still dealt with sequentially. Hence, for a problem like CP_8_22 with 105 steps, we can hardly expect to gain much time as the last three steps take a comparably small amount of time to compute. The late split strategy is better suited for problems where most of the calculation is concentrated within the last few steps. Indeed, such problems arise often in practice as the dd method tends to produce more and more intermediate rays with each step which stalls the computation towards the end. In order to demonstrate such a scenario, five additional problems are presented

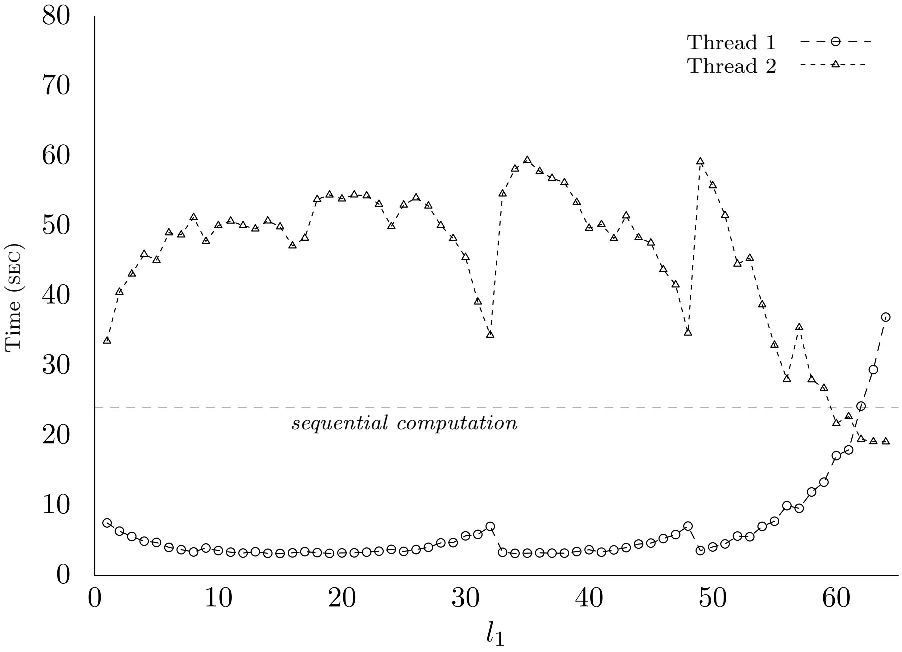


Figure 7.2: Early split on two threads for CCP_7

in Table 7.3[1]. The first two problems feature the facet enumeration of polytopes which arise from the union of a 4-cube with a random 0/1 polytope. The latter three require building the convex hull of randomly generated integer vectors. Note that late split is more competitive when more of the time intensive steps are distributed on multiple threads (e.g., compare 2 vs 16 threads).

	$n \times d$	1 thread	2 threads	4 threads	8 threads	16 threads
RND_CUBE_28	45 × 29	61.0	35.0/ 29.9	28.6/ 15.8	17.1/ 9.7	11.1/ 8.1
RND_CUBE_30	47 × 31	200.4	170.2/115.1	140.9/ 96.0	83.7/ 79.4	52.6/ 64.7
RND_6_64_96	96 × 64	129.5	69.2/ 46.9	52.9/ 35.8	32.9/ 26.0	22.7/ 23.0
RND_8_128_157	157 × 128	101.8	52.9/ 38.6	27.7/ 34.2	21.8/ 17.0	14.7/ 14.6
RND_16_64_167	167 × 64	2483.1	1262.8/395.1	590.4/349.0	479.4/374.5	278.1/394.6

Table 7.3: Late/Early split on randomly generated polytopes

[1] Not included in the initial version of the work

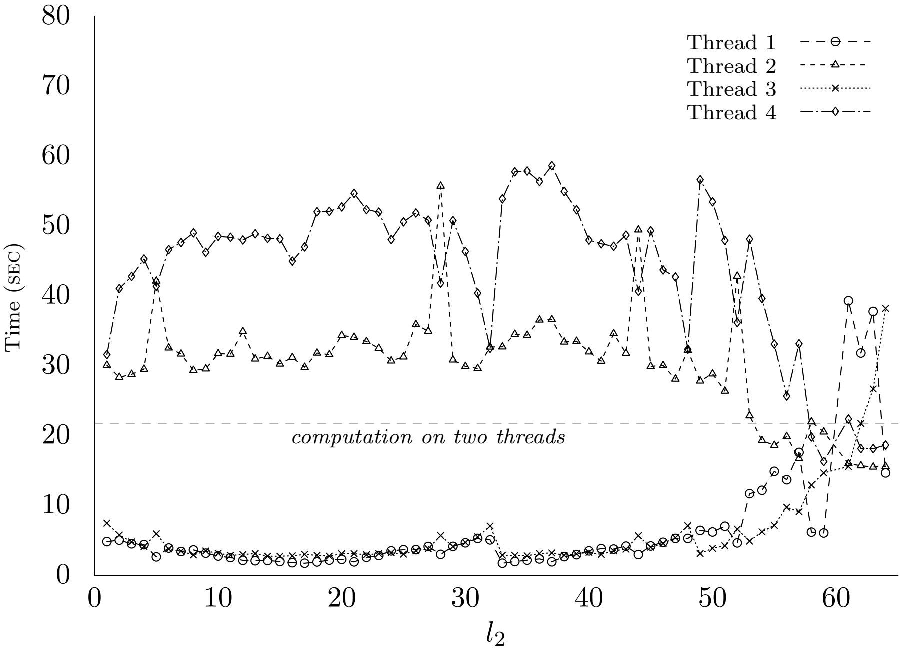


Figure 7.3: Early split on four threads for CCP_7 ($l_1 = 60$)

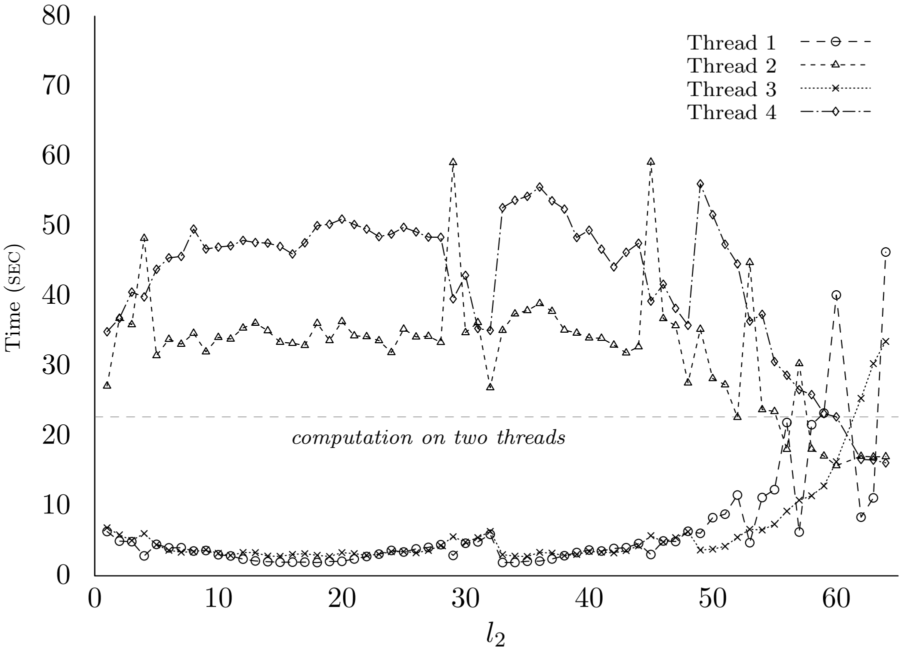


Figure 7.4: Early split on four threads for CCP_7 ($l_1 = 61$)

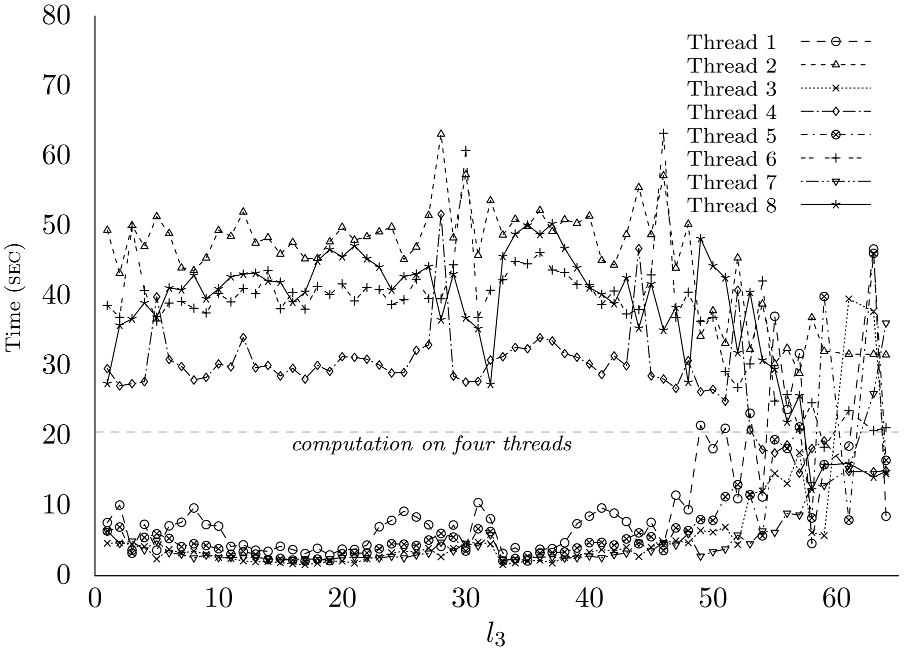


Figure 7.5: Early split on eight threads for CCP_7

8 Addibit: Benchmark and Results

In the course of this work, a new double description implementation ADDIBIT[1] emerged as a framework to conduct the necessary experiments on. Consequently, it is interesting to see how it performs against other noncommercial implementations. Several freely available programs are natural candidates in that respect: CDD [75], FOUR-TI-TWO [1], LRS [8], POLCO [142], PORTA [50], PPL [15] and SKELETON [157]. An overview of their properties is given in Table 8.1.

	FOUR-TI-TWO	SKELETON	PPL	CDD	POLCO	PORTA	LRS	ADDIBIT
Algorithm	PL[1]	DD[2]	DD	DD	DD	FM[3]	RS[4]	DD
Input Format	.mat	.ine/.ext	.ine/.ext	.ine/.ext	.ine/.ext	.ieq	.ine/.ext	.ine/.ext
Preprocessing	cutoff	lex cutoff	-	lex cutoff	lex zero cutoff	-	-	lex zero
Rational Arithmetic	✓	✓	✓	✓	✓	✓	✓	✓
Float Arithmetic	✗	✓	✓	✓	✓	✗	✗	✗
Precision Library	GMP[5]	ARAGELI	GMP	GMP	JDK	PORTA	GMP	GMP
Prog. Language	C/C++	C/C++	C/C++	C/C++	Java	C	C	C/C++
Version	1.6.2	2.1.5	1.1	0.77a	4.7.1	1.4.1	5.0	0.3.0
Released	Feb 14	May 13	Oct 13	Oct 07	Dec 09	Sep 09	May 13	Jun 14

[1] Project and Lift Algorithm [87]
[2] Double Description Method
[3] Fourier Motzkin Elimination
[4] Reverse Search
[5] GNU Multiple Precision Arithmetic Library [84]

Table 8.1: Noncommercial programs for convex hull computations

All programs had to solve instances from the benchmark given in Section 5.6 as well as some previously studied problems. The latter involved dwarfed cubes, products of cyclic polytopes, truncated and SSC-polytopes from the work of Avis et al. [10, 11], metric polytopes [61], cut polytopes [18] and products of randomly generated 0/1 polytopes and hypercubes. The programs CDD, POLCO, SKELETON and ADDIBIT were configured

[1] Available at `www.informatik.uni-bremen.de/agbs/bgenov`

to impose a lexicographic order on the input matrix (*lexmin* option). For FOUR-TI-TWO and PORTA, the matrix had been externally sorted in lexicographic order. Unfortunately, PPL does not accept any preprocessing options. It sorts the matrix internally according to its own strategy so that no real influence on the constraints ordering could be exerted. For LRS, preprocessing was not considered significant as it employs a pivoting algorithm.

	FOUR-TI-TWO	SKELETON	PPL	CDD	POLCO	PORTA	LRS	ADDIBIT
RANDOM_4_16_64 Timeout: 3600	🕒	🕒	🕒	🕒	2 605.384	🕒	920.066	**197.003**
RANDOM_4_16_96 Timeout: 3600	🕒	938.751	🕒	🕒	1 365.430	🕒	640.907	**54.523**
RANDOM_6_16_104 Timeout: 3600	🕒	🕒	🕒	🕒	🕒	🕒	🕒	**783.976**
RANDOM_4_16_112 Timeout: 3600	🕒	1 746.262	🕒	🕒	1 108.111	🕒	3 339.556	**78.314**
RANDOM_16_68_115 Timeout: 3600	372.113	🕒	🕒	🕒	1 500.815	🕒	🕒	**206.889**
RANDOM_8_20_160 Timeout: 3600	🕒	🕒	🕒	🕒	1 449.405	🕒	🕒	**251.982**

Table 8.2: Computational results for random problems

Table 8.2 illustrates the computational results related to the benchmark of random problems. Table 8.3 illustrates the results related to all other problems. Here, each row corresponds to a group of similar vertex/facet enumeration problems. For each program, the execution time needed to solve all of them is given. If not able to solve all problems within the given time constraints, the number of successfully computed instances is given together with the corresponding execution time. It should be noted that the problems within each group have been ordered by growing difficulty; that is, each problem had an ID corresponding to its position in the execution queue and was generally harder to solve than the previous one. In Figures 8.1, 8.2, 8.3, 8.4, 8.5 and 8.6, the data from Table 8.3 is illustrated in a more detailed form. Each figure contains two different charts. In the first one, the accumulated running time to solve all problem instances up to a certain ID is illustrated. The second one is related to the amount of operational memory required to solve each single problem.

All experiments were made under UBUNTU 13.10 on a single core of an INTEL® Core™ i7 CPU with 2.4 GHz clock frequency and 16 GB available RAM. Execution

	FOUR-TI-TWO	SKELETON	PPL	CDD	POLCO	PORTA	LRS	ADDIBIT
CYCLIC Timeout: 3600	1416.923 20/22	3252.606 18/22	185.839	2609.208 18/22	3538.800	709.483	2545.142 16/22	**141.376**
SSC Timeout: 3600	**8.138**	4.115 9/16	969.861 13/16	313.791	678.785 11/16	21.518	896.069 6/16	49.338
TRUNCATED Timeout: 3600	633.483	1001.869	1164.061 4/25	2451.494	↯	172.364	1736.209 6/25	**54.951**
DWARFED Timeout: 1000	26.678	2.255	732.635 16/18	572.806 16/18	24.069	237.288 17/18	**0.379**	2.356
METRIC Timeout: 3600	0.594 4/5	0.954 4/5	0.144 4/5	2.028 4/5	↯	0.240 4/5	483.515 4/5	**195.439**
CUT Timeout: 3600	127.770 11/13	1280.690 11/13	778.261 12/13	3400.079 12/13	↯	1102.754 12/13	1036.581 7/13	**98.682**
RND_CUBE Timeout: 3600	**24.438**	1230.457 10/11	1605.315 10/11	1272.860 9/11	367.128	3458.443 9/11	59.763	161.081

Table 8.3: Computational results for specific problems

times are given in seconds. Experiments which could not be finished due to lack of memory or an internal error are denoted with ↯. Experiments which could not be solved within the given time period are denoted with 🕒.

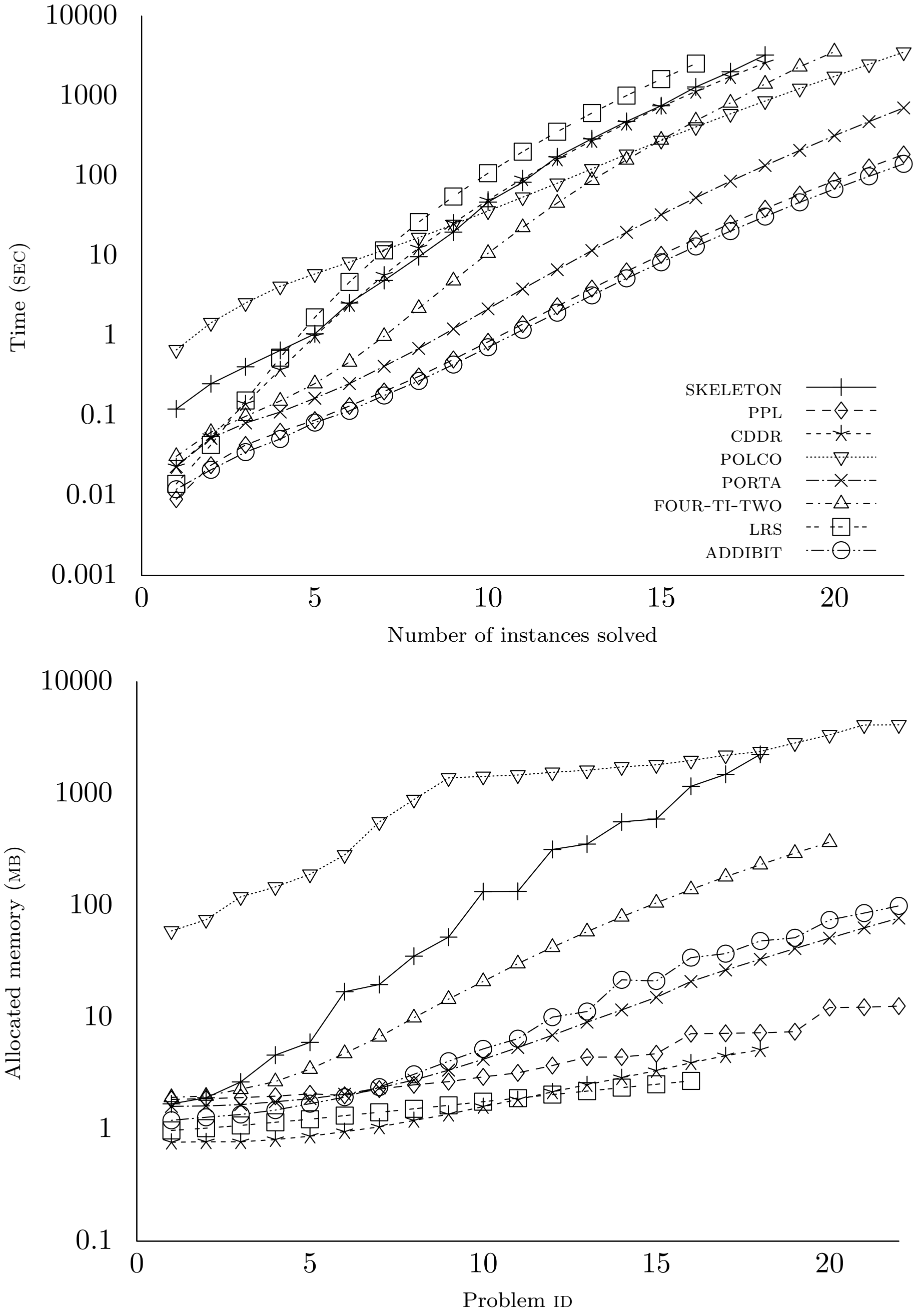
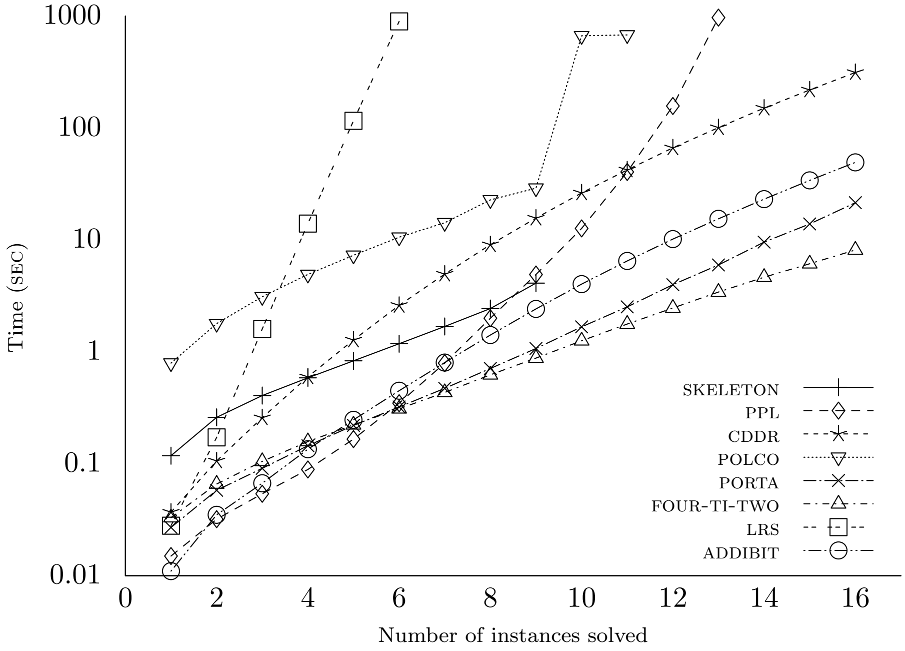


Figure 8.1: Computational results for products of cyclic polytopes



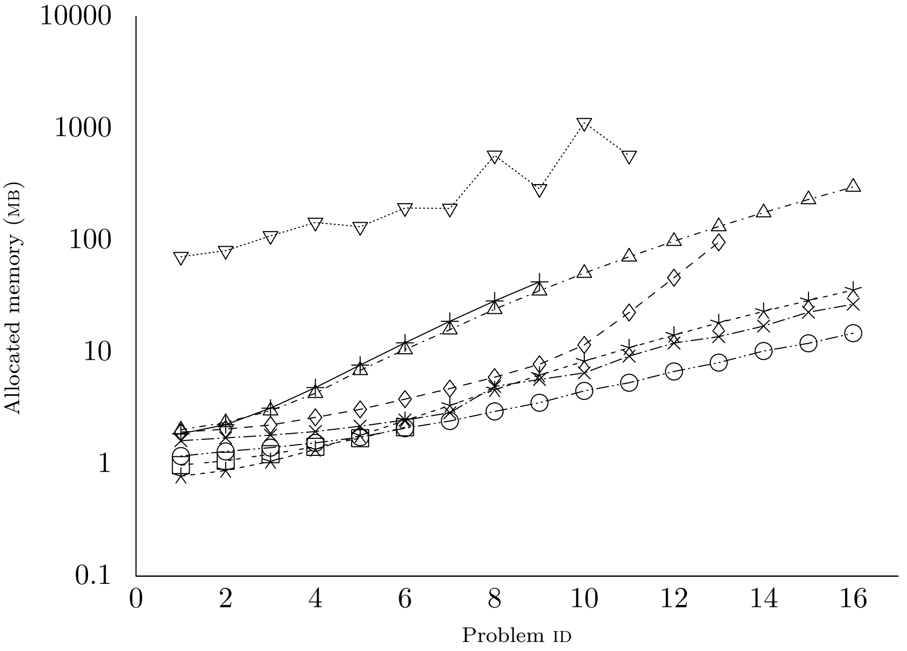


Figure 8.2: Computational results for SSC-polytopes

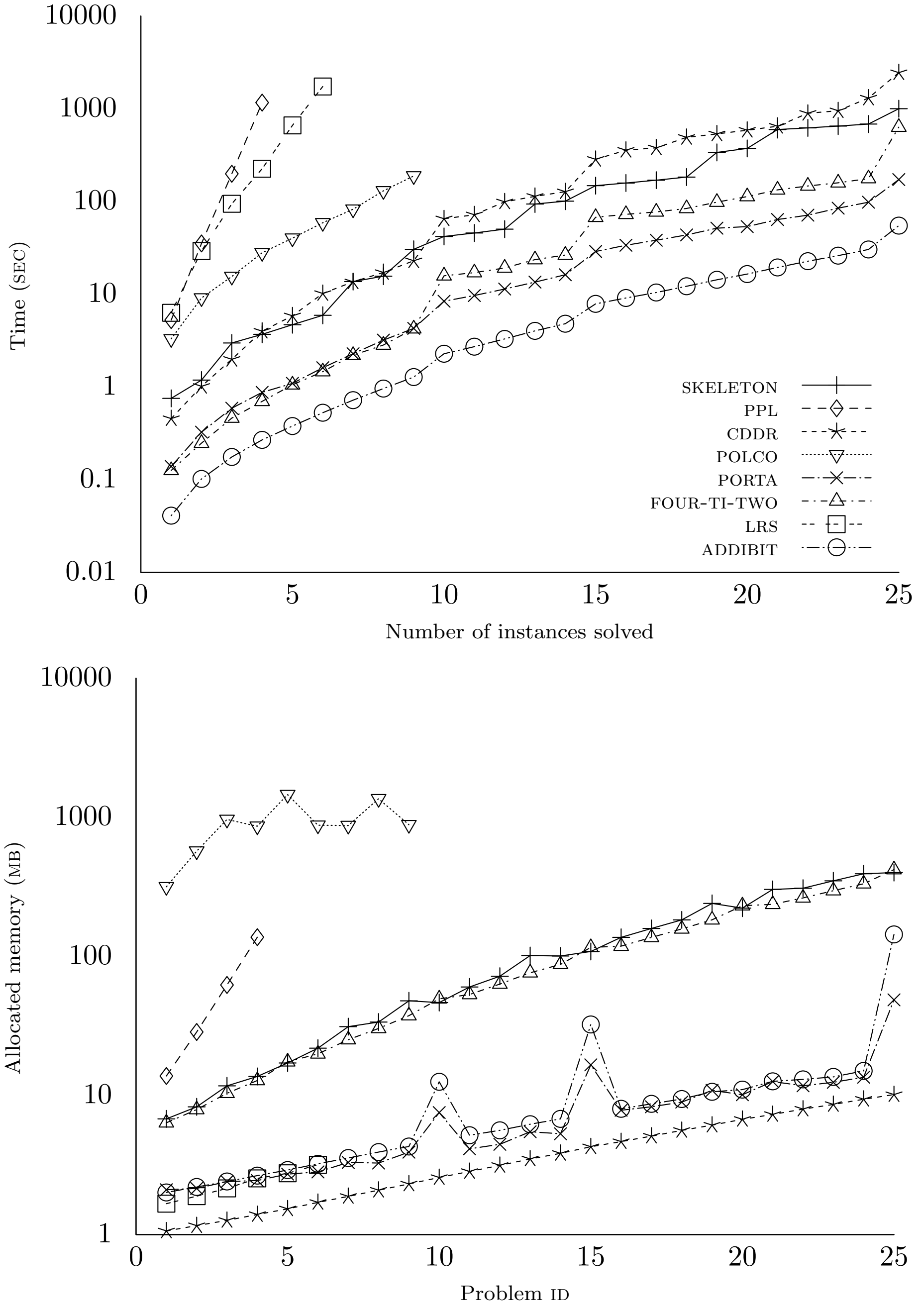


Figure 8.3: Computational results for truncated polytopes

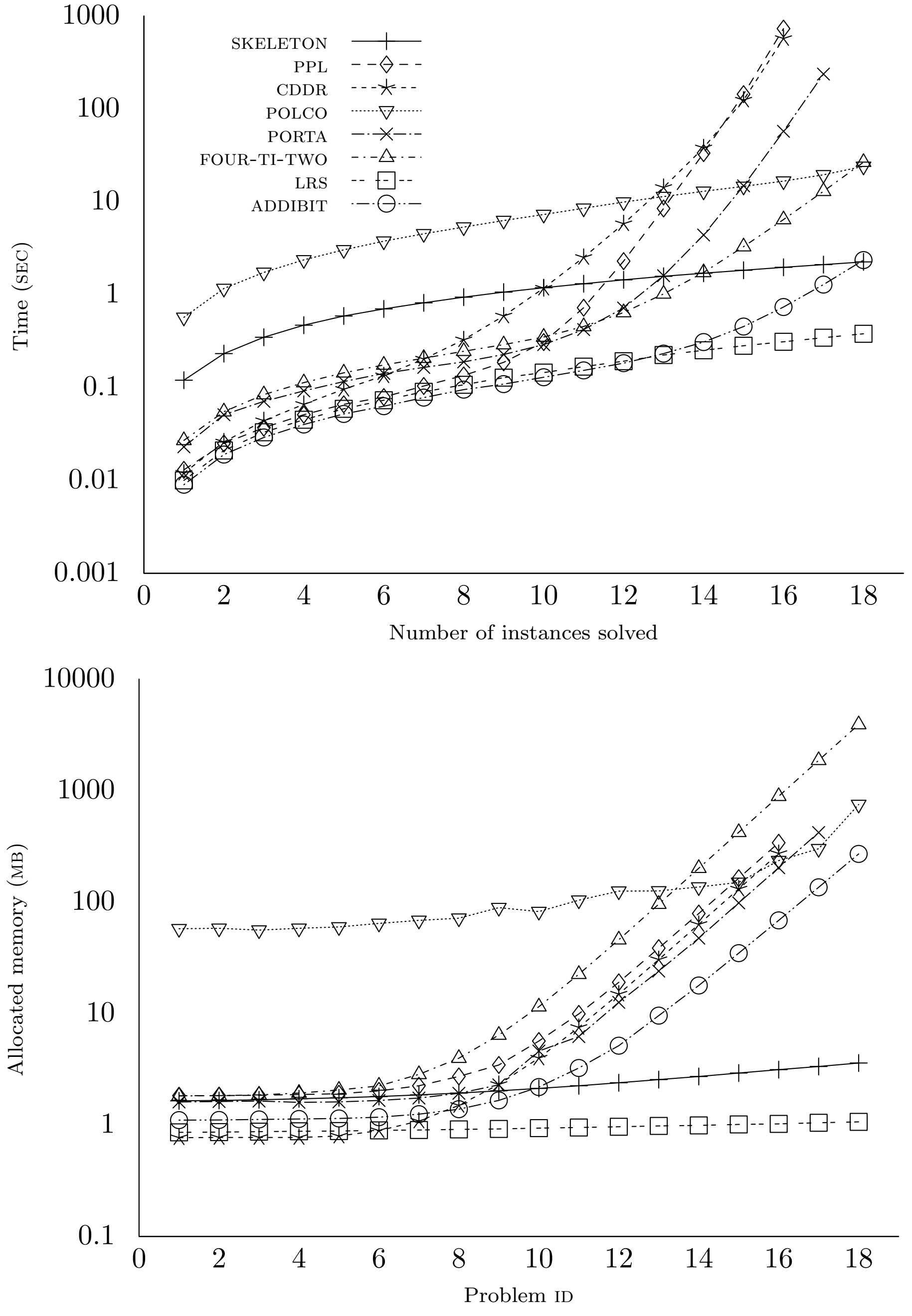


Figure 8.4: Computational results for dwarfed cubes

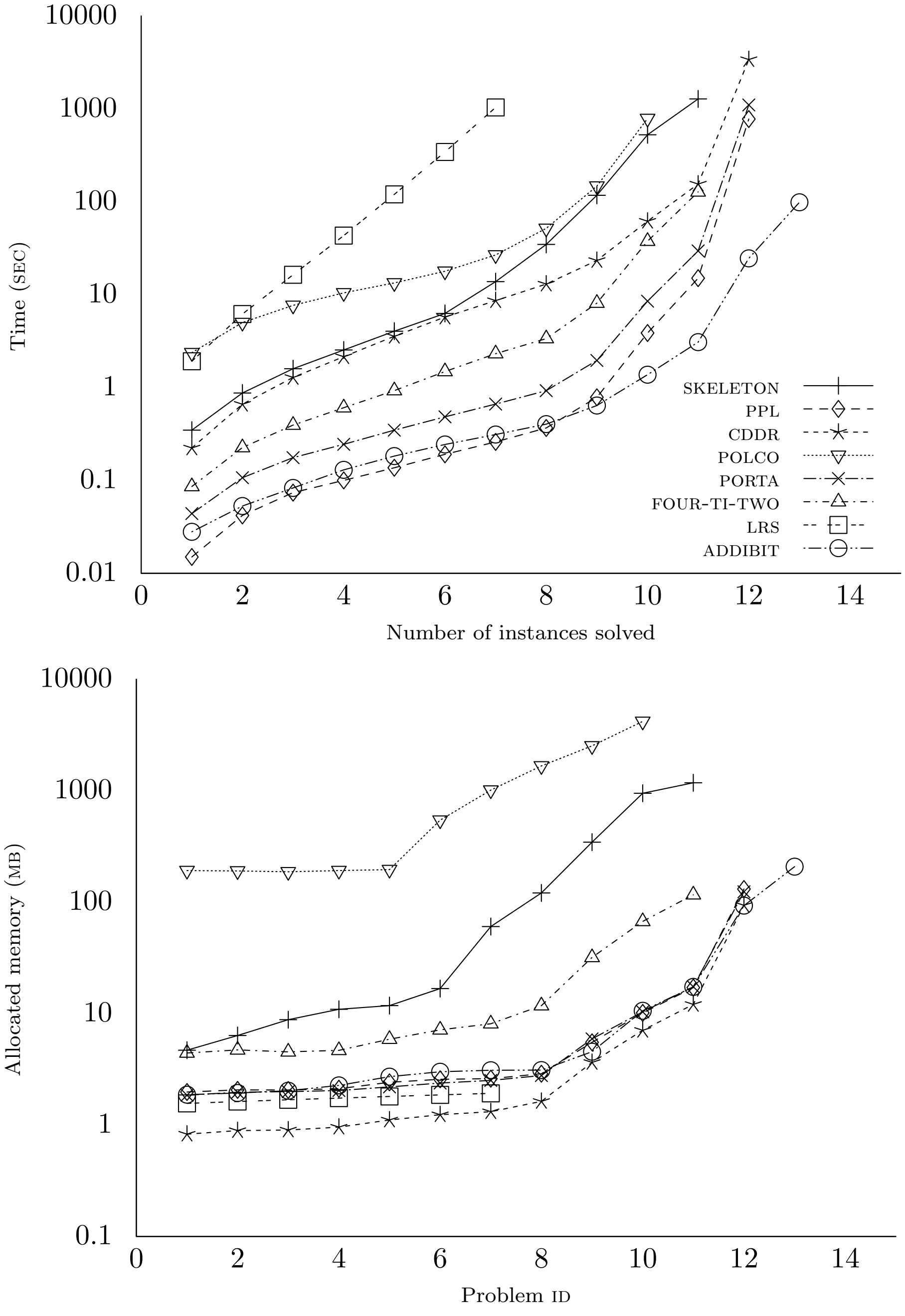
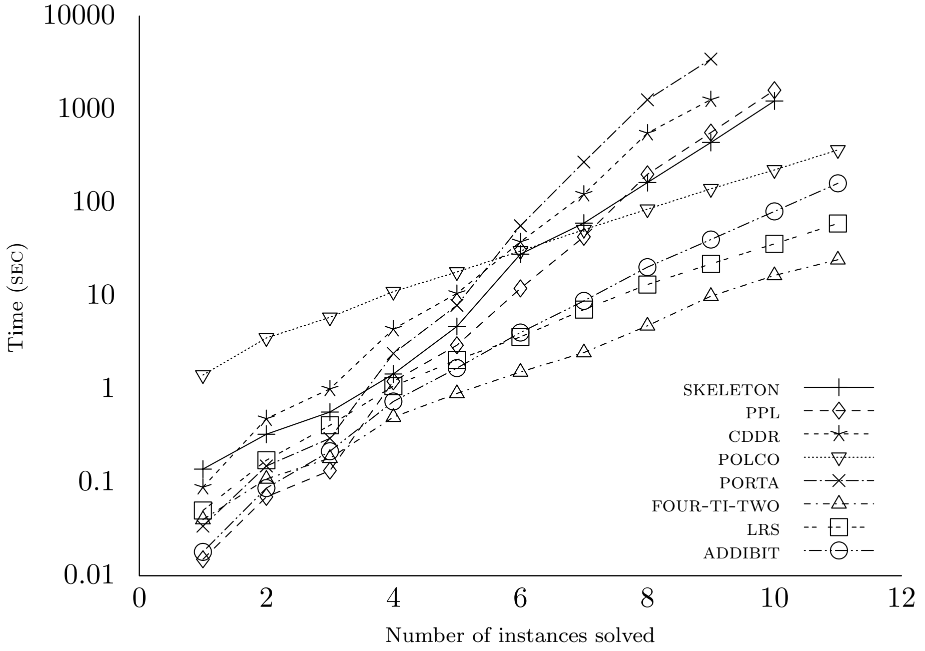


Figure 8.5: Computational results for cut polytopes



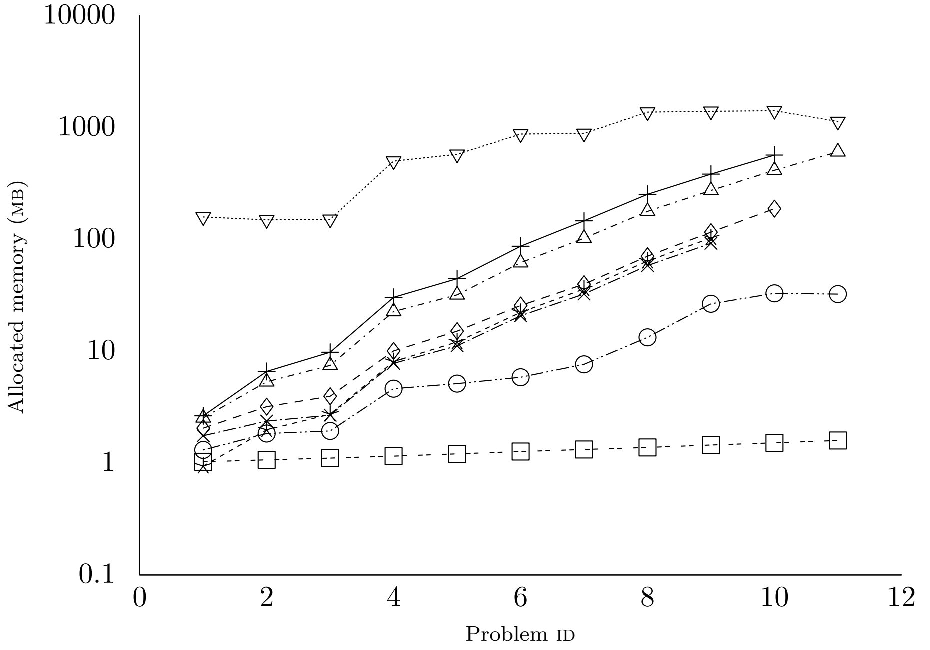


Figure 8.6: Computational results for products of hypercubes and 0/1 polytopes

9 Conclusions

The convex hull problem has been a field of intensive research for many years. The countless theoretical contributions have been accompanied by a reasonable amount of practical work; still, both theory and practice provide open issues. The lack of an optimal double description implementation is one such issue arising from the practical application of the algorithm. While, by itself, this work did not completely fill that gap, it resulted in four noteworthy achievements related to adjacency testing and parallel computation.

First, multiple data structures suitable for application within the adjacency testing process were introduced. In addition, several heuristics were proposed to improve the performance of bit pattern trees, which have been the best solution so far. All data structures were then compared through practical experiments involving various degenerate problems. As a result, the optimized bit pattern trees were superior to all other investigated data structures. The combination of simplicity and efficacy which they embody resulted in excellent performance. Thus, for general problems, they can be viewed as the best candidate to obtain fast computations. Nevertheless, problems were found to show that their superiority is not absolute. In the presented survey, only data structures based on binary search trees have been considered. In that regard, it would be interesting to see whether *quad trees* [72] or other tree-based data structures can be successfully adapted for adjacency testing as well.

Second, a new improved algebraic test was introduced. Its correctness was formally proven. Its competitiveness with the currently best solution, the combinatorial test, was demonstrated in a series of experiments. Problems with moderate degeneracy and problems with small numbers of constraints were identified as a potential domain of superiority for that new adjacency test. It should be noted that it is a more complicated solution than the currently existing ones; thus, considerable potential for further theoretical and practical improvements exists. The internal structure of the global cache, for example, is one important aspect which deserves further attention. It has a considerable impact on the overall performance as cached data is retrieved during the majority of adjacency tests. Furthermore, the current implementation of the new adjacency test does not utilize as many technical optimizations as the one embodying the combinatorial test. In that respect, the presented computational results are certainly improvable.

Third, one particular divide-and-conquer technique related to the parallelization of the double description method was revisited. It was shown that for a class of cut polytopes, the parallel computation is considerably slower than the sequential one due to a serious computational explosion. This introduced one important question. Is there a row ordering of the input matrix such that divide-and-conquer works for cut polytopes? A potentially positive answer, however, may still not solve the practical aspect of the problem. A row ordering which is not competitive with the lexicographic one examined here is of little help in practice. As an alternative, it was demonstrated that the computational explosion could be bypassed by splitting the computation at a later stage and thus employing only a partial parallelization.

Finally, empirical evidence was provided that the currently existing double description implementations are yet to reach optimum performance. A significant performance gain was achieved by employing modern instruction sets and ideas related to *cache-oblivious* data structures [37, 4, 22]. It is likely that this performance can be further improved, albeit not by such a large margin. The recently introduced GNU Multiprecision Library version 6, for instance, promises a better support for the latest processor generations and is certainly worth a try in that respect. Furthermore, branch prediction can potentially speed up the tree traversals and needs to be examined more thoroughly as well.

References

[1] 4ti2 team (2014). 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces, version 1.6.2. available at `http://www.4ti2.de` (last visit: 01 May 2014).

[2] Abdullahi, S. D., Dyer, M. E., and Proll, L. G. (2003). Listing vertices of simple polyhedra associated with dual li(2) systems. In *Discrete Mathematics and Theoretical Computer Science*, volume 2731 of *Lecture Notes in Computer Science*, pages 89–96. Springer.

[3] Altherr, W. (1975). An algorithm for enumerating all vertices of a convex polyhedron. *Computing*, 15(3):181–193.

[4] Arge, L., Brodal, G., and Fagerberg, R. (2005). Cache-oblivious data structures. *Handbook of Data Structures and Applications*, 27.

[5] Avis, D. (1994). A c implementation of the reverse search vertex enumeration algorithm. Technical report, McGill University of Technology, Montreal, Canada.

[6] Avis, D. (1998). Computational experience with the reverse search vertex enumeration algorithm. *Optimization Methods and Software*, 10:107–124.

[7] Avis, D. (2000a). Living with lrs. In *Japan Conference on Discrete and Computational Geometry*, volume 1763 of *Lecture Notes in Computer Science*, pages 47–56. Springer.

[8] Avis, D. (2000b). lrs: A revised implementation of the reverse search vertex enumeration algorithm. In Kalai, G. and Ziegler, G. M., editors, *Polytopes - Combinatorics and Computation*, DMV Seminar Band 29, pages 177–198. Birkhauser, Basel, Switzerland.

[9] Avis, D. (2013). lrslib: Implementation of the reverse search method, version 5.0. available at `http://cgm.cs.mcgill.ca/~avis/C/lrs.html` (last visit: 01 May 2014).

[10] Avis, D. and Bremner, D. (1995). How good are convex hull algorithms? In *Proceedings of the 11th annual symposium on Computational geometry*, SCG '95, pages 20–28. ACM Press.

[11] Avis, D., Bremner, D., and Seidel, R. (1997). How good are convex hull algorithms? *Computational Geometry: Theory and Applications*, 7:265–301.

[12] Avis, D. and Fukuda, K. (1991a). A basis enumeration algorithm for linear systems with geometric applications. *Applied Mathematics Letters*, 4(5):39–42.

[13] Avis, D. and Fukuda, K. (1991b). A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In *Proceedings of the 7th annual symposium on Computational geometry*, SCG '91, pages 98–104. ACM Press.

[14] Avis, D. and Fukuda, K. (1992). A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, 8(3):295–313.

[15] Bagnara, R., Hill, P. M., and Zaffanella, E. (2008). The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72:3–21.

[16] Bagnara, R., Hill, P. M., and Zaffanella, E. (2009). Applications of polyhedral computations to the analysis and verification of hardware and software systems. *Theoretical Computer Science*, 410(46):4672–4691.

[17] Balinski, M. L. (1961). An algorithm for finding all vertices of convex polyhedral sets. *Journal of the Society for Industrial and Applied Mathematics*, 9(1):72–88.

[18] Barahona, F. and Mahjoub, A. (1986). On the cut polytope. *Mathematical Programming*, 36:157–173.

[19] Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483.

[20] Barrett, C., Sebastiani, R., Seshia, S., and Tinelli, C. (2009). Satisfiability modulo theories. In Biere, A., Heule, M. J. H., van Maaren, H., and Walsh, T., editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press.

[21] Bell, S. L. and Palsson, B. O. (2005). expa: a program for calculating extreme pathways in biochemical reaction networks. *Bioinformatics*, 21(8):1739–1740.

[22] Bender, M., Demaine, E., and Farach-Colton, M. (2005). Cache-oblivious b-trees. *SIAM Journal on Computing*, 35(2):341–358.

[23] Bentley, J. L. (1975a). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

[24] Bentley, J. L. (1975b). A survey of techniques for fixed radius near neighbor searching. Technical report, Stanford University, California, USA.

[25] Bentley, J. L. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229.

[26] Birkhoff, G. (1984). *Lattice Theory.* American Mathematical Society colloquium publications. American Mathematical Society.

[27] Bland, R. G. (1977). New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107.

[28] Boissonnat, J.-D., Devillers, O., Schott, R., Teillaud, M., and Yvinec, M. (1992). Applications of random sampling to on-line algorithms in computational geometry. *Discrete & Computational Geometry*, 8(1):51–71.

[29] Borgwardt, K. (1997). Average complexity of a gift-wrapping algorithm for determining the convex hull of randomly given points. *Discrete & Computational Geometry*, 17(1):79–109.

[30] Borgwardt, K. H. (2007). Average-case analysis of the double description method and the beneath-beyond algorithm. *Discrete & Computational Geometry*, 37(2):175–204.

[31] Borodin, A., Ostrovsky, R., and Rabani, Y. (1999). Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the 31st annual ACM symposium on Theory of computing*, STOC '99, pages 312–321. ACM Press.

[32] Boros, E., Elbassioni, K., Gurvich, V., and Tiwary, H. (2011). The negative cycles polyhedron and hardness of checking some polyhedral properties. *Annals of Operations Research*, 188(1):63–76.

[33] Bremner, D. (1996). Incremental convex hull algorithms are not output sensitive. In *Proceedings of the 7th International Symposium on Algorithms and Computation*, ISAAC '96, pages 26–35. Springer.

[34] Bremner, D. (1997). *On the complexity of vertex and facet enumeration for convex polytopes.* PhD thesis, McGill University, Montreal, Canada.

[35] Bremner, D. (1999). Incremental convex hull algorithms are not output sensitive. *Discrete & Computational Geometry*, 21(1):57–68.

[36] Bremner, D., Fukuda, K., and Marzetta, A. (1998). Primal-dual methods for vertex and facet enumeration. *Discrete & Computational Geometry*, 20:333–357.

[37] Brodal, G. S., Fagerberg, R., and Jacob, R. (2002). Cache oblivious search trees via binary trees of small height. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 39–48. Society for Industrial and Applied Mathematics.

[38] Brönnimann, H. (1998). Degenerate convex hulls on-line in any fixed dimension. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 249–258. ACM Press.

[39] Brönnimann, H. (1999). Degenerate convex hulls on-line in any fixed dimension. *Discrete & Computational Geometry*, 22(4):527–545.

[40] Brüngger, A., Marzetta, A., Fukuda, K., and Nievergelt, J. (1999). The parallel search bench ZRAM and its applications. *Annals of Operations Research*, 90(0):45–63.

[41] Burdet, C.-A. (1974). Generating all the faces of a polyhedron. *SIAM Journal on Applied Mathematics*, 26(3):479–489.

[42] Burnikel, C., Mehlhorn, K., and Schirra, S. (1994). On degeneracy in geometric computations. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, pages 16–23. Society for Industrial and Applied Mathematics.

[43] Burton, B. A. (2012). Complementary vertices and adjacency testing in polytopes. In *Proceedings of the 18th International Computing and Combinatorics Conference*, COCOON '12, pages 507–518. Springer.

[44] Bussieck, M. R. and Lübbecke, M. E. (1998). The vertex set of a 0/1-polytope is strongly p-enumerable. *Computational Geometry: Theory and Applications*, 11(2):103–109.

[45] Chand, D. R. and Kapur, S. S. (1970). An algorithm for convex polytopes. *Journal of the ACM*, 17(1):78–86.

[46] Charnes, A. (1952). Optimality and degeneracy in linear programming. *Econometrica*, 20(2):160–170.

[47] Chazelle, B. (1993). An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409.

[48] Chernikova, N. V. (1964). Algorithm for finding a general formula for the non-negative solutions of a system of linear equations. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 4(4):151–158.

[49] Chernikova, N. V. (1965). Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5:228–233.

[50] Christof, T. and Löbel, A. (2009). PORTA - POlyhedron Representation Transformation Algorithm, version 1.4.1. available at `http://typo.zib.de/opt-long_projects/Software/Porta/` (last visit: 01 May 2014).

[51] Clarkson, K. and Shor, P. (1989). Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(1):387–421.

[52] Clarkson, K. L., Mehlhorn, K., and Seidel, R. (1992). Four results on randomized incremental constructions. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '92, pages 463–474. Springer.

[53] Clarkson, K. L., Mehlhorn, K., and Seidel, R. (1993). Four results on randomized incremental constructions. *Computational Geometry: Theory and Applications*, 3(4):185–212.

[54] Clarkson, K. L. and Shor, P. W. (1988). Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *Proceedings of the 4th annual symposium on Computational geometry*, SCG '88, pages 12–17. ACM Press.

[55] Cousot, P. and Cousot, R. (1976). Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming*, pages 106–130. Dunod.

[56] Cousot, P. and Cousot, R. (1977). Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, pages 238–252. ACM Press.

[57] Cousot, P. and Halbwachs, N. (1978). Automatic discovery of linear restraints among variables of a program. In *Conference Record of the 5th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97. ACM Press.

[58] Dantzig, G. B. (1963). *Linear programming and extensions.* Princeton University Press.

[59] Davey, B. and Priestley, H. (2002). *Introduction to Lattices and Order.* Cambridge mathematical text books. Cambridge University Press.

[60] Devadoss, S. and O'Rourke, J. (2011). *Discrete and Computational Geometry.* Princeton University Press.

[61] Deza, A., Fukuda, K., Pasechnik, D., and Sato, M. (2001). On the skeleton of the metric polytope. In *Japan Conference on Discrete and Computational Geometry*, volume 2098 of *Lecture Notes in Computer Science*, pages 125–136. Springer.

[62] Dyer, M. and Proll, L. (1977a). An algorithm for determining all extreme points of a convex polytope. *Mathematical Programming*, 12(1):81–96.

[63] Dyer, M. and Proll, L. (1977b). Vertex enumeration in convex polyhedra - a comparative computational study. In *Proceedings of the CP77 Combinatorial Programming Conference*, pages 23–43.

[64] Dyer, M. and Proll, L. (1982). An improved vertex enumeration algorithm. *European Journal of Operational Research*, 9(4):359–368.

[65] Edelsbrunner, H. (1987). *Algorithms in combinatorial geometry.* Springer, New York.

[66] Edelsbrunner, H. and Mücke, E. P. (1990). Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104.

[67] Emiris, I. and Canny, J. (1992). An efficient approach to removing geometric degeneracies. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, SCG '92, pages 74–82. ACM Press.

[68] Emiris, I. Z., Canny, J. F., and Seidel, R. (1997). Efficient perturbations for handling geometric degeneracies. *Algorithmica*, 19(1-2):219–242.

[69] Fang, X. G. and Havas, G. (1997). On the worst-case complexity of integer gaussian elimination. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, ISSAC '97, pages 28–31. ACM Press.

[70] Fernández, F. and Quinton, P. (1988). *Extension of Chernikova's Algorithm for Solving General Mixed Linear Programming Problems*. Rapports de recherche. Institut National de Recherche en Informatique et en Automatique.

[71] Fieldhouse, M. (1961). *Linear Programming*. PhD thesis, Cambridge University, England.

[72] Finkel, R. and Bentley, J. (1974). Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9.

[73] Flajolet, P. and Puech, C. (1986). Partial match retrieval of multidimensional data. *Journal of the ACM*, 33(2):371–407.

[74] Frumkin, M. (1977). Polynomial time algorithms in the theory of linear diophantine equations. In *Fundamentals of Computation Theory*, volume 56 of *Lecture Notes in Computer Science*, pages 386–392. Springer, Berlin/Heidelberg.

[75] Fukuda, K. (2007). cdd+: Implementation of the double description method, version 0.77a. available at `http://www.inf.ethz.ch/personal/fukudak/cdd_home` (last visit: 01 May 2014).

[76] Fukuda, K., Liebling, T. M., and Margot, F. (1997). Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron. *Computational Geometry: Theory and Applications*, 8:1–12.

[77] Fukuda, K. and Prodon, A. (1996). Double description method revisited. In *Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer, Berlin/Heidelberg.

[78] Fukuda, K. and Rosta, V. (1994). Combinatorial face enumeration in convex polytopes. *Computational Geometry: Theory and Applications*, 4(4):191–198.

[79] Gagneur, J. and Klamt, S. (2004). Computation of elementary modes: a unifying framework and the new binary approach. *BMC Bioinformatics*, 5:175.

[80] Gal, T. (1993). Selected bibliography on degeneracy. *Annals of Operations Research*, 46-47(1):1–7.

[81] Gale, D. (1964). On the number of faces of a convex polytope. *Canadian Journal of Mathematics*, 16:12–17.

[82] Garg, N. and Vazirani, V. V. (1995). A polyhedron with all s-t cuts as vertices, and adjacency of cuts. *Mathematical Programming*, pages 17–25.

[83] Genov, B. (2013). Data structures for incremental extreme ray enumeration algorithms. In *Proceedings of the 25th Canadian Conference on Computational Geometry*, CCCG '13. Carleton University, Ottawa, Canada.

[84] Granlund, T. (2012). The GNU Multiple Precision Arithmetic Library, version 5. available at `http://gmplib.org/` (last visit: 01 May 2014).

[85] Hadley, G. (1962). *Linear Programming.* Addison-Wesley, Reading, Massachusetts.

[86] Hansen, P. B. (1993). Model programs for computational science: A programming methodology for multicomputers. *Concurrency - Practice and Experience*, 5(5):407–423.

[87] Hemmecke, R. (2002). On the computation of hilbert bases of cones. *Mathematical Software, ICMS*, pages 307–317.

[88] Henk, M., Richter-Gebert, J., and Ziegler, G. M. (2004). *Basic Properties of Convex Polytopes*, chapter 16, pages 355–382. Chapman and Hall/CRC, second edition.

[89] Horst, R. and Hoang, T. (1993). *Global optimization: deterministic approaches.* Springer.

[90] Ibarra, O. H., Moran, S., and Hui, R. (1982). A generalization of the fast lup matrix decomposition algorithm and applications. *Journal of Algorithms*, 3:45–56.

[91] Jeannet, B. and Miné, A. (2009). Apron: A library of numerical abstract domains for static analysis. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 661–667. Springer.

[92] Jevremović, D., Boley, D., and Sosa, C. P. (2011a). Divide-and-conquer approach to the parallel computation of elementary flux modes in metabolic networks. In *Proceedings of the 25th International Parallel & Distributed Processing Symposium*, IPDPS '11, pages 502–511. IEEE Computer Society.

[93] Jevremović, D., Trinh, C. T., Srienc, F., Sosa, C. P., and Boley, D. (2011b). Parallelization of nullspace algorithm for the computation of metabolic pathways. *Parallel Computing*, 37(6-7):261–278.

[94] Joswig, M. (2003). Beneath-and-beyond revisited. In *Algebra, Geometry and Software Systems*, pages 1–21. Springer.

[95] Joswig, M. and Ziegler, G. (2004). Convex hulls, oracles, and homology. *Journal of Symbolic Computation*, 38(4):1247–1259.

[96] Kallay, M. (1981). Convex hull algorithms in higher dimensions. unpublished manuscript.

[97] Kannan, R. and Bachem, A. (1979). Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *SIAM Journal of Computing*, 8(4):499–507.

[98] Khachiyan, L., Boros, E., Borys, K., Elbassioni, K. M., and Gurvich, V. (2008). Generating all vertices of a polyhedron is hard. *Discrete & Computational Geometry*, 39(1-3):174–190.

[99] Khang, D. B. and Fujiwara, O. (1989). A new algorithm to find all vertices of a polytope. *Operations Research Letters*, 8(5):261–264.

[100] Klamt, S., Gagneur, J., and von Kamp, A. (2005). Algorithmic approaches for computing elementary modes in large biochemical reaction networks. *Systems Biology, IEE Proceedings*, 152(4):249–255.

[101] Klee, V. (1964). The number of vertices of a convex polytope. *Canadian Journal of Mathematics*, 16:701–720.

[102] Klee, V. (1974). Polytope pairs and their relationship to linear programming. *Acta Mathematica*, 133(1):1–25.

[103] Lee, L.-Q., Varner, J., and Ko, K. (2004). Parallel extreme pathway computation for metabolic networks. In *Proceedings of the 3rd International Conference on Computational Systems Bioinformatics*, CSB '04, pages 636–639. IEEE Computer Society.

[104] Mani, P. and Bruggesser, H. (1971). Shellable decompositions of cells and spheres. *Mathematica Scandinavica*, 29:197–205.

[105] Marzetta, A. (1998). *ZRAM: A Library of Parallel Search Algorithms and Its Use in Enumeration and Combinatorial Optimization.* PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland.

[106] Matoušek, J. (1993). Linear optimization queries. *Journal of Algorithms*, 14(3):432–448.

[107] Matoušek, J. (2002). *Lectures on Discrete Geometry.* Springer, New York.

[108] Mattheiss, T. and Schmidt, B. K. (1980). Computational results on an algorithm for finding all vertices of a polytope. *Mathematical Programming*, 18(1):308–329.

[109] Mattheiss, T. H. (1973). An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities. *Operations Research*, 21(1):247–260.

[110] Mattheiss, T. H. and Rubin, D. S. (1980). A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Mathematics of Operations Research*, 5(2):167–185.

[111] Maňas, M. and Nedoma, J. (1968). Finding all vertices of a convex polyhedron. *Numerische Mathematik*, 12(3):226–229.

[112] McMullen, P. (1970). The maximum numbers of faces of a convex polytope. *Mathematika*, 17:179–184.

[113] McMullen, P. and Shephard, G. (1971). *Convex Polytopes and the Upper Bound Conjecture.* Cambridge University Press.

[114] McRae, W. B. and Davidson, E. R. (1973). An algorithm for the extreme rays of a pointed convex polyhedral cone. *SIAM Journal of Computing*, 2(4):281–293.

[115] Motzkin, T. S. (1957). Comonotone curves and polyhedra.

[116] Motzkin, T. S., Raiffa, H., Thompson, G. L., and Thrall, R. M. (1953). *The double description method*, volume II, pages 51–73. Princeton University Press.

[117] Mulmuley, K. (1999). Randomized algorithms in computational geometry. In Sack, J. R. and Urrutia, J., editors, *Handbook of Computational Geometry.* Elsevier Science.

[118] Murty, K. G. (2009). A problem in enumerating extreme points, and an efficient algorithm for one class of polytopes. *Optimization Letters*, 3(2):211–237.

[119] Murty, K. G. and Chung, S.-J. (1995). Segments in enumerating faces. *Mathematical Programming*, 70(1–3):27–45.

[120] Oxley, J. G. (1992). *Matroid theory.* Oxford University Press.

[121] Peleska, J., Vorobev, E., and Lapschies, F. (2011). Automated test case generation with smt-solving and abstract interpretation. In *Proceedings of the 3rd international conference on NASA Formal methods*, NFM '11, pages 298–312. Springer.

[122] Pernet, C. and Stein, W. (2010). Fast computation of hermite normal forms of random integer matrices. *Journal of Number Theory*, 130(7):1675–1683.

[123] Posthoff, C. and Steinbach, B. (2004). *Logic functions and equations: Binary models for computer science.* Springer, Dordrecht, The Netherlands.

[124] Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction.* Springer, New York.

[125] Provan, J. (1994). Efficient enumeration of the vertices of polyhedra associated with network LP's. *Mathematical Programming*, 63(1-3):47–64.

[126] Ramanathan, R. M. (2006). Extending the world's most popular processor architecture. Technical report, Intel Corporation. White Paper.

[127] Rivest, R. (1976). Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50.

[128] Rote, G. (1992). Degenerate convex hulls in high dimensions without extra storage. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, SCG '92, pages 26–32. ACM Press.

[129] Sahni, S. and Vairaktarakis, G. (1996). The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization*, 9(3-4):357–377.

[130] Samatova, N. F., Geist, A., Ostrouchov, G., and Melechko, A. V. (2002). Parallel out-of-core algorithm for genome-scale enumeration of metabolic systemic pathways. In *Proceedings of the 16th International Parallel & Distributed Processing Symposium*, IPDPS '02, pages 249–264. IEEE Computer Society.

[131] Schrijver, A. (1986). *Theory of linear and integer programming.* John Wiley & Sons, Inc.

[132] Schuster, S. and Hilgetag, C. (1994). On elementary flux modes in biochemical reaction systems at steady state. *Journal of Biological Systems*, 2:165–182.

[133] Seidel, R. (1981). A convex hull algorithm optimal for point sets in even dimensions. Technical report, University of British Columbia, Vancouver, Canada.

[134] Seidel, R. (1986a). Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 404–413. ACM Press.

[135] Seidel, R. (1986b). Output-size sensitive algorithms for constructive problems in computational geometry. Technical report, Cornell University, Department of Computer Science, Ithaca, USA.

[136] Seidel, R. (1991). Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(1):423–434.

[137] Seidel, R. (1995). The upper bound theorem for polytopes: An easy proof of its asymptotic version. *Computational Geometry: Theory and Applications*, 5(2):115–116.

[138] Seidel, R. (1998). The nature and meaning of perturbations in geometric computing. *Discrete & Computational Geometry*, 19(1):1–17.

[139] Storjohann, A. (1998). Computing hermite and smith normal forms of triangular integer matrices. *Linear Algebra and its Applications*, 282(1–3):25–45.

[140] Swart, G. (1985). Finding the convex hull facet by facet. *J. Algorithms*, 6(1):17–48.

[141] Terzer, M. (2009a). *Large scale methods to enumerate extreme rays and elementary modes.* PhD thesis, ETH, Zurich, Switzerland.

[142] Terzer, M. (2009b). polco (polyhedral computations), version 4.7.1. available at `http://www.csb.ethz.ch/tools/polco` (last visit: 01 May 2014).

[143] Terzer, M. and Stelling, J. (2006). Accelerating the computation of elementary modes using pattern trees. In *Proceedings of the 6th international conference on Algorithms in Bioinformatics*, WABI '06, pages 333–343. Springer.

[144] Terzer, M. and Stelling, J. (2008). Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24:2229–2235.

[145] Terzer, M. and Stelling, J. (2010). Parallel extreme ray and pathway computation. In *Parallel Processing and Applied Mathematics*, volume 6068 of *Lecture Notes in Computer Science*, pages 300–309. Springer.

[146] Tijssen, G. A. and Sierksma, G. (1995). Balinski-Tucker simplex tableaus : dimensions, degeneracy degrees, and interior points of optimal faces. Technical Report 95A15, University of Groningen, Research Institute SOM (Systems, Organisations and Management).

[147] Uhlmann, J. K. (1991). Metric trees. *Applied Mathematics Letters*, 4(5):61–62.

[148] Vanderbei, R. (1996). *Linear Programming: Foundations and Extensions.* International Series in Operations Research & Management Science. Springer.

[149] Verge, H. L. (1992). A note on Chernikova's algorithm. Technical Report 635, IRISA, Rennes, France.

[150] von Kamp, A. and Schuster, S. (2006). Metatool 5.0: fast and flexible elementary modes analysis. *Bioinformatics*, 22(15):1930–1931.

[151] Wagner, C. (2004). Nullspace approach to determine the elementary modes of chemical reaction systems. *The Journal of Physical Chemistry B*, 108(7):2425–2431.

[152] Whitney, H. (1935). On the abstract properties of linear dependence. *American Journal of Mathematics*, 57:509–533.

[153] Williams, V. V. (2012). Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 887–898. ACM Press.

[154] Yap, C.-K. (1990). Symbolic treatment of geometric degeneracies. *Journal of Symbolic Computation*, 10(3–4):349–370.

[155] Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 311–321. Society for Industrial and Applied Mathematics.

[156] Ziegler, G. (1995). *Lectures on Polytopes.* Graduate texts in mathematics. Springer.

[157] Zolotykh, N. (2012). New modification of the double description method for constructing the skeleton of a polyhedral cone. *Computational Mathematics and Mathematical Physics*, 52:146–156.