Université catholique de Louvain
Faculté des Sciences Appliquées

Computing Markov bases, Gröbner bases, and extreme rays

Thèse présentée en vue de l'obtention du grade de
Docteur en Sciences Appliquées par

Peter N. MALKIN

Promoteur:	L.A. Wolsey
Jury:	V. Blondel (Président)
	K. Aardal
	R. Hemmecke
	Y. Nesterov
	Y. Pochet

Acknowledgements

There are many people I would like that thank who have supported me during my doctorate.

Firstly, I would like to thank my supervisor Laurence Wolsey. He guided me and gave me the freedom to explore my interests. I especially appreciate his patience in deciphering the many drafts of this thesis.

I feel especially indebted to Raymond Hemmecke for his encouragement and advice. The time I have spent working with him was some of the most productive and inspiring time during my doctorate.

It has been a privilege for me to do my doctorate at CORE. The environment at CORE is extremely friendly and conducive to research. I would like to thank everyone at CORE who together make it such a unique place.

As any international student or visitor to CORE will tell you, the support given by the administrative staff at CORE is invaluable. Only with their help have I navigated through the vagaries of the Belgian administrative system. So, I give many thanks to Mady, Catherine, Sheila, Fabienne, and Sylvie.

During my doctorate, I was fortunate enough to spend several months in Magdeburg with Robert Weismantel, Raymond Hemmecke, Matthias Köppe, and Utz-Uwe Haus. My time there proved extremely productive. I am very grateful for having had this opportunity to work with them.

I would also like to thank the other members of my doctoral jury Karen Aardal, Vincent Blondel (president), Yurii Nesterov, and Yves Pochet for their advice and useful comments.

I gratefully acknowledge the support of the Marie Curie Research Training Network ADONET 504438 and the support of the Université catholique de Louvain.

I particularly appreciate the friendship of François who paradoxically both drove me mad and kept me sane throughout my doctorate. I would also like to express my appreciation for the friendship of the following people during my doctorate: Jeremy, Giles, Virginia, Ruslan, Hamish, Helena, Andreas, Malika, Fabienne, Maziar, Quentin, Michel, Robert, Sylvain, Gilles, Diego, Joao, Kent, Ayse, the people from UKRO, and Estelle's family.

I would like to thank my parents and my brother for supporting me throughout my thesis. Finally, my fiancée Estelle has especially been an enormous source of support and compassion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we address problems from two topics of applied mathematics: *linear integer programming* and *polyhedral computation*.

Linear integer programming concerns solving optimisation problems to maximise or minimise a linear cost function over the set of integer points in a polyhedron. The framework of integer programming is remarkably flexible and expressive, and a wide variety of problems can be modelled as integer programs. Applications of integer programming include production scheduling, facility location, and VLSI circuit design. There is a plethora of techniques and variations thereof for solving integer programs far too numerous to mention. Standard techniques have proven to be extreme effective in practice despite the fact that integer programming is NP-hard. In this thesis, we explore the theory and computation of Gröbner bases and Markov bases for integer programming.

Markov basis methods concern the set of feasible solutions of an integer program. A Markov basis is a set of integer vectors such that we can move between any two feasible solutions by adding or subtracting a vector in the Markov basis while staying feasible – we never move outside the set of feasible solutions. The major contribution of this thesis is a fast algorithm for computing Markov bases. A Markov basis has two main applications: sampling from a set of feasible solutions and computing Gröbner bases of integer programs. Sampling from a set of feasible solutions is used in algebraic statistics to determine whether a set of observed events follow a given frequency distribution. Using the fast algorithm, we were able to solve a previously intractable computational challenge in algebraic statistics for computing Markov bases. Also, there is another potential application of Markov bases in local search heuristics. Other problems have also been successfully tackled using this algorithm.

Gröbner basis methods for integer programming are iterative improvement techniques for solving an integer program. A Gröbner basis is a set of vectors such that we can improve any given non-optimal feasible solution of an integer programming by subtracting a vector in the Gröbner basis. So, using a Gröbner basis, we can solve an integer program by iteratively improving a given initial feasible solution. The process of constructing a Gröbner basis and then solving an integer program using the Gröbner basis is called the Gröbner basis approach. Gröbner basis methods are relatively new compared to standard methods for solving integer programs.

The worst case complexity of Gröbner basis methods is not fully known but it is at least exponential as is expected since integer programming is NP-hard. One of the reasons we are interested in Gröbner basis techniques from a theoretical viewpoint is that they are *exact* local search approaches for integer programs meaning that local optimality in some neighbourhood of a feasible solution implies global optimality, which is an interesting concept in itself to study. Also, despite the effectiveness of standard techniques, there still remain some classes of problems for which standard techniques are ineffective, so we explore Gröbner basis methods in the search for another method that can either provide some insight into what makes these classes of problems difficult, or actually allow us to solve these difficult problems. In this thesis, we extend and improve Gröbner basis methods. Previous Gröbner basis methods effectively solved for many different integer programs simultaneously, so problem structure of a particular integer program could not be used. We present a new Gröbner basis approach that uses structure of a particular integer program in order to solve the integer program more efficiently.

Polyhedral computation is concerned with algorithms for computing different properties of *convex polyhedra*. A convex polyhedron is a set of points that satisfy a finite set of linear constraints. Convex polyhedra appear in many different areas of mathematics such as linear programming, integer programming, combinatorial optimisation, computational geometry, and algebraic geometry. Computing the properties of convex polyhedra is interesting in itself, and it is also useful for solving problems within the above areas of mathematics. For example, in integer programming, polyhedral computation is used to analyse the structure of the convex hulls of the set of feasible solutions of an integer program. In this thesis, we investigate ways of improving existing methods for computing certain properties of convex polyhedra. More explicitly, we investigate and improve an algorithm for converting between different representations of cones and polyhedra; that is, we convert between a generator representation of a cone or polyhedron and a constraint representation of the cone or polyhedron. This algorithm can be extended to compute circuits of matrices, which are used in computational biology for metabolic pathway analysis.

In the rest of this introduction, we informally introduce and discuss these concepts. We will define these concepts in a more formal framework in the rest of this thesis.

1.1 Polyhedral cones

In this section, we introduce polyhedral cones. They play an important role in polyhedral computation.

The expression $\lambda^1 x^1 + \lambda^2 x^2 + ... + \lambda^k x^k$ where $\lambda^1, \lambda^2, ..., \lambda^k \in \mathbb{R}_+$ is called a *conic combination* of $x^1, x^2, ..., x^k \in \mathbb{R}^n$. A *cone* is a set $\mathcal{C} \subseteq \mathbb{R}^n$ that is closed under *conic* combinations. More explicitly, if $x^1, x^2, ..., x^k \in \mathcal{C}$, then $\lambda^1 x^1 + \lambda^2 x^2 + ... + \lambda^k x^k \in \mathcal{C}$ for all $\lambda^1, \lambda^2, ..., \lambda^k \in \mathbb{R}_+$.

A set of points in $\mathbb{R}^n$ that satisfy a finite set of linear homogeneous constraints is a cone, called a *finitely constrained* cone; that is, the set $\mathcal{C}(A) := \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$ where $A \in \mathbb{R}^{m \times n}$ is a finitely constrained cone. For example, the following set of

points in $\mathbb{R}^n$ is a cone: $\mathcal{C} = \{x \in \mathbb{R}^2 : 3x_1 - 2x_2 \geq 0, -x_1 + 2x_2 \geq 0\}$. It is depicted in Figure 1.1. Note that the cone in the diagram extends forever in the direction of the vectors in the diagram and that the cone includes all the points in between the vectors as well on the vectors themselves.

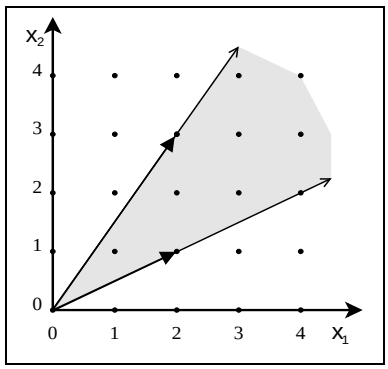

Figure 1.1: A simple polyhedral cone in two dimensions.

A set of points in $\mathbb{R}^n$ generated by all the possible conic combinations of a finite set of points in $\mathbb{R}^n$ is a cone, called a *finitely generated* cone; that is, the set $\{\lambda B : \lambda \in \mathbb{R}^k_+\}$ where $B \in \mathbb{R}^{k \times n}$ is a finitely generated cone. Here, the set $\{\lambda B : \lambda \in \mathbb{R}^k_+\}$ is all possible conic combinations of the rows of the matrix B, and we call the rows of B the generators of the cone. We will see later (Section 2.2) that if the cone contains no lines that go to infinity in both directions, then there is a unique (up to positive scaling) minimal set of generators of any finitely generated cone – a set of generators is minimal if we cannot remove a generator from the set without changing the cone that is generated. If there are no lines in a cone, then we say that the cone is *pointed*. Geometrically speaking, a pointed cone has edges, and any vector that lies along an edge of the cone is called an *extreme ray* of the cone. We will see that every cone has a finite number of distinct extreme rays (i.e. extreme rays that are not positive scalar multiples of each other) and that every pointed cone is generated by the set of its extreme rays; moreover, the set of extreme rays is a minimal generating set of the cone.

The cone $\mathcal{C}$ in Figure 1.1 can also be represented as a conic combination of a set of points: $\mathcal{C} = \{\lambda_1(2,1) + \lambda_2(2,3) : \lambda_1, \lambda_2 \in \mathbb{R}_+\}$. This set of two generators $\{(2,1),(2,3)\}$ is minimal, meaning that the cone $\mathcal{C}$ cannot be generated by only one of the two vectors. Note that the cone is pointed since there are no lines contained within the cone. We have drawn these two vectors in Figure 1.1. Observe that these two vectors lie on the edge of the cone, so these two vectors are extreme rays of $\mathcal{C}$.

A fundamental theorem in polyhedral theory is that a cone is finitely constrained if and only if it is finitely generated. We saw this with our small example cone of Figure 1.1. We represented it is as both a finitely constrained cone and a finitely generated cone. This theorem in the forward direction (only if) is most commonly known as *Minkowski's* theorem, and in the backward direction (if), the theorem is most commonly known as *Weyl's* theorem. More explicitly, Minkowski's theorem says that, for any finitely constrained cone $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq 0\}$ where $A \in \mathbb{R}^{m \times n}$, there exists a matrix $B \in \mathbb{R}^{k \times n}$ such that $\mathcal{C} = \{\lambda B : \lambda \in \mathbb{R}^k_+\}$, and Weyl's theorem

says that, for any finitely generated cone $\mathcal{C} = \{\lambda B : \lambda \in \mathbb{R}^k_+\}$ where $B \in \mathbb{R}^{k\times n}$, there exists a matrix $A \in \mathbb{R}^{m\times n}$ such that $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq 0\}$.

Due to the essential equivalence of finitely constrained cones and finitely generated cones, we refer to them both as *polyhedral* cones. In this thesis, we are interested in finding a finitely constrained formulation of a polyhedral cone given a finitely generated formulation and conversely in finding a finitely generated formulation of a polyhedral cone given a finitely constrained formulation. These two problems are essentially the same problem meaning that one problem can be phrased in terms of the other (see Section 2.3). In this thesis, we approach this problem from the perspective of converting from constraints to generators.

Another problem that we address is to find all the generators of a set of related but different cones; in other words, given a set of finitely constrained cones, we wish to find the generators of all the cones in the set. Here, the set of cones consists of all cones with the same set of linear homogeneous constraints (the coefficients of the variables are fixed) except that constraints may vary between being either *less-than-or-equal-to-zero* ($\leq \mathbf{0}$) constraints or *greater-than-or-equal-to-zero* ($\geq \mathbf{0}$) constraints. More specifically, given some constraint matrix A, this set of cones consists of every cone $\{x \in \mathbb{R}^n : A^1x \geq \mathbf{0}, A^2x \leq \mathbf{0}\}$ where the matrices $A^1 \in \mathbb{R}^{k\times n}$ and $A^2 \in \mathbb{R}^{m-k\times n}$ are some partition of the rows of the matrix A. For example, this set of cones includes the cone $\mathcal{C}(A) = \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$, and the cone $\mathcal{C}(\text{-}A) = \{x \in \mathbb{R}^n : Ax \leq \mathbf{0}\}$. Another way of describing this set of cones is as the set of all cones in the form $\{x \in \mathbb{R}^n : Ax \in \mathcal{O}^n\}$ where $\mathcal{O}^n$ is a subset of $\mathbb{R}^n$, called an *orthant*, given by constraining each coordinate to be either non-negative or (exclusively) non-positive; that is, $\mathcal{O}^n = \{x \in \mathbb{R}^n : \delta_i x_i \geq 0, i = 1, ..., n\}$ for some $\delta \in \{1, \text{-}1\}^n$. Note that $\mathbb{R}^n_+$ is an orthant and $\mathcal{C}(A) = \{x \in \mathbb{R}^n : Ax \in \mathbb{R}^n_+\}$.

In this thesis, we describe how an algorithm for computing the generators of a single cone given a constrained representation can be efficiently adapted to solve this problem. This is well-known. This problem is equivalent to the problem of computing the *circuits of matrices* for which there are applications in computational biology: metabolic pathway analysis ([38]) and gene interaction analysis ([11]). We will defer discussion of circuits of matrices and its applications until Chapter 9.

For the example cone in Figure 1.1, $\mathcal{C} = \{x \in \mathbb{R}^2 : -x_1 + 2x_2 \geq 0, 3x_1 - x_2 \geq 0\}$, there are four cones in the set of cones:

(i). $\mathcal{C}^1 = \{x \in \mathbb{R}^2 : 3x_1 - 2x_2 \geq 0, -x_1 + 2x_2 \geq 0\}$;

(ii). $\mathcal{C}^2 = \{x \in \mathbb{R}^2 : 3x_1 - 2x_2 \geq 0, -x_1 + 2x_2 \leq 0\}$;

(iii). $\mathcal{C}^3 = \{x \in \mathbb{R}^2 : 3x_1 - 2x_2 \leq 0, -x_1 + 2x_2 \geq 0\}$; and

(iv). $\mathcal{C}^4 = \{x \in \mathbb{R}^2 : 3x_1 - 2x_2 \leq 0, -x_1 + 2x_2 \leq 0\}$.

We have depicted these four cones in Figure 1.2. These four cones completely cover the space $\mathbb{R}^2$. The set $\{(2, 1), (2, 3), (\text{-}2, \text{-}1), (\text{-}2, \text{-}3)\}$ is the set of all generators of each of these four cones.

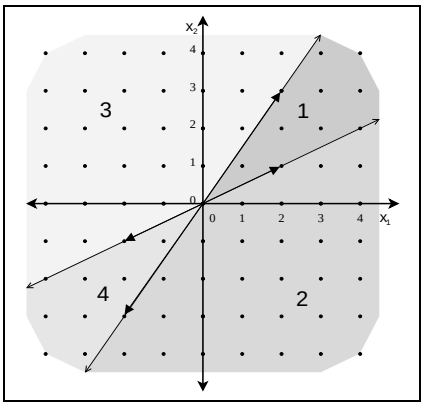

Figure 1.2: A set of cones in two dimensions.

1.2 Convex polyhedra

In this section, we discuss convex polyhedra and their properties. s The expression $\lambda^1 x^1 + \lambda^2 x^2 + ... + \lambda^k x^k$ where $\lambda^1, \lambda^2, ..., \lambda^k \in \mathbb{R}_+$ and $\lambda^1 + \lambda^2 + ... + \lambda^k = 1$ is called a *convex combination* of the points $x^1, x^2, ..., x^k \in \mathbb{R}^n$. A *convex set* is a set $\mathcal{K} \subseteq \mathbb{R}^n$ that is closed under *convex* combinations. More explicitly, if $x^1, x^2, ..., x^k \in \mathcal{K}$, then $\lambda^1 x^1 + \lambda^2 x^2 + ... + \lambda^k x^k \in \mathcal{K}$ for all $\lambda^1, \lambda^2, ..., \lambda^k \in \mathbb{R}_+$ where $\lambda^1 + \lambda^2 + ... + \lambda^k = 1$.

A set of points in $\mathbb{R}^n$ that satisfy a finite set of linear constraints is a convex set, called a *finitely constrained* convex set: the set $\mathcal{P}_A(b) := \{x \in \mathbb{R}^n : Ax \geq b\}$ where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ is a finitely constrained convex set. Note that a polyhedral cone $\mathcal{C}(A)$ is thus a finitely constrained convex set: $\mathcal{C}(A) = \mathcal{P}_A(\mathbf{0})$.

For example, the set $\mathcal{P} = \{x \in \mathbb{R}^2 : x_1 \geq 1, -x_1 + 2x_2 \geq 0, x_1 + x_2 \geq 3, x_1 - x_2 \geq -2\}$ is a finitely constrained convex set. It is depicted in Figure 1.3.

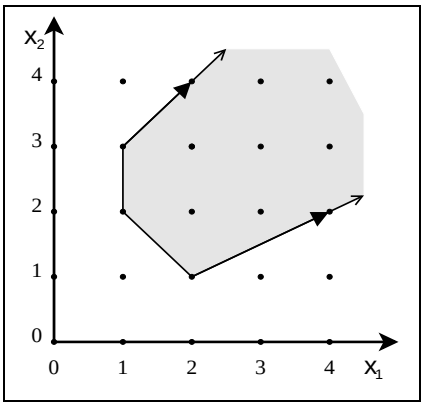

Figure 1.3: A simple convex set in two dimensions.

A set of points in $\mathbb{R}^n$ generated by all the possible convex combinations of a finite set of points in $\mathbb{R}^n$ is a convex set, and also, a set of points in $\mathbb{R}^n$ generated by all the possible conic combinations of a finite set of points in $\mathbb{R}^n$ is a convex set. Moreover, a set of points in $\mathbb{R}^n$ generated by all the possible convex combinations of a one set

of points in $\mathbb{R}^n$ plus all the possible conic combinations of another set of points in $\mathbb{R}^n$ is also a convex set and it is called a *finitely generated* convex set; that is, the set $\{\lambda B + \delta D : \sum_{i=1}^{q} \delta_i = 1, \delta \in \mathbb{R}^q_+, \lambda \in \mathbb{R}^p_+\}$ where $B \in \mathbb{R}^{p\times n}$ and $D \in \mathbb{R}^{q\times n}$ is finitely generated convex set. Here, the set $\{\lambda B + \delta D : \sum_{i=1}^{q} \delta_i = 1, \delta \in \mathbb{R}^q_+, \lambda \in \mathbb{R}^p_+\}$ is all the possible convex combinations of the rows of the matrix B plus all the possible conic combinations of the rows of the matrix D. Note that a polyhedral cone is a finitely generated convex set.

We shall see later in the thesis (Section 2.2) that if a convex set contains no lines, then there is a unique set of minimal convex generators and a unique (up to positive scaling) set of conic generators of any finitely generated convex set. If there are no lines in the polyhedron, then we say that the polyhedron is *pointed*. Geometrically speaking, we call the vertices of a pointed polyhedron the set of *extreme points* of the polyhedron, and we call a vector that lies along an edge that goes to infinity in one direction an *extreme ray* of the polyhedron. This definition of an extreme ray coincides precisely with the definition of an extreme ray given earlier for cones when we regard a cone as a polyhedron. We will see that every polyhedron has a finite number of extreme points and extreme rays and that a polyhedron is generated by a convex combination of its extreme points plus a conic combination of its extreme rays; moreover, the set of extreme points and extreme rays is a minimal generating set of the polyhedron.

The convex set in Figure 1.3 can also be represented as a finitely generated convex set:

$$\mathcal{P} = \{\lambda_1(1,1)+\lambda_2(2,1)+\delta_1(1,3)+\delta_2(1,2)+\delta_3(2,1) : \delta_1+\delta_2+\delta_3 = 1, \delta \in \mathbb{R}^2_+, \lambda \in \mathbb{R}^2_+\}.$$

The three points $(1,3)$, $(1,2)$, and $(2,1)$ are vertices of the polyhedron $\mathcal{P}$, so they are also the extreme points of $\mathcal{P}$. The two vectors $(1,1)$ and $(2,1)$ lie along the two edges that go to infinity, so they are the extreme rays of $\mathcal{P}$. Thus, the polyhedron $\mathcal{P}$ is generated by a convex combination of the extreme points of $\mathcal{P}$ plus a conic combination of the extreme rays of $\mathcal{P}$.

Analogously to the case for cones, a fundamental theorem is that a convex set is finitely constrained if and only if it is finitely generated. We saw this with our small example convex set of Figure 1.3, which we formulated as a finitely constrained convex set and a finitely generated convex set. As for cones, in the forward direction this theorem is commonly known as Minkowski's theorem, and in the other direction, it is known as Weyl's theorem.

We define a *convex polyhedron* as a finitely constrained convex set or equivalently a finitely generated convex set. In this thesis, as with cones, we are interested in converting between different formulations of a convex polyhedron.

Fortunately, this problem of converting between different representations of a polyhedron can be reduced to the problem of converting between different representations of cones. To achieve this, we embed the polyhedron in a cone and perform the conversion from one representation of a cone to another and then extract the polyhedron from the cone in its converted representation. Consequently, since converting from one representation of a cone to another representation is essentially the same problem in both directions, the same is true for polyhedra; that is, the problem of

converting from a finitely constrained formulation to a finitely generated formulation is equivalent to the problem of converting from a finitely generated formulation to a finitely constrained formulation. In this thesis, we concentrate on the problem of converting between different formulations of cones.

1.3 Linear programs

In this section, we define Linear Programs and discuss their properties. Linear programming is a useful tool for integer programming. We only give a brief discussion of linear programming; for a full description of linear programming see for example [71].

A linear program involves minimising or maximising some linear function over the set of points in a polyhedron. We define a linear program as the following:

$$LP := \min\{cx : x \in \mathcal{P}\} \text{ or } LP := \max\{cx : x \in \mathcal{P}\}$$

where $\mathcal{P} \subseteq \mathbb{R}^n$ is a polyhedron and $c \in \mathbb{R}^n$ is a cost vector. We often refer to the set of points in the polyhedron $\mathcal{P}$ as the *feasible* points of the linear program LP. By convention, we assume that we are minimising c and not maximising; indeed, we can maximise c by minimising $-c$. We can also optimise $cx + k$ over $\mathcal{P}$ where k is some constant, but k can effectively be ignored since $\min\{cx + k : x \in \mathcal{P}\} = \min\{cx : x \in \mathcal{P}\} + k$, so we usually omit it. The standard form of a linear program that we consider is $LP := \min\{cx : x \in \mathcal{P}\}$ where $c \in \mathbb{R}^n$, and $\mathcal{P}$ is a polyhedron.

The standard algorithm for solving a linear program is the *simplex method* developed by George Dantzig ([29]). This method starts from some vertex of the polyhedron $\mathcal{P}$ and then iteratively moves along the edges of $\mathcal{P}$ from one vertex to another adjacent vertex of lower cost. Crucially, at least one of the vertices of the polyhedron $\mathcal{P}$ is an optimal solution of LP if LP has an optimal solution and $\mathcal{P}$ is pointed. The method stops when we can no longer find an adjacent vertex of lower cost, in which case, the current vertex is an optimal solution. The simplex algorithm has proven to be extremely efficient for practical problems even though it has exponential worst-case complexity for all current variations ([64]). But, it is an open problem whether a variation exists that has polynomial time worst-case complexity. For more information on linear programming, see [71].

There is also the class of methods called *interior point methods*. They solve a linear program by generating a sequence of points in the interior of the polyhedron $\mathcal{P}$ that converge to the optimal solution. Interior point methods are derived from the *projective method* proposed by Karmarkar in [61]. Interior point methods have polynomial time worse-case complexity and they are particularly effective on extremely large problem instances in the order of millions of variables (provided that the input data is sparse). For more information on interior point methods, see [96].

1.4 Integer programs

We define an integer program as the following:

$$IP := \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$$

where $\mathcal{P}$ is a rational polyhedron, and $c \in \mathbb{R}^n$ is a cost vector. So, an integer program involves minimising some linear function over the set of integral points in some polyhedron. We call the set of integer points in a polyhedron, $\mathcal{F} = (\mathcal{P} \cap \mathbb{Z}^n)$, the *feasible* points of the integer program IP.

In general, integer programs are hard to solve. In contrast with linear programs $LP := \min\{cx : x \in \mathcal{P}\}$ where there is always a vertex of the polyhedron $\mathcal{P}$ that is an optimal solution (assuming an optimal solution exists), the set of optimal solutions of $IP := \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$ may lie in the interior of the polyhedron $\mathcal{P}$, which can make them difficult to find.

A common approach for solving an integer program is first to solve the *linear programming relaxation* of the integer program. The linear programming relaxation of an integer program $IP := \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$ is the linear program $LP := \min\{cx : x \in \mathcal{P}\}$. Here, we have the same cost function and the same polyhedron, but we have *relaxed* the requirement that the feasible solutions are integral. If an optimal solution of the linear program is integral, then it is also a feasible solution of the integer program, and moreover, it is also an optimal solution of IP. Indeed, every feasible solution of IP is also a feasible solution of LP, and therefore, an optimal solution of IP cannot have a lower cost than an optimal solution of LP. Therefore, by solving the linear programming relaxation, we potentially also solve the original integer program. However, it often occurs that the optimal solution of the linear programming relaxation is not integral, in which case, we need other techniques to solve the integer program.

One approach is to reformulate the integer program using different and or additional variables and constraints such that ideally the linear programming relaxation has an integral optimal solution or at least the integer program is easier to solve for whatever reason. One such reformulation technique, called the *cutting-plane* algorithm, involves adding additional constraints to create a new polyhedron $\mathcal{P}'$ such that $\mathcal{P}' \subseteq \mathcal{P}$ and $(\mathcal{P}' \cap \mathbb{Z}^n) = (\mathcal{P} \cap \mathbb{Z}^n)$. Thus, $IP = \min\{cx : x \in (\mathcal{P}' \cap \mathbb{Z}^n)\}$. In doing so, we hope that the optimal solution of the linear relaxation $LP' = \min\{cx : x \in \mathcal{P}'\}$ is integral and thus optimal for IP. The basic procedure for finding additional constraints is as follows. We first solve the linear relaxation $LP = \min\{cx : x \in \mathcal{P}\}$ given an optimal solution x^*. If x^* is integral, then it is an optimal solution of IP. Otherwise, we search for a constraint $\pi x \leq \pi_0$ where $\pi \in \mathbb{R}^n$ and $\pi_0 \in \mathbb{R}$ such that $\pi x \leq \pi_0$ for all $x \in (\mathcal{P} \cap \mathbb{Z}^n)$, but $\pi x^* > \pi_0$. The constraint $\pi x \leq \pi_0$ is called a *cut*. Then, we add the constraint to the polyhedron $\mathcal{P}$ creating a new smaller polyhedron $\mathcal{P}' = \mathcal{P} \cap \{x \in \mathbb{R}^n : \pi x \leq \pi_0\}$. By construction, $\mathcal{P}' \subseteq \mathcal{P}$ and $(\mathcal{P}' \cap \mathbb{Z}^n) = (\mathcal{P} \cap \mathbb{Z}^n)$. Importantly, $x^* \notin \mathcal{P}'$, and thus, $LP' = \min\{cx : x \in \mathcal{P}'\}$ has a different optimal solution than $LP = \min\{cx : x \in \mathcal{P}\}$, which is hopefully integral. If the new optimal solution is not integral, then we repeat the process and try to find a cut for the new optimal solution of the linear relaxation.

Another approach is called *branch-and-bound*. This approach involves splitting the integer program IP into smaller sub-problems such that every feasible solution of IP is also feasible for some sub-problem. The idea is that each integer sub-program is in some way easier to solve than IP. Then, by solving each of the sub-problems, we can solve IP. Specifically, we split the polyhedron $\mathcal{P}$ into two or more (usually disjoint) smaller polyhedra $\mathcal{P}^1, \mathcal{P}^2, ..., \mathcal{P}^K$ such that $(\mathcal{P} \cap \mathbb{Z}^n) = [(\mathcal{P}^1 \cup \mathcal{P}^2 \cup \cdots \cup \mathcal{P}^K) \cap \mathbb{Z}^n]$. Then, $IP = \min_k IP^k$ where $IP^k = \min\{cx : x \in (\mathcal{P}^k \cap \mathbb{Z}^n)\}$ for $k = 1, ..., K$ since the optimal solution of IP must also be an optimal solution of IP^k for some $k = 1, ..., k$. This process of splitting the integer program IP into sub-problems is called *branching*, and it can be applied recursive by further splitting each sub-problem into sub-sub-problems and so on until each individual sub-problem is *easy* to solve (e.g. the optimal solution of the linear relaxation is integral). Additionally, in some circumstances, we can deduce that we do not actually need to solve a sub-problem. Assume that we have found a feasible solution $x \in (\mathcal{P} \cap \mathbb{Z}^n)$ of IP perhaps by solving one of the sub-problems. Then, we know that $cx \geq IP$, or in other words, the feasible solution provides an upper bound on IP. Also, the optimal value of the linear relaxation of a sub-problem IP^k provides a lower bound on the optimal solution of IP^k: $IP^k \geq LP^k = \min\{cx : x \in \mathcal{P}^k)\}$. Therefore, if $LP^k \geq cx$, then there can be no feasible solution of IP^k that is better than x, and thus, we do not need to solve IP^k, in which case, we say that the sub-problem has been pruned by *bound*.

We can combine the two techniques of cutting-planes and branch-and-bound giving the technique called *branch-and-cut*. In this case, we apply the branch-and-bound approach, and in addition, we apply the cutting-plane technique to sub-problems within the branch-and-bound framework. Branch-and-cut is the standard technique used for solving integer programs. The branch-and-cut technique is implemented in most commercial software packages that solve integer programs. It is remarkably effective and continues to be improved to this day. However, there still remain some classes of problems for which this technique is ineffective. One such class of problems is the "Market Split" problems (see [26]).

For a more detailed and thorough presentation of the theory and algorithms for integer programming, see for example the texts [71, 95].

1.5 Gröbner bases

In this thesis, we present an alternative exact technique for solving integer programs; this technique is called the *Gröbner basis* method. The basic framework of the Gröbner basis method was first introduced by Graver in [45]. For a given integer program, the term Gröbner basis refers to a set of vectors such that we can move from any non-optimal feasible solution of the integer program to a better feasible solution by subtracting a vector in the Gröbner basis from the feasible solution. A Gröbner basis is thus a set of *augmenting* or *improving* vectors. More formally, consider the integer program $IP := \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$. A Gröbner basis of IP is a set of vectors $G \subseteq \mathbb{Z}^n$ such that for every non-optimal solution $x \in (\mathcal{P} \cap \mathbb{Z}^n)$ of IP, there exists a vector $u \in G$ such that $x - u \in (\mathcal{P} \cap \mathbb{Z}^n)$ and $c(x - u) < cx$.

The definition of a Gröbner basis guarantees that if a feasible solution of IP is not optimal, then we can improve it by some vector in the Gröbner basis, and a feasible solution must be optimal if we cannot improve it by some vector in the Gröbner basis. We can thus solve the integer program IP by first finding a feasible solution and then iteratively improving the solution using vectors in the Gröbner basis until we attain an optimal solution (assuming that an optimal solution exists). Note when solving an IP using a Gröbner basis, we construct a strictly cost decreasing sequence of feasible solutions.

Example 1.5.1. *Consider the integer program* $IP_{A,c}(b) := \min\{cx : x \in \mathcal{P}_A(b) \cap \mathbb{Z}^n\}$ *where*

$$A = \begin{pmatrix} 2 & 3 \\ -2 & +1 \\ -2 & -1 \\ 1 & -1 \end{pmatrix}, b = \begin{pmatrix} 6 \\ -4 \\ -10 \\ -1 \end{pmatrix}, \text{ and } c = (3,4),$$

so $IP_{A,c}(b) = \min\{3x_1+4x_2 : 2x_1+3x_2 \geq 6, 2x_1-x_2 \leq 4, 2x_1+x_2 \leq 10, x_1-x_2 \geq \text{-}1\}$. *We have depicted the set of feasible solutions in Figure 1.4. In this diagram, the solid lines represent the constraints and the area within the solid lines is the polyhedron* $\mathcal{P}_A(b)$. *The feasible solutions of* $IP_{A,c}(b)$, *the integer points within* $\mathcal{P}_A(b)$, *are drawn as dots within the polyhedron* $\mathcal{P}_A(b)$. *The optimal solution of* $IP_{A,c}(b)$ *is the point* $(2,1)$.

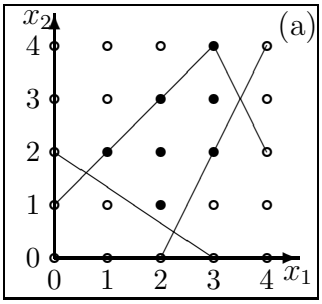

Figure 1.4: The set of feasible solutions for $IP_{A,c}(b)$.

Consider the set $G = \{(1,0),(0,1)\}$. *In Figure 1.5(a), we have drawn the vectors of* G *onto the set of feasible solutions of* IP *where it is possible to a use a vector from* G *to move from one feasible solution to another. For example, we can improve the feasible solution* $(2,3)$ *using the the vector* $(0,1)$ *in* G *to move to the feasible solution* $(2,3)-(0,1) = (2,2)$. *However, we cannot improve the non-optimal feasible solution* $(1,2)$ *since both points* $(1,2)-(1,0) = (0,2)$ *and* $(1,2)-(0,1) = (1,1)$ *are infeasible. Therefore, this set is not a Gröbner basis of* IP.

The point $(1,2)$ *is the second best solution of* $IP_{A,c}(b)$, *so if we improved* $(1,2)$, *we must arrive at the optimal solution* $(2,1)$. *Therefore, we must have the vector* $(\text{-}1,1) = (1,2)-(2,1)$ *in a Gröbner basis of* $IP_{A,c}(b)$. *Let* $G' := \{(1,0),(0,1),(\text{-}1,1)\}$. *This set* G' *is a Gröbner basis of* $IP_{A,c}(b)$. *In Figure 1.5(a), we have drawn the vectors of* G' *onto the set of feasible solutions. We can verify from this picture that indeed every non-optimal solution can be improved using a vector in* G'.

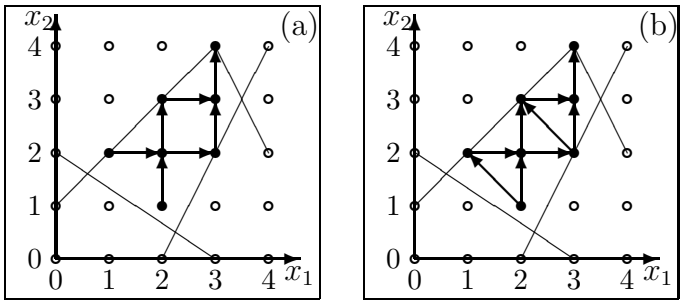

Figure 1.5: The possible improvements by vectors in G and G'.

The basic and well-known Gröbner basis method for an integer program involves first computing a Gröbner basis and then solving the integer program using the vectors in the Gröbner basis by iteratively improving a given feasible solution until we attain an optimal solution (see [45, 25]). Almost all of the computation time of the Gröbner basis method is spent in first computing the Gröbner basis for an integer program; once we have a Gröbner basis, we can optimise the integer program in a comparatively small amount of time.

The Gröbner basis method is an *exact local search method*. It is exact in the sense that it is guaranteed to find the optimal solution if one exists and given sufficient time and computing resources. This approach differs from other local search methods such as *tabu search* and *simulated annealing*, which are heuristics for which there is no guarantee that the solution returned by these methods is optimal.

In some cases, for example in *stochastic* integer programming (see for example [83]), we want to solve not just one integer program but a set of many different but related integer programs. The Gröbner basis method is well suited to this situation. Here, the set of different integer programs consists of all integer programs that have the same given cost function and the same given set of equality and inequality constraints except that the constant term in a constraint (the right-hand-side) is allowed to vary from one integer program to another. More formally, this is the set of integer programs $IP_{A,c}(b)$ where A and c are fixed, but b is allowed to vary, and we write this set as $IP_{A,c}(\cdot) := \{IP_{A,c}(b) : b \in \mathbb{R}^m\}$.

Above, we considered a Gröbner basis of one particular integer program. We now define a Gröbner basis of a set of integer programs as a set of integer vectors that is simultaneously a Gröbner basis for every integer program in the set. Specifically, a set $G \subseteq \mathbb{Z}^n$ is a Gröbner basis of $IP_{A,c}(\cdot)$ if G is a Gröbner basis of $IP_{A,c}(b)$ for all $b \in \mathbb{R}^m$.

Example 1.5.2. *Consider the integer program from Example 1.5.1 above. The set* $G' = \{(1,0),(0,1),(\text{-}1,1)\}$ *is a Gröbner basis of* $IP_{A,c}(b)$ *where* $b = (6, \text{-}4, \text{-}10, \text{-}1)$*, but it is not a Gröbner basis of* $IP_{A,c}(\cdot)$*.*

Consider the integer program $IP_{A,c}(b')$ *where* $b' = (6, \text{-}4, \text{-}10, 1)$*; the feasible set is depicted in 1.6(a). Note that there are only two feasible solutions of* $IP_{A,c}(b')$*, and they are* $\{(2,1),(3,2)\}$*. Here,* $(2,1)$ *is the optimal solution, and* $(3,2)$ *is a non-optimal solution. So, a Gröbner basis of* $IP_{A,c}(b')$ *must contain the vector* $(1,1)$*, which is the*

only vector that can improve $(3, 2)$. *Let* $G'' = \{(1, 0), (0, 1), (-1, 1), (1, 1)\}$. *The set* G'' *is also not a Gröbner basis of* $IP_{A,c}(\cdot)$.

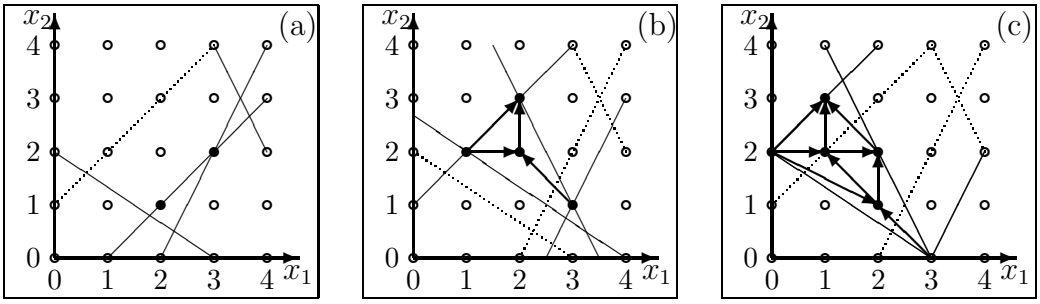

Figure 1.6: The feasible sets of $IP_{A,c}(b')$, $IP_{A,c}(b'')$, and $IP_{A,c}(b''')$.

Additionally, consider the set integer program $IP_{A,c}(b'')$ *where* $b'' = (8, -5, -7, -1)$*; the feasible set is depicted in 1.6(b).* G'' *is not a Gröbner basis of* $IP_{A,c}(b'')$ *since the non-optimal point* $(3, 1)$ *cannot be improved. In this case,* $(1, 2)$ *is the optimal solution and* $(3, 1)$ *is the second best solution. Therefore, we need the vector* $(2, -1) = (3, 1) - (1, 2)$ *in any Gröbner basis of* $IP_{A,c}(b'')$. *Let* $G''' = \{(1, 0), (0, 1), (-1, 1), (1, -1), (2, -1)\}$. *Again,* G''' *is not a Gröbner basis of* $IP_{A,c}(\cdot)$.

Finally, consider the set integer program $IP_{A,c}(b''')$ *where* $b''' = (6, -6, -6, -2)$*; the feasible set is depicted in 1.6(c).* G'' *is not a Gröbner basis of* $IP_{A,c}(b''')$ *since the non-optimal point* $(3, 0)$ *cannot be improved. In this case,* $(0, 2)$ *is the optimal solution and* $(3, 0)$ *is the second best solution. Therefore, we need the vector* $(3, -2) = (3, 0) - (0, 2)$ *in any Gröbner basis of* $IP_{A,c}(b''')$.

Let $G''' = \{(1, 0), (0, 1), (-1, 1), (1, -1), (2, -1), (3, -2)\}$. *At last,* G''' *is a Gröbner basis of* $IP_{A,c}(\cdot)$. *This is certainly not obvious even for such a small example.*

A fundamental result in the theory of Gröbner bases of integer programs is that there always exists a finite set of vectors that is a Gröbner basis of $IP_{A,c}(\cdot)$. Another fundamental result is that we can compute a finite Gröbner basis of $IP_{A,c}(\cdot)$. The algorithm for computing a Gröbner bases of $IP_{A,c}(\cdot)$ is called the *completion procedure* or the *Buchberger algorithm* named after Bruno Buchberger who discovered the algorithm (see [14]).

The Gröbner basis method for solving a set of integer programs (i.e. a finite subset of $IP_{A,c}(\cdot)$) involves first computing a Gröbner basis of $IP_{A,c}(\cdot)$, then solving each individual integer program in the set using the vectors in the Gröbner basis as described above. As in the case for one integer program, almost all of the computation time of the Gröbner basis method for a set of integer programs is spent in first computing the Gröbner basis for the set of integer programs; once we have a Gröbner basis, we can optimise all of the individual integer programs in the set in a comparatively small amount of time.

Since there is an algorithm for computing a Gröbner basis of $IP_{A,c}(\cdot)$, the basic Gröbner basis method for solving one integer program $IP_{A,c}(b)$ actually involves computing a Gröbner basis of $IP_{A,c}(\cdot)$. This approach of first computing a Gröbner

basis of $IP_{A,c}(\cdot)$ is very powerful when we wish to solve many different integer programs in the set $IP_{A,c}(\cdot)$, but in the more common case when we wish to only a particular integer program, it is disadvantageous to compute a Gröbner basis of $IP_{A,c}(\cdot)$ since we often compute a much larger set than is necessary. Indeed, a Gröbner basis of $IP_{A,c}(\cdot)$ may be much larger than a Gröbner basis of $IP_{A,c}(b)$, and in general, the larger the size of the Gröbner basis the longer it takes to compute.

Fortunately, there is variation of the algorithm for computing a Gröbner basis of $IP_{A,c}(\cdot)$ that computes what is called a *truncated* Gröbner basis, which is more specific to a particular integer program. A truncated Gröbner basis of an integer program $IP_{A,c}(b)$ consists of the vectors in a Gröbner basis for $IP_{A,c}(\cdot)$ except for those vectors that cannot be used to move from one feasible solution of the integer program of $IP_{A,c}(b)$ to another feasible solution – such vectors are essentially *too long* to be useful; in other words, we exclude vectors $v \in \mathbb{Z}^n$ for which there do not exist two feasible solutions x and y of $IP_{A,c}(b)$ such that $x - y = v$. A vector that cannot move between two feasible solutions of $IP_{A,c}(b)$ is never needed in a Gröbner basis of $IP_{A,c}(b)$, and thus, a truncated Gröbner basis of $IP_{A,c}(b)$ is thus a Gröbner basis of $IP_{A,c}(b)$ since we have not thrown away any vectors that we might need. A vector that can step between two feasible solutions of $IP_{A,c}(b)$ still may not be strictly needed in a Gröbner basis of $IP_{A,c}(b)$, but hopefully, there are not too many unnecessary vectors in the truncated Gröbner basis.

Example 1.5.3. *We saw in Example 1.5.2 that the set*

$$G''' = \{(1,0), (0,1), (\text{-}1,1), (1,\text{-}1), (2,\text{-}1), (3,\text{-}2)\}$$

is a Gröbner basis of $IP_{A,c}(\cdot)$ for the integer program $IP_{A,b}(c)$ in Example 1.5.1 above. In Figure 1.7, we have drawn the vectors of G''' onto the set of feasible solutions of $IP_{A,b}(c)$ where it is possible to a use a vector from G''' to move from one feasible solution to another. Note that the vectors $(2,\text{-}1)$ and $(3,\text{-}2)$ are never used. They are too long to fit inside the feasible set of $IP_{A,c}(b)$. Therefore, the set $\{(1,0), (0,1), (\text{-}1,1), (1,1)\}$ is a truncated Gröbner basis of $IP_{A,c}(b)$.

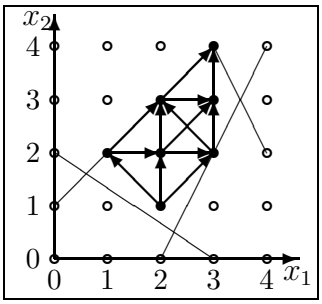

Figure 1.7: The feasible set of $IP_{A,c}(b)$ with vectors form G'''.

Note that the set $\{(1,0), (0,1), (\text{-}1,1)\}$ is a Gröbner basis of $IP_{A,c}(b)$ as shown previously. So, although the truncated Gröbner basis of $IP_{A,c}(b)$ is smaller than a Gröbner basis of $IP_{A,c}(\cdot)$, it is still larger than strictly needed for a Gröbner basis of $IP_{A,c}(b)$.

A truncated Gröbner basis of $IP_{A,c}(b)$ is potentially much smaller than a Gröbner basis of $IP_{A,c}(\cdot)$; it is possibly just an empty set, which occurs when the set of feasible solutions of $IP_{A,c}(b)$ is empty or contains only one solution – every vector is then too long by definition. Crucially, we can compute truncated Gröbner bases, and we can do so without first computing a Gröbner basis of $IP_{A,c}(\cdot)$ and then removing vectors that are too long. Truncated Gröbner bases for integer programs in their most general form were first introduced by Weismantel and Thomas in [85], and in a simplified form in [92]. In this thesis, we extend the techniques proposed in [85] for computing truncated Gröbner bases.

The truncated Gröbner basis approach is still not adequate though since a truncated Gröbner basis may still be prohibitively larger than is necessary for solving a particular integer program, so we still need to improve upon this approach. Thus, in this thesis, we look at ways to reduce the size of the truncated Gröbner basis.

The size of a truncated Gröbner basis of an integer program depends on the size of the set of feasible solutions of the integer program; in general, the larger the feasible set of the integer program the larger the truncated Gröbner basis since the larger the feasible set the more vectors that fit inside it. But, the size of the feasible set does not necessarily reflect the difficulty of solving the integer program. We can remedy this by introducing a bound constraint on the cost of feasible solutions (an upper bound for minimisation and a lower bound for maximisation) to reduce the size of the feasible set and thus to strengthen truncation. The value of this bound can be computed from a given *good* feasible solution. For example, if $\bar{x}$ is a feasible solution of $IP_{A,c}(b)$, then $cx \leq c\bar{x}$ is a bound constraint for $IP_{A,c}(b)$. Importantly, introducing this bound constraint does not increase the size of a truncated Gröbner basis, and it has the potential to drastically reduce the size of a truncated Gröbner basis.

Example 1.5.4. *The set* $\{(1,0),(0,1),(\text{-}1,1),(1,1)\}$ *is a truncated Gröbner basis of* $IP_{A,c}(b)$ *from Example 1.5.1 above. Now, consider that we have found the feasible solution* $(1,2)$*. This is second best solution, so it provides a very tight upper bound on the optimal solution. The cost of the point* $(1,2)$ *is* $(3,4)\cdot(1,2) = 11$*. If we introduce the constraint* $3x_1 + 4x_2 \leq 11$ *into the integer program, then there are only two feasible solutions, and the set* $\{(\text{-}1,1)\}$ *is a truncated Gröbner basis of the bounded integer program.*

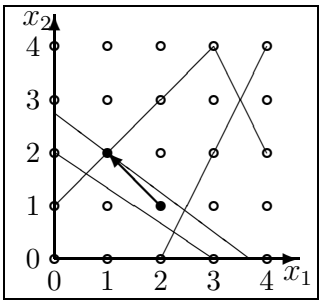

Figure 1.8: The bounded feasible set of $IP_{A,c}(b)$ with vectors from G'''.

Another possible reason for an unnecessarily large truncated Gröbner bases is the presence of *unnecessary* constraints. A constraint is unnecessary if we can *relax* it (i.e. remove it from the formulation) and still solve the integer program meaning that the optimal solution of the relaxed integer program is the optimal solution of the original integer program. Redundant constraints are unnecessary since relaxing redundant constraints does not change the set of feasible solutions and thus does not change the optimal solution. Irredundant constraints may also be unnecessary particularly when they are in some sense not active around the optimal solution. To avoid computing with unnecessary constraints, we solve a hierarchy of relaxations of the integer program. This involves the following steps. Initially, we choose some initial relaxation of the integer program that has an optimal solution; in particular, we choose what is called the *Gomory group relaxation* named after Gomory who first suggested this relaxation in [43]. Then, we solve the relaxation by computing a Gröbner basis and iteratively improving a given feasible solution. If the optimal solution of the relaxation is optimal for the original problem we can stop, otherwise we add a relaxed constraint that is violated by the optimal solution of the relaxation creating what is called an *extended group relaxation* and repeat. Crucially, we can compute the Gröbner basis for each iteration incrementally. By only adding constraints that are violated, we potentially avoid adding unnecessary constraints.

Combining the technique of adding a bound constraint and the technique of solving a hierarchy of relaxations gives a novel Gröbner basis approach to solve a particular integer program. This novel Gröbner basis approach performs better than other general Gröbner basis methods because it can use structure specific to the particular integer program. Solving a hierarchy of group relaxations was explored in [43, 94, 59, 87], and truncated methods were explored in [85]; however, combining these two approaches and strengthening truncation by using bounds has not been done before.

Gröbner basis methods can also be applied to the *integer feasibility enumeration problem* where we wish to enumerate all of the integer feasible solutions. Tayur et al. in [83] showed that we can use a Gröbner basis to enumerate all feasible solutions. We present this enumeration method in this thesis and discuss how it can help to solve optimisation problems.

The general concept of a Gröbner basis of an integer program, also known as a *test set*, was first introduced by Graver in 1975 in [45]. The test sets that Graver introduced, called *Graver bases*, are Gröbner bases of a set of integer programs with a particular structure. Specifically, a Graver basis is a set of integer vectors that is simultaneously a Gröbner basis for the family of integer programs $IP_A(c,u,b) := \min\{cx : u \geq Ax \geq b, x \in \mathbb{Z}^n\}$ where $A \in \mathbb{R}^{m\times n}$ is fixed and $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $u \in \mathbb{R}^m$ are allowed to vary. Here, we define a Gröbner basis of $IP_A(c,u,b)$ in the same way as a Gröbner bases of $IP_{A,c}(b)$. In 1991, Conti and Traverso ([25]) first defined Gröbner bases of $IP_{A,c}(\cdot)$ and showed that a Gröbner basis of a special type of polynomial ideal in algebraic geometry was essentially a Gröbner basis of $IP_{A,c}(\cdot)$ (see also [3]); these special types of polynomial ideals are called toric ideals or more generally lattice ideals ([81]). See [28, 10, 3] for a general introduction to polynomial ideals in algebraic geometry. A Gröbner basis of an integer program is a translation of these Gröbner bases of certain polynomial ideals into the context of integer programming; it is from Gröbner bases of polyno-

mial ideals that Gröbner bases of integer programs received their name.[1] In 1995, Thomas translated results on Gröbner bases of polynomial ideals from algebraic geometry into an integer programming context ([86]). The main algorithm for computing Gröbner bases of integer programs is derived from the Buchberger algorithm for computing Gröbner bases of polynomial ideals ([14]). Most of the results and algorithms concerning Gröbner bases of integer programs have an analogous result or algorithm in the context of Gröbner bases of polynomial ideals. We will present the results in an integer programming context following the lead of Thomas in [86] and not an algebraic context. We refer the reader to [83, 92] for a description of the link between Gröbner bases of integer programs and Gröbner bases of polynomial ideals. Also, for a very brief summary of the relationship, see Appendix A.

1.6 Markov bases

A Markov basis relates to the set feasible solutions of a set of discrete or integer points in a polyhedron $\mathcal{F} = (\mathcal{P} \cap \mathbb{Z}^n)$, or in other words, the set of feasible solutions of an integer program. We refer to the set $\mathcal{F} = (\mathcal{P} \cap \mathbb{Z}^n)$ as a *fiber*. A Markov basis of a fiber is a set of integer vectors such that we can move from any point in the fiber to any other point in the fiber in a finite number of steps via other points in the fiber using the vectors in the Markov basis. We can move from one point to another by adding or subtracting a vector in the Markov basis. Note that when moving from one point to another we must stay within $\mathcal{F}$. More formally, a set of vectors $M \subseteq \mathbb{Z}^n$ is a Markov basis of $\mathcal{F}$ if, for every pair of points $x, y \in \mathcal{F}$, there exists a sequence of points $(x^1, x^2, ..., x^k) \subseteq \mathcal{F}$ where $x^1 = x$ and $x^k = y$ such that either $x^i - x^{i+1} \in M$ or $x^{i+1} - x^i \in M$ for all $i = 1, ..., k-1$.

We can describe Markov bases very nicely and succinctly using the concept of connected graphs. Consider the graph $\mathcal{G}(\mathcal{F}, M)$ where the nodes of the graph are the points in $\mathcal{F}$ and there is an edge between two nodes $x, y \in \mathcal{F}$ if either $x - y \in M$ or $y - x \in M$ (the graph is undirected). Then, M is a Markov basis of $\mathcal{F}$ if and only if the graph $\mathcal{G}(\mathcal{F}, M)$ is connected. Recall that a graph is connected if there exists a path between any two nodes in the graph, and a path from x to y in $\mathcal{G}(\mathcal{F}, M)$ is a sequence of points $(x^1, x^2, ..., x^k) \subseteq \mathcal{F}$ where $x^1 = x$ and $x^k = y$ such that either $x^i - x^{i+1} \in M$ or $x^{i+1} - x^i \in M$ for all $i = 1, ..., k-1$.

Example 1.6.1. *Consider the fiber $\mathcal{F}_A(b) = (\mathcal{P}_A(b) \cap \mathbb{Z}^n)$ where the polyhedron $\mathcal{P}_A(b)$ is as defined in Example 1.5.1. Let $M = \{(1,0), (0,1)\}$. Then, M is a Markov basis of $\mathcal{F}_A(b)$. We can see this by examining the graph $\mathcal{G}(\mathcal{F}_A(b), M)$ as depicted in Figure 1.9(a). This graph is connected, so M is a Markov basis of $\mathcal{F}_A(b)$.*

On the other hand, the set $M' = \{(1,0), (-1,1)\}$ is not a Markov basis of $\mathcal{F}_A(b)$. The graph $\mathcal{G}(\mathcal{F}_A(b), M')$ depicted in Figure 1.9(b) is not connected because the point $(3,4)$ is isolated from the rest of the graph.

[1] A Gröbner basis of a polynomial ideal is a particular generating set of the ideal which has certain properties. There are many uses for Gröbner bases: for example, deciding equality of ideals, deciding ideal membership, and solving polynomial systems of equations.

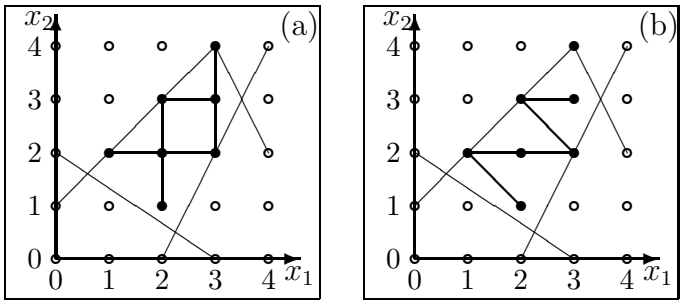

Figure 1.9: The graphs $\mathcal{G}(\mathcal{F}_A(b), M)$ and $\mathcal{G}(\mathcal{F}_A(b), M')$.

Three applications of Markov bases are computing Gröbner bases, sampling from fibers, and local search heuristics. Next, we discuss these applications.

We need Markov bases to compute Gröbner bases of integer programs. The first stage of computing a Gröbner bases of an integer program is computing a Markov basis of the feasible solutions of the integer program. This is often the most expensive part of the Gröbner basis computation and many of the improvements for computing Gröbner bases presented in this thesis are due to improvements for computing Markov bases.

Markov bases can be used to sample from a finite fiber. The fiber may be too large to enumerate, and instead, we wish to find a representative subset of the fiber. We can construct such a representative set using a Markov basis as follows: first, we choose some initial point in the fiber; second, we add the point to the sample; thirdly, using the vectors in the Markov basis, we move randomly from one point in the fiber to another a certain number of times and then add the current point to the sample and repeat. This technique is called *Markov chain Monte Carlo* (see for example [40]); Markov bases are named after Markov chains. It is important that there is a non-zero probability of sampling any point in the fiber, which is guaranteed by definition of a Markov basis. One of the applications of this approach is in algebraic statistics to test validity of statistical models via sampling (see Diaconis and Sturmfels [30]). Another area of application is in computational biology for problems arising from phylogenetic trees (see [33]).

Markov bases are used by local search heuristics to solve integer programs although they are not known under this name. In a local search method, we start from some given feasible solution, and we iteratively move to another feasible solution within some set of candidate feasible solutions called a *neighbourhood* of the current feasible solution. We desire that, given any starting point of the local search algorithm, it is at least possible to reach the global optimum of the integer program by taking a finite number of steps from one feasible solution to another. Often this property is called *reachability*. One possible way to define a neighbourhood is using a set of vectors where the neighbourhood consists of all feasible solutions given by adding or subtracting a vector in the set of vectors from the current feasible solution. The reachability property is satisfied for this definition of a neighbourhood precisely when the set of vectors is a Markov basis of the set of feasible solutions.

Under some circumstances, we can find a Markov basis of a fiber easily; however, in

general, this is not the case. It is always possible to find a Markov basis of a superset of the fiber easily (which is a standard approach for local search heuristics); in this case, we allow paths to be infeasible for the original fiber but they still must be feasible for the superset of the fiber and they still must connect every two points. However, this is not always desirable. One of the topics we address in this thesis is how to compute Markov bases of fibers.

Most existing approaches for computing Markov bases compute a Markov basis that is simultaneously a Markov basis for each fiber in a set of different but related fibers analogously to Gröbner bases. Here, the set of different fibers consists of all possible fibers given a set of constraints where the coefficients of the variables are fixed and the constant term (right-hand-side) is allowed to vary. More formally, this is the set of fibers $\mathcal{F}_A(b) = (\mathcal{P}_A(b) \cap \mathbb{Z}^n)$ where the constraint matrix A is fixed and b is allowed to vary, and we write this set as $\mathcal{F}_A(\cdot) := \{\mathcal{F}_A(b) : b \in \mathbb{R}^m\}$. In this thesis, we present a new approach for computing Markov bases of $\mathcal{F}_A(\cdot)$. This approach computes a Markov basis of $\mathcal{F}_A(\cdot)$ incrementally. Initially, we relax some constraints of the fiber such that we can easily find a Markov basis of the relaxed fiber. Then, we iteratively add a relaxed constraint and incrementally compute the Markov basis for the new relaxed fiber and repeat. After we have added all of the relaxed constraints, we must have a Markov basis of the original fiber.

Example 1.6.2. *Consider the fiber $\mathcal{F}_A(b) = (\mathcal{P}_A(b) \cap \mathbb{Z}^n)$ as in Example 1.6.1 above. Again, let $M = \{(1,0),(0,1)\}$. Then, M is a Markov basis of $\mathcal{F}_A(b)$ as we saw in the previous example. However, M is not a Markov basis of $\mathcal{F}_A(\cdot)$.*

Consider $b' = (6, \text{-}4, \text{-}10, 1)$. The fiber $\mathcal{F}_A(b')$ contains only two points: $(2,1)$ and $(3,2)$ (see Figure 1.10(a)), and thus, any Markov basis of $\mathcal{F}_A(\cdot)$ needs to contain the vector $(1,1) = (3,2) - (2,1)$ (or $(\text{-}1, \text{-}1)$).

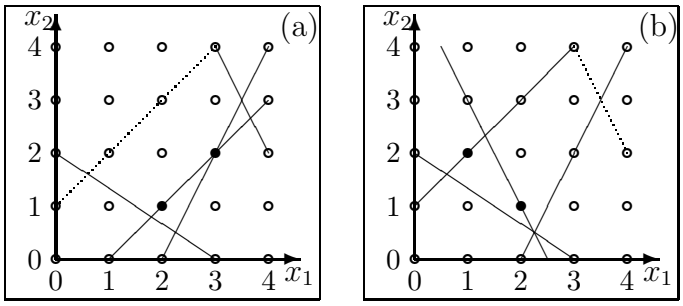

Figure 1.10: The graphs $\mathcal{G}(\mathcal{F}_A(b'), M)$ and $\mathcal{G}(\mathcal{F}_A(b''), M)$.

Also, consider $b'' = (6, \text{-}4, \text{-}5, \text{-}1)$. The fiber $\mathcal{F}_A(b')$ also contains only two points: $(2,1)$ and $(1,2)$ (see Figure 1.10(b)), and thus, any Markov basis of $\mathcal{F}_A(\cdot)$ needs to contain the vector $(\text{-}1, 1) = (1,2) - (2,1)$ (or $(1, \text{-}1)$).

The set $M'' = \{(1,0),(0,1),(1,1),(\text{-}1,1)\}$ is a Markov basis of $\mathcal{F}_A(\cdot)$, but this is certainly not obvious.

In some circumstances, we do need a Markov basis of $\mathcal{F}_A(\cdot)$, but sometimes, we want a Markov basis for only one fiber $\mathcal{F}_A(b)$. As with Gröbner bases, a Markov basis of

$\mathcal{F}_A(\cdot)$ may be much larger than a Markov basis of one fiber $\mathcal{F}_A(b)$, so we would prefer to compute a Markov basis which is specific to $\mathcal{F}_A(b)$. Analogously, to truncated Gröbner bases, we define truncated Markov bases: a truncated Markov basis of a set of feasible solutions is a Markov basis for $\mathcal{F}_A(\cdot)$ after removing all vectors that are too long to be useful. Previously, to compute a truncated Markov basis required first computing a Markov basis of $\mathcal{F}_A(\cdot)$ and then removing the vectors that were too long. But, using our incremental approach for computing Markov bases of $\mathcal{F}_A(\cdot)$, we can truncate at each iteration, and thus, in most cases, we never compute the entire Markov basis of $\mathcal{F}_A(\cdot)$.

Example 1.6.3. *Consider again the fiber $\mathcal{F}_A(b) = (\mathcal{P}_A(b) \cap \mathbb{Z}^n)$ from Example 1.6.1. We saw previously that $M'' = \{(1,0),(0,1),(1,1),(\text{-}1,1)\}$ is a Markov basis of $\mathcal{F}_A(\cdot)$. Also, M'' is a truncated Gröbner basis of $\mathcal{F}_A(b)$ and we cannot remove any vectors from M'' since they all fit within the fiber $\mathcal{F}_A(b)$ (see Figure 1.11(a)). In this case, truncation has no effect on the size of the Markov basis.*

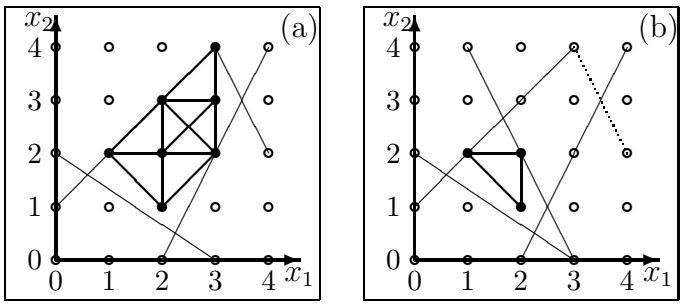

Figure 1.11: The graphs $\mathcal{G}(\mathcal{F}_A(b), M'')$ and $\mathcal{G}(\mathcal{F}_A(b'''), M'')$.

However, suppose that $b''' = (6,\text{-}4,\text{-}6,\text{-}1)$. The fiber $\mathcal{F}_A(b''')$ is depicted in Figure 1.11(b). In this case, the vector $(1,1)$ does not fit within the fiber, and hence, the set $M''' = \{(1,0),(0,1),(\text{-}1,1)\}$ is a truncated Markov basis of $\mathcal{F}_A(b''')$.

Markov basis algorithms can also be applied to the *integer feasibility problem*. In general, integer feasibility problems are as difficult to solve as integer programs. Here, we want a feasible solution of a given fiber. Integer feasibility problems, like integer programs, are very useful in modelling a wide variety of problems such as time-tabling problems. Actually, all Markov basis algorithms that we know of can solve the integer feasibility problem while computing a Markov basis for a very small computational overhead; indeed, the notion of a Markov basis is strongly related to feasibility. This is true of our incremental approach, which as a consequence of computing Markov bases faster than other algorithms, also computes feasible solutions faster than other algorithms. Finding a feasible solution is important for Gröbner basis methods since they require an initial feasible solution as well as a Markov basis.

Markov bases also have a direct analogy in algebraic geometry: a Markov basis corresponds to a generating set of a lattice ideal (see [30]). As with Gröbner bases, we will present the results on Markov bases in an integer programming context. Also, for a very brief formal statement and proof of the result that Markov bases correspond to generating sets of lattice ideas, see Appendix A.

1.7 Outline

In this section, we describe the structure of the thesis and highlight original contributions. The thesis is divided into two main parts. The first part deals with Markov bases and Gröbner bases and their applications (Chapters 3 to 7) and the second part deals with extreme rays of cones and circuits of matrices and their applications (Chapters 8 and 9). These two parts to the thesis are separate and self contained.

Before presenting the two main parts of thesis, we first present the necessary background material in Chapter 2. Here, we formally define and discuss some of the concepts in this introduction and some additional relevant concepts and extend them to lay the framework for the rest of the thesis. The results in the chapter are all well-known or variants of well-known results. We try to present the results in a concise way to facilitate the presentation of results in subsequent chapters and to establish notation for the rest of the thesis. We define and discuss relevant concepts and results for linear spaces, polyhedral cones, convex polyhedra, integer lattices, integer programs, and linear programs. Also, we define and discuss lattice programs, linear space programs, and term orders. A linear space program is a reformulation of a linear program and a lattice program is a reformulation of an integer program; they are extremely useful when presenting the theory of and algorithms for Gröbner bases and Markov bases. A term order is a special total ordering on the set of feasible solutions. We need term orders to compute Gröbner bases and Markov bases of lattice programs.

In the next two chapters, we formally define Markov bases and Gröbner bases. The definitions here differ from the definitions given in this introduction in that we define Markov bases and Gröbner bases in the context of lattice programs. Most of concepts and theory in these chapters are known in some form or another. The aim of these chapters is to bring together theory on Markov bases and Gröbner bases and present it in a coherent form in a integer programming context. Chapter 3 contains formal definitions of Markov bases. In the next chapter, Chapter 4, we describe Gröbner bases of lattice programs and present some novel results on the size of truncated Gröbner bases for equality knapsack problems.

In Chapter 5, we present the completion procedure, which is the basic algorithm for computing Gröbner bases. We present this chapter on computing Gröbner bases before the chapter on computing Markov bases (Chapter 6) because the algorithms for computing Markov bases rely heavily the completion procedure. Most of results and algorithms in this chapter are known, but we do present some results in a lattice programming context that were previously only known in an computational algebraic geometry context. We describe the truncated version of the completion procedure where we describe new ways of improving the performance of truncation. Next, we detail some improvements to the completion procedure. These improvements are mostly known results, although, for some results, this is the first time they have been translated from computational algebraic geometry into an integer programming context. Also, we do present a new approach to optimise the process of finding a reductor in Section 5.3.3, which turns out to be very efficient in practice. At the end of this chapter, we present some alternative approaches to computing Gröbner base.

The next chapter on computing Markov bases contains one of our main contributions. There are three main methods for computing a Markov basis of a lattice: the algorithm of Hosten and Sturmfels in [57] called the "Saturation" algorithm; the algorithm of Bigatti, LaScala, and Robbiano in [12] that we call the "Lift-and-Project" algorithm, and our algorithm ([52]) called the "Project-and-Lift" algorithm as discussed in this introduction. Computationally, the Project-and-Lift algorithm is the fastest of the three algorithms in general ([52]), and we provide computational results that support this claim. The Project-and-Lift algorithm is fundamental to the later algorithms we present to optimise integer programs and to solve the integer feasibility problem. Also, this chapter contains the novel "truncated Project-and-Lift" algorithm. This algorithm is based upon our Project-and-Lift algorithm combined with the truncated completion procedure.

The next chapter focuses on applying the algorithms from the previous chapters. First, we present an application of Markov bases in algebraic statistics in some detail. Using the algorithms that are developed in this thesis for computing Markov bases, we were able to solve an open challenge in algebraic statistics, which we explain in the chapter. Next, we describe an application using Markov bases to show that a semigroup is not normal. Then, we show how we can solve the integer feasibility problem using an extension of the truncated Markov basis algorithm, and we present results of applying this approach to solve equality constrained integer knapsacks ([2]). Then, we present a new algorithm for optimising integer programs, which is also based upon an extension of the truncated Markov basis algorithm, and we present some preliminary but promising results. Lastly, we discuss an approach to the feasibility enumeration problem, which is based upon the approach by Tayur et al. in [83], and we discuss how this method could be used in our optimisation algorithm for integer programs.

The next two chapters constitute part two of the thesis. In Chapter 8, we describe the double-description-method for computing extreme rays of cones ([69, 37]). This method is well-known and we present new optimisations for the method that prove effective in practice. In the next chapter, Chapter 9, we describe how the double-description-method for computing extreme rays of cones can be adapted to compute circuits of matrices. This is well-known ([48, 93, 39, 91]). Then, we present some computational results from using our optimisations for the double-description-method.

Lastly, in Appendix A, we describe the concepts in computational algebraic geometry that correspond to Gröbner bases of integer programs and Markov bases of fibers.

Chapter 2

Foundations

In this chapter, we present fundamental results that are used in the rest of the thesis with the intention of reminding the reader of the result and to establish notation. All the theory presented in this section is well-known or at least readily derivable from well-known theory, so some proofs have been omitted. We have repeated some results here from the introduction for completeness.

We define and discuss relevant concepts and results for linear spaces, polyhedral cones, convex polyhedra, integer lattices, integer programs, and linear programs, linear space programs, lattice programs, and term orders.[1]

2.1 Linear spaces

In this section, we introduce linear spaces. We need the concept of linear spaces when we describe polyhedral cones in the next section and when describing linear space programs in Section 2.8.

The expression $\lambda^1 x^1 + \lambda^2 x^2 + ... + \lambda^k x^k$ where $\lambda^1, \lambda^2, ..., \lambda^k \in \mathbb{R}$ is called a **linear combination** of $x^1, x^2, ..., x^k \in \mathbb{R}^n$.

Definition 2.1.1. *A **linear space** is a set $\mathcal{S} \subseteq \mathbb{R}^n$ that is closed under linear combinations.*

More explicitly, if $x^1, x^2, ..., x^k \in \mathcal{S}$, then $\lambda^1 x^1 + \lambda^2 x^2 + ... + \lambda^k x^k \in \mathcal{S}$ for all $\lambda^1, \lambda^2, ..., \lambda^k \in \mathbb{R}$; in other words, a linear space is a vector subspace of $\mathbb{R}^n$. The set $\mathcal{S}(A) := \{x \in \mathbb{R}^n : Ax = 0\}$ where $A \in \mathbb{R}^{m \times n}$ is a linear space, called a **finitely constrained** linear space. The set $\{\lambda B : \lambda \in \mathbb{R}^k\}$ where $B \in \mathbb{R}^{k \times n}$ is also a linear space, called a **finitely generated** linear space. From linear algebra, we know that every linear space is finitely generated and finitely constrained. We can convert between constrained representations of linear spaces and generator representations of linear spaces using *Gaussian elimination*.

[1] The sections on linear spaces, polyhedral cones, convex polyhedra, and integer lattices follow the approach of Trotter in [90]. Some terminology is non-standard.

Definition 2.1.2. *The set* $\mathcal{S}^* = \{x \in \mathbb{R}^n : xs = 0 \ \forall s \in \mathcal{S}\}$ *is the* ***dual***[2] *of a linear space* $\mathcal{S} \subseteq \mathbb{R}^n$.

The duals of linear spaces are also linear spaces (see Lemma 2.1.3). Given a vector $a \in \mathbb{R}^n$ if $ax = 0$ for all $x \in \mathcal{S}$, we say that $ax = 0$ is a **valid equality** for $\mathcal{S}$. So, the dual space of $\mathcal{S}^*$ precisely determines the set of valid equalities of $\mathcal{S}$.

Lemma 2.1.3. *Let* $\mathcal{S} \subseteq \mathbb{R}^n$ *be a linear space.*

(i). $\mathcal{S}^*$ *is a subspace.*

(ii). $\mathcal{S} = \mathcal{S}^{**}$.

(iii). If $\mathcal{S} = \{x \in \mathbb{R}^n : Ax = 0\}$ *where* $A \in \mathbb{R}^{m \times n}$, *then* $\mathcal{S}^* = \{yA : y \in \mathbb{R}^m\}$.

(iv). If $\mathcal{S} = \{yB : y \in \mathbb{R}^k\}$ *where* $B \in \mathbb{R}^{k \times n}$, *then* $\mathcal{S}^* = \{x \in \mathbb{R}^n : Bx = 0\}$.

If $\mathcal{S}^1$ and $\mathcal{S}^2$ are linear spaces, then $\mathcal{S}^1 \cap \mathcal{S}^2$ and $\mathcal{S}^1 + \mathcal{S}^2 := \{s^1 + s^2 : s^1 \in \mathcal{S}^1, s^2 \in \mathcal{S}^2\}$ are also linear spaces. Moreover, the operations of linear space intersection and summation are in some sense dual operations since $(\mathcal{S}^1 \cap \mathcal{S}^2)^* = (\mathcal{S}^1)^* + (\mathcal{S}^2)^*$ and similarly $(\mathcal{S}^1 + \mathcal{S}^2)^* = (\mathcal{S}^1)^* \cap (\mathcal{S}^2)^*$.

2.2 Polyhedral cones

We describe polyhedral cones in this section in a more formal way than in the introduction. There are two main purposes of this section. The first is establishing the essential equivalence of converting between constraints and generators of cones as we briefly discussed in the introduction. We use the concept of *duality* to establish this. The second purpose is establishing that polyhedral cones are generated by their extreme rays. We also present some crucial result on extreme rays and the facial structure of polyhedral cones.

Definition 2.2.1. *A* ***cone*** *is a set* $\mathcal{C} \subseteq \mathbb{R}^n$ *that is closed under conic combinations.*

A linear space is also by definition a cone. The intersection of two cones is a cone and the summation of two cones is also a cone: given cones $\mathcal{C}^1$ and $\mathcal{C}^2$, the set $\mathcal{C}^1 \cap \mathcal{C}^2$ is a cone and the set $\mathcal{C}^1 + \mathcal{C}^2 = \{x^1 + x^2 : x^1 \in \mathcal{C}^1, x^2 \in \mathcal{C}^2\}$ is also a cone.

The set $\mathcal{C}(A) := \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$ where $A \in \mathbb{R}^{m \times n}$ is a cone, called a **finitely constrained** cone. The set $\{\lambda B : \lambda \in \mathbb{R}^k_+\}$ where $B \in \mathbb{R}^{k \times n}$ is also a cone, called a **finitely generated** cone. The following theorem is one of the most fundamental theorems in linear algebra.

Theorem 2.2.2. *A cone is finitely constrained if and only if it is finitely generated.*

[2]The dual of a linear space is also known as the orthogonal space of the linear space.

Due to the essential equivalence of finitely constrained cones and finitely generated cones, we shall refer to them both as **polyhedral** cones. Also, we call a polyhedral cone rational if the input data (either constraints or generators) is rational. We are only interested in rational polyhedral cones, and it is assumed that all cones are rational polyhedral cones if not explicitly stated.

In this thesis, we are interested in finding a finitely constrained representation of a cone given a finitely generated representation and also conversely in finding a finitely generated representation of a cone given a finitely constrained representation. These two problems of finding the other representation of a cone are really the same problem. We demonstrate this using the concept of conic duality.

Definition 2.2.3. *The set* $\mathcal{C}^* = \{x \in \mathbb{R}^n : xy \geq 0 \ \forall y \in \mathcal{C}\}$ *is the* ***dual*** *of a cone* $\mathcal{C} \subseteq \mathbb{R}^n$.

Given a vector $a \in \mathbb{R}^n$ if $ax \geq 0$ for all $x \in \mathcal{C}$, we say that $ax \geq 0$ is a **valid inequality** for $\mathcal{C}$. So, the dual cone of $\mathcal{C}^*$ determines precisely the set of valid inequalities of $\mathcal{C}$. Recall that a linear subspace $\mathcal{S}$ is also a cone, and in this case, the dual of $\mathcal{S}$ treated as a subspace is identical to the dual of $\mathcal{S}$ treated as a cone, so there is no ambiguity created from using the same notation $(*)$ to indicate conic duality and linear subspace duality.

Lemma 2.2.4. *Let* $\mathcal{C} \subseteq \mathbb{R}^n$ *be a polyhedral cone.*

(i). $\mathcal{C}^*$ *is a polyhedral cone.*

(ii). $\mathcal{C} = \mathcal{C}^{**}$.

(iii). If $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq 0\}$ *where* $A \in \mathbb{R}^{m \times n}$, *then* $\mathcal{C}^* = \{yA : y \in \mathbb{R}^m_+\}$.

(iv). If $\mathcal{C} = \{yB : y \in \mathbb{R}^k_+\}$ *where* $B \in \mathbb{R}^{k \times n}$, *then* $\mathcal{C}^* = \{x \in \mathbb{R}^n : Bx \geq 0\}$.

Note that part (iii) of Proposition 2.2.4 says that if we know a constraint representation of a cone, then we know a generator representation of the dual of the cone. Conversely, part (iv) of Proposition 2.2.4 says that if we know a generator representation of a cone, then we know a constraint representation of the dual of the cone.

As with linear spaces, the operations of conic intersection and summation are also in some sense dual operations since $(\mathcal{C}^1 \cap \mathcal{C}^2)^* = (\mathcal{C}^1)^* + (\mathcal{C}^2)^*$ and similarly $(\mathcal{C}^1 + \mathcal{C}^2)^* = (\mathcal{C}^1)^* \cap (\mathcal{C}^2)^*$ for any two cones $\mathcal{C}^1$ and $\mathcal{C}^2$.

Assume that we know how to convert from a constraint representation of a cone to a generator representation. Now, we wish to convert from a generator representation $\mathcal{C} = \{yB : y \in \mathbb{R}^k_+\}$ where $B \in \mathbb{R}^{k \times n}$ to a constraint representation of C. We know the constraint representation of the dual cone $\mathcal{C}^* = \{x \in \mathbb{R}^k : Bx \geq 0\}$, and we can convert this into a generator representation of the dual cone $\mathcal{C}^* = \{yA : y \in \mathbb{R}^m_+\}$ for some $A \in \mathbb{R}^{m \times n}$. Then, we take the dual of the dual, $\mathcal{C}^{**} = \{x \in \mathbb{R}^n : Ax \geq 0\}$, and since $\mathcal{C}^{**} = \mathcal{C}$, we have thus found a constraint representation of $\mathcal{C}$. Similarly, if we know how to convert from generators to constraints, then we also know how convert

from constraints to generators. Throughout this thesis, we approach this problem from the perspective of converting from constraints to generators.

The **lineality** (subspace) of $\mathcal{C}$, written $\text{lin}(\mathcal{C})$, is the set $\mathcal{C} \cap -\mathcal{C} = \{x : x \in \mathcal{C}, -x \in \mathcal{C}\}$, which is the largest subspace contained within the cone $\mathcal{C}$. If $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq 0\}$, then $\text{lin}(\mathcal{C}) = \{x \in \mathbb{R}^n : Ax = \mathbf{0}\}$. When the lineality of $\mathcal{C}$ is trivial (i.e. $\text{lin}(\mathcal{C}) = \{\mathbf{0}\}$), then we say the cone is **pointed**. Note that $\text{lin}(\mathcal{C}) = \{x \in \mathbb{R}^n : Ax = \mathbf{0}\} = \{\mathbf{0}\}$ if and only if $\text{rank}(A) = n$. Pointed cones have a unique (up to positive scaling) minimal generator representation as we shall see below.

Every cone can be written as the sum of its lineality subspace and a pointed cone.

Lemma 2.2.5. *For a cone $\mathcal{C}$, we have $\mathcal{C} = \mathcal{S} + (\mathcal{C} \cap \mathcal{S}^*)$ where $\mathcal{S} = \text{lin}(\mathcal{C})$ and $(\mathcal{C} \cap \mathcal{S}^*)$ is a pointed cone.*

Lemma 2.2.5 means that we can compute the generator representation of a cone $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$ by separately computing the generators of the cone's linear subspace $\mathcal{S} = \text{lin}(\mathcal{C}) = \{x \in \mathbb{R}^n : Ax = \mathbf{0}\}$ and the generators of the pointed cone $(\mathcal{C} \cap \mathcal{S}^*)$. We can compute the generators of the linear space $\mathcal{S}$ from the constraint representation of $\mathcal{S}$: $\mathcal{S} = \{\lambda B : \lambda \in \mathbb{R}^k\}$ for some $B \in \mathbb{R}^{k \times n}$. Hence, $\mathcal{S}^* = \{x \in \mathbb{R}^n : Bx = \mathbf{0}\}$, and therefore, we know a constraint representation of the pointed cone $(\mathcal{C} \cap \mathcal{S}^*)$: $(\mathcal{C} \cap \mathcal{S}^*) = \{x \in \mathbb{R}^n : Bx = \mathbf{0}, Ax \geq \mathbf{0}\}$. We then compute the generators of the pointed cone $(\mathcal{C} \cap \mathcal{S}^*)$: $(\mathcal{C} \cap \mathcal{S}^*) = \{\delta D : \delta \in \mathbb{R}^l_+\}$ for some $D \in \mathbb{R}^{l \times n}$. Therefore, $\mathcal{C} = \mathcal{S} + (\mathcal{C} \cap \mathcal{S}^*) = \{\lambda B + \delta D : \lambda \in \mathbb{R}^k, \delta \in \mathbb{R}^l_+\}$. So, Lemma 2.2.5 is important because it means that, in this thesis, we can focus on pointed cones.

Often, we wish to compute generators of a cone given in the form

$$\mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$$

where $A \in \mathbb{R}^{m \times n}$, $\sigma \subseteq \{1, ..., m\}$ and $\bar{\sigma} = \{1, ..., m\} \setminus \sigma$. Note that A_σ is the submatrix of A consisting of the rows indexed by σ, and $A_{\bar{\sigma}}$ is the submatrix of A consisting of the rows of A indexed by $\bar{\sigma}$; that is, we have partitioned the rows of the matrix A into two submatrices A_σ and $A_{\bar{\sigma}}$. So, σ refers to the rows of the matrix A that are inequality constraints, and $\bar{\sigma}$ refers to the rows of the matrix A that are equality constraints. The cone $\mathcal{C}_\sigma(A)$ is thus the intersection of a linear space $\mathcal{S}(A_{\bar{\sigma}}) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}\}$ and the cone $\mathcal{C}(A_\sigma) := \{x \in \mathbb{R}^n : A_\sigma x \geq \mathbf{0}\}$. The dual of the cone $\mathcal{C}_\sigma(A) = \mathcal{S}(A_{\bar{\sigma}}) \cap \mathcal{C}(A_\sigma)$ is the cone $\mathcal{C}_\sigma(A)^* = \mathcal{S}(A_{\bar{\sigma}})^* + \mathcal{C}(A_\sigma)^*$, and since $\mathcal{S}(A_{\bar{\sigma}})^* = \{yA_{\bar{\sigma}} : y \in \mathbb{R}^{|\bar{\sigma}|}\}$ and $\mathcal{C}(A_\sigma)^* = \{yA_\sigma : y \in \mathbb{R}^{|\sigma|}_+\}$, we have $\mathcal{C}_\sigma(A)^* = \{yA : y_\sigma \geq \mathbf{0}, y \in \mathbb{R}^m\}$. We find the form $\mathcal{C}_\sigma(A)$ convenient for discussing faces of cones below.

We can reformulate the cone $\mathcal{C}_\sigma(A)$ in the form $\mathcal{C}(\tilde{A})$ for some matrix $\tilde{A}$ by replacing each equality constraint with two inequalities:

$$\mathcal{C}_\sigma(A) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x \geq \mathbf{0}, A_{\bar{\sigma}} x \leq \mathbf{0}, A_\sigma x \geq \mathbf{0}\}.$$

However, in general, it is *not* a good idea to perform this transformation as doing so might adversely affect the algorithms to compute the generators of the cone. This is certainly true for the algorithm that we discuss in this thesis. There are other

possible transformations that do not adversely affect the algorithm, and we discuss these possible reformulations of the cone $\mathcal{C}_\sigma(A)$ in the chapter on computing extreme rays of cones (Section 8.2.3).

We now discuss the facial structure of polyhedral cones.

Definition 2.2.6. *Given a cone $\mathcal{C}$ and a vector $a \in \mathcal{C}^*$, the set $F := \{x \in \mathcal{C} : ax = 0\}$ is called a **face** of $\mathcal{C}$.*

A face of a cone is itself a cone. The cone $\mathcal{C}$ is a face of itself, and the unique smallest inclusion-wise face of $\mathcal{C}$ is the lineality subspace $\text{lin}(\mathcal{C})$. We call a face F of a cone $\mathcal{C}$ a **proper face** if $F \neq \mathcal{C}$ and $F \neq \text{lin}(\mathcal{C})$. If the dimension of a face F of the cone $\mathcal{C}$ is one less than the dimension of the cone $\mathcal{C}$ ($\dim(F) = \dim(\mathcal{C}) - 1$), then we call F a **facet** of $\mathcal{C}$. Note that the dimension of a cone $\mathcal{C}$ is the number of linearly independent points in $\mathcal{C}$, or equivalently, the dimension of $\mathcal{C}$ is the dimension of the linear space it spans (i.e. the smallest linear space containing the cone).

Theorem 2.2.7. *Given a set $F \subseteq \mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$, F is a face of $\mathcal{C}_\sigma(A)$ if and only if $F = \mathcal{C}_\tau(A) = \{x \in \mathbb{R}^n : A_{\bar{\tau}}x = \mathbf{0}, A_\tau x \geq \mathbf{0}\}$ for some set $\tau \subseteq \sigma$.*

From Theorem 2.2.7, we can immediately deduce that there are a finite number of faces of a cone and given two faces F^1 and F^2 of a cone $\mathcal{C}_\sigma(A)$, $F^1 \cap F^2$ is also a face of $\mathcal{C}_\sigma(A)$. However, it is not necessarily true that $F^1 + F^2$ is also a face of $\mathcal{C}_\sigma(A)$.

Definition 2.2.8. *Given a pointed cone $\mathcal{C}$, we call any vector $r \in \mathcal{C}$ where $r \neq \mathbf{0}$ a **ray** of $\mathcal{C}$. Also, we call a ray $r \in \mathcal{C}$ an **extreme ray** of $\mathcal{C}$ if the set $\{\lambda r : \lambda \in \mathbb{R}^n_+\}$ is a one-dimensional face of $\mathcal{C}$.*

The extreme rays of a cone $\mathcal{C}$ are unique up to positive scaling, and we shall treat two extreme rays that differ by positive scaling as the same extreme ray. The result that we are working towards is that a cone is generated by its extreme rays. Thus, the algorithms that we present for computing a generator representation of a cone given a constraint representation compute the extreme rays of a cone.

Definition 2.2.9. *Given a ray $r \in \mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$, we define the **support** of the ray r as the set $\text{supp}_A(r) := \{i \in \{1, ..., m\} : A_i r \neq 0\}$.*

Observe that $\text{supp}_A(r) \subseteq \sigma$ for a ray $r \in \mathcal{C}_\sigma(A)$ since $A_i x = \mathbf{0}$ for all $i \in \bar{\sigma}$. It follows from Theorem 2.2.7 that, for a given ray r of a cone $\mathcal{C}_\sigma(A)$, the face $F := \mathcal{C}_\tau(A) = \{x \in \mathbb{R}^n : A_{\bar{\tau}}x = \mathbf{0}, A_\tau x \geq \mathbf{0}\}$ where $\tau = \text{supp}_A(r)$, is the inclusion-minimal face of $\mathcal{C}_\sigma(A)$ containing r. Moreover, if r is an extreme ray of $\mathcal{C}_\sigma(A)$, then $F = R = \{\lambda r : \lambda \in \mathbb{R}^n_+\}$. Thus, we arrive at the following useful lemma below that gives a geometric characterisation of extreme rays.

Lemma 2.2.10. *Given a pointed cone $\mathcal{C}_\sigma(A)$, a ray $r \in \mathcal{C}_\sigma(A)$ is an extreme ray of $\mathcal{C}_\sigma(A)$ if and only if $\dim(F) = 1$ where $F = \mathcal{C}_\tau(A)$ and $\tau = \text{supp}_A(r)$.*

We can compute the dimension of the inclusion-minimal face of $\mathcal{C}_\sigma(A)$ containing a ray r as follows.

Lemma 2.2.11. *Let r be a ray of a cone $\mathcal{C}_\sigma(A)$ and let $F = \mathcal{C}_\tau(A)$ where $\tau = \mathrm{supp}_A(r)$. Then, $\dim(F) = n - \mathrm{rank}(A_{\bar{\tau}})$.*

Proof. Let $\bar{F} = \{x \in \mathbb{R}^n : A_{\bar{\tau}}x = 0\}$. Note that $r \in \bar{F}$. The set $\bar{F}$ is the inclusion-minimal linear space containing F. It has the same dimension as F. The dimension of the linear space $\bar{F}$ is $n - \mathrm{rank}(A_{\bar{\tau}})$. □

Combining Lemma 2.2.11 and Lemma 2.2.10, we arrive at the following corollary.

Corollary 2.2.12. *Given a pointed cone $\mathcal{C}_\sigma(A)$, a ray $r \in \mathcal{C}_\sigma(A)$ is an extreme ray of $\mathcal{C}_\sigma(A)$ if and only if $n - \mathrm{rank}(A_{\bar{\tau}}) = 1$ where $\tau = \mathrm{supp}_A(r)$.*

The following lemma gives a combinatorial characterisation of extreme rays. First, note that a ray $r \in \mathcal{C}_\sigma(A)$ is a support-minimal ray of $\mathcal{C}_\sigma(A)$ if there does not exists another ray $r' \in \mathcal{C}_\sigma(A)$ such that $\mathrm{supp}_A(r') \subsetneq \mathrm{supp}_A(r)$.

Lemma 2.2.13. *The extreme rays of a pointed cone $\mathcal{C}_\sigma(A)$ are the support-minimal rays of $\mathcal{C}_\sigma(A)$.*

Proof. Let $r \in \mathcal{C}_\sigma(A)$ be an extreme ray of $\mathcal{C}_\sigma(A)$, and let r' be a ray of $\mathcal{C}_\sigma(A)$ such that $\mathrm{supp}_A(r') \subsetneq \mathrm{supp}_A(r)$. Let $F = \mathcal{C}_\tau(A)$ where $\tau = \mathrm{supp}_A(r)$. Since $\mathrm{supp}_A(r') \subsetneq \mathrm{supp}_A(r)$, we must have $r' \subseteq F$. Also, $F = \{\lambda r : \lambda \in \mathbb{R}^n_+\}$ because r is an extreme ray of $\mathcal{C}_\sigma(A)$. Thus, $r' = \lambda r$ for some $\lambda \in \mathbb{R}^n_+$. Therefore, $\mathrm{supp}_A(r') = \mathrm{supp}_A(r)$, which is a contradiction. □

The next corollary follows from the proof of the previous lemma. It says that we can uniquely identify extreme rays by their support. Recall that two extreme rays that differ by positive scaling are considered as the same extreme ray.

Corollary 2.2.14. *Let r^1 and r^2 be two extreme rays of a pointed cone $\mathcal{C}_\sigma(A)$. If $\mathrm{supp}_A(r^1) \subseteq \mathrm{supp}_A(r^2)$, then $r^1 = \lambda r^2$ for some $\lambda \in \mathbb{R}_+$.*

An extreme ray r of a pointed cone $\mathcal{C}_\sigma(A)$ is also an extreme ray of any face F where $r \in F$. This follows since an extreme ray is support-minimal in $\mathcal{C}_\sigma(A)$ from Lemma 2.2.13 and thus also support-minimal in F from Theorem 2.2.7. Moreover, an extreme ray of a face F of $\mathcal{C}_\sigma(A)$ is also an extreme ray of $\mathcal{C}_\sigma(A)$ since a one-dimensional face of F is also a one-dimensional face of $\mathcal{C}_\sigma(A)$.

The following result is fundamental for computing generators of cones. It says that every pointed cone is generated by its extreme rays. We need at least all the extreme rays of a pointed cone as generators of a pointed cone, but the fact that they are sufficient to generate the cone is not immediate.

Lemma 2.2.15. *For every pointed cone $\mathcal{C}$, we have $\mathcal{C} = \{\lambda B : \lambda \in \mathbb{R}^n_+\}$ where the rows of B are the extreme rays of $\mathcal{C}$.*

Proof. From Theorem 2.2.2, $\mathcal{C} = \mathcal{C}_\sigma(A)$ for some matrix A and some set σ. Let $\mathcal{C}' = \{\lambda B : \lambda \in \mathbb{R}^n_+\}$; then, $\mathcal{C}' \subseteq \mathcal{C}$. Assume that $\mathcal{C} \neq \mathcal{C}'$. Choose a ray $r \in \mathcal{C} \setminus \mathcal{C}'$ with minimal-support in $\mathcal{C} \setminus \mathcal{C}'$. The ray r cannot be an extreme ray otherwise $r \in \mathcal{C}'$.

There must exists an extreme ray r' such that $\text{supp}_A(r') \subseteq \text{supp}_A(r)$ by Lemma 2.2.13. Let $s = Ar$ and $s' = Ar'$, and let $\lambda = \min\{\frac{s_i}{s'_i} : i \in \text{supp}_A(r')\}$. Then, $r - \lambda r' \in \mathcal{C}$ and $\text{supp}_A(r - \lambda r') \subsetneq \text{supp}_A(r)$ by construction. Also, we must have that $r - \lambda r' \notin \mathcal{C}'$ otherwise $r \in \mathcal{C}'$, but this contradicts the support-minimal assumption on r since $\text{supp}_A(r - \lambda r') \subsetneq \text{supp}_A(r)$. □

The set of extreme rays of a pointed cone is a unique (up to positive scaling) inclusion-minimal generating set of the cone since we must have all the extreme rays of a cone in any generating set and Lemma 2.2.15 shows that the extreme rays are actually all we need.

2.3 Convex polyhedra

We discuss convex polyhedra in this section. The main purpose of this section is establishing that we can convert between different representations of a polyhedron (i.e., constraint to generator representation or generator to constraint representation) using an algorithm for converting between different representations of a polyhedral cone as discussed in the previous section.

Definition 2.3.1. *A **convex set** is a set $\mathcal{K} \subseteq \mathbb{R}^n$ that is closed under convex combinations.*

Note that cones are convex sets.

The set $\mathcal{P}_A(b) := \{x \in \mathbb{R}^n : Ax \geq b\}$ where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ is a convex set, and we call the set $\mathcal{P}_A(b)$ a **finitely constrained** convex set. The set $\{yB + zC : \sum_{i=1}^{q} z_i = 1, y \in \mathbb{R}^p_+, z \in \mathbb{R}^q_+\}$ where $B \in \mathbb{R}^{p \times n}$ and $C \in \mathbb{R}^{q \times n}$ is also a convex set, called a **finitely generated** convex set.

Analogously to Theorem 2.2.2 for cones, we have the following inhomogeneous version.

Theorem 2.3.2. *A convex set is finitely constrained if and only if it is finitely generated.*

We call a convex set of points $\mathcal{P} \subseteq \mathbb{R}^n$ a **convex polyhedron** if it is finitely constrained or equivalently if it is finitely generated. We only deal with convex polyhedra, so we usually omit the term convex. Note that a polyhedral cone is a polyhedron. In this thesis, we only deal with rational polyhedra, so when we say a polyhedron, we mean a rational polyhedron.

Often, we write a polyhedron in the form $\mathcal{P}^{\sigma}_A(b) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_{\sigma}x \geq b_{\sigma}\}$. Recall that $A_{\bar{\sigma}}$ is the submatrix of A consisting of the rows of A indexed by $\bar{\sigma}$ and A_{σ} is the submatrix of A consisting of the rows of A indexed by σ. Also, $b_{\bar{\sigma}}$ is the vector b projected onto the $\bar{\sigma}$ components and b_{σ} is the vector b projected onto the σ components. We could remove the equality constraints by replacing $A_{\bar{\sigma}}x = b_{\bar{\sigma}}$ with $A_{\bar{\sigma}}x \geq b_{\bar{\sigma}}$ and $A_{\bar{\sigma}}x \leq b_{\bar{\sigma}}$, but this is not a good transformation as it can adversely affects algorithms for converting between a constraint representation to a generator representation of the polyhedron.

A polyhedron $\mathcal{P}$ is *bounded* if every coordinate in has a maximum and minimum value in $\mathcal{P}$: that is, if $\mathcal{P} \subseteq \{x \in \mathbb{R}^n : -\omega \leq x_i \leq \omega, \forall i = 1, ..., n\}$ for some $\omega \in \mathbb{R}_+$. If a polyhedron is *bounded*, then we call it a **polytope**. The **recession cone** of a polyhedron $\mathcal{P}$ is the set of all vectors $r \in \mathbb{R}^n$ such that for every $x \in \mathcal{P}$, $x + \lambda r \in \mathcal{P}$ for all $\lambda \in \mathbb{R}_+$. It follows that a polyhedron $\mathcal{P}$ is bounded if and only if its recession cone $\text{rec}(P)$ is trivial (i.e. $\text{rec}(P) = \{\mathbf{0}\}$). Specifically, the recession cone of $\mathcal{P}_A^\sigma(b)$ is the cone $\text{rec}(\mathcal{P}_A^\sigma(b)) := \mathcal{P}_A^\sigma(\mathbf{0}) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\} = \mathcal{C}_\sigma(A)$. Also, the recession cone of the polyhedron $\mathcal{P} = \{yB + zC : \sum_{i=1}^q z_i = 1, y \in \mathbb{R}_+^p, z \in \mathbb{R}_+^q\}$ is the cone $\text{rec}(\mathcal{P}) = \{yB : y \in \mathbb{R}_+^p\}$. The polyhedron $\mathcal{P}' = \{zC : \sum_{i=1}^q z_i = 1, z \in \mathbb{R}_+^q\}$ is bounded, so we can formulate the polyhedron $\mathcal{P}$ as the sum of its recession cone and a bounded polyhedron $\mathcal{P}'$; that is, $\mathcal{P} = \text{rec}(\mathcal{P}) + \mathcal{P}'$. This applies to any polyhedron.

We say that a polyhedron $\mathcal{P}$ is **pointed** if the polyhedron contains no lines; in other words, $\mathcal{P}$ is pointed if there does not exist $x \in \mathcal{P}$ such that $x + \lambda r \in \mathcal{P}$ for all $\lambda \in \mathbb{R}$. Note that if $x + \lambda r \in \mathcal{P}$ for all $\lambda \in \mathbb{R}$, then $\lambda r \in \text{rec}(\mathcal{P})$ for all $\lambda \in \mathbb{R}$, in which case, the recession cone is not pointed. Conversely, if $\text{rec}(\mathcal{P})$ is not pointed, then neither is $\mathcal{P}$. So, a polyhedron $\mathcal{P}$ is pointed if and only if $\text{rec}(\mathcal{P})$ is pointed.

We now introduce the notions of faces of polyhedra. First, we need the concept of a valid inequality. The inequality $\pi x \leq \pi_0$ is called a **valid inequality** for $\mathcal{P}$ if $\pi x \leq \pi_0$ for every $x \in \mathcal{P}$.

Definition 2.3.3. *Given a polyhedron $\mathcal{P}$ and a valid inequality $\pi x \leq \pi_0$ for $\mathcal{P}$, the set $F := \{x \in \mathcal{P} : \pi x = \pi_0\}$ is called a face of $\mathcal{P}$.*

A face of a polyhedron is itself a polyhedron. The polyhedron $\mathcal{P}$ is by definition a face of itself, and the unique inclusion-minimal face of $\mathcal{P}$ is $F = \{x \in \mathbb{R}^n : Ax = b\}$, which might be empty when $b \neq \mathbf{0}$ or trivial when $b = \mathbf{0}$ (i.e. $F = \{\mathbf{0}\}$).

If $\mathcal{P}$ is a d-dimensional polyhedron, and $F \subseteq \mathcal{P}$ a $d-1$-dimensional face of $\mathcal{P}$, then we call F a facet of $\mathcal{P}$. If F is a zero-dimensional face of $\mathcal{P}$ (i.e. $F = \{x\}$), then we call F (or just x) an *extreme point* or *vertex* of $\mathcal{P}$. Note that a polyhedron only has 0-dimensional faces if $\mathcal{P}$ is pointed.

This definition of a faces of a polyhedron coincides exactly with the definition of a face of a cone when considering a cone as a polyhedron, and so, there is no ambiguity in using the same terminology.

Theorem 2.3.4. *Given a set $F \subseteq \mathcal{P}_A^\sigma(b) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x \geq b_\sigma\}$, F is a face of $\mathcal{P}_A^\sigma(b)$ if and only if $F = \mathcal{P}_A^\tau(b) = \{x \in \mathbb{R}^n : A_{\bar{\tau}}x = b_{\bar{\tau}}, A_\tau x \geq b_\tau\}$ for some set $\tau \subseteq \sigma$.*

From Theorem 2.3.4, we can immediately deduce that there are a finite number of faces of a polyhedron and given two faces F^1 and F^2 of a polyhedron $\mathcal{P}_A^\sigma(b)$, $F^1 \cap F^2$ is also a face of $\mathcal{P}_A^\sigma(b)$.

We wish to convert between different representations of a polyhedron: to convert from a constraint representation to a generator representation and to convert from a generator representation to a constraint representation. Fortunately, this problem of converting between different representations of a polyhedron can be reduced to the problem of converting between different representations of cones. To achieve this, we

embed the polyhedron in a cone and perform the conversion from one representation of a cone to another and then extract the polyhedron from the cone in its converted representation.

First, we show how to convert from constraints to generators. Consider the polyhedron $\mathcal{P}_A^\sigma(b) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x \geq b_\sigma\}$. The set

$$\mathcal{C} = \{(x, x_{n+1}) \in \mathbb{R}^{n+1} : A_{\bar{\sigma}}x - b_{\bar{\sigma}}x_{n+1} = \mathbf{0}, A_\sigma x - b_\sigma x_{n+1} \geq \mathbf{0}, x_{n+1} \geq 0\}$$

is a finitely constrained cone, and $\mathcal{P}_A^\sigma(b) = \{x \in \mathbb{R}^n : (x, 1) \in \mathcal{C}\}$. By theorem 2.2.2, $\mathcal{C} = \{\lambda D : \lambda \in \mathbb{R}_+^k\}$ for some matrix $D \in \mathbb{R}^{k\times(n+1)}$, and we can compute D using an algorithm to convert from constraints to generators of cones. Note the D is the set of extreme rays of $\mathcal{C}$ assuming $\mathcal{C}$ is pointed, which it is if and only if $\mathcal{P}$ is pointed. After scaling and re-arranging the rows of D so that the entries in the last column of D are 0 and 1 and the rows with 0 entries come first, we may write

$$D = \begin{bmatrix} B & \mathbf{0} \\ C & \mathbf{1} \end{bmatrix}$$

where $B \in \mathbb{R}^{p\times n}$ and $C \in \mathbb{R}^{q\times n}$. Then, $\mathcal{C} = \{(yB, 0) + (zC, \mathbf{1}z) : y \in \mathbb{R}_+^p, z \in \mathbb{R}_+^q\}$, and therefore, $\mathcal{P} = \{x \in \mathbb{R}^n : (x, 1) \in \mathcal{C}\} = \{yB + zC : \sum_{i=1}^q z_i = 1, y \in \mathbb{R}_+^p, z \in \mathbb{R}_+^q\}$. Recall that $\{yB : y \in \mathbb{R}_+^p\}$ is the recession cone of $\mathcal{P}$, and thus, B is the set of extreme rays of $\text{rec}(\mathcal{P})$ assuming $\mathcal{P}$ is pointed. Also, the vector $(c, 1)$ where c is a row of C is an extreme ray of $\mathcal{C}$ since D is the set of extreme rays of $\mathcal{C}$ and $(c, 1)$ is a row of D. Then, assuming that $\mathcal{P}$ is pointed, $\mathcal{C}$ is pointed, and $F = \{\lambda(c, 1) : \lambda \in \mathbb{R}_+\}$ is a one-dimensional face of $\mathcal{C}$, which implies that $(c, 1)$ is a zero-dimensional face of the polyheron $\mathcal{C} \cap \{(x, 1) : x \in \mathbb{R}^n\}$. Thus, $c \in \mathcal{P}$ is also a zero-dimensional face of $\mathcal{P}$, so c is an extreme point of $\mathcal{P}$. So, if $\mathcal{P}$ is pointed, then it is generated by conic combinations of the extreme rays of its recession cone plus convex combinations of its extreme points. Moreover, from the above construction, since the extreme rays of a cone are the unique minimal generators of the cone, the extreme points and extreme rays are the unique minimal generators of the polyhedron.

Conversely, let $\mathcal{P} = \{yB + zC : \sum_{i=1}^q z_i = 1, y \in \mathbb{R}_+^p, z \in \mathbb{R}_+^q\}$. Then, the cone $\mathcal{C} = \{(yB + zC, \mathbf{1}z) : y \in \mathbb{R}_+^p, z \in \mathbb{R}_+^q\}$ is a finitely generated cone, and we have $\mathcal{P} = \{x \in \mathbb{R}^n : (x, 1) \in \mathcal{C}\}$. By theorem 2.2.2, $\mathcal{C} = \{(x, x_{n+1}) \in \mathbb{R}^{n+1} : D(x, x_{n+1}) \geq \mathbf{0}\}$ for some matrix $D \in \mathbb{R}^{m\times(n+1)}$, and we can compute D using an algorithm to convert from generators to constraints of cones, which is the essentially the same algorithm as converting from constraints to generators. We may write $D = \begin{bmatrix} A & -b \end{bmatrix}$ where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Therefore, $\mathcal{P} = \{x \in \mathbb{R}^n : (x, 1) \in \mathcal{C}\} = \{x \in \mathbb{R}^n : Ax \geq b\}$.

We can thus convert between representations of polyhedra using an algorithm to convert between representations of cones. Note that we have just effectively proven theorem 2.3.2.

2.4 Integer lattices

Integer lattices play a crucial role in describing Gröbner bases and Markov bases. In particular, the *Hermite Normal Form* algorithm presented here, which computes

bases of lattices and solves linear integer equality systems, is frequently used in the rest of this thesis.

An **integer lattice** is a set $\mathcal{L} \subseteq \mathbb{Z}^n$ which is closed under addition, which implies that $\mathcal{L}$ is also closed under multiplication by an integral scalar.[3] So, if $x^1, ..., x^k \in \mathcal{L}$, then $\lambda^1 x^1 + ... + \lambda^k x^k \in \mathcal{L}$ for all $\lambda^1, ..., \lambda^k \in \mathbb{Z}$. In other words, a integer lattice is a $\mathbb{Z}$-submodule of $\mathbb{Z}^n$. We are only concerned with integer lattices, so for brevity, we just say lattice when we mean an integer lattice.

Given an integer matrix $B \in \mathbb{Z}^{k \times n}$, the set $\{\lambda B : \lambda \in \mathbb{Z}^k\}$ is a lattice, called a **finitely generated** lattice. In this case, we say that B *spans* $\mathcal{L}$, and if $\text{rank}(B) = k$ (full row rank), then we say that B is a *basis* of $\mathcal{L}$.

Lemma 2.4.1. *Every lattice is a finitely generated lattice, and every lattice has a basis.*

The set of integer points in a linear space is a lattice: the set $\mathcal{L} = \{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$ where $A \in \mathbb{R}^{k \times n}$ is a lattice since such a set is closed under addition. If a lattice can be represented in such a form, then we say it is **saturated**. However, not all lattices $\mathcal{L} \in \mathbb{Z}^n$ can be written in the form $\mathcal{L} = \{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$ for some matrix A. The reason being is that if $\mathcal{L} = \{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$, then for all $x \in \mathcal{L}$, we must also have $\lambda x \in \mathcal{L}$ for all $\lambda \in \mathbb{R}^n$ where $\lambda x \in \mathbb{Z}^n$. However, this property is not true for every lattice. Indeed, consider the lattice finitely generated by the single vector $(2, \text{-}2)$: $\mathcal{L} = \{\lambda(2, \text{-}2) : \lambda \in \mathbb{Z}\}$. If $\mathcal{L}$ were saturated, then $(1, \text{-}1)$ would be in $\mathcal{L}$, but it is not.

Every lattice is not saturated, but every lattice is the projection of some saturated lattice. Let $\mathcal{L} \subseteq \mathbb{Z}^n$ be a lattice, and let $B \in \mathbb{Z}^{k \times n}$ be a basis of $\mathcal{L}$, so we have $\mathcal{L} = \{\lambda B : \lambda \in \mathbb{Z}^k\}$. Then, let $\tilde{\mathcal{L}} = \{(x, \lambda) : x - \lambda B = \mathbf{0}, \lambda \in \mathbb{Z}^k\}$, so $\tilde{\mathcal{L}}$ is a saturated lattice and (B, I) is a basis of $\tilde{\mathcal{L}}$. Note that $\mathcal{L} = \{x : (x, \lambda) \in \tilde{\mathcal{L}}\}$, so $\mathcal{L}$ is the projection of $\tilde{\mathcal{L}}$ onto the x variables. For example, the lattice generated by the vector $(2, \text{-}2)$ is the projection of the saturated lattice generated by $(2, \text{-}2, 1)$ projected onto the first two components.

There are two related lattice problems that we frequently need to solve in this thesis. One is the problem of finding a feasible solution or showing infeasibility of the set $\{x \in \mathbb{Z}^n : Ax = b\}$ for some matrix $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. Another problem is finding a basis of the lattice $\mathcal{L} = \{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$. We now show how to solve these problems using the Hermite Normal Form algorithm.

First, we must introduce the concept of *unimodular* matrices.

Definition 2.4.2. *An integer matrix $C \in \mathbb{Z}^{n \times n}$ is **unimodular** if $|\det(C)| = 1$.*

A unimodular matrix C has some very useful properties. The inverse of a unimodular matrix $C^{\text{-}1}$ is also unimodular. Also, $Cx \in \mathbb{Z}^n$ if and only if $x \in \mathbb{Z}^n$. If $x \in \mathbb{Z}^n$, then $Cx \in \mathbb{Z}^n$ because C is integral and if $Cx \in \mathbb{Z}^n$, then $C^{\text{-}1}Cx = x \in \mathbb{Z}^n$ since $C^{\text{-}1}$ is integral. Moreover, for every $x \in \mathbb{Z}^n$, there exists $y \in \mathbb{Z}^n$ such that $Cy = x$ by taking $y = C^{\text{-}1}x$. So, a unimodular matrix induces a one-to-one map from $\mathbb{Z}^n$ to $\mathbb{Z}^n$.

The next concept we need is that of matrices is *Hermite Normal Form*.

[3] More general definitions of lattices exist, but for our purposes, this definition will suffice.

Definition 2.4.3. *An n by n non-singular matrix H is said to be in* ***Hermite Normal Form*** *(HNF) if*

(i). H is lower triangle (i.e. $H_{ij} = 0$ for $j > i$),

(ii). $H_{ii} > 0$ for $i = 1, ..., n$, and

(iii). $H_{ij} \leq 0$ and $|H_{ij}| < H_{ii}$ for $j < i$.

For example, the matrix $H = \begin{pmatrix} 1 & 0 \\ -3 & 5 \end{pmatrix}$ is in Hermite Normal Form.

The next lemma is a fundamental result.

Lemma 2.4.4. *Let $A \in \mathbb{Z}^{m \times n}$ where* rank$(A) = m$. *There exists a unimodular matrix $C \in \mathbb{Z}^{n \times n}$ such that $AC = (H, \mathbf{0})$ where $H \in \mathbb{Z}^{m \times m}$ is in Hermite Normal Form and $H^{-1}A$ is an integer matrix.*

The matrices H and C given above in Lemma 2.4.4 can be computed using the *Hermite Normal Form algorithm*. We refer the reader to [71] for a description of the basic algorithm. There are polynomial time algorithms to compute C and H (see for example [77]).

Let $A \in \mathbb{Z}^{m \times n}$ where rank$(A) = m$. Let $C \in \mathbb{Z}^{n \times n}$ be a unimodular matrix such that $AC = (H, \mathbf{0})$ where $H \in \mathbb{Z}^{m \times m}$ is in Hermite Normal Form and $H^{-1}A$ is an integer matrix as in Lemma 2.4.4. Also, we write $C = (C_1, C_2)$ where $C_1 \in \mathbb{Z}^{n \times m}$ and $C_2 \in \mathbb{Z}^{n \times n-m}$. We now reformulate the set $\{x \in \mathbb{Z}^n : Ax = b\}$ into an equivalent and useful form that enables us to find a feasible solution of $\{x \in \mathbb{Z}^n : Ax = b\}$ and to find a basis for the lattice $\{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$.

$$\begin{aligned}
\{x \in \mathbb{Z}^n : Ax = b\} =& \{x : x = Cw, ACw = b, w \in \mathbb{Z}^n\} \\
=& \{x : x = Cw, (H, \mathbf{0})w = b, w \in \mathbb{Z}^n\} \\
=& \{x : x = C_1 w_1 + C_2 w_2, Hw_1 = b, w_1 \in \mathbb{Z}^m, w_2 \in \mathbb{Z}^{n-m}\} \\
=& \{x : x = C_1 H^{-1} b + C_2 w_2, H^{-1} b \in \mathbb{Z}^m, w_2 \in \mathbb{Z}^{n-m}\}.
\end{aligned}$$

This implies that $\{x \in \mathbb{Z}^n : Ax = b\} \neq \emptyset$ if and only if $H^{-1}b \in \mathbb{Z}^m$. Moreover, if $H^{-1}b \in \mathbb{Z}^m$, then $x = C_1 H^{-1} b \in \mathbb{Z}^n$ is a feasible solution of $\{x \in \mathbb{Z}^n : Ax = b\}$. Furthermore, when $b = \mathbf{0}$, $\{x \in \mathbb{Z}^n : Ax = \mathbf{0}\} = \{x : x = C_2 w_2, w_2 \in \mathbb{Z}^{n-m}\}$, so the set $C_2^{\intercal}$ (the transpose of C_2) is a generating set of the following lattice: $\{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$. Also, since rank$(C) = n$, rank$(C_2^{\intercal}) = n - m$ (i.e. $C_2^{\intercal}$ is full row rank); therefore, $C_2^{\intercal}$ is a basis of $\{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$.

Lastly, we consider different possible ways of representing a given lattice using a set of generators. We show that if B generates the lattice $\mathcal{L}$ and C is a unimodular matrix, then CB also generates $\mathcal{L}$. Firstly, by definition $\mathcal{L} = \{\lambda B : \lambda \in \mathbb{Z}^k\}$, and, from the properties of unimodular matrices ($\mathbb{Z}^n = \{\delta C : \delta \in \mathbb{Z}^k\}$), we can substitute δC for λ giving $\mathcal{L} = \{\delta CB : \delta \in \mathbb{Z}^k\}$.

Consider the special case where $B \in \mathbb{Z}^{k \times k}$ and rank$(B) = k$. The matrix B is a basis of the lattice $\mathcal{L} = \{\lambda B : \lambda \in \mathbb{Z}^k\}$. Using the HNF algorithm we can compute

a unimodular matrix C such that $B^\intercal C = H$ or equivalently $C^\intercal B = H^\intercal$. Thus, $C^\intercal B = H^\intercal$ is also a basis of $\mathcal{L}$ since $C^\intercal$ (the transpose of C) is also unimodular. Note that the matrix $H^\intercal$ (the transpose of H) is an upper triangle matrix with positive diagonal entries and non-positive entries elsewhere. Also, H is unique and so is $H^\intercal$. We say that any matrix whose transpose is in Hermite Normal Form is in **Upper Hermite Normal Form** (UHNF). So, we can always find a basis in Upper Hermite Normal Form for any lattice that has a square matrix for a basis; this fact is often used in this thesis.

2.5 Gordan-Dickson's lemma

In this section, we briefly state Gordan-Dickson's lemma. This result is very useful and we refer to it and its variations several times throughout the thesis.

Lemma 2.5.1. *For any set $S \subseteq \mathbb{N}^n$, there exists a unique inclusion-minimal finite subset $T \subseteq S$ such that $T \leq S$, or more explicitly, for every $s \in S$, there exists $t \in T$ such that $t \leq s$.*

A useful consequence of Gordan-Dickson's lemma is that given a sequence of points $(x^1, x^2, ..., x^k, ...)$, there must exist $k \in \mathbb{N}$ such that for all $j > k$, there exists $i \leq k$ such that $x^i \leq x^j$. We will use this to show that algorithms for computing Gröbner bases and Markov bases terminate.

2.6 Linear programs

In this section, we define Linear Programs and discuss their properties. The properties we are most interested in are those of duality and boundedness (when the linear program has an optimal solution), which are strongly related. We will only give a brief discussion of linear programming; for a more complete description see for example [71]. Linear programming is a useful tool for integer programming and we will use it many times in this thesis.

Recall from the introduction that we define a linear program as the following:

$$LP := \min\{cx : x \in \mathcal{P}\}$$

where $\mathcal{P} \subseteq \mathbb{R}^n$ is a polyhedron, and $c \in \mathbb{R}^n$ is a cost vector. We often refer to the set of points in the polyhedron $\mathcal{P}$ as the **feasible** points of the linear program LP.

One of the most important concepts in linear programming is duality.

Definition 2.6.1. *The dual of $LP^\sigma_{A,c}(b) := \min\{cx : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x \geq b_\sigma\}$ is the dual linear program $DP^\sigma_{A,b}(c) := \max\{yb : yA = c, y_\sigma \geq \mathbf{0}\}$.*

Importantly, the dual of the dual of $LP^\sigma_{A,c}(b)$ is again $LP^\sigma_{A,c}(b)$. This can be seen by expressing the dual in primal form and performing the above transformation. Note that the polyhedron $\mathcal{P}^\sigma_A(b) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x \geq b_\sigma\}$ is the set

of feasible solutions of $LP^{\sigma}_{A,c}(b)$. For convenience, we define the dual polyhedron $\mathcal{D}^{\sigma}_{A}(c) := \{y \in \mathbb{R}^m : yA = c, y_\sigma \geq \mathbf{0}\}$, which is the set of feasible solutions of the dual. The following result is fundamental to linear programming and is called *weak duality*.

Lemma 2.6.2 (Weak Duality)**.** *We have $cx \geq yb$, for every $x \in \mathcal{P}^{\sigma}_{A}(b)$ and for every $y \in \mathcal{D}^{\sigma}_{A}(c)$.*

Weak duality says that if we have a feasible solution of the dual problem $y \in \mathcal{D}^{\sigma}_{A}(b)$, then $LP^{\sigma}_{A,c}(b) \geq yb$, so each feasible solution of the dual provides a lower bound on the value of $LP^{\sigma}_{A,c}(b) \geq yb$. Conversely, each feasible solution of the primal provides an upper bound on the value of $DP^{\sigma}_{A,b}(c)$. The implication of this is that if $\mathcal{P}^{\sigma}_{A}(b) \neq \emptyset$ and $\mathcal{D}^{\sigma}_{A}(c) \neq \emptyset$, then $LP^{\sigma}_{A,c}(b) \geq DP^{\sigma}_{A,b}(c)$. Furthermore in this situation where the primal problem is feasible and the dual problem is feasible, we have *strong duality*, which says that the optimal solutions of the primal and dual problems are equivalent.

Lemma 2.6.3 (Strong Duality)**.** *If $\mathcal{P}^{\sigma}_{A}(b) \neq \emptyset$ and $\mathcal{D}^{\sigma}_{A}(c) \neq \emptyset$, then $LP^{\sigma}_{A,c}(b) = DP^{\sigma}_{A,b}(c)$.*

Also, another implication of weak duality is that if $LP^{\sigma}_{A,c}(b)$ has an unbounded optimal value, then $\mathcal{D}^{\sigma}_{A}(c) = \emptyset$ since any feasible solution of $DP^{\sigma}_{A,b}(c)$ gives a lower bound on $LP^{\sigma}_{A,c}(b)$. Similarly, if $DP^{\sigma}_{A,b}(c)$ has an unbounded optimal value, then $\mathcal{P}^{\sigma}_{A}(b) = \emptyset$. For brevity, we will say that a feasible linear program that has an unbounded optimal value is **unbounded**, and **bounded** otherwise. The following corollary brings all the above results on duality together.

Corollary 2.6.4. *There are four possibilities for $LP^{\sigma}_{A,c}(b)$ and $DP^{\sigma}_{A,b}(c)$.*

(i). $LP^{\sigma}_{A,c}(b) = DP^{\sigma}_{A,b}(c)$.

(ii). $LP^{\sigma}_{A,c}(b) = -\infty$ *and* $\mathcal{D}^{\sigma}_{A}(c) = \emptyset$.

(iii). $DP^{\sigma}_{A,b}(c) = \infty$ *and* $\mathcal{P}^{\sigma}_{A}(b) = \emptyset$.

(iv). $\mathcal{P}^{\sigma}_{A}(b) = \emptyset$ *and* $\mathcal{D}^{\sigma}_{A}(c) = \emptyset$.

An interesting implication of Corollary 2.6.4 above is that if $\mathcal{D}^{\sigma}_{A}(c) = \emptyset$, then $LP^{\sigma}_{A,c}(b)$ is unbounded for all feasible b, and conversely, if $\mathcal{D}^{\sigma}_{A}(c) \neq \emptyset$, then $LP^{\sigma}_{A,c}(b)$ is bounded for all feasible $b \in \mathbb{R}^m$. Moreover, $\mathcal{P}^{\sigma}_{A}(\mathbf{0}) \neq \emptyset$ since $\mathbf{0} \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})$. Therefore, $LP^{\sigma}_{A,c}(b)$ is bounded for all feasible b if and only if $LP^{\sigma}_{A,c}(\mathbf{0})$ is bounded. In this thesis, we are often interested in the boundedness of $LP^{\sigma}_{A,c}(b)$ for all feasible $b \in \mathbb{R}^m$, so this result is particularly useful. The same is true for the dual program: $DP^{\sigma}_{A,b}(c)$ is bounded for all feasible c if and only if $DP^{\sigma}_{A,b}(\mathbf{0})$ is bounded.

Moreover, $\mathcal{P}^{\sigma}_{A}(\mathbf{0})$ is the recession cone of the polyhedron $\mathcal{P}^{\sigma}_{A}(b)$ for any $b \in \mathbb{R}^m$, and similarly, $\mathcal{D}^{\sigma}_{A}(\mathbf{0})$ is the recession cone of $\mathcal{D}^{\sigma}_{A}(c)$ for any $c \in \mathbb{R}^n$. Hence, the boundedness of a linear program is determined by its recession cone, so we may write the above results in a form that is independent of the representation of the linear program.

Corollary 2.6.5. *A feasible linear program* $LP := \min\{cx : x \in \mathcal{P}\}$ *is bounded if and only if* $\min\{cx : x \in \text{rec}(\mathcal{P})\}$ *is bounded.*

The linear program $LP^{\sigma}_{A,c}(\mathbf{0})$ is bounded if and only if there does not exist $x \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})$ such that $cx < 0$. Firstly, if no such x exists, then $LP^{\sigma}_{A,c}(\mathbf{0}) \geq 0$ and $LP^{\sigma}_{A,c}(\mathbf{0})$ is bounded by definition. Secondly, if there exists $x \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})$ where $cx < 0$, then we must have $LP^{\sigma}_{A,c}(\mathbf{0}) = -\infty$ because $\lambda x \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})$ for all $\lambda \in \mathbb{R}_+$ and $c\lambda x \to -\infty$ as $\lambda \to \infty$. In other words, $LP^{\sigma}_{A,c}(\mathbf{0})$ is bounded if and only if $LP^{\sigma}_{A,c}(\mathbf{0}) = 0$ because $LP^{\sigma}_{A,c}(\mathbf{0}) \geq 0$ implies that $LP^{\sigma}_{A,c}(\mathbf{0}) = 0$ since $\mathbf{0} \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})$. Additionally, note that, since $\mathcal{P}^{\sigma}_{A}(\mathbf{0})$ is a cone, there does not exist $x \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})$ where $cx < 0$ if and only if $c \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})^*$ (the dual cone of $\mathcal{P}^{\sigma}_{A}(\mathbf{0})$) from the definition of a dual cone. Thus, we also have that $LP^{\sigma}_{A,c}(\mathbf{0})$ is bounded if and only if $c \in \mathcal{P}^{\sigma}_{A}(\mathbf{0})^* = \{yA : y_\sigma \geq \mathbf{0}, y \in \mathbb{R}^m\}$, which is equivalent to the condition $\mathcal{D}^{\sigma}_{A}(c) \neq \emptyset$. This is exactly what Corollary 2.6.4(ii) says.

There is an analogous result for the dual program, which says that $DP^{\sigma}_{A,b}(\mathbf{0})$ is bounded if and only if there does not exist $y \in \mathcal{D}^{\sigma}_{A}(\mathbf{0})$ such that $yb > 0$ or equivalently $-b \in \mathcal{D}^{\sigma}_{A}(\mathbf{0})^* = \{y \in \mathbb{R}^n : A_{\bar{\sigma}}x = -y_{\bar{\sigma}}, A_\sigma x \geq -y_\sigma\}$, which is equivalent to the condition $\mathcal{P}^{\sigma}_{A}(b) \neq \emptyset$. This is exactly what Corollary 2.6.4(iii) says.

Lemma 2.6.6. *A feasible linear program* $LP := \min\{cx : x \in \mathcal{P}\}$ *is bounded if and only if there does not exist* $x \in \text{rec}(\mathcal{P})$ *such that* $cx < 0$ *or equivalently* $c \in \text{rec}(\mathcal{P})^*$.

2.7 Integer programs

In this section, we define Integer Programs and discuss their properties. We are most interested in the boundedness of integer programs. For a more detailed and thorough presentation of the theory and algorithms for integer programming, see for example the texts [71, 95].

Recall from the introduction that we define an integer program as the following:

$$IP := \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$$

where $\mathcal{P}$ is a rational polyhedron, and c is a linear function over $\mathbb{R}^n$. We call the set of integer points in a polyhedron, $\mathcal{F} = (\mathcal{P} \cap \mathbb{Z}^n)$, a **polyhedral fiber**, and we also refer to the polyhedral fiber $\mathcal{F} = (\mathcal{P} \cap \mathbb{Z}^n)$ as the **feasible** points of the integer program IP.

Fortunately, as the lemma below says, we can determine whether an integer program has a bounded optimal value from its linear programming relaxation. Note that, in the following lemma, we need to assume that $\mathcal{P}$ is a rational polyhedron.

Lemma 2.7.1. *A feasible integer program* $IP = \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$ *is bounded if and only if* $LP = \min\{cx : x \in \mathcal{P}\}$ *is bounded.*

Proof. If IP is unbounded, then LP must also be unbounded.

Assume that $LP = \min\{cx : x \in \mathcal{P}\}$ is unbounded. Recall from Lemma 2.6.6 that LP is unbounded if and only if there exists $x \in \text{rec}(\mathcal{P})$ where $cx < 0$. Since we

assume that all data is rational, $\mathcal{P}$ is a rational polyhedron, and we may assume that x is rational (i.e. $x \in \mathbb{Q}^n$). Now $\lambda x \in \text{rec}(\mathcal{P})$ for all $\lambda \in \mathbb{R}_+$, so $\lambda x \in \mathbb{Z}^n$ for some $\lambda \in \mathbb{R}_+$. Furthermore, if $\bar{x} \in \mathcal{P} \cap \mathbb{Z}^n$, then $(\bar{x} + \lambda x) \in \mathcal{P} \cap \mathbb{Z}^n$ for all $\lambda \in \mathbb{R}_+$ where $\lambda x \in \mathbb{Z}^n$. Therefore, IP is unbounded because $c(\bar{x} + \lambda x) \to -\infty$ if $\lambda \to \infty$. □

The following corollary was shown during the proof of Lemma 2.7.1 above.

Corollary 2.7.2. *A feasible integer program* $IP = \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$ *is bounded if and only if there does not exist* $x \in \text{rec}(\mathcal{P}) \cap \mathbb{Z}^n$ *such that* $cx < 0$.

Note that a linear program $LP = \min\{cx : x \in \mathcal{P}\}$ is bounded if and only if $\min\{cx : x \in \text{rec}(\mathcal{P})\}$ is bounded from Corollary 2.6.5 and $\min\{cx : x \in \text{rec}(\mathcal{P})\}$ is bounded if and only if the integer program $\min\{cx : x \in (\text{rec}(\mathcal{P}) \cap \mathbb{Z}^n)\}$ is bounded from Lemma 2.7.1 above. Thus, the corollary below follows.

Corollary 2.7.3. *A feasible integer program* $IP = \min\{cx : x \in (\mathcal{P} \cap \mathbb{Z}^n)\}$ *is bounded if and only if* $\min\{cx : x \in (\text{rec}(\mathcal{P}) \cap \mathbb{Z}^n)\}$ *is bounded.*

Often in this thesis, we are concerned with solving $IP^{\sigma}_{A,c}(b)$ for many different $b \in \mathbb{R}^m$. From 2.7.1 above, we have that each feasible $IP^{\sigma}_{A,c}(b)$ is unbounded if and only if $LP^{\sigma}_{A,c}(b)$ is unbounded. Moreover, from Lemma 2.6.6, $LP^{\sigma}_{A,c}(b)$ is unbounded if and only if $\min\{cx : x \in \mathcal{P}^{\sigma}_{A}(\mathbf{0}))\}$ is unbounded. Thus, by checking whether one linear program is unbounded, we can check whether every feasible $IP^{\sigma}_{A,c}(b)$ for $b \in \mathbb{R}^m$ is unbounded.

2.8 Linear space programs

Linear Subspace programs are essentially the same as linear programs; they are basically just a reformulation of a linear program. Although we will not use linear space programs much in this thesis, we find it useful to introduce them now as an introduction to lattice programs in the next section, which we will use quite frequently.

We define a linear space program as the following:

$$LP^{\sigma}_{\mathcal{S},c}(\nu) := \min\{cx : x - \nu \in \mathcal{S}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{R}^n\}$$

where $\sigma \subseteq \{1, ..., n\}$, $\bar{\sigma} = \{1, ..., n\} \setminus \sigma$, $\mathcal{S} \subseteq \mathbb{R}^n$ is a linear space, $c \in \mathbb{R}^n$ is some cost function, and $\nu \in \mathbb{R}^n$. We write the set of feasible solutions as

$$\mathcal{P}^{\sigma}_{\mathcal{S}}(\nu) := \{x \in \mathbb{R}^n : x - \nu \in \mathcal{S}, x_{\bar{\sigma}} \geq \mathbf{0}\},$$

so $LP^{\sigma}_{\mathcal{S},c}(\nu) := \min\{cx : x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)\}$. So, we have relaxed the non-negativity constraints on the x_σ variables. In the particular case where $\sigma = \emptyset$ and all variables are non-negative, we will usually omit σ from $LP^{\sigma}_{\mathcal{S},c}(\nu)$ and just write $LP_{\mathcal{S},c}(\nu)$ instead, and similarly, we write $\mathcal{P}_{\mathcal{S}}(\nu)$.

Remark 2.8.1. *Analogous to linear programming duality, linear space programs also have dual programs. The dual of* $\min\{cx : x - \nu \in \mathcal{S}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{R}^n\}$ *is the program* $\max\{y\nu : y_\sigma = -c_\sigma, y_{\bar{\sigma}} \geq -c_{\bar{\sigma}}, y \in \mathcal{S}^*\}$.

Any linear space program can be reformulated as a linear program. Consider the linear space $\mathcal{S} \in \mathbb{R}^n$. We know from section 2.1 that, for every linear space $\mathcal{S}$, we have $\mathcal{S} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}\}$ for some matrix $A \in \mathbb{R}^{m \times n}$. Then,

$$\begin{aligned} LP^{\sigma}_{\mathcal{S},c}(\nu) :=& \min\{cx : x - \nu \in \mathcal{S}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{R}^n\} \\ =& \min\{cx : A(x-\nu) = \mathbf{0}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{R}^n\} \\ =& \min\{cx : Ax = A\nu, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{R}^n\} \\ =& DP^{\bar{\sigma}}_{A^{\mathsf{T}},\text{-}c}(A\nu). \end{aligned}$$

The final formulation of the problem is in the form of the dual linear program $DP^{\bar{\sigma}}_{A^{\mathsf{T}},\text{-}c}(A\nu)$ where A^{T} is the transpose of the matrix A. Note that $\mathcal{P}^{\sigma}_{\mathcal{S}}(\nu) = \mathcal{D}^{\bar{\sigma}}_{A^{\mathsf{T}}}(A\nu)$, and so, $\mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)$ is a polyhedron. Thus, we may now apply Corollary 2.6.5 to obtain results about the boundedness of $LP^{\sigma}_{\mathcal{S},c}(\nu)$. First, observe that the recession cone of the polyhedron $\mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)$ is the cone $\mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0}) = \{x \in \mathcal{S} : x_{\bar{\sigma}} \geq \mathbf{0}\}$.

Lemma 2.8.2. *A feasible linear space program $LP^{\sigma}_{\mathcal{S},c}(\nu) := \min\{cx : x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)\}$ is bounded if and only if $LP^{\sigma}_{\mathcal{S},c}(\mathbf{0}) := \min\{cx : x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0})\}$ is bounded.*

Also, we may apply Lemma 2.6.6 to linear space programs to obtain the following useful lemma on the boundedness of $LP^{\sigma}_{\mathcal{S},c}(\nu)$. Note that the dual of the cone $\mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0})$ is the cone $\mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0})^* = \{y \in \mathbb{R}^n : y_{\sigma} = s_{\sigma}, y_{\bar{\sigma}} \geq s_{\bar{\sigma}}, s \in \mathcal{S}^*\}$.

Lemma 2.8.3. *A feasible linear space program $LP^{\sigma}_{\mathcal{S},c}(\nu) := \min\{cx : x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)\}$ is bounded if and only if there does not exist $x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0}) = \{x \in \mathcal{S} : x_{\bar{\sigma}} \geq \mathbf{0}\}$ such that $cx < 0$ or equivalently $c \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0})^* = \{y \in \mathbb{R}^n : y_{\sigma} = s_{\sigma}, y_{\bar{\sigma}} \geq s_{\bar{\sigma}}, s \in \mathcal{S}^*\}$.*

Importantly, if the linear space program $LP^{\sigma}_{\mathcal{S},c}(\nu)$ has an optimal solution, then it can always be reformulated as a linear space program where all the variables are non-negative. This is achieved by projecting the linear space program onto the $x_{\bar{\sigma}}$ variables as shown below. We may assume that $c_{\sigma} = \mathbf{0}$ without loss of generality, which follows from Lemma 2.8.4 below. Then,

$$\begin{aligned} LP^{\sigma}_{\mathcal{S},c}(\nu) :=& \min\{cx : x - \nu \in \mathcal{S}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{R}^n\} \\ =& \min\{c_{\bar{\sigma}} x_{\bar{\sigma}} : x_{\bar{\sigma}} - \nu_{\bar{\sigma}} \in \mathcal{S}^{\sigma}, x_{\sigma} - \nu_{\sigma} \in \mathcal{S}^{\bar{\sigma}}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{R}^n\} \\ =& \min\{c_{\bar{\sigma}} x_{\bar{\sigma}} : x_{\bar{\sigma}} - \nu_{\bar{\sigma}} \in \mathcal{S}^{\sigma}, x_{\bar{\sigma}} \geq \mathbf{0}, x_{\bar{\sigma}} \in \mathbb{R}^{|\bar{\sigma}|}\} \\ =& LP_{\mathcal{S}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}}). \end{aligned}$$

Note that $\mathcal{S}^{\sigma}$ denotes the projection of $\mathcal{S}$ onto the $\bar{\sigma}$ components and $\mathcal{S}^{\bar{\sigma}}$ denotes the projection of $\mathcal{S}$ onto the σ components. Thus, we can solve $LP^{\sigma}_{\mathcal{S},c}(\nu)$ by solving $LP_{\mathcal{S}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}})$, and all the variables are non-negative. Furthermore, if we know the optimal solution of $LP_{\mathcal{S}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}})$, then we can easily reconstruct the optimal solution of $LP^{\sigma}_{\mathcal{S},c}(\nu)$ as follows. Let $x^*_{\bar{\sigma}}$ be the optimal solution of $LP_{\mathcal{S}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}})$. Assuming that $\mathcal{S} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}\}$, we then compute x^*_{σ} by solving the following equation: $A_{*\sigma}x^*_{\sigma} + A_{*\bar{\sigma}}x^*_{\bar{\sigma}} = A\nu$. Note that $A_{*\sigma}$ is a submatrix of A consisting of the columns of A indexed by σ, and $A_{*\bar{\sigma}}$ is the columns of A indexed by $\bar{\sigma}$. This equation must have a solution since $x^*_{\bar{\sigma}} - \nu_{\bar{\sigma}} \in \mathcal{S}^{\sigma}$. Then, $x^* \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)$ and x^* is the optimal solution of $LP^{\sigma}_{\mathcal{S},c}(\nu)$ since $cx^* = c_{\bar{\sigma}}x^*_{\bar{\sigma}}$.

We now show that we can assume that $c_{\sigma} = \mathbf{0}$.

Lemma 2.8.4. *For every feasible bounded linear space program $LP^{\sigma}_{\mathcal{S},c}(\nu)$, there exists $\tilde{c} \in \mathbb{R}^n$ and $k \in \mathbb{R}$ such that $\tilde{c}_\sigma = \mathbf{0}$, $\tilde{c}_{\bar{\sigma}} \geq \mathbf{0}$, $\tilde{c}x = cx + k$ for all $x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)$, and $LP^{\sigma}_{\mathcal{S},c}(\nu) = LP^{\sigma}_{\mathcal{S},\tilde{c}}(\nu) + k$.*

Proof. From Lemma 2.8.3, the linear space program $LP^{\sigma}_{\mathcal{S},c}(\nu)$ is bounded if and only if $c \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0})^* = \{y \in \mathbb{R}^n : y_\sigma = s_\sigma, y_{\bar{\sigma}} \geq s_{\bar{\sigma}}, s \in \mathcal{S}^*\}$. So, assuming $LP^{\sigma}_{\mathcal{S},c}(\nu)$ is bounded, there exists $s \in \mathcal{S}^*$ such that $c_\sigma = s_\sigma$ and $c_{\bar{\sigma}} \geq s_{\bar{\sigma}}$. Let $\tilde{c} = c - s$; then, $\tilde{c}_\sigma = \mathbf{0}$, and $\tilde{c}_{\bar{\sigma}} \geq \mathbf{0}$ by construction. Now, for every $x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\nu)$, we have $\tilde{c}x = cx - sx = cx - s\nu$ since $x - \nu \in \mathcal{S}$ and $s \in \mathcal{S}^*$ implies that $s(x - \nu) = 0$ and $sx = s\nu$. Therefore, $LP^{\sigma}_{\mathcal{S},\tilde{c}}(\nu) = LP^{\sigma}_{\mathcal{S},c}(\nu) - s\nu$. □

Note that the proof of the Lemma 2.8.4 is constructive: if we are given a linear space program $LP^{\sigma}_{\mathcal{S},c}(\nu)$, then we can find a cost vector $\tilde{c} \in \mathbb{R}^n$ where $\tilde{c}_\sigma = \mathbf{0}$ or $\tilde{c}_{\bar{\sigma}} \geq \mathbf{0}$ by finding $s \in \mathcal{S}^*$ such that $c_\sigma = s_\sigma$ and $c_{\bar{\sigma}} \geq s_{\bar{\sigma}}$ and setting $\tilde{c} = c - s\nu$.

Crucially, any linear program can be formulated as a linear space program. First consider the dual linear program, $DP^{\sigma}_{A,b}(c) := \max\{yb : yA = c, y_\sigma \geq \mathbf{0}, y \in \mathbb{R}^m\}$. Let $\mathcal{S} := \{y \in \mathbb{R}^m : yA = \mathbf{0}\}$, which is a linear space. Also, let $\nu \in \mathbb{R}^m$ be such that $\nu A = c$. We can always find such a ν using linear algebra or if no such ν exists, then LP is infeasible and there is nothing more to do. We saw previously that the linear space program $LP^{\bar{\sigma}}_{\mathcal{S},-b}(\nu)$ may be reformulated as $DP^{\sigma}_{A,b}(\nu A)$, and now, we proceed in the other direction in the same manner.

$$\begin{aligned}
DP^{\sigma}_{A,b}(c) :=& \max\{yb : yA = c, y_\sigma \geq \mathbf{0}, y \in \mathbb{R}^m\} \\
=& \max\{yb : yA = \nu A, y_\sigma \geq \mathbf{0}, y \in \mathbb{R}^m\} \\
=& \max\{yb : (x - \nu)A = \mathbf{0}, y_\sigma \geq \mathbf{0}, y \in \mathbb{R}^m\} \\
=& \max\{yb : x - \nu \in \mathcal{S}, y_\sigma \geq \mathbf{0}, y \in \mathbb{R}^m\} \\
=& LP^{\bar{\sigma}}_{\mathcal{S},-b}(\nu).
\end{aligned}$$

Note that $\mathcal{D}^{\sigma}_{A}(\nu) = \mathcal{F}^{\bar{\sigma}}_{\mathcal{S}}(\nu)$ in this case.

To reformulate a linear program in primal form $LP^{\sigma}_{A,c}(b) := \min\{cx : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x \geq b_\sigma\}$ as a linear space program, we will need to introduce some additional *slack* variables to change the inequality constraints into equality constraints. Then,

$$\begin{aligned}
LP^{\sigma}_{A,b}(c) :=& \min\{cx : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x \geq b_\sigma, x \in \mathbb{R}^n\} \\
=& \min\{cx : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x - Is = b_\sigma, x \in \mathbb{R}^n, s \in \mathbb{R}^{|\sigma|}_+\}.
\end{aligned}$$

Now let $\mathcal{S} \subseteq \mathbb{R}^{n+m}$ where $\mathcal{S} := \{(x, s) \in \mathbb{R}^{n+m} : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x - Is = \mathbf{0}\}$. Also, let $\nu \in \mathbb{R}^{n+m}$ where $\nu = (\nu_x, \nu_s)$ such that $A_{\bar{\sigma}}\nu_x = b_{\bar{\sigma}}$ and $A_\sigma \nu_x - I\nu_s = b_\sigma$. We can find ν using linear algebra or if no such ν exists, then $LP^{\sigma}_{A,b}(c)$ is infeasible. Then,

$$\begin{aligned}
LP^{\sigma}_{A,b}(c) =& \min\{cx : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_\sigma x - Is = b_\sigma, x \in \mathbb{R}^n, s \in \mathbb{R}^{|\sigma|}_+\} \\
=& \min\{cx : A_{\bar{\sigma}}x = A_{\bar{\sigma}}\nu_x, A_\sigma x - Is = A_\sigma \nu_x - I\nu_s, x \in \mathbb{R}^n, s \in \mathbb{R}^{|\sigma|}_+\} \\
=& \min\{cx : A_{\bar{\sigma}}(x - \nu_x) = \mathbf{0}, A_\sigma(x - \nu_x) - I(s - \nu_s) = \mathbf{0}, x \in \mathbb{R}^n, s \in \mathbb{R}^{|\sigma|}_+\} \\
=& \min\{cx : (x, s) - (\nu_x, \nu_s) \in \mathcal{S}, x \in \mathbb{R}^n, s \in \mathbb{R}^{|\sigma|}_+\}.
\end{aligned}$$

The final formulation is a linear space program. Recall that we can reformulate a linear space program into an equivalent form with only non-negative variables by just projecting onto the non-negative variables assuming that the linear space program has an optimal solution. In this case, this means projecting onto the slack variables s. In order to perform this reformulation, we must first find a cost function $\tilde{c} = (\tilde{c}_x, \tilde{c}_y) \in \mathbb{R}^{m+n}$ that is equivalent to c such that $\tilde{c}_x = \mathbf{0}$ (and $\tilde{c}_s \geq \mathbf{0}$), which is always possible from Lemma 2.8.4. So,

$$\begin{aligned} LP^{\sigma}_{A,b}(c) :=& \min\{cx : A_{\bar{\sigma}}x = b_{\bar{\sigma}}, A_{\sigma}x \geq b_{\sigma}, x \in \mathbb{R}^n\} \\ =& \min\{cx : (x,s) - (\nu_x, \nu_s) \in \mathcal{S}, x \in \mathbb{R}^n, s \in \mathbb{R}^{|\sigma|}_+\} \\ \equiv& \min\{\tilde{c}_s s : s - \nu_s \in \mathcal{S}^x, s \in \mathbb{R}^{|\sigma|}_+\} \\ =& LP_{\mathcal{S}^x, \tilde{c}_s}(\nu_s) \end{aligned}$$

where $\mathcal{S}^x \subseteq \mathbb{R}^{|\sigma|}$ denotes the projection of $\mathcal{S}$ onto the slack variables s. In other words, we can formulate any integer program as a linear space program involving only the slack variables of the inequality constraints of the integer program. Recall that $\mathcal{F}^{\sigma}_A(b)$ is the set of feasible solutions of $LP^{\sigma}_{A,b}(c)$ and $\mathcal{F}_{\mathcal{S}^x}(\nu_s)$ is the set of feasible solutions of $LP_{\mathcal{S}^x,\tilde{c}_s}(\nu_s)$. Then, for every $x \in \mathcal{F}^{\sigma}_A(b)$, we have $(A_{\sigma}x - b_{\sigma}) \in \mathcal{F}_{\mathcal{S}^x}(\nu_s)$, and also, for every $s \in \mathcal{F}_{\mathcal{S}^x}(\nu_s)$, there exists an $x \in \mathcal{F}^{\sigma}_A(b)$ such that $(A_{\sigma}x - b_{\sigma}) = s$.

2.9 Lattice programs

Lattice programs are essentially the same as integer programs; they are basically just a reformulation of an integer program. In this thesis, we will use lattice programs as the preferred formulation of an integer program.

We define a lattice program as the following:

$$IP^{\sigma}_{\mathcal{L},c}(\nu) := \min\{cx : x - \nu \in \mathcal{L}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{Z}^n\}$$

where $\sigma \subseteq \{1, ..., n\}$, $\bar{\sigma} = \{1, ..., n\} \setminus \sigma$, $\mathcal{L} \subseteq \mathbb{Z}^n$ is an integer lattice, $c \in \mathbb{R}^n$ is some cost function, and $\nu \in \mathbb{Z}^n$. We call the set of feasible points of a lattice program a **lattice fiber**, and we write it as $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu) := \{x \in \mathbb{Z}^n : x - \nu \in \mathcal{L}, x_{\bar{\sigma}} \geq \mathbf{0}\}$. In the particular case where $\sigma = \emptyset$ and all variables are non-negative, we will usually omit σ from $IP^{\sigma}_{\mathcal{L},c}(\nu)$ and just write $IP_{\mathcal{L},c}(\nu)$ instead, and similarly, we write $\mathcal{F}_{\mathcal{L}}(\nu)$.

Any lattice program can be formulated as an integer program. Consider the lattice $\mathcal{L} \in \mathbb{Z}^n$. First, we consider the special case where $\mathcal{L} = \{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$ for some matrix $A \in \mathbb{R}^{m \times n}$. Then,

$$\begin{aligned} IP^{\sigma}_{\mathcal{L},c}(\nu) :=& \min\{cx : x - \nu \in \mathcal{L}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{cx : A(x - \nu) = \mathbf{0}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{cx : Ax = A\nu, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{Z}^n\} \end{aligned}$$

The final formulation of the problem is an integer program. Incidentally, we have also shown here how to reformulate the lattice fiber $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ as a polyhedral fiber.

Unfortunately, in general, not all lattices are saturated: we cannot represent all lattices in the form $\mathcal{L} = \{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$ for some matrix $A \in \mathbb{R}^{m\times n}$. In this case, we can still formulate the lattice program as an integer program, but we need some additional variables. Let $B \in \mathbb{Z}^{k\times n}$ be a basis for the lattice $\mathcal{L}$ where the rows of the matrix B span $\mathcal{L}$, so $\mathcal{L} = \{yB : y \in \mathbb{Z}^k\}$. Then,

$$\begin{aligned} IP^{\sigma}_{\mathcal{L},c}(\nu) :=& \min\{cx : x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)\} \\ =& \min\{cx : x - \nu \in \mathcal{L}, x_{\bar\sigma} \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{cx : x - \nu = yB, x_{\bar\sigma} \geq \mathbf{0}, x \in \mathbb{Z}^n, y \in \mathbb{Z}^k\}. \end{aligned}$$

The final formulation is an integer program. We could use the equation $x = yB + \nu$ to eliminate the x variables from the formulation if we wanted a more compact formulation. Note that we have also shown here how to reformulate the lattice fiber $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ as the projection of a polyhedral fiber.

The linear relaxation of $IP^{\sigma}_{\mathcal{L},c}(\nu)$ is the linear space program $LP^{\sigma}_{\mathcal{S},c}(\nu)$ where $\mathcal{S}$ is the inclusion-minimal linear space containing the lattice $\mathcal{L}$; i.e., for the lattice $\mathcal{L} = \{yB : y \in \mathbb{Z}^k\}$ where $B \in \mathbb{R}^{k\times n}$, the linear space $\mathcal{S} = \{yB : y \in \mathbb{R}^k\}$ is the inclusion-minimal linear space containing $\mathcal{L}$. This follows since we can formulate $LP^{\sigma}_{\mathcal{S},c}(\nu)$ as a linear program in the same form as the integer programming formulation of $IP^{\sigma}_{\mathcal{L},c}(\nu)$ above but without the integrality constraints:

$$\begin{aligned} LP^{\sigma}_{\mathcal{S},c}(\nu) :=& \min\{cx : x - \nu \in \mathcal{S}, x_{\bar\sigma} \geq \mathbf{0}, x \in \mathbb{R}^n\} \\ =& \min\{cx : x - \nu = yB, x_{\bar\sigma} \geq \mathbf{0}, x \in \mathbb{R}^n, y \in \mathbb{R}^k\}. \end{aligned}$$

As with integer programs, we can determine if a lattice program has a bounded optimal value from its linear relaxation. The proof follows closely the proof of 2.7.1.

Lemma 2.9.1. *A feasible lattice program $IP^{\sigma}_{\mathcal{L},c}(\nu)$ is bounded if and only if $LP^{\sigma}_{\mathcal{S},c}(\nu)$ is bounded where $\mathcal{S}$ is the inclusion-minimal linear space containing $\mathcal{L}$.*

Proof. If $LP^{\sigma}_{\mathcal{L},c}(\nu)$ is unbounded, then $LP^{\sigma}_{\mathcal{S},c}(\nu)$ must also be unbounded since it is the linear relaxation of $LP^{\sigma}_{\mathcal{L},c}(\nu)$.

Assume that $LP^{\sigma}_{\mathcal{S},c}(\nu)$ is unbounded. Recall from Lemma 2.8.3 that $LP^{\sigma}_{\mathcal{S},c}(\nu)$ is bounded if and only if there does exists $x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0}) = \{x \in \mathcal{S} : x_{\bar\sigma} \geq \mathbf{0}\}$ such that $cx < 0$. Since we assume that all data is rational, $\mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0})$ is a rational polyhedron, and we may assume that x is rational. Now $\lambda x \in \mathcal{P}^{\sigma}_{\mathcal{S}}(\mathbf{0})$ for all $\lambda \in \mathbb{R}_+$, so $\lambda x \in \mathcal{L}$ for some $\lambda \in \mathbb{R}_+$, in which case, $\lambda x \in \{x \in \mathcal{L} : x_{\bar\sigma} \geq \mathbf{0}\} = \mathcal{F}^{\sigma}_{\mathcal{S}}(\mathbf{0})$. Furthermore, if $\bar x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, then $(\bar x + \lambda x) \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for all $\lambda \in \mathbb{R}_+$ where $\lambda x \in \mathcal{L}$. Therefore, $IP^{\sigma}_{\mathcal{L},c}(\nu)$ is unbounded because $c(\bar x + \lambda x) \to -\infty$ if $\lambda \to \infty$. □

The following corollary was shown during the proof of Lemma 2.9.1 above.

Corollary 2.9.2. *A feasible lattice program $IP^{\sigma}_{\mathcal{L},c}(\nu)$ is bounded if and only if there does not exist $x \in \mathcal{F}^{\sigma}_{\mathcal{S}}(\mathbf{0}) = \{x \in \mathcal{L} : x_{\bar\sigma} \geq \mathbf{0}\}$ such that $cx < 0$.*

Note that in Corollary 2.9.2 above, whether a lattice program $IP^{\sigma}_{\mathcal{L},c}(\nu)$ is bounded or not is independent of ν except that $IP^{\sigma}_{\mathcal{L},c}(\nu)$ must be feasible. Also, the lattice program $IP^{\sigma}_{\mathcal{L},c}(\mathbf{0})$ is always feasible since $\mathbf{0} \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\mathbf{0})$. Thus, we arrive at the following useful corollary.

Corollary 2.9.3. *A feasible lattice program $IP^{\sigma}_{\mathcal{L},c}(\nu)$ is bounded if and only if $IP^{\sigma}_{\mathcal{L},c}(\mathbf{0})$ is bounded.*

Importantly, if the relaxation $IP^{\sigma}_{\mathcal{L},c}(\nu)$ has an optimal solution, then it may be reformulated as a lattice program where all the variables are non-negative. This is one of the main reasons why lattice programs are a convenient formulation. This is achieved by projecting the integer program onto the $x_{\bar{\sigma}}$ variables as shown below. Firstly, we can assume that $c_{\sigma} = \mathbf{0}$, which we showed was true for the linear relaxation $IP^{\sigma}_{\mathcal{S},c}(\nu)$ (see Lemma 2.8.4), so it is also true for $IP^{\sigma}_{\mathcal{L},c}(\nu)$. Then,

$$\begin{aligned} IP^{\sigma}_{\mathcal{L},c}(\nu) :=& \min\{cx : x - \nu \in \mathcal{L}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{c_{\bar{\sigma}} x_{\bar{\sigma}} : x_{\bar{\sigma}} - \nu_{\bar{\sigma}} \in \mathcal{L}^{\sigma}, x_{\sigma} - \nu_{\sigma} \in \mathcal{L}^{\bar{\sigma}}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{c_{\bar{\sigma}} x_{\bar{\sigma}} : x_{\bar{\sigma}} - \nu_{\bar{\sigma}} \in \mathcal{L}^{\sigma}, x_{\bar{\sigma}} \geq \mathbf{0}, x_{\bar{\sigma}} \in \mathbb{Z}^{|\bar{\sigma}|}\} \\ =& IP_{\mathcal{L}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}}). \end{aligned}$$

Thus, we can solve $IP^{\sigma}_{\mathcal{L},c}(\nu)$ by solving $IP_{\mathcal{L}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}})$, and all the variables are non-negative. Conceptually, this means that it is the non-negative constraints that make the lattice program difficult to solve, and we can effectively ignore the other variables. Indeed, if there are no non-negativity constraints on the variables, then the lattice program is easy to solve. Furthermore, if we know the optimal solution of $IP_{\mathcal{L}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}})$, then we can easily reconstruct the optimal solution of $IP^{\sigma}_{\mathcal{L},c}(\nu)$ as follows. Let $x^*_{\bar{\sigma}}$ be the optimal solution of $IP_{\mathcal{L}^{\sigma},c_{\bar{\sigma}}}(\nu_{\bar{\sigma}})$. Let $B \in \mathbb{Z}^{k \times n}$ be a basis of $\mathcal{L}$, so we have $\mathcal{L} = \{\lambda B : \lambda \in \mathbb{Z}^k\}$. Then, $B_{*\bar{\sigma}}$ is a basis of $\mathcal{L}^{\sigma}$. Thus, $x^*_{\bar{\sigma}} = \lambda B_{*\bar{\sigma}} + \nu_{\bar{\sigma}}$ for some $\lambda \in \mathbb{Z}^k$. Let $x^*_{\sigma} = \lambda B_{*\sigma} + \nu_{\sigma}$. Then, $x^* \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, and x^* is the optimal solution of $IP^{\sigma}_{\mathcal{S},c}(\nu)$ since $cx^* = c_{\bar{\sigma}} x^*_{\bar{\sigma}}$.

In this thesis, we often consider projecting the set of points in $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ onto the set $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$. We are often interested in doing this under the condition that there is a one-to-one correspondence between points in $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ and points in $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$ for every $\nu \in \mathbb{Z}^n$: for every $\nu \in \mathbb{Z}^n$ and every $x, y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, if $x_{\bar{\sigma}} = y_{\bar{\sigma}}$, then $x = y$, or in other words, the projective map $\pi_{\sigma} : \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu) \to \mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$ is a bijection for every $\nu \in \mathbb{Z}^n$. There is such a one-to-one correspondence if and only if there is a one-to-one correspondence between vectors in $\mathcal{L}$ and vectors in $\mathcal{L}^{\sigma}$: for $u, v \in \mathcal{L}$, if $u_{\bar{\sigma}} = v_{\bar{\sigma}}$, then $u = v$, or in other words, the projective map $\pi_{\sigma} : \mathcal{L} \to \mathcal{L}^{\sigma}$ is a bijection, and thus, the inverse map π^{-1}_{σ} is well-defined. First, note that there exists $u, v \in \mathcal{L}$ such that $u_{\bar{\sigma}} = v_{\bar{\sigma}}$ and $u \neq v$ if and only if $\{u \in \mathcal{L} : u_{\bar{\sigma}} = \mathbf{0}\} \neq \{\mathbf{0}\}$. If there exists $x, y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$ such that $x_{\bar{\sigma}} = y_{\bar{\sigma}}$ and $x \neq y$, then $(x - y) \neq \mathbf{0}$ and $(x - y) \in \{u \in \mathcal{L} : u_{\bar{\sigma}} = \mathbf{0}\}$. Conversely, let $u \in \{u \in \mathcal{L} : u_{\bar{\sigma}} = \mathbf{0}\}$. Let $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ and let $y = x + u$. Note that $y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$. Then, $x_{\bar{\sigma}} = y_{\bar{\sigma}}$ but $x \neq y$. Hence, there is a one-to-one correspondence between points in $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ and points in $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$ if and only if $\ker(\pi_{\sigma}) \cap \mathcal{L} = \{u \in \mathcal{L} : u_{\bar{\sigma}} = \mathbf{0}\} = \{\mathbf{0}\}$.[4]

Crucially, any integer program can be formulated as a lattice program. Consider the special case, $IP := \min\{cx : Ax = b, x_{\sigma} \geq \mathbf{0}, x \in \mathbb{Z}^n\}$. Let $\mathcal{L} := \{x \in \mathbb{Z}^n : Ax = \mathbf{0}\}$, which is a lattice. Also, let $\nu \in \mathbb{Z}^n$ such that $A\nu = b$. We can always find such an

[4]The notation $\ker(\pi_{\sigma})$ means the kernel of the map π_{σ}; that is, the points that are mapped to zero.

ν using the HNF algorithm. If no such ν exists, then IP is infeasible and there is nothing more to do. Then,

$$\begin{aligned} IP :=& \min\{cx : Ax = b, x_\sigma \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{cx : Ax = A\nu, x_\sigma \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{cx : A(x-\nu) = \mathbf{0}, x_\sigma \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& \min\{cx : x - \nu \in \mathcal{L}, x_\sigma \geq \mathbf{0}, x \in \mathbb{Z}^n\} \\ =& IP^\sigma_{\mathcal{L},c}(\nu). \end{aligned}$$

Note that $\mathcal{F}^\sigma_A(b) = \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ in this case, so we have reformulated a polyhedral fiber as a lattice fiber..

To reformulate integer programs in general form as a lattice program, we will need to introduce some additional *slack* variables to change the inequality constraints into equality constraints. Consider the integer program $IP^\sigma_{A,b}(c)$. We always assume that all data is rational, and for this situation, we also will assume that all data has been scaled so that it is integral (i.e. $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$, and $c \in \mathbb{Z}^n$). Then,

$$\begin{aligned} IP^\sigma_{A,b}(c) :=& \min\{cx : A_{\bar\sigma}x = b_{\bar\sigma}, A_\sigma x \geq b_\sigma, x \in \mathbb{Z}^n\} \\ =& \min\{cx : A_{\bar\sigma}x = b_{\bar\sigma}, A_\sigma x - Is = b_\sigma, x \in \mathbb{Z}^n, s \in \mathbb{N}^{|\sigma|}\}. \end{aligned}$$

Now let $\mathcal{L} \subseteq \mathbb{Z}^{n+m}$ where $\mathcal{L} := \{(x,s) \in \mathbb{Z}^{n+m} : A_{\bar\sigma}x = \mathbf{0}, A_\sigma x - Is = \mathbf{0}\}$. Also, let $\nu \in \mathbb{Z}^{n+m}$ where $\nu = (\nu_x, \nu_s)$ such that $A_{\bar\sigma}\nu_x = b_{\bar\sigma}$ and $A_\sigma\nu_x - I\nu_s = b_\sigma$. We can find ν using the HNF algorithm. If no such ν exists, then $IP^\sigma_{A,b}(c)$ is infeasible. Then,

$$\begin{aligned} IP^\sigma_{A,b}(c) =& \min\{cx : A_{\bar\sigma}x = b_{\bar\sigma}, A_\sigma x - Is = b_\sigma, x \in \mathbb{Z}^n, s \in \mathbb{N}^{|\sigma|}\} \\ =& \min\{cx : A_{\bar\sigma}x = A_{\bar\sigma}\nu_x, A_\sigma x - Is = A_\sigma\nu_x - I\nu_s, x \in \mathbb{Z}^n, s \in \mathbb{N}^{|\sigma|}\} \\ =& \min\{cx : A_{\bar\sigma}(x-\nu_x) = \mathbf{0}, A_\sigma(x-\nu_x) - I(s-\nu_s) = \mathbf{0}, x \in \mathbb{Z}^n, s \in \mathbb{N}^{|\sigma|}\} \\ =& \min\{cx : (x,s) - (\nu_x,\nu_s) \in \mathcal{L}, x \in \mathbb{Z}^n, s \in \mathbb{N}^{|\sigma|}\}. \end{aligned}$$

The final formulation is a lattice program. Note that we have also shown that a polyhedral fiber can be reformulated as the projection of a lattice fiber.

Now, recall that we can reformulate a lattice program into an equivalent form with only non-negative variables by just projecting onto the non-negative variables assuming that the lattice program has an optimal solution. In this case, this means projecting onto the slack variables s. In order to perform this reformulation, we must first find a cost function $\tilde c = (\tilde c_x, \tilde c_s) \in \mathbb{Z}^{n+m}$ that is equivalent to c such that $\tilde c_x = \mathbf{0}$, which is always possible. So,

$$\begin{aligned} IP^\sigma_{A,b}(c) :=& \min\{cx : A_{\bar\sigma}x = b_{\bar\sigma}, A_\sigma x \geq b_\sigma, x \in \mathbb{Z}^n\} \\ =& \min\{cx : (x,s) - (\nu_x,\nu_s) \in \mathcal{L}, x \in \mathbb{Z}^n, s \in \mathbb{N}^{|\sigma|}\} \\ =& \min\{\tilde c_s s : s - \nu_s \in \mathcal{L}^x, s \in \mathbb{N}^{|\sigma|}\} \\ =& IP_{\mathcal{L}^x,\tilde c_s}(\nu_s) \end{aligned}$$

where $\mathcal{L}^x$ denotes the projection of $\mathcal{L}$ onto the slack variables s. In other words, we can formulate any integer program as a lattice program involving only the slack

variables of the inequality constraints of the integer program. Recall that $\mathcal{F}_A^\sigma(b)$ is the set of feasible solutions of $IP_{A,b}^\sigma(c)$ and $\mathcal{F}_{\mathcal{L}^x}(\nu_s)$ is the set of feasible solutions of $IP_{\mathcal{L}^x,\tilde{c}_s}(\nu_s)$. Then, for every $x \in \mathcal{F}_A^\sigma(b)$, we have $(A_\sigma x - b_\sigma) \in \mathcal{F}_{\mathcal{L}^x}(\nu_s)$, and also, for every $s \in \mathcal{F}_{\mathcal{L}^x}(\nu_s)$, there exists an $x \in \mathcal{F}_A^\sigma(b)$ such that $(A_\sigma x - b_\sigma) = s$.

Example 2.9.4. *Consider* $IP_{A,c}(b) := \min\{cx : x \in (\mathcal{P}_A(b) \cap \mathbb{Z}^n)\}$ *where*

$$A = \begin{pmatrix} 2 & 3 \\ -2 & +1 \\ -2 & -1 \\ 1 & -1 \end{pmatrix}, b = \begin{pmatrix} 6 \\ -4 \\ -10 \\ -1 \end{pmatrix}, \text{ and } c = (3,4),$$

so $IP_{A,c}(b) = \min\{3x_1+4x_2 : 2x_1+3x_2 \geq 6, 2x_1-x_2 \leq 4, 2x_1+x_2 \leq 10, -x_1+x_2 \leq 1, x \in \mathbb{Z}^2\}$. *This is the integer program from Section 1.5 in the introduction. We will reformulate this as a lattice program. Let* $\mathcal{L} = \{(x,s) \in \mathbb{Z}^6 : Ax - Is = \mathbf{0}\}$, *and let* $\sigma = \{1,2\}$, *so* σ *indexes the* x *variables and* $\bar{\sigma}$ *indexes the* s *variables. A basis for* $\mathcal{L}$ *is*

$$B = (I, A^\intercal) = \begin{pmatrix} 1 & 0 & 2 & -2 & -2 & 1 \\ 0 & 1 & 3 & 1 & -1 & -1 \end{pmatrix}.$$

Let $\nu = (\nu_\sigma, \nu_{\bar{\sigma}}) \in \mathbb{Z}^6$ *where* $\nu_\sigma = (0,0)$ *and* $\nu_{\bar{\sigma}} = -b = (-6,4,10,1)$, *so* $A\nu_\sigma - I\nu_{\bar{\sigma}} = b$. *Finally, let* $\bar{c} = (\bar{c}_\sigma, \bar{c}_{\bar{\sigma}}) \in \mathbb{Z}^6$ *where* $\bar{c}_\sigma = c$ *and* $\bar{c}_{\bar{\sigma}} = \mathbf{0}$. *We have now reformulated the integer program* $IP_{A,c}(b)$ *as the lattice program* $IP_{\mathcal{L},\bar{c}}^\sigma(\nu)$. *The linear relaxation of* $IP_{\mathcal{L},\bar{c}}^\sigma(\nu)$ *is the linear subspace program* $LP_{\mathcal{S},\bar{c}}^\sigma(\nu)$ *where* $\mathcal{S} = \{yB : y \in \mathbb{R}^2\}$.

We can map any feasible solution of the integer program $IP_{A,c}(b)$ *to a feasible solution of the lattice program* $IP_{\mathcal{L},\bar{c}}^\sigma(\nu)$ *as follows: if* $x \in \mathcal{F}_A(b)$, *then* $(x,s) \in \mathcal{F}_{\mathcal{L}}^x(\nu)$ *where* $s = Ax - b$. *Recall from Example 1.5.1 that the optimal solution of* $IP_{A,c}(b)$ *is* $x^* = (2,1)$, *which maps to* $(x^*, s^*) = (2,1,1,1,5,2) \in \mathcal{F}_{\mathcal{L}}^\sigma(\nu)$, *which is the optimal solution of* $LP_{\mathcal{S},\bar{c}}^\sigma(\nu)$.

We can project this lattice program onto its non-negative variables (the s *variables). To do so, we must find an equivalent cost vector* $\tilde{c} = (\tilde{c}_x, \tilde{c}_s)$ *such that* $\tilde{c}_x = \mathbf{0}$ *and* $\tilde{c}_s \geq \mathbf{0}$. *From Lemma 2.8.4, to find* $\tilde{c}$, *we must find a vector* y *in the set* $\mathcal{S}^* = \{y \in \mathbb{R}^6 : By = \mathbf{0}\}$ *such that* $\bar{c}_x = c = y_x$ *and* $\bar{c}_s = \mathbf{0} \geq y_s$. *The vector* $y = (3,4,\text{-}2,0,\text{-}1,\text{-}1)$ *satisfies the conditions. Then, let* $\tilde{c} = \bar{c} - y = (0,0,2,0,1,1)$. *So,* $IP_{\mathcal{L},\bar{c}}^x(\nu) = IP_{\mathcal{L},\tilde{c}}^x(\nu) - y\nu$ *and* $y\nu = -1$. *We can now project* $IP_{\mathcal{L},\tilde{c}}^x(\nu)$ *onto the* s *variables giving the lattice program* $IP_{\mathcal{L}^x,\tilde{c}_s}(\nu_s)$. *Explicitly,* $\tilde{c}_s = (2,0,1,1)$, $\nu_s = (\text{-}6,4,10,1)$, *and a basis of* $\mathcal{L}^x$ *is*

$$B_{*s} = A^\intercal = \begin{pmatrix} 2 & \text{-}2 & \text{-}2 & 1 \\ 3 & 1 & \text{-}1 & \text{-}1 \end{pmatrix}.$$

Note that the lattice program $IP_{\mathcal{L}^x,\tilde{c}_s}(\nu_s)$ *is in the space of the slack variables of the integer program* $IP_{A,c}(b)$.

From above, the optimal solution of $IP_{\mathcal{L},\tilde{c}}^x(\nu)$ *is* $(2,1,1,1,5,2)$, *so the optimal solution of* $IP_{\mathcal{L}^x,\tilde{c}_s}(\nu_s)$ *is* $(1,1,5,2)$. *The optimal value of* $IP_{\mathcal{L},\tilde{c}}^x(\nu)$ *is* $(2,1,1,1,5,2) \cdot (3,4,0,0,0,0) = 10$, *and the optimal value of* $IP_{\mathcal{L}^x,\tilde{c}_s}(\nu_s)$ *is* $(1,1,5,2) \cdot (2,0,1,1) = 9$, *and* $10 = 9 - y\nu$. *Equivalently, we can directly map any feasible solution of the integer program* $IP_{A,c}(b)$ *to a feasible solution of the lattice program* $IP_{\mathcal{L}^x,\tilde{c}_s}(\nu_s)$ *as follows: if* $x \in \mathcal{F}_A(b)$, *then* $s \in \mathcal{F}_{\mathcal{L}^x}(\nu_s)$ *where* $s = Ax - b$.

The lattice $\mathcal{L}^x$ *is saturated since* $\mathcal{L}^x = \{x \in \mathbb{Z}^4 : \tilde{A}x = \mathbf{0}\}$ *where*

$$\tilde{A} = \begin{pmatrix} 1 & 5 & 0 & 8 \\ 1 & 2 & 1 & 4 \end{pmatrix}.$$

So, we can reformulate the lattice program $IP_{\mathcal{L}^x,\tilde{c}_s}(\nu_s)$ *as the following integer program:* $\min\{\tilde{c}x, \tilde{A}x = \tilde{b}, x \geq \mathbf{0}, x \in \mathbb{Z}^4\}$ *where* $\tilde{b} = \tilde{A}\nu_s = (22, 16)$.

We have shown in this section that we can reformulate any integer program $IP^{\sigma}_{A,c}(b)$ as a lattice program $IP_{\mathcal{L},\tilde{c}}(\nu)$. The form $IP_{\mathcal{L},\tilde{c}}(\nu)$ is our preferred formulation for describing Gröbner bases of integer programs. We prefer it because all the variables in the formulation $IP_{\mathcal{L},\tilde{c}}(\nu)$ are non-negative and the non-negativity constraints on the variables are the only inequality constraints in the formulation. We have also effectively shown that we can reformulate the set of points $\mathcal{F}^{\sigma}_{A}(b)$ in the form $\mathcal{F}_{\mathcal{L}}(\nu)$, and this also is our preferred formulation for describing Markov bases of fibers for the same reasons as above.

2.10 Term orders

In this section, we define term orders, which may be thought of as perturbations of cost functions as we shall see. Terms orders as defined here are related to the standard concept of term orders in algebraic geometry where they are also known as monomial orders (see for example [28]).

A cost vector $c \in \mathbb{R}^n$ determines a partial order $\succ$ on $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$ defined by $x \succ y$ if $cx > cy$ for $x, y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$. Note that $x \succ y$ means $x \succeq y$ and $x \neq y$. Except in special circumstances (e.g. $n = 1$), $\succ$ is not a total order on $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$ since we may have $cx = cy$ but $x \neq y$ for some $x, y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, in which case, neither $x \succ y$ nor $y \succ x$. However, for computing Gröbner bases and Markov bases, we require a total order on $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$.

Specifically, for computing Gröbner bases and Markov bases, we need an order $\succ$ on $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$ that satisfies the following conditions:

(i). $\succ$ is a *total order* on the set $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$,

(ii). there exists a $\succ$-minimal solution of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$ for which $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu) \neq \emptyset$, and

(iii). $\succ$ is an *additive order* meaning that, for all $\nu \in \mathbb{Z}^n$, for all $x, y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, and for all $\gamma \in \mathbb{N}^n$, $x \succ y$ if and only if $x+\gamma \succ y+\gamma$ (note that $x+\gamma, y+\gamma \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu+\gamma)$).

We call an order $\succ$ that satisfies the above conditions a **term order** for $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ or just a term order for $\mathcal{L}$ when $\sigma = \emptyset$.

We can define an optimisation problem using a total order instead of a cost function as follows:

$$IP^{\sigma}_{\mathcal{L},\succ}(\nu) := \min_{\succ}\{x : x - \nu \in \mathcal{L}, x_{\bar{\sigma}} \geq \mathbf{0}, x \in \mathbb{Z}^n\}$$

where $\mathcal{L} \subseteq \mathbb{Z}^n$ is a lattice, $\sigma \subseteq \{1, ..., n\}$, $\succ$ is a total order on $\mathcal{F}^\sigma_\mathcal{L}(\nu)$,and $\nu \in \mathbb{Z}^n$. Here, we wish to find the $\succ$-minimal point in the fiber $\mathcal{F}_\mathcal{L}(\nu)$. We will also refer to this problem also as a lattice program. So, if $\succ$ is a term order, then from condition (ii) above, the lattice program $IP^\sigma_{\mathcal{L},\succ}(\nu)$ has an optimal solution for every feasible $\nu \in \mathbb{Z}^n$.

Crucially, for any lattice program $IP^\sigma_{\mathcal{L},c}(\nu)$ that has an optimal solution, there exists a term order $\succ$ such that the optimal solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$ is also the optimal solution of $IP^\sigma_{\mathcal{L},c}(\nu)$. We show this below. So, we can solve lattice programs with linear cost functions – which is one of our overall aims – by solving lattice programs with term orders.

Given a vector $c \in \mathbb{Z}^n$, we say that a term order $\succ$ is **compatible** with c if the optimal solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$ is also an optimal solution of $IP^\sigma_{\mathcal{L},c}(\nu)$ for all $\nu \in \mathbb{Z}^n$ where $\mathcal{F}^\sigma_\mathcal{L}(\nu) \neq \emptyset$. We can easily construct a c compatible order $\succ_c$ given some (tie-breaking) term order $\succ$ as follows: $x \succ_c y$ if $cx > cy$, or $cx = cy$ and $x \succ y$. We must be a little careful here though since $\succ_c$ is not necessarily a term order. The order $\succ_c$ satisfies conditions (i) and (iii) for being a term order, but condition (ii) is not always satisfied. The order $\succ_c$ is a term order if and only if $IP^\sigma_{\mathcal{L},c}(\nu)$ has an optimal solution for every $\nu \in \mathbb{Z}^n$ where $\mathcal{F}^\sigma_\mathcal{L}(\nu) \neq \emptyset$.

We can now derive a sufficient and necessary condition for a total additive order to be a term order, which is analogous to Corollary 2.9.2.

Lemma 2.10.1. *Let $\succ$ be a total additive order on the set $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ for every $\nu \in \mathbb{Z}^n$. The order $\succ$ is a term order if and only if there does not exist a vector $x \in \mathcal{F}^\sigma_\mathcal{L}(\mathbf{0}) = \{x \in \mathcal{L} : x_{\bar{\sigma}} \geq \mathbf{0}\}$ such that $\mathbf{0} \succ x$.*

Proof. Assume that there exists $x \in \mathcal{F}^\sigma_\mathcal{L}(\mathbf{0}) = \{x \in \mathcal{L} : x_{\bar{\sigma}} \geq \mathbf{0}\}$ such that $\mathbf{0} \succ x$. Also, assume that $IP^\sigma_{\mathcal{L},\succ}(\mathbf{0})$ has an optimal solution (a $\succ$-minimal solution) $x^* \in \mathcal{F}^\sigma_\mathcal{L}(\mathbf{0})$. Since $\succ$ is an additive order, $\mathbf{0} \succ x$ implies that $x^* \succ x + x^*$ contradicting the minimality of x^*. Therefore, $IP^\sigma_{\mathcal{L},\succ}(\mathbf{0})$ has no optimal solution, so $\succ$ is not a term order.

Assume that $\succ$ is not a term order. Then, there exists $\nu \in \mathbb{Z}^n$ such that $IP^\sigma_{\mathcal{L},\succ}(\nu)$ has no optimal solution. Hence, there must exists a $\succ$-decreasing infinite sequence of points in $\mathcal{F}^\sigma_\mathcal{L}(\nu)$; i.e. $(x^1, x^2, ..., x^i, ...)$ such that $x^i \succ x^{i+1}$ for all $i \in \mathbb{N}$. By the Gordan-Dickson Lemma 2.5.1, there must exist an $i < j$ such that $x^i_{\bar{\sigma}} \leq x^j_{\bar{\sigma}}$, in which case, $x^j - x^i \in \mathcal{F}^\sigma_\mathcal{L}(\mathbf{0})$. Since $\succ$ is an additive order, $x^i \succ x^j$ implies that $\mathbf{0} \succ (x^j - x^i)$ as required. □

The following corollary, which is analogous to Corollary 2.9.3, follows from the proof of Lemma 2.10.1 above.

Corollary 2.10.2. *Let $\succ$ be a total additive order on the set $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ for every $\nu \in \mathbb{Z}^n$. The order $\succ$ is a term order if and only if $IP^\sigma_{\mathcal{L},\succ}(\mathbf{0})$ has an optimal solution.*

Lemma 2.10.1 above implies that there exists a term order $\succ$ for $\mathcal{F}^\sigma_\mathcal{L}(\cdot)$ only if $\{x \in \mathcal{L} : x_{\bar{\sigma}} = \mathbf{0}\} = \{\mathbf{0}\}$ since otherwise if there exists $x \in \mathcal{L}$ such that $x_{\bar{\sigma}} = \mathbf{0}$, then $x \succ \mathbf{0}$ implies $\mathbf{0} \succ -x$ (by the additive property) and $-x \in \mathcal{F}^\sigma_\mathcal{L}(\mathbf{0})$, so $\succ$ cannot be

a term order by Lemma 2.10.1. Recall that the condition $\{x \in \mathcal{L} : x_{\bar{\sigma}} = \mathbf{0}\} = \{\mathbf{0}\}$ means that there is a one-to-one correspondence between points in $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ and points in $\mathcal{F}_{\mathcal{L}^\sigma}(\nu^\sigma)$ for every $\nu \in \mathbb{Z}^n$. So, if $\{x \in \mathcal{L} : x_{\bar{\sigma}} = \mathbf{0}\} = \{\mathbf{0}\}$, then a term order $\succ$ for $\mathcal{L}^\sigma$ determines a term order $\succ'$ for $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ defined by $x \succ' y$ if $x_{\bar{\sigma}} \succ y_{\bar{\sigma}}$ for all $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$, and conversely, any term order $\succ$ for $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ determines a term order $\succ'$ for $\mathcal{L}^\sigma$ defined by $x_{\bar{\sigma}} \succ' y_{\bar{\sigma}}$ if $x \succ y$ for all $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. Therefore, instead of solving $IP_{\mathcal{L},\succ}^{\sigma}(\nu)$, we can instead solve $IP_{\mathcal{L}^\sigma,\succ'}(\nu_{\bar{\sigma}})$ and lift the optimal solution from $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$ to $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. However, we will find it conceptually useful to sometimes still consider the lattice programs $IP_{\mathcal{L},\succ}^{\sigma}(\nu)$.

Total orders on $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ for every $\nu \in \mathbb{Z}^n$ are actually straight-forward to construct. Consider the cost vector $c \in \mathbb{R}^n$. As we mentioned above, the cost vector determines a partial order $\succ$ on every fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ for $\nu \in \mathbb{Z}^n$ defined by $x \succ y$ if $cx > cy$ for $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. We want a total order, so the situation we want to avoid is when $cx = cy$ but $x \neq y$ for $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ for some $\nu \in \mathbb{Z}^n$. We can choose another cost vector $c' \in \mathbb{R}^n$ that is linearly independent from c to try to break these ties; we define another order $\succ'$ such that $x \succ' y$ if $cx > cy$ or $cx = cy$ and $c'x > c'y$ for $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ and $\nu \in \mathbb{Z}^n$. The order $\succ'$ is a refinement of $\succ$ in the sense that $x \succ y$ implies $x \succ' y$. But, $\succ'$ still may not be a total order for every fiber, in which case, we can choose another cost vector $c'' \in \mathbb{R}^n$ that is linearly independent from c and c', and define another order $\succ''$ such that $x \succ'' y$ if $cx > cy$ or $cx = cy$ and $c'x > c'y$ or $cx = cy$ and $c'x = c'y$ and $c''x > c''y$ for $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ and $\nu \in \mathbb{Z}^n$. The order $\succ''$ is again a refinement of $\succ'$. We can keep choosing cost vectors in this way and continue to refine the order until we have a total order. At most, we need n linearly independent cost vectors $c^1, c^2, ..., c^n$ since if $c^i x = c^i y$ for $i = 1, ..., n$, then $x = y$.

What we are doing here is using a cost matrix as opposed to a cost vector. Let $C \in \mathbb{R}^{k \times n}$. Then, the matrix C determines a partial order $\succ_C$ on every fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ for $\nu \in \mathbb{Z}^n$ defined by $x \succ_C y$ if the first non-zero entry of the vector $C(x - y)$ is positive for $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ and for $\nu \in \mathbb{Z}^n$. This is the same approach as above for constructing an ordering from a sequence of cost vectors $c^1, c^2, ..., c^k$ where the rows of the matrix C are the cost vectors $c^1, c^2, ..., c^k$. A matrix C determines a total order if $\text{rank}(C) = n$ since, as above, if $C(x - y) = \mathbf{0}$, then $x = y$. Actually, in general, we do not need $\text{rank}(C) = n$. What we specifically need is that $C(x-y) = \mathbf{0}$ implies $x - y = \mathbf{0}$, and since $x - y \in \mathcal{L}$ for $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ and for $\nu \in \mathbb{Z}^n$, we require that $\mathcal{L} \cap \{x \in \mathbb{Z}^n : Cx = \mathbf{0}\} = \{\mathbf{0}\}$. The set $\mathcal{L} \cap \{x \in \mathbb{Z}^n : Cx = \mathbf{0}\}$ is also a lattice (as the intersection of two lattices) and we can check the condition that $\mathcal{L} \cap \{x \in \mathbb{Z}^n : Cx = \mathbf{0}\} = \{\mathbf{0}\}$ easily using linear algebra.

One can think of using a cost matrix as a way of perturbing a cost vector. Let $C \in \mathbb{R}^{k \times n}$, and let $c^i = C_i$ (i.e. c^i is the ith row of C). Now, let $\tilde{c} = c^1 + \epsilon c^2 + \epsilon^2 c^3 + ... + \epsilon^{n-1} c^n$ where $\epsilon \in \mathbb{R}_+$ is very small.[5] The vector $\tilde{c}$ may be viewed a perturbation of the cost vector c^1 because $\tilde{c}$ is *close* to c^1 since ϵ is *very small*. Consider a fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ for some $\nu \in \mathbb{Z}^n$ and assume for the moment that it has a finite number of feasible points. Then, for all $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$, $x \succ_C y$ if and only if $\tilde{c}x > \tilde{c}y$ when ϵ is chosen *small enough*. In practice, we never actually perturb a cost vector because we may run into numerical problems, but this is still a useful way of thinking about

[5]Here, the expression ϵ^i means ϵ to the power of i, and not the ith element in a sequence.

the order $\succ_C$.

The total order $\succ_C$ for some matrix $C \in \mathbb{R}^{k\times n}$ has property (iii) of term orders. If $x \succ y$ where $x, y \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$ for some $\nu \in \mathbb{Z}^n$, then $x+\gamma \succ y+\gamma$ for every $\gamma \in \mathbb{N}^n$ since $C((x+\gamma)-(y+\gamma)) = C(x-y)$. Also, the order $\succ_C$ may be a total order on $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ for every $\nu \in \mathbb{Z}^n$ and thus satisfy property (i) of term orders, but there may not be an optimal solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$ (i.e. a $\succ_C$-minimal solution) for every feasible $\nu \in \mathbb{Z}^n$ (property (ii)), in which case, $\succ_C$ is not a term order. We now examine how we can check whether $\succ_C$ satisfies property (ii). This following corollary follows directly from 2.10.1 after using the definition of the order $\succ_C$.

Corollary 2.10.3. *Let $\succ_C$ be a total order on the set $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ for every $\nu \in \mathbb{Z}^n$ for some matrix $C \in \mathbb{R}^{k\times n}$. The order $\succ_C$ is a term order if and only if there does not exist $x \in \mathcal{F}^\sigma_\mathcal{L}(\mathbf{0}) = \{x \in \mathcal{L} : x_{\bar\sigma} \geq \mathbf{0}\}$ such that the first non-zero entry of Cx is negative (i.e. $\mathbf{0} \succ_C x$).*

In this thesis, we only ever need to use term orders $\succ_C$ for some rational matrix C.

Consider the case where $\sigma = \emptyset$. If C is non-negative, then $Cx \geq \mathbf{0}$ for all $x \in \mathcal{F}_\mathcal{L}(\mathbf{0})$ since $x \geq \mathbf{0}$; thus, for any non-negative matrix C where $\mathcal{L} \cap \{x \in \mathbb{Z}^n : Cx = \mathbf{0}\} = \{\mathbf{0}\}$, $\succ_C$ is a term order of $\mathcal{L}$. Also, if the first row of the matrix C is a positive vector, then the first entry of Cx is positive since $x \geq \mathbf{0}$, and thus, for any matrix C with a positive first row where $\mathcal{L} \cap \{x \in \mathbb{Z}^n : Cx = \mathbf{0}\} = \{\mathbf{0}\}$, $\succ_C$ is a term order of $\mathcal{L}$. Using these two sufficient conditions for term orders, we now define two of the most useful term orders are the lexicographic term order and the degree reverse lexicographic term order (see for example [28]).

Definition 2.10.4 (Lexicographic Order)**.** *Let $\mathcal{L}$ be a sub-lattice of $\mathbb{Z}^n$ and let $\alpha, \beta \in \mathcal{F}_\mathcal{L}(\nu)$ for some $b \in \mathbb{Z}^n$. We say $\alpha \succ_{lex} \beta$ if the first non-zero component of the vector $\alpha - \beta$ is positive.*

The lexicographic order is the order $\succ_C$ where $C = I$ (the $n \times n$ identity matrix). From our discussion above, this must be a term order for any lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ since C is non-negative and $\operatorname{rank}(C) = n$.

Definition 2.10.5 (Degree Reverse Lexicographic Order)**.** *Let $\mathcal{L}$ be a sub-lattice of $\mathbb{Z}^n$ and let $\alpha, \beta \in \mathcal{F}_\mathcal{L}(\nu)$ for some $b \in \mathbb{Z}^n$. We say $\alpha \succ_{drlex} \beta$ if $\sum_{i=1}^n (\alpha_i - \beta_i) > 0$ or $\sum_{i=1}^n (\alpha_i - \beta_i) = 0$ and the first non-zero component in the vector $\alpha - \beta$ is negative.*

The degree reverse lexicographic order is the order $\succ_C$ where C is a matrix such that the first row of C is a vectors of all ones and the rest of the matrix is $-I$ (the negative of the $n \times n$ identity matrix I). From our discussion above, this must be a term order since $\operatorname{rank}(C) = n$ and the first row of C is positive.

Another situation in which a matrix C determines a term order $\succ_C$ is when $\mathcal{F}_\mathcal{L}(\mathbf{0}) = \{\mathbf{0}\}$ and $\mathcal{L} \cap \{x \in \mathbb{Z}^n : Cx = \mathbf{0}\} = \{\mathbf{0}\}$. If $\mathcal{F}_\mathcal{L}(\mathbf{0}) = \{\mathbf{0}\}$, then Corollary 2.10.3 above is trivially satisfied. Note that $\mathcal{F}_\mathcal{L}(\mathbf{0}) = \{\mathbf{0}\}$ means that $\mathcal{F}_\mathcal{L}(\nu)$ is finite for all $\nu \in \mathbb{Z}^n$, so it follows that any total additive order is a term order.

Definition 2.10.6 (Reverse Lexicographic Order)**.** *Let $\mathcal{L}$ be a sub-lattice of $\mathbb{Z}^n$ and let $\alpha, \beta \in \mathcal{F}_\mathcal{L}(\nu)$ for some $b \in \mathbb{Z}^n$. We say $\alpha \succ_{rlex} \beta$ if the first non-zero component in the vector $\alpha - \beta$ is negative.*

The reverse lexicographic order is the order $\succ_C$ where $C = -I$. This is a total order since $\text{rank}(C) = n$, and it is a term order if and only if $\mathcal{F}_{\mathcal{L}}(\mathbf{0}) = \{\mathbf{0}\}$.

All of the above term orders for $\mathcal{L}$ can be easily extended to term orders for $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ since, as we observed earlier, any term order on $\mathcal{L}^{\sigma}$ determines a term order on $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$.

Next, we demonstrate how one could solve $IP^{\sigma}_{\mathcal{L},\succ_C}(\nu)$ using standard integer programming techniques given $C \in \mathbb{R}^{k\times n}$ such that $\succ_C$ is a term order of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. This aim of this is to show that for any term order $\succ_C$ of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, that we can construct a non-negative matrix C' such that $C'_{*\sigma} = \mathbf{0}$ and $C'_{*\bar{\sigma}} \geq \mathbf{0}$ such that $\succ_{C'}$ is equivalent to $\succ_C$. Note that $C'_{*\sigma}$ is the matrix consisting of the columns of C' indexed by σ, and $C'_{*\bar{\sigma}}$ is the matrix consisting of the columns of C' indexed by $\bar{\sigma}$. Also, note that the term order $\succ_{C'_{*\bar{\sigma}}}$ is a term order on $\mathcal{L}^{\sigma}$ that is equivalent to the term order $\succ_C$ on $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, which confirms that, as we discussed, for any term order on $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, there is an equivalent term order on $\mathcal{L}^{\sigma}$.

Let x^* be the optimal solution of $IP^{\sigma}_{\mathcal{L},\succ_C}(\nu)$. Let $c^i = C_i$ (i.e. the ith row of C) for $i = 1, ..., k$. The first step is to solve the lattice program $IP^1 := IP^{\sigma}_{\mathcal{L},c^1}(\nu)$. If IP^1 has no optimal solution, then $x \succ_C y$ cannot be a term order because $c^1x > c^1y$ implies that $x \succ_C y$ for $x, y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$. So, an optimal solution exists because $\succ_C$ is a term order. Let x^1 be the optimal solution of IP^1 obtained by standard integer programming techniques. Then, we must have $c^1x^* = c^1x^1$. Next, consider the modified lattice program $IP^2 := \min\{c^2x : x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu), c^1x = c^1x^1\}$. Again, if IP^2 has no optimal solution, then $\succ_C$ cannot be a term order because $c^1x = c^1y$ and $c^2x > c^2y$ implies that $x \succ_C y$ for $x, y \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu) \cap \{x \in \mathbb{Z}^n : c^1x = c^1x^1\}$. Let x^2 be the optimal solution of IP^2. Then, $c^1x^2 = cx^*$ and $c^2x^2 = cx^*$. Next, we solve the modified lattice program $IP^3 := \min\{c^3x : x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu), c^1x = c^1x^1, c^2x = c^2x^2\}$ and so on until we solve $IP^k := \min\{c^kx : x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu), c^1x = c^1x^1, ..., c^{k-1}x = c^{k-1}x^{k-1}\}$. Let x^k be the optimal solution of IP^k; then, $x^k = x^*$ since $c^1x^k = c^1x^*, ..., c^kx^k = c^kx^*$ and $\succ_C$ is a total order.

We can write the modified lattice program $IP^2 := \min\{c^2x : x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu), c^1x = c^1x^1\}$ from above as a normal lattice program as follows: $IP^2 = IP_{\mathcal{L}^2,c^2}(x^1)$ where $\mathcal{L}^2 = \mathcal{L} \cap \{x \in \mathbb{Z}^n : c^1x = 0\}$. Note that $\mathcal{L}^2$ is a lattice since $\{x \in \mathbb{Z}^n : c^1x = 0\}$ is a lattice and the intersection of two lattices is a lattice. Let $\mathcal{F}^2 = \{x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu) : c^1x = c^1x^1\}$, so $IP^2 = \min\{c^2x : x \in \mathcal{F}^2\}$. We will show that $\mathcal{F}^2 = \mathcal{F}^{\sigma}_{\mathcal{L}^2}(x^1)$; then, it follows that $IP^2 = IP_{\mathcal{L}^2,c^2}(x^1)$. If $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ and $c^1x = c^1x^1$, then $x \in \mathcal{F}^{\sigma}_{\mathcal{L}^2}(x^1)$ since $x - x^1 \in \mathcal{L}^2$, and thus, $\mathcal{F}^2 \subseteq \mathcal{F}^{\sigma}_{\mathcal{L}^2}(x^1)$. If $x \in \mathcal{F}^{\sigma}_{\mathcal{L}^2}(x^1)$, then $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ since $\mathcal{L}^2 \subseteq \mathcal{L}$, and also, $c^1x = c^1x^1$ since $x - x^1 \in \mathcal{L}^2$, and thus, $x \in \mathcal{F}^2$. Thus, $\mathcal{F}^{\sigma}_{\mathcal{L}^2}(x^1) \subseteq \mathcal{F}^2$. Therefore, $\mathcal{F}^{\sigma}_{\mathcal{L}^2}(x^1) = \mathcal{F}^2$. and $IP^2 = IP_{\mathcal{L}^2,c^2}(x^1)$. Extending this argument, we have $IP^i = IP_{\mathcal{L}^i,c^i}(x^{i\text{-}1})$ where $\mathcal{L}^i = \mathcal{L} \cap \{x \in \mathbb{Z}^n : c^1x = 0, ..., c^{i\text{-}1}x = 0\}$ for all $i = 2, ..., k$.

Now, from Lemma 2.8.4, since IP^1 has an optimal solution, we can construct a cost vector $\bar{c}^1$ that is equivalent to c^1 where $\bar{c}^1_{\sigma} = \mathbf{0}$ and $\bar{c}^i \geq \mathbf{0}$. Similarly, since IP^i has an optimal solution for all $i = 1, ..., k$, we can construct a cost vector $\bar{c}^i$ that is equivalent to c^i where $\bar{c}^i_{\sigma} = \mathbf{0}$ and $\bar{c}^i \geq \mathbf{0}$. Then, the matrix $\bar{C}$ where $\bar{c}^i$ is the ith row of $\bar{C}$ determines an equivalent term order $\succ_{\bar{C}}$ to $\succ_C$ where $\bar{C}$ is non-negative and $\bar{C}_{*\sigma} = \mathbf{0}$ by construction.

Chapter 3

Markov bases

In this chapter, we formally define Markov bases of lattice fibers, Markov bases of lattices, and truncated Markov bases of lattices. The Markov bases that we define in this section are essentially the same objects as those described in the introduction, but here we define Markov bases in the context of lattice fibers. Most of this section consists of definitions of the different types of Markov bases with some examples. The concept of Markov bases of lattices is well-known as it is analogous to a well-known concept in algebraic geometry (see Appendix A). However, we found no explicit reference to a Markov basis of one lattice fiber in the literature, but we found this concept was quite useful in the context of integer programming. Also, the precise concept of a truncated truncated Markov bases of a lattice has not appeared in the literature to date as far as we are aware, but it is essentially the same as the concept of truncation for Gröbner bases (see Section 4.3).

3.1 Markov bases of lattice fibers

In this section, we define Markov bases of lattice fibers, and we show that we can restrict our attention to Markov bases of lattice fibers that have all non-negative components. Markov bases of lattice fibers are analogous to Markov bases of polyhedral fibers as we discussed in the introduction.

A Markov basis of a lattice fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ is a set of vectors $M \subseteq \mathcal{L}$ such that, for every pair of points $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$, there exists a sequence of points $(x^1, x^2, ..., x^k) \subseteq \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ where $x^1 = x$ and $x^k = y$ such that either $x^i - x^{i+1} \in M$ or $x^{i+1} - x^i \in M$ for all $i = 1, ..., k-1$. In other words, we can move from any point $x \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ to any other point $y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ in a finite number of steps via other points in $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ using the vectors in the Markov basis. We can move from one point to another by adding or subtracting a vector in the Markov basis. Note that we restrict Markov bases to be subsets of $\mathcal{L}$ since if $v \in \mathbb{Z}^n$ and $v = x - y$ for two points $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$, then $v \in \mathcal{L}$, so any vector not in $\mathcal{L}$ is useless.

We formally define Markov bases using the concept of fiber graphs.

Definition 3.1.1. *Given a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$, a vector $\nu \in \mathbb{Z}^n$, and a set $S \subseteq \mathcal{L}$, we define the* ***fiber graph*** $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, S)$ *as the* undirected *graph with nodes $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ and edges*

(x, y) if $x - y \in S$ or $y - x \in S$ for $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. If $\sigma = \emptyset$, we will usually omit it and write $\mathcal{G}_{\mathcal{L}}(\nu, S)$.

We define Markov bases using the notion of the connectedness of fiber graphs. Such a link between Markov bases and the connectedness of fiber graphs was made by Thomas and Weismantel in [88]. We remind the reader that connectedness of $\mathcal{G}_{\mathcal{L}}(\nu, M)$ means that between each pair $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ there exists a path from x to y in the graph $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, M)$, and a path in the graph fiber $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, M)$ is a sequence of points $(x^1, x^2, ..., x^k) \subseteq \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ such that either $x^i - x^{i+1} \in M$ or $x^{i+1} - x^i \in M$ for all $i = 1, ..., k - 1$.

Definition 3.1.2. *Given $\nu \in \mathbb{Z}^n$, we call a set $M \subseteq \mathcal{L}$ a* **Markov basis of** $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ *if the graph $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, M)$ is connected.*

The definition of a Markov basis above is the same as the description in the beginning of this section.

Remark 3.1.3. *In some circumstances, the Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ can be empty: the empty set is a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ if and only if $|\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)| \leq 1$ (i.e. $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ has zero or one points).*

Example 3.1.4. *Consider the lattice fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ where $\mathcal{L} \in \mathbb{Z}^n$ is the lattice generated by the two vectors $(1, 0, 2, -2, -2, 1)$ and $(0, 1, 3, 1, -1, -1)$, $\nu = (0, 0, -6, 4, 10, 1)$, and $\sigma = \{1, 2\}$. This is exactly the same lattice fiber that we encountered in Example 2.9.4. This lattice fiber corresponds to the polyhedral fiber in Example 1.5.1 as described in Example 2.9.4. There are 7 feasible solutions of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$:*

$$\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu) = \left\{ \begin{array}{cccccc} (& 1, & 2, & 2, & 4, & 6, & 0) \\ (& 2, & 1, & 1, & 1, & 5, & 2) \\ (& 2, & 2, & 4, & 2, & 4, & 1) \\ (& 2, & 3, & 7, & 3, & 3, & 0) \\ (& 3, & 2, & 6, & 0, & 2, & 2) \\ (& 3, & 3, & 9, & 1, & 1, & 1) \\ (& 3, & 4, & 12, & 2, & 0, & 0) \end{array} \right\}.$$

The set $M := \{(1, 0, 2, -2, -2, 1), (0, 1, 3, 1, -1, -1)\}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$; we can connect any two points in $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ by adding or subtracting vectors in M. For example, consider the points $(2, 1, 1, 1, 5, 2)$ and $(3, 2, 6, 0, 3, 2)$. We can step from $(2, 1, 1, 1, 5, 2)$ to $(2, 2, 4, 2, 4, 1)$ by adding $(0, 1, 3, 1, -1, -1)$, and then, we can step from $(2, 2, 4, 2, 4, 1)$ to $(3, 2, 6, 0, 3, 2)$ by adding the other vector $(1, 0, 2, -2, -2, 1)$.

The set $M' := \{(1, 0, 2, -2, -2, 1), (-1, 1, 1, 3, 1, -2)\}$ is not a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ since the point $(3, 4, 12, 2, 0, 0)$ is not connected to any other feasible point because adding or subtracting either vector in M' leads to an infeasible point (i.e. one of the $\bar{\sigma}$ components becomes non-negative).

There is a strong relationship between Markov bases of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ and Markov bases of $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$. Recall that $\mathcal{L}^{\sigma}$ is the projection of $\mathcal{L}$ onto the $\bar{\sigma}$ components, and $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$ is the projection of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ onto the $\bar{\sigma}$ components, which are the non-negative components of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. We can always construct a Markov basis of $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$ from a Markov

basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$, and conversely, we can always construct a Markov basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ from a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. We will explore this relationship here since it plays an important role in describing algorithms for computing Markov bases. This relationship has not been reported in the literature, but it is only a slight extension of known results.

First, we consider the case where $\ker(\pi_\sigma)\cap\mathcal{L} = \{\mathbf{0}\}$. Recall from Section 2.9 that the condition $\ker(\pi_\sigma)\cap\mathcal{L} = \{u\in\mathcal{L} : u_{\bar\sigma} = \mathbf{0}\} = \{\mathbf{0}\}$ means that there is a one-to-one correspondence between points in $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ and points in $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$, or in other words, the projective map $\pi_\sigma : \mathcal{L}\to\mathcal{L}^\sigma$ is a bijection. Note that $M_{\bar\sigma}$ is the projection of M onto the $\bar\sigma$ components in Lemma 3.1.5 below.

Lemma 3.1.5. *Given $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ where $\ker(\pi_\sigma)\cap\mathcal{L} = \{\mathbf{0}\}$, the set $M\subseteq\mathcal{L}$ is a Markov basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ if and only if $M_{\bar\sigma}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$.*

Proof. Let M be a Markov basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$. Paths in the fiber graph $\mathcal{G}^\sigma_\mathcal{L}(\nu,M)$ project to paths in the fiber graph $\mathcal{G}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma},M_{\bar\sigma})$; that is, if $(x^1,x^2,...,x^k)$ is a path in $\mathcal{G}^\sigma_\mathcal{L}(\nu,M)$, then $(x^1_{\bar\sigma},x^2_{\bar\sigma},...,x^k_{\bar\sigma})$ is a path in $\mathcal{G}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma},M_{\bar\sigma})$. Now, let $x',y'\in\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. There must exist $x,y\in\mathcal{F}^\sigma_\mathcal{L}(\nu)$ such that $x_{\bar\sigma}=x'$ and $y_{\bar\sigma}=y'$. Since M is a Markov basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$, there exists a path from x to y in $\mathcal{G}^\sigma_\mathcal{L}(\nu,M)$, which projects to a path from x' to y' in $\mathcal{G}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma},M_{\bar\sigma})$. Therefore, $M_{\bar\sigma}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$.

Conversely, assume that $M_{\bar\sigma}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. Paths in the fiber graph $\mathcal{G}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma},M_{\bar\sigma})$ lift to paths in the fiber graph $\mathcal{G}^\sigma_\mathcal{L}(\nu,M)$; that is, if $(\bar x^1,\bar x^2,...,\bar x^k)$ is a path in $\mathcal{G}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma},M_{\bar\sigma})$, then $(x^1,x^2,...,x^k)\subseteq\mathcal{F}^\sigma_\mathcal{L}(\nu)$ where $x^i_{\bar\sigma}=\bar x^i$ for $i=1,...,k$ is a path in $\mathcal{G}^\sigma_\mathcal{L}(\nu,M)$. Note that $\bar x^i-\bar x^{i+1}\in M_{\bar\sigma}$ implies that $x^i-x^{i+1}\in M$ since $\ker(\pi_\sigma)\cap\mathcal{L}=\{\mathbf{0}\}$, and similarly, $\bar x^{i+1}-\bar x^i\in M_{\bar\sigma}$ implies that $x^{i+1}-x^i\in M$ for $i=1,...,k-1$. Let $x,y\in\mathcal{F}^\sigma_\mathcal{L}(\nu)$. There must exist a path $(\bar x^1,\bar x^2,...,\bar x^k)$ from $x_{\bar\sigma}$ to $y_{\bar\sigma}$ in $\mathcal{G}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma},M_{\bar\sigma})$ since $M_{\bar\sigma}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. From above, this path lifts to a path from x to y in $\mathcal{G}^\sigma_\mathcal{L}(\nu,M)$. □

Example 3.1.6. *Consider again the lattice fiber $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ from Example 3.1.4 above. The projection of this fiber onto the $\bar\sigma$ variables is the lattice fiber $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$ where $\mathcal{L}^\sigma\subseteq\mathbb{Z}^n$ is the lattice generated by the two vectors* (2, -2, -2, 1) *and* (3, 1, -1, -1)*, and $\nu_{\bar\sigma}$ =* (-6, 4, 10, 1)*. Note that we have $\ker(\pi_\sigma)\cap\mathcal{L}=\{\mathbf{0}\}$. Recall from Example 2.9.4 that $\mathcal{L}^\sigma=\{x\in\mathbb{Z}^4 : \tilde Ax=\mathbf{0}\}$, so $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})=\mathcal{P}_{\tilde A}(\tilde b)=\{x\in\mathbb{Z}^4 : \tilde Ax=\tilde b, x\geq\mathbf{0}\}$ where*

$$\tilde A=\begin{pmatrix}1&5&0&8\\1&2&1&4\end{pmatrix},\ \text{and}\ \tilde b=\begin{pmatrix}22\\16\end{pmatrix}.$$

Projecting the fiber $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ onto the $\bar\sigma$ components, we arrive at the set $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$, which is the following set of vectors:

$$\{(2,4,6,0),(1,1,5,2),(4,2,4,1),(7,3,3,0),(6,0,2,2),(9,1,1,1),(12,2,0,0)\}.$$

Applying Lemma 3.1.5, we have that $M_{\bar\sigma}:=\{(2,-2,-2,1),(3,1,-1,-1)\}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$, and the set $M'_{\bar\sigma}:=\{(2,-2,-2,1),(1,3,1,-2)\}$ is not a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$.

If $\ker(\pi_\sigma)\cap\mathcal{L}\neq\{\mathbf{0}\}$, then we can still easily construct a Markov basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ given a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. Let $M\subseteq\mathcal{L}$ be such that $M_{\bar\sigma}$ is a Markov basis of

$\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$. Also, let S be a spanning set of the lattice $\ker(\pi_\sigma)\cap\mathcal{L} = \{u \in \mathcal{L} : u_{\bar{\sigma}} = \mathbf{0}\}$. We show that $M \cup S$ is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$.

Lemma 3.1.7. *Given $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ and a spanning set S of $\ker(\pi_\sigma)\cap\mathcal{L}$, the set $M \cup S \subseteq \mathcal{L}$ is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ if and only if $M_{\bar{\sigma}}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$.*

Proof. The proof that if $M \cup S \subseteq \mathcal{L}$ is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$, then $M_{\bar{\sigma}}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$ follows the proof of 3.1.5 after noting that $S_{\bar{\sigma}} = \{\mathbf{0}\}$, so $M \cup S$ effectively projects to just $M_{\bar{\sigma}}$.

Conversely, let $x, y \in \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$. There must exist a path $(\bar{x}^1, \bar{x}^2, ..., \bar{x}^k)$ from $x_{\bar{\sigma}}$ to $y_{\bar{\sigma}}$ in $\mathcal{G}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}}, M_{\bar{\sigma}})$ since $M_{\bar{\sigma}}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$. Let $(x^1, x^2, ..., x^k) \subseteq \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ where $x^i_{\bar{\sigma}} = \bar{x}^i$ for $i = 1, ..., k$. Unfortunately, $(x^1, x^2, ..., x^k)$ is not necessarily a path in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, M)$. We next show that there is a path in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, M \cup S)$ from x^i to x^{i+1} for $i = 1, ..., k-1$, which implies that there is a path from x to y in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, M \cup S)$ as required. Let $i \in \{1, ..., k-1\}$ such that $x^i_{\bar{\sigma}} - x^{i+1}_{\bar{\sigma}} \in M_{\bar{\sigma}}$, and let $m^i \in M$ be such that $m^i_{\bar{\sigma}} = x^i_{\bar{\sigma}} - x^{i+1}_{\bar{\sigma}}$. Let $z^{i+1} = x^i - m^i$; then, $z^{i+1} \in \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$, and $z^{i+1} - x^{i+1} \in \ker(\pi_\sigma)\cap\mathcal{L}$. If $x^{i+1} = z^{i+1}$, then we are done, so assume otherwise. Since S is a spanning set of $\ker(\pi_\sigma) \cap \mathcal{L}$, there must exist a path from z^{i+1} to x^{i+1} in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, S)$, and therefore, there is a path from x^i to x^{i+1} in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, M \cup S)$. Similarly, there exists a path from x^i to x^{i+1} in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, M \cup S)$ for all $i = 1, ..., k-1$ where $x^{i+1}_{\bar{\sigma}} - x^i_{\bar{\sigma}} \in M_{\bar{\sigma}}$ as required. □

Since we can always easily construct a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ from a Markov basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$, we can restrict our attention to Markov bases of $\mathcal{F}_{\mathcal{L}}(\nu)$ (all variables are non-negative).

3.2 Markov bases of lattices

In this section, we formally present Markov bases of sets of different but related fibers. Markov bases of lattices are analogous to the Markov bases of sets of polyhedral fibers as discussed in the introduction. We will first define Markov bases of lattices and give a simple example, and then, importantly, we will give known sufficient conditions for a set of vectors to be a Markov basis of a lattice.

Definition 3.2.1. *Given a lattice $\mathcal{L}$ and a set $\sigma \subseteq \{1, ..., n\}$, we call a set $M \subseteq \mathcal{L}$ a* **Markov basis of** *$\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ if M is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$. If $\sigma = \emptyset$, then we call M a Markov basis of $\mathcal{L}$.*

We will focus our attention on Markov bases of lattices (i.e. $\sigma = \emptyset$) since, from Lemma 3.1.5, assuming that $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$, $M \subseteq \mathcal{L}$ is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ if and only if $M_{\bar{\sigma}}$ is a Markov basis of $\mathcal{L}^\sigma$. Also, in the more general case where $\ker(\pi_\sigma) \cap \mathcal{L} \neq \{\mathbf{0}\}$, given a set S that spans $\ker(\pi_\sigma) \cap \mathcal{L}$ and a set $M \subseteq \mathcal{L}$, the set $M \cup S$ is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ if and only if $M_{\bar{\sigma}}$ is a Markov basis of $\mathcal{L}^\sigma$ from Lemma 3.1.7. So, we can always reconstruct a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ from a Markov basis of $\mathcal{L}^\sigma$.

Note the difference between a *Markov basis* of a lattice and a *spanning set* of a lattice: a spanning set of $\mathcal{L}$ is any set $M \subseteq \mathcal{L}$ such that any point in $\mathcal{L}$ can be

represented as a linear integer combination of the vectors in M. A Markov basis of $\mathcal{L}$ is a spanning set of $\mathcal{L}$, but the converse is not necessarily true.

Example 3.2.2. *Consider again the projected lattice $\mathcal{L}$ from Example 3.1.6 that is generated by the two vectors $(2, -2, -2, 1)$ and $(3, 1, -1, -1)$. Recall that the set $M = \{(2, -2, -2, 1), (3, 1, -1, -1)\}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ where $\nu = (-6, 4, 10, 1)$. The set M is a spanning set of $\mathcal{L}$, but we will show that it is not a Markov basis of $\mathcal{L}$.*

Consider $\nu' = (-6, 4, 10, -1)$. The fiber $\mathcal{F}_{\mathcal{L}}(\nu')$ has two feasible solutions: $(1, 1, 5, 0)$ and $(6, 0, 3, 2)$. This lattice fiber corresponds to the polyhedral fiber of Figure 1.10(a). M is not a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu')$ because there is no way to move from $(1, 1, 5, 0)$ to $(6, 0, 3, 2)$ using the vectors in M while staying within the set $\mathcal{F}_{\mathcal{L}}(\nu')$; in other words, the graph $\mathcal{G}_{\mathcal{L}}(\nu, M)$ is disconnected. Thus, we need the vector $(5, -1, -3, 0)$.

The set $M' = \{(2, -2, -2, 1), (3, 1, -1, -1), (5, -1, -3, 0)\}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ and $\mathcal{F}_{\mathcal{L}}(\nu')$, but it is still not a Markov basis of $\mathcal{L}$. Consider $\nu'' = (-6, 4, 5, 1)$. Again, the fiber $\mathcal{F}_{\mathcal{L}}(\nu'')$ has only two feasible solutions: $\{(2, 4, 1, 0), (1, 1, 0, 2)\}$. This lattice fiber corresponds to the polyhedral fiber of Figure 1.10(b). The set M' is not a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu'')$ since there is no way to move from $(2, 4, 1, 0)$ to $(1, 1, 0, 2)$ using the vectors in M'. Thus, we need the vector $(1, 3, 1, -2)$.

The set $M'' = \{(2, -2, -2, 1), (3, 1, -1, -1), (5, -1, -3, 0), (1, 3, 1, -2)\}$ is in fact a Markov basis of $\mathcal{L}$, but this fact is certainly not obvious even for such a small example.

In general, it is difficult to compute a Markov basis of $\mathcal{L}$ or to prove that a given set is a Markov basis of $\mathcal{L}$, but in some special cases, it is straight-forward to find a small Markov basis of a lattice using some simple well-known sufficient conditions. We will use these special cases for computing Markov bases (see Chapter 6). We describe these special cases below.

The following lemmas use the result that if M is a spanning set of $\mathcal{L}$, then M is a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ where $\sigma = \{1, ..., n\}$. This follows since there are no non-negativity constraints for $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu) = \{x \in \mathbb{Z}^n : x - \nu \in \mathcal{L}\}$ for $\nu \in \mathbb{Z}^n$, and thus, a spanning set of $\mathcal{L}$ is all that we need for a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$.

Lemma 3.2.3. *Let $M \subseteq \mathcal{L}$ be a spanning set of $\mathcal{L}$. If there is a vector $u \in M$ where $u > \mathbf{0}$, then M is a Markov basis of $\mathcal{L}$.*

Proof. Let $x, y \in \mathcal{G}_{\mathcal{L}}(\nu, M)$ for some $\nu \in \mathbb{Z}^n$. Since M is a spanning set of $\mathcal{L}$, there exists a path from x to y in $\mathcal{G}^{\sigma}_{\mathcal{L}}(\nu, M)$ where $\sigma = \{1, ..., n\}$. We need to show that there is a path in $\mathcal{G}^{\sigma}_{\mathcal{L}}(\nu, M)$ from x to y such that every point on the path is non-negative since such a path is then a valid path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, M)$. Let $(x^0, x^1, ..., x^k)$ be a path from x to y in $\mathcal{G}^{\sigma}_{\mathcal{L}}(\nu, M)$. If this path is non-negative, then we are done. Otherwise, we can add u to the front of the path and subtract u from the end of the path as many times as necessary so that the path becomes non-negative; in other words, let $l \in \mathbb{N}$ such that $x^i + lu \geq \mathbf{0}$ for all $i \in \{0, ..., k\}$, and consider the new path

$$(x^0, x^0 + u, x^0 + 2u, ..., x^0 + lu, x^1 + lu, ..., x^k + lu, x^k + (l-1)u, ..., x^k).$$

This path is non-negative since $x^0 = x \geq \mathbf{0}$, $x^k = y \geq \mathbf{0}$, and $x^i + lu \geq \mathbf{0}$ for all $i \in \{1, ..., k\}$. Therefore, we have found a path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, M)$ as required. □

The following lemma is a useful alternative result to Lemma 3.2.3.

Lemma 3.2.4. *Let $M = \{u^1, u^2, ..., u^k\} \subseteq \mathcal{L}$ be a spanning set of $\mathcal{L}$. If the first non-zero entry in the sequence $(u_i^1, u_i^2, ..., u_i^k)$ is positive for every $i \in \{1, ..., n\}$, then M is a Markov basis of $\mathcal{L}$.*

Proof. Let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$. Since M is a spanning set of $\mathcal{L}$, there exists a path from x to y in $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, M)$ where $\sigma = \{1, ..., n\}$. By construction, the first vector u^1 must be non-negative, and we can assume it is non-zero. Then, using u^1 as in the proof of Lemma 3.2.3, we can make the path from x to y in $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, M)$ non-negative on all components in $\text{supp}(u^1) = \{i \in \{1, ..., n\} : u_i^1 \neq \mathbf{0}\}$. If $\text{supp}(u) = \{1, ..., n\}$, then we have found a path in $\mathcal{G}_{\mathcal{L}}(\nu, M)$ from x to y. Otherwise, consider u^2. Note that, by construction, if $u_i^1 = 0$, then $u_i^2 \geq 0$. Also, we can assume $\text{supp}(u^2) \setminus \text{supp}(u^1) \neq \emptyset$ (i.e. $u_i^2 > 0$ and $u_i^1 = 0$ for some $i \in \{1, ..., n\}$) since otherwise we just reorder the vectors in M. Using u^2, we can make the path from x to y in $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, M)$ non-negative on $\text{supp}(u^2) \setminus \text{supp}(u^1)$, and then, using u^1, we can make this modified path non-negative on $\text{supp}(u^1)$ as well. So, now we have a path from x to y that is non-negative on $\text{supp}(u^1) \cup \text{supp}(u^2)$. Repeating a similar procedure with u^3, we can find a path from x to y that is non-negative on $\text{supp}(u^1) \cup \text{supp}(u^2) \cup \text{supp}(u^3)$. Continuing like this until u^k, we arrive at a path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, M)$. □

Note that if there exists a vector $u \in M$ such that $u > \mathbf{0}$, then M would satisfy the conditions of Lemma 3.2.4 ($u^1 = u$), so the previous sufficient condition is a special case of Lemma 3.2.4. Also, note that if a set M satisfies that conditions of Lemma 3.2.4, then there must exist a vector $u \in \mathcal{L}$ where $u > \mathbf{0}$ – we can construct such a vector u by taking an appropriate non-negative combination of the vectors in M.

Example 3.2.5. *If $\mathcal{L} \subseteq \mathbb{Z}^n$ is an n-dimensional lattice (i.e. there are n vectors in a basis of $\mathcal{L}$), then it is straight-forward to compute a Markov basis of $\mathcal{L}$. Let $B \in \mathbb{Z}^{n \times n}$ be an upper triangle matrix with positive diagonal entries and non-positive entries elsewhere such that the rows of B form a basis of $\mathcal{L}$ (i.e., B is in UHNF). We can always construct such a matrix B from any basis of $\mathcal{L}$ using the HNF algorithm (see Section 2.4). The matrix B satisfies the conditions of Lemma 3.2.4 (if we list the rows of B in reverse order), and therefore, B is a Markov basis of $\mathcal{L}$.*

3.3 Truncated Markov bases of lattices

Here, we explore truncation for Markov bases. A truncated Markov basis is analogous to a truncated Markov basis as discussed in the introduction. We closely follow the approach of Thomas and Weismantel in [85].

A **truncated Markov basis** is a special type of Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ for some $\nu \in \mathbb{Z}^n$ that is not necessarily a Markov basis of all fibers. Essentially, a truncated Markov basis with respect to the fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ is a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ after removing all vectors $u \in \mathcal{L}$ for which there does not exist $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ such that $x - y = u$. We call the act of removing such vectors *truncation*. More explicitly, let G be a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$; then, the set $M := \{u \in G : u = x - y \text{ for some } x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)\}$

is a truncated Markov basis. Any vector $u \in \mathcal{L}$ for which there does not exist $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ such that $u = x - y$ can never be an edge in a fiber graph – the vector u is essentially *too long* to fit in the feasible region. Hence, we never need such a vector u in a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. Therefore, M must be a Markov basis of the fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$.

The set M above is by default also a Markov basis of other related fibers. Let $\nu' \in \mathbb{Z}^n$ where $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu') \neq \emptyset$ and $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu - \nu') \neq \emptyset$. The set M is also a Markov basis of the fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu')$. Let $u \in \mathcal{L}$ for which there exists $x, y \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu')$ such that $u = x - y$, and let $\gamma \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu - \nu')$. Then, $x + \gamma, y + \gamma \in \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$, and moreover, $u = (x + \gamma) - (y + \gamma)$; thus, u would not be removed during truncation. So, any vector needed in a Markov basis of the fiber $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu')$ would not be removed by truncation, and therefore, M is still a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu')$. The set M is thus a Markov basis of the following set of fibers:

$$\mathcal{B}_{\mathcal{L}}^{\sigma}(\nu) := \{\nu' \in \mathbb{Z}^n : \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu') \neq \emptyset \text{ and } \mathcal{F}_{\mathcal{L}}^{\sigma}(\nu - \nu') \neq \emptyset\}.$$

If $\sigma = \emptyset$, we usually omit σ and write $\mathcal{B}_{\mathcal{L}}(\nu)$. We use this property of a truncated Markov basis as the defining property of truncated Markov bases.

Definition 3.3.1. *Given $\nu \in \mathbb{Z}^n$, we call a set $M \subseteq \mathcal{L}$ a ν-**truncated Markov basis of** $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ if G is a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu')$ for every $\nu' \in \mathcal{B}_{\mathcal{L}}^{\sigma}(\nu)$. If $\sigma = \emptyset$, we call M a ν-truncated Markov basis of $\mathcal{L}$.*

The notion of truncation has been explored in [92] and [85], but only for computing truncated Gröbner bases, and we apply it here to Markov bases. Also, the above observation that truncation involves omitting vectors that do not fit inside the feasible set was not described in [92] and [85].

If $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu) \neq \emptyset$, then $\nu \in \mathcal{B}_{\mathcal{L}}^{\sigma}(\nu)$ since $\mathcal{F}_{\mathcal{L}}^{\sigma}(\mathbf{0}) \neq \emptyset$. Therefore, a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ is by definition a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. But, a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ is not necessarily a ν-truncated Markov basis $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$; it may not contain all the necessary vectors. Moreover, a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ is not necessarily a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$. In the special case where $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu) = \emptyset$, we have $\mathcal{B}_{\mathcal{L}}^{\sigma}(\nu) = \emptyset$, which is consistent since by definition an empty set is a Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ if $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu) = \emptyset$.

We can restrict our attention to truncated Markov bases of lattices (i.e. $\sigma = \emptyset$). Importantly, $\nu' \in \mathcal{B}_{\mathcal{L}}^{\sigma}(\nu)$ if and only if $\nu'_{\bar{\sigma}} \in \mathcal{B}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$ because we have $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu') \neq \emptyset$ and $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu - \nu') \neq \emptyset$ if and only if $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu'_{\bar{\sigma}}) \neq \emptyset$ and $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}} - \nu'_{\bar{\sigma}}) \neq \emptyset$. Then, from Lemma 3.1.5, assuming that $\ker(\pi_{\sigma}) \cap \mathcal{L} = \{\mathbf{0}\}$, $M \subseteq \mathcal{L}$ is a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ if and only if $M_{\bar{\sigma}}$ is a $\nu_{\bar{\sigma}}$-truncated Markov basis of $\mathcal{L}^{\sigma}$. Also, in the more general case where $\ker(\pi_{\sigma}) \cap \mathcal{L} \neq \{\mathbf{0}\}$, given a set S that spans $\ker(\pi_{\sigma}) \cap \mathcal{L}$ and a set $M \subseteq \mathcal{L}$, the set $M \cup S$ is a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ if and only if $M_{\bar{\sigma}}$ is a $\nu_{\bar{\sigma}}$-truncated Markov basis of $\mathcal{L}^{\sigma}$ from Lemma 3.1.7. So, we can always reconstruct a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ from a $\nu_{\bar{\sigma}}$-truncated Markov basis of $\mathcal{L}^{\sigma}$. We now focus on truncated Markov bases of lattices.

Checking whether a given vector is too long and thus not needed in a truncated Gröbner basis is equivalent to solving a feasibility problem. Given a vector $u \in \mathcal{L}$, there exists $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where $x - y = u$ if and only if $\mathcal{F}_{\mathcal{L}}(\nu - u^+) \neq \emptyset$ since

$x - y = u$ means that $x = \gamma + u^+$ and $y = \gamma + u^-$ for some $\gamma \in \mathbb{N}^n$ in which case $\gamma \in \mathcal{F}_{\mathcal{L}}(\nu - u^+) \neq \emptyset$. Here, $u^+ \in \mathbb{N}^n$ is the positive part of u, or in other words, $u_i^+ = \max\{0, u_i\}$ for all $i = 1, ..., n$, and similarly, u^- is the negative part of u, so $u = u^+ - u^-$. Moreover, since $u^+ \in \mathcal{F}_{\mathcal{L}}(u^+) \neq \emptyset$, we have that there exists $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where $x - y = u$ if and only if $u^+ \in \mathcal{B}_{\mathcal{L}}(\nu)$.

Lemma 3.3.2. *Given a vector $u \in \mathcal{L}$ and $\nu \in \mathbb{Z}^n$, there exists $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where $x - y = u$ if and only if $\mathcal{F}_{\mathcal{L}}(\nu - u^+) \neq \emptyset$ or equivalently $u^+ \in \mathcal{B}_{\mathcal{L}}(\nu)$.*

Crucially, a ν-truncated Markov basis of $\mathcal{L}$ can be much smaller than a Markov basis of $\mathcal{L}$. In some situations, the ν-truncated Markov basis of $\mathcal{L}$ might be empty: a minimal ν-truncated Markov basis is empty if and only if $|\mathcal{F}_{\mathcal{L}}(\nu)| \leq 1$ (i.e. there are zero or one points in $\mathcal{F}_{\mathcal{L}}(\nu)$). On the other hand, in some situations, truncation has no effect – a ν-truncated Markov basis of $\mathcal{L}$ may be the same size as a Markov basis of $\mathcal{L}$. We can find such a situation as follows. Let M be a finite minimal Markov basis of $\mathcal{L}$. Next, choose a vector $\nu \in \mathbb{N}^n$ such that $\nu - u^+ \geq \mathbf{0}$ for every $u \in M$. Then, we have $\nu - u^+ \in \mathcal{F}_{\mathcal{L}}(\nu - u^+) \neq \emptyset$ for every $u \in M$. Therefore, we cannot remove any vectors from M and still have a ν-truncated Markov basis of $\mathcal{L}$, so M is also a minimal ν-truncated Markov basis of $\mathcal{L}$. Intuitively, the strength of truncation is directly related to the size of $\mathcal{F}_{\mathcal{L}}(\nu)$.

Example 3.3.3. *Consider again the lattice $\mathcal{L}$ from Example 3.2.2 that is generated by the two vectors* (2, -2, -2, 1) *and* (3, 1, -1, -1)*. The set*

$$M = \{(2, -2, -2, 1), (3, 1, -1, -1), (5, -1, -3, 0), (1, 3, 1, -2)\}$$

is a Markov basis of $\mathcal{L}$.

Consider $\nu = (-6, 4, 10, 1)$. In this case, there are no vectors in M that are truncated since, for every vector $m \in M$, there exists $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ such that $x - y = m$. For example, for the vector $m = (2, -2, -2, 1)$, we have $(6, 0, 3, 2), (4, 2, 4, 1) \in \mathcal{F}_{\mathcal{L}}(\nu)$ and $(2, -2, -2, 1) = (6, 0, 3, 2) - (4, 2, 4, 1)$.

Suppose that $\nu' = (-6, 4, 6, 1)$. Then, $\mathcal{F}_{\mathcal{L}}(\nu') = \{(2, 4, 2, 0), (1, 1, 1, 2), (4, 2, 0, 1)\}$. In this case, the vector (5, -1, -3, 0) *is truncated; the vector is too long to fit inside the feasible set. Thus, $M' = \{(2, -2, -2, 1), (3, 1, -1, -1), (1, 3, 1, -2)\}$ is a ν'-truncated Markov basis of $\mathcal{L}$.*

We now discuss the properties of the set $\mathcal{B}_{\mathcal{L}}(\nu)$ and provide some extra motivation for the definition of a truncated Markov basis. Towards this aim, we show that given a set $M \subseteq \mathcal{L}$, the connectivity of the graph $\mathcal{G}_{\mathcal{L}}(\nu, M)$ is strongly related to the connectivity of $\mathcal{G}_{\mathcal{L}}(\nu', M)$ for $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$.

Firstly, given $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$, we have $x + y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for every $x \in \mathcal{F}_{\mathcal{L}}(\nu')$ and $y \in \mathcal{F}_{\mathcal{L}}(\nu - \nu')$, which implies that $y + \mathcal{F}_{\mathcal{L}}(\nu') = \{y + x : x \in \mathcal{F}_{\mathcal{L}}(\nu')\} \subseteq \mathcal{F}_{\mathcal{L}}(\nu)$ for every $y \in \mathcal{F}_{\mathcal{L}}(\nu - \nu')$. So, given $M \subseteq \mathcal{L}$, any path $(x^0, ..., x^k)$ in $\mathcal{G}_{\mathcal{L}}(\nu', M)$ can be translated by y to a path $(x^0 + y, ..., x^k + y)$ in $\mathcal{G}_{\mathcal{L}}(\nu, M)$ for every $y \in \mathcal{F}_{\mathcal{L}}(\nu - \nu')$. Hence, if M is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu')$, then any two points in $y + \mathcal{F}_{\mathcal{L}}(\nu') \subseteq \mathcal{F}_{\mathcal{L}}(\nu)$ are connected in $\mathcal{G}_{\mathcal{L}}(\nu, M)$.

Furthermore, we have $\mathcal{F}_{\mathcal{L}}(\nu') + \mathcal{F}_{\mathcal{L}}(\nu - \nu') \subseteq \mathcal{F}_{\mathcal{L}}(\nu)$ where $\mathcal{F}_{\mathcal{L}}(\nu') + \mathcal{F}_{\mathcal{L}}(\nu - \nu') = \{x + y : x \in \mathcal{F}_{\mathcal{L}}(\nu'), y \in \mathcal{F}_{\mathcal{L}}(\nu - \nu')\}$. Note that $\nu - \nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$ when $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$. If M is both a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu')$ and a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu - \nu')$, then any two points in $\mathcal{F}_{\mathcal{L}}(\nu') + \mathcal{F}_{\mathcal{L}}(\nu - \nu')$ are connected in $\mathcal{G}_{\mathcal{L}}(\nu, M)$. This is shown as follows. Any two points in $\mathcal{F}_{\mathcal{L}}(\nu') + \mathcal{F}_{\mathcal{L}}(\nu - \nu')$ can be written in the form $x^1 + y^1$ and $x^2 + y^2$ where $x^1, x^2 \in \mathcal{F}_{\mathcal{L}}(\nu')$ and $y^1, y^2 \in \mathcal{F}_{\mathcal{L}}(\nu - \nu')$. Now, from above, the points $x^1 + y^1, x^2 + y^1 \in y^1 + \mathcal{F}_{\mathcal{L}}(\nu') \subseteq \mathcal{F}_{\mathcal{L}}(\nu)$ are connected in $\mathcal{G}_{\mathcal{L}}(\nu, M)$ and the points $x^2 + y^1, x^2 + y^2 \in x^2 + \mathcal{F}_{\mathcal{L}}(\nu - \nu') \subseteq \mathcal{F}_{\mathcal{L}}(\nu)$ are connected in $\mathcal{G}_{\mathcal{L}}(\nu, M)$; hence, the points $x^1 + y^1$ and $x^2 + y^2$ are connected in $\mathcal{G}_{\mathcal{L}}(\nu, M)$ as required.

We have just shown how the connectivity of $\mathcal{G}_{\mathcal{L}}(\nu', M)$ implies connectivity of $\mathcal{G}_{\mathcal{L}}(\nu, M)$ where $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$, so it is reasonable that, in order to compute a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$, we also compute a Markov basis of $\mathcal{G}_{\mathcal{L}}(\nu', M)$ for all $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$.

Chapter 4

Gröbner bases

In this chapter, we define and discuss the properties of Gröbner bases of fibers, Gröbner bases of lattices (see for example [25, 79, 81]), and truncated Gröbner bases (see [92, 85]). These concepts are analogous to the concepts of Gröbner as discussed in the introduction. We discuss how we can use Gröbner bases to solve integer programs. Most of the material here is well-known, which we try to present in a fresh clear and concise form. However, we found no explicit reference to a Gröbner basis of one lattice fiber in the literature, but we found this concept was quite useful in the context of integer programming. Also, we present a new result on the size of truncated Gröbner bases for equality knapsack problems (see Section 4.3).

4.1 Gröbner bases of lattice fibers

In this section, we discuss Gröbner bases of lattices fibers, their properties, and how they may be used to solve lattice programs. Gröbner bases of lattice fibers are analogous to Gröbner bases of polyhedral fibers as discussed in the introduction.

Definition 4.1.1. *Given a lattice $\mathcal{L}$, $\sigma \subseteq \{1, ..., n\}$, $\nu \in \mathbb{Z}^n$, and a term order $\succ$, we call $G \subseteq \mathbb{Z}^n$ a* **$\succ$-Gröbner basis** *of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ if, for every $x \in \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ that is a non-optimal solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$, there exists a vector $u \in G$ such that $x - u \in \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ and $x \succ x - u$.*

A Gröbner basis is thus a set of *improving* vectors. The definition of a Gröbner basis guarantees that if a feasible solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$ is not optimal, then we can improve it by some vector in the Gröbner basis, and a feasible solution must be optimal if we cannot improve it by some vector in the Gröbner basis. We can thus solve the lattice program $IP^\sigma_{\mathcal{L},\succ}(\nu)$ by first finding a feasible solution and then iteratively improving the solution using vectors in the Gröbner basis until we attain an optimal solution.

Note that we restrict Gröbner bases to be subsets of $\mathcal{L}$ since if $u \in \mathbb{Z}^n$ and $x - u \in \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ for some point $x \in \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$, then $u \in \mathcal{L}$, so any vector not in $\mathcal{L}$ is useless.

Example 4.1.2. *Consider again the lattice program $IP_{\mathcal{L},\succ_c}(\nu)$ from Example 2.9.4 where $\mathcal{L} \subseteq \mathbb{Z}^n$ is the lattice generated by the two vectors* (2, -2, -2, 1) *and* (3, 1, -1, -1)*,*

$c = (2, 0, 1, 1)$, and $\nu = (\text{-}6, 4, 10, 1)$. We leave the tie-breaking term order $\succ$ unspecified since we never need it for this example: the cost vector c is sufficient for a total order on $\mathcal{F}_{\mathcal{L}}(\nu)$. Recall from Example 2.9.4 that $\mathcal{L} = \{x \in \mathbb{Z}^4 : \tilde{A}x = \mathbf{0}\}$, and thus, $\mathcal{F}_{\mathcal{L}}(\nu) = \mathcal{P}_{\tilde{A}}(\tilde{b}) = \{x \in \mathbb{Z}^4 : \tilde{A}x = \tilde{b}, x \geq \mathbf{0}\}$ where

$$\tilde{A} = \begin{pmatrix} 1 & 5 & 0 & 8 \\ 1 & 2 & 1 & 4 \end{pmatrix}, \text{ and } \tilde{b} = \begin{pmatrix} 22 \\ 16 \end{pmatrix}.$$

There are 7 feasible points of $IP_{\mathcal{L},\succ_c}(\nu)$:

$$\{(2, 4, 6, 0), (1, 1, 5, 2), (4, 2, 4, 1), (7, 3, 3, 0), (6, 0, 2, 2), (9, 1, 1, 1), (12, 2, 0, 0)\}.$$

The point $(1, 1, 5, 2)$ is the optimal solution of $IP_{\mathcal{L},\succ_c}(\nu)$. From Example 3.1.4, the set $G := \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1)\}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$. But, it is not a $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ since we cannot improve the non-optimal feasible solution $(2, 4, 6, 0)$ by subtracting a vector in G: $(2, 4, 6, 0) - (2, \text{-}2, \text{-}2, 1) = (0, 6, 8, \text{-}1)$, which is not feasible, and $(2, 4, 6, 0) - (3, 1, \text{-}1, \text{-}1) = (\text{-}1, 3, 7, 1)$, which is also not feasible. The point $(2, 4, 6, 0)$ is the second best solution, so if we improved it, we would arrive at the optimal solution $(1, 1, 5, 2)$; thus, we need the vector $(2, 4, 6, 0) - (1, 1, 5, 2) = (1, 3, 1, \text{-}2)$ in any $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$.

The set $G' := \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (1, 3, 1, \text{-}2)\}$ is a $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ because we can improve every non-optimal solution of $\mathcal{F}_{\mathcal{L}}(\nu)$.

The *Normal Form algorithm* below (Algorithm 1) computes the optimal solution of a lattice program $IP^{\sigma}_{\mathcal{L},\succ}(\nu)$ given an initial feasible solution $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ and a set $G \subseteq \mathcal{L}$ that is a $\succ$-Gröbner basis of $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$.

Algorithm 1 Normal Form algorithm

Input: a lattice $\mathcal{L}$, a vector $\nu \in \mathbb{Z}^n$, a point $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, a term order $\succ$, and a set $G \subseteq \mathcal{L}$.

Output: a vector x^* such that $\nexists u \in G$ where $x^* - u \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ and $x^* \succ x^* - u$.

```
x* := x
while ∃u ∈ G such that x* − u ∈ F^σ_L(ν) and x* ≻ x* − u do
    x* := x* − u
end while
return x*
```

We write $\mathcal{NF}^{\sigma}_{\mathcal{L}}(x, G)$ for the output of the Normal Form algorithm, and we usually omit σ when $\sigma = \emptyset$ and write $\mathcal{NF}_{\mathcal{L}}(x, G)$. During the algorithm, we construct a $\succ$-decreasing sequence of feasible points $(x^1, x^2, ...) \subseteq \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ where x^i is the value of x^* at the start of the ith iteration of the algorithm. This sequence must terminate since $\succ$ is a term order, and therefore, the algorithm always terminates. When the algorithm terminates with $\mathcal{NF}^{\sigma}_{\mathcal{L}}(x, G) = x^*$, it guarantees that there does not exist a vector $u \in G$ such that $x^* - u \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ and $x^* \succ x^* - u$. Hence, if G is a $\succ$-Gröbner basis, the point $\mathcal{NF}^{\sigma}_{\mathcal{L}}(x, G) = x^*$ must be the optimal solution of $IP^{\sigma}_{\mathcal{L},\succ}(\nu)$ from the definition of Gröbner bases. Conversely, given some set $G \subseteq \mathcal{L}$, if $\mathcal{NF}^{\sigma}_{\mathcal{L}}(x, G)$ is *not* the optimal solution for some $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, then G cannot be a $\succ$-Gröbner

basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$. Therefore, G is a $\succ$-Gröbner basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ if and only if, for every $x \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$, the point $\mathcal{NF}^\sigma_\mathcal{L}(x, G)$ is the optimal solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$.

Algorithm 1 is not quite complete in the sense that it does not specify which $u \in G$ such that $x^* - u \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$ and $x^* \succ x^* - u$ to select when there are many such vectors. However, the method for selecting which u is not important for the correctness of the algorithm, so we leave it unspecified.

Dubé et al. in [31] give an upper bound on the number of iterations of the Normal Form algorithm. This upper bound is exponential in the size of the input data and in the dimension. Given a lattice $\mathcal{L}$, a vector $\nu \in \mathbb{Z}^n$, a point $x \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$, a matrix order $\succ_C$ for some non-negative matrix $C \in \mathbb{R}^{n\times n}_+$, and a finite set $G \subseteq \mathcal{L}$, there are at most $(1 + R\bar{C})$ iterations where $R = \frac{1}{r_1}(\frac{r_1}{r_2} + 1)^n$, $\bar{C} = \max_i\{C_i x\}$, $r_1 = \max\{C_i u : C_i u > 0, u \in G, i = 1, ..., n\}$, and $r_2 = \min\{C_i u : C_i u > 0, u \in G, i = 1, ..., n\}$. Recall that every term order we consider is a matrix order for some non-negative matrix (see Section 2.10). It is not known whether this bound is tight.

Recall from Section 2.9 that we can always reformulate a lattice program $IP^\sigma_{\mathcal{L},c}(\nu)$ in the form $IP_{\mathcal{L}^\sigma,c_{\bar\sigma}}(\nu_{\bar\sigma})$, and recall from Section 2.10 we can also reformulate a lattice program $IP^\sigma_{\mathcal{L},\succ}(\nu)$ in the form $IP_{\mathcal{L}^\sigma,\succ}(\nu_{\bar\sigma})$. Recall that we can effectively use the same term order $\succ$ for $IP^\sigma_{\mathcal{L},\succ}(\nu)$ and $IP_{\mathcal{L}^\sigma,\succ}(\nu_{\bar\sigma})$ since there must be a one-to-one correspondence between feasible solutions of $IP^\sigma_{\mathcal{L},\succ}(\nu)$ and feasible solutions of $IP_{\mathcal{L}^\sigma,\succ}(\nu_{\bar\sigma})$ if $\succ$ is a term order (see Section 2.10). So, strictly speaking, we never need $\succ$-Gröbner bases of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ where $\sigma \neq \emptyset$. However, we will find such Gröbner bases very useful as a conceptual tool in the Chapter 6 when describing computing Markov bases and in Section 7.4 when solving integer programs.

There is a strong relationship between $\succ$-Gröbner bases of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ and $\succ$-Gröbner bases of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. Recall that $\mathcal{L}^\sigma$ is the projection of $\mathcal{L}$ onto the $\bar\sigma$ components, and $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$ is the projection of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ onto the $\bar\sigma$ components, which are the non-negative components of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$.

Lemma 4.1.3. *Given $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ where $\ker(\pi_\sigma)\cap\mathcal{L} = \{\mathbf{0}\}$, the set $G \subseteq \mathcal{L}$ is a $\succ$-Gröbner basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ if and only if $G_{\bar\sigma}$ is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$.*

Proof. Recall that $\ker(\pi_\sigma)\cap\mathcal{L} = \{\mathbf{0}\}$ means that there is a one-to-one correspondence between points in $\mathcal{F}^\sigma_\mathcal{L}(\nu)$ and points in $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. Let G be a $\succ$-Gröbner basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$. Let $x \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$ be such that $x_{\bar\sigma}$ is a non-optimal solution of $IP_{\mathcal{L}^\sigma,\succ}(\nu_{\bar\sigma})$. Then, x is a non-optimal solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$. Thus, there exists a vector $u \in G$ such that $x - u \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$ and $x \succ x - u$. This implies that $x_{\bar\sigma} - u_{\bar\sigma} \in \mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$ and $x_{\bar\sigma} \succ x_{\bar\sigma} - u_{\bar\sigma}$. Thus, since $u_{\bar\sigma} \in G_{\bar\sigma}$, $G_{\bar\sigma}$ is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$.

Conversely, assume that $G_{\bar\sigma}$ is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$. Let $x \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$ be a non-optimal solution of $IP^\sigma_{\mathcal{L},\succ}(\nu)$. Then, $x_{\bar\sigma}$ is a non-optimal solution of $IP_{\mathcal{L}^\sigma,\succ}(\nu_{\bar\sigma})$. Thus, there exists $u \in G$ such that $x_{\bar\sigma} - u_{\bar\sigma} \in \mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar\sigma})$ and $x_{\bar\sigma} \succ x_{\bar\sigma} - u_{\bar\sigma}$. This implies that $x - u \in \mathcal{F}^\sigma_\mathcal{L}(\nu)$ and $x \succ x - u$. Thus, G is a $\succ$-Gröbner basis of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$. □

Lemma 4.1.3 above means that we can focus our attention on $\succ$-Gröbner bases of $\mathcal{F}_\mathcal{L}(\nu)$ since any results or properties of $\succ$-Gröbner bases of $\mathcal{F}_\mathcal{L}(\nu)$ easily transfer to $\succ$-Gröbner bases of $\mathcal{F}^\sigma_\mathcal{L}(\nu)$.

Note that given $u \in \mathcal{L}$ and $x \in \mathcal{F}_{\mathcal{L}}(\nu)$ where $x - u \in \mathcal{F}_{\mathcal{L}}(\nu)$, we have $x \succ x - u$ if and only if $u^+ \succ u^-$. Recall that $u^+ \in \mathbb{N}^n$ is the positive part of u and $u^- \in \mathbb{N}^n$ is the negative part and that $u^+, u^- \in \mathcal{F}_{\mathcal{L}}(u^+) = \mathcal{F}_{\mathcal{L}}(u^-)$. This property follows since $\succ$ is an additive order: $x \succ x - u \Leftrightarrow x + u^+ \succ x + u^- \Leftrightarrow u^+ \succ u^-$. Thus, for Gröbner bases, we only need to consider vectors in the set $\mathcal{L}_\succ := \{u \in \mathcal{L} : u^+ \succ u^-\}$. Secondly, note that given a vector $u \in \mathcal{L}$ and a feasible point $x \in \mathcal{F}_{\mathcal{L}}(\nu)$, we have $x - u \in \mathcal{F}_{\mathcal{L}}(\nu)$ if and only if $u^+ \leq x$ since $x - u \in \mathcal{F}_{\mathcal{L}}(\nu) \Leftrightarrow \mathbf{0} \leq x - u \Leftrightarrow u^+ \leq x$. So, we arrive at the following straight-forward but very useful lemma.

Lemma 4.1.4. *Given a lattice $\mathcal{L}$, $\nu \in \mathbb{Z}^n$, and a term order $\succ$, a set $G \subseteq \mathcal{L}_\succ$ is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ if and only if, for every non-optimal solution $x \in \mathcal{F}_{\mathcal{L}}(\nu)$ of $IP_{\mathcal{L},\succ}(\nu)$, there exists a vector $u \in G$ such that $u^+ \leq x$.*

Conceptually, given a set $G \subseteq \mathcal{L}_\succ$, the Normal Form algorithm, Algorithm 1 constructs a finite $\succ$-decreasing path in the graph $\mathcal{G}_{\mathcal{L}}(\nu, G)$ from x to $x^* = \mathcal{NF}_{\mathcal{L}}(x, G)$; that is, the Normal Form algorithm constructs a $\succ$-decreasing sequence of feasible points $(x^1, x^2, ..., x^k) \subseteq \mathcal{F}_{\mathcal{L}}(\nu)$ where $x^1 = x$, the given feasible solution, and $x^k = x^*$, the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$.

Example 4.1.5. *Consider again the lattice program $IP_{\mathcal{L},\succ_c}(\nu)$ from Example 4.1.2 above, and again, let $G' := \{(2, -2, -2, 1), (3, 1, -1, -1), (1, 3, 1, -2)\}$. Consider the point $(12, 2, 0, 0) \in \mathcal{F}_{\mathcal{L}}(\nu)$. We now apply the Normal Form algorithm 1 above for the point $(12, 2, 0, 0)$ and the set G':*

$$\begin{aligned}(12,2,0,0) &\xrightarrow{(3,1,-1,-1)} (9,1,1,1)\\ &\xrightarrow{(3,1,-1,-1)} (6,0,2,2)\\ &\xrightarrow{(2,-2,-2,1)} (4,2,4,1)\\ &\xrightarrow{(2,-2,-2,1)} (2,4,6,0)\\ &\xrightarrow{(1,3,1,-2)} (1,1,5,2)\end{aligned}$$

The sequence of feasible solutions

$$((12,2,0,0),(9,1,1,1),(6,0,2,2),(4,2,4,1),(2,4,6,0),(1,1,5,2))$$

is a $\succ_c$-decreasing path in the fiber graph $\mathcal{G}_{\mathcal{L}}(\nu, G')$ from $(12, 2, 0, 0)$ to $(1, 1, 5, 2)$ (the optimal solution). This is not the only possible path from $(12, 2, 0, 0)$ to $(1, 1, 5, 2)$.

$$\begin{aligned}(12,2,0,0) &\xrightarrow{(3,1,-1,-1)} (9,1,1,1)\\ &\xrightarrow{(2,-2,-2,1)} (7,3,3,0)\\ &\xrightarrow{(3,1,-1,-1)} (4,2,4,1)\\ &\xrightarrow{(3,1,-1,-1)} (1,1,5,2)\end{aligned}$$

The sequence $((12, 2, 0, 0), (9, 1, 1, 1), (7, 3, 3, 0), (4, 2, 4, 1), (1, 1, 5, 2))$ is also a $\succ_c$-decreasing path in the fiber graph $\mathcal{G}_{\mathcal{L}}(\nu, G')$ from $(12, 2, 0, 0)$ to $(1, 1, 5, 2)$. It does not matter which particular path we choose; we are guaranteed to finish at the optimal solution by the properties of Gröbner bases.

We can now equivalently characterise Gröbner bases in terms of paths in the graph $\mathcal{G}_{\mathcal{L}}(\nu, G)$ in Lemma 4.1.6 below.

Lemma 4.1.6. *Given a lattice $\mathcal{L}$, $\nu \in \mathbb{Z}^n$, and a term order $\succ$, a set $G \subseteq \mathcal{L}_{\succ}$ is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ if and only if, for every $x \in \mathcal{F}_{\mathcal{L}}(\nu)$, there exists a $\succ$-decreasing path in $\mathcal{G}_{\mathcal{L}}(\nu, G)$ from x to the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$.*

Importantly, if $G \subseteq \mathcal{L}_{\succ}$ is a $\succ$-Gröbner basis, then G is a Markov basis of $\mathcal{L}$ since given $x, y \in \mathcal{G}_{\mathcal{L}}(\nu, G)$ for some $\nu \in \mathbb{Z}^n$, there exists a $\succ$-decreasing path from x to the unique optimal solution in $\mathcal{F}_{\mathcal{L}}(\nu)$ and from y to the same optimal solution, and thus, x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, G)$.

Corollary 4.1.7. *Given a lattice $\mathcal{L}$ and a vector $\nu \in \mathbb{Z}^n$, if a set $G \subseteq \mathcal{L}_{\succ}$ is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ for some term order $\succ$, then G is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$.*

4.2 Gröbner bases of lattices

In this section, we discuss Gröbner bases of lattices, which have very interesting properties that are not true of Gröbner bases of fibers. Gröbner bases of lattices are analogous to Gröbner bases of set of integer programs where the constraint matrix is fixed and the constant terms in constraints are allowed to vary, as discussed in the introduction. The results in this section are known, and we try to clearly and concisely present the most important results.

Definition 4.2.1. *Given a lattice $\mathcal{L}$, $\sigma \subseteq \{1, ..., n\}$, $\nu \in \mathbb{Z}^n$, and a term order $\succ$, we call $G \subseteq \mathcal{L}$ a* **$\succ$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$** *if it is a Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$ for every $\nu \in \mathbb{Z}^n$. If $\sigma = \emptyset$, then we call G a $\succ$-Gröbner basis of $\mathcal{L}$.*

A Gröbner basis of a lattice is simultaneously a Gröbner basis of every possible fiber.

We will focus our attention on Gröbner bases of lattices (i.e. $\sigma = \emptyset$) since, from Lemma 3.1.5, assuming that $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$, $G \subseteq \mathcal{L}$ is a $\succ$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ if and only if $G_{\bar{\sigma}}$ is a $\succ$-Gröbner basis of $\mathcal{L}^{\sigma}$.

Example 4.2.2. *Consider again the lattice program $IP_{\mathcal{L},\succ_c}(\nu)$ from Example 2.9.4 and Example 4.1.2 above. Recall that $G' := \{(2, -2, -2, 1), (3, 1, -1, -1), (1, 3, 1, -2)\}$ is a $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$. However, we now show that G' is not a $\succ_c$-Gröbner basis of $\mathcal{L}$.*

Consider $\nu' = (-6, 4, 10, -1)$. There are two feasible solutions of $\mathcal{F}_{\mathcal{L}}(\nu')$: $(1, 1, 5, 0)$ and $(6, 0, 3, 2)$. Here, $(1, 1, 5, 0)$ is the optimal solution of $IP_{\mathcal{L},\succ_c}(\nu')$. The set G' is not a $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu')$ since there is no vector in G' that can improve the non-optimal solution $(6, 0, 3, 2)$. The vector $(1, 3, 1, -2)$ is the only vector that can improve $(6, 0, 3, 2)$; thus, any $\succ_c$-Gröbner basis of $\mathcal{L}$ must include the vector $(5, -1, -3, 0)$.

Let $G'' := \{(2, -2, -2, 1), (3, 1, -1, -1), (1, 3, 1, -2), (5, -1, -3, 0)\}$. The set G'' is a $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ and $\mathcal{F}_{\mathcal{L}}(\nu')$, but it is not a $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu'')$ where

$\nu'' = (\text{-}8, 5, 7, 1)$. *There are four feasible solutions:* $(0, 5, 3, 0)$, $(2, 3, 1, 1)$, $(5, 4, 0, 0)$, *and* $(1, 0, 0, 3)$. *The optimal solution is* $(0, 5, 3, 0)$, *and* G'' *is not a* $\succ_c$*-Gröbner basis of* $\mathcal{F}_{\mathcal{L}}(\nu'')$ *since no vector in* G'' *can improve the non-optimal solution* $(1, 0, 0, 3)$. *The point* $(1, 0, 0, 3)$ *is the second best optimal solution, so we need the vector* $(1, \text{-}5, \text{-}3, 3)$ *in any* $\succ_c$*-Gröbner basis of* $\mathcal{F}_{\mathcal{L}}(\nu'')$.

Let $G''' := \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (1, 3, 1, \text{-}2), (5, \text{-}1, \text{-}3, 0), (1, \text{-}5, \text{-}3, 3)\}$. *Again, the set* G''' *is still not a* $\succ_c$*-Gröbner basis of* $\mathcal{L}$. *Consider* $\nu''' = (\text{-}6, 6, 6, 2)$. *There are six feasible solutions:*

$$\mathcal{F}_{\mathcal{L}}(\nu''') = \{(2, 6, 2, 1), (1, 3, 1, 3), (4, 4, 0, 2), (0, 0, 0, 5), (5, 7, 1, 0), (0, 8, 4, 0)\}.$$

The optimal solution is $(0, 8, 4, 0)$. *The set* G''' *is not a* $\succ_c$*-Gröbner basis of* $\mathcal{F}_{\mathcal{L}}(\nu''')$ *since the point* $(0, 0, 0, 5)$ *cannot be improved by a vector in* G'''. *This is the second best solution, and thus, we need the vector* $(0, \text{-}8, \text{-}4, 5)$.

Finally, the set

$$G'''' = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (1, 3, 1, \text{-}2), (5, \text{-}1, \text{-}3, 0), (1, \text{-}5, \text{-}3, 3), (0, \text{-}8, \text{-}4, 5)\}$$

is a Gröbner basis of $\mathcal{L}$, *but this is certainly not obvious.*

Gröbner bases of lattices have some special properties. Firstly, we introduce some new notation. Given a set of vectors $S \subseteq \mathbb{Z}^n$, we define $S^+ := \{s^+ : s \in S\}$ and similarly $S^- := \{s^- : s \in S\}$.

Lemma 4.2.3. *Given a lattice* $\mathcal{L}$, *and a term order* $\succ$, *a set* $G \subseteq \mathcal{L}_\succ$ *is a Gröbner basis of* $\mathcal{L}$ *if and only if* $G^+ \leq \mathcal{L}^+_\succ$, *which means that, for every vector* $v \in \mathcal{L}_\succ$, *there exists a vector* $u \in G$ *such that* $u^+ \leq v^+$.

Proof. Assume G is a $\succ$-Gröbner basis of $\mathcal{L}$. Let $v \in \mathcal{L}_\succ$, so $v^+ \succ v^-$ and $v^+, v^- \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$ (for example, $\nu = v^+$ or $\nu = v^-$). Hence, $v^+ \in \mathcal{L}^+_\succ$ is not an optimal solution of $IP_{\mathcal{L},\succ}(\nu)$, and therefore, there exists a vector $u \in G$ such that $u^+ \leq v^+$.

Conversely, assume that $G^+ \leq \mathcal{L}^+_\succ$. We will show that G is a $\succ$-Gröbner basis of $\mathcal{L}$ by Lemma 4.1.4. Let $x \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$, and let x^* be the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$. If $x = x^*$, then there is nothing to show, so assume $x \neq x^*$. Now $x - x^* \in \mathcal{L}_\succ$, and thus, there exists a vector $u \in G$ such that $u^+ \leq (x - x^*)^+$ which implies that $u^+ \leq x$ as required (see Lemma 4.1.4). □

Most importantly, using these properties, we can show that for every lattice $\mathcal{L}$ and every term order $\succ$ there exists a finite Gröbner basis.

Lemma 4.2.4. *For every lattice* $\mathcal{L}$ *and term order* $\succ$, *there exists a finite* $\succ$*-Gröbner basis of* $\mathcal{L}$.

Proof. From the Gordan-Dickson's lemma 2.5.1, there must exists a finite set of vectors $V \subseteq \mathcal{L}^+_\succ$ such that $V \leq \mathcal{L}^+_\succ$. Let $G \subseteq \mathcal{L}_\succ$ be such that $G^+ = V$ and $|G| = |V|$. Therefore, from Lemma 4.2.3, the set G is a finite $\succ$-Gröbner basis of $\mathcal{L}$. □

If G is a $\succ$-Gröbner basis of $\mathcal{L}$ such that there exist vectors $u, v \in G$ where $u \neq v$ and $u^+ \leq v^+$, then the set $G' = G \setminus \{v\}$ is still a Gröbner basis since we must still have $G'^+ \leq \mathcal{L}^+_\succ$. If we remove all such vectors v, we then arrive at a **minimal** $\succ$-Gröbner basis of $\mathcal{L}$ (minimal with respect to set inclusion).

Lemma 4.2.5. *Given a lattice $\mathcal{L}$, and a term order $\succ$, if $G, H \subseteq \mathcal{L}_\succ$ are minimal $\succ$-Gröbner bases of $\mathcal{L}$, then $|G| = |H|$, and moreover, $G^+ = H^+$.*

Proof. Let $G \subseteq \mathcal{L}_\succ$ be a minimal $\succ$-Gröbner basis of $\mathcal{L}$. Then, from Lemma 4.2.3, G must be an inclusion-wise minimal subset of $\mathcal{L}_\succ$ such that $G^+ \leq \mathcal{L}^+_\succ$. Now, from the Gordan-Dickon's lemma 2.5.1, there exists a *unique* minimal set $V \subseteq \mathcal{L}^+_\succ$ such that $V \leq \mathcal{L}^+_\succ$. We must have $G^+ = V$ and $|G| = |V|$ otherwise G would not be minimal. The same is true for any minimal $\succ$-Gröbner basis of $\mathcal{L}$, so the result follows. □

In general, minimal $\succ$-Gröbner bases of $\mathcal{L}$ are not unique since the negative components of their vectors may differ ($G^- \neq H^-$) even though the positive components are identical ($G^+ = H^+$). There is, however, a type of minimal Gröbner basis called a *reduced* Gröbner basis that is unique and which has some very interesting properties.

Definition 4.2.6. *Given a lattice $\mathcal{L}$ and a term order $\succ$, let $G \subseteq \mathcal{L}^+_\succ$ be a $\succ$-Gröbner basis of $\mathcal{L}$. The set G is **reduced** if G is minimal and for every $u \in G$, we have $G^+ \not\leq u^-$ where $G^+ \not\leq u^-$ means that there does not exist a vector $v \in G$ such that $v^+ \leq u^-$.*

The first interesting property of a reduced $\succ$-Gröbner basis $G \subseteq \mathcal{L}^+_\succ$ is that for each $u \in G$, u^- is the optimal solution of $IP_{\mathcal{L},\succ}(u^+) = IP_{\mathcal{L},\succ}(u^-)$ since G is a $\succ$-Gröbner basis of the fiber $\mathcal{F}_\mathcal{L}(u^+) = \mathcal{F}_\mathcal{L}(u^-)$ and $G^+ \not\leq u^-$. This property implies that reduced Gröbner bases are unique since G^+ is unique by Lemma 4.2.5 and optimal solutions are also unique.

Lemma 4.2.7. *Given a lattice $\mathcal{L}$ and a term order $\succ$, there is a unique reduced $\succ$-Gröbner basis of $\mathcal{L}$.*

The previous lemmas provide deep insights into the structure of all the optimal solutions of $IP_{\mathcal{L},\succ}(\nu)$ for different $\nu \in \mathbb{Z}^n$. Let G be a minimal $\succ$-Gröbner basis of $\mathcal{L}$, and let $x \in \mathcal{F}_\mathcal{L}(\nu)$. The point x is a non-optimal of $IP_{\mathcal{L},\succ}(\nu)$ if and only if $G^+ \leq x$. Thus, if x is non-optimal, then $x + \gamma$ is a non-optimal solution of $IP_{\mathcal{L},\succ}(\nu + \gamma)$ for all $\gamma \in \mathbb{N}^n$ since we must have $G^+ \leq (x + \gamma)$. Moreover, for every vector $u \in G$, the vector u^+ is a non-optimal solution of $IP_{\mathcal{L},\succ}(u^+)$. So, we can completely determine the collective set $\mathcal{N}_{\mathcal{L},\succ}$ of all non-optimal solutions of $IP_{\mathcal{L},\succ}(\nu)$ for every $\nu \in \mathbb{Z}^n$ as follows:

$$\mathcal{N}_{\mathcal{L},\succ} = \{u^+ + \gamma : u \in G, \gamma \in \mathbb{N}^n\}.$$

This also gives us a description of the set $\mathcal{O}_{\mathcal{L},\succ}$ consisting of the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$ for every $\nu \in \mathbb{Z}^n$ as $\mathcal{O}_{\mathcal{L},\succ} = \mathbb{N}^n \setminus \mathcal{N}_{\mathcal{L},\succ}$ since every feasible solution is either optimal or non-optimal. Note that the set of all optimal solutions $\mathcal{O}_{\mathcal{L},\succ}$ has the property that if $x \in \mathcal{O}_{\mathcal{L},\succ}$, then $x - \gamma \in \mathcal{O}_{\mathcal{L},\succ}$ for all $\gamma \in \mathbb{N}^n$ where $\gamma \leq x$.

There is a lot more that we know about the structure of a reduced Gröbner basis than its uniqueness. In Lemma 4.2.8 below, we give a complete combinatorial description of the vectors in a reduced Gröbner basis.

Lemma 4.2.8. *Let $G \subseteq \mathcal{L}_\succ$ be the unique reduced $\succ$-Gröbner basis of $\mathcal{L}$. A vector $u \in \mathcal{L}_\succ$ is in G if and only if u^- is the optimal solution of $IP_{\mathcal{L},\succ}(u^+)$ and $(u^+ - \mathbf{e}^i)$ is the optimal solution of $IP_{\mathcal{L},\succ}(u^+ - \mathbf{e}^i)$ for all $i \in \text{supp}(u^+)$.*

Proof. Let $u \in G$. Then, from above, u^- is the optimal solution of $IP_{\mathcal{L},\succ}(u^+)$ since $G \not\leq u^-$ from the definition of a reduced Gröbner basis. We know that u^+ is a non-optimal solution. Assume that $(u^+ - \mathbf{e}^i)$ is a non-optimal solution of $IP_{\mathcal{L},\succ}(u^+ - \mathbf{e}^i)$ for some $i \in \text{supp}(u^+)$. Then, there exists a vector $v \in G$ such that $v^+ \leq (u^+ - \mathbf{e}^i)$ implying that $v^+ \leq u^+$ and $v^+ \neq u^+$, which contradicts the minimality of G.

Conversely, let $u \in \mathcal{L}_\succ$ such that $(u^+ - \mathbf{e}^i)$ is the optimal solution of $IP_{\mathcal{L},\succ}(u^+ - \mathbf{e}^i)$ for all $i \in \text{supp}(u^+)$. Since u^+ is non-optimal, there exists a vector $v \in G$ such that $v^+ \leq u^+$, and since $(u^+ - \mathbf{e}^i)$ is the optimal solution of $IP_{\mathcal{L},\succ}(u^+ - \mathbf{e}^i)$ for all $i \in \text{supp}(u^+)$, we must also have $v^+ \not\leq (u^+ - \mathbf{e}^i)$, which implies that $v^+ = u^+$. Note that v^- is the optimal solution of $IP_{\mathcal{L},\succ}(v^-)$, which follows from the first part of this proof. Furthermore, this implies that $v^- = u^-$ since v^- and u^- are optimal solutions of $IP_{\mathcal{L},\succ}(u^+) = IP_{\mathcal{L},\succ}(u^-)$. □

An interesting consequence of Lemma 4.2.8 is that if G is a reduced $\succ$-Gröbner basis of $\mathcal{L}$ and $u \in G$, then $u^+ = x^* + \mathbf{e}^i$ for some optimal solution $x^* \in \mathcal{O}_{\mathcal{L},\succ}$ and some $i \in \{1, ..., n\}$. In the next example, we will use this to give upper bounds on the size of some special Gröbner bases.

An important special case is when $\mathcal{L} \subseteq \mathbb{Z}^n$ is an n-dimensional lattice (i.e. there are n vectors in a basis of $\mathcal{L}$). Let $B \in \mathbb{Z}^{n \times n}$ be an upper triangle matrix with positive diagonal entries and non-positive entries elsewhere such that the rows of B form a basis of $\mathcal{L}$. We can always construct such a matrix B from any basis of $\mathcal{L}$ using the HNF algorithm (Section 2.4); that is, B is in UHNF form. The set B is a $\succ_{lex}$-Gröbner basis of $\mathcal{L}$ where $\succ_{lex}$ is the lexicographic term order. We can show this using Lemma 4.2.3. First, observe that for every vector $u = B_i$ for $i = 1, ..., n$, we have $u^+ = \lambda \mathbf{e}^i$ for some $\lambda \in \mathbb{N}$. Let $u \in \mathcal{L}_{\succ_{lex}}$; we now show $B^+ \leq u^+$ and the result follows by Lemma 4.2.3. Let i be the first non-zero component in u. Since $u^+ \succ_{lex} u^-$ and $\succ_{lex}$ is the lexicographic term order, $u_i > 0$. Let v be the ith row of B, so $v_i > 0$. Now, since B is a basis of $\mathcal{L}$ and it is an upper triangle matrix, v_i divides u_i. Therefore, $v^+ \leq u^+$ because, by construction, $v_j \leq 0$ for all $j \neq i$. Observe that, also by construction, B is a reduced $\succ_{lex}$-Gröbner basis of $\mathcal{L}$.

Since we now have a $\succ_{lex}$-Gröbner basis of $\mathcal{L}$, we can describe the set of all non-optimal solutions $\mathcal{N}_{\mathcal{L},\succ_{lex}}$ as follows:

$$\begin{aligned}\mathcal{N}_{\mathcal{L},\succ} =& \{u^+ + \gamma : u \in B, \gamma \in \mathbb{N}^n\} \\ =& \{x \in \mathbb{N}^n : x_i \geq B_{ii} \text{ for some } i \in \{1, ..., n\}\}.\end{aligned}$$

The set of all optimal solutions $\mathcal{O}_{\mathcal{L},\succ_{lex}}$ is actually finite in this case, which is not true for general lattices. Moreover, this implies that the set of all non-empty fibers is also finite because there is a one-to-one correspondence between optimal solutions and non-empty fibers. Because of the special structure of B, we can write the set of all optimal solutions $\mathcal{O}_{\mathcal{L},\succ_{lex}} = \mathbb{N}^n \setminus \mathcal{N}_{\mathcal{L},\succ}$ in the following simple form:

$$\mathcal{O}_{\mathcal{L},\succ_{lex}} = \{x \in \mathbb{N}^n : x_i < B_{ii} \; \forall i = 1, ..., n\}.$$

It follows that the number of optimal solutions (and the number of non-empty fibers) is $|\mathcal{O}_{\mathcal{L},\succ_{lex}}| = \det(B)$ since the ith component of an optimal solution can take any value in the set $\{0, 1, ..., B_{ii} - 1\}$.

This result on the number of optimal solutions combined with Lemma 4.2.8 gives an upper bound on the size of a reduced $\succ$-Gröbner basis of $\mathcal{L}$ for any term order $\succ$. Let $\mathcal{O}_{\mathcal{L},\succ}$ be the set of all optimal solutions, and let G be a reduced $\succ$-Gröbner basis of $\mathcal{L}$. Then, from above, we know that $|\mathcal{O}_{\mathcal{L},\succ}| = \det(B)$ since the number of possible optimal solutions is equal to the number of non-empty fibers, which is $\det(B)$ as we saw above. Furthermore, from Lemma 4.2.8, we know that

$$G^+ \subseteq \{x^* + \mathbf{e}^i : x^* \in \mathcal{O}_{\mathcal{L},\succ}, i \in \{1, ..., n\}\}.$$

Therefore, $|G| = |G^+| \leq n \det(B)$ because there are $\det(B)$ optimal solutions and we can potentially add n different unit vectors $\mathbf{e}^i$ to each optimal solution to arrive at the set $|G^+|$. This bound was also obtained by Faugère et al. in [34], and Sturmfels et al. prove the tighter bound in [81] that $|G| \leq (n-1)\det(B) + 1$ and they claim that this can be strengthened to $|G| \leq (n-2)\det(B) + n + 1$. Sturmfels et al. also give a tighter bound in [81] and Clements in [22] gives a sharp bound, but we will not investigate this question further.

Until now, we have given a combinatorial description of a reduced Gröbner basis, but there it is also possible to describe vectors in a reduced Gröbner basis using a geometric viewpoint. Sturmfels et al. in [81] show that there are only two types of vectors in a reduced Gröbner basis. We present the idea without proof (see [81]) because we do not use this result nor its proof in the rest of the thesis. Let u be a vector in a reduced $\succ$-Gröbner bases of $\mathcal{L}$. Then,

(i). $u^- = \mathbf{0}$ and u^+ is an extreme point of the polyhedron $\text{conv}(\mathcal{F}_{\mathcal{L}}(\mathbf{0}) \setminus \{\mathbf{0}\})$, or

(ii). $u^- \neq \mathbf{0}$ and u^+ and u^- are adjacent extreme points of $\text{conv}(\mathcal{F}_{\mathcal{L}}(u^-))$ where by adjacent we mean that u is an edge of the polyhedron from u^+ and u^-.

Note that $\text{conv}(\mathcal{F}_{\mathcal{L}}(\mathbf{0}) \setminus \{\mathbf{0}\})$ is the set of all possible convex combinations of points in $\mathcal{F}_{\mathcal{L}}(\mathbf{0}) \setminus \{\mathbf{0}\}$, which is a polyhedron, and similarly, $\text{conv}(\mathcal{F}_{\mathcal{L}}(u^-))$ is the set of all possible convex combinations of points in $\mathcal{F}_{\mathcal{L}}(u^-)$, which again is a polyhedron.

4.3 Truncated Gröbner bases of lattices

Analogously to truncated Markov bases, we define truncated Gröbner bases. Again, truncated Gröbner bases of lattices are analogous to the truncated Gröbner bases that we saw in the introduction. Most results in this section are known, but we do present a new result giving a pseudo-polynomial upper bound on the size of a truncated Gröbner basis for equality knapsacks. We closely follow the approach of Thomas and Weismantel in [85].

Definition 4.3.1. *Given a lattice $\mathcal{L}$, $\sigma \subseteq \{1, ..., n\}$, $\nu \in \mathbb{Z}^n$, and a term order $\succ$, we call a set $G \subseteq \mathcal{L}$ a* **ν-truncated $\succ$-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$** *if G is a $\succ$-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu')$ for every $\nu' \in \mathcal{B}^\sigma_{\mathcal{L}}(\nu)$. If $\sigma = \emptyset$, then we call G a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$.*

As in the Markov basis case, a ν-truncated $\succ$-Gröbner basis is by definition a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$, but a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ is not necessarily a ν-truncated $\succ$-Gröbner basis and furthermore, a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ is not necessarily a $\succ$-Gröbner basis of $\mathcal{L}$. Also again, we can restrict our attention to truncated Gröbner bases of lattices (i.e. $\sigma = \emptyset$). Recall that $\nu' \in \mathcal{B}^{\sigma}_{\mathcal{L}}(\nu)$ if and only if $\nu'_{\bar{\sigma}} \in \mathcal{B}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$. Then, from Lemma 3.1.5, assuming that $\ker(\pi_{\sigma}) \cap \mathcal{L} = \{\mathbf{0}\}$, $G \subseteq \mathcal{L}$ is a ν-truncated $\succ$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ if and only if $G_{\bar{\sigma}}$ is a $\nu_{\bar{\sigma}}$-truncated $\succ$-Gröbner basis of $\mathcal{L}^{\sigma}$.

Example 4.3.2. *Consider again the lattice program $IP_{\mathcal{L},\succ_c}(\nu)$ from Example 2.9.4 and Example 4.1.2 above. Recall that the set*

$$G'''' = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (1, 3, 1, \text{-}2), (5, \text{-}1, \text{-}3, 0), (1, \text{-}5, \text{-}3, 3), (0, \text{-}8, \text{-}4, 5)\}$$

is a $\succ_c$-Gröbner basis of $\mathcal{L}$. Recall that there are 7 feasible points of $IP_{\mathcal{L},\succ_c}(\nu)$:

$$\{(2, 4, 6, 0), (1, 1, 5, 2), (4, 2, 4, 1), (7, 3, 3, 0), (6, 0, 2, 2), (9, 1, 1, 1), (12, 2, 0, 0)\}.$$

The vector $(2, \text{-}2, \text{-}2, 1) \in G''''$ fits within the feasible set since $(4, 2, 4, 1) - (2, 4, 6, 0) = (2, \text{-}2, \text{-}2, 1)$. All the other vectors of G'''' fit within the feasible set except for the two vectors $(1, \text{-}5, \text{-}3, 3)$ and $(0, \text{-}8, \text{-}4, 5)$, which are too long to fit within the feasible set $\mathcal{F}_{\mathcal{L}}(\nu)$. Thus, the set

$$T = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (1, 3, 1, \text{-}2), (5, \text{-}1, \text{-}3, 0)\}$$

is a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$, but it is not a $\succ_c$-Gröbner basis of $\mathcal{L}$ as we have seen in Example 4.2.2.

Recall from the previous section, that a Gröbner basis of a lattice has some very special properties. Fortunately, these properties transfer nicely to the truncated case. We will define minimal truncated Gröbner bases and show that their size is invariant, and we will also define reduced truncated Gröbner bases and show that they are unique. Moreover, we will show that, after removing all the vectors that are truncated from the unique reduced Gröbner basis of a lattice (non-truncated), we obtain the unique reduced truncated Gröbner basis of a lattice.

Firstly, for every lattice $\mathcal{L} \subseteq \mathbb{Z}^n$, term order $\succ$, and vector $\nu \in \mathbb{Z}^n$, there must exist a finite ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ since a $\succ$-Gröbner basis of $\mathcal{L}$ is also, by definition, a ν-truncated Gröbner basis of $\mathcal{L}$ and, from Lemma 4.2.4, there always exists a finite $\succ$-Gröbner basis of $\mathcal{L}$. Also, there is an analogous result to Lemma 4.2.3 for the truncated case, which we present below in Lemma 4.3.3. The proof of this lemma follows the proof of Lemma 4.2.3, but we have reproduced it for the truncated case for the sake of clarity. First, we need a truncated version of the set $\mathcal{L}_{\succ}$. Given a lattice $\mathcal{L}$ and a vector $\nu \in \mathbb{Z}^n$, we define $\mathcal{L}_{\succ}(\nu) := \{u \in \mathcal{L} : u^+ \succ u^-, u^+ \in \mathcal{B}_{\mathcal{L}}(\nu)\}$.

Lemma 4.3.3. *Given a lattice $\mathcal{L}$, a term order $\succ$, and a vector $\nu \in \mathbb{Z}^n$, a set $G \subseteq \mathcal{L}_{\succ}$ is a ν-truncated Gröbner basis of $\mathcal{L}$ if and only if $G^+ \leq \mathcal{L}^+_{\succ}(\nu)$.*

Proof. This proof follows the proof of Lemma 4.2.3. Assume G is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$. Let $v \in \mathcal{L}^+_{\succ}(\nu)$, so $v^+ \succ v^-$ and $v^+, v^- \in \mathcal{F}_{\mathcal{L}}(\nu')$ for some

$\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$ (for example, $\nu' = v^+$ or $\nu' = v^-$). Hence, v^+ is not an optimal solution of $IP_{\mathcal{L},\succ}(\nu')$, and therefore, there exists a vector $u \in G$ such that $u^+ \leq v^+$.

Conversely, assume that $G^+ \leq \mathcal{L}^+_\succ(\nu)$. We will show that G is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ by Lemma 4.1.4. Let $x \in \mathcal{F}_{\mathcal{L}}(\nu')$ for some $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$, and let x^* be the optimal solution of $IP_{\mathcal{L},\succ}(\nu')$. If $x = x^*$, then there is nothing to show, so assume $x \neq x^*$. Now, $x - x^* \in \mathcal{L}_\succ$, and we need to show that $x - x^* \in \mathcal{L}_\succ(\nu)$; that is, we need to show that $(x - x^*)^+ \in \mathcal{B}_{\mathcal{L}}(\nu)$. Let $\nu'' = (x - x^*)^+$. Then, $\nu'' \in \mathcal{B}_{\mathcal{L}}(\nu')$ since $x, x^* \in \mathcal{B}_{\mathcal{L}}(\nu')$, so $\mathcal{F}_{\mathcal{L}}(\nu' - \nu'') \neq \emptyset$. Lastly, $\mathcal{F}_{\mathcal{L}}(\nu - \nu') \neq \emptyset$ and $\mathcal{F}_{\mathcal{L}}(\nu' - \nu'') \neq \emptyset$ together imply that $\mathcal{F}_{\mathcal{L}}(\nu - \nu'') \neq \emptyset$, and thus, $\nu'' \in \mathcal{B}_{\mathcal{L}}(\nu)$. Therefore, since $G^+ \leq \mathcal{L}^+_\succ(\nu)$, there exists a vector $u \in G$ such that $u^+ \leq (x - x^*)^+$, which implies that $u^+ \leq x$ as required. □

As in the non-truncated case, if G is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ such that there exist vectors $u, v \in G$ where $u \neq v$ and $u^+ \leq v^+$, then the set $G' = G \setminus \{v\}$ is still a truncated Gröbner basis since we must still have $G'^+ \leq \mathcal{L}^+_\succ(\nu)$. Also, if $v \in G$ such that $v^+ \notin \mathcal{B}_{\mathcal{L}}(\nu)$, then again $G' = G \setminus \{v\}$ is still a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$. If we remove all such vectors v, we then arrive at a **minimal** ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$.

If G is a minimal $\succ$-Gröbner basis of $\mathcal{L}$ (not truncated) and we remove all vectors $u \in G$ such that $u^+ \notin \mathcal{B}_{\mathcal{L}}(\nu)$, then we are left with a a set G' ($G' = G \cap \mathcal{L}_\succ(\nu)$) that is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$, and moreover, the set G' must also be a minimal ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$. If G' were not minimal, then there would exist vectors $u, v \in G'$ where $u \neq v$ and $u^+ \leq v^+$, but these vectors would also be in G implying that G is also not minimal.

Lemma 4.3.4. *Given a lattice $\mathcal{L}$, a term order $\succ$, and a vector $\nu \in \mathbb{Z}^n$, if $G \subseteq \mathcal{L}_\succ$ is a minimal $\succ$-Gröbner basis of $\mathcal{L}$, then $G \cap \mathcal{L}_\succ(\nu)$ is a minimal ν-truncated $\succ$-Gröbner bases of $\mathcal{L}$.*

Minimal truncated Gröbner bases have the same size and the same positive components. The argument for this is essentially the same as in the non-truncated case (see Lemma 4.2.5). This result also follows from the above observation that truncating a minimal Gröbner basis of a lattice gives a minimal truncated Gröbner basis of a lattice.

Lemma 4.3.5. *Given a lattice $\mathcal{L}$, a term order $\succ$, and a vector $\nu \in \mathbb{Z}^n$, if $G, H \subseteq \mathcal{L}_\succ(\nu)$ are minimal ν-truncated $\succ$-Gröbner bases of $\mathcal{L}$, then $|G| = |H|$, and moreover, $G^+ = H^+$.*

We also define *reduced* truncated Gröbner bases in exactly the same way as in the non-truncated case. That is, a set G that is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ is **reduced** if G is minimal and for every $u \in G$, we have $G^+ \not\leq u^-$. Recall that $G^+ \not\leq u^-$ means that there does not exist a vector $v \in G$ such that $v^+ \leq u^-$.

Let G be a reduced ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$. Then, by definition G is minimal, which means that $u^+, u^- \in \mathcal{B}_{\mathcal{L}}(\nu)$ for all $u \in G$ by Lemma 3.3.2 since otherwise the vector u is too long. Also, since G is reduced, we have $G^+ \not\leq u^-$, which implies that u^- is the unique optimal solution in $\mathcal{F}_{\mathcal{L}}(u^-)$ since G is a $\succ$-Gröbner

basis of $IP_{\mathcal{L},\succ}(u^-)$. Thus, as in the non-truncated case, a reduced truncated Gröbner basis must be unique since G^+ is unique by Lemma 4.3.5 and G^- is also unique.

Lemma 4.3.6. *Given a lattice $\mathcal{L}$, a term order $\succ$, and a vector $\nu \in \mathbb{Z}^n$, there is a unique reduced ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$.*

Fortunately, the unique reduced Gröbner basis of a lattice after removing truncated vectors gives us exactly the unique truncated reduced Gröbner basis of the lattice. Let G be the reduced $\succ$-Gröbner basis of $\mathcal{L}$ (non-truncated), and let G' be the ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ obtained by truncating G ($G' = G \cap \mathcal{L}_\succ(\nu)$). From our discussion above, G' is a minimal ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$, and moreover, it also must be a reduced ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$.

Lemma 4.3.7. *Given a lattice $\mathcal{L}$, a term order $\succ$, and a vector $\nu \in \mathbb{Z}^n$, if $G \subseteq \mathcal{L}_\succ$ is the unique reduced $\succ$-Gröbner bases of $\mathcal{L}$, then $G \cap \mathcal{L}_\succ(\nu)$ is the unique reduced ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$.*

As in the non-truncated case, for the same reasons, given a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$, we can give an explicit description of the collective set $\mathcal{N}_{\mathcal{L},\succ}(\nu)$ of all non-optimal solutions of $IP_{\mathcal{L},\succ}(\nu')$ for every $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$ as follows:

$$\mathcal{N}_{\mathcal{L},\succ}(\nu) = \{(u^+ + \gamma) \in \mathcal{B}_\mathcal{L}(\nu) : u \in G, \gamma \in \mathbb{N}^n\}.$$

This also gives us a description of the collective set $\mathcal{O}_{\mathcal{L},\succ}(\nu)$ of optimal solutions of $IP_{\mathcal{L},\succ}(\nu')$ for every $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$ as $\mathcal{O}_{\mathcal{L},\succ}(\nu) = \{x \in \mathbb{N}^n : x \notin \mathcal{N}_{\mathcal{L},\succ}(\nu), x \in \mathcal{B}_\mathcal{L}(\nu)\}$. Note that the set of all optimal solutions $\mathcal{O}_{\mathcal{L},\succ}(\nu)$ also has the property that if $x \in \mathcal{O}_{\mathcal{L},\succ}(\nu)$, then $x - \gamma \in \mathcal{O}_{\mathcal{L},\succ}(\nu)$ for all $\gamma \in \mathbb{N}^n$ where $\gamma \leq x$ since if $x \in \mathcal{B}_\mathcal{L}(\nu)$, then $x - \gamma \in \mathcal{B}_\mathcal{L}(\nu)$.

In Lemma 4.3.8 below, we give a complete combinatorial description of the vectors in a reduced truncated Gröbner basis analogously to Lemma 4.2.8. The proof is essentially the same after observing that if $u^+ \in \mathcal{B}_\mathcal{L}(\nu)$, then $(u^+ - \mathbf{e}^i) \in \mathcal{B}_\mathcal{L}(\nu)$ for all $i \in \text{supp}(u^+)$.

Lemma 4.3.8. *Let $G \subseteq \mathcal{L}_\succ(\nu)$ be the unique ν-truncated reduced $\succ$-Gröbner basis of $\mathcal{L}$. A vector $u \in \mathcal{L}_\succ(\nu)$ is in G if and only if u^- is the optimal solution of $IP_{\mathcal{L},\succ}(u^+)$ and $(u^+ - \mathbf{e}^i)$ is the optimal solution of $IP_{\mathcal{L},\succ}(u^+ - \mathbf{e}^i)$ for all $i \in \text{supp}(u^+)$.*

As in the non-truncated case, an interesting consequence of Lemma 4.3.8 above is that if G is a reduced ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ and $u \in G$, then $u^+ = x^* + \mathbf{e}^i$ for some optimal solution $x^* \in \mathcal{O}_{\mathcal{L},\succ}(\nu)$ and some $i \in \{1, ..., n\}$.

In the next example, we will use this to give upper bounds on the size of truncated Gröbner bases for equality knapsack problems. This has not been done before to the best of our knowledge.

In the special case of equality knapsack problems, we can give a pseudo-polynomial upper bound on the size of a minimal truncated Gröbner basis. An equality knapsack problem is an integer program with one equality constraint and non-negativity constraints on the variables:

$$KP_{a,c}(b) = \max\{cx : ax = b, x \geq \mathbf{0}, x \in \mathbb{Z}^n\}$$

where $c \in \mathbb{N}^n$, $a \in \mathbb{N}^n$, and $b \in \mathbb{N}$. Let $\mathcal{L} = \{x \in \mathbb{Z}^n : ax = 0\}$, and let $\nu \in \mathbb{Z}^n$ such that $a\nu = b$. Then, $\{x \in \mathbb{Z}^n : ax = b, x \geq \mathbf{0}\} = \mathcal{F}_{\mathcal{L}}(\nu)$. We will show that a minimal ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$ has at most nb vectors. We assume that $KP_{a,c}(b)$ is feasible otherwise an empty set is a ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$. Let $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$; then, $\mathcal{F}_{\mathcal{L}}(\nu') \neq \emptyset$ and $\mathcal{F}_{\mathcal{L}}(\nu - \nu') \neq \emptyset$. Let $y \in \mathcal{F}_{\mathcal{L}}(\nu')$ and let $z \in \mathcal{F}_{\mathcal{L}}(\nu - \nu')$, so $y + z \in \mathcal{F}_{\mathcal{L}}(\nu)$. Now, $ay \geq 0$, $az \geq 0$, and $a(y + z) = b$, which implies that $\mathbf{0} \leq ay \leq b$. This implies that there are only at most $b + 1$ many distinct fibers in $\mathcal{B}_{\mathcal{L}}(\nu)$ because each fiber corresponds to a b' which is between $\mathbf{0}$ and b inclusive. Thus, $\mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ contains all the optimal solutions of all the knapsack problems $KP_{a,c}(b')$ for every $b' \in \mathbb{N}^n$ where $b' \leq b$ and $KP_{a,c}(b')$ is feasible. This means that $|\mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)| \leq b + 1$ since there is only one optimal solution per distinct fiber in $\mathcal{B}_{\mathcal{L}}(\nu)$.

Recall from above that if G is a reduced ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ and $u \in G$, then $u^+ = x^* + \mathbf{e}^i$ for some optimal solution $x^* \in \mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ and some $i \in \{1, ..., n\}$. Therefore, there are at most $n(b + 1)$ vectors in a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$. This upper bound can be improved to nb vectors since if x^* is the optimal solution of ν (i.e. $ax^* = b$), then $x^* + \mathbf{e}^i \notin \mathcal{B}_{\mathcal{L}}(\nu)$ for every $i \in \{1, ..., n\}$, so there are at most b optimal solutions that we must consider. We then arrive at the following lemma.

Lemma 4.3.9. *Let $a \in \mathbb{N}^n$, $c \in \mathbb{N}^n$, $\mathcal{L} = \{x \in \mathbb{Z}^n : ax = 0\}$, and $\nu \in \mathbb{Z}^n$. A minimal ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$ has at most $na\nu$ vectors.*

We could also improve this bound to nk where k is the actual number of feasible distinct fibers in $\mathcal{B}_{\mathcal{L}}(\nu)$ excluding ν itself.

It is possible to extend this result to integer programs in the following form: $\max\{cx : Ax = b, x \geq \mathbf{0}, x \in \mathbb{Z}^n\}$ where $c \in \mathbb{N}^n$, $A \in \mathbb{N}^{n \times m}$, and $b \in \mathbb{N}^m$. For analogous reasons as the single dimensional knapsack above, an upper bound on the size of a truncated Gröbner basis in this case is $n[\prod_{i=1}^{n}(b_i + 1)]$.

Chapter 5

Computing Gröbner bases

In this chapter, we describe the completion procedure or Buchberger algorithm (see [15, 14, 28]) for computing a Gröbner basis of a lattice or a truncated Gröbner basis of a lattice starting from a Markov basis of a lattice or truncated Markov basis of a lattice respectively. This algorithm has been translated from algebraic geometry into a combinatorial context following the lead of Thomas in [86].[1] Although the main building blocks of the completion procedure as presented here are well known, we do offer some novel optimisations of the algorithm, which we will identify where appropriate. For example, our approach to truncation is novel in some respects.

In this chapter, we only discuss $\succ$-Gröbner bases of $\mathcal{L}$ and not the more general form of $\succ$-Gröbner bases of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. As we saw in Section 4.2, we can construct a $\succ$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ from a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}^{\sigma}}(\cdot)$. As a result of this, all of the results in this chapter can equally be applied to the more general case. The same applies when using truncation (see Section 4.3).

5.1 Completion procedure

In this section, we present the basic completion procedure for computing a $\succ$-Gröbner basis of $\mathcal{L}$ from a Markov basis of $\mathcal{L}$. The completion procedure is the standard procedure for computing Gröbner bases.

The definition of a Gröbner basis explicitly refers to the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$ as does the Lemma 4.1.6. We now describe Gröbner bases in terms of reduction paths, so that we avoid explicitly mentioning the optimal solution of a fiber. A path $(x^0, \ldots, x^k)$ in $\mathcal{G}_{\mathcal{L}}(\nu, G)$ is a **$\succ$-reduction path** if, for all $i \in \{1, \ldots, k-1\}$, we have either $x^0 \succ x^i$ or $x^k \succ x^i$. For example, see Figure 5.1.

Lemma 5.1.1. *Given $\nu \in \mathbb{Z}^n$, a set $G \subseteq \mathcal{L}_{\succ}$ is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ if and only if, for each pair $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$, there exists a $\succ$-reduction path in $\mathcal{G}_{\mathcal{L}}(\nu, G)$ between x and y.*

[1] In computational algebraic geometry, the Buchberger algorithm computes a Gröbner bases of an ideal of a polynomial ring with respect to some term order.

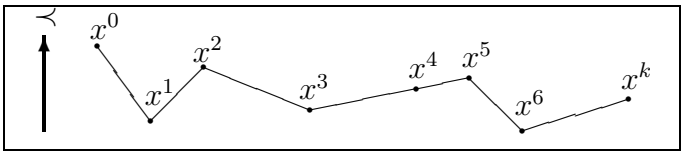

Figure 5.1: Reduction path between x and y.

Proof. Let $G \subseteq \mathcal{L}_\succ$ be a $\succ$-Gröbner basis of $\mathcal{F}_\mathcal{L}(\nu)$, and let $x, y \in \mathcal{F}_\mathcal{L}(\nu)$. From Lemma 4.1.6, $\mathcal{G}_\mathcal{L}(\nu, G)$ contains $\succ$-decreasing paths from x and from y to the unique optimal solution of $IP_{\mathcal{L},\succ}(\nu)$, then joining the two paths (and removing cycles if necessary) forms a $\succ$-reduction path between x and y.

For the other direction, we assume that there is a $\succ$-reduction path between each pair $x, y \in \mathcal{F}_\mathcal{L}(\nu)$. Let x^* be the the unique optimal solution of $IP_{\mathcal{L},\succ}(\nu)$; thus, every $x \in \mathcal{F}_\mathcal{L}(\nu)$ is connected to x^* by a $\succ$-reduction path in $\mathcal{F}_\mathcal{L}(\nu)$. By the definition of a $\succ$-reduction path, if $x \neq x^*$, then the first node $x^1 \neq x$ in this path must satisfy $x \succ x^1$ and $x - x^1 \in G$, and therefore, G is a Gröbner basis by the definition of a Gröbner basis. □

Checking for a given $G \subseteq \mathcal{L}_\succ$ whether there exists a $\succ$-reduction path in $\mathcal{G}_\mathcal{L}(\nu, G)$ for every $\nu \in \mathbb{Z}^n$ and for each pair $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ involves many situations that need to be checked. In fact, far fewer checks are needed: we only need to check for a $\succ$-reduction path from x to y if there exists a **$\succ$-critical path** from x to y.

Definition 5.1.2. *Given $G \subseteq \mathcal{L}_\succ$ and $\nu \in \mathbb{Z}^n$, a path (x, z, y) in $\mathcal{G}_\mathcal{L}(\nu, G)$ is a* **$\succ$-critical path** *if $z \succ x$ and $z \succ y$.*

If (x, z, y) is a $\succ$-critical path in $\mathcal{G}_\mathcal{L}(\nu, G)$, then $x + u = z = y + v$ for some pair $u, v \in G$, in which case, we call (x, z, y) a $\succ$-critical path for (u, v) (see Figure 5.2).

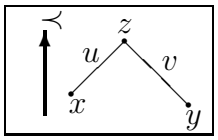

Figure 5.2: A critical path for (u, v) between x, z, and y.

Lemma 5.1.3. *Given $\nu \in \mathbb{Z}^n$, let $G \subseteq \mathcal{L}_\succ$ where G is a Markov basis of $\mathcal{F}_\mathcal{L}(\nu)$. The set G is a $\succ$-Gröbner basis of $\mathcal{F}_\mathcal{L}(\nu)$ if and only if, for every $\succ$-critical path (x, z, y) in $\mathcal{G}_\mathcal{L}(\nu, G)$, there exists a $\succ$-reduction path in $\mathcal{G}_\mathcal{L}(\nu, G)$ between x and y .*

Proof. Firstly, assume that G is a Gröbner basis of $\mathcal{F}_\mathcal{L}(\nu)$. Then, from Lemma 5.1.1, there exists a $\succ$-reduction path in $\mathcal{G}_\mathcal{L}(\nu, G)$ between x and y for every $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ including those x and y for which there exists a $\succ$-critical path (x, z, y) in $\mathcal{G}_\mathcal{L}(\nu, G)$.

Secondly, assume that there exists a $\succ$-reduction path between x and y for every $\succ$-critical path (x, z, y) in $\mathcal{G}_\mathcal{L}(\nu, G)$. We now show that G is a Gröbner basis by showing that there exists a $\succ$-reduction path from x to y for every $x, y \in \mathcal{F}_\mathcal{L}(\nu)$. Assume on

the contrary that there exists $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ such that there is no $\succ$-reduction path from x to y. Among all possible paths (not reduction paths) from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, G)$, $(x = x^0, \ldots, x^k = y)$, choose one such that $\max_{\succ}\{x^0, \ldots, x^k\}$ is minimal. At least one such path exists since G is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$, and a minimal such path exists since $\succ$ is a term order. Let $j \in \{0, \ldots, k\}$ where x^j attains this maximum. By assumption $(x^0, \ldots, x^k)$ is not a $\succ$-reduction path, and thus, $x^j \succ x^0$ and $x^j \succ x^k$, and since x^j is maximal, we have $x^j \succ x^{j-1}$ and $x^j \succ x^{j+1}$. Let $u = x^j - x^{j-1}$ and $v = x^j - x^{j+1}$; then, $u, v \in \mathcal{L}_{\succ}$, and also, $u, v \in G$, and thus, (x^{j-1}, x^j, x^{j+1}) forms a $\succ$-critical path for (u, v). Consequently, we can replace the path (x^{j-1}, x^j, x^{j+1}) with the $\succ$-reduction path $(x^{j-1} = \tilde{x}^0, \ldots, \tilde{x}^s = x^{j+1})$ in the path $(x^0, \ldots, x^k)$ to obtain a new path between x and y with the property that the $\succ$-maximum of the intermediate nodes is strictly less than $x^j = \max_{\succ}\{x^0, \ldots, x^k\}$ (see Figure 5.3). This contradiction proves our claim. □

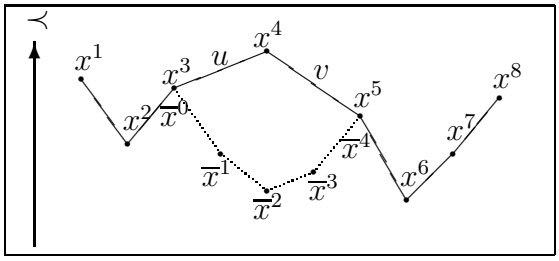

Figure 5.3: Replacing a critical path by a reduction path

We can extend Lemma 5.1.3 to Gröbner bases of lattices. It is a straight-forward consequence of Lemma 5.1.3, but nevertheless, it is worthwhile stating explicitly.

Corollary 5.1.4. *Given a lattice $\mathcal{L}$ and a term order $\succ$, let $G \subseteq \mathcal{L}_{\succ}$ such that G is a Markov basis of $\mathcal{L}$. The set G is a $\succ$-Gröbner basis of $\mathcal{L}$ if and only if, for all $\nu \in \mathbb{Z}^n$, there exists a $\succ$-reduction path between x and y for every $\succ$-critical path (x, z, y) in $\mathcal{G}_{\mathcal{L}}(\nu, G)$.*

It is not necessary to check for a $\succ$-reduction path from x to y for every $\succ$-critical path (x, z, y) in $\mathcal{G}_{\mathcal{L}}(\nu, G)$ for all $\nu \in \mathbb{Z}^n$. If there exists another $\succ$-critical path (x', z', y') in $\mathcal{G}_{\mathcal{L}}(\nu', G)$ for some $\nu' \in \mathbb{Z}^n$ such that $(x, z, y) = (x' + \gamma, y' + \gamma, z' + \gamma)$ for some $\gamma \in \mathbb{N}^n$, then a $\succ$-reduction path from x' to y' in $\mathcal{G}_{\mathcal{L}}(\nu', G)$ translates by γ to a $\succ$-reduction path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, G)$. More formally, if $(x^0, x^1, ..., x^k)$ is a $\succ$-reduction path from x' to y' in $\mathcal{G}_{\mathcal{L}}(\nu', G)$, then $(x^0 + \gamma, x^1 + \gamma, ..., x^k + \gamma)$ is a $\succ$-reduction path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, G)$. So, we only need to check for a $\succ$-reduction path from x' to y' and not from x to y.

We call a $\succ$-critical path (x, z, y) **minimal** if there does not exist another $\succ$-critical path (x', z', y') such that $(x, z, y) = (x' + \gamma, y' + \gamma, z' + \gamma)$ for some $\gamma \in \mathbb{N}^n$ where $\gamma \neq \mathbf{0}$, or equivalently, $\min\{x_i, y_i, z_i\} = 0$ for all $i = 1, \ldots, n$. Consequently, if there exists a $\succ$-reduction path between x and y for all minimal $\succ$-critical paths (x, z, y) in $\mathcal{G}_{\mathcal{L}}(\nu, G)$ for all $\nu \in \mathbb{Z}^n$, then there exists a $\succ$-reduction path between x' and y' for all $\succ$-critical paths (x', z', y') in $\mathcal{G}_{\mathcal{L}}(\nu', G)$ for all $\nu' \in \mathbb{Z}^n$.

For each pair of vectors $u, v \in G$, there exists a unique minimal $\succ$-critical path $(x^{(u,v)}, z^{(u,v)}, y^{(u,v)})$ in $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$ determined by $z^{(u,v)} := u^+ \vee v^+$ (i.e. $z_i^{(u,v)} = \max\{0, u_i, v_i\}$ for all $i = 1, ..., n$), $x^{(u,v)} := z^{(u,v)} - u$, $y^{(u,v)} := z^{(u,v)} - v$, and $\nu^{(u,v)} := z^{(u,v)}$. So, any $\succ$-critical path for (u, v) is of the form $(x^{(u,v)} + \gamma, z^{(u,v)} + \gamma, y^{(u,v)} + \gamma)$ for some $\gamma \in \mathbb{N}^n$. Using minimal $\succ$-critical paths, we can rewrite Corollary 5.1.4, so that we only need to check for a *finite* number of $\succ$-reduction paths.

Lemma 5.1.5. *Given a lattice $\mathcal{L}$ and a term order $\succ$, let $G \subseteq \mathcal{L}_{\succ}$ such that G is a Markov basis of $\mathcal{L}$. The set $G \subseteq \mathcal{L}_{\succ}$ is a $\succ$-Gröbner basis of $\mathcal{L}$ if and only if, for all $u, v \in G$, there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$.*

We now turn Lemma 5.1.5 into an algorithmic tool to compute Gröbner bases. Algorithm 2 below, called a completion procedure ([15]), starts from a Markov basis of $\mathcal{L}$ and computes a $\succ$-Gröbner basis of $\mathcal{L}$.

Given a set $M \subseteq \mathcal{L}$, the completion procedure first sets $G := M$, and then directs all vectors in G according to $\succ$ such that $G \subseteq \mathcal{L}_{\succ}$. Note that G is a Markov basis of $\mathcal{L}$ since M is a Markov basis and we have only changed the direction of vectors. The completion procedure then determines whether the set G satisfies Lemma 5.1.5; in other words, it tries to find a reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ for every pair $u, v \in G$ where $\nu^{(u,v)} \in \mathcal{B}_{\mathcal{L}}(\nu)$. If G satisfies Lemma 5.1.5, then we are done. Otherwise, no $\succ$-reduction path was found for some pair $u, v \in G$, in which case, we add a vector to G so that a $\succ$-reduction path exists, and then again, test whether G satisfies Lemma 5.1.5 and so on.

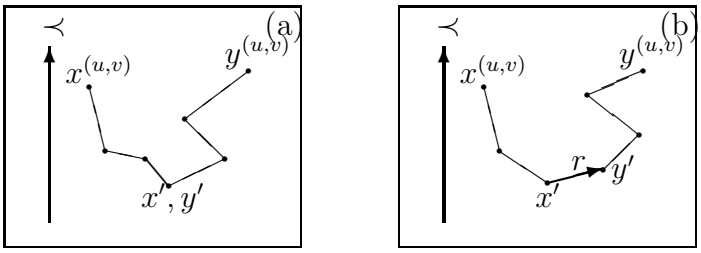

Figure 5.4: Checking for a reduction path from $x^{(u,v)}$ to $y^{(u,v)}$.

To check for a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$, we run the Normal Form algorithm (Algorithm 1) to construct a $\succ$-decreasing path in $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$ from $x^{(u,v)}$ to some $x' = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$ and from $y^{(u,v)}$ to some $y' = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. If $x' = y'$ (see Figure 5.4a), then we have found a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ by joining the path from x to x' to the path from y to y' (and removing cycles if necessary). Otherwise, we add the vector $r \in \mathcal{L}_{\succ}$ to G where $r := x' - y'$ if $x' \succ y'$, and $r := y' - x'$ otherwise, and then, there is now a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ in $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$, which is again formed by joining the path from x to x' to the path from y to y' where the vector r' is used to step between x' and y' (see Figure 5.4b).

We write $\mathcal{CP}_{\mathcal{L}}(\succ, M)$ for the output of the completion procedure. Also, as mentioned at the start of the section, we can compute a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\cdot)$ using the Completion Procedure, in which case, we write $\mathcal{CP}_{\mathcal{L}}^{\sigma}(\succ, M)$ for the output of the completion procedure.

Algorithm 2 Completion procedure

Input: a lattice $\mathcal{L}$, a term order $\succ$, and a Markov basis $M \subseteq \mathcal{L}$.
Output: a $\succ$-Gröbner basis $G \subseteq \mathcal{L}_\succ$ of $\mathcal{L}$.

```
G := {u : u+ ≻ u−, u ∈ M} ∪ {−u : u− ≻ u+, u ∈ M}
C := {(u, v) : u, v ∈ G}
while C ≠ ∅ do
  Select (u, v) ∈ C
  C := C \ {(u, v)}
  r := NF_L(x^(u,v), G) − NF_L(y^(u,v), G)
  if r ≠ 0 then
    if r− ≻ r+ then r := −r end if
    C := C ∪ {(w, r) : w ∈ G}
    G := G ∪ {r}
  end if
end while
return G.
```

Lemma 5.1.6. *Algorithm 2 terminates and satisfies its specifications.*

Proof. We first show that the algorithm terminates. Note that before we add r to G, since the paths from x to x' and from y to y' are maximal, there does not exist $u \in G$ such that $x' \geq u^+$ or $y' \geq u^+$. Therefore, there does not exist $u \in G$ such that $r^+ \geq u^+$. This condition is needed to ensure that the completion procedure terminates. Let $(r^1, r^2, \dots)$ be the sequence of vectors r that are added to the set G during the Algorithm 2. Since before we add r to G, there does not exist $u \in G$ such that $r^+ \geq u^+$, the sequence satisfies $(r^i)^+ \not\leq (r^j)^+$ whenever $i < j$. Thus, the sequence $(r^1, r^2, \dots)$ must be finite by the Gordan-Dickson Lemma 2.5.1, and therefore, Algorithm 2 must terminate.

When the algorithm terminates, the set G is a Markov basis of $\mathcal{L}$, and G must satisfy the property that for each $u, v \in G$ there exists a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$, and therefore, by Lemma 5.1.5, G is a Gröbner basis of $\mathcal{L}$. Note that when the algorithm terminates, the set G is a Markov basis of $\mathcal{L}$ since it is initially a Markov basis and we only add vectors to G during the algorithm. □

The worst-case complexity of the completion procedure for computing Gröbner bases of polynomials ideals is known to be doubly exponential in the number of variables (see for example [60, 65]), but it is not known what the worst-case complexity of the completion procedure is for the special case of computing Gröbner bases of lattices. However, there is a result by Sturmfels in [78] that gives a singly exponential (in the number of variables) upper bound on the l_1-norm[2] of the vectors in Gröbner bases of saturated lattices whereas the more general bound for polynomial ideals is doubly exponential (see for example [60, 65]). So, the worst-case complexity of the completion procedure for computing Gröbner bases of lattices may possibly be singly exponential and not doubly exponential.

[2]Given a vector $v \in \mathbb{Z}^n$, the l_1-norm of v is $\sum_{i=1}^n |v_i|$.

The completion procedure as written in Algorithm 2 is technically incomplete since it does not specify how to select a critical pair $(u, v) \in C$. However, the order in which we select critical pairs from C does not affect the correctness of the algorithm. In practice, a *good* approach is to treat the set of critical pairs C as a *first-in-first-out* queue, and thus, any critical pairs involving new vectors in G are always selected after critical pairs of old vectors in G. Specifically, we consider the set $G := \{v^1, v^2, ..., v^k\}$ as an ordered list of vectors, and we set $C := \{(v^i, v^j) : 1 \leq i < j \leq k\}$. Then, we choose the critical pair $(v^{i_1}, v^{j_1}) \in C$ before the critical pair $(v^{i_2}, v^{j_2}) \in C$ if $j_1 < j_2$, or $j_1 = j_2$ and $i_1 < i_2$, and when adding vectors to G, we always add a new vector to the end of the list of vectors in G. The initial order of the set G is not important for this approach; what matters is that new vectors are placed at the end of the list. Also, Gionvini et al. in [42] discuss other strategies for choosing the order in which to select critical pairs $(u, v) \in C$ in the completion procedure. Furthermore, for some lattices, there exists a critical pair selection strategy such that the completion procedure is guaranteed to compute a minimal Gröbner basis; this is known as the homogeneous completion procedure.

Example 5.1.7. *Consider again the lattice $\mathcal{L}$ as in Example 3.1.6. Again let $c := (2, 0, 1, 1)$. We now present an example computation of a $\succ_c$-Gröbner basis of a $\mathcal{L}$ using the completion procedure (Algorithm 2). In this case, we leave the tie-breaking term order $\succ$ unspecified since it is never needed during the algorithm: the cost c is sufficient to direct all the vectors encountered during the algorithm.*

The set $M :=${(3,1,-1,-1),(1,3,1,-2),(2,-2,-2,1),(5,-1,-3,0)}*. is a Markov basis of $\mathcal{L}$ as shown in Example 3.2.2. We will construct a Gröbner basis starting from this set.*

Firstly, we set $G := M$ and direct all vectors in G according to the term order $\succ_c$. Note that each of the vectors in M has a positive cost, so we do not have to change the direction of any vector in G. Then secondly, we construct the set of critical pairs $C := \{(u, v) : u, v \in G\}$. We will select critical pairs in the order given above where the vectors in the set G are ordered as written above for the set M.

Now, for each critical pair of vectors $(u, v) \in C$, we check whether $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. If this is true, then G is a $\succ_c$-Gröbner basis of $\mathcal{L}$. Otherwise if $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) \neq \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$ for some pair $(u, v) \in C$, then we add the vector $r = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) - \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$ or $-r$ to G whichever is correctly directed according to the cost. Then, we check again whether the larger set G is a Gröbner basis.

(i). Select $(u, v) =$((3,1,-1,-1),(1,3,1,-2))*. Then, we have $z^{(u,v)} = (3, 3, 1, 0)$, $x^{(u,v)} = (0, 2, 2, 1)$ and $y^{(u,v)} = (2, 0, 0, 2)$. Next, we compute the normal form of $x^{(u,v)}$ and the normal form of $y^{(u,v)}$. Since there is no vector $w \in G$ such that $w^+ \leq x^{(u,v)}$, we must have $x^{(u,v)} = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$. Since* (2,-2,-2,1)$^+ \leq y^{(u,v)}$*, we compute a new vector*

$$y' = (2, 0, 0, 2) - (2,\text{-}2,\text{-}2,1) = (0, 2, 2, 1).$$

Now since there is no vector $w \in G$ such that $w^+ \leq y'$, we must have $y' = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. Since $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$, we have found a path from $x^{(u,v)}$ to $y^{(u,v)}$, so there is nothing left to do for this critical pair of vectors.

(ii). Select (u,v) =((3,1,-1,-1),(2,-2,-2,1)). *Then, we have* $x^{(u,v)} = (0,0,1,2) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and*

$$y^{(u,v)} = (1,3,2,0) \overset{(1,3,1,-2)}{\longrightarrow} (0,0,1,2) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(iii). Select (u,v) =((1,3,1,-2),(2,-2,-2,1)). *Then, we have* $x^{(u,v)} = (1,0,0,3) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and* $y^{(u,v)} = (0,5,3,0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. *Therefore, we have* $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) \neq \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$*! Therefore, we have found a critical pair* (u,v) *such that there is no reduction path from* $x^{(u,v)}$ *to* $y^{(u,v)}$ *in* $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$, *and thus, from Lemma 5.1.1, we have proven that* G *is not a* $\prec_c$*-Gröbner basis of* $\mathcal{L}$. *Let* $r = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) - \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$ =(1,-5,-3,3). *We must now add* (v,r) *to* C *for all* $v \in G$ *and add* r *to* G. *So, now*

$$G := \{(3,1,-1,-1),(1,3,1,-2),(2,-2,-2,1),(5,-1,-3,0),(1,-5,-3,3)\}.$$

(iv). Select (u,v) =((3,1,-1,-1),(5,-1,-3,0)). *Then, we have* $y^{(u,v)} = (0,2,3,0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$, *and*

$$x^{(u,v)} = (2,0,1,1) \overset{(2,-2,-2,1)}{\longrightarrow} (0,2,3,0) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(v). Select (u,v) =((1,3,1,-2),(5,-1,-3,0)). *Then, we have* $y^{(u,v)} = (0,4,4,0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$, *and*

$$x^{(u,v)} = (4,0,0,2) \overset{(2,-2,-2,1)}{\longrightarrow} (2,2,2,1) \overset{(2,-2,-2,1)}{\longrightarrow} (0,4,4,0) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(vi). Select (u,v) =((2,-2,-2,1),(5,-1,-3,0)). *Then, we have* $y^{(u,v)} = (0,1,3,1) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$, *and*

$$x^{(u,v)} = (3,2,2,0) \overset{(3,1,-1,-1)}{\longrightarrow} (0,1,3,1) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(vii). Select (u,v) =((3,1,-1,-1),(1,-5,-3,3)). *Then, we have* $x^{(u,v)} = (0,0,1,4) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and*

$$y^{(u,v)} = (2,6,3,0) \overset{(1,3,1,-2)}{\longrightarrow} (1,3,2,2) \overset{(1,3,1,-2)}{\longrightarrow} (0,0,1,4) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(viii). *Select* (u,v) =((1,3,1,-2),(1,-5,-3,3)). *Then, we have* $x^{(u,v)} = (0,0,0,5) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and* $y^{(u,v)} = (0,8,4,0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. *Thus, we have found another critical pair* (u,v) *such that there is no reduction path from* $x^{(u,v)}$ *to* $y^{(u,v)}$ *in* $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$ *because* $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) \neq \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$, *and thus from Lemma 5.1.1, we have proven that* G *is still not a* $\prec_c$*-Gröbner basis of* $\mathcal{L}$. *Let* $r = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) - \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$ =(0,-8,-4,5). *We must now add* (v,r) *to* C *for all* $v \in G$ *and add* r *to* G. *So, now* $G :=$

$$\{(3,1,\text{-}1,\text{-}1),(1,3,1,\text{-}2),(2,\text{-}2,\text{-}2,1),(5,\text{-}1,\text{-}3,0),(1,\text{-}5,\text{-}3,3),(0,\text{-}8,\text{-}4,5)\}.$$

(ix). *Select* (u,v) =((2,-2,-2,1),(1,-5,-3,3)). *Then, we have* $x^{(u,v)} = (0,2,2,2) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and*

$$y^{(u,v)} = (1,5,3,0) \xrightarrow{(1,3,1,\text{-}2)} (0,2,2,2) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(x). *Select* (u,v) =((5,-1,-3,0),(1,-5,-3,3)). *Then, we have* $x^{(u,v)} = (0,1,3,3) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and*

$$\begin{aligned} y^{(u,v)} = (4,5,3,0) &\xrightarrow{(3,1,\text{-}1,\text{-}1)} (1,4,4,1) \\ &\xrightarrow{(1,3,1,\text{-}2)} (0,1,3,3) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G). \end{aligned}$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(xi). *Select* (u,v) =((3,1,-1,-1),(0,-8,-4,5)). *Then,*

$$x^{(u,v)} = (0,0,1,6) \xrightarrow{(0,\text{-}8,\text{-}4,5)} (0,8,5,1) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G).$$

$$\begin{aligned} y^{(u,v)} = (3,9,4,0) &\xrightarrow{(1,3,1,\text{-}2)} (2,6,3,2) \\ &\xrightarrow{(1,3,1,\text{-}2)} (1,3,2,4) \\ &\xrightarrow{(1,3,1,\text{-}2)} (0,0,1,6) \\ &\xrightarrow{(0,\text{-}8,\text{-}4,5)} (0,8,5,1) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G). \end{aligned}$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(xii). *Select* (u,v) =((1,3,1,-2),(0,-8,-4,5)). *Then,*

$$x^{(u,v)} = (0,0,0,7) \xrightarrow{(0,\text{-}8,\text{-}4,5)} (0,8,4,2) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G).$$

$$y^{(u,v)} = (1,11,5,0) \xrightarrow{(1,3,1,\text{-}2)} (0,8,4,2) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(xiii). Select (u, v) =((2,-2,-2,1),(0,-8,-4,5)). *Then, we have* $x^{(u,v)} = (0, 2, 2, 4) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and*

$$y^{(u,v)} = (2, 8, 4, 0) \overset{(1,3,1,-2)}{\longrightarrow} (1, 5, 3, 2) \overset{(1,3,1,-2)}{\longrightarrow} (0, 2, 2, 4) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(xiv). Select (u, v) =((5,-1,-3,0),(0,-8,-4,5)). *Then,*

$$x^{(u,v)} = (0, 1, 3, 5) \overset{(0,-8,-4,5)}{\longrightarrow} (0, 9, 7, 0) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G).$$

$$y^{(u,v)} = (5, 8, 4, 0) \overset{(5,-1,-3,0)}{\longrightarrow} (0, 9, 7, 0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(xv). Select (u, v) =((1,-5,-3,3),(0,-8,-4,5)). *Then, we have* $x^{(u,v)} = (0, 5, 3, 2) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and*

$$y^{(u,v)} = (1, 8, 4, 0) \overset{(1,3,1,-2)}{\longrightarrow} (0, 5, 3, 2) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G).$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

So now, for all $u, v \in G$, *there exists a* $\prec_c$*-reduction path from* $x^{(u.v)}$ *to* $y^{(u,v)}$, *so*

$$G := \{(3,1,-1,-1),(1,3,1,-2),(2,-2,-2,1),(5,-1,-3,0),(1,-5,-3,3),(0,-8,-4,5)\}.$$

is a $\prec_c$*-Gröbner basis of* $\mathcal{L}$.

5.2 Truncated completion procedure

In this section, we present existing algorithms for truncated Gröbner bases including a new approach to truncation. We follow the approach of Weismantel and Thomas in [85] and extend the approaches they give for truncation. The algorithm for computing truncated Gröbner bases of lattices is based on the algorithm for computing Gröbner bases of lattices presented in the previous section with some additional steps added for performing truncation.

Firstly, recall that, given a lattice $\mathcal{L}$, a vector $\nu \in \mathbb{Z}^n$, and a term order $\succ$, a ν-truncated Gröbner basis of $\mathcal{L}$ is a set of vectors that is simultaneously a Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu')$ for every $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$. In the previous section, we computed a Gröbner basis of a lattice which is a Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu')$ for every ν', so now we wish to only compute a Gröbner basis for a subset of fibers.

Firstly, we extend Lemma 5.1.3 to truncated Gröbner bases of lattices.

Corollary 5.2.1. *Given a lattice $\mathcal{L}$, a term order $\succ$, and a vector $\nu \in \mathbb{Z}^n$, let $G \subseteq \mathcal{L}_\succ$ be such that G is a ν-truncated Markov basis of $\mathcal{L}$. The set G is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ if and only if there exists a $\succ$-reduction path between x' and y' for every $\succ$-critical path (x', z', y') in $\mathcal{G}_\mathcal{L}(\nu', G)$ for all $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$.*

As before for Gröbner bases of lattices, we do not need to check for a $\succ$-reduction path from x' to y' for every $\succ$-critical path (x', z', y') in $\mathcal{G}_\mathcal{L}(\nu', G)$ for all $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$, but instead, it is sufficient just to check for a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ for every critical path $(x^{(u,v)}, z^{(u,v)}, y^{(u,v)})$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$ for all $u, v \in G$ where $\nu^{(u,v)} \in \mathcal{B}_\mathcal{L}(\nu)$.

Lemma 5.2.2. *Given a lattice $\mathcal{L}$, a term order $\succ$, and a vector $\nu \in \mathbb{Z}^n$, let $G \subseteq \mathcal{L}_\succ$ such that G is a ν-truncated Markov basis of $\mathcal{L}$. The set G is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$ if and only if, for all $u, v \in G$ where $\nu^{(u,v)} \in \mathcal{B}_\mathcal{L}(\nu)$, there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$.*

Proof. If $G \subseteq \mathcal{L}_\succ$ is a ν-truncated $\succ$-Gröbner basis of $\mathcal{L}$, then by Corollary 5.2.1, there exists a reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ for all $u, v \in G$ where $\nu^{(u,v)} \in \mathcal{B}_\mathcal{L}(\nu)$.

Conversely, assume that G is a ν-truncated Markov basis of $\mathcal{L}$ and that for all $u, v \in G$ where $\nu^{(u,v)} \in \mathcal{B}_\mathcal{L}(\nu)$, there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$. Let (x', z', y') be a $\succ$-critical path in $\mathcal{G}_\mathcal{L}(\nu', G)$ where $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$, and let $u, v \in G$ where $x'+u = z' = y'+v$. Hence, $(x', z', y') = (x^{(u,v)}+\gamma, z^{(u,v)}+\gamma, y^{(u,v)}+\gamma)$ for some $\gamma \in \mathbb{N}^n$. Crucially, it follows that $\nu^{(u,v)} \in \mathcal{B}_\mathcal{L}(\nu)$ since $\gamma \in \mathcal{F}_\mathcal{L}(\nu' - \nu^{(u,v)}) \neq \emptyset$ and $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset$ which imply that $\mathcal{F}_\mathcal{L}(\nu - \nu^{(u,v)}) \neq \emptyset$. By assumption, there is a $\succ$-reduction path $x^{(u,v)}$ to $y^{(u,v)}$, and thus, there is a $\succ$-reduction path from x' to y'. Therefore, by Corollary 5.2.1, G is a ν-truncated Gröbner basis of $\mathcal{L}$. □

As above for Gröbner bases of lattices, we now turn Lemma 5.1.5 into an algorithmic tool to compute truncated Gröbner bases of lattices. Algorithm 2 below, called a truncated completion procedure ([15]), starts from a truncated Markov basis of $\mathcal{L}$ and computes a truncated $\succ$-Gröbner basis of $\mathcal{L}$. An important part of this algorithm is checking whether $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$ for some $\nu' \in \mathbb{Z}^n$. How exactly we perform this check in practice is discussed at length after first presenting the overall algorithm.

Given a set $S \subseteq \mathcal{L}$, the completion procedure first sets $G := S$, and then directs all vectors in G according to $\succ$ such that $G \subseteq \mathcal{L}_\succ$. It also removes from the set G any vectors $u \in G$ such that $u^+ \notin \mathcal{B}_\mathcal{L}(\nu)$ – recall that these vectors are not needed in a truncated Markov basis. Note that at this point $\mathcal{G}_\mathcal{L}(\nu', S) = \mathcal{G}_\mathcal{L}(\nu', G)$ for all $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$, and thus, G is also a ν-truncated Markov basis of $\mathcal{L}$. The completion procedure then determines whether the set G satisfies Lemma 5.2.2; in other words, it tries to find a reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ for every pair $u, v \in G$ where $\nu^{(u,v)} \in \mathcal{B}_\mathcal{L}(\nu)$. If G satisfies Lemma 5.2.2, then we are done. Otherwise, no $\succ$-reduction path was found for some (u, v), in which case, we add a vector to G so that a $\succ$-reduction path exists, and then again, test whether G satisfies Lemma 5.2.2 and so on.

We write $\mathcal{CP}_{\mathcal{L},\nu}(\succ, S)$ for the output of the truncated completion procedure. We can also use the truncated completion procedure to compute ν-truncated $\succ$-Gröbner

Algorithm 3 Truncated completion procedure

Input: a vector $\nu \in \mathbb{Z}^n$, a term order $\succ$, and a ν-truncated Markov basis $S \subseteq \mathcal{L}$.
Output: a ν-truncated Gröbner basis $G \subseteq \mathcal{L}_\succ$.

```
G := {u : u+ ≻ u−, u ∈ S} ∪ {−u : u− ≻ u+, u ∈ S}
G := {u : u ∈ G, u+ ∈ B_L(ν)}
C := {(u, v) : u, v ∈ G, ν^(u,v) ∈ B_L(ν)}
while C ≠ ∅ do
  Select (u, v) ∈ C
  C := C \ {(u, v)}
  r := NF_L(x^(u,v), G) − NF_L(y^(u,v), G)
  if r ≠ 0 then
    if r− ≻ r+ then r := −r end if
    C := C ∪ {(w, r) : w ∈ G, ν^(w,r) ∈ B_L(ν)}
    G := G ∪ {r}
  end if
end while
return G.
```

basis of $\mathcal{F}^\sigma_\mathcal{L}(\cdot)$, in which case, we write $\mathcal{CP}^\sigma_{\mathcal{L},\succ}(S,)$ for the output of the truncated completion procedure.

There is a trade-off between the computational benefit of computing a ν-truncated Markov basis (computing a smaller set) and the computational cost of computing whether $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$ for some $\nu' \in \mathbb{Z}^n$ many times. In general, it is NP-hard to determine whether $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$ since we must know if $\mathcal{F}_\mathcal{L}(\nu') \neq \emptyset$ and $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset$. Instead, we can check a sufficient condition for when $\nu' \notin \mathcal{B}_\mathcal{L}(\nu)$, so we compute a superset of a ν-truncated Gröbner basis since we keep some vectors that are not needed.

Firstly, note that in Algorithm 2, whenever we check whether $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$, we always have $\mathcal{F}_\mathcal{L}(\nu') \neq \emptyset$ since either $\nu' = u^+ \geq 0$ for some $u \in \mathcal{L}$ or $\nu' = \nu^{(u,v)} \geq 0$ for some $u, v \in \mathcal{L}$ and in either case $\nu' \in \mathcal{F}_\mathcal{L}(\nu') \neq \emptyset$. So, in the algorithm, we only need to check whether $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset$.

We could instead check for feasibility of a relaxation of the feasible set $\mathcal{F}_\mathcal{L}(\nu - \nu')$. One possible relaxation of $\mathcal{F}_\mathcal{L}(\nu - \nu')$ to check is

$$\mathcal{F}^\sigma_\mathcal{L}(\nu - \nu') := \{x : x - (\nu - \nu') \in \mathcal{L}, x \in \mathbb{Z}^n\}$$

where $\sigma = \{1, ..., n\}$, so we have relaxed all of the non-negativity constraints. But since $\nu - \nu' \in \mathbb{Z}^n$, we have $\nu - \nu' \in \mathcal{F}^\sigma_\mathcal{L}(\nu - \nu') \neq \emptyset$, so this is trivially always satisfied. Another possible relaxation is the linear relaxation of $\mathcal{F}_\mathcal{L}(\nu - \nu')$:

$$\mathcal{F}_\mathcal{S}(\nu - \nu') := \{x : x - (\nu - \nu') \in \mathcal{S}, x \in \mathbb{R}^n_+\}$$

where $\mathcal{S} \subseteq \mathbb{R}^n$ is the smallest subspace containing $\mathcal{L}$ (see Section 2.9). We can thus solve a linear program to check whether $\mathcal{F}_\mathcal{S}(\nu - \nu') = \emptyset$ implying that $\mathcal{F}_\mathcal{L}(\nu - \nu') = \emptyset$. Note that $\mathcal{F}_\mathcal{L}(\nu - \nu') = \mathcal{F}_\mathcal{S}(\nu - \nu') \cap \mathcal{F}^\sigma_\mathcal{L}(\nu - \nu')$, but $\mathcal{F}_\mathcal{S}(\nu - \nu') \neq \emptyset$ and $\mathcal{F}^\sigma_\mathcal{L}(\nu - \nu') \neq \emptyset$ do not imply that $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset$.

In practice, computational experiments show that it is usually not worthwhile performing the full check whether $\mathcal{F}_{\mathcal{S}}(\nu-\nu') = \emptyset$, so instead, we use a sufficient condition for when $\mathcal{F}_{\mathcal{S}}(\nu - \nu') = \emptyset$ and thus $\nu' \notin \mathcal{B}_{\mathcal{L}}(\nu)$ that is quick to check. Consider the set

$$\mathcal{S}^* \cap \mathbb{R}^n_+ := \{a \in \mathbb{R}^n_+ : as = 0 \; \forall s \in \mathcal{S}\}.$$

Recall that $\mathcal{S}^*$ is the dual of $\mathcal{S}$. Note that $\mathcal{S}^* \cap \mathbb{R}^n_+$ is a pointed convex cone. Firstly, observe that, for any $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ and any $\nu' \in \mathbb{Z}^n$, we have $ax = a\nu'$ for all $x \in \mathcal{F}_{\mathcal{L}}(\nu')$ because $x - \nu' \in \mathcal{L} \subseteq \mathcal{S}$. Secondly, if $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$, then, for all $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$, we have $a\nu \geq a\nu'$ since if $x \in \mathcal{F}_{\mathcal{L}}(\nu - \nu') \neq \emptyset$, then $a(\nu - \nu') = ax$ and $ax \geq 0$ because $a \geq 0$ and $x \geq 0$. Therefore, if $a\nu < a\nu'$, then $\mathcal{F}_{\mathcal{S}}(\nu - \nu') = \emptyset$ implying $\mathcal{F}_{\mathcal{L}}(\nu - \nu') = \emptyset$, and thus $\nu' \notin \mathcal{B}_{\mathcal{L}}(\nu)$. Moreover, it follows from Farkas' lemma (see Lemma 2.8.3) that $\mathcal{F}_{\mathcal{S}}(\nu - \nu') = \emptyset$ if and only if there exists an $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ where $a\nu < a\nu'$, and furthermore, $\mathcal{F}_{\mathcal{S}}(\nu - \nu') = \emptyset$ if and only if there exists an extreme ray a of the cone $\mathcal{S}^* \cap \mathbb{R}^n_+$ where $a\nu < a\nu'$. The set of extreme rays is finite but there are far too many of them in general to check this condition. So, we need a way of selecting one $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ or a small set of a. Choosing different a's can produce very different results, and the best a's to choose vary from fiber to fiber.

We now present a novel approach for selecting a *good* $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$. Now, note that when we run Algorithm 2 and check whether $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$, we have $a\nu' \geq 0$ for all $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ since from above $\mathcal{F}_{\mathcal{L}}(\nu') \neq \emptyset$ ($\nu' \geq 0$). Ideally, there exists $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ where $a\nu = 0$, implying that $a\nu' = 0$ (i.e. if $a_i \neq 0$, then $\nu'_i = 0$) for every $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$. This condition is very strong and is quick to check and effectively means that we compute using a sub-lattice of $\mathcal{L}$. Otherwise if $a\nu > 0$ for all $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$, then a useful heuristic is to choose a single $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ such that $a\nu$ is minimal with respect to some norm $||\cdot||$ of a. More formally, we solve the following problem:

$$\text{argmin}\{a\nu : ||a|| = 1, a \in \mathcal{S}^* \cap \mathbb{R}^n_+\}.$$

If we use the l_1-norm (i.e. $||a||_1 = \sum_i a_i$), then we can find a using linear programming. In this case, note that we only need to solve one linear program to compute a as opposed to solving a linear program every time we check whether $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$.

Example 5.2.3. *Consider again the lattice program $IP_{\mathcal{L},\succ_c}(\cdot)$ from Example 2.9.4 and Example 4.2.2 above. Recall that the set*

$$G := \{(2, -2, -2, 1), (3, 1, -1, -1), (1, 3, 1, -2), (5, -1, -3, 0), (1, -5, -3, 3), (0, -8, -4, 5)\}$$

$\succ_c$-Gröbner basis of $\mathcal{L}$. We now examine truncated $\succ_c$-Gröbner basis for two different fibers.

(i). Consider $\nu = (-6, 4, 10, -1)$. There are two feasible solutions of $IP_{\mathcal{L},\succ_c}(\nu)$: $(1, 1, 5, 0)$ and $(6, 0, 3, 2)$. Since the feasible set consists of only two feasible solutions, the minimal ν-truncated $\succ_c$-Gröbner basis contains only one vector, $(5, -1, -3, 0)$: the vector between the two feasible solutions. Therefore, the set $\{(5,-1,-3,0)\}$ is a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{L}$.

If we run the truncated completion procedure, Algorithm 3, using $\mathcal{F}_{\mathcal{S}}(\nu-\nu') \neq \emptyset$ as a check for truncation, we compute the set $\mathcal{CP}_{\mathcal{L},\nu}(\succ, s) = \{(5, -1, -3, 0)\}$, which is exactly the minimal ν-truncated $\succ_c$-Gröbner basis of $\mathcal{L}$.

If we run the truncated completion procedure using the quick test $a\nu < a\nu'$ *where* $a := \frac{1}{14}(1,5,0,8) = \text{argmin}\{a\nu : ||a||_1 = 1, a \in \mathcal{S}^* \cap \mathbb{R}^n_+\}$, *we again compute the set* $\mathcal{CP}_{\mathcal{L},\nu}(\succ, s) = \{(5,\text{-}1,\text{-}3,0)\}$. *Note that* $a\nu = \frac{1}{14}(1,5,0,8) \cdot (\text{-}6,4,10,\text{-}1) = \frac{3}{7}$. *Then, for example, the vector* $(2,\text{-}2,\text{-}2,1)$ *is not needed in a* ν*-truncated* $\succ_c$*-Gröbner basis of* $\mathcal{L}$ *because* $a\nu' = \frac{1}{14}(1,5,0,8) \cdot (2,\text{-}2,\text{-}2,1)^+ = \frac{5}{7} > a\nu$.

On the other hand, if we used the vector $a := \frac{1}{12}(3,0,5,4) \in \mathcal{S}^* \cap \mathbb{R}^n_+$, *then the quick truncation check* $a\nu < a\nu'$ *is useless, and we would compute all six vectors of* G, *the* $\succ_c$*-Gröbner basis of* $\mathcal{L}$.

(ii). Consider $\nu' := (\text{-}6,4,5,1)$*; then,* $\mathcal{F}_{\mathcal{L}}(\nu') = \{(2,4,1,0),(1,1,0,2)\}$. *Since, the feasible set consists of only two feasible solutions, the set* $\{(1,3,1,\text{-}2)\}\}$ *is a* ν'*-truncated* $\succ_c$*-Gröbner basis of* $\mathcal{L}$.

Using $\mathcal{F}_{\mathcal{S}}(\nu' - \nu'') \neq \emptyset$ *as a check for truncation, we compute the following* ν*-truncated* $\succ_c$*-Gröbner basis of* $\mathcal{L}$*:* $G := \{(2,\text{-}2,\text{-}2,1),(3,1,\text{-}1,\text{-}1),(1,3,1,\text{-}2)\}$. *So, we have computed two additional vectors that are not strictly needed since* $\mathcal{F}_{\mathcal{L}}(\nu' - \nu'') = \emptyset$ *even though* $\mathcal{F}_{\mathcal{S}}(\nu' - \nu'') \neq \emptyset$ *where* $\nu'' := (2,\text{-}2,\text{-}2,1)^+ = (2,0,0,1)$ *or* $\nu'' := (3,1,\text{-}1,\text{-}1)^+ = (3,1,0,0)$.

Instead, using $a := \frac{1}{12}(3,0,5,4) = \text{argmin}\{a\nu' : ||a||_1 = 1, a \in \mathcal{S}^* \cap \mathbb{R}^n_+\}$ *as a quick check for truncation, we compute the following three vectors of* G*:* $\{(2,\text{-}2,\text{-}2,1),(3,1,\text{-}1,\text{-}1),(1,3,1,\text{-}2)\}$. *So here, we have again computed two additional unnecessary vectors.*

If instead we use the vector $\frac{1}{14}(1,5,0,8)$*, then we would compute the following four vectors of the set* G*:* $\{(2,\text{-}2,\text{-}2,1),(3,1,\text{-}1,\text{-}1),(1,3,1,\text{-}2),(5,\text{-}1,\text{-}3,0)\}$.

The following example demonstrates the potential speed increase from computing a truncated Gröbner basis as opposed to computing a full Gröbner basis. We use three different methods for checking whether $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$ in order of increasing effectiveness but also increasing in time complexity:

(i). $a\nu < a\nu'$ where $a = \text{argmin}\{a\nu : ||a||_1 = 1, a \in \mathcal{S}^* \cap \mathbb{R}^n_+\}$,

(ii). $\mathcal{F}_{\mathcal{S}}(\nu - \nu') = \emptyset$, or

(iii). $\mathcal{F}_{\mathcal{L}}(\nu - \nu') = \emptyset$.

Since criterion (i) is in general much faster to check than (ii), we always check (i) before (ii), and similarly, since criterion (i) and (ii) are in general much faster to check than (iii), we always check them both before applying criterion (iii). In practice, we solve $\mathcal{F}_{\mathcal{S}}(\nu - \nu') = \emptyset$ using a simplex algorithm implementation in the GLPK (GNU Linear Programming Kit) package.[3] We solve $\mathcal{F}_{\mathcal{L}}(\nu - \nu') = \emptyset$ using a branch-and-bound implementation in GLPK.

[3]GLPK is open source and freely available from `http://www.gnu.org/software/glpk/`

ν	$a \in \mathcal{S}^* \cap \mathbb{R}^n_+$		$\mathcal{F}_{\mathcal{S}}(\nu - \nu'')$		$\mathcal{F}_{\mathcal{L}}(\nu - \nu'')$	
ν^1	307	0.02	1	0.09	0	0.10
ν^2	418	0.10	36	0.19	0	0.31
ν^3	6494	1.61	201	1.20	0	2.18
ν^4	12191	6.31	5028	4.43	158	186.81
ν^5	24748	108.19	24334	107.25	24284	> 3600

Table 5.1: Timings for computing truncated Gröbner bases.

Example 5.2.4. *Let* $\mathcal{L} = \mathcal{L}_A := \{u : Au = \mathbf{0}, u \in \mathbb{Z}^n\}$ *where*

$$A = \begin{bmatrix} 15 & 4 & 14 & 19 & 2 & 1 & 10 & 17 & 11 & 9 & 4 & 15 & 20 \\ 18 & 11 & 13 & 5 & 16 & 16 & 8 & 19 & 18 & 21 & 5 & 7 & 1 \\ 11 & 7 & 8 & 19 & 15 & 18 & 14 & 6 & 1 & 23 & 11 & 3 & 10 \\ 17 & 10 & 13 & 17 & 16 & 14 & 15 & 18 & 3 & 2 & 1 & 17 & 1 \end{bmatrix}$$

The size of a minimal Markov basis of $\mathcal{L}$ *is* 10868.

Let $c = (3, 15, 1, 5, 2, 17, 16, 16, 15, 9, 7, 11, 13)$. *The size of a minimal* $\prec_c$*-Gröbner basis of* $\mathcal{L}$ *for some term order* $\prec$ *is* 24941. *This takes* 106.96 *seconds to compute using* 4ti2. *In Table 5.1, we list the time taken to compute* ν*-truncated Gröbner bases of* $\mathcal{L}$ *from the minimal Markov basis of* $\mathcal{L}$ *for the following five different* $\nu \in \mathbb{Z}^n$*:*

ν^1	$(1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0)$
ν^2	$(1, 0, 1, 0, 3, 0, 1, 5, 0, 1, 0, 9, 0)$
ν^3	$(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$
ν^4	$(1, 2, 0, 3, 5, 0, 1, 3, 0, 4, 0, 1, 0)$
ν^5	$(19, 7, 3, 8, 13, 11, 1, 15, 4, 8, 17, 9, 5)$

The first column in Table 5.1 lists which ν *was used. In the following columns, we list the size of the computed set and the time taken for each of the three possible ways to check whether* $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$.

For the example above, choosing $a := \text{argmin}\{a\nu : ||a||_1 = 1, a \in \mathcal{S}^* \cap \mathbb{R}^n_+\}$ works reasonably well when used for checking for truncation. However, in general, using more than one such a might be significantly better particularly when the support of a (the set of non-zero components) is small. So, an alternate approach is to search for a set of $a's$ such that their combined support is as large as possible.

Observe that to compute a truncated Gröbner basis in the previous example, we first needed to compute a Markov basis, and in some cases, computing the Markov basis took significantly longer than computing the truncated Gröbner basis. This provides motivation for computing truncated Markov bases.

5.3 Optimising the completion procedure

The completion procedure as it is presented in Sections 5.1 and 5.2 is not very efficient. In this section, we show how to increase the efficiency of the completion

procedure and the truncated completion procedure. These optimisations are indispensable for any practical implementation of the completion procedure. We have endeavoured to provide a description of the optimisations that we found to be the most important; the list of optimisations we present is not exhaustive. There are certainly many minor optimisations that greatly improve the performance of the algorithm, but it is not appropriate to present them here. For simplicity, we only explicitly describe optimisations for the completion procedure, but all the optimisations described here also apply to the truncated completion procedure.

5.3.1 Critical pair elimination criteria

The completion procedures (Algorithms 2 and 3) must test for a reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ for each pair of vectors $u, v \in G$ to compute a Gröbner basis. Computational profiling shows that testing for reduction paths is the most time consuming part of the computation. So, we wish to reduce the number of critical pairs that we need to test and to avoid this expensive test as often as possible. We present three criteria that can reduce the number of critical pairs that need to be tested. All three criteria presented here are direct translations of critical pair elimination criteria for the completion procedure in algebraic geometry (we provide references where appropriate).

Criterion 1: The Disjoint-Positive-Support criterion

We now present **Criterion 1**, which we also call the *disjoint-positive-support* criterion for reasons that will become apparent. Criterion 1 has been translated from the theory of Gröbner bases of polynomial rings into a geometric context. See for example [13, 14] for the original description.

Definition 5.3.1. *Given a lattice $\mathcal{L}$ and a set $G \subseteq \mathcal{L}$, we say the critical pair $u, v \in G$ satisfies **Criterion 1** if* $\operatorname{supp}(u^+) \cap \operatorname{supp}(v^+) = \emptyset$.

For a given pair $u, v \in G$, Criterion 1 is a simple and quick test for a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$. Essentially, if (u, v) satisfies Criterion 1, then there exists a simple $\succ$-reduction path $x^{(u,v)}$ to $y^{(u,v)}$ only using the vectors u and v: there exists a $\succ$-reduction path $(x^{(u,v)}, \bar{z}, y^{(u,v)})$ in $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$ such that $x^{(u,v)} - v = \bar{z} = y^{(u,v)} - u$ (see Figure 5.5). Such a path exists if and only if $\bar{z} \geq \mathbf{0}$. Recall that $z^{(u,v)} := u^+ \vee v^+$,

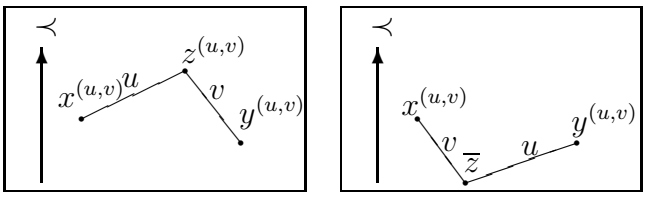

Figure 5.5: Criterion 1.

$x^{(u,v)} := z^{(u,v)} - u$, and $y^{(u,v)} := z^{(u,v)} - v$, so $\bar{z} = z^{(u,v)} - u - v = (u^+ \vee v^+) - u_i - v_i$.

Thus, for some $i \in \{1, ..., n\}$, we have $\bar{z}_i < 0$ if and only if $u_i > 0$ and $v_i > 0$. Therefore, $\bar{z} \geq 0$ if and only if $\text{supp}(u^+) \cap \text{supp}(v^+) = \emptyset$.

We can now add criterion 1 to Lemma 5.1.5 as follows.

Lemma 5.3.2. *Given a lattice $\mathcal{L}$ and a term order $\succ$, let $G \subseteq \mathcal{L}_\succ$ be such that G is a Markov basis of $\mathcal{L}$. The set $G \subseteq \mathcal{L}_\succ$ is a $\succ$-Gröbner basis of $\mathcal{L}$ if and only if, for all pairs $u, v \in G$ that do not satisfy criterion 1, there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$.*

Criterion 2: The Cancellation criterion

We now discuss **Criterion 2**, which we also call the *cancellation criterion*. Criterion 2 is specific to lattice ideals and follows from the theory behind the homogeneous Buchberger algorithm ([16, 89]) for computing Gröbner bases of polynomial rings. However, Criterion 2 as presented here is slightly more general in a way that we will explain below.

To explain criterion 2, we need the concept of a grading. Let $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$, and recall that $ax = ay$ for all $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ for all $\nu \in \mathbb{Z}^n$. We call a a **grading** of $\mathcal{L}$. Importantly, if τ is the set of *bounded* components, then there exists a grading a of $\mathcal{L}$ such that $\text{supp}(a) = \tau$. A component $i \in \{1, ..., n\}$ is **bounded** if $IP_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu) := \max\{x_i : x - \nu \in \mathcal{L}, x \geq \mathbf{0}, x \in \mathbb{Z}^n\}$ is bounded for every $\nu \in \mathbb{Z}^n$, otherwise i is **unbounded**. Note that $\mathbf{e}^i$ is the ith unit vector. We now show that if τ is the set of bounded components that there exists a grading $s \in \mathcal{S}^* \cap \mathbb{R}^n_+$ such that $\text{supp}(a) = \tau$. Recall from Lemma 2.9.1 that $IP_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu)$ for all $\nu \in \mathbb{Z}^n$ is bounded if and only if the linear space program $LP_{\mathcal{S},\text{-}\mathbf{e}^i}(\nu)$ is bounded for all $\nu \in \mathbb{Z}^n$. Furthermore, from Lemma 2.8.3, the linear space program $LP_{\mathcal{S},\text{-}\mathbf{e}^i}(\nu)$ is bounded for all $\nu \in \mathbb{Z}^n$ if and only if $\text{-}\mathbf{e}^i \in \mathcal{F}_\mathcal{S}(\mathbf{0})^* = \{y \in \mathbb{R}^n : y \geq s, s \in \mathcal{S}^*\}$ or equivalently there exists $s \in \mathcal{S}^*$ such that $s \geq \mathbf{e}^i$. Any vector $s \in \mathcal{S}^*$ such that $s \geq \mathbf{e}^i$ is a grading, and if we take the sum of all such gradings for every bounded component, we have a grading a such that $\text{supp}(a) = \tau$. Note that, using the arguments from above, we cannot have a grading a such that $a_i > 0$ where i is unbounded. Thus, the set of bounded components precisely describes the maximum support of any grading.

First, we prove an analogous result to Corollary 5.1.4.

Lemma 5.3.3. *Given a lattice $\mathcal{L}$, and a term order $\succ$, let $G \subseteq \mathcal{L}_\succ$ be such that G is a Markov basis of $\mathcal{L}$, and let τ be the bounded components. The set G is a $\succ$-Gröbner basis of $\mathcal{L}$ if and only if for every $\succ$-critical path (x, z, y) in $\mathcal{G}_\mathcal{L}(\nu, G)$ for all $\nu \in \mathbb{Z}^n$ where $\text{supp}(x) \cap \text{supp}(y) \cap \tau = \emptyset$, there exists a $\succ$-reduction path between x and y in $\mathcal{G}_\mathcal{L}(\nu, G)$.*

Proof. The forward implication follows from Corollary 5.1.4. For the backward implication, we need to show that for every $\succ$-critical path (x, z, y) in $\mathcal{G}_\mathcal{L}(\nu, G)$ for all $\nu \in \mathbb{Z}^n$ where $\text{supp}(x) \cap \text{supp}(y) \cap \tau \neq \emptyset$, there exists a $\succ$-reduction path from x to y in $\mathcal{G}_\mathcal{L}(\nu, G)$, in which case, there is a $\succ$-reduction path for all $\succ$-critical paths, so by Corollary 5.1.4, G is a Gröbner basis. Assume on the contrary that this is not the case.

Let $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ where $\text{supp}(a) = \tau$. Then, amongst all possible $\succ$-critical paths (x, z, y) in $\mathcal{G}_\mathcal{L}(\nu, G)$ for some $\nu \in \mathbb{Z}^n$ such that there is no $\succ$-reduction path from x to y ($\text{supp}(x) \cap \text{supp}(y) \cap \tau \neq \emptyset$), choose a path (x, z, y) such that $a\nu = ax = ay$ is minimal. Let $\gamma := x \wedge y$ (the component-wise minimum, $\gamma_i = \min\{x_i, y_i\}$ for all $i = 1, ..., n$), $\bar{x} := x - \gamma$, $\bar{y} := y - \gamma$, and $\bar{\nu} = \nu - \gamma$. Note that $\gamma \neq \mathbf{0}$ since $\text{supp}(x) \cap \text{supp}(y) \cap \tau \neq \emptyset$. Because G is a Markov basis of $\mathcal{L}$, there must exist a path from $\bar{x}$ to $\bar{y}$ in $\mathcal{G}_\mathcal{L}(\bar{\nu}, G)$. Also, since $\text{supp}(a) = \tau$, we have $a\bar{\nu} = a\bar{x} < ax = a\nu$; therefore, by the minimality assumption on $a\nu$, we can now conclude that for all $\succ$-critical paths in $\mathcal{G}_\mathcal{L}(\bar{\nu}, G)$ there exists a $\succ$-reduction path. Consequently, by Lemma 5.1.3, there exists a $\succ$-reduction path between $\bar{x}$ and $\bar{y}$ in $\mathcal{G}_\mathcal{L}(\bar{\nu}, G)$. This $\succ$-reduction path, however, can be translated by γ to a $\succ$-reduction path from x to y in $\mathcal{G}_\mathcal{L}(\nu, G)$ (see Figure 5.6a). But this contradicts our assumption that there is no such $\succ$-reduction path between x and y. □

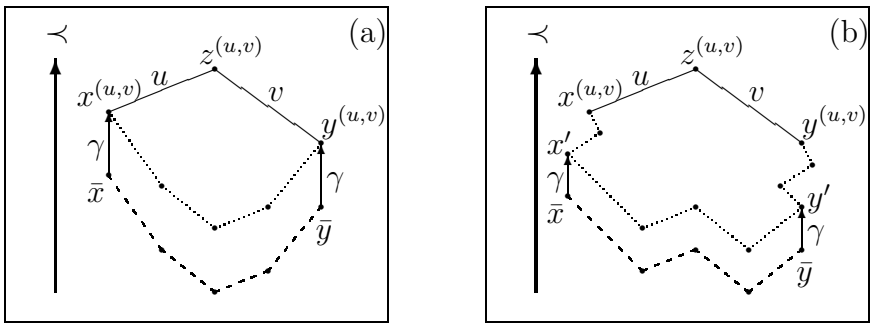

Figure 5.6: Criterion 2.

Now, given $u, v \in G$, if $\text{supp}(x^{(u,v)}) \cap \text{supp}(y^{(u,v)}) \cap \tau \neq \emptyset$, then $\text{supp}(x) \cap \text{supp}(y) \cap \tau \neq \emptyset$ for all $\succ$-critical paths (x, z, y) for (u, v). Using this observation, we arrive at an analogous result to Lemma 5.1.5.

Lemma 5.3.4. *Let $G \subseteq \mathcal{L}$ be a Markov basis of $\mathcal{L}$; then, G is a $\prec$-Gröbner basis of $\mathcal{L}$ if and only if, for each pair $u, v \in G$ where $\text{supp}(x^{(u,v)}) \cap \text{supp}(y^{(u,v)}) \cap \tau = \emptyset$, there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$.*

We can extend these results further leading to a more powerful elimination criterion. We now present Criterion 2: let $u, v \in G$. We say the pair (u, v) satisfies Criterion 2 if there exists $x', y' \in \mathcal{F}_\mathcal{L}(\nu^{(u,v)})$ such that there exists a $\succ$-decreasing path in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$ from $x^{(u,v)}$ to x' and from $y^{(u,v)}$ to y', and $\text{supp}(x') \cap \text{supp}(y') \cap \tau \neq \emptyset$ where τ is the set of bounded components. Importantly, if (u, v) satisfies Criterion 2, then we do not have to test for a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$. Thus, we arrive at an extension of the above result. Observe that the previous results are just a special case where $x' = x^{(u,v)}$ and $y' = y^{(u,v)}$.

Lemma 5.3.5. *Let $G \subseteq \mathcal{L}$ be a Markov basis of $\mathcal{L}$; then, G is a $\prec$-Gröbner basis of $\mathcal{L}$ if and only if, for each pair $u, v \in G$ where (u, v) does not satisfy Criterion 2, there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$.*

If there is a $\succ$-reduction path from x' to y', then there exists a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$. Since $\operatorname{supp}(x') \cap \operatorname{supp}(y') \cap \tau \neq \emptyset$, then $\gamma = x' \wedge y' \neq \mathbf{0}$. So, if there exists a $\succ$-reduction path from $(x' - \gamma)$ to $(y' - \gamma)$, then there must exist a $\succ$-reduction path from x' to y', and therefore also, there must exist a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ (see Figure 5.6b). Again, we let a be a strictly positive grading of $\mathcal{L}$, so similarly to above, $a\bar{\nu} < a\nu^{(u,v)}$. So, the proof of Lemma 5.3.5 is essentially as before.

For a pair $u, v \in G$, Criterion 2 can be checked not only before we search for a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ but also while searching for a $\succ$-reduction path. When searching for a $\succ$-reduction path, we construct a $\succ$-decreasing path from $x^{(u,v)}$ to $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$ and a $\succ$-decreasing path from $y^{(u,v)}$ to $\mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. Therefore, we can take any point x' on the $\succ$-decreasing path from $x^{(u,v)}$ to $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$ and any point y' on the decreasing path from $y^{(u,v)}$ to $\mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$ and check Criterion 2, that is, we check if $\operatorname{supp}(x') \cap \operatorname{supp}(y') \cap \tau \neq \emptyset$. If this is true, then we can eliminate (u, v).

We wish to point out explicitly here that Criterion 2 can be applied without choosing the vector pairs $u, v \in G$ in a particular order during Algorithm 2. In fact, when running Algorithm 2, if we apply Criterion 2 to eliminate a pair $u, v \in G$, it does not necessarily mean that there is a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ in $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$ at that particular point in time in the algorithm but instead that a $\succ$-reduction path will exist when the algorithm terminates. This approach is in contrast to existing approaches that use the homogeneous Buchberger algorithm to compute a Gröbner basis whereby vector pairs $u, v \in G$ must be chosen in an order compatible with increasing $a\nu^{(u,v)}$ for some strictly positive grading a. This can be computationally costly. When we use these existing approaches, if a pair (u, v) is eliminated by Criterion 2, then it is necessarily the case that there already exists a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$.

Observe that we can apply Criteria 1 and 2 simultaneously, so we only need to check for a reduction path for a critical pair if it does not satisfy Criteria 1 and 2. Actually in practice, Criterion 2 is usually more effective for eliminating critical pairs than Criterion 1, and thus, it is better to check Criterion 2 and then check Criterion 1.

Note that the effectiveness of Criterion 2 depends on the set τ of bounded components. In particular, if $\tau = \emptyset$, then Criterion 2 is useless. For cases that are mostly unbounded (empirically, when less than one third of the components are bounded), we need another criterion for eliminating critical pairs.

We now redo the completion procedure computation in example 5.1.7 using Criterion 2 to eliminate critical pairs in order to demonstrate the effectiveness of Criterion 2.

Example 5.3.6. *Consider again the lattice $\mathcal{L}$ as in Example 5.1.7. Again let $\nu :=$ (-6, 4, 10, 1), and let $c := (2, 0, 1, 1)$. We now present an example computation of a $\succ_c$-Gröbner basis of a $\mathcal{L}$ using the completion procedure (Algorithm 2) and this time, we also use Criterion 2 and Criterion 1 to eliminate critical pairs. Note that for this example, every component is bounded.*

Again let $M :=${(3,1,-1,-1),(1,3,1,-2),(2,-2,-2,1),(5,-1,-3,0)}*. Then, set $G := M$ and and set $C := \{(u, v) : u, v \in G\}$. We now process the critical pairs C of G.*

(i). Select (u, v) =((3,1,-1,-1),(1,3,1,-2)). *Criterion 2 eliminates this pair since* $\text{supp}(u^-) = \{3, 4\}$ *and* $\text{supp}(v^-) = \{4\}$*; hence,* $\text{supp}(u^-) \cap \text{supp}(v^-) \neq \emptyset$.

(ii). Select (u, v) =((3,1,-1,-1),(2,-2,-2,1)). *Criterion 2 eliminates this pair.*

(iii). Select (u, v) =((1,3,1,-2),(2,-2,-2,1)). *Neither Criterion 2 nor Criterion 1 eliminate this pair. Then,* $x^{(u,v)} = (1, 0, 0, 3) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and,* $y^{(u,v)} = (0, 5, 3, 0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. *So,* $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) \neq \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$*! Therefore, we have found a critical pair* (u, v) *such that there is no reduction path from* $x^{(u,v)}$ *to* $y^{(u,v)}$ *in* $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$, *and thus from Lemma 5.1.1, we have proven that* G *is not a* $\prec_c$*-Gröbner basis of* $\mathcal{L}$.

Let $r = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) - \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$ =(1,-5,-3,3). *We must now add* (v, r) *to* C *for all* $v \in G$ *and add* r *to* G. *So, now*

$$G := \{(3, 1, -1, -1), (1, 3, 1, -2), (2, -2, -2, 1), (5, -1, -3, 0), (1, -5, -3, 3)\}.$$

(iv). Select (u, v) =((3,1,-1,-1),(5,-1,-3,0)). *Criterion 2 eliminates this pair.*

(v). Select (u, v) =((1,3,1,-2),(5,-1,-3,0)). *Neither Criterion 2 nor Criterion 1 eliminate this pair. Then,* $y^{(u,v)} = (0, 4, 4, 0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

$$\begin{aligned} x^{(u,v)} = (4, 0, 0, 2) &\xrightarrow{(2,-2,-2,1)} (2, 2, 2, 1) \\ &\xrightarrow{(2,-2,-2,1)} (0, 4, 4, 0) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G). \end{aligned}$$

So, $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$.

(vi). Select (u, v) =((2,-2,-2,1),(5,-1,-3,0)). *Criterion 2 eliminates this pair.*

(vii). Select (u, v) =((3,1,-1,-1),(1,-5,-3,3)). *Criterion 2 eliminates this pair.*

(viii). Select (u, v) =((1,3,1,-2),(1,-5,-3,3)). *Neither Criterion 2 nor Criterion 1 eliminate this pair. Then,* $x^{(u,v)} = (0, 0, 0, 5) = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G)$, *and,* $y^{(u,v)} = (0, 8, 4, 0) = \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$. *So,* $\mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) \neq \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$*! Therefore, we have found another critical pair* (u, v) *such that there is no reduction path from* $x^{(u,v)}$ *to* $y^{(u,v)}$ *in* $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$, *and thus from Lemma 5.1.1, we have proven that* G *is still not a* $\prec_c$*-Gröbner basis of* $\mathcal{L}$.

Let $r = \mathcal{NF}_{\mathcal{L}}(x^{(u,v)}, G) - \mathcal{NF}_{\mathcal{L}}(y^{(u,v)}, G)$ =(0,-8,-4,5). *We must now add* (v, r) *to* C *for all* $v \in G$ *and add* r *to* G. *So, now* $G :=$

$$\{(3, 1, -1, -1), (1, 3, 1, -2), (2, -2, -2, 1), (5, -1, -3, 0), (1, -5, -3, 3), (0, -8, -4, 5)\}.$$

(ix). Select (u, v) =((2,-2,-2,1),(1,-5,-3,3)). *Criterion 2 eliminates this pair.*

(x). Select (u, v) =((5,-1,-3,0),(1,-5,-3,3)). *Criterion 2 eliminates this pair.*

(xi). Select (u, v) =((3,1,-1,-1),(0,-8,-4,5)). *Criterion 2 eliminates this pair.*

(xii). Select (u, v) =((1,3,1,-2),(0,-8,-4,5)). *Criterion 1 eliminates this pair since* $\text{supp}(u^+) = \{1, 2, 3\}$ *and* $\text{supp}(v^+) = \{4\}$, *so* $\text{supp}(u^+) \cap \text{supp}(v^+) = \emptyset$.

(xiii). Select (u, v) =((2,-2,-2,1),(0,-8,-4,5)). *Criterion 2 eliminates this pair.*

(xiv). Select (u, v) =((5,-1,-3,0),(0,-8,-4,5)). *Criterion 2 eliminates this pair.*

(xv). Select (u, v) =((1,-5,-3,3),(0,-8,-4,5)). *Criterion 2 eliminates this pair.*

So now, for all $u, v \in G$, *there exists a* $\prec_c$*-reduction path from* $x^{(u.v)}$ *to* $y^{(u,v)}$, *so*

$$G := \{(3,1,-1,-1),(1,3,1,-2),(2,-2,-2,1),(5,-1,-3,0),(1,-5,-3,3),(0,-8,-4,5)\}.$$

is a $\prec_c$*-Gröbner basis of* $\mathcal{L}$. *Note how effective the two criteria were in decreasing the number of pairs that we must check for a reduction path. In only one case (step (v)) was there a reduction path for a pair that wasn't eliminated. Also, note that criterion 2 was more effective than criterion 1. In general, computational experiments show that most critical pairs are eliminated with criterion 2 and some with criterion 1 so that there are not many pairs that must be checked for a reduction path.*

Criterion 3: The (u, v, w) criterion

We now explain **Criterion** 3, which we also call the (u, v, w) criterion. This is the most complicated of the three criteria and as such it is also the most time consuming to check. However, it is particularly useful when most components are unbounded. Criterion 3 has been translated from the theory of Gröbner bases of polynomial rings into a geometric context. See for example [41, 28] for the original description.

Before presenting the (u, v, w) criterion, we need a another result, Lemma 5.3.7, that is a less strict version of Lemma 5.1.3. First, we need to define a new type of path. A path $(x^0, \ldots, x^k)$ is z-**bounded** (with respect to $\succ$) if $z \succ x^i$ for all $i = 0, \ldots, k$. So, z is a strict upper bound on the path. Note that for a $\succ$-critical path (x, z, y), a $\succ$-reduction path from x to y is by definition a z-bounded path.

Lemma 5.3.7. *Given* $\nu \in \mathbb{Z}^n$, *let* $G \subseteq \mathcal{L}_\succ$ *where* G *is a Markov basis of* $\mathcal{F}_\mathcal{L}(\nu)$. *The set* G *is a* $\succ$*-Gröbner basis of* $\mathcal{F}_\mathcal{L}(\nu)$ *if and only if there exists a* z*-bounded path between* x *and* y *for every* $\succ$*-critical path* (x, z, y) *in* $\mathcal{G}_\mathcal{L}(\nu, G)$.

If we now re-examine the proof of Lemma 5.1.3, we find that we only need z'-bounded paths between x' and y' for every $\succ$-critical path (x', z', y') in $\mathcal{G}_\mathcal{L}(\nu, G)$, and that, a $\succ$-reduction path from x' and y' is more than we need. The proof then proceeds in the same way as Lemma 5.1.3.

From Lemma 5.3.7, we arrive at an analogous result to Lemma 5.1.5.

Corollary 5.3.8. *Given a lattice* $\mathcal{L}$ *and a term order* $\succ$, *let* $G \subseteq \mathcal{L}_\succ$ *such that* G *is a Markov basis of* $\mathcal{L}$. *The set* $G \subseteq \mathcal{L}_\succ$ *is a* $\succ$*-Gröbner basis of* $\mathcal{L}$ *if and only if, for all* $u, v \in G$, *there exists a* $z^{(u,v)}$*-bounded path between* $x^{(u,v)}$ *and* $y^{(u,v)}$ *in* $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$.

Corollary 5.3.8 does not mean that we should change Algorithm 2 since to test for a $z^{(u,v)}$-bounded path from $x^{(u,v)}$ to $y^{(u,v)}$, we still test for a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ which is a $z^{(u,v)}$-bounded path. However, we can use Corollary 5.3.8

to reduce the number of critical pairs $u, v \in G$ for which we need to compute a $\succ$-reduction path.

Now, we are able to present the (u, v, w) criterion. Let $u, v, w \in G$ where $z^{(u,v)} \geq w^+$ (or equivalently, $z^{(u,v)} \geq z^{(u,w)}$ and $z^{(u,v)} \geq z^{(w,v)}$), and let $\bar{z} = z^{(u,v)} - w$ (see Figure 5.7). Then, a $z^{(u,v)}$-bounded path from $x^{(u,v)}$ to $\bar{z}$, and a $z^{(u,v)}$-bounded path from $\bar{z}$ to $y^{(u,v)}$ combine to form a $z^{(u,v)}$-bounded path from $x^{(u,v)}$ to $y^{(u,v)}$. Moreover, $(x^{(u,v)}, z^{(u,v)}, \bar{z})$ is a $\succ$-critical path for (u, w) (see Figure 5.7), so $x^{(u,v)} = x^{(u,w)} + \gamma$, $z^{(u,v)} = z^{(u,w)} + \gamma$, and $\bar{z} = y^{(u,w)} + \gamma$ for some $\gamma \in \mathbb{N}^n$, and thus, a $z^{(u,w)}$-bounded path from $x^{(u,w)}$ to $y^{(u,w)}$ translates to a $z^{(u,v)}$-bounded path from $x^{(u,v)}$ to $\bar{z}$. Furthermore, $(\bar{z}, z^{(u,v)}, y^{(u,v)})$ is a $\succ$-critical path for (w, v), and thus, a $z^{(w,v)}$-bounded path from $x^{(w,v)}$ to $y^{(w,v)}$ translates to a $z^{(u,v)}$-bounded path from $\bar{z}$ to $y^{(u,v)}$. Therefore, a $z^{(u,w)}$-bounded path from $x^{(u,w)}$ to $y^{(u,w)}$ and a $z^{(w,v)}$-bounded path from $x^{(w,v)}$ to $y^{(w,v)}$ combine to form a $z^{(u,v)}$-bounded path from $x^{(u,v)}$ to $y^{(u,v)}$, so we can remove (u, v) from C.

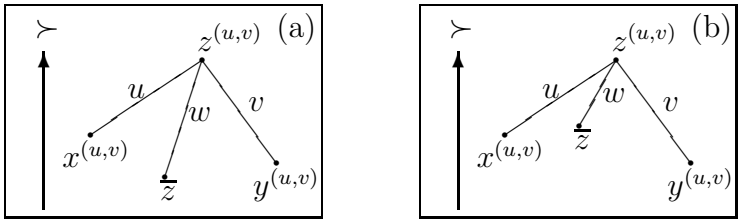

Figure 5.7: Criterion 3.

Note that in Figure 5.7a, a $\succ$-reduction path from $x^{(u,v)}$ to $\bar{z}$ and a $\succ$-reduction path from $\bar{z}$ to $y^{(u,v)}$ do combine to give a $\succ$-reduction path from $x^{(u,v)}$ to $y^{(u,v)}$; however, this is not the case in Figure 5.7b, which is why we need the concept of bounded paths.

We can extend the previous result. Let $u, v, \in G$, and $w^1, \ldots, w^k \in G$ such that $z^{(u,v)} \geq (w^i)^+$ for all $i = 1, \ldots, k$. If there exists a $z^{(u,v)}$ bounded path for the critical pairs (u, w^1), (w^k, v), and (w^i, w^{i+1}) for all $i = 1, \ldots, k-1$, then there is a bounded path for (u, v). However, note that this can also be implied by a bounded path for (u, w^i) and (w^i, v) for any $i = 1, \ldots, k$.

Unfortunately, we cannot just remove from C all pairs $u, v \in G$ where there exists a $w \in G$ such that $z^{(u,v)} \geq w^+$. It may happen that in addition to $z^{(u,v)} \geq w^+$, we also have $z^{(u,w)} \geq v^+$, in which case, we would eliminate both the pairs (u, v) and (u, w) leaving only (v, w) which is not sufficient. Moreover, at the same time, we may also have $z^{(w,v)} \geq u^+$, and we would eliminate all three pairs. To avoid these circular relationships, Gebauer and Möller [41] devised the following critical pair elimination criteria, which we use in practice in 4ti2.

Let $G = \{u^1, u^2, \ldots, u^{|G|}\}$, and let $u^i, u^j \in G$ where $i < j$. We now define Criterion 3. the pair (u^i, u^j) satisfies Criterion 3 if there exists $u^k \in G$ such that one of the following conditions hold:

(i). $z^{(u^i,u^j)} \gneq z^{(u^i,u^k)}$, and $z^{(u^i,u^j)} \gneq z^{(u^j,u^k)}$;

(ii). $z^{(u^i,u^j)} = z^{(u^i,u^k)}$, $z^{(u^i,u^j)} \gneq z^{(u^j,u^k)}$, and $k < j$;

(iii). $z^{(u^i,u^j)} \gneq z^{(u^i,u^k)}$, $z^{(u^i,u^j)} = z^{(u^j,u^k)}$, and $k < i$; or

(iv). $z^{(u^i,u^j)} = z^{(u^i,u^k)} = z^{(u^j,u^k)}$, and $k < i < j$.

So, if a pair (u^i, u^j) satisfies Criterion 3, we can eliminate it. For example, if $G = \{u^1, u^2, u^3\}$ where $z^{u^1u^2} = z^{u^1u^3} \gneq z^{u^2u^3}$, then applying Criterion 3 to all three pairs (u^1, u^2), (u^1, u^3), and (u^2, u^3) would eliminate only (u^1, u^3).

After eliminating all pairs that satisfy Criterion 3, we are left with a set of critical pairs $C' \subseteq C = \{(u, v) : u, v \in G\}$ such that if there exists a $z^{(u',v')}$-bounded path from $x^{(u',v')}$ to $y^{(u',v')}$ for all $(u', v') \in C'$, then there exists a $z^{(u,v)}$-bounded path from $x^{(u,v)}$ to $y^{(u,v)}$ for all $(u, v) \in C$. However, this set of pairs may not be minimal. In [16], Caboara, Kreuzer, and Robbiano describe an algebraic algorithm for computing a minimal set of critical pairs with computational results. Their computational results show that the Gebauer and Möller criteria give a good approximation to the minimal set of critical pairs. We found that the Gebauer and Möller criteria were sufficient for our computations.

5.3.2 Finding reduction paths

As mentioned above, computational profiling shows that most of the time spent in the completion procedure is spent in checking for a reduction path from $x^{(u,v)}$ to $y^{(u,v)}$ in the graph $\mathcal{G}_{\mathcal{L}}(\nu^{(u,v)}, G)$ for some set of vectors G and vectors $u, v \in G$. In general, this is the case even after applying the critical pair elimination criteria to avoid performing the check as often as possible. In this section, we describe how we can improve the algorithm for checking for a reduction path.

Recall that to check for a reduction path from x to y in the graph $\mathcal{G}_{\mathcal{L}}(\nu, G)$, we construct a maximal decreasing path from x to $x' = \mathcal{NF}_{\mathcal{L}}(x, G)$ using the Normal Form algorithm, and also, we construct a maximal decreasing path from y to $y' = \mathcal{NF}_{\mathcal{L}}(y, G)$, and then, if $x' = y'$ (i.e. the paths cross), we have found a reduction path by joining the two decreasing paths and removing cycles if necessary.

In situations where $x' = y'$ (i.e. we have found a reduction path from x to y), this simple approach potentially performs some unnecessary steps. This happens when the path from x to x' and the path from y to y' cross in the middle of the paths, and thus, when we join the paths, we create cycles. We can improve the algorithm by avoiding these cycles and only computing the paths up until the point where they first cross. More formally, let $(x^1, ..., x^k)$ be the decreasing path from x to x' constructed by the Normal Form algorithm, and similarly, let $(y^1, ..., y^l)$ be a decreasing path from y to y' constructed by the Normal Form algorithm. It may happen that $x^i = y^j$ for some $i < k$ and $j < l$. In this case, we only need to compute the shorter paths $(x^1, ..., x^i)$ and $(y^1, ..., y^j)$ since joining them creates a reduction path from x to y, and in only computing these shorter paths, we have avoided performing some iterations of the Normal Form algorithm.

To avoid performing unnecessary steps of the Normal Form algorithm, we compute the two decreasing paths in parallel. For example, assume that we have computed

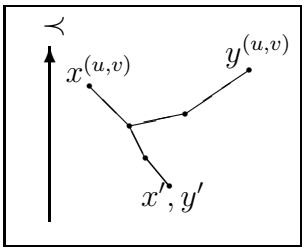

Figure 5.8: Intersecting decreasing paths from $x^{(u,v)}$ and $y^{(u,v)}$.

$(x^1, ..., x^i)$ and $(y^1, ..., y^j)$. If $x^i = y^j$, then we stop. Otherwise, if $x^i \succ y^j$, then we try to decrease x^i, or if $y^i \succ x^j$, then we try to decrease y^i. In doing this, we avoid cycles.

5.3.3 Finding a reductor

The main step of the reduction path algorithm and the Normal Form algorithm, Algorithm 1, is to find a vector $u \in G$ such that $u^+ \leq x$ for some $x \in \mathbb{N}^n$, in which case, we call u a reductor of x. This single operation is the most expensive operation of the completion procedure, and overall, the time spent performing this operation accounts for the most computation time in the completion procedure. So, any enhancements to this operation make a substantial improvement to the performance of the completion procedure and the Normal Form algorithm. There are many possible ways of implementing this operation. The approach that we present here is the way that we found to be the best after evaluating many different approaches. To the best of our knowledge, no other software packages for computing Gröbner bases use the approach presented in this chapter. We do not wish to present all the other possibilities that we considered, but instead, we will focus on the method that we found to be the best in practice. Fortunately, we found that the method presented here really dominates other approaches that we tried in the sense that it performs at least as well if not better for all different types of sets G of a reasonable size.

The simplest approach to finding a vector $u \in G$ such that $u^+ \leq x$ is to treat the set G as an ordered list of vectors – $G = \{u^1, u^2, ..., u^k\}$ – and to iterate through the list from start to end checking for each vector $u^j \in G$ whether $(u^j)^+ \leq x$. The simplicity of this approach means that it performs well when G is small (less than a few hundred vectors), but we are interested in cases where G is well over 100,000 vectors. It is worth noting here that when adding a new vector to the list G during the completion procedure, in our experience, this simple method performs better when new vectors are added at the end of the list and not at the beginning. We can improve on this simple method a little by noting that $u^+ \leq x$ only if $||u^+||_1 \leq ||x||_1$. So, we sort the list of vectors in G in order of increasing norm $|| \cdot ||_1$ and if two vectors have the same norm, the vectors are ordered from oldest to newest. In this way, we only have to iterate through the list until $||u^+||_1 > ||x||_1$, at which point, if we have not yet found a reductor of x, then none exists.

We can again improve upon this method using the observation that $u^+ \leq x$ only if

$\text{supp}(u^+) \subseteq \text{supp}(x)$. Thus, we first check if $\text{supp}(u^+) \subseteq \text{supp}(x)$ before checking if $u^+ \leq x$. Also, since potentially many different vectors in G have the same positive support, we group the vectors in G into sets of vectors with the same positive support. By doing this, we only need to perform one positive support check per group, which is a big advantage of this method. Let $G = G^1 \cup G^2 \cup ... \cup G^l$ where G^j is a list of vectors in G with the same support, and let $S = \{s^1, s^2, ..., s^l\}$ where $s^j = \text{supp}(u^+)$ for all $u \in G^j$ for $j = 1, ..., l$. Then, to find a vector $u \in G$ such that $u^+ \leq x$, we can iterate through the list S from start to end checking for each $s^j \in S$ whether $s^j \subseteq \text{supp}(x)$ and if so, then we iterate through G^j as above looking for a vector $u^j \in G^j$ where $(u^j)^+ \leq x$. Note that, as above, we can sort the vectors of each G^j in order of increasing norm $||\cdot||_1$ and if two vectors have the same norm, the vectors are ordered from oldest to newest. We can again improve this method by observing that $\text{supp}(u^+) \subseteq \text{supp}(x)$ only if $|\text{supp}(u^+)| \leq |\text{supp}(x)|$, so we can order the groups in increasing size of the positive support. In this way, we only have to iterate through the list of groups until $|s^j| > |\text{supp}(x)|$ for some $j = 1, ..., l$.

The approach is inefficient when the number of different groups of vectors with the same positive support is larger than a few thousand, in which case, it is no longer efficient to iterate through the list S of positive supports from start to end. Instead, we use a tree structure to search for $s \in S$ such that $s \subseteq \text{supp}(x)$. Let $T = (V, E)$ be a tree where V are the vertices or nodes and E are the edges. Each node $v \in V$ of the tree T except the root node has a label $l_v \in \{1, ..., n\}$, and by convention, we give the root node the label 0. The label of each node is always strictly greater than the label of its parent node, and we require that the label of a node is unique amongst its siblings, although the label might not be unique amongst all nodes. For any node $v \in V$, the list of labels of its ancestors including its own label but excluding the root node's label is a subset of $\{1, ..., n\}$. Moreover, this subset is unique for every node of the tree since labels are increasing as we move down the tree and that labels are unique amongst the siblings of a node. So, given a list of supports S, we can construct a tree T such that each support $s \in S$ and the set of vectors in G with positive support s is associated with one node of the tree. Let s_v be the support associated with node v and let G_v be the set of vectors associated with node v that all have positive support s_v.

Then, if we wish to find a vector $u \in G$ such that $u^+ \leq x$ we traverse the tree T in a depth first search fashion starting from the root node where we only visit a node $v \in V$ if $l_v \in \text{supp}(x)$. So, if we visit a node $v \in V$ in the tree T, then $l_v \in \text{supp}(x)$ and also $l_{v'} \in \text{supp}(x)$ for every node $v' \in V$ that is an ancestor of v excluding the root node implying that $s_v \subseteq \text{supp}(x)$, in which case, we should check whether $u^+ \leq x$ for every $u \in G_v$. If we do not visit a node $v \in V$, then $l_{v'} \notin \text{supp}(x)$ for some ancestor of v' of v excluding the root node implying that $s_v \not\subseteq \text{supp}(x)$ and thus $u^+ \not\leq x$ for every $u \in G_v$.

Example 5.3.9. *Consider again the lattice $\mathcal{L}$ and cost vector c from Example 4.2.2. We saw that the set*

$$G = \{(2, -2, -2, 1), (3, 1, -1, -1), (1, 3, 1, -2), (5, -1, -3, 0), (1, -5, -3, 3), (0, -8, -4, 5)\}$$

was a $\succ_c$-Gröbner basis of $\mathcal{L}$. We list the positive supports of these vectors in Table 5.2 below.

#	Vector	Positive Support
1	(2,-2,-2,1)	{1,4}
2	(3,1,-1,-1)	{1,2}
3	(1,3,1,-2)	{1,2,3}
4	(5,-1,-3,0)	{1}
5	(1,-5,-3,3)	{1,4}
6	(0,-8,-4,5)	{4}

Table 5.2: Positive supports of vectors in G.

We have depicted the support tree for the set G in Figure 8.2(a). The set of numbers to the bottom right of each node is the set of vectors of G (numbered as in Table 5.2) that are associated with the node. In Figure 5.9(b), we show the part of the support tree that we traverse using a depth first search approach when looking for a vector $u \in G$ such that $u^+ \leq x = (2, 2, 0, 0)$. Note that $\mathrm{supp}(x) = \{1, 2\}$*. We have highlighted the nodes that are visited.*

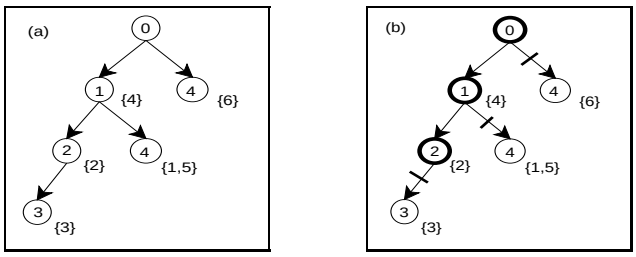

Figure 5.9: A support tree.

5.4 Alternative Algorithms

In this section, we discuss some alternative approaches for computing a Gröbner basis of a lattice that do not use the completion procedure to compute a Gröbner basis as presented above. Our purpose here is not to describe in detail the alternatives, but instead only to make the reader aware that there are possible alternatives to the completion procedure and what those alternatives are.

5.4.1 Graver basis algorithm

The first alternative approach to compute a $\succ$-Gröbner basis of $\mathcal{L}$ is to compute a Graver basis of $\mathcal{L}$ instead.

Definition 5.4.1. *A Graver basis of $\mathcal{L}$ is a set of vectors $G \subset \mathcal{L}$ such that, for every $v \in \mathbb{Z}^n$, there exists $u \in G$ such that $u^+ \leq v^+$ and $u^- \leq v^-$.*

It then follows directly from the definition that if G is a Graver basis of $\mathcal{L}$, then $G^+ \leq \mathcal{L}^+_\succ$, and thus, G is a $\succ$-Gröbner basis of $\mathcal{L}$ by Lemma 4.2.3. Interestingly,

it also follows that, since G is defined independently of $\succ$, a Graver basis of $\mathcal{L}$ is a $\succ$-Gröbner basis of $\mathcal{L}$ for every possible term order $\succ$. Importantly, every lattice $\mathcal{L}$ has a finite Graver basis (see for example [50]).

There is a specialised algorithm to compute a Graver basis of a lattice by Hemmecke called the "Project-and-Lift" algorithm ([48, 49]). This algorithm does not require a Markov basis to start with. The fundamental idea behind this algorithm is that we can compute a Graver basis of $\mathcal{L}$ from a Graver basis of $\mathcal{L}^i$ for any $i \in \{1, ..., n\}$. Recall that $\mathcal{L}^i$ is the lattice $\mathcal{L}$ projected onto all components except i. More generally, this implies that we can compute a Graver basis of $\mathcal{L}^{\sigma\text{-}i}$ from a Graver basis of $\mathcal{L}^\sigma$ for any set $\sigma \subseteq \{1, ..., n\}$ and for any $i \in \sigma$. This follows since $\mathcal{L}^{\sigma\text{-}i}$ is a lattice and the lattice $\mathcal{L}^\sigma$ is a projection of $\mathcal{L}^{\sigma\text{-}i}$ onto all components except i. The Project-and-Lift algorithm starts from the trivial case where $\sigma = \{1, ..., n\}$. Then, we iteratively compute a Graver basis of $\mathcal{L}^{\sigma\text{-}i}$ for some $i \in \sigma$, and set $\sigma = \sigma\text{-}i$ and repeat until $\sigma = \emptyset$, in which case, we have a Graver basis of $\mathcal{L}$.

A Graver basis of $\mathcal{L}$ may be a lot larger than a $\succ$-Gröbner basis of $\mathcal{L}$, but for some lattices with special structure, a Gröbner basis is essentially the same as a Graver basis, and also a Markov basis is essentially the same as a Graver basis (see for example [79, 80]). In the situations where a Graver basis is not much larger than a Gröbner basis it may be computationally worthwhile to compute a Graver basis using the Project-and-Lift algorithm of Hemmecke instead of the completion procedure.

We should also point out that it is possible modify the Project-and-Lift algorithm to compute a subset of a Graver basis of $\mathcal{L}$ called a truncated Graver basis of $\mathcal{L}$, which is related to a truncated Gröbner bases of $\mathcal{L}$. Also, it is possible to exploit symmetry when computing a Graver basis (see [51]).

5.4.2 Gröbner walk algorithm

One alternative method is called the *Gröbner walk* algorithm (see [24, 36]). This algorithm computes a $\succ$-Gröbner basis of $\mathcal{L}$ starting from a $\succ'$-Gröbner basis of $\mathcal{L}$ for some other term order $\succ'$. This approach involves computing a sequence of Gröbner bases of $\mathcal{L}$ for every term order in a sequence of terms orders $\succ^1, \succ^2, ..., \succ^k$ where $\succ^1 = \succ'$ and $\succ^k = \succ$. The fundamental ideal behind this approach is that we choose the sequence of term orders in such a way that $\succ^i$ is *almost* equivalent to $\succ^{i+1}$ for $i = 1, ..., k-1$, and consequently, converting from a $\succ^i$-Gröbner bases of $\mathcal{L}$ to a $\succ^{i+1}$-Gröbner basis of $\mathcal{L}$ can be done very quickly using a specialised technique that is based upon the completion procedure. We have implemented this approach, but in practice, we did not find that it was competitive with the completion procedure, because the number of term orders needed in the sequence may be prohibitively large; however, it is certainly possible that this technique is useful under some circumstances, so this approach deserves further investigation.

5.4.3 Knapsack algorithm

Next we discuss a new alternative pseudo-polynomial time algorithm for computing a truncated Gröbner basis of the equality knapsack problem

$$KP_{a,c}(b) = \max\{cx : ax = b, x \geq \mathbf{0}, x \in \mathbb{Z}^n\}$$

where $a \in \mathbb{N}^n$, $c \in \mathbb{N}^n$, and $b \in \mathbb{N}$. Let $\mathcal{L} = \{x \in \mathbb{Z}^n : ax = 0\}$, and let $\nu \in \mathbb{Z}^n$ such that $a\nu = b$. Then, $KP_{a,c}(b) \equiv IP_{\mathcal{L},\succ_c}(\nu)$. This alternative algorithm is fundamentally different from the completion procedure, and the main idea behind the algorithm comes from Lemma 4.3.8. This algorithm is not interesting in practice for reasons we will explain below, but it is theoretically interesting because of the bound on the time complexity it provides for computing Gröbner bases of equality knapsack problems. We have not found this algorithm in the literature, but the concepts behind this algorithm are derived from the algorithm presented by Faugère et al. in [34]. The algorithm we briefly discuss here can compute a ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$ in $\mathcal{O}(n^2b)$ time. Recall that a minimal ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$ has at most nb vectors from the end of Section 4.3.

Let $\mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ be the set of optimal solutions of all the knapsack problems $KP_{a,c}(b')$ for every feasible $b' \in \{0, ..., b\}$ as in Section 4.3. From Lemma 4.3.8, if G is a reduced ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$ and $u \in G$, then u^- is the optimal solution of $IP_{\mathcal{L},\succ_{-c}}(u^+)$ and $u^+ = x^* + \mathbf{e}^i$ for some optimal solution $x^* \in \mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ and some $i \in \{1, ..., n\}$. We can compute the set $\mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ using *dynamic programming* methods in $\mathcal{O}(nb)$ time (see for example [76, 95]). For every optimal solution $x^* \in \mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ and every $i \in \{1, ..., n\}$ such that $a(x^* + \mathbf{e}^i) = b' \leq b$, we check whether $x^* + \mathbf{e}^i$ is the optimal solution of $KP_{a,c}(b')$, or in other words, $x^* + \mathbf{e}^i \in \mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$. We can perform this check in $\mathcal{O}(n)$ time. If $x^* + \mathbf{e}^i \in \mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$, there is nothing to do. If $x^* + \mathbf{e}^i \notin \mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ and $a(x^* + \mathbf{e}^i) \leq b$, then we construct the vector u where $u^+ = x^* + \mathbf{e}^i$ and u^- is the optimal solution of $KP_{a,c}(b')$. We can construct u in $\mathcal{O}(n)$ time. By Lemma 4.3.8, u is a vector in the reduced ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$. At the end of this procedure, we have a ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$.

There are $\mathcal{O}(nb)$ iterations and each iteration takes $\mathcal{O}(n)$ time, so the algorithm is $\mathcal{O}(n^2b)$ time. This algorithm is not interesting in practice since the reason for computing a ν-truncated $\succ_{-c}$-Gröbner basis of $\mathcal{L}$ would be to solve $KP_{a,c}(b)$, but we solve $KP_{a,c}(b)$ during the algorithm when we compute the set $\mathcal{O}_{\mathcal{L},\succ_{-c}}(\nu)$ using dynamic programming.

5.4.4 FLGM algorithm

We next discuss an alternative algorithm for computing a $\succ$-Gröbner basis of a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ for the special case where $\mathcal{L}$ is an n-dimensional lattice. This algorithm uses the same basic idea as the above algorithm for computing Gröbner bases for equality knapsack problems. As above, this algorithm is derived from an algorithm in algorithm geometry by Faugère et al. in [34] (it is named after the authors).[4] The

[4]The original FLGM algorithm computes Gröbner bases of zero-dimensional ideals of polynomial rings.

algorithm is also similar in some ways to an algorithm presented by Hosten and Thomas in [58] for computing a $\succ$-Gröbner basis of a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ where $\mathcal{L}$ is an n-dimensional lattice. The main idea behind the algorithm comes from Lemma 4.2.8. Recall from the end of Section 4.2 that a minimal $\succ$-Gröbner basis of $\mathcal{L}$ has at most $n \det(B)$ vectors where B is a basis of $\mathcal{L}$ and that there are $\det(B)$ possible optimal solutions since there are $\det(B)$ different possible non-empty fibers.

As in Section 4.2, let $\mathcal{O}_{\mathcal{L},\succ}$ be the set of all optimal solutions. We can compute the set of optimal solutions $\mathcal{O}_{\mathcal{L},\succ}$ using a dynamic programming or shortest path type algorithm as suggested by Gomory in [44]. Recall from Lemma 4.2.8 that if G is a reduced $\succ$-Gröbner basis of $\mathcal{L}$ and $u \in G$, then u^- is the optimal solution of $IP_{\mathcal{L},\succ}(u^+)$ and $u^+ = x^* + \mathbf{e}^i$ for some optimal solution $x^* \in \mathcal{O}_{\mathcal{L},\succ}$ and some $i \in \{1, ..., n\}$. We can proceed as per the knapsack problem above. For every optimal solution $x^* \in \mathcal{O}_{\mathcal{L},\succ_c}$, we check whether $x^* + \mathbf{e}^i \in \mathcal{O}_{\mathcal{L},\succ_c}$ for all $i = \{1, ..., n\}$. If $x \notin \mathcal{O}_{\mathcal{L},\succ_c}$, then we construct the vector u where $u^+ = x^* + \mathbf{e}^i$ and u^- is the optimal solution of $\mathcal{F}_{\mathcal{L}}(x^* + \mathbf{e}^i)$. By Lemma 4.2.8, u is a vector in the reduced $\succ$-Gröbner basis of $\mathcal{L}$. At the end of this procedure, we have a $\succ$-Gröbner basis of $\mathcal{L}$.

As for the truncated knapsack algorithm in the previous section, this algorithm is not interesting in practice as an algorithm to solve $IP_{\mathcal{L},\succ}(\nu)$. However, this procedure may be interesting when used as part of other algorithms for computing Gröbner bases and Markov bases.

5.4.5 Hosten and Thomas's algorithm

There is also an alternative algorithm suggested by Hosten and Thomas in [58] for computing a $\succ$-Gröbner basis of a lattice $\mathcal{L}$. This algorithm does not use the completion procedure; instead it uses ideas similar to the idea behind the FLGM algorithm. We will explain the algorithm in only very rough terms.

The fundamental idea behind the algorithm is that we can construct a $\succ$-Gröbner basis of a lattice $\mathcal{L}$ from the $\succ$-Gröbner bases of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ for every $i \in \{1, ..., n\}$ such that $\succ$ is a term order for $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. We are effectively decomposing the problem of computing a $\succ$-Gröbner basis of $\mathcal{L}$. We can apply this idea recursively and construct a $\succ$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ from the $\succ$-Gröbner bases of $\mathcal{F}^{i\cup j}_{\mathcal{L}}(\cdot)$ for every $j \in \{1, ..., n\}$ and $j \neq i$ such that $\succ$ is a term order for $\mathcal{F}^{i\cup j}_{\mathcal{L}}(\cdot)$. Extending this recursive idea to the general case, the algorithm constructs a $\succ$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ from the $\succ$-Gröbner bases of $\mathcal{F}^{\sigma\cup i}_{\mathcal{L}}(\cdot)$ for every $i \in \bar{\sigma}$ such that $\succ$ is a term order for $\mathcal{F}^{\sigma\cup i}_{\mathcal{L}}(\cdot)$. We cannot keep decomposing the problem of computing a Gröbner basis indefinitely. We eventually reach the base case where we must construct a $\succ$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ where $\succ$ is not term order of $\mathcal{F}^{\sigma\cup i}_{\mathcal{L}}(\cdot)$ for any $i \in \bar{\sigma}$. Hosten and Thomas propose a special algorithm for computing a $\succ$-Gröbner basis in a base case situation in [58]. Actually, this special algorithm for the base case is very similar to the FLGM algorithm from the previous section, and we could actually use the FLGM algorithm instead. We are not sure whether this algorithm would be competitive with the completion procedure, because we know of no available implementation of the algorithm.

Chapter 6

Computing Markov bases

In this chapter, we finally present three algorithms to compute a Markov basis of $\mathcal{L}$: the *new* "Project-and-Lift" algorithm, the "Saturation" algorithm (Hosten and Sturmfels [57]), and the "Lift-and-Project" algorithm (Bigatti, LaScala, and Robbiano [12]). Each algorithm produces a Markov basis of $\mathcal{L}$ that is not necessarily minimal, so once a Markov basis of $\mathcal{L}$ is known, a minimal Markov basis of $\mathcal{L}$ can be computed by a single Gröbner basis computation (see [16] for more details). The fundamental idea behind all three algorithms is essentially the same, and the main algorithmic building block of the algorithms is the completion procedure.

We also present a truncated version of the Project-and-Lift algorithm that computes a truncated Markov basis. By definition, any algorithm that computes a non-truncated Markov basis also by definition computes a truncated Markov basis. The advantage of the truncated Project-and-Lift over non-truncated algorithms is that it specifically computes a truncated Markov basis and not necessarily a non-truncated Markov basis, so in situations in which a Markov basis is much larger than a truncated Markov basis, the truncated Project-and-Lift algorithm is potentially much faster.

In this chapter, we present the Project-and-Lift algorithm first because we feel that it is most straight-forward of the three algorithms and thus the best algorithm for presenting the fundamental idea behind all three algorithms. Next, we present a truncated version of the Project-and-Lift algorithm. Then, we present the Saturation algorithm, and we give the first translation of the Saturation algorithm into a combinatorial context. Finally, we present the Lift-and-Project algorithm, which uses ideas from the Saturation algorithm.

We conclude the chapter with a computational comparison of our implementation of the Project-and-Lift algorithm in 4ti2 with implementations of the Saturation algorithm and the Lift-and-Project algorithm implemented in 4ti2 and CoCoA. Our algorithm outperforms the other algorithms in every single instance we have tried.

In this chapter, we only discuss computing Markov bases of $\mathcal{L}$ and not the more general form of Markov bases of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. As we saw in Section 4.2, we can construct a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ from a Markov basis of $\mathcal{F}_{\mathcal{L}^{\sigma}}(\cdot)$. As a result of this, all of the results in this chapter can equally be applied to the more general case. The same applies when using truncation (see Section 3.3).

6.1 Project-and-Lift algorithm

In this section, we present our algorithm for computing Markov basis of lattices, which we call the Project-and-Lift algorithm.

The fundamental idea behind the Project-and-Lift algorithm is that, given a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ for some $i \in \{1, \ldots, n\}$, we can compute a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ (a Markov basis of $\mathcal{L}$). Note that, given $\nu \in \mathbb{Z}^n$, the fiber $\mathcal{F}^i_{\mathcal{L}}(\nu)$ is the relaxation of $\mathcal{F}_{\mathcal{L}}(\nu)$ given by relaxing the non-negativity constraint on x_i; that is, $\mathcal{F}_{\mathcal{L}}(\nu) = \mathcal{F}^i_{\mathcal{L}}(\nu) \cap \{x \in \mathbb{Z}^n : x_i \geq 0\}$. Recall that a set $M \subseteq \mathcal{L}$ is a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ if there is a path from x to y in $\mathcal{G}^i_{\mathcal{L}}(\nu, M)$ for all $x, y \in \mathcal{F}^i_{\mathcal{L}}(\nu)$ for all $\nu \in \mathbb{Z}^n$ and M is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ if there is a path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, M)$ for all $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for all $\nu \in \mathbb{Z}^n$. In some sense, a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ is *almost* a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$. The difference is that a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ guarantees the existence of paths between feasible points of any fiber $\mathcal{F}_{\mathcal{L}}(\nu) \subseteq \mathcal{F}^i_{\mathcal{L}}(\nu)$ that are non-negative on all components except the ith component which may be negative whereas a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ guarantees the existence of paths between feasible points of any fiber $\mathcal{F}_{\mathcal{L}}(\nu)$ that are non-negative on all components. In the Project-and-Lift algorithm, we use the completion procedure to transform paths that are non-negative on all components except the ith component into paths that are non-negative on all components thereby transforming a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ into a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$. Recall that a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ is essentially the same as a Markov basis of $\mathcal{F}_{\mathcal{L}^i}(\cdot)$ (a Markov basis of $\mathcal{L}^i$). So, this fundamental idea equivalently says that we can compute a Markov basis of $\mathcal{L}$ from a Markov basis of $\mathcal{L}^i$. Hence, this approach is called the Project-and-Lift algorithm – we start with a Markov basis of $\mathcal{L}^i$, a projection of $\mathcal{L}$, and lift it to a Markov basis of $\mathcal{L}$.

We can extend this fundamental idea: for some $\sigma \subseteq \{1, \ldots, n\}$, given a Markov basis of $\mathcal{L}^\sigma$, we can compute a Markov basis of $\mathcal{L}^{\sigma\text{-}i}$ for some $i \in \sigma$, or equivalently, given a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$, we can compute a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$. Recall that a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ is essentially the same as a Markov basis of $\mathcal{L}^\sigma$, and similarly, a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ is essentially the same as a Markov basis of $\mathcal{L}^{\sigma\text{-}i}$. Note that the fiber $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ for $\nu \in \mathbb{Z}^n$ is the relaxation of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\nu)$ given by relaxing the non-negativity constraint on x_i; that is, $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\nu) = \mathcal{F}^\sigma_{\mathcal{L}}(\nu) \cap \{x \in \mathbb{Z}^n : x_i \geq 0\}$. A Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ guarantees the existence of paths between feasible points of any fiber $\mathcal{F}_{\mathcal{L}}(\nu)$ that are non-negative on the $\bar{\sigma}$ components whereas a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ guarantees the existence of paths between feasible points of any fiber $\mathcal{F}_{\mathcal{L}}(\nu) \subseteq \mathcal{F}^i_{\mathcal{L}}(\nu)$ that are non-negative on the $\bar{\sigma}$ components as well as the ith component.

The Project-and-Lift algorithm is then as follows. First, we find a $\sigma \subseteq \{1, ..., n\}$ such that it is straight-forward to find a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$. Then, for some $i \in \sigma$, we compute a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ from a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$, and then, we set $\sigma = \sigma\text{-}i$, and repeat. Intuitively, we start out with a set such that there are paths between feasible points of any fiber that are guaranteed to be non-negative on some subset of the components, and then, we iteratively construct paths that are non-negative on one more component until there are paths between any two feasible points of any fiber that are non-negative on all components.

Now, we show how to construct a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ from a Markov basis of

$\mathcal{F}^i_{\mathcal{L}}(\cdot)$. There are two cases to consider here. The first case is where the component i is bounded and the second where i is unbounded. Recall that i is bounded if $IP_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu) := \max\{x_i : x - \nu \in \mathcal{L}, x \geq \mathbf{0}, x \in \mathbb{Z}^n\}$ is bounded for every $\nu \in \mathbb{Z}^n$, otherwise i is unbounded. Note that $\mathbf{e}^i$ is the ith unit vector. First, we consider when i is bounded.

The result that we are working towards is that, when i is bounded, a $\succ_{\text{-}\mathbf{e}^i}$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$. Thus, given a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, we can compute a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ by a Gröbner basis computation. Computing a $\succ_{\text{-}\mathbf{e}^i}$-Gröbner basis means computing a Gröbner basis to solve the lattice program $IP_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu)$ to maximise the ith component. So, conceptually, by computing such a Gröbner basis, we force paths that are non-negative on all components except i to become non-negative on i as well.

First, we introduce the concept of *weak* Gröbner bases. We will describe Markov bases in terms of weak Gröbner bases. Given some vector $c \in \mathbb{R}^n$, a path $(x^1, \ldots, x^k)$ in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, G)$ is a c-reduction path if, for all $j \in \{1, \ldots, k\}$, we have either $cx^1 \geq cx^j$ or $cx^k \geq cx^j$ (i.e. $\max\{cx^1, cx^k\} \geq cx^j$). In some sense, a c-reduction path is a *weak* form of a $\succ_c$-reduction path. A set $G \subseteq \mathcal{L}$ is a **c-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$** if for every pair $x, y \in \mathcal{F}^\sigma_{\mathcal{L}}(\nu)$, there exists a c-reduction path from x to y in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, G)$. Again, in some sense, a c-Gröbner basis of $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, G)$ is a *weak* form of a $\succ_c$-Gröbner basis. As in the case of $\succ_c$-Gröbner bases, we assume here that $IP^\sigma_{\mathcal{L},c}(\nu)$ is bounded for every $\nu \in \mathbb{Z}^n$ and that $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$. A set $G \subseteq \mathcal{L}$ is a **c-Gröbner basis** of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ if G is a c-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ for all $\nu \in \mathbb{Z}^n$ (or a c-Gröbner basis of $\mathcal{L}$ if $\sigma = \emptyset$).

In particular, we are interested in (-$\mathbf{e}^i$)-Gröbner bases of $\mathcal{L}$ where i is bounded because they are actually equivalent to Markov bases of $\mathcal{L}$. Observe that a path $(x^1, \ldots, x^k)$ in $\mathcal{G}_{\mathcal{L}}(\nu, G)$ is an (-$\mathbf{e}^i$)-reduction path if $x^j_i \geq \min\{x^1_i, x^k_i\}$ for all $j \in \{1, ..., k\}$. By definition, an (-$\mathbf{e}^i$)-Gröbner basis of $\mathcal{L}$ is a Markov basis of $\mathcal{L}$ since there is a (-$\mathbf{e}^i$)-reduction path between any two feasible points in a fiber. Conversely, a Markov basis of $\mathcal{L}$ is also an (-$\mathbf{e}^i$)-Gröbner basis of $\mathcal{L}$. We can show this as follows. Given a Markov basis of $\mathcal{L}$, for any $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for any $\nu \in \mathbb{Z}^n$, there must exist a path $(x^1, ..., x^k)$ from $(x - \lambda\mathbf{e}^i)$ to $(y - \lambda\mathbf{e}^i)$ in $\mathcal{G}_{\mathcal{L}}(\nu - \lambda\mathbf{e}^i, G)$ where $\lambda = \min\{x_i, y_i\}$. This path is an (-$\mathbf{e}^i$)-reduction path from $(x - \lambda\mathbf{e}^i)$ to $(y - \lambda\mathbf{e}^i)$ in $\mathcal{G}_{\mathcal{L}}(\nu - \lambda\mathbf{e}^i, G)$ because $x^j_i \geq \min\{x^1_i, x^k_i\} = \min\{x_i - \lambda, y_i - \lambda\} = 0$ for all $j \in \{1, ..., k\}$. Then, by translating such a path by $\lambda\mathbf{e}^i$ giving $(x^1 + \lambda\mathbf{e}^i, ..., x^k + \lambda\mathbf{e}^i)$, we get an (-$\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, G)$. So, we arrive at the following lemma.

Lemma 6.1.1. *Let $\mathcal{L} \subseteq \mathbb{Z}^n$ and $i \in \{1, \ldots, n\}$ where i is bounded. A set $S \subseteq \mathcal{L}$ is a (-$\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ if and only if S is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$.*

The following lemma is fundamental to the Project-and-Lift algorithm; it says that (-$\mathbf{e}^i$)-Gröbner bases of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ are identical to (-$\mathbf{e}^i$)-Gröbner bases of $\mathcal{F}_{\mathcal{L}}(\cdot)$. This is important since (-$\mathbf{e}^i$)-Gröbner bases of $\mathcal{F}_{\mathcal{L}}(\cdot)$ are identical to Markov bases of $\mathcal{F}_{\mathcal{L}}(\cdot)$ from Lemma 6.1.1 above, so the following lemma identifies Markov bases of $\mathcal{F}_{\mathcal{L}}(\cdot)$ with (-$\mathbf{e}^i$)-Gröbner bases of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. Note that if i is bounded in $\mathcal{F}_{\mathcal{L}}(\cdot)$, then $IP^i_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu) := \max\{x_i : x - \nu \in \mathcal{L}, x_{\overline{\imath}} \geq \mathbf{0}, x \in \mathbb{Z}^n\}$ is bounded for all $\nu \in \mathbb{Z}^n$ where $\overline{\imath} = \{1, ..., n\} \setminus i$, and also, by Corollary 2.9.2, i is bounded if and only if there not

exist $x \in \mathcal{L}$ such that $x \geq \mathbf{0}$ and $x_i > 0$, which implies that $\ker(\pi_i) \cap \mathcal{L} = \{\mathbf{0}\}$, and thus, ($\text{-}\mathbf{e}^i$)-Gröbner bases of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ are well-defined.

Lemma 6.1.2. *Let $\mathcal{L} \subseteq \mathbb{Z}^n$, $i \in \{1, \ldots, n\}$ where i is bounded. A set $S \subseteq \mathcal{L}$ is a ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ if and only if S is also a ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$.*

Proof. Let S be a ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, and let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$. We need to show that there is an ($\text{-}\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, S)$. By assumption, there exists an ($\text{-}\mathbf{e}^i$)-reduction path $(x^1, x^2, \ldots, x^k)$ in $\mathcal{G}^i_{\mathcal{L}}(\nu, S)$. So, we have either $x^j_i \geq x_i$ or $x^j_i \geq y_i$ for all $j = 1, ..., k$. Thus, $x^j \in \mathcal{F}_{\mathcal{L}}(\nu)$ for all $j = 1, ..., k$. Therefore, $(x^1, x^2, \ldots, x^k)$ is an ($\text{-}\mathbf{e}^i$)-reduction path in $\mathcal{G}_{\mathcal{L}}(\nu, S)$ as required.

Conversely, assume S is a ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$. Let $x, y \in \mathcal{F}^i_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$. We must show that there exists an ($\text{-}\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}^i_{\mathcal{L}}(\nu, S)$. Let $\lambda = \min\{x_i, y_i\}$. By assumption, there exists a ($\text{-}\mathbf{e}^i$)-reduction path $(x^0, \ldots, x^k)$ from $(x - \lambda\mathbf{e}^i)$ to $(y - \lambda\mathbf{e}^i)$ in $\mathcal{G}_{\mathcal{L}}(\nu - \lambda\mathbf{e}^i, S)$. Note that $x - \lambda\mathbf{e}^i \geq \mathbf{0}$, $y - \lambda\mathbf{e}^i \geq \mathbf{0}$, and $x - \lambda\mathbf{e}^i, y - \lambda\mathbf{e}^i \in \mathcal{F}_{\mathcal{L}}(\nu - \lambda\mathbf{e}^i)$. Therefore, the path $(x = x^0 + \lambda\mathbf{e}^i, \ldots, x^k + \lambda\mathbf{e}^i = y)$ is a ($\text{-}\mathbf{e}^i$)-reduction path in $\mathcal{G}^i_{\mathcal{L}}(\nu, S)$ from x to y as required. □

Given any vector $c \in \mathbb{Z}^n$ and a term order $\succ$ for $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$, recall that for the order $\succ_c$, we have $x \succ_c y$ if $cx < cy$ or $cx = cy$ and $x \succ y$. Also, recall that the order $\succ_c$ is a term order if and only if $IP^\sigma_{\mathcal{L},c}(\nu)$ is bounded for every $\nu \in \mathbb{Z}^n$. Importantly, since $x \succ_c y$ implies $cx \geq cy$, a $\succ_c$-reduction path is also a c-reduction path, and thus, a $\succ_c$-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ is a c-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$.

If i is bounded, then $\succ_{\text{-}\mathbf{e}^i}$ is a term order of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. For brevity, we write $\succ_i$ instead of $\succ_{\text{-}\mathbf{e}^i}$. Thus, a $\succ_i$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ is a ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. So, given a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, using the completion procedure Algorithm 2, we can compute a ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{L}^i$ which is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ by Lemma 6.1.2 and Lemma 6.1.1. More explicitly, given a set M that is a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, the set $M' = \mathcal{CP}^i_{\mathcal{L}}(\succ_i, M)$ is a a Markov basis of $\mathcal{L}$. Furthermore, as we state in Lemma 6.1.3 below, the set M' is actually also a $\succ_i$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. The proof of Lemma 6.1.3 is essentially the same as the proof of Lemma 6.1.2. The following lemma is not strictly necessary in order to describe the algorithm; however, we present it here nevertheless because we feel that is a nice extension of Lemma 6.1.2.

Lemma 6.1.3. *Let $\mathcal{L} \subseteq \mathbb{Z}^n$, $i \in \{1, \ldots, n\}$ where i is bounded. A set $G \subseteq \mathcal{L}$ is a $\succ_i$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ if and only if G is also a $\succ_i$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$.*

We have shown how we can use a Gröbner basis computation to compute a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ from a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ when i is bounded. Next, we describe how this is done when i is unbounded.

If i is unbounded, then finding a Markov basis of $\mathcal{L}$ from a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ is actually more straight-forward than if i is bounded. Crucially, from Corollary 2.9.2, i is unbounded if and only if there exists $u \in \mathcal{L} \cap \mathbb{N}^n$ where $u_i > 0$. Then, given a set $M \subseteq \mathcal{L}$ that is a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, it suffices to add u to M to create a Markov basis of $\mathcal{L}$ (see Lemma 6.1.4 below). We can use linear programming to check whether i is unbounded and also to find a $u \in \mathcal{L} \cap \mathbb{N}^n$ where $u_i > 0$ by Lemma 2.9.1.

Lemma 6.1.4. *Let $i \in \{1, \ldots, n\}$ and $u \in \mathcal{L} \cap \mathbb{N}^n$ where $u_i > 0$. If $M \subseteq \mathcal{L}$ is a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, then $M \cup \{u\}$ is a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$.*

Proof. Let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$. Since M is a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, there exists a path from x to y in $\mathcal{G}^i_{\mathcal{L}}(\nu,)$. We can convert this path into a path in $\mathcal{G}_{\mathcal{L}}(\nu, M \cup \{u\})$ by adding u to the start of the path as many times as necessary and subtracting u from the end of the path the same number of times. This works since $u_i > 0$ and $u \geq 0$. □

We have shown how we can use the a Gröbner basis computation to compute a Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ from a Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ when i is bounded or unbounded. Assuming that $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$, we can extend these results to compute a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ from a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ when $i \in \sigma$ is bounded or unbounded in $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. Explicitly, given set $M \subseteq \mathcal{L}$ that is a Markov basis $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, if i is bounded, then $M' = \mathcal{CP}^{\sigma}_{\mathcal{L}}(\succ_i, M)$ is a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$. Note that we require the $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$, otherwise $\succ_i$ is not a term order of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. Also, if i is unbounded, then there exists a vector $u \in \mathcal{L}$ such that $u_{\bar{\sigma}} \geq \mathbf{0}$ and $u_i > 0$, from Corollary 2.9.2, and then, $M \cup \{u\}$ is a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$.

We can now present our Project-and-Lift algorithm (Algorithm 4).

Algorithm 4 Algorithm: Project-and-Lift

Input: a lattice $\mathcal{L}$.
Output: a Markov basis M of $\mathcal{L}$.

```
Find a set σ ⊆ {1, ..., n} such that ker(π_σ) ∩ L = {0}.
Compute a set M ⊆ L that is a Markov basis of F^σ_L(·).
while σ ≠ ∅ do
    Select i ∈ σ
    if i is bounded then
        M := CP^σ_L(≻_i, M)
    else
        Find u ∈ L such that u_σ̄ ≥ 0 and u_i > 0.
        M := M ∪ {u}
    end if
    σ := σ-i
end while
return M.
```

Lemma 6.1.5. *Algorithm 4 terminates and satisfies its specifications.*

Proof. Algorithm 4 terminates, since Algorithm 2, which computes $\mathcal{CP}^{\sigma}_{\mathcal{L}}(\succ_i, M)$, always terminates.

We claim that for each iteration of the algorithm, M is a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ and $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$; therefore, at termination, M is a Markov basis of $\mathcal{L}$. This is true for the first iteration, so we assume it is true for the current iteration.

If $\sigma = \emptyset$, then there is nothing left to do, so assume otherwise. Since by assumption, $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$, we must have $\ker(\pi_{\sigma\text{-}i}) \cap \mathcal{L} = \{\mathbf{0}\}$. Let $i \in \sigma$, and $\sigma' := \sigma\text{-}i$.

If i is bounded, then let $M := \mathcal{CP}^{\sigma}_{\mathcal{L}}(\succ_i, M)$. Then, M is now a $\succ_i$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, and then by Lemma 6.1.2, M is a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$. Otherwise, let $M := M \cup \{u\}$ where $u_{\bar{\sigma}} \geq \mathbf{0}$ and $u_i > 0$, and by Lemma 6.1.4, M is a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$. Thus, the claim is true for the next iteration. □

Initially in the Project-and-Lift algorithm, we need to find a set $\sigma \subseteq \{1, \ldots, n\}$ such that $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$, and then, we need to compute a Markov basis for $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. This is actually quite straight-forward and can be done in polynomial time. Let B be a basis for the lattice $\mathcal{L}$ ($\mathcal{L}$ is spanned by the rows of the matrix B). Let $k := \text{rank}(B)$. Any k linearly independent columns of B then suffice to give a set $\bar{\sigma}$ such that there is a one-to-one correspondence between vectors in $\mathcal{L}^{\sigma}$ and vectors in $\mathcal{L}$; that is, $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$. Such a set σ can be found via Gaussian elimination. Let $S = \pi_\sigma(B)$; then, S spans $\mathcal{L}^{\sigma}$, and $S \in \mathbb{Z}^{k \times k}$ since $|\bar{\sigma}| = k$. Let S' be an upper triangle matrix with positive diagonal entries and non-positive entries elsewhere such that (the rows of) S' span $\mathcal{L}^{\sigma}$ (S' is in Upper Hermite Normal Form). We can always construct such a matrix S' from S in polynomial time using the *Hermite Normal Form* (HNF) algorithm (see for example [71]). As required, S' is a Markov basis of $\mathcal{L}^{\sigma}$ as we saw in Example 3.2.5.

The next example shows an example computation using the Project-and-Lift algorithm.

Example 6.1.6. *Consider again the projected lattice $\mathcal{L}$ from Example 3.1.6 that is generated by the vectors* $(2, \text{-}2, \text{-}2, 1)$ *and* $(3, 1, \text{-}1, \text{-}1)$. *Recall that the set*

$$M = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (5, \text{-}1, \text{-}3, 0), (1, 3, 1, \text{-}2)\}$$

is a Markov basis of $\mathcal{L}$. We will compute this set using the Project-and-Lift algorithm.

The set $B = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1)\}$ is a basis of $\mathcal{L}$. Let $\sigma = \{3, 4\}$. Then, $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$. Let $S = \pi_\sigma(B) = \{(2, \text{-}2), (3, 1)\}$. The set S is a Markov basis of $\mathcal{L}^{\sigma}$ by Lemma 3.2.3 because there is a strictly positive vector in S. This implies that $B = \pi^{\text{-}1}_{\sigma}(S)$ is a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. As suggested above, we also could have found a Markov basis of $\mathcal{L}^{\sigma}$ by computing the UNHF of S, which is $S' := \{(1, \text{-}5), (0, 8)\}$, which is also a Markov basis of $\mathcal{L}^{\sigma}$ by Lemma 3.2.4. Here, $B' = \pi^{\text{-}1}_{\sigma}(S') = \{(1, \text{-}5, \text{-}3, 3), (0, 8, 4, \text{-}5)\}$. Then, the set B' is also a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. It does not matter which Markov basis we use for the algorithm. We choose to use B'.

Set $M = \{(1, \text{-}5, \text{-}3, 3), (0, 8, 4, \text{-}5)\}$.

(i). Select $i := 3$. Then, i is unbounded since $u = (0, 8, 4, \text{-}5) \in \mathcal{L}$ and $u_{\bar{\sigma}} \geq \mathbf{0}$ and $u_i > 0$. Thus the set $M = \{(1, \text{-}5, \text{-}3, 3), (0, 8, 4, \text{-}5)\}$ is a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$. Set $\sigma = \{4\}$.

(ii). Select $i := 4$. Then, i is bounded. The set $M = \mathcal{CP}^{\sigma}_{\mathcal{L}}(\succ_i, M) =$

$$\{(\text{-}5, 1, 3, 0), (\text{-}2, 2, 2, \text{-}1), (1, 3, 1, \text{-}2), (3, 1, \text{-}1, \text{-}1), (8, 0, \text{-}4, \text{-}1)\}.$$

is a minimal $\succ_i$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. Note that we have removed unnecessary vectors from M to arrive at a minimal Gröbner basis after running the completion procedure. So, M is a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot) = \mathcal{F}_{\mathcal{L}}(\cdot)$.

The set $M = \{(-5, 1, 3, 0), (-2, 2, 2, -1), (1, 3, 1, -2), (3, 1, -1, -1), (8, 0, -4, -1)\}$ *is thus a Markov basis of* $\mathcal{L}$*. Note that we computed one more vector than necessary: the vector* $(8, 0, -4, -1)$ *is not needed in a Markov basis of* $\mathcal{L}$*. See [16] for an algorithm to compute a minimal Markov basis.*

Lastly, it is worth mentioning that the Project-and-Lift algorithm presented in this section for computing Markov basis of lattices is very similar to the Project-and-Lift algorithm for computing Graver bases of lattices by Hemmecke in [48, 49]. This Project-and-Lift algorithm here computes Markov basis of $\mathcal{L}^\sigma$ incrementally for $\sigma \subseteq \{1, ..., n\}$, and the Project-and-Lift of Hemmecke compute Graver basis of $\mathcal{L}^\sigma$ incrementally for $\sigma \subseteq \{1, ..., n\}$. For some lattices with special structure, a Markov basis of $\mathcal{L}^\sigma$ is effectively the same as a Graver basis of $\mathcal{L}^\sigma$ for any $\sigma \subseteq \{1, ..., n\}$, in which case, the two algorithms are fundamentally the same under some assumptions.

6.2 Truncated Project-and-Lift algorithm

In this section, we give a truncated Project-and-Lift algorithm for computing truncated Markov bases of lattices. The algorithm is essentially the same as the Project-and-Lift algorithm for computing Markov bases of lattices as presented in the previous section except that we use the truncated completion procedure instead, so we now show that the results presented in the previous section are still valid when used with truncation.

Essentially the same fundamental idea is behind the truncated Project-and-Lift algorithm as the Project-and-Lift algorithm: for some $\nu \in \mathbb{Z}^n$, given a ν-truncated Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ for some $i \in \{1, \ldots, n\}$, we can compute a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ (a ν-truncated Markov basis of $\mathcal{L}$). Equivalently, we can compute a ν-truncated Markov basis of $\mathcal{L}$ from a Markov basis of $\nu_{\overline{i}}$-truncated Markov basis of $\mathcal{L}^i$ where $\nu_{\overline{i}}$ is the projection of ν onto all components except the ith component. Extending this, we can compute a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ from a ν-truncated Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$. The truncated Project-and-Lift algorithm is then as follows: given an initial ν-truncated Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ for some $\sigma \subseteq \{1, \ldots, n\}$, we compute a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ for some $i \in \sigma$, and then, we set $\sigma = \sigma\text{-}i$, and repeat until we attain a ν-truncated Markov basis of $\mathcal{L}$.

Now, we show how to construct a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ from a ν-truncated Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. As in the non-truncated situation, there are two cases to consider: i is bounded and i is unbounded. We consider the case where i is bounded first.

We now derive an analogous result to Lemma 6.1.1 for truncated Gröbner bases of lattices, which shows that we can describe truncated Markov bases in terms of truncated weak Gröbner bases. The proof is essentially the same as the proof of Lemma 6.1.1. First, we must extend the notion of weak Gröbner bases to truncated weak Gröbner bases. Given $\mathcal{L} \subseteq \mathbb{Z}^n$, $c \in \mathbb{R}^n$, and let $\nu \in \mathbb{Z}^n$, a set $G \subseteq \mathcal{L}$ is a **ν-truncated c-Gröbner basis of** $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ if G is a c-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu')$ for every $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$.

Lemma 6.2.1. *Let $\mathcal{L} \subseteq \mathbb{Z}^n$, $i \in \{1, \ldots, n\}$ where i is bounded, and let $\nu \in \mathbb{Z}^n$. A set $S \subseteq \mathcal{L}$ is a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}_\mathcal{L}(\cdot)$ if and only if S is a ν-truncated Markov basis of $\mathcal{F}_\mathcal{L}(\cdot)$.*

Proof. By definition, a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}_\mathcal{L}(\cdot)$ is a ν-truncated Markov basis of $\mathcal{F}_\mathcal{L}(\cdot)$ since there is a ($\text{-}\mathbf{e}^i$)-reduction path between any two feasible points in any fiber in $\mathcal{B}_\mathcal{L}(\nu)$.

Conversely, assume S is a ν-truncated Markov basis of $\mathcal{F}_\mathcal{L}(\cdot)$. Let $x, y \in \mathcal{F}_\mathcal{L}(\nu')$ for some $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$, and let $\lambda = \min\{x_i, y_i\}$. First, note that $(\nu' - \lambda\mathbf{e}^i) \in \mathcal{B}_\mathcal{L}(\nu)$ because $\mathcal{F}_\mathcal{L}(\nu' - \lambda\mathbf{e}^i) \neq \emptyset$ and $\mathcal{F}_\mathcal{L}(\nu - (\nu' - \lambda\mathbf{e}^i)) \neq \emptyset$ since $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset$. There must exist a path $(x^1, ..., x^k)$ from $(x - \lambda\mathbf{e}^i)$ to $(y - \lambda\mathbf{e}^i)$ in $\mathcal{G}_\mathcal{L}(\nu' - \lambda\mathbf{e}^i, G)$ since $(\nu' - \lambda\mathbf{e}^i) \in \mathcal{B}_\mathcal{L}(\nu)$. This path is an ($\text{-}\mathbf{e}^i$)-reduction path from $(x - \lambda\mathbf{e}^i)$ to $(y - \lambda\mathbf{e}^i)$ in $\mathcal{G}_\mathcal{L}(\nu' - \lambda\mathbf{e}^i, G)$, so by translating such a path by $\lambda\mathbf{e}^i$, we get an ($\text{-}\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}_\mathcal{L}(\nu', G)$. □

We can also derive an analogous result to Lemma 6.1.2 for truncated Gröbner bases of lattices. This result is fundamental to the truncated Project-and-Lift algorithm. The proof proceeds in much the same fashion as the proof of Lemma 6.1.2. First, note that $\mathcal{B}_\mathcal{L}(\nu) \subseteq \mathcal{B}^i_\mathcal{L}(\nu)$ since $\mathcal{F}_\mathcal{L}(\nu') \neq \emptyset$ and $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset$ implies that $\mathcal{F}^i_\mathcal{L}(\nu') \neq \emptyset$ and $\mathcal{F}^i_\mathcal{L}(\nu - \nu') \neq \emptyset$.

Lemma 6.2.2. *Let $\mathcal{L} \subseteq \mathbb{Z}^n$, $i \in \{1, \ldots, n\}$ where i is bounded, and let $\nu \in \mathbb{Z}^n$. If a set $M \subseteq \mathcal{L}$ is a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_\mathcal{L}(\cdot)$, then M is also a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}_\mathcal{L}(\cdot)$.*

Proof. Let M be a ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_\mathcal{L}(\cdot)$, and let $x, y \in \mathcal{F}_\mathcal{L}(\nu')$ for some $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$. We need to show that there is an ($\text{-}\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}_\mathcal{L}(\nu, S)$. Since $\nu' \in \mathcal{B}^i_\mathcal{L}(\nu)$, by assumption, there exists an ($\text{-}\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}^i_\mathcal{L}(\nu, S)$, which is also a ($\text{-}\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}_\mathcal{L}(\nu, S)$ as required. □

In contrast with the non-truncated case, the converse of Lemma 6.2.2 is not necessarily true: if S is a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}_\mathcal{L}(\cdot)$, then S is *not necessarily* a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_\mathcal{L}(\cdot)$. The reason being that $\mathcal{B}_\mathcal{L}(\nu) \subseteq \mathcal{B}^i_\mathcal{L}(\nu)$ and we may have $\mathcal{B}_\mathcal{L}(\nu) \subsetneq \mathcal{B}^i_\mathcal{L}(\nu)$, in which case, there may be a vector $u \in \mathcal{L}$ such that there exists $x, y \in \mathcal{F}^i_\mathcal{L}(\nu')$ for some $\nu' \in \mathcal{B}^i_\mathcal{L}(\nu)$ where $x - y = u$ but there does not exist $x, y \in \mathcal{F}^i_\mathcal{L}(\nu'')$ for any $\nu'' \in \mathcal{B}_\mathcal{L}(\nu)$ where $x - y = u$. However, if $\mathcal{B}_\mathcal{L}(\nu) = \mathcal{B}^i_\mathcal{L}(\nu)$, then the converse is true.

If i is bounded, then the order $\succ_i$ is a term order for $\mathcal{F}^i_\mathcal{L}(\cdot)$, so given a set $S \subseteq \mathcal{L}$ that is a ν-truncated Markov basis of $\mathcal{F}^i_\mathcal{L}(\cdot)$, we can compute a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_\mathcal{L}(\cdot)$ using Algorithm 3. In other words, the set $S' = \mathcal{CP}^i_{\mathcal{L},\nu}(\succ_i, S)$ is a ν-truncated ($\text{-}\mathbf{e}^i$)-Gröbner basis of $\mathcal{F}^i_\mathcal{L}(\cdot)$, and by Lemma 6.2.2, the set S' is a ν-truncated Markov basis of $\mathcal{L}$. Moreover, the set S' is also a ν-truncated $\succ_i$-Gröbner basis of $\mathcal{F}_\mathcal{L}(\cdot)$. The proof proceeds as in Lemma 6.2.2. This result is not strictly needed to describe the algorithm, but we find that it is a nice extension of Lemma 6.2.2.

Lemma 6.2.3. *Let $\mathcal{L} \subseteq \mathbb{Z}^n$, $\nu \in \mathbb{Z}^n$, and $i \in \{1, \ldots, n\}$ where i is bounded. If $G \subseteq \mathcal{L}$ is a ν-truncated $\succ_i$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, then G is also a ν-truncated $\succ_i$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$.*

As in the non-truncated case, if i is unbounded, then computing a ν-truncated Markov basis of $\mathcal{L}$ from a ν-truncated Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$ is actually more straightforward than otherwise.

Lemma 6.2.4. *Let $\mathcal{L} \subseteq \mathbb{Z}^n$, $i \in \{1, \ldots, n\}$, $\nu \in \mathbb{Z}^n$, and $u \in \mathcal{L} \cap \mathbb{N}^n$ where $u_i > 0$. If $S \subseteq \mathcal{L}$ is a ν-truncated Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, then $S \cup \{u\}$ is a ν-truncated Markov basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$.*

The proof of Lemma 6.2.4 proceeds as per the proof of Lemma 6.1.4.

We can now present the truncated Project-and-Lift algorithm (Algorithm 5).

Algorithm 5 Truncated Project-and-Lift algorithm

Input: a lattice $\mathcal{L}$ and a vector $\nu \in \mathbb{Z}^n$.
Output: a ν-truncated Markov basis M of $\mathcal{L}$

```
Find a set σ ⊆ {1, ..., n} such that ker(π_σ) ∩ L = {0}.
Compute a set M ⊆ L that is a ν-truncated Markov basis of F^σ_L(·).
while σ ≠ ∅ do
    Select i ∈ σ
    if i is bounded then
        M := CP^σ_{L,ν}(≻_i, M)
    else
        Compute u ∈ L such that u_σ̄ ≥ 0 and u_i > 0
        M := M ∪ {u}
    end if
    σ := σ-i
    M := {u ∈ M : u^+_σ̄ ∈ B_{L^σ}(ν_σ̄)}
end while
return M.
```

Algorithm 5 is the same as Algorithm 4 except that we use the truncated completion procedure and there is an extra step at the end: $M := \{u \in M : u^+_{\bar{\sigma}} \in \mathcal{B}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})\}$. A vector $u \in \mathcal{L}$ fits inside the feasible region $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ if and only if $u_{\bar{\sigma}}$ fits inside $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$, and by Lemma 3.3.2, $u_{\bar{\sigma}}$ fits inside $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$ if and only if $u^+_{\bar{\sigma}} \in \mathcal{B}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$. Thus, this extra step just truncates vectors that are too long and the proof of correctness and termination of Algorithm 5 is essentially the same as for Algorithm 4.

Initially in the truncated Project-and-Lift algorithm, we need to find a set $\sigma \subseteq \{1, \ldots, n\}$ such that $\ker(\pi_\sigma) \cap \mathcal{L}^\sigma = \{\mathbf{0}\}$, and then, we need to compute a ν-truncated Markov basis for $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$. We proceed in exactly the same way as per the non-truncated case in the previous section.

The next example shows an example computation using the truncated Project-and-Lift algorithm.

Example 6.2.5. *Consider again the lattice $\mathcal{L}$ from Example 3.1.6 that is generated by the vectors* $(2, \text{-}2, \text{-}2, 1)$ *and* $(3, 1, \text{-}1, \text{-}1)$*. Recall from Example 3.3.3 that the set*

$$M = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (1, 3, 1, \text{-}2)\}$$

is a ν-truncated Markov basis of $\mathcal{L}$ where $\nu = (\text{-}6, 4, 6, 1)$. We will compute a ν-truncated Markov basis of $\mathcal{L}$ using the truncated Project-and-Lift algorithm. Recall from Example 6.1.6 that the set $B = \{(1, \text{-}5, \text{-}3, 3), (0, 8, 4, \text{-}5)\}$ is a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ where $\sigma = \{3, 4\}$; thus, B is also a ν-truncated Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$.

Set $M = \{(1, \text{-}5, \text{-}3, 3), (0, 8, 4, \text{-}5)\}$.

(i). Select $i := 3$. Then, i is unbounded since $u = (0, 8, 4, \text{-}5) \in \mathcal{L}$ and $u_{\bar{\sigma}} \geq \mathbf{0}$ and $u_i > 0$. Thus the set $M = \{(1, \text{-}5, \text{-}3, 3), (0, 8, 4, \text{-}5)\}$ is a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$. Set $\sigma = \{4\}$.

(ii). Select $i := 4$. Then, i is bounded. The set

$$M = \mathcal{CP}^{\sigma}_{\mathcal{L},\nu}(\succ_i, M) = \{(\text{-}2, 2, 2, \text{-}1), (1, 3, 1, \text{-}2), (3, 1, \text{-}1, \text{-}1)\}.$$

is a minimal ν-truncated $\succ_i$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$. So, M is a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$.

The set $M = \{(\text{-}2, 2, 2, \text{-}1), (1, 3, 1, \text{-}2), (3, 1, \text{-}1, \text{-}1)\}$ is thus a ν-truncated Markov basis of $\mathcal{L}$.

In the next example, we show the computational benefits of computing a truncated Markov basis as opposed to the full Markov basis.

Example 6.2.6. *Let $\mathcal{L} = \mathcal{L}_A$ for the matrix A given in Example 5.2.4. The size of a minimal Markov basis of $\mathcal{L}$ is* 10868*. It takes* 36.39 *seconds to compute.*

Let $\nu = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$. The size of a minimal ν-truncated Markov basis of $\mathcal{L}$ is 0. In Table 6.1, we list the times taken to compute a ν-truncated Markov basis using the three different criteria for truncation. The first column lists which truncation criteria was used (the first row is the case without using truncation). The next five columns list the sizes of the intermediate Gröbner basis computations at the end of each iteration of the truncated Project-and-Lift algorithm (the column heading is the iteration number). Note how the intermediate sizes of the truncated computations remain much smaller than the final size of a full Markov basis of $\mathcal{L}$. The last column lists the time taken.

In Table 6.2, we list the times taken to compute a minimal truncated Markov basis for the same five different ν as in Example 5.2.4. The first column lists which ν was used. In the next columns, we list the size of the computed set and the time taken for each of the three possible ways to check whether $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$. Not only have we reduced the time to compute a Markov basis, we also have reduced the time to compute a truncated Gröbner basis by using a truncated Markov basis instead of a full Markov basis. Table 6.3 lists the times for computing a truncated Gröbner basis from a truncated Markov basis using the same cost vector $c = (3, 15, 1, 5, 2, 17, 16, 16, 15, 9, 7, 11, 13)$ as before.

Truncation	1	2	3	4	5	Time
none	545	1822	3681	12573	10868	36.39s
$a \in \mathcal{S}^* \cap \mathbb{R}^n_+$	545	977	1302	1846	564	1.20s
$\mathcal{F}_{\mathcal{S}}(\nu - \nu'')$	545	977	878	697	194	2.16s
$\mathcal{F}_{\mathcal{L}}(\nu - \nu'')$	545	6	3	4	0	20.73s

Table 6.1: Comparison of different truncation approaches.

ν	$a \in \mathcal{S}^* \cap \mathbb{R}^n_+$		$\mathcal{F}_{\mathcal{S}}(\nu - \nu'')$		$\mathcal{F}_{\mathcal{L}}(\nu - \nu'')$	
ν^1	4	0.76	1	0.90	0	3.33
ν^2	158	16.70	36	21.28	0	> 3600
ν^3	546	1.20	194	2.16	0	20.73
ν^4	7381	7.61	3734	11.47	146	> 3600
ν^5	10814	39.04	10761	44.07	10739	> 3600

Table 6.2: Timings for computing truncated Markov bases of different fibers.

Observe that in the previous example, the size of a truncated Markov basis for the fiber where $\nu = (1, 0, 1, 0, 3, 0, 1, 5, 0, 1, 0, 9, 0)$ was very small, but it took much longer to compute than a much larger truncated Markov basis of other fibers. This anomaly can be attributed to the order in which the variables are chosen during the Project-and-Lift algorithm. If we reorder the variables so that the zero components in $\nu = (1, 0, 1, 0, 3, 0, 1, 5, 0, 1, 0, 9, 0)$ are chosen first, then the algorithm computes a truncated Markov basis much faster. Hence, the efficiency of the algorithm is sensitive to the order in which the variables are chosen; therefore, future experimentation is needed to determine a heuristic for choosing a *good* variable order.

6.3 Saturation algorithm

In this section, we describe the Saturation algorithm. The description of the Saturation algorithm in this section is the first translation of the Saturation algorithm from algebraic geometry into a combinatorial setting. The underlying concept behind the algorithm is called *saturation*. Saturation is concerned with the connectivity of the fiber graphs of a set T in relation to the connectivity of the fiber graphs of another set S.

Definition 6.3.1. *Let $\sigma \subseteq \{1, \ldots, n\}$, and let $S, T \subseteq \mathcal{L}$. The set T is σ-**saturated on** S if for all $\nu \in \mathbb{Z}^n$ and for all $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where x and y are connected in $\mathcal{G}^{\sigma}_{\mathcal{L}}(\nu, S)$, the points x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, T)$.*

In other words, the definition of saturation says that T is σ-saturated on S if whenever two feasible points $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$ are connected by a path using the vectors in S that is non-negative on the $\bar{\sigma}$ components, then the same two feasible points x and y are connected by a path using the vectors in T that is non-negative on all components.

ν	$a \in \mathcal{S}^* \cap \mathbb{R}^n_+$		$\mathcal{F}_{\mathcal{S}}(\nu - \nu'')$		$\mathcal{F}_{\mathcal{L}}(\nu - \nu'')$	
ν^1	4	0.00	1	0.00	0	0.00
ν^2	167	0.00	36	0.19	0	0.00
ν^3	844	0.05	201	0.11	0	0.00
ν^4	11768	5.92	5028	4.07	158	5.98
ν^5	24729	109.57	24334	107.18	24284	> 3600

Table 6.3: Timings for computing a truncated Gröbner basis from a truncated Markov basis.

The following lemma is the most important result concerning saturation.

Lemma 6.3.2. *Let $S, T \subseteq \mathcal{L}$ where S spans $\mathcal{L}$. If T is $\{1, ..., n\}$-saturated on S, then T is a Markov basis of $\mathcal{L}$.*

Proof. First, recall that any set S that spans $\mathcal{L}$ is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ where $\sigma = \{1, ..., n\}$ (no non-negativity requirements). Let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$. Since S is a Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$, x and y are connected in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu, S)$, which implies that x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, T)$ since T is σ-saturated on S. Hence, T is a Markov basis of $\mathcal{L}$. □

The fundamental idea behind the Saturation algorithm is given $S, T \subseteq \mathcal{L}$ where T is σ-saturated on S for some $\sigma \subseteq \{1, \ldots, n\}$, we can compute a set T' that is a $(\sigma \cup i)$-saturated on S for any $i \in \bar{\sigma}$. Note that, by definition, any set of vectors is $\emptyset$-saturated on itself. Therefore, given a set $S \subseteq \mathcal{L}$ that spans $\mathcal{L}$, starting from a set $T = S$, which is $\emptyset$-saturated on S, if we do this repeatedly for each $i \in \{1, \ldots, n\}$, we arrive at a set $T' \subseteq \mathcal{L}$ that is $\{1, \ldots, n\}$-saturated on S and, therefore, a Markov basis of $\mathcal{L}$.

The following lemma is essential to the saturation algorithm. The implication of Lemma 6.3.3 below is that if we wish to compute a set T' that is $(\sigma \cup i)$-saturated on S given a set T that is σ-saturated on S, then we just need to ensure that T' is i-saturated on T.

Lemma 6.3.3. *Let $\sigma, \tau \subseteq \{1, \ldots, n\}$ and $S, T, U \subseteq \mathcal{L}$. If U is σ-saturated on S, and T is τ-saturated on U, then T is $(\sigma \cup \tau)$-saturated on S.*

Proof. Let $\nu \in \mathbb{Z}^n$, and $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where x and y are connected in $\mathcal{G}^{(\sigma\cup\tau)}_{\mathcal{L}}(\nu, S)$; that is, using the vectors in S, there exists a path from x to y that is non-negative on the $\bar{\sigma} \cap \bar{\tau}$ variables. Let $(x^1, x^2, ..., x^k)$ be a path from x to y in $\mathcal{G}^{(\sigma\cup\tau)}_{\mathcal{L}}(\nu, S)$. Let $\gamma \in \mathbb{N}^n$ such that $\gamma_{\bar{\tau}} = \mathbf{0}$ and $\gamma_\tau + x^i_\tau \geq \mathbf{0}$ for $i = 1, ..., k$. Such a γ always exists. Consider the path $(x^1 + \gamma, x^2 + \gamma, ..., x^k + \gamma)$ from $x + \gamma$ to $y + \gamma$. By construction, this path is now non-negative on the τ components, and the $\bar{\tau}$ components are unaltered. Thus, this path is a path from $x + \gamma$ to $y + \gamma$ in $\mathcal{G}^\sigma_{\mathcal{L}}(\nu + \gamma, S)$. Since U is σ-saturated on S, this implies that there is a path from $x + \gamma$ to $y + \gamma$ in $\mathcal{G}_{\mathcal{L}}(\nu + \gamma, U)$. Let $(\bar{x}^1, \bar{x}^2, ..., \bar{x}^k)$ be such a path from $x + \gamma$ to $y + \gamma$. Then, $(\bar{x}^1 - \gamma, \bar{x}^2 - \gamma, ..., \bar{x}^k - \gamma)$ is a path from x to y in $\mathcal{G}^\tau_{\mathcal{L}}(\nu, U)$ since $\gamma_{\bar{\tau}} = \mathbf{0}$. Then, since T is τ-saturated on U, x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, T)$ as required. □

We now focus on computing a set T that is i-saturated on S. There are two situations to consider: i is bounded and i is unbounded. We first consider when i is bounded. Recall that $\ker(\pi_i) \cap \mathcal{L} = \{\mathbf{0}\}$ implies that i is bounded by Corollary 2.9.2.

Lemma 6.3.4. *Let $S, T \subseteq \mathcal{L}$ and $i \in \{1, ..., n\}$ where $\ker(\pi_i) \cap \mathcal{L} = \{\mathbf{0}\}$. The set T is i-saturated on S if and only if, for all $\nu \in \mathbb{Z}^n$ and for all $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, S)$, there exists a ($\text{-}\mathbf{e}^i$)-reduction path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, T)$.*

Proof. Assume T is i-saturated on S. Let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$ where x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, S)$. Let $\lambda = \min\{x_i, y_i\}$. Let $(x^1, x^2, ..., x^k)$ be a path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, S)$. Then, $(x^1 - \lambda\mathbf{e}^i, x^2 - \lambda\mathbf{e}^i, ..., x^k - \lambda\mathbf{e}^i)$ is a path from $(x - \lambda\mathbf{e}^i)$ to $(y - \lambda\mathbf{e}^i)$ in $\mathcal{G}^i_{\mathcal{L}}(\nu - \lambda\mathbf{e}^i, S)$. Since T is i-saturated on S, $(x - \lambda\mathbf{e}^i)$ and $(y - \lambda\mathbf{e}^i)$ are connected in $\mathcal{G}_{\mathcal{L}}(\nu - \lambda\mathbf{e}^i, S)$. Let $(\bar{x}^1, \bar{x}^2, ..., \bar{x}^k)$ be a path from $(x - \lambda\mathbf{e}^i)$ to $(y - \lambda\mathbf{e}^i)$ in $\mathcal{G}_{\mathcal{L}}(\nu - \lambda\mathbf{e}^i, S)$. Note that, by construction, either $\bar{x}^1_i = (x - \lambda\mathbf{e}^i)_i = 0$ or $\bar{x}^k_i = (y - \lambda\mathbf{e}^i)_i = 0$, so $\bar{x}^j_i \geq \bar{x}^1_i$ or $\bar{x}^j_i \geq \bar{x}^k_i$ for all $j = 1, ..., k$. Then, $(\bar{x}^1 + \lambda\mathbf{e}^i, \bar{x}^2 + \lambda\mathbf{e}^i, ..., \bar{x}^k + \lambda\mathbf{e}^i)$ is a path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, S)$ where either $(\bar{x}^j + \lambda\mathbf{e}^i)_i \geq (\bar{x}^1 + \lambda\mathbf{e}^i)_i = x_i$ or $(\bar{x}^j + \lambda\mathbf{e}^i)_i \geq (\bar{x}^k + \lambda\mathbf{e}^i)_i = y_i$. Therefore, $(\bar{x}^1 + \lambda\mathbf{e}^i, \bar{x}^2 + \lambda\mathbf{e}^i, ..., \bar{x}^k + \lambda\mathbf{e}^i)$ is an ($\text{-}\mathbf{e}^i$)-reduction path from x to y as required.

Conversely, let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ such that x and y are connected in $\mathcal{G}^i_{\mathcal{L}}(\nu, S)$. Let $(x^1, x^2, ..., x^k)$ be a path from x to y in $\mathcal{G}^i_{\mathcal{L}}(\nu, S)$. Let $\lambda \in \mathbb{N}$ such that $x^j_i + \lambda \geq \mathbf{0}$ for all $j = 1, ..., k$. Then, $(x^1 + \lambda\mathbf{e}^i, x^2 + \lambda\mathbf{e}^i, ..., x^k + \lambda\mathbf{e}^i)$ is a path from $(x + \lambda\mathbf{e}^i)$ to $(y + \lambda\mathbf{e}^i)$ in $\mathcal{G}_{\mathcal{L}}(\nu + \lambda\mathbf{e}^i, S)$; the path is now non-negative on the ith component. By assumption, there exists a ($\text{-}\mathbf{e}^i$)-reduction path from $(x + \lambda\mathbf{e}^i)$ to $(y + \lambda\mathbf{e}^i)$ in $\mathcal{G}_{\mathcal{L}}(\nu + \lambda\mathbf{e}^i, T)$. Let $(\bar{x}^1, \bar{x}^2, ..., \bar{x}^k)$ be such a path. Then, either $\bar{x}^j_i \geq \bar{x}^1_i$ or $\bar{x}^j_i \geq \bar{x}^k_i$ for $j = 1, ..., k$, so $\bar{x}^j - \lambda\mathbf{e}^i \geq \mathbf{0}$ for $j = 1, ..., k$. Thus, $(\bar{x}^1 - \lambda\mathbf{e}^i, \bar{x}^2 - \lambda\mathbf{e}^i, ..., \bar{x}^k - \lambda\mathbf{e}^i)$ is an path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, T)$ as required. □

Since we assume i is bounded, the order $\succ_{\text{-}\mathbf{e}^i}$ (or $\succ_i$) is thus a term order for $\mathcal{L}$. Importantly then, a $\succ_i$-reduction path is also an ($\text{-}\mathbf{e}^i$)-reduction path. So, if we compute a set T such that for all $\nu \in \mathbb{Z}^n$ and $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, S)$, there exists a $\succ_i$-reduction path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, T)$, then T is i-saturated on S. We can actually compute such a set T using the completion procedure, Algorithm 2. At first, this does not seem possible since the completion procedure requires that we start with a Markov basis, but we can relax this requirement on the completion procedure and run the completion procedure with a set that is not a Markov basis. To know what exactly we would compute by running the completion procedure on a set that is not a Markov basis, we need more general versions of Lemma 5.1.3, Corollary 5.1.4, and Lemma 5.1.5 from Section 5.1.

The following lemma is a more general version of Lemma 5.1.3. The lemma is essentially the same as Lemma 5.1.3 – the proof is the same – but it does not assume that the given set of vectors is a Markov basis.

Lemma 6.3.5. *Let $\nu \in \mathbb{Z}^n$, and let $G \subseteq \mathcal{L}_{\succ}$. If there exists a $\succ$-reduction path between x' and y' for every $\succ$-critical path (x', z', y') in $\mathcal{G}_{\mathcal{L}}(\nu, G)$, then there exists a $\succ$-reduction path between x and y for all $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ where x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, G)$.*

We can easily extend the previous lemma to arrive at a corresponding result to Corollary 5.1.4.

Corollary 6.3.6. *Let $\mathcal{L}$ be a lattice, and let $\succ$ be a term order, and let $G \subseteq \mathcal{L}_\succ$. If there exists a $\succ$-reduction path between x' and y' for every $\succ$-critical path (x', z', y') in $\mathcal{G}_\mathcal{L}(\nu, G)$ for all $\nu \in \mathbb{Z}^n$, then there exists a $\succ$-reduction path between x and y for all $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ where x and y are connected in $\mathcal{G}_\mathcal{L}(\nu, G)$ for all $\nu \in \mathbb{Z}^n$.*

For the exactly same reasons behind Lemma 5.1.5, we only need to check for a reduction path between minimal critical paths, and thus, we have the follow Lemma corresponding to Lemma 5.1.5.

Lemma 6.3.7. *Let $\mathcal{L}$ be a lattice, and let $\succ$ be a term order, and let $G \subseteq \mathcal{L}_\succ$. If for all $u, v \in G$, there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$, then then there exists a $\succ$-reduction path between x and y for all $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ for all $\nu \in \mathbb{Z}^n$ where x and y are connected in $\mathcal{G}_\mathcal{L}(\nu, G)$.*

If we re-examine the completion procedure, we see that, given some starting set G, which may or may not be a Markov basis, the completion procedure adds vectors to G until there exists a $\succ$-reduction path between $x^{(u,v)}$ and $y^{(u,v)}$ in $\mathcal{G}_\mathcal{L}(\nu^{(u,v)}, G)$ for all $u, v \in G$. Thus, by Lemma 6.3.7 above, after running the completion procedure, there exists a $\succ$-reduction path between x and y for all $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ for all $\nu \in \mathbb{Z}^n$ where x and y are connected in $\mathcal{G}_\mathcal{L}(\nu, G)$. We must be careful however when optimising the completion procedure without a Markov basis since Criterion 2 is no longer valid, but fortunately, Criteria 1 and 3 are still valid though. It is possible to use a weaker form of Criterion 2, but we will not detail it here since Criteria 1 and 3 are usually adequate for a fast implementation. Additionally, the truncated completion procedure no longer works without a Markov basis. Again, it is possible to use a weaker form of the truncated completion procedure, but we will not detail it here.

Let $T = \mathcal{CP}^i_\mathcal{L}(\succ_i, S)$. Then, by the properties of the completion procedure as discussed above, for all $\nu \in \mathbb{Z}^n$ and $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ where x and y are connected in $\mathcal{G}_\mathcal{L}(\nu, S)$, there exists a $\succ_i$-reduction path from x to y in $\mathcal{G}_\mathcal{L}(\nu, T)$; T is therefore i-saturated on S.

We now discuss the case where i is unbounded, which is actually easier than the case where i is bounded. Let T be σ-saturated on S. If i is unbounded, then there exists a vector $u \in \mathcal{L} \cap \mathbb{N}^n$ such that $u_i > 0$, and thus, by Lemma 6.3.8 below, the set $T' = T \cup \{u\}$ is i-saturated on T; therefore, T' is $(\sigma \cup i)$-saturated on S.

Lemma 6.3.8. *Let $S \subseteq \mathcal{L}$. If there exists $u \in S$ where $u \in \mathcal{L} \cap \mathbb{N}^n$ and $u \neq \mathbf{0}$, then S is $\operatorname{supp}(u)$-saturated on S.*

Proof. Let $x, y \in \mathcal{F}_\mathcal{L}(\nu)$ for some $\nu \in \mathbb{Z}^n$ where x and y are connected in $\mathcal{G}^\sigma_\mathcal{L}(\nu, S)$ where $\sigma = \operatorname{supp}(u)$. We can translate a path from x to y in $\mathcal{G}^\sigma_\mathcal{L}(\nu, S)$ (i.e. a path that is non-negative on the $\bar{\sigma}$ components) to a path in $\mathcal{G}_\mathcal{L}(\nu, S)$ (i.e. a path that is non-negative on all components) by adding u to the start of the path as many times as necessary and subtracting u from the end of the path the same number of times. Therefore, x and y are connected in $\mathcal{G}_\mathcal{L}(\nu, S)$ as required. □

Algorithm 6 Saturation algorithm

Input: a lattice $\mathcal{L}$, and a spanning set S of $\mathcal{L}$.
Output: a Markov basis M of $\mathcal{L}$.

```
M := S
σ := ∅
while σ ≠ {1, ..., n} do
  Select i ∈ σ̄
  if i is bounded then
    M := CP_L(≻_i, M)
  else
    Compute u ∈ L ∩ N^n such that u_i > 0
    M := M ∪ {u}
  end if
  σ := σ ∪ i
end while
return M.
```

We now arrive at the Saturation algorithm below.

Lemma 6.3.9. *Algorithm 6 terminates and satisfies its specifications.*

Proof. Algorithm 6 terminates, since Algorithm 2 always terminates. We show at the beginning of each iteration that M is σ-saturated on S, so at the end of the algorithm M is $\{1,\dots,n\}$-saturated on S, and therefore, M is a Markov basis of $\mathcal{L}$. At the beginning of the first iteration, M is σ-saturated on S since $\sigma = \emptyset$ and $M = S$. So, we can assume it is true for the current iteration, and now, we show it is true for the next iteration. Assume that i is bounded. Let $M' := \mathcal{CP}_{\mathcal{L}}(\succ_i, M)$. Then, by Lemma 6.3.4, M' is i-saturated on M, so by Lemma 6.3.3, M' is $(\sigma \cup i)$-saturated on S. On the other hand, assume that i is unbounded. Let $M' := M \cup \{u\}$ for some $u \in \mathcal{L} \cap \mathbb{N}^n$ where $u_i > 0$. Then, by Lemma 6.3.8, M' is i-saturated on M and thus $(\sigma \cup i)$-saturated on S. So whether i is bounded or unbounded, M is σ-saturated on S at the beginning of the next iteration. □

During the Saturation algorithm, we saturate n times, once for each $i \in \{1,\dots,n\}$. However, as proven in [56], it is in fact only necessary to perform at most $\lfloor \frac{n}{2} \rfloor$ saturations. Given $S, T \subseteq \mathcal{L}$, we can show that there always exists a $\sigma \subseteq \{1,\dots,n\}$ where $|\sigma| \leq \lfloor \frac{n}{2} \rfloor$ such that if T is σ-saturated on S, then T is $\{1,\dots,n\}$-saturated on S. The following two lemmas prove the result.

Lemma 6.3.10. *Let $\sigma \subseteq \{1,\dots,n\}$, $S, T \subseteq \mathcal{L}$ where T is σ-saturated on S, and $u \in S$. If $\operatorname{supp}(u^-) \subseteq \sigma$ or $\operatorname{supp}(u^+) \subseteq \sigma$, then T is $(\operatorname{supp}(u) \cup \sigma)$-saturated on S.*

Proof. Assume that $\operatorname{supp}(u^-) \subseteq \sigma$. Let $\tau = \operatorname{supp}(u)$. Let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$ where x and y are connected in $\mathcal{G}_{\mathcal{L}}^{\tau \cup \sigma}(\nu, S)$. We must show that x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, T)$. A path from x to y in $\mathcal{G}_{\mathcal{L}}^{\tau \cup \sigma}(\nu, S)$ can be transformed into a path from x to y in $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, S)$ by adding u to the start of the path as many times as necessary and subtracting u from the end of the path the same number of times in

such a way that the $\operatorname{supp}(u^+)$ components of the path become non-negative. Such a path transformation also affects the $\operatorname{supp}(u^-)$ components, but since $\operatorname{supp}(u^-) \subseteq \sigma$, the resulting path is a path in $\mathcal{G}_{\mathcal{L}}^{\sigma}(\nu, S)$. Then, since T is σ-saturated on S, there exists a path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, S)$ as required.

The case where T is $\operatorname{supp}(u^+) \subseteq \sigma$ is essentially the same as above. □

Note that Lemma 6.3.8 is a special case of Lemma 6.3.10 above where $\operatorname{supp}(u^-) = \emptyset$.

Lemma 6.3.11. *Let $S, T \subseteq \mathcal{L}$. There exists a $\sigma \subseteq \{1, \ldots, n\}$ where $|\sigma| \leq \lfloor \frac{n}{2} \rfloor$ such that if T is σ-saturated on S, then T is $\{1, \ldots, n\}$-saturated on S.*

Proof. We show this by construction. Without loss of generality, we assume that $\mathcal{L}$ is not contained in any of the linear subspaces $\{x_i : x_i = 0, x \in \mathbb{R}^n\}$ for $i = 1, \ldots, n$, otherwise we may simply delete this component.

Let $\sigma = \emptyset$, $\tau = \emptyset$, and $U = \emptyset$. Repeat the following steps until $\tau = \{1, \ldots, n\}$.

(i). Select $u \in S$ such that $\operatorname{supp}(u) \setminus \tau \neq \emptyset$.

(ii). If $|\operatorname{supp}(u^+) \setminus \tau| \geq |\operatorname{supp}(u^-) \setminus \tau|$, then $\sigma := \sigma \cup \operatorname{supp}(u^-)$, else $\sigma := \sigma \cup \operatorname{supp}(u^+)$.

(iii). Set $\tau := \tau \cup \operatorname{supp}(u)$, and set $U := U \cup \{u\}$.

The procedure must terminate since during each iteration we increase the size of τ. Note that, at termination, $U \subseteq S$, $\bigcup_{u \in U} \operatorname{supp}(u) = \tau = \{1, \ldots, n\}$, and for all $u \in U$ either $\operatorname{supp}(u^+) \subseteq \sigma$ or $\operatorname{supp}(u^-) \subseteq \sigma$. Therefore, by applying Lemma 6.3.10 recursively for each $u \in U$, we have that if T is σ-saturated on S, then T is $\{1, \ldots, n\}$-saturated on S. Lastly, since in each iteration we add at least twice as many components to τ as to σ, we conclude that at termination $|\sigma| \leq \lfloor \frac{n}{2} \rfloor$. □

Example 6.3.12. *Consider again the projected lattice $\mathcal{L}$ from Example 3.1.6 that is generated by the vectors $(2, \text{-}2, \text{-}2, 1)$ and $(3, 1, \text{-}1, \text{-}1)$. Recall that the set*

$$M = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (5, \text{-}1, \text{-}3, 0), (1, 3, 1, \text{-}2)\}$$

is a Markov basis of $\mathcal{L}$. We will compute this set using the Saturation algorithm.

We start the Saturation using the set $S := \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1)\}$, which is a spanning set of $\mathcal{L}$. Then, by Lemma 6.3.10, since $\operatorname{supp}((3, 1, \text{-}1, \text{-}1)^+) = \{1, 2\}$ and $\operatorname{supp}((3, 1, \text{-}1, \text{-}1)^-) = \{3, 4\}$, if a set $T \subseteq \mathcal{L}$ is $\{1, 2\}$-saturated on S, then T is $\{1, 2, 3, 4\}$-saturated on S, and thus, a Markov basis of $\mathcal{L}$. So, to compute a Markov basis of $\mathcal{L}$, we only need to saturate on $\{1, 2\}$.

First, set $T := S$.

(i). Select $i := 1$. The component i is bounded. Set

$$T := \mathcal{CP}_{\mathcal{L}}(\succ_i, T) = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (1, 3, 1, \text{-}2)\}.$$

So, T is $\{1\}$-saturated on S. Note that T is not a Markov basis of $\mathcal{L}$.

(ii). Select $i := 2$. The component i is bounded. Set

$$T := \mathcal{CP}_{\mathcal{L}}(\succ_i, T) = \{(2, -2, -2, 1), (3, 1, -1, -1), (1, 3, 1, -2), (5, -1, -3, 0)\}.$$

So, T is $\{1, 2\}$-saturated on S.

From our discussion above, since T is $\{1, 2\}$-saturated on S, T is also $\{1, 2, 3, 4\}$-saturated on S, and therefore, T is a Markov basis of $\mathcal{L}$.

In the last part of this section on the Saturation algorithm, we will compare what the Saturation algorithm computes with what the Project-and-Lift algorithm of Section 6.1 computes. First, recall from Section 6.1 that the Project-and-Lift algorithm sequentially computes Markov bases of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ for each value of σ starting from some initial set until the empty set where we incrementally remove elements from σ. It follows from Lemma 6.3.13 below that the Saturation algorithm also sequentially computes Markov bases of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, so in this respect the algorithms are very similar.

Lemma 6.3.13. *Let $S, T \subseteq \mathcal{L}$ where S spans $\mathcal{L}$. If T is σ-saturated on S, then T is a Markov basis of $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\cdot)$.*

Proof. First, recall that any set S that spans $\mathcal{L}$ is a Markov basis of $\mathcal{F}^{\tau}_{\mathcal{L}}(\cdot)$ where $\tau = \{1, ..., n\}$. Let $x, y \in \mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$. Since S is a Markov basis of $\mathcal{F}^{\tau}_{\mathcal{L}}(\cdot)$, x and y are connected in $\mathcal{G}^{\tau}_{\mathcal{L}}(\nu, S)$. Let $(x^1, ..., x^k)$ be a path from x to y in $\mathcal{G}^{\tau}_{\mathcal{L}}(\nu, S)$. Then, let $\gamma \in \mathbb{Z}^n$ such that $\gamma_{\sigma} = \mathbf{0}$ and $\gamma_{\bar{\sigma}} + x^i_{\bar{\sigma}} \geq \mathbf{0}$ for $i = 1, ..., k$. Then, $(x^1 + \gamma, ..., x^k + \gamma)$ is a path from $x + \gamma$ to $y + \gamma$ in $\mathcal{G}^{\sigma}_{\mathcal{L}}(\nu + \gamma, S)$, and thus, since T is σ-saturated on S, $x + \gamma$ to $y + \gamma$ are connected in $\mathcal{G}_{\mathcal{L}}(\nu + \gamma, T)$. Let $(\bar{x}^1, ..., \bar{x}^k)$ be a path from $x + \gamma$ to $y + \gamma$ in $\mathcal{G}_{\mathcal{L}}(\nu + \gamma, T)$. Then, $(\bar{x}^1 - \gamma, ..., \bar{x}^k - \gamma)$ is a path from x to y in $\mathcal{G}^{\bar{\sigma}}_{\mathcal{L}}(\nu, T)$ since $\gamma_{\sigma} = \mathbf{0}$. Hence, T is a Markov basis of $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\cdot)$. □

The Saturation algorithm sequentially computes σ-saturated sets for values σ starting from the empty set until $\{1, ..., n\}$, which implies that the Saturation algorithm sequentially computes Markov bases of $\mathcal{F}^{\tau}_{\mathcal{L}}(\cdot)$ starting from $\tau = \{1, ..., n\}$ until $\tau = \emptyset$.

Although a σ-saturated set is a Markov basis of $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\cdot)$, the converse is not true. Let $x, y \in \mathcal{F}_{\bar{\sigma}}(\mathcal{L})\nu$ for some $\nu \in \mathbb{Z}^n$ such that x and y are connected in $\mathcal{G}_{\mathcal{L}}(\nu, S)$ for some set S that spans $\mathcal{L}$. If a set T is σ-saturated on S, then there exists a path from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, T)$, or in other words, a path from x to y that is non-negative on all components. On the other hand, if T is a Markov basis of $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\cdot)$, then there exists a path from x to y in $\mathcal{G}^{\bar{\sigma}}_{\mathcal{L}}(\nu, T)$, or in other words, there is a path from x to y that remains non-negative on the σ components, but we can infer nothing about the non-negativity of the $\bar{\sigma}$ components on the path. So, a σ-saturated set is more than just a Markov basis of $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\cdot)$. For this reason, σ-saturated sets can be much larger than Markov bases of $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\cdot)$, so the Project-and-Lift algorithm computes with smaller intermediate sets because it only really computes Markov bases of $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}}(\cdot)$. We will see this in the Section 6.5.

6.4 Lift-and-Project algorithm

The idea behind this algorithm is to lift a spanning set S of $\mathcal{L} \subseteq \mathbb{Z}^n$ to a spanning set $S' \subseteq \mathbb{Z}^{n+1}$ of $\mathcal{L}' \subseteq \mathbb{Z}^{n+1}$ in such a way that we can compute a set $G' \subseteq \mathcal{L}'$ that generates $\mathcal{L}'$ in only one saturation step. Then, we project G' to $G \subseteq \mathcal{L}$, so that G is a Markov basis of $\mathcal{L}$.

Let S be a spanning set of $\mathcal{L} \subseteq \mathbb{Z}^n$. Let $S' := \{(u,0) : u \in S\} \cup \{(1,\ldots,1,\text{-}1)\}$, and let $\mathcal{L}' \subseteq \mathbb{Z}^{n+1}$ be the lattice spanned by S'. Since the vector $(1,\ldots,1,\text{-}1)$ is in S', it follows from Lemma 6.3.10, that if a set $G' \subseteq \mathcal{L}'$ is $\{n+1\}$-saturated on S', then G' is $\{1,\ldots,n+1\}$-saturated on S, and hence, G' is a Markov basis of $\mathcal{L}'$. Also, by construction, the new component $n+1$ is bounded since there does not exist a non-negative vector $\mathcal{L}'$ with a strictly positive $(n+1)$th component. Now, using exactly the same idea behind the Saturation algorithm, if we let $G' := \mathcal{CP}_{\mathcal{L}'}(\succ_{n+1}, S')$, then G' must be $\{n+1\}$-saturated on S' by Lemma 6.3.4 and thus a Markov basis of $\mathcal{L}'$.

So, at the moment, we have a Markov basis G' for $\mathcal{L}'$, and from this, we need to extract a Markov basis of $\mathcal{L}$. We define the linear map $\rho : \mathbb{Z}^{n+1} \mapsto \mathbb{Z}^n$ where

$$\rho(u') := (u'_1 + u'_{n+1}, u'_2 + u'_{n+1}, \ldots, u'_n + u'_{n+1}).$$

Observe that ρ maps $\mathbb{Z}^{n+1}$ onto $\mathbb{Z}^n$, maps $\mathcal{L}'$ onto $\mathcal{L}$, and maps $\mathcal{F}_{\mathcal{L}'}(\nu')$ onto $\mathcal{F}_{\mathcal{L}}(\nu)$ where $\nu = \rho(\nu')$. Let $G := \{\rho(u') : u' \in G'\} \setminus \{\mathbf{0}\}$. So, $G \subset \mathcal{L}$, and we now show that in fact G is a Markov basis of $\mathcal{L}$. Let $x, y \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$, and let $x' := (x,0)$, $y' := (y,0)$, and $\nu' := (\nu,0)$; hence, $\rho(x') = x$, $\rho(y') = y$, and $\rho(\nu') = \nu$. Since G' is a Markov basis of $\mathcal{L}'$ there must exist a path from x' to y' in $\mathcal{G}_{\mathcal{L}'}(\nu', G')$. Let $(x'^0, \ldots, x'^k)$ be a path in $\mathcal{G}_{\mathcal{L}'}(\nu', G')$ from x' to y'. Then, $(\rho(x'^0), \ldots, \rho(x'^k))$ is a walk from x to y in $\mathcal{G}_{\mathcal{L}}(\nu, G)$, so after removing cycles, we have a path from x to y. Cycles may exist because the kernel of ρ is non-trivial: $\ker(\rho) = \{(\gamma,\ldots,\gamma,-\gamma) : \gamma \in \mathbb{Z}\}$. Hence, G is a Markov basis of $\mathcal{L}$. We thus arrive at the Lift-and-Project algorithm.

Algorithm 7 Lift-and-Project algorithm

Input: a set $S \subseteq \mathcal{L}$ that spans $\mathcal{L}$.
Output: a Markov basis G of $\mathcal{L}$
  $S' := \{(u,0) : u \in S\} \cup \{(1,\ldots,1,\text{-}1)\}$
  $G' := \mathcal{CP}_{\mathcal{L}'}(\succ_{n+1}, S')$
  $G := \{\rho(u') : u' \in G'\} \setminus \{0\}$
  return G.

Example 6.4.1. *Consider again the projected lattice $\mathcal{L}$ from Example 3.1.6 that is generated by the vectors* $(2,\text{-}2,\text{-}2,1)$ *and* $(3,1,\text{-}1,\text{-}1)$. *Recall that the set*

$$M = \{(2,\text{-}2,\text{-}2,1), (3,1,\text{-}1,\text{-}1), (5,\text{-}1,\text{-}3,0), (1,3,1,\text{-}2)\}$$

is a Markov basis of $\mathcal{L}$. We will compute this set using the Lift-and-Project algorithm.

The set $S := \{(2,\text{-}2,\text{-}2,1), (3,1,\text{-}1,\text{-}1)\}$ spans $\mathcal{L}$. Next, we extend the lattice $\mathcal{L}$ to $\mathcal{L}'$, and we extend spanning set S to $S' := \{(2,\text{-}2,\text{-}2,1,0), (3,1,\text{-}1,\text{-}1,0), (1,1,1,1,\text{-}1)\}$,

which is a spanning set of $\mathcal{L}'$. Next, we compute $G' = \mathcal{CP}_{\mathcal{L}'}(\succ_5, S')$. This has the following eight vectors:

$$G' := \left\{ \begin{array}{rrrrr} (\text{-}5, & 1, & 3, & 0, & 0) \\ (\text{-}2, & 2, & 2, & \text{-}1, & 0) \\ (\text{-}3, & \text{-}1, & 1, & 1, & 0) \\ (4, & 2, & 0, & 0, & \text{-}1) \\ (\text{-}1, & \text{-}3, & \text{-}1, & 2, & 0) \\ (6, & 0, & \text{-}2, & 1, & \text{-}1) \\ (9, & 1, & \text{-}3, & 0, & \text{-}1) \\ (14, & 0, & \text{-}6, & 0, & \text{-}1) \end{array} \right\}.$$

Note that G' is twice as large as a Markov basis of $\mathcal{L}$. Next, we perform the transformation $G := \{\rho(u') : u' \in G'\} \setminus \{0\}$ removing any duplicate vectors (or vectors that are negative multiples of each other). We then have the following six vectors:

$$G := \{(\text{-}5, 1, 3, 0), (\text{-}2, 2, 2, \text{-}1), (\text{-}3, \text{-}1, 1, 1), (\text{-}1, \text{-}3, \text{-}1, 2), (8, 0, \text{-}4, \text{-}1), (13, \text{-}1, \text{-}7, \text{-}1)\}.$$

The set G is a Markov basis of $\mathcal{L}$.

To make the algorithm more efficient, we can use a different additional vector to $(1, \ldots, 1, \text{-}1)$. By Lemma 6.3.11, we know that given a spanning set S, there exists a σ where $|\sigma| \leq \lfloor \frac{n}{2} \rfloor$ such that if T is σ-saturated on S, then T is a Markov basis of $\mathcal{L}$. Then, instead of $(1, \ldots, 1, \text{-}1)$, it suffices to use the additional vector $s = \sum_{i \in \sigma} \mathbf{e}^i - \mathbf{e}^{n+1}$, which has the important property that $\text{supp}(s^+) = \sigma$ and $\text{supp}(s^-) = \{n+1\}$. Set $S' := \{(u, 0) : u \in S\} \cup \{s\}$, and let $\mathcal{L}'$ be the lattice spanned by S'. Then, from Lemma 6.3.10, since $s \in S'$, if a set $G' \subseteq \mathcal{L}'$ is $\{n+1\}$-saturated on S', then G' is $(\sigma \cup \{n+1\})$-saturated on S'. Also, since $\{(u, 0) : u \in S\} \subseteq S'$, from the proof of Lemma 6.3.11, it follows that if G' is σ-saturated on S', then G' is $\{1, \ldots, n\}$-saturated. Hence, G' is $\{1, \ldots, n+1\}$-saturated, and therefore, a Markov basis of $\mathcal{L}'$. So again, we can compute a Markov basis G' of $\mathcal{L}'$ in one saturation step. Also, we similarly define the linear map $\rho_\sigma : \mathbb{Z}^{n+1} \mapsto \mathbb{Z}^n$ where $\rho_\sigma(x') := (x'_1, x'_2, \ldots, x'_n) + (\sum_{i \in \sigma} \mathbf{e}^i) x_{n+1}$. Then, $G := \{\rho_\sigma(x') : x' \in G'\}$ is a Markov basis of $\mathcal{L}$. As a general rule, the smaller the size of σ, the faster the algorithm.

Example 6.4.2. *Consider again the projected lattice $\mathcal{L}$ from Example 3.1.6 that is generated by the vectors $(2, \text{-}2, \text{-}2, 1)$ and $(3, 1, \text{-}1, \text{-}1)$. Recall that the set*

$$M = \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1), (5, \text{-}1, \text{-}3, 0), (1, 3, 1, \text{-}2)\}$$

is a Markov basis of $\mathcal{L}$. We will compute this set using the Lift-and-Project algorithm with the above improvement. We will see that using the above improvement means that a smaller set is computed.

The set $S := \{(2, \text{-}2, \text{-}2, 1), (3, 1, \text{-}1, \text{-}1)\}$ spans $\mathcal{L}$. Next, we extend the lattice $\mathcal{L}$ to $\mathcal{L}'$ such that $S' = \{(2, \text{-}2, \text{-}2, 1, 0), (3, 1, \text{-}1, \text{-}1, 0), (1, 1, 0, 0, \text{-}1)\}$ spans $\mathcal{L}'$. Note that, from Lemma 6.3.10, if a set G' is $\{n+1\}$-saturated on S', then G' is $\{1, 2, n+1\}$-saturated on S' since $\text{supp}((1, 1, 0, 0, \text{-}1)^-) = \{n+1\}$ and $\text{supp}((1, 1, 0, 0, \text{-}1)^+) = \{1, 2\}$. Moreover, if G' is $\{1, 2, n+1\}$-saturated on S', then G' is $\{1, 2, 3, 4\}$-saturated

on S' *since* $\text{supp}((3,1,\text{-}1,\text{-}1)^+) = \{1,2\}$ *and* $\text{supp}((3,1,\text{-}1,\text{-}1)^-) = \{3,4\}$*, and* G' *is a Markov basis of* $\mathcal{L}'$*. Thus, we only need to perform one saturation step on* $n+1$*. So, we compute* $G' = \mathcal{CP}_{\mathcal{L}'}(\succ_5, S')$*. This has the following seven vectors:*

$$G' := \left\{ \begin{array}{rrrrr} (\text{-}5, & 1, & 3, & 0, & 0) \\ (\text{-}2, & 2, & 2, & \text{-}1, & 0) \\ (\text{-}2, & 0, & 1, & 1, & \text{-}1) \\ (1, & 1, & 0, & 0, & \text{-}1) \\ (0, & \text{-}2, & \text{-}1, & 2, & \text{-}1) \\ (3, & \text{-}1, & \text{-}2, & 1, & \text{-}1) \\ (6, & 0, & \text{-}3, & 0, & \text{-}1) \end{array} \right\}.$$

Note that G' *is smaller than in Example 6.4.1, but it is still a bit larger than a Markov basis of* $\mathcal{L}$*. Next, we perform the transformation* $G := \{\rho_\sigma(u') : u' \in G'\} \setminus \{0\}$ *removing any duplicate vectors (or vectors that are negative multiples of each other). We then have the following four vectors:*

$$G := \{(\text{-}5,1,3,0), (\text{-}2,2,2,\text{-}1), (\text{-}3,\text{-}1,1,1), (\text{-}1,\text{-}3,\text{-}1,2)\}.$$

The set G *is a Markov basis of* $\mathcal{L}$*.*

There are two major problems with the Lift-and-Project algorithm. The first being, as we saw in the examples, that the size of the set computed may be much larger than necessary, and, as with the Saturation algorithm, we cannot use Criterion 3 when running the completion procedure since we are not computing with a Markov basis. Another problem is that we cannot perform any truncation until after we have computed the Markov basis also because we are not computing with a Markov basis.

Bigatti et al. in [12] describe some improvements for the completion procedure specifically for computing $G' := \mathcal{CP}_{\mathcal{L}}(\succ_{n+1}, S')$, so the improvements are specific to the Lift-and-Project algorithm. We will not discuss these improvements here except to say that they do not decrease the size of G', so the main problem with the approach remains.

6.5 Comparison of algorithms

In this section, we compare the performance of implementations of the Saturation algorithm, Lift-and-Project algorithm, and the Project-and-Lift algorithm in `4ti2` v.1.3 ([1]) with the implementation of the Saturation algorithm and the Lift-and-Project algorithm in CoCoA 4.6 ([23]). We will also compare the performance of these algorithms with an implementation of the algorithm of Hemmecke ([50]) to compute a Graver basis in `4ti2` v.1.3 ([1]) (see Section 5.4.1). Recall that a Graver basis is a Gröbner basis and thus a Markov basis and usually much larger than a minimal Markov basis, but sometimes a Graver basis is a minimal Markov basis. The software package Singular ([46]) also has an implementation of the Saturation algorithm and the Lift-and-Project algorithm, but CoCoA seems the faster implementation, at least on the example problems in this section, so we only compare

`4ti2` with CoCoA. In summary, the computational results show that the Project-and-Lift algorithm is significantly faster than the other algorithms. The exception to this rule is in the special case where a Markov basis is the same as a Graver basis, in which case, the Graver basis algorithm is faster.

In Table 6.5, we list the time taken to compute Markov bases of different problems. The computations were done on a Intel XEON 3.2 GHz machine with 4Gb of RAM running Linux Redhat. Computation times are given in seconds rounded to the nearest one-hundredth of a second. The table entries with a "$*$" indicate after several hours of computation time the computation was still nowhere near completion. In the first column of Table 6.5, we list the problems for which we computed the Markov bases. The second column is the number of variables in the problem. The third column is the size of a minimal MB basis. The last six columns correspond to the different algorithms and their different implementations. The column headings are explained in Table 6.5 below.

Name	Software	Algorithm
C-SAT	CoCoA v4.6	Saturation
C-L&P	CoCoA v4.6	Lift-and-Project
SAT	`4ti2` v1.3	Saturation
L&P	`4ti2` v1.3	Lift-and-Project
P&L	`4ti2` v1.3	Project-and-Lift
Grav	`4ti2` v1.3	Graver basis

Table 6.4: Software and algorithm

The problems that we compute the Markov basis for are as follows. The first 4 problems, (333, 334, 335, and 344) correspond to three-dimensional contingency tables. The example `grin4x8` is taken from [57], and the examples `grin4x10` and `grin4x13` are extensions of the example `grin4x8`, an the examples `hppi10,hppi12,hppi14` correspond to the computation of homogeneous primitive partition identities (see for example Chapters 6 and 7 in [79]). Finally, the examples `cuww1,cuww3,cuww5` arise from knapsack problems presented in [27].

The running times give a clear ranking of three of the algorithms: first comes our Project-and-Lift method, second comes the Saturation algorithm, and last comes the Lift-and-Project method. For all large problems, the presented Project-and-Lift algorithm wins significantly amongst the above three algorithms, and moreover, the Project-and-Lift algorithm seems quite robust in the sense that the computations times reflect the size of the minimal Markov bases.

The Graver basis algorithm, listed in the last column, is the faster algorithm for the examples *hppi10,hppi12,hppi14*.[1] For these examples, a Graver basis is essentially the

[1]We should point out here that the implementation of the Graver basis algorithm does not check for integer arithmetic overflow, but the implementation of the Project-and-Lift algorithm does, and without this overflow checking the implementations take about the same time for the examples *hppi10,hppi12,hppi14*. Also, the Graver basis implementation computes with vectors that are half the size as in the Project-and-Lift implementation, which is an advantage for the Graver basis implementation.

Problem	Vars	Size	C-SAT	C-L&P	SAT	L&P	P&L	Grav
333	27	81	0.05	0.19	0.07	1.49	0.00	0.08
334	36	450	1.72	2095.61	2.92	8309.15	0.12	27.52
335	45	2670	176.25	*	136.50	*	2.80	8192.56
344	48	4068	2798.02	*	1177.61	*	27.20	*
grin4x8	8	211	0.02	0.02	0.03	0.09	0.01	*
grin4x10	10	1412	1.34	2.36	1.84	4.24	0.37	*
grin4x13	13	10868	2100.32	3609.54	257.35	417.09	35.75	*
hppi10	20	1830	5.14	6.51	8.54	25.80	0.50	0.26
hppi12	24	8569	600.22	614.20	419.57	1299.59	13.94	6.81
hppi14	14	34355	18510.63	*	9261.00	*	276.14	144.41
cuww1	5	5	1.72	1.42	0.00	0.00	0.00	*
cuww3	6	16	6.31	4.94	0.11	0.12	0.00	*
cuww5	8	27	467.72	0.15	376.68	0.11	0.00	*

Table 6.5: Comparison of computing times.

same as a Markov basis, and moreover, the Graver basis algorithm is essentially the same as the Project-and-Lift algorithm, so the specialised algorithm and implementation of a Graver basis is faster than the more general Project-and-Lift algorithm and implementation for Markov bases. The Graver basis algorithm is much slower however on the other examples for which the Graver basis is much larger than a minimal Markov basis. For example, for the problem 335, a minimal Markov basis has 2670 many elements, but a Graver basis has 263610 many elements. Note that the running times of our Project-and-Lift algorithm are not far from the running times of the Graver basis algorithm, and indeed, the algorithms are very similar.

The advantage of our Project-and-Lift algorithm is that it effectively performs computations in projected subspaces of $\mathcal{L}$. Thus, we obtain comparably small intermediate sets during the computation. Only the final iteration that deals with all variables reaches the true output size. In contrast with this, the Saturation algorithm usually comes close to the final output size even after just the first iteration of the algorithm and then continues computing with as many vectors. See Figure 6.1, for a comparison of intermediate set sizes in each iteration for computing $3 \times 4 \times 4$ contingency tables.

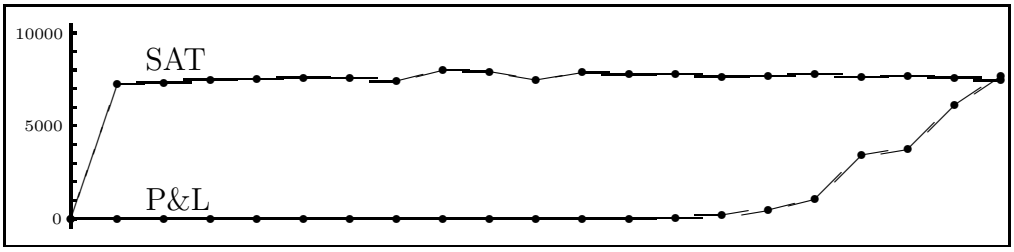

Figure 6.1: Comparison of intermediate set sizes in each iteration.

Moreover, the Project-and-Lift algorithm, performs Gröbner basis computations using a Markov basis, and thus can take full advantage of Criterion 2 which, as computational experience shows, is extremely effective. In fact, we only applied Criterion 2 and 1 (applied in that order) for the Project-and-Lift algorithm since Criterion 3 only slowed down the algorithm. However, for the Saturation algorithm where we cannot apply Criterion 2 fully, Criterion 3 was very effective. In this case, we applied Criterion 1, then 3, and then 2, in that order.

Chapter 7

Applications

In this chapter, we present applications of Gröbner bases and Markov bases. First, we present an application of computing Markov bases in algebraic statistics. Using the algorithms that are developed in this thesis for computing Markov bases, we were able to solve an open challenge in algebraic statistics. Next, we describe an application using Markov bases to show that a semigroup is not normal. Then, we present a new Gröbner basis approach for computing a feasible solution of an integer program, which is based upon an extension of the Project-and-Lift method for computing Markov bases. We also further extend the Project-and-Lift method again to arrive at a Gröbner bases approach for solving integer programs that uses problem specific structure to solve an integer program. Lastly, we present an approach to the feasibility enumeration problem, and we discuss how this method could be used in our optimisation algorithm for integer programs.

7.1 Algebraic statistics

We now discuss one of the main applications of Markov bases to provide some motivation for defining and computing Markov bases. An application of Markov bases is in algebraic statistics where we wish to test whether the relative frequencies of observed events follow a given frequency distribution. The events should be independent and have the same distribution, and the outcomes of each event must be mutually exclusive. We only give a very brief account of this application of Markov bases. See [30] for a more detailed discussion.

For example, we wish to test the hypothesis that there is an association between birthday and deathday. In this case, that we have two observables: the month of birth and the month of death. Table 7.1 gives the frequencies of a person being born in a specific month and dying in a specific month from a sample of people consisting of 82 descendants of Queen Victoria (data from [30]).

We want to know if the sample data is statistically significantly different from the expected data if the months of birth and death were independent. If the sample data is statistically significantly different, we conclude that there is a statistically significant relationship between the variables. Note that a statistically significant

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Jan	1	0	0	0	1	2	0	0	1	0	1	0	6
Feb	1	0	0	1	0	0	0	0	0	1	0	2	5
Mar	1	0	0	0	2	1	0	0	0	0	0	1	5
Apr	3	0	2	0	0	0	1	0	1	3	1	1	12
May	2	1	1	1	1	1	1	1	1	1	1	0	12
Jun	2	0	0	0	1	0	0	0	0	0	0	0	3
Jul	2	0	2	1	0	0	0	0	1	1	1	2	10
Aug	0	0	0	3	0	0	1	0	0	1	0	2	7
Sep	0	0	0	1	1	0	0	0	0	0	1	0	3
Oct	1	1	0	2	0	0	1	0	0	1	1	0	7
Nov	0	1	1	1	2	0	0	2	0	1	1	0	9
Dec	0	1	1	0	0	0	1	0	0	0	0	0	3
Total	13	4	7	10	8	4	5	3	4	9	7	8	82

Table 7.1: A contingency table of month of birth versus month of death ([30]).

relationship may not be theoretically or practically important or even very large – a relationship need only exist. A test of statistical significance tells us how confidently we can generalise to a larger population from a sample of that population. In other words, we test whether our actual results are different enough from what is expected to surpass a certain probability that they are due to sampling error.

The question first is then what do we expect? If the months of birth and death were independent, then we would expect that the number of people who were born in month b and died in month d in the sample is $\frac{p_b \cdot p_d}{p}$ where p is the total number of people in the sample, p_b is the total number of people born in month b, and p_d is the total number of people who died in month d. For example, we would expect that there are $\frac{13\times12}{82} \approx 1.9$ people in the sample who were born in April and died in January.

Now that we know what we would expect, we need some way of measuring how different the observed data is to the expected data. We can measure the difference between the observed data and the expected data using using the *chi-square statistic*:

$$\chi^2 = \sum_{b\in M, d\in M} \frac{(O_{bd} - E_{bd})^2}{E_{bd}}$$

where M is the set of 12 months, O_{bd} is the observed number of people who were born in month b and died in month d, and E_{bd} is the expected number of people who were born in month b and died in month d. The χ^2 statistic for this table is 115.6 suggesting no association between birthday and deathday. The next question is whether this value is statistically significant or in other words whether the value large enough to imply that the sample data is statistically significantly different from the expected data.

One way of determining whether the χ^2 statistic for this table is statistically significant is to compare it to the χ^2 statistic for all the other possible tables with

the same row and columns sums as the sample table (and thus the same expected values as the sample). We define the p-value of a sample of data for this example is the proportion of tables among all those possible tables with the same row and column sums as the sample for which the value of the chi-square statistic equals or exceeds the chi-square statistic for the sample data. So, if the p-value is *small*, then the sample data is statistically significantly different from the expected value and we have strong evidence against the hypothesis that the months of birth and death are independent ([68]).

The traditional method for computing the p-value is to use the chi-square approximation, which is valid when all the cell counts are reasonably large (at least 5). This assumption is certainly not valid for the sample data for our example. Another method, *Fisher's exact test* is equivalent to enumerating all the possible tables with the same row sums and column sums as the sample. The set of all tables with the sample row sums and columns sums as our sample data is the set

$$\mathcal{F} = \{x \in \mathbb{N}^{144} : \sum_{b \in M} x_{bd} = p_b \; \forall b \in M, \sum_{d \in M} x_{bd} = p_d \; \forall d \in M\}$$

where p_b the total number of people born in month b and p_d is the total number of people who died in month d. However, enumerating all the possible contingency tables is really only tractable for small size tables (smaller than a 3×3 table). Indeed, Diaconis and Sturmfels in [30] give an example and a 4×4 contingency table for a sample data set with 592 observations for which there are 1,225,914,276,276,768,514 tables with the same row sums and columns sums as the sample data.

Instead of enumerating all the possible tables, we collect a sample of tables to approximate the p-value. The set of possible tables above $\mathcal{F}$ is a fiber. Let

$$\mathcal{L} = \{x \in \mathbb{Z}^{144} : \sum_{b \in M} x_{bd} = 0 \; \forall b \in M, \sum_{d \in M} x_{bd} = 0 \; \forall d \in M\}$$

and let $\nu = s$ where $s \in \mathbb{Z}^{144}$ is the given sample data. Then, $\mathcal{F} = \mathcal{F}_{\mathcal{L}}(\nu)$. We can therefore construct a sample of all the possible tables in $\mathcal{F}_{\mathcal{L}}(\nu)$ using a Markov basis of this fiber as described in the Section 1.6 in the introduction. The technique for collecting a sample of all the possible tables is called *Markov chain Monte Carlo* (see for example [40]).

Diaconis and Sturmfels in [30] give a histogram of the χ^2 statistic given by sampling from all possible tables with the same column and row sums as in Table 7.1. The histogram also suggests that there is no association between birthday and deathday.

Note that a Markov basis of $\mathcal{L}$ is a Markov basis for all 12×12 tables with any possible set of fixed row sums and fixed column sums. For this application, it is particularly useful to compute a Markov basis of $\mathcal{L}$ because then we only ever need to compute it once for all 12×12 tables that we would ever need. Additionally, in theory, we can apply the techniques in this section for two dimensional contingency tables of any size, and moreover, we can extend this techniques for contingency tables in higher dimensions.

Using our Project-and-Lift algorithm, we were able to solve a computational challenge posed by Seth Sullivant concerning three dimensional $4 \times 4 \times 4$ contingency

tables with fixed row sums in the x, y and z direction. This is a problem involving 64 variables. The challenge posed by Seth Sullivant amounts to checking whether a given set of $145,512$ integer vectors in $\mathbb{Z}^{64}$ is a Markov basis of three dimensional $4 \times 4 \times 4$ contingency tables with every possible variation of fixed rows sums in every direction.

Specifically, we computed a minimal Markov basis of the lattice

$$\begin{aligned} \sum_{i=1}^{4} x_{ijk} &= 0 \qquad \text{for } j,k = 1,\ldots,4, \\ \sum_{j=1}^{4} x_{ijk} &= 0 \qquad \text{for } i,k = 1,\ldots,4, \\ \sum_{k=1}^{4} x_{ijk} &= 0 \qquad \text{for } i,j = 1,\ldots,4. \end{aligned}$$

Computing an entire Markov basis for this problem was previously intractable with only partial results previously known. Aoki and Takemura in [4] had computed these $145,512$ and claimed that these vectors formed a minimal Markov basis. However, we managed to compute a complete minimal Markov basis, which has $148,968$ vectors. We computed the Markov basis within less than 7 days on a Sun Fire V890 Ultra Sparc IV processor with 1200 MHz.

The $148,968$ vectors of a minimal Markov basis actually fall into 15 different equivalence classes due to the inherent symmetry of the problem; for example, we can permute the rows in the table and still have a valid $4 \times 4 \times 4$ table. We currently are not able to use this symmetry to speed up the computation. This leaves room for significant further improvement of the Project-and-Lift method if it could also take symmetry into account.

7.2 Normality of semigroups

Another application of a Markov basis of a lattice concerns the normality of the semigroup $\{Ax : x \in \mathbb{N}^n\}$ where $A \subseteq \mathbb{Z}^{m \times n}$ such that the columns of A span $\mathbb{Z}^m$ ($\{Ax : x \in \mathbb{Z}^n\} = \mathbb{Z}^m$). We say that this semigroup is **normal** if $\{Ax : x \in \mathbb{N}^n\} = \{Ax : x \in \mathbb{R}^n_+\} \cap \mathbb{Z}^n$. A simple example of a semigroup that is not normal is the semigroup given by the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 3 \end{bmatrix}.$$

For this matrix, we have $(1,1) \in \{Ax : x \in \mathbb{R}^n_+\} \cap \mathbb{Z}^n$, but $(1,1) \notin \{Ax : x \in \mathbb{N}^n\}$, so $\{Ax : x \in \mathbb{N}^n\}$ is not a normal semigroup. The question of whether a semigroup is normal or not relates to the linear integer feasibility problem (see [82]). The semigroup normality question also arises in other areas such as statistical learning theory (see [53]).

There is an algorithm for determining whether a semigroup is normal or not (see [54]), but this algorithm involves computing the Hilbert basis of $\{Ax : x \in \mathbb{R}^n_+\} \cap \mathbb{Z}^n$, which may be computationally intractable. Instead, by analysing the vectors in a Markov basis of $\mathcal{L}_A = \{x \in \mathbb{N}^n : Ax = 0\}$, we may find evidence that the semigroup $\{Ax : x \in \mathbb{N}^n\}$ is *not* normal. Note that if we do not find such evidence, then we cannot conclude that the semigroup is normal.

The following lemma gives a criterion for when a semigroup is not normal ([72]). We will show after the lemma how we can use the criterion with Markov bases. For the lemma, we need the concept of an *indispensable* vector. We say that a vector $u \in \mathcal{L}_A$ is **indispensable** if $\mathcal{F}_{\mathcal{L}_A}(u^+) = \{u^+, u^-\}$.

Lemma 7.2.1. *Let $A \subseteq \mathbb{Z}^{m \times n}$. The semigroup $\{Ax : x \in \mathbb{N}^n\}$ is not normal if there exists an indispensable vector $u \in \mathcal{L}_A$ such that $u_i > 1$ and $u_j <$ -1 for some $i, j \in \{1, ..., n\}$.*

Proof. Let $u \in M$ such that u is indispensable and $u_i > 1$ and $u_j <$ -1 for some $i, j \in \{1, ..., n\}$. We can thus write $u^+ = (x+2\mathbf{e}^i)$ for some $x \in \mathbb{N}^n$ and $u^- = (y+2\mathbf{e}^j)$ for some $y \in \mathbb{N}^n$. Also, since u is indispensable, the only two points in $\mathcal{F}_{\mathcal{L}_A}(x + 2\mathbf{e}^i)$ are $(x+2\mathbf{e}^i)$ and $(y+2\mathbf{e}^j)$. Note that since $Au = 0$, we have $A(x+2\mathbf{e}^i) = A(y+2\mathbf{e}^j)$, and thus, $A(x+\mathbf{e}^i-\mathbf{e}^j) = \frac{1}{2}A(x+y)$. We now show that $A(x+\mathbf{e}^i-\mathbf{e}^j) \notin \{Ax : x \in \mathbb{N}^n\}$ and the result follows since $A(x + \mathbf{e}^i - \mathbf{e}^j) = \frac{1}{2}A(x + y) \in \{Ax : x \in \mathbb{R}^n_+\} \cap \mathbb{Z}^n$. Suppose that $A(x + \mathbf{e}^i - \mathbf{e}^j) \in \{Ax : x \in \mathbb{N}^n\}$, so there exists $z \in \mathbb{N}^n$ such that $Az = A(x+\mathbf{e}^i-\mathbf{e}^j)$. Then, $(z+\mathbf{e}^i+\mathbf{e}^j) \in \mathcal{F}_{\mathcal{L}_A}(x+2\mathbf{e}^i)$, but $(z+\mathbf{e}^i+\mathbf{e}^j) \neq (x+2\mathbf{e}^i)$ since $x_j = 0$ and $(z + \mathbf{e}^i + \mathbf{e}^j) \neq (y + 2\mathbf{e}^j)$ since $y_i = 0$, which contradicts that u is indispensable. □

Any indispensable vector $u \in \mathcal{L}_A$ must be in every possible Markov basis of $\mathcal{L}_A$ since otherwise the graph of the fiber $\mathcal{F}_{\mathcal{L}_A}(u^+)$ would be disconnected. Thus, any Markov basis of $\mathcal{L}_A$ must contain every indispensable vector. Not every vector in a Markov basis is indispensable, but we can check whether a vector is indispensable using the other vectors in the Markov basis from the following lemma.

Lemma 7.2.2. *Let M be a Markov basis of $\mathcal{L}_A$ and let $u \in M$. A vector $u \in M$ is indispensable if and only if, for all $v \in M$ where $v \neq u$, we have $v^+ \not\leq u^+$, $v^+ \not\leq u^-$, $v^- \not\leq u^+$, and $v^- \not\leq u^-$.*

Proof. Assume u is not indispensable. Then there exists another point in $\mathcal{F}_{\mathcal{L}_A}(u^+)$ besides u^+ and u^-. Since M is a Markov basis of $\mathcal{L}_A$, there must be a path in the fiber graph from either u^+ or u^- to the other point. Thus, we must be able to move from either u^+ or u^- to the other point using another vector $v \in M$, which means that either $u^+ - v$, $u^- - v$, $u^+ + v$, or $u^- + v$ is in $\mathcal{F}_{\mathcal{L}_A}(u^+)$ or equivalently that either $v^+ \leq u^+$, $v^+ \leq u^-$, $v^- \leq u^+$, or $v^- \leq u^-$.

Conversely, if there exists $v \in M$ such that $u \neq v$ and $v^+ \leq u^+$, then $u^+ - v \in \mathcal{F}_{\mathcal{L}_A}(u^+)$ and $u^+ - v \neq u^-$. An analogous argument holds for $v^+ \leq u^-$, $v^- \leq u^+$, or $v^- \leq u^-$. □

So, to prove that the semigroup $\{Ax : x \in \mathbb{N}^n\}$ is not normal, we compute a Markov basis of $\mathcal{L}_A$ and extract the indispensable vectors from the Markov basis, and then, for each indispensable vector u, we check whether $u_i > 1$ and $u_j <$ -1 for some $i, j \in \{1, ..., n\}$.[1]

7.3 Feasibility of Integer Programs

In this section, we present a new Gröbner basis approach for finding a feasible solution of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$. We will focus on the special case where $\sigma = \emptyset$ (i.e. $\mathcal{F}_{\mathcal{L}}(\nu)$) because it is straight-forward to lift a feasible solution of $\mathcal{F}_{\mathcal{L}^{\sigma}}(\nu_{\bar{\sigma}})$ to a feasible solution of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$.

We show that the Project-and-Lift algorithm from Section 6.1 that computes Markov bases of lattices can be modified to find a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ for any $\nu \in \mathbb{Z}^n$ as well as compute a Markov basis of $\mathcal{L}$. The increase in computation time to compute the feasible solution is trivial in comparison to the time taken to compute the Markov basis. It is important for solving lattice programs using Gröbner bases that we can compute feasible solutions because we need a feasible solution of the lattice program to solve the lattice program using a Gröbner basis. Furthermore, we can also compute a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ for a finite set of $\nu \in \mathbb{Z}^n$ simultaneously while computing a Markov basis of $\mathcal{L}$, or if we only want a feasible solution for one $\nu \in \mathbb{Z}^n$, we can use a modification of the truncated Project-and-Lift algorithm. We will present the case where we only want a feasible solution for one $\nu \in \mathbb{Z}^n$, but the algorithm we present may be easily extended for the more general case of computing feasible solutions for more than one $\nu \in \mathbb{Z}^n$.

It is possible to modify the other algorithms for computing Markov bases to compute a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ for any $\nu \in \mathbb{Z}^n$ as well as compute a Markov basis of $\mathcal{L}$. But, we will only present the modification to the Project-and-Lift algorithm since it is generally the fastest of the Markov basis algorithms.

The basic idea is that, given a feasible solution of $\mathcal{F}_{\mathcal{L}}^{i}(\nu)$ for some $i \in \{1, \ldots, n\}$, we can construct a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ if such a feasible solution exists. Note that $\mathcal{F}_{\mathcal{L}}^{i}(\nu)$ is the same as $\mathcal{F}_{\mathcal{L}}(\nu)$ except that we have removed the non-negativity constraint on the ith component. Hence, for some $\sigma \subseteq \{1, \ldots, n\}$, starting with a feasible solution of $\mathcal{F}_{\mathcal{L}}^{\sigma}(\nu)$ (we choose σ such that this feasible solution is easy to find), we can compute a feasible solution of $\mathcal{F}_{\mathcal{L}}^{\sigma\text{-}i}(\nu)$ for some $i \in \sigma$. By doing this repeatedly for every $i \in \sigma$, we attain a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$.

We now show how to construct a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ from a feasible solution of $\mathcal{F}_{\mathcal{L}}^{i}(\nu)$ for some $i \in \{1, ..., n\}$. Let $\nu \in \mathbb{Z}^n$, and $x \in \mathcal{F}_{\mathcal{L}}^{i}(\nu)$. If $x_i \geq 0$, then we are done. So, assume $x_i < 0$. There are two distinct cases to handle: i is bounded and i is unbounded.

[1]The Markov basis may be too large to compute it in its entirety. Instead, we can compute a partial Markov basis by stopping the Markov basis computation prematurely, and then, we can check whether the vectors in the partial Markov basis are indispensable by using a Hilbert basis computation (see [54] for how to prove a vector is indispensable using a Hilbert basis computation).

Firstly, if i is unbounded, then there exists $u \in \mathcal{L}$ where $u \geq \mathbf{0}$ and $u_i > 0$ from Corollary 2.9.2; therefore, $x + \lambda u$ for some $\lambda \in \mathbb{N}$ is non-negative on the ith component and thus a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$.

Secondly, if i is bounded, then we compute an optimal solution of $IP^i_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu) = \max\{x_i : x - \nu \in \mathcal{L}, x_{\bar{i}} \geq \mathbf{0}\}$, which must have an optimal solution since i is bounded. Let x^* be the optimal solution of $IP^i_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu)$. If $x^*_i < 0$, then $\mathcal{F}_{\mathcal{L}}(\nu) = \emptyset$, otherwise x^* is a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$. To compute an optimal solution of $IP^i_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu)$, we can compute the optimal solution of $IP^i_{\mathcal{L},\succ_i}(\nu)$ using a a ν-truncated $\succ_i$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. More explicitly, given a ν-truncated Markov basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$, we compute a set $G \subseteq \mathcal{L}$ that is a ν-truncated $\succ_i$-Gröbner basis of $\mathcal{F}^i_{\mathcal{L}}(\cdot)$. Then, $x' = \mathcal{NF}_{\mathcal{L}}(x, G)$ is the optimal solution of $IP^i_{\mathcal{L},\succ_i}(\nu)$ and also an optimal solution of $IP^i_{\mathcal{L},\text{-}\mathbf{e}^i}(\nu)$. Conceptually, when computing $\mathcal{NF}_{\mathcal{L}}(x, G)$, we are just maximising the ith component which will thus become non-negative if a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ exists.

In summary, the basic feasibility algorithm for finding a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ is as follows. First, we find a set $\sigma \subseteq \{1, ..., n\}$ where $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$ such that we can find a ν-truncated Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ and a feasible solution of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$. We need the condition $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$ so that we can find term orders of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ (see Section 2.10). Let x be such a feasible solution of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$, and let M be a ν-truncated Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$. Then, we select $i \in \sigma$. If i is unbounded, then we can find a vector $u \in \mathcal{L}$ such that $u_{\bar{\sigma}} \geq \mathbf{0}$ and $u_i > 0$. Then, $x + \lambda u \in \mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\nu)$ for some $\lambda \in \mathbb{N}$. Also, we add u to M so that M is a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ by Lemma 6.2.4. If i is bounded, then using the ν-truncated Markov basis M, we compute a set $G \subseteq \mathcal{L}$ that is a $\succ_i$-Gröbner basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ (i.e. $G = \mathcal{CP}^\sigma_{\mathcal{L},\nu}(\succ_i, M)$), and then, we compute the normal form $x := \mathcal{NF}_{\mathcal{L}}(x, G)$, and if $x_i < 0$, then $\mathcal{F}_{\mathcal{L}}(\nu) = \emptyset$. Also, we set $M = G$ so that M is again ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ by Lemma 6.2.2. Lastly, we set $\sigma := \sigma\text{-}i$ and repeat the above steps until $\sigma = \emptyset$. Note that the set M is a truncated Markov basis for the next iteration by construction. See Algorithm 8 for a description of this feasibility algorithm. We could easily modify this algorithm so that we compute feasible solutions for many different $\nu \in \mathbb{Z}^n$ simultaneously.

To compute a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$, we need to start from a feasible solution of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$ and a ν-truncated Markov basis of $\mathcal{F}^\sigma_{\mathcal{L}}(\cdot)$ for some $\sigma \subseteq \{1, ..., n\}$ where $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$. As before in Chapter 6, we can find a set $\sigma \subseteq \{1, .., n\}$ such that $\ker(\pi_\sigma) \cap \mathcal{L}^\sigma = \{\mathbf{0}\}$: σ is any set of k independent columns of a basis B of $\mathcal{L}$ where $\text{rank}(B) = \mathcal{L}$. Moreover, as in Chapter 6, we can also find a set $S \subseteq \mathcal{L}^\sigma$ such that S is a basis of $\mathcal{L}^\sigma$, and S is an upper triangle square matrix with positive diagonal entries and non-positive entries elsewhere (i.e. S is in Upper Hermite Normal Form – see Section 2.4). Now, the vector $\nu_{\bar{\sigma}}$ is a solution to the relaxation $\mathcal{F}^{\bar{\sigma}}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$ (all non-negativity constraints are removed). Then, we can add appropriate non-negative multiples of the vectors in S to $\nu_{\bar{\sigma}}$ such that it becomes non-negative, and thus, we arrive at a feasible solution of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$. From a feasible solution of $\mathcal{F}_{\mathcal{L}^\sigma}(\nu_{\bar{\sigma}})$, we can compute a feasible solution of $\mathcal{F}^\sigma_{\mathcal{L}}(\nu)$. Also, S is a Markov basis of $\mathcal{L}^\sigma$ as we saw in Example 3.2.5. We can then lift this to a Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$.

The steps in this algorithm are essentially the same steps as those in the truncated Project-and-Lift algorithm (Algorithm 5) for computing a truncated Markov basis. So, at the same time as computing a Markov basis, we can compute a feasible

Algorithm 8 Feasibility algorithm

Input: a lattice $\mathcal{L}$ and a vector $\nu \in \mathbb{Z}^n$.
Output: a feasible solution $x \in \mathcal{F}_{\mathcal{L}}(\nu)$ or infeasible.

```
Find a set σ ⊆ {1,...,n} such that ker(π_σ) ∩ L = {0}.
Compute a set G ⊆ L that is a ν-truncated Markov basis of F^σ_L(·).
Compute a feasible solution x ∈ F^σ_L(ν).
while σ ≠ ∅ do
  Select i ∈ σ
  if i is bounded then
    G := CP^σ_{L,ν}(≻_i, G)
    x := NF^σ_L(x, G)
    if x_i < 0 then return infeasible end if
  else
    Find u ∈ L such that u_σ̄ ≥ 0 and u_i > 0.
    x := x + λu where λ ∈ ℕ such that (x + λu)_i ≥ 0
    G := G ∪ {u}
  end if
  σ := σ-i
  M := {u ∈ M : u^+_σ ∈ B_{L^σ}(ν_σ̄)}
end while
return x.
```

solution.

Example 7.3.1. *We apply the above method to find a feasible solution of equality constrained integer knapsack problems (see [2]):*

$$\mathcal{F} = \{x : Ax = b, x \in \mathbb{N}^n\}$$

where $A \in \mathbb{N}^{1\times n}$ and $b \in \mathbb{N}$. Let $\mathcal{L} = \mathcal{L}_A := \{u : Au = \mathbf{0}, u \in \mathbb{Z}^n\}$. Then $\mathcal{F} = \mathcal{F}_{\mathcal{L}}(\nu)$ where $\nu \in \{x : Ax = b, x \in \mathbb{Z}^n\}$. Finding such a vector ν can be done in polynomial time using the HNF algorithm. If no such ν exists, then the original problem $\mathcal{F}$ is infeasible. Computing a Markov basis and thus solving the feasibility problem of any such knapsack problem involves only one Gröbner basis computation because there are $n-1$ vectors in any lattice basis of $\mathcal{L}_A$ and so the initial size of σ is always one.[2]

Consider the following knapsack feasibility problem:

$$\mathcal{F} := \{x : 12223x_1 + 12224x_2 + 36674x_3 + 61119x_4 + 85569x_5 = 89643481 : x \in \mathbb{N}^5\}.$$

Let $\mathcal{L} = \mathcal{L}_A$ where $A = \begin{bmatrix}12223 & 12224 & 36674 & 61119 & 85569\end{bmatrix}$. The set

$$S = \{(\text{-}12224,12223,0,0,0),(2,\text{-}5,1,0,0),(\text{-}1,\text{-}4,0,1,0),(1,\text{-}8,0,0,1)\}$$

is a basis of $\mathcal{L}$. Let $\nu = (\text{-}4889, 12222, 0, 0, 0)$, then $\mathcal{F}_{\mathcal{L}}(\nu) = \mathcal{F}$. Let $\sigma = \{1\}$. Then, $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$. *The projection of S onto all components except 1 is a Markov*

[2]This Gröbner basis method for computing a feasible solution for the special case of equality constrained integer knapsack problems was found independently by Bjarke H. Roune ([74]), who also shows how this method may be extended to compute the Frobenius number.

basis of $\mathcal{L}^\sigma$*:*

$$\pi_i(S) = S' = \{(12223,0,0,0),(-5,1,0,0),(-4,0,1,0),(-8,0,0,1)\}$$

since this set satisfies Lemma 3.2.4. Thus, S *is a Markov basis of* $\mathcal{F}^\sigma_\mathcal{L}(\cdot)$ *by Lemma 3.1.5. The set* $G =$

$$\{(-7336,3,2444,0,0),(-7334,-2,2445,0,0),(-2,5,-1,0,0),(-1,-4,0,1,0),\ (-1,-3,-1,0,1)\}$$

is a $\succ_i$*-Gröbner basis of* $\mathcal{F}^i_\mathcal{L}(\cdot)$*. We can compute* G *using the completion procedure. Note that* $x = (-4889, 12222, 0, 0, 0) \in \mathcal{F}^i_\mathcal{L}(\nu)$*. The normal form of* x *is* $\mathcal{NF}^i_\mathcal{L}(x, G) = x' = (-1, 2, 2444, 0, 0)$*. This is not a feasible solution of* $\mathcal{F}_\mathcal{L}(\nu)$*, and the problem is therefore infeasible.*

We list the knapsack problems in Table 7.2, and we list the times to solve the feasibility problem in Table 7.3 as well as the size of the Gröbner bases that was computed to solve the knapsack problem. In each case, the problem was infeasible. The right hand sides used for each knapsack was the Frobenius number – the largest infeasible right hand side. These knapsack instances were taken from the paper [2] by Aardal and Lenstra; the first five instances first appeared in the paper [27] by Cornuejols et al. The first fifteen instances were designed such that there is a basis of the lattice $\mathcal{L}_A$ *(*A *is equality constraint matrix) containing one vector which is much longer than the others. The last ten instances were generated randomly in such a way that the determinant of a basis of the lattice* $\mathcal{L}_A$ *was about the same order of magnitude as the first fifteen instances. This may explain why the sizes of the Gröbner bases are small for the first fifteen problems and larger for the remaining ten problems.*

In the paper [2], the feasibility problem is solved for the same set of equality constrained integer knapsack problems by using a reduced lattice basis approach. The solutions times in [2] and our solutions times are all less than a second, so it would be interesting to compare the two methods on larger problems with a significant computation time.

This approach for computing a feasible solution of a fiber could potentially be used when computing a truncated Markov basis by the Project-and-Lift algorithm since during the algorithm, we check whether $\nu' \in \mathcal{B}_\mathcal{L}(\nu)$ which is the feasibility problem $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset$. Note that the feasibility approach is well-suited to computing feasibility for many different fibers simultaneously. It would be interesting to see the performance of this approach.

Lastly, we wish to point out that Hemmecke in [50] proposed an approach for finding feasible solutions using Graver bases. This approach is useful when the size of a Gräver basis is not much larger than the size of a Markov basis.

7.4 Integer programming

In this section, we discuss the use of Gröbner bases to solve the lattice program

$$IP^\sigma_{\mathcal{L},c}(\nu) := \min\{cx : x \in \mathcal{F}^\sigma_\mathcal{L}(\nu)\}.$$

#	Equality constraint (A)										RHS (b)
1	12223	12224	36674	61119	85569						89643481
2	12228	36679	36682	48908	61139	73365					89716838
3	12137	24269	36405	36407	48545	60683					58925134
4	13211	13212	39638	52844	66060	79268	92482				104723595
5	13429	26850	26855	40280	40281	53711	53714	67141			45094583
6	25067	49300	49717	62124	87608	88025	113673	119169			3367335
7	11948	23330	30635	44197	92754	123389	136951	140745			14215206
8	39559	61679	79625	99658	133404	137071	159757	173977			58424799
9	48709	55893	62177	65919	86271	87692	102881	109765			60575665
10	28637	48198	80330	91980	102221	135518	165564	176049			62442884
11	20601	40429	42407	45415	53725	61919	64470	69340	78539	95043	22382774
12	18902	26720	34538	34868	49201	49531	65167	66800	84069	137179	27267751
13	17035	45529	48317	48506	86120	100178	112464	115819	125128	129688	21733990
14	13719	20289	29067	60517	64354	65633	76969	102024	106036	199930	13385099
15	45276	70778	86911	92634	97839	125941	134269	141033	147279	153525	106925261
16	11615	27638	32124	48384	53542	56230	73104	73884	112951	130204	577134
17	14770	32480	75923	86053	85747	91772	101240	115403	137390	147371	944183
18	15167	28569	36170	55419	70945	74926	95821	109046	121581	137695	6765260
19	11828	14253	46209	52042	55987	72649	119704	129334	135589	138360	80230
20	13128	37469	39391	41928	53433	59283	81669	95339	110593	131989	1663281
21	35113	36869	46647	53560	81518	85287	102780	115459	146791	147097	109710
22	14054	22184	29952	64696	92752	97364	118723	119355	122370	140050	752109
23	20303	26239	33733	47223	55486	93776	119372	136158	136989	148851	783879
24	20212	30662	31420	49259	49701	62688	74254	77244	139477	142101	677347
25	32663	41286	44549	45674	95772	111887	117611	117763	141840	149740	1037608

Table 7.2: Hard Knapsack Constraint Instances.

Recall from Section 2.9 that if $IP^{\sigma}_{\mathcal{L},c}(\nu)$ has an optimal solution (which can be determined using Linear Programming methods), then we can reformulate the problem as $IP_{\mathcal{L}^{\sigma},c_{\bar{\sigma}}}(\nu_{\sigma})$ (assuming without loss of generality that $c_{\sigma} = \mathbf{0}$), and any optimal solution of $IP_{\mathcal{L}^{\sigma},c_{\bar{\sigma}}}(\nu_{\sigma})$ may be lifted to an optimal solution of $IP^{\sigma}_{\mathcal{L},c}(\nu)$. So, we restrict our attention to the special case where $\sigma = \emptyset$.

The most straight-forward way to solve $IP_{\mathcal{L},c}(\nu)$ using Gröbner basis methods is to solve $IP_{\mathcal{L},\succ_c}(\nu)$ for some term order $\succ$ by first computing a Markov basis of $\mathcal{L}$ and a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ and then computing a $\succ_c$-Gröbner basis of $\mathcal{L}$ for some term order $\succ$, and finally, computing the normal form of the feasible solution (see Algorithm 1) giving the optimal solution. With this method, if we want to solve $IP_{\mathcal{L},c}(\nu)$ for a finite set of ν, we only need to compute a feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ for every ν in the set and redo the normal form computation without needing to recompute the Markov basis or the Gröbner basis. Note that the feasible solutions can be computed at the same time as computing the Markov basis without much additional computational overhead (see Section 7.3).

If we wish to solve $IP_{\mathcal{L},\succ_c}(\nu)$ for just one ν, then we should use information specific to that fiber to solve the problem. Towards this aim, we can compute a ν-truncated

#	1	2	3	4	5	6	7	8	9	10	11	12	13
Time	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.01	0.30	0.00	0.01
Size	5	15	16	7	27	50	17	47	76	51	92	40	49
#	14	15	16	17	18	19	20	21	22	23	24	25	
Time	0.01	0.05	0.48	0.32	0.78	0.23	0.17	0.75	0.22	0.51	0.29	0.45	
Size	43	94	514	619	600	407	549	496	394	514	581	782	

Table 7.3: Hard Knapsack Constraint Instance Timings.

Markov basis $\mathcal{L}$ and a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{L}$. This method is still not satisfactory in many cases because a ν-truncated Gröbner basis of $\mathcal{L}$ may still be prohibitively large thus making this approach undesirable. We must therefore search for new ways of reducing the size of a truncated Gröbner basis. There are two ways in which we can decrease the size of a truncated Gröbner basis.

The first way to reduce the size of a truncated Gröbner basis is to relax some of the non-negativity constraints of $IP_{\mathcal{L},\succ_c}(\nu)$ such that solving the relaxation still solves $IP_{\mathcal{L},\succ_c}(\nu)$. That is, we find $\sigma \subseteq \{1,...,n\}$, such that an optimal solution of $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ is a feasible solution of $IP_{\mathcal{L},\succ_c}(\nu)$ and thus optimal. The idea being that a $\succ_c$-truncated Gröbner basis of the relaxation $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ is hopefully much smaller than a $\succ_c$-truncated Gröbner basis of $\mathcal{L}$ – it is certainly no larger.

Some of the non-negativity constraints may be redundant, so we can relax them without changing the set of feasible solutions, and therefore, solving the relaxation solves the original problem. Somewhat counter intuitively and rather unfortunately, redundant constraints do affect the size of a truncated Gröbner basis. The reason for this is that, although a non-negativity constraint may be redundant for $\mathcal{F}_{\mathcal{L}}(\nu)$, it is usually not redundant for $\mathcal{F}_{\mathcal{L}}(\nu')$ for all $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$, and a ν-truncated Gröbner basis of $\mathcal{L}$ must be a Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu')$ for all $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$.

Irredundant constraints that are not *active* for $IP_{\mathcal{L},\succ_c}(\nu)$ – they do not affect the feasibility of points *near* the optimal solution – may also be relaxed without essentially changing the problem. Again, such constraints do affect the size of a truncated Gröbner basis since, although they are not active for $IP_{\mathcal{L},\succ_c}(\nu)$, they are usually active for some problems $IP_{\mathcal{L},\succ_c}(\nu')$ for which $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu)$.

Remark 7.4.1. *The standard view is that it is the integrality constraints on the variables that make the problem $IP_{\mathcal{L},c}(\nu)$ difficult to solve since the linear relaxation of $IP_{\mathcal{L},c}(\nu)$ can be solved efficiently using standard linear programming techniques. So, standard techniques relax the integrality constraints when trying to solve $IP_{\mathcal{L},c}(\nu)$ as discussed in Section 1.4. We however take a slightly different view that it is the non-negativity constraints on the x variables that make the problem difficult; hence, it is these constraints that we relax.*

The difficulty here is to find a set σ such that the optimal solution of $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ is feasible and thus optimal for $IP_{\mathcal{L},\succ_c}(\nu)$. The solution is to solve a hierarchy of relaxations. We start from some initial relaxation $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ for some $\sigma \subseteq \{1,...,n\}$ such that $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ has an optimal solution $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$. We compute the optimal solution of $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ by first computing a ν-truncated Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ and a feasible

solution $x \in \mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, and then, from the Markov basis, computing a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, and finally, using the Gröbner basis, computing the normal form of x giving the optimal solution x^* of $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$. If x^* is feasible for $IP_{\mathcal{L},\succ_c}(\nu)$ (i.e. $x^* \geq \mathbf{0}$), then we are done. Otherwise, we add a constraint that is violated by the optimal solution of the relaxation (i.e. if $x^*_i < 0$, then we set $\sigma = \sigma\text{-}i$), and repeat. The algorithm must terminate with a solution because in the worst case we end up solving the original problem $IP_{\mathcal{L},\succ_c}(\nu)$ (i.e. $\sigma = \emptyset$). In each iteration of the above approach, we need to compute a truncated Markov basis, a feasible solution, and a truncated Gröbner basis. In a similar fashion to the Project-and-Lift algorithm (Algorithm 4) and the feasibility algorithm (Algorithm 8), we can compute the truncated Markov basis and the feasible solution incrementally as opposed to computing them from scratch for each iteration.

This iterative approach, solving a hierarchy of relaxations, for solving lattice programs or integer programs was certainly previously known (see for example [94, 59, 87]). Our contribution to the approach is to show that we can use Gröbner bases to solve the relaxations in an effective way by computing Gröbner bases, Markov bases, and feasible solutions for successive iterations incrementally; that is, we do not need to compute the Gröbner basis, Markov basis, and feasible solution for each relaxation from scratch each time.

The relaxation that we start with is called a *group relaxation* (introduced by Gomory in [44]). Let B be a basis of $\mathcal{L}$, and let $\sigma \subseteq \{1, ..., n\}$ be $k = \text{rank}(B)$ linearly independent columns of B such that $IP^{\sigma}_{\mathcal{L},c}(\nu)$ has an optimal solution. The program $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ is called a group relaxation. Such a set σ can be found using the simplex algorithm by solving the linear relaxation of $IP_{\mathcal{L},c}(\nu)$.[3] Recall from Example 3.2.5 that it is straight-forward to compute a Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, and also, from Section 7.3, it is straight-forward to compute a feasible solution of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$. Thus, we can compute a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$, and then, compute the optimal solution x^* of $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$. If the optimal solution x^* of $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ is not feasible and thus optimal for $IP_{\mathcal{L},\succ_c}(\nu)$, then we need to add some non-negativity constraints to solve the problem. We call these relaxations created by adding non-negativity constraints to the group problem an *extended group relaxation*.

An iteration of the optimisation algorithm proceeds as follows. We assume that at the start of the iteration we have a set $\sigma \subseteq \{1, ..., n\}$ such that $\ker(\pi_\sigma) \cap \mathcal{L} = \{\mathbf{0}\}$, a ν-truncated Markov basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$, and a feasible solution of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\nu)$. We can assume this in the first iteration as discussed in the previous paragraph, and we will ensure it is true in the next iteration; thus by induction, it is true for every iteration. First, we compute a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{L}$: $G := \mathcal{CP}^{\sigma}_{\mathcal{L},\nu}(\succ_c, M)$. Then, we compute the normal form of x thus solving $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$: $x := \mathcal{NF}^{\sigma}_{\mathcal{L}}(x, G)$. If the optimal solution x^* of $IP^{\sigma}_{\mathcal{L},\succ_c}(\nu)$ is feasible for $IP_{\mathcal{L},\succ_c}(\nu)$ ($x^* \geq \mathbf{0}$), then we have found the optimal of $IP_{\mathcal{L},\succ_c}(\nu)$. Otherwise, we next compute a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ and a feasible solution of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\nu)$ as in the feasibility algorithm (Algorithm 8) as follows. If i is unbounded, then we can find a vector $u \in \mathcal{L}$ such that $u_{\bar{\sigma}} \geq \mathbf{0}$ and $u_i > 0$. Then, $x + \lambda u \in \mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\nu)$ for some $\lambda \in \mathbb{N}$. Also, we add u to M so that M is a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ by Lemma 6.2.4. If i is

[3] Here, σ is the set of basic variables of the optimal tableau.

bounded, from M, we compute a set $G \subseteq \mathcal{L}$ that is a ν-truncated $\succ_i$-Gröbner basis of $\mathcal{F}^{\sigma}_{\mathcal{L}}(\cdot)$ (i.e. $M = \mathcal{CP}^{\sigma}_{\mathcal{L},\nu}(\succ_i, M)$), and then, M is a ν-truncated Markov basis of $\mathcal{F}^{\sigma\text{-}i}_{\mathcal{L}}(\cdot)$ by Lemma 6.2.2. Also, we compute the normal form $x := \mathcal{NF}_{\mathcal{L}}(x, G)$, and if $x_i < 0$, then $\mathcal{F}_{\mathcal{L}}(\nu) = \emptyset$. So, we have computed a Markov basis and a feasible solution for the next iteration for $\sigma := \sigma\text{-}i$.

See Algorithm 9 for a description of the optimisation algorithm. At each iteration, we must select the next i. An obvious choice is to select the component of the optimal solution for the previous iteration with the most violated non-negativity constraint; that is, the most negative component.

Algorithm 9 Optimisation algorithm

Input: an integer program $IP_{\mathcal{L},c}(\nu)$.
Output: an optimal solution $x \in \mathcal{F}_{\mathcal{L}}(\nu)$ or *infeasible*.

```
Find a set σ ⊆ {1,...,n} such that ker(π_σ) ∩ L = {0}.
Compute a set M ⊆ L that is a ν-truncated Markov basis of F^σ_L(·).
Compute a feasible solution x ∈ F^σ_L(ν).
loop
  G := CP^σ_{L,ν}(≻_c, M)
  x := NF^σ_L(x, G)
  if x ≥ 0 then return x end if
  Select i ∈ σ such that x_i < 0.
  if i is bounded then
    M := CP^σ_{L,ν}(≻_i, M)
    x := NF^σ_L(x, M)
    if x_i < 0 then return infeasible end if
  else
    Find u ∈ L such that u_σ̄ ≥ 0 and u_i > 0.
    x := x + λu where λ ∈ ℕ such that (x + λu)_i ≥ 0
    M := M ∪ {u}
  end if
  σ := σ-i
end loop
```

The second way to reduce the size of a truncated Gröbner basis is to reduce the size of the feasible set $\mathcal{F}_{\mathcal{L}}(\nu)$ by adding constraints. As a general rule, the size of a truncated Gröbner basis is directly related to the size of $\mathcal{F}_{\mathcal{L}}(\nu)$ because the smaller the feasible set $\mathcal{F}_{\mathcal{L}}(\nu)$ the fewer vectors in a Gröbner basis of a lattice that fit within the feasible set; in some cases, the truncated Gröbner basis is as large as the non-truncated Gröbner basis. At a first glance, this approach of adding constraints to reduce the size of a truncated Gröbner basis is in direct opposition to our earlier approach of relaxing redundant or inactive constraints to reduce the size of a truncated Gröbner basis. However, there is a special type of constraint that can be added to the formulation that will never increase the size of a truncated Gröbner basis; the constraints we refer to are upper bound constraints on the cost – most other constraints will increase the size of a truncated Gröbner basis. So, given some

upper bound $k \in \mathbb{Z}$ on $IP_{\mathcal{L},c}(\nu)$, we reformulate the problem as follows:

$$\begin{aligned}
IP_{\mathcal{L},c}(\nu) :=& \min\{cx : x - \nu \in \mathcal{L}, x \in \mathbb{N}^n\} \\
=& \min\{cx : x - \nu \in \mathcal{L}, cx \leq k, x \in \mathbb{N}^n\} \\
=& \min\{-y : x - \nu \in \mathcal{L}, cx + y = k, x \in \mathbb{N}^n, y \in \mathbb{N}\} + k \\
=& \min\{-y : x - \nu = u, y - (k - c\nu) = -cu, u \in \mathcal{L}, x \in \mathbb{N}^n, y \in \mathbb{N}\} + k \\
=& \min\{-y : (x - \nu, y - (k - c\nu)) = (u, -cu), u \in \mathcal{L}, (x, y) \in \mathbb{N}^{n+1}\} + k \\
=& \min\{-y : (x, y) - (\nu, k - c\nu) \in \mathcal{L}', (x, y) \in \mathbb{N}^{n+1}\} + k \\
=& IP_{\mathcal{L}',c'}(\nu') + k.
\end{aligned}$$

where $\mathcal{L}' := \{(u, \text{-}cu) : u \in \mathcal{L}\} \subseteq \mathbb{Z}^{n+1}$ (which is a lattice), $\nu' := (\nu, k - c\nu)$, and $c' = \text{-}\mathbf{e}^{n+1}$. Thus, $IP_{\mathcal{L},c}(\nu) = IP_{\mathcal{L}',c'}(\nu') + k$. Hopefully, the feasible set $\mathcal{F}_{\mathcal{L}'}(\nu')$ is a lot smaller than $\mathcal{F}_{\mathcal{L}}(\nu)$, and consequently, a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{L}'$ is a lot smaller than a ν-truncated Gröbner basis of $\mathcal{L}$ and thus a lot quicker to compute. How much smaller a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{L}'$ is than a ν-truncated Gröbner basis of $\mathcal{L}$ will depend on the strength of the upper bound. It is potentially just the empty set.

We can combine this approach with the previous approach of solving a hierarchy of relaxations; that is, we use Algorithm 9 to solve $IP_{\mathcal{L}',\succ_{c'}}(\nu')$ with a slight modification that we add the non-negativity constraint on the $(n+1)$th component in the first iteration of Algorithm 9 otherwise it would never be added since it is always satisfied by any optimal solution of a relaxation.

Crucially, as we mentioned above, the size of a minimal ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{L}'$ cannot exceed the size of a minimal ν-truncated $\succ_c$-Gröbner basis, so we are not computing more than before. Consider the set $\mathcal{F}^{n+1}_{\mathcal{L}'}(\nu')$. Note that we have relaxed the non-negativity constraint on the $(n+1)$th component, or in other words, we have relaxed the upper bound constraint. Thus, we have $\pi_{n+1}(\mathcal{F}^{n+1}_{\mathcal{L}'}(\nu')) = \mathcal{F}_{\mathcal{L}}(\nu)$ since $\pi_{n+1}(\mathcal{L}') = \mathcal{L}$ and $\pi_{n+1}(\nu') = \nu$. So, Lemma 4.1.3 implies that a set $G' \subseteq \mathcal{L}'$ is a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{F}^{n+1}_{\mathcal{L}'}(\cdot)$ if and only if $\pi_{n+1}(G') \subseteq \mathcal{L}$ is a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$ since $\succ_c$ and $\succ_{c'}$ are really the same term order. Thus, ν'-truncated $\succ_{c'}$-Gröbner bases of $\mathcal{F}^{n+1}_{\mathcal{L}'}(\cdot)$ are essentially the same as ν-truncated $\succ_c$-Gröbner bases of $\mathcal{F}_{\mathcal{L}}(\cdot)$. Also, by Lemma 6.2.2, if a set $G' \subseteq \mathcal{L}'$ is a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{F}^{n+1}_{\mathcal{L}'}(\cdot)$, then G' is a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{F}_{\mathcal{L}'}(\cdot)$ since $c' = \text{-}\mathbf{e}^{n+1}$. Thus, a minimal ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{F}_{\mathcal{L}'}(\cdot)$ cannot exceed the size of a minimal ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{F}^{n+1}_{\mathcal{L}'}(\cdot)$ or equivalently the size of a minimal ν-truncated $\succ_c$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\cdot)$. However, a G' is a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{F}_{\mathcal{L}'}(\cdot)$ may be smaller than a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{F}^{n+1}_{\mathcal{L}'}(\cdot)$.

Standard methods for integer programming can take advantage of a good feasible solution of $IP_{\mathcal{L},c}(\nu)$, but previous Gröbner basis methods could not. However, we can take advantage of a good feasible solution since a feasible solution gives us an upper bound on $IP_{\mathcal{L},c}(\nu)$ that can be used to strengthen truncation; that is, if we are given a feasible solution $x \in \mathcal{F}_{\mathcal{L}}(\nu)$, then we can set $k = cx - 1$ as an upper bound on the cost. If x is the optimal solution of $IP_{\mathcal{L},c}(\nu)$, then setting $k = cx - 1$, we have $\mathcal{F}_{\mathcal{L}'}(\nu') = \emptyset$. Therefore, a minimal ν'-truncated Markov basis of

$\mathcal{L}'$ is empty and a minimal ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{L}'$ is also empty. So, potentially, computing a ν'-truncated $\succ_{c'}$-Gröbner basis of $\mathcal{L}'$ is a lot more efficient than computing a ν-truncated $\succ_c$-Gröbner basis of $\mathcal{L}$; we only have to compute an empty set to show optimality!

Even if the bound k is not very good and thus does not help truncation much, it is still definitely worthwhile to solve $IP_{\mathcal{L}',c'}(\nu')$ instead of $IP_{\mathcal{L},c}(\nu)$. The reason is that by introducing the constraint $cx \leq k$ into the problem, more components may become *bounded* and thus the Gröbner basis and the Markov basis computations for the extended group relaxations, $IP^{\sigma}_{\mathcal{L}',c'}(\nu')$, are faster (see the section in [52] on Criterion 2).

Example 7.4.2. *Let $\mathcal{L} = \mathcal{L}_A$ for the matrix A given in Example 5.2.4. In the following table, we list the time taken to compute the optimal solution of $IP_{\mathcal{L},c}(\nu)$ for the same five different ν's as in Example 5.2.4 given a feasible solution where the cost is $c = (3, 15, 1, 5, 2, 17, 16, 16, 15, 9, 7, 11, 13)$. The column "Initial σ" gives the starting value of σ (i.e. the value of σ from the group relaxation), and the column "Final σ" gives the final value of σ when the problem was solved. The column labelled "Old Time" gives the solution times in seconds that it took to compute the optimal solution using the previous Gröbner basis method of computing a truncated Markov basis and a truncated Gröbner basis of the original problem and then optimising using the Normal Form algorithm. The column labelled "New Time" gives the solutions times using the techniques in this section. Here, we used the check for truncation using a vector $a \in \mathcal{S}^* \cap \mathbb{R}^n_+$ where $\mathcal{S}^*$ is the smallest linear subspace containing $\mathcal{L}$ (see Section 4.3).*

ν	*Initial* σ	*Final* σ	*Old Time*	*New Time*
ν^1	$\{3, 4, 5, 10\}$	$\{3, 4, 5, 10\}$	0.76	0.28
ν^2	$\{1, 5, 8, 12\}$	$\{1, 5, 8, 12\}$	16.70	0.02
ν^3	$\{3, 4, 5, 10\}$	$\{5, 10\}$	1.25	0.61
ν^4	$\{3, 4, 5, 10\}$	$\{3, 5, 10\}$	13.53	0.46
ν^5	$\{3, 4, 5, 10\}$	$\{3, 4, 5, 10\}$	148.61	0.28

Note how more stable the technique is now than previously, meaning that the technique is not very sensitive to the fiber for which we are computing the optimal solution and the times no longer reflect the size of the feasible set.

All times are using `4ti2` *version 1.3 on a Pentium 4 3.0 GHz machine with 1.0Gb of RAM running Linux.*

Another possible improvement to the algorithm above would be to try and improve the given feasible solution $x \in \mathcal{F}_{\mathcal{L}}(\nu)$ while running the algorithm to reach a better feasible solution and thus a better upper bound using vectors that we have already computed in intermediate iterations.

If we do not have a good feasible solution or any feasible solution at all available, we can still use the extended formulation. Assume that we are given a lower bound l on the optimal value, which we can always find by solving the linear relaxation. We then try to find a feasible solution by computing a ν'-truncated Markov basis of $\mathcal{L}'$

where $k = l$. If we find a feasible solution, then it must be optimal. Otherwise, we recompute a ν'-truncated Markov basis of $\mathcal{L}'$ where $k = l+1$ and again try to find a feasible solution. We repeat this procedure by incrementing k until we find a feasible solution which must be an optimal solution. This procedure has the advantage that we only compute feasible solutions and not optimal solutions, and thus, we avoid some Gröbner basis computations; that is we avoid computing a truncated Gröbner basis with respect to $\succ_c$.

The re-computation of ν'-truncated Markov bases to find initial solutions initially seems inefficient; however, this is not the case because we can reuse previous computations. Let $\nu' = (\nu, k - c\nu)$ and $\nu'' = (\nu, k + 1 - c\nu)$. Then $\nu' \in \mathcal{B}_{\mathcal{L}}(\nu'')$ since $(\mathbf{0}, 1) \in \mathcal{F}_{\mathcal{L}'}(\nu'' - \nu')$ and so $\mathcal{B}_{\mathcal{L}}(\nu') \subseteq \mathcal{B}_{\mathcal{L}}(\nu'')$. Hence, to compute a ν''-truncated Gröbner basis or Markov basis of $\mathcal{L}$ requires also computing a ν'-truncated Gröbner basis or Markov basis respectively anyway. This applies not only at the final stage of the algorithm for each value of k, but also at each intermediate stage for the extended group relaxations of $IP_{\mathcal{L}',c'}(\nu')$. So, it requires storing all the intermediate stages because we may need to reuse them. This algorithm is much more complex than the first algorithm presented in this section. We have not implemented such an approach yet. It would be interesting to see how it performs. Given a very good initial feasible solution, we would expect that the previous method is faster.

7.5 Enumeration

In this section, we describe approaches for enumerating the points of a fiber $\mathcal{F}_{\mathcal{L}}(\nu)$.

Markov bases can be used to enumerate all the feasible points of a finite set $\mathcal{F}_{\mathcal{L}}(\nu)$. Let M be a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$. Since the graph $\mathcal{G}_{\mathcal{L}}(\nu, M)$ is connected, we can enumerate all feasible points in $\mathcal{F}_{\mathcal{L}}(\nu)$ by traversing the graph $\mathcal{G}_{\mathcal{L}}(\nu, M)$ using any standard graph traversal algorithm such as the breadth-first search or depth-first search method starting from some initial feasible point $x \in \mathcal{F}_{\mathcal{L}}(\nu)$. We assume that we are given a feasible solution $x \in \mathcal{F}_{\mathcal{L}}(\nu)$, but as shown in Section 7.3, a feasible solution can easily be computed as a by-product of computing a Markov basis of $\mathcal{F}_{\mathcal{L}}(\nu)$.

Using a set G that is a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$ for some term order $\succ$, we can also enumerate all feasible points as above since a Gröbner basis is also a Markov basis (see Corollary 4.1.7). Moreover, using the properties of a Gröbner basis, we can enumerate all the feasible solutions in a $\succ$-increasing order starting from the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$. This enumeration method can be useful for solving integer programs. Consider the integer program $IP := \min_{\succ}\{x : x \in \mathcal{F}_{\mathcal{L}}(\nu) \cap Q\}$ where $Q \subseteq \mathbb{N}^n$ represents some additional complicating constraints which may be non-linear. The idea is that we can solve IP by solving the relaxation $IP_{\mathcal{L},\succ}(\nu) := \min_{\succ}\{x : x \in \mathcal{F}_{\mathcal{L}}(\nu)\}$, and if the optimal solution $x^* \in IP_{\mathcal{L},\succ}(\nu)$ is feasible for the original problem IP (i.e. $x^* \geq \mathbf{0}$), then we are done, otherwise we enumerate the feasible points of $IP_{\mathcal{L},\succ}(\nu)$ in a $\succ$-increasing order starting from the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$, and the first solution that we encounter that is also feasible for IP must be the optimal solution of IP. In doing this, we hope that the relaxation $IP_{\mathcal{L},\succ}(\nu)$ is much easier to solve than the original problem IP, that determining whether $x \in Q$

is relatively inexpensive, and that the optimal solution to the original problem IP is *not too far away* from the optimal solution of the relaxation $IP_{\mathcal{L},\succ}(\nu)$.

This approach is due to Tayur, Thomas, and Natraj in [83], who applied this approach for scheduling different jobs on different machines with setup costs and where the demands for the different jobs' outputs are correlated random variables. We can also possibly use this approach with the Optimisation Algorithm 9, since at intermediate stages of the algorithm, we could enumerate the feasible points of a group relaxation in a $\succ$-increasing order to try and find an optimal solution for the original problem.

Recall that, from Lemma 4.1.6, there exists a $\succ$-decreasing path from every feasible solution of $\mathcal{F}_{\mathcal{L}}(\nu)$ to the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$. Or in other words, there is a $\succ$-increasing path from the optimal solution to every feasible solution. So, it follows that, in the graph traversal algorithm, we can process the nodes of $\mathcal{G}_{\mathcal{L}}(\nu, M)$ (the feasible points of $\mathcal{F}_{\mathcal{L}}(\nu)$) in a $\succ$-increasing order starting from the optimal solution.

The algorithm works as follows (see Algorithm 10). We use two sets: a set F to store the feasible solutions of $\mathcal{F}_{\mathcal{L}}(\nu)$ that we have processed and a set H to store the feasible solutions that we need to process. At the beginning $F = \emptyset$, and $H = \{x^*\}$ where x^* is the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$. Then, while $H \neq \emptyset$, we repeat the following steps: firstly, we find the $\succ$-minimal point in H ($x := \min_{\succ}\{x \in H\}$), remove it from H ($H := H \setminus x$), and add it to F ($F := F \cup x$); and secondly, we add to H all the feasible solutions x' that are adjacent to x in the graph $\mathcal{G}_{\mathcal{L}}(\nu, G)$ where $x' \succ x$ ($H := H \cup \{x + u : u^- \leq x, u \in G\}$). Then, we are done.

Algorithm 10 Enumeration Algorithm

Input: a set G a $\succ$-Gröbner basis of $\mathcal{F}_{\mathcal{L}}(\nu)$, and x^* the optimal solution of $IP_{\mathcal{L},\nu}()$.
Output: a set $F = \mathcal{F}_{\mathcal{L}}(\nu)$.

```
F := ∅.
H := {x*}.
while H ≠ ∅ do
    x := min_≻{x ∈ H}.
    F := F ∪ x.
    H := H \ x.
    H := H ∪ {x + u : u⁻ ≤ x, u ∈ G}.
end while
return F
```

As we indicated above, the set $\{x + u : u^- \leq x, u \in G\}$ is the set of all the feasible solutions x' that are adjacent to x in the graph $\mathcal{G}_{\mathcal{L}}(\nu, G)$ where $x' \succ x$. We can show this as follows. Firstly, if x' is adjacent to x in $\mathcal{G}_{\mathcal{L}}(\nu, G)$ and $x' \succ x$, then there exists a vector $u \in G$ such that $x' - x = u$ ($x' = x + u$), in which case, we must have $u^- \leq x'$. Secondly, for some $u \in G$, if $u^- \leq x$, then $x + u$ is feasible and adjacent to x in $\mathcal{G}_{\mathcal{L}}(\nu, G)$, and also, $x + u \succ x$ since $u^+ \succ u^-$.

The algorithm is correct because, at the start of each iteration of the while loop, we have $x \succ x^F$ for every $x \in \mathcal{F}_{\mathcal{L}}(\nu) \setminus F$ and $x^F \in F$, so it follows that we add points to F in a $\succ$-increasing order starting from the optimal solution of $IP_{\mathcal{L},\succ}(\nu)$. This is certainly true in the first iteration since $F = \emptyset$, and it is true for the second

iteration since $F = \{x^*\}$. So, we assume it is true for the current iteration (at least the second iteration) and we show it is true for the next iteration. If $\mathcal{F}_{\mathcal{L}}(\nu) = F$, then we are done, otherwise let $x := \min_{\succ}\{x \in \mathcal{F}_{\mathcal{L}}(\nu) \setminus F\}$. Since G is a $\succ$-Gröbner basis and x is non-optimal, there exists $u \in G$ such that $x \succ x - u \in \mathcal{F}_{\mathcal{L}}(\nu)$. By construction, $x - u$ is adjacent to x, and by assumption, $x - u \in F$; thus, we must have $x \in H$ and $x = \min_{\succ}\{x \in H\}$. Therefore, at the start of the next iteration, we have $F := F \cup x$, and the property that $x \succ x^F$ for every $x \in \mathcal{F}_{\mathcal{L}}(\nu) \setminus F$ and $x^F \in F$ still holds as required.

This completes the part of the thesis devoted to Gröbner bases and Markov bases. Next we consider computing extreme rays of cones and circuits of matrices.

Chapter 8

Computing extreme rays

The next two chapters concern extreme rays of polyhedral cones and circuits of matrices. They are completely separate from the previous chapters on Markov bases and Gröbner bases. In this chapter, we present an algorithm to compute the extreme rays of a pointed cone

$$\mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}.$$

where $A \in \mathbb{R}^{m\times n}$, $\sigma \subseteq \{1, ..., m\}$ and $\bar{\sigma} = \{1, ..., m\} \setminus \sigma$. Note that A_σ is the submatrix of A consisting of the rows indexed by σ, and $A_{\bar{\sigma}}$ is the submatrix of A consisting of the rows of A indexed by $\bar{\sigma}$; that is, we have partitioned the rows of the matrix A into two submatrices A_σ and $A_{\bar{\sigma}}$. So, σ refers to the rows of the matrix A that give inequality constraints, and $\bar{\sigma}$ refers to the rows of the matrix A that give equality constraints. The cone $\mathcal{C}_\sigma(A)$ is thus the intersection of a linear space $\mathcal{S}(A_{\bar{\sigma}}) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}\}$ and the cone $\mathcal{C}(A_\sigma) = \{x \in \mathbb{R}^n : A_\sigma x \geq \mathbf{0}\}$.

The problem of computing the extreme rays of $\mathcal{C}_\sigma(A)$ has been studied considerably. Despite this, to the best of our knowledge, it is an open problem whether there exists a extreme ray enumeration algorithm that runs in polynomial time in both the input size (the number of bits required to encode the constraint matrix) and the output size (the number of bits required to encode the size of the set of extreme rays) (see [6]). Note that it is impossible that an algorithm exists that runs in polynomial time in the just input size since the number of extreme rays of a d-dimensional cone with m constraints is $\mathcal{O}(m^{\lfloor \frac{d}{2} \rfloor})$ and this bound is achieved (see [66, 67, 47]).

Recall from Section 2.3 that computing the vertices and extreme rays of a d-dimensional polyhedron is equivalent to computing the extreme rays of a $d+1$-dimensional cone, which makes algorithms for computing extreme rays very versatile. Interestingly, it was shown by Khachiyan et al. in [62] that no polynomial time (in input and output size) algorithm exists to compute just the vertices of a polyhedron without the extreme rays of the polyhedron. From Section 2.3, computing the vertices of a d-dimensional polyhedron without the extreme rays is equivalent to computing the extreme rays of a $d+1$-dimensional cone that are strictly positive in one component. Note that here the output size is the size of the set of vertices not including the extreme rays, so this result does not preclude a polynomial time algorithm (in input and output size) for computing the extreme rays of a cone since

a polyhedron may have exponentially many extreme rays but only polynomially many vertices, and thus, a polynomial time algorithm for computing both vertices and extreme rays may not be a polynomial time algorithm for computing just the vertices.

There are two main classes of algorithms for the extreme ray enumeration problem: insertion algorithms and pivoting algorithms. The algorithm that we focus one here fits within the class of insertion algorithms, but before presenting this algorithm, we will give a brief overview of existing algorithms for computing extreme rays of cones.

We first describe the class of pivoting algorithms. For a d-dimensional cone, a *basis* is a set $\tau \subseteq \sigma$ of $d-1$ constraints where $|\tau| = d-1$ such that $F := \mathcal{C}_{\sigma\setminus\tau}(A)$ is a one-dimensional face of $\mathcal{C}_\sigma(A)$ (i.e. F is generated by an extreme ray of $\mathcal{C}_\sigma(A)$). We say that two bases are *adjacent* if they differ by one constraint, and a *pivot* is the operation of moving from one basis to an adjacent basis; that is, we add one constraint into the basis and we remove one constraint from the basis. This pivoting operation is the same as in the simplex algorithm. A pivoting algorithm begins by finding a basis of some initial extreme ray of the cone $\mathcal{C}_\sigma(A)$, and starting from this initial basis, we can generate all bases of $\mathcal{C}_\sigma(A)$, and thus, all extreme rays of $\mathcal{C}_\sigma(A)$ using the pivot operation. Algorithms that fall into this class are the *gift wrapping algorithm* of Chand and Kapur [17], Seidel's algorithm [75], and the well-known *reverse search algorithm* of Avis and Fukuda [7, 8, 9]. A popular and efficient implementation of the reverse search algorithm is the software package *lrs* [5]. The performance of this class of algorithm depends upon how many different bases there are in comparison to the number of extreme rays. If there is exactly one basis per extreme ray, then pivoting algorithms are polynomial in the input and output size, in which case we say that the cone is *non-degenerate*. However, there may be more than one basis per extreme ray, in which case, we say that the cone is *degenerate*, and for some polytopes, the number of bases is exponential in the input and output size. For these polytopes, the pivoting algorithm first perturbs the cone, either numerically or symbolically, so that there is one basis per extreme ray for the perturbed cone – the perturbed cone is non-degenerate – and the set of bases of the perturbed cone is a subset of the bases of the original cone. The perturbation is performed in such a way that, even though the number of extreme rays has possibly increased, the number of bases in the perturbed cone is hopefully much smaller than the number of bases in the original unperturbed cone. Then, we compute the bases of the perturbed cone, and from these bases, we compute the extreme rays of the original cone removing duplicates if necessary. Avis and Bremmer show in [6] that it is not always possible to find a perturbation such that the number of extreme rays of the perturbed cone is polynomial in the input and output size of the original cone.

We now describe the class of insertion algorithms. Insertion algorithms compute the set of extreme rays of a cone incrementally by computing the extreme rays of a hierarchy of relaxations. Initially, we compute the extreme rays for a relaxation that is a *simple* cone – a simple cone is a d-dimensional cone with d facets. Then, we sequentially add the relaxed constraints incrementally computing the extreme rays for each relaxation. The insertion algorithm we present here is the *double description method*

which was originally discovered by Motzkin et al. [69]. It has been rediscovered many times: the method known as Chernikova's algorithm (see [19]), and the *beneath-and-beyond* method (see [75, 32, 70]) are essentially the same as the double description method.[1] The beneath-beyond method is the dual of the double description method (i.e. computes a constraint representation from a generator representation). The most well-known and widely used implementation of the double description method is the software package `cdd` by Fukuda (see [35]). Another insertion algorithm is the Fourier-Motzkin elimination method, which is more general than the double description method according to Fukuda and Prodon [37]. The software package `PORTA` [20] is an efficient implementation of the Fourier-Motzkin elimination method. Finally, there are the *randomised algorithm* of Clarkson and Shor [21] and the *derandomised algorithm* of Chazelle [18], which rely on triangulation.[2].

The double description method and its variants and the Fourier-Motzkin elimination method perform one insertion step in polynomial time in the size of the cone for the current iteration. However, the method is not necessarily polynomial since the size of a cone for an intermediate iteration can be exponentially larger than the size of the original cone. The choice of insertion order is a crucial factor. Avis and Bremmer show in [6] that there exists a class of cones for which any insertion algorithm that inserts constraints in a fixed order (independent of the input) must construct exponentially large intermediate cones in the worst case. It is an open problem whether there exists an insertion order, which may depend on the input and also on the output of the previous iteration, that would mean that the double description method is polynomial in input and output size. The methods relying upon triangulation have a worst-case exponential complexity in input and output size irrespective of the insertion order chosen and even when the intermediate cones have a small size ([6]).

As a general rule, we found that the reverse search method is the better method in practice for non-degenerate cones or cones with low degeneracy, otherwise the double description method is the better method (also, the Fourier-Motzkin elimination method may be useful in these circumstances).

In this thesis, we will focus on the *double description method*. We present new optimisations for the method, which work well in practice.

8.1 Double Description Method

The fundamental idea behind the double description method is that we can compute the extreme rays of a pointed cone $\mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$ from the set of extreme rays of a pointed cone $\mathcal{C}^i_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_{\sigma\setminus i}x \geq \mathbf{0}\}$ where $i \in \sigma$. The cone $\mathcal{C}^i_\sigma(A)$ is a relaxation of $\mathcal{C}_\sigma(A)$ given by relaxing the constraint $A_i x \geq 0$, or in other words, $\mathcal{C}_\sigma(A) = \mathcal{C}^i_\sigma(A) \cap \{x \in \mathbb{R}^n : A_i x \geq 0\}$. We shall see that the set of extreme rays of $\mathcal{C}_\sigma(A)$ consists of all extreme rays r of $\mathcal{C}^i_\sigma(A)$ that

[1] References were taken from [37] and [6].
[2] References were taken from [6].

are feasible for $\mathcal{C}_\sigma(A)$ (i.e. $A_i r \geq 0$) and some new extreme rays that lie on the hyperplane $\{x \in \mathbb{R}^n : A_i x = 0\}$.

Initially in the double description method, we choose some initial relaxation of $\mathcal{C}_\sigma(A)$ given by relaxing some of the inequality constraints such that the relaxation is still a pointed cone and we can easily find the extreme rays of the relaxation. That is, we choose some $\tau \subseteq \sigma$ such that the cone $\mathcal{C}_\sigma^\tau(A) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_{\sigma \setminus \tau} x \geq \mathbf{0}\}$ is a pointed cone, and we can easily find the extreme rays of $\mathcal{C}_\sigma^\tau(A)$. The cone $\mathcal{C}_\sigma^\tau(A)$ is a relaxation of $\mathcal{C}_\sigma(A)$ given by relaxing the inequality constraints $A_\tau x \geq 0$. Then, we iteratively apply the above fundamental idea: we compute the extreme rays of the cone $\mathcal{C}_\sigma^{\tau\text{-}i}(A) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_{\sigma \setminus \tau} x \geq \mathbf{0}, A_i x \geq 0\}$ from the set of extreme rays of the cone $\mathcal{C}_\sigma^\tau(A)$, and we set $\tau = \tau \setminus i$, and repeat until $\tau = \emptyset$. Note that $\mathcal{C}_\sigma^\tau(A)$ is the relaxation of the cone $\mathcal{C}_\sigma^{\tau\text{-}i}(A)$ given by removing the constraint $A_i x \geq 0$; that is, $\mathcal{C}_\sigma^{\tau\text{-}i}(A) = \mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : A_i x \geq 0\}$.

Before presenting the main result behind the algorithm, Lemma 8.1.5, we first introduce the notion of *adjacency* of extreme rays of pointed cones. This concept will prove to be invaluable since the extreme rays of the cone $\mathcal{C}_\sigma(A)$ are either extreme rays of $\mathcal{C}_\sigma^i(A)$ or created from adjacent extreme rays of $\mathcal{C}_\sigma^i(A)$. This notion of adjacency corresponds exactly with the intuitive notion of what it means for two extreme rays of a cone to be adjacent. For the definition, note that two extreme rays r^1 and r^2 are *distinct* if $\text{supp}_A(r^1) \neq \text{supp}_A(r^2)$ or equivalently $r^1 \neq \lambda r^2$ for any $\lambda \in \mathbb{R}_+$.

Definition 8.1.1. *Two distinct extreme rays r^1 and r^2 of a pointed cone $\mathcal{C}$ are **adjacent** if $F = \{\lambda_1 r^1 + \lambda_2 r^2 : \lambda_1, \lambda_2 \in \mathbb{R}_+\}$ is a face of $\mathcal{C}$, or in other words, r^1 and r^2 generate a face F of $\mathcal{C}$.*

We now discuss some results concerning the adjacency of extreme rays. These results will be pivotal in proving the main result behind the double description method. Also, some of these results are used in the double description method to determine whether two given extreme rays are adjacent, which is a crucial part of the algorithm.

Note that, for two rays r^1 and r^2 of a pointed cone $\mathcal{C}_\sigma(A)$, we have $\text{supp}_A(r^1 + r^2) = \text{supp}_A(r^1) \cup \text{supp}_A(r^2) = \{i \in \{1, ..., m\} : A_i r^1 \neq 0, A_i r^2 \neq 0\}$, and moreover, the face $F = \mathcal{C}_\tau(A)$ where $\tau = \text{supp}_A(r^1 + r^2)$ is the inclusion-minimal face of $\mathcal{C}_\sigma(A)$ containing both rays r^1 and r^2.

Lemma 8.1.2. *Two distinct extreme rays r^1 and r^2 of the pointed cone $\mathcal{C}_\sigma(A)$ are adjacent if and only if $\dim(F) = 2$ where $F = \mathcal{C}_\tau(A)$ and $\tau = \text{supp}_A(r^1 + r^2)$.*

Proof. Assume r^1 and r^2 are adjacent. Then, they generate some face F of $\mathcal{C}_\sigma(A)$. This face has dimension two and must be the inclusion-minimal face containing both r^1 and r^2.

Let $F = \mathcal{C}_\tau(A)$ where $\tau = \text{supp}_A(r^1 + r^2)$. By definition, F is a face of $\mathcal{C}_\sigma(A)$. Assume that $\dim(F) = 2$. Since r^1 and r^2 are extreme rays of $\mathcal{C}_\sigma(A)$, they are also extreme rays of F. The face F is a two-dimensional pointed cone, and thus, it has exactly two extreme rays. So, r^1 and r^2 are the only extreme rays of F, and therefore, by Lemma 2.2.15, r^1 and r^2 generate F: $F = \{\lambda_1 r^1 + \lambda_2 r^2 : \lambda_1, \lambda_2 \in \mathbb{R}_+\}$. □

Combining Lemma 8.1.2 with Lemma 2.2.11, we arrive at the following corollary, which is an important algebraic characterisation of adjacency. This corollary is important because it provides a means for computing whether two given extreme rays are adjacent.

Corollary 8.1.3. *Two distinct extreme rays r^1 and r^2 of the pointed cone $\mathcal{C}_\sigma(A)$ are adjacent if and only if* $\text{rank}(A_{\bar{\tau}}) = n - 2$ *where* $\tau = \text{supp}_A(r^1 + r^2)$.

From Lemma 8.1.2 follows Lemma 8.1.4 below, which is a useful combinatorial characterisation of adjacency. This lemma is important because it also provides a means for computing whether two given extreme rays are adjacent.

Lemma 8.1.4. *Let r^1 and r^2 be distinct extreme rays of a pointed cone $\mathcal{C}_\sigma(A)$. The rays r^1 and r^2 are adjacent if and only if there does not exist another extreme ray r of $\mathcal{C}_\sigma(A)$ distinct from r^1 and r^2 where* $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$.

Proof. Assume r^1 and r^2 are adjacent extreme rays and let r be an extreme ray of $\mathcal{C}_\sigma(A)$ distinct from r^1 and r^2. By saying that r is distinct from r^1 and r^2, we mean that $\text{supp}_A(r) \neq \text{supp}_A(r^1)$ and $\text{supp}_A(r) \neq \text{supp}_A(r^2)$. By Lemma 8.1.2, r^1 and r^2 generate a face F of C: $F = \{\lambda^1 r^1 + \lambda^2 r^2 : \lambda^1, \lambda^2 \in \mathbb{R}_+\}$. Moreover, F is the inclusion-minimal face of $\mathcal{C}_\sigma(A)$ containing r^1 and r^2: $F = \mathcal{C}_\tau(A)$ where $\tau = \text{supp}_A(r^1 + r^2)$. Now, suppose $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$; then, $r \in F$, and therefore, $r = \lambda^1 r^1 + \lambda^2 r^2$ for some $\lambda^1, \lambda^2 \in \mathbb{R}_+$. If $\lambda^1 > 0$, then $\text{supp}_A(r^1) \subseteq \text{supp}_A(r)$, and if $\lambda^2 > 0$, then $\text{supp}_A(r^2) \subseteq \text{supp}_A(r)$. Since we cannot have both $\lambda^1 = 0$ and $\lambda^2 = 0$, either $\text{supp}_A(r^1) \subseteq \text{supp}_A(r)$ or $\text{supp}_A(r^2) \subseteq \text{supp}_A(r)$. But, since r is an extreme ray of $\mathcal{C}_\sigma(A)$, by Lemma 2.2.13, we must have either $\text{supp}_A(r^1) = \text{supp}_A(r)$ or $\text{supp}_A(r^2) = \text{supp}_A(r)$, which is a contradiction.

Conversely, assume that there does not exist an extreme ray r of C distinct from r^1 and r^2 where $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$. Let $F = \mathcal{C}_\tau(A)$ where $\tau = \text{supp}_A(r^1 + r^2)$. Then, r^1 and r^2 are the only extreme rays of $\mathcal{C}_\sigma(A)$ in the face F since $r \in F$ implies $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$. Thus, r^1 and r^2 are the only extreme rays of F, and then, from Lemma 2.2.15, r^1 and r^2 generate F. □

The following lemma, Lemma 8.1.5, is the most important lemma for computing extreme rays. We illustrate the idea behind Lemma 8.1.5 in Figure 8.1. Here, we show how new extreme points of a polytope are created when intersecting a half-space with a polytope. We find that the concepts behind the lemma are better demonstrated using a polytope than using a cone.

Consider the polytope $\mathcal{P}$ in Figure 8.1 given by the vertices a, b, c, d, e, f, g, and h. Consider the affine hyperplane H which intersects $\mathcal{P}$ at a, c, i, and j, and consider the half-space $\mathcal{H}$ defined by H and extending into the page (i.e. d is in $\mathcal{H}$ but b is not in $\mathcal{H}$). The vertices of the new polytope $\mathcal{P} \cap \mathcal{H}$ are a, c, d, e, g, h, i, and j. This list of vertices includes all the vertices of $\mathcal{P}$ that lie in the half-space $\mathcal{H}$ as well as two new vertices i and j created where the defining hyperplane of $\mathcal{H}$ intersects the interior of the edges of $\mathcal{P}$. The vertex i is created by intersecting the edge from e to f with H and the vertex j is created by intersecting the edge from f to g with H.

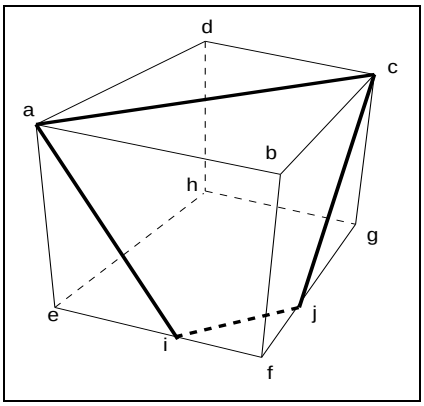

Figure 8.1: The intersection of a polytope $\mathcal{P}$ and a half-space $\mathcal{H}$.

Generalising this observation, for any polytope $\mathcal{P}$ and any half-space $\mathcal{H}$ with affine hyperplane H, the vertices of $\mathcal{P} \cap \mathcal{H}$ are precisely the vertices of $\mathcal{P}$ that lie in $\mathcal{H}$ and also new vertices created by intersecting H with the interior of an edge in $\mathcal{P}$. Recall that to compute the extreme points of a polytope, we embed it in a cone and compute the extreme rays of that cone. The extreme rays of the cone correspond to the extreme points of the polytope. Also, the 2-dimensional faces of the cone correspond to the edges of the polytope (the 1-dimensional faces of the polytope). Thus, applying this observation to cones, we induce that, for any cone $\mathcal{C}$ and any half-space $\mathcal{H}$ with hyperplane H (not affine), the extreme rays of $\mathcal{C} \cap \mathcal{H}$ are precisely the extreme rays of $\mathcal{C}$ that lie in $\mathcal{H}$ and new extreme rays created by intersecting H with the interior of a 2-dimensional face of $\mathcal{C}$. This generalisation is true, and it is exactly what Lemma 8.1.5 says.

More formally, Lemma 8.1.5 gives an exact description of the extreme rays of $\mathcal{C}_\sigma(A)$ from the extreme rays of $\mathcal{C}^i_\sigma(A)$ for some $i \in \sigma$. Recall that $\mathcal{C}^i_\sigma(A)$ is a relaxation of the $\mathcal{C}_\sigma(A)$ given by removing the constraint $A_i x \geq 0$, or in other words, $\mathcal{C}_\sigma(A) = \mathcal{C}^i_\sigma(A) \cap \{x \in \mathbb{R}^n : A_i x \geq 0\}$. Lemma 8.1.5 says that the set of extreme rays of $\mathcal{C}_\sigma(A)$ consists precisely of all the extreme rays r of $\mathcal{C}^i_\sigma(A)$ that are feasible for $\mathcal{C}_\sigma(A)$ (i.e. $A_i r \geq 0$) and all the rays created by intersecting the hyperplane $H = \{x \in \mathbb{R}^n_+ : A_i x = 0\}$ with every two-dimensional face F of $\mathcal{C}^i_\sigma(A)$ where the hyperplane H intersects the interior of the face F. Note that every two-dimensional face F of $\mathcal{C}^i_\sigma(A)$ is generated by two adjacent extreme rays r^1 and r^2, and that H intersects the interior of F if $A_i r^1 > 0$ and $A_i r^2 < 0$ (or vice-versa). Also, the ray created by intersecting F and H is exactly the ray $r = (-A_i r^2) r^1 + (A_i r^1) r^2$.

We can formulate $\mathcal{C}^i_\sigma(A)$ in the form $\mathcal{C}_{\sigma'}(A')$ where $\sigma' = \sigma \setminus i$ and $A' = A_{\overline{i}}$ (the matrix A without the ith row), so we may use all of the previous results on cones. In particular, note that, given a ray $r \in \mathcal{C}^i_\sigma(A) = \mathcal{C}_{\sigma'}(A')$, the support of r is $\text{supp}_{A'}(r) = \text{supp}_A(r) \setminus i$ and the complement of the support is $\overline{\text{supp}}_A(r) \setminus i$.

Lemma 8.1.5. *Given a pointed cone $\mathcal{C}_\sigma(A)$ and $i \in \sigma$ such that $\mathcal{C}^i_\sigma(A)$ is also a pointed cone, a ray $r \in \mathcal{C}_\sigma(A)$ is an extreme ray of $\mathcal{C}_\sigma(A)$ if and only if either r is also an extreme ray of $\mathcal{C}^i_\sigma(A)$ or (exclusively) there exist adjacent extreme rays $r^1, r^2 \in \mathcal{C}^i_\sigma(A)$ such that $A_i r^1 > 0$, $A_i r^2 < 0$ and $r = \lambda((-A_i r^2) r^1 + (A_i r^1) r^2)$ for some $\lambda \in \mathbb{R}_+$.*

Proof. Let $r \in \mathcal{C}_\sigma(A)$ be an extreme ray of $\mathcal{C}_\sigma(A)$. Assume r is not an extreme ray of $\mathcal{C}^i_\sigma(A)$. We must have $A_i r = 0$, otherwise if r' is an extreme ray of $\mathcal{C}^i_\sigma(A)$ such that $\text{supp}_A(r') \setminus i \subsetneq \text{supp}_A(r) \setminus i$, then $\text{supp}_A(r') \subsetneq \text{supp}_A(r)$ contradicting that r is an extreme ray of $\mathcal{C}_\sigma(A)$. The face $F = \mathcal{C}_\tau(A) = \{x \in \mathbb{R}^n : A_{\bar{\tau}} x = \mathbf{0}, A_\tau x \geq \mathbf{0}\}$ where $\tau = \text{supp}_A(r)$ is the inclusion-minimal face of the cone $\mathcal{C}_\sigma(A)$ containing r, and $\dim(F) = 1$ by Lemmas 2.2.10 and 2.2.11. Similarly, the face $F' = \mathcal{C}^i_\tau(A) = \{x \in \mathbb{R}^n : A_{\bar{\tau}\setminus i} x = \mathbf{0}, A_\tau x \geq \mathbf{0}\}$ is the inclusion minimal face of $\mathcal{C}^i_\sigma(A)$ containing r. Since r is not an extreme ray of $\mathcal{C}^i_\sigma(A)$, the face F' must have dimension greater than one by Lemma 2.2.10. Moreover, since $F = F' \cap \{x \in \mathbb{R}^n : A_i x = \mathbf{0}\}$ and F has dimension 1, F' must have dimension 2, so F' is generated by two adjacent extreme rays r^1 and r^2 of $\mathcal{C}^i_\sigma(A)$; thus, $r = \lambda_1 r^1 + \lambda_2 r^2$ for some $\lambda_1, \lambda_2 \in \mathbb{R}_+$ since $r \in F'$. Now, $\text{supp}_A(r^1) \setminus i \neq \text{supp}_A(r) \setminus i$ and $\text{supp}_A(r^2) \setminus i \neq \text{supp}_A(r) \setminus i$ because r is not an extreme ray of $\mathcal{C}^i_\sigma(A)$, so $\lambda_1 \neq \mathbf{0}$ and $\lambda_2 \neq \mathbf{0}$. We must have $A_i r^1 \neq \mathbf{0}$ and $A_i r^2 \neq \mathbf{0}$ since otherwise $r^1 \in \mathcal{C}_\sigma(A)$ and $\text{supp}_A(r^1) \subsetneq \text{supp}_A(r)$ or $r^2 \in \mathcal{C}_\sigma(A)$ and $\text{supp}_A(r^2) \subsetneq \text{supp}_A(r)$ contradicting that r is an extreme ray of $\mathcal{C}_\sigma(A)$. So, since $A_i r = 0$, either $A_i r^1 > 0$ and $A_i r^2 < 0$ or $A_i r^1 < 0$ and $A_i r^2 > 0$. Assume that $A_i r^1 > 0$ without loss of generality. It follows that that $r = \lambda((-A_i r^2) r^1 + (A_i r^1) r^2)$ for some $\lambda \in \mathbb{R}^n$ as required.

We now prove the converse. First, let $r \in \mathcal{C}_\sigma(A)$ be an extreme ray of $\mathcal{C}^i_\sigma(A)$. Then, $\text{supp}_A(r) \setminus i$ is minimal in $\mathcal{C}_\sigma(A)$ implying that $\text{supp}_A(r)$ is minimal in $\mathcal{C}_\sigma(A)$. Therefore, r is an extreme ray of $\mathcal{C}_\sigma(A)$. Second, let r^1 and r^2 be adjacent extreme rays of $\mathcal{C}^i_\sigma(A)$ where $A_i r^1 > 0$ and $A_i r^2 < 0$, and let $r = (-A_i r^2) r^1 + (A_i r^1) r^2$. Let r' be a ray of $\mathcal{C}_\sigma(A)$ where $\text{supp}_A(r') \subseteq \text{supp}_A(r)$. This implies that $\text{supp}_A(r') \setminus i \subseteq \text{supp}_A(r) \setminus i = \text{supp}_A(r^1 + r^2) \setminus i$, and thus, r' is in the face of $\mathcal{C}^i_\sigma(A)$ generated by r^1 and r^2 because r^1 and r^2 are adjacent extreme rays of $\mathcal{C}^i_\sigma(A)$. Thus, $r' = \lambda_1 r^1 + \lambda_2 r^2$ for some $\lambda_1, \lambda_2 \in \mathbb{R}_+$. However, note that $A_i r' = 0$ since $\text{supp}_A(r') \subseteq \text{supp}_A(r)$ and $A_i r = 0$ by construction. Thus, we must have $\lambda_1 = \lambda(-A_i r^2)$ and $\lambda_2 = \lambda(A_i r^1)$ for some $\lambda \in \mathbb{R}_+$. Hence, $r' = \lambda r$, and therefore, r is an extreme ray of $\mathcal{C}^i_\sigma(A)$ because it is a support-minimal ray. □

We can directly translate this result to the case where we wish to compute the extreme rays of the cone $\mathcal{C}^{\tau\text{-}i}_\sigma(A)$ from the extreme rays of a pointed cone $\mathcal{C}^\tau_\sigma(A)$ where $\tau \subseteq \sigma$, and $i \in \tau$. This is really just a matter of notation. Recall that the cone $\mathcal{C}^\tau_\sigma(A)$ is a relaxation of the $\mathcal{C}_\sigma(A)$ where we have relaxed the inequality constraints indexed by τ; i.e. $\mathcal{C}^\tau_\sigma(A) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_{\sigma\setminus\tau} x \geq \mathbf{0}\}$. Also, $\mathcal{C}^{\tau\text{-}i}_\sigma(A) = \mathcal{C}^\tau_\sigma(A) \cap \{x \in \mathbb{R}^n : A_i x \geq 0\}$. Therefore, if R is the set of extreme rays of $\mathcal{C}^\tau_\sigma(A)$, then the extreme rays of $\mathcal{C}^{\tau\text{-}i}_\sigma(A)$ consists of the extreme rays in R that are feasible for $\mathcal{C}^{\tau\text{-}i}_\sigma(A)$ (i.e. $r \in R$ such that $A_i r \geq 0$) combined with all rays $r = (-A_i r^2) r^1 + (A_i r^1) r^2$ where $r^1, r^2 \in R$, $A_i r^1 > 0$, $A_i r^2 < 0$, and r^1 and r^2 are adjacent in $\mathcal{C}^\tau_\sigma(A)$.

We can now describe the double description method (Algorithm 11) for computing extreme rays of a pointed cone $\mathcal{C}_\sigma(A)$. The algorithm starts by computing a minimal set of extreme rays of $\mathcal{C}^\tau_\sigma(A)$ for some τ such that $\mathcal{C}^\tau_\sigma(A)$ is pointed. Then, using Lemma 8.1.5, we compute the extreme rays of $\mathcal{C}^{\tau\text{-}i}_\sigma(A)$. We then remove i from τ and repeat until $\tau = \emptyset$.

We can formulate $\mathcal{C}^\tau_\sigma(A)$ in the form $\mathcal{C}_{\sigma'}(A')$ where $\sigma' = \sigma \setminus \tau$ and $A' = A_{\bar{\tau}}$ (the

matrix A without the rows indexed by τ), so we may use all of the previous results on cones. In particular, note that, given a ray $r \in \mathcal{C}_\sigma^\tau(A) = \mathcal{C}_{\sigma'}(A')$, the support of r is $\text{supp}_{A'}(r) = \text{supp}_A(r) \setminus \tau$ and the complement of the support is $\overline{\text{supp}}_A(r) \setminus \tau$.

The algorithm is incomplete: firstly, it does not specify how to find an initial $\tau \subseteq \sigma$ such that $\mathcal{C}_\sigma^\tau(A)$ is pointed and such that we can compute the extreme rays of $\mathcal{C}_\sigma^\tau(A)$; secondly, it does not specify how to select $i \in \tau$; and thirdly and lastly, it does not specify how to check that r^1 and r^2 are adjacent. We will address these issues below.

Algorithm 11 Ray Algorithm

Input: a pointed cone $\mathcal{C}_\sigma(A)$
Output: the set R of extreme rays of $\mathcal{C}_\sigma(A)$.
  Find a set $\tau \subseteq \sigma$ such that $\mathcal{C}_\sigma^\tau(A)$ is pointed.
  Compute the minimal set R of extreme rays of $\mathcal{C}_\sigma^\tau(A)$.
  while $\tau \neq \emptyset$ **do**
    Select $i \in \tau$.
    $R^+ := \{r \in R : A_i r > 0\}$.
    $R^- := \{r \in R : A_i r < 0\}$.
    $R := R \setminus R^-$.
    for all $r^1 \in R^+$ and $r^2 \in R^-$ **do**
      if r^1 and r^2 are adjacent in $\mathcal{C}_\sigma^\tau(A)$ **then**
        $r := (-A_i r^2) r^1 + (A_i r^1) r^2$.
        $R := R \cup \{r\}$.
      end if
    end for
    $\tau := \tau \setminus i$.
  end while
  return R

Next, we show how to find a set $\tau \subseteq \sigma$ such that $\mathcal{C}_\sigma^\tau(A)$ is pointed and such that we can find the extreme rays of $\mathcal{C}_\sigma^\tau(A)$ easily. In order to show this, we will first discuss some properties of special types of cones.

Consider the cone $\mathcal{C}(A) := \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$ where $A \in \mathbb{R}^{n \times n}$ and $\text{rank}(A) = n$. We can easily find the extreme rays of cones of this type. Note that $\mathcal{C}(A)$ is a pointed cone because $\text{lin}(\mathcal{C}(A)) := \{x \in \mathbb{R}^n : Ax = \mathbf{0}\} = \{\mathbf{0}\}$ since $\text{rank}(A) = n$. From linear algebra, the matrix A is invertible. Let R be the inverse matrix of A; i.e. $AR = I$. Let $r^i = R_{*i}$, the ith column of the matrix R. Thus, we have $Ar^i = \mathbf{e}^i$, so $r^i \in \mathcal{C}(A)$. Moreover, $\text{supp}_A(r^i) = i$, so r^i is support-minimal, and therefore, r^i is an extreme ray of $\mathcal{C}(A)$ by Lemma 2.2.13. So the columns of R are all extreme rays of $\mathcal{C}(A)$. Moreover, the columns of R are the only possible extreme rays of $\mathcal{C}(A)$ because any ray $r \in \mathcal{C}(A)$ must have $\text{supp}(r^i) \subseteq \text{supp}(r)$ for some $i \in \{1, ..., m\}$. In conclusion, for this special cone, there are n extreme rays, which we can find in polynomial time.

Next, consider the cone $\mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$ for some $\sigma \subseteq \{1, ..., n\}$ where again $A \in \mathbb{R}^{n \times n}$ and $\text{rank}(A) = n$. We can also easily find extreme rays for cones of this type. This cone is also pointed since $\text{rank}(A) = n$. Again let R be the inverse matrix of A, and let $r^i = R_{*i}$. Then, r^i is an extreme ray of $\mathcal{C}_\sigma(A)$

if $i \in \sigma$ for the same reason as above. But, if $i \in \bar{\sigma}$, then r^i is not an extreme ray of $\mathcal{C}_\sigma(A)$ since $r^i \notin \mathcal{C}_\sigma(A)$. Note that, for any ray $r \in \mathcal{C}_\sigma(A)$, we must have $\text{supp}_A(r) \subseteq \sigma$, and thus, $\text{supp}(r^i) \subseteq \text{supp}(r)$ for some $i \in \sigma$. Therefore, the rays r^i where $i \in \sigma$ are the only extreme rays of $\mathcal{C}_\sigma(A)$. So, the columns of $R_{*\sigma}$ are all extreme rays of $\mathcal{C}(A)$. In conclusion, for this cone, there are $|\sigma|$ extreme rays of $\mathcal{C}_\sigma(A)$, which also can be found in polynomial time.

Now, we show how to find a set $\tau \subseteq \sigma$ such that $\mathcal{C}^\tau_\sigma(A)$ is pointed ($\text{rank}(A) = n$) and such that we can find the extreme rays of $\mathcal{C}^\tau_\sigma(A)$ easily in polynomial time. Note that $\mathcal{C}^\tau_\sigma(A)$ is pointed if $\text{rank}(A_{\bar{\tau}}) = n$ since $\mathcal{C}^\tau_\sigma(A) = \mathcal{C}_{\sigma\setminus\tau}(A_{\bar{\tau}})$. Let $k = \text{rank}(A_{\bar{\sigma}})$. We can assume that the matrix $A_{\bar{\sigma}}$ has full row rank (i.e. $|\bar{\sigma}| = k$) because $\mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$, so we can remove any linearly dependent rows of $A_{\bar{\sigma}}$ without changing the cone. Now, let $\tau \subseteq \{1, ..., m\}$ be such that $|\bar{\tau}| = n$, $\bar{\sigma} \subseteq \bar{\tau}$, and $\text{rank}(A_{\bar{\tau}}) = n$. Thus, the cone $\mathcal{C}^\tau_\sigma(A)$ is pointed, and we can find the extreme rays of $\mathcal{C}^\tau_\sigma(A)$ by computing the inverse matrix of $A_{\bar{\tau}}$ as we discussed above. Note that, in this case, there are $n - k$ extreme rays of $\mathcal{C}^\tau_\sigma(A)$, and for each extreme ray r, we have $\text{supp}_A(r) \setminus \bar{\tau} = \mathbf{e}^i$ for some $i \in \sigma \setminus \tau$.

The performance of the algorithm is extremely sensitive to how we select $i \in \tau$, called the insertion order. There are several heuristic approaches that we found useful. There is the *max-cut-off* approach for which we select $i \in \tau$ such that the constraint $A_i r \geq 0$ is violated by the largest number of extreme rays r of $\mathcal{C}^\tau_\sigma(A)$, and also, there is the opposite *min-cut-off* approach for which we select $i \in \tau$ such that the constraint $A_i r \geq 0$ is violated by the smallest number of extreme rays r of $\mathcal{C}^\tau_\sigma(A)$. There is the *lexicographic* approach for which we select the constraints A_i in a lexicographic order. Lastly, there is the *max-intersection* approach for which we select $i \in \tau$ such that we have $A_i r = 0$ for the largest number of extreme rays r of $\mathcal{C}^\tau_\sigma(A)$. The max-cut-off, min-cut-off, and lexicographic approaches were suggested by Fukuda and Prodon in [37], but we came across the max-intersection approach. As a general rule, it is difficult to say which of these orderings will be better for a particular problem. There are instances in which each one of them is better than the others, and it is not obvious why one is better than the others for a particular instance.

There are two approaches for checking whether two extreme rays r^1 and r^2 of $\mathcal{C}^\tau_\sigma(A)$ are adjacent or not. The first approach is to check whether $\dim(F) = 2$ (see Lemma 8.1.2) where $F = \mathcal{C}^\tau_\rho(A) = \{x \in \mathbb{R}^n : A_{\bar{\rho}\setminus\tau}x = \mathbf{0}, A_{\rho\setminus\tau}x \geq \mathbf{0}\}$ and $\rho = \text{supp}_A(r^1 + r^2)$. We check whether $\dim(F) = 2$ by computing the rank of the matrix $A_{\bar{\rho}\setminus\tau}$ since $\dim(F) = n - \text{rank}(A_{\bar{\rho}\setminus\tau})$. We refer to this approach as the *algebraic* approach, The second approach, which we refer to as the *combinatorial* approach, is to check whether there exists another extreme ray r^3 (not r^1 nor r^2) such that $\text{supp}_A(r^3)\setminus\tau \subseteq \text{supp}_A(r^1+r^2)\setminus\tau$. If such a vector exists, then r^1 and r^2 are not adjacent (see Lemma 8.1.4). Our implementation of the extreme ray algorithm only uses integral data, so all rational data is scaled so that it is integral. From our computational experience, the algebraic approach seems better than the combinatorial approach when the input data is small. However, if the input data is large, so large that we must use arbitrary precision arithmetic, then the "combinatorial" approach is better since computing the rank of matrices with arbitrary precision is quite expensive. The effectiveness of these two approaches depends greatly on the efficiency of the implementation. We

discuss how these approaches can be optimised in the next section.

Example 8.1.6. *We present a small example computation of the extreme rays of the cone of magic squares. A magic square is an $n \times n$ grid of non-negative real numbers such that the rows, columns, and diagonals all add up to the same value. For example, the following 3×3 grid is a magic square where each row, column, and diagonal adds up to 3:*

0	*2*	*1*
2	*1*	*0*
1	*0*	*2*

The set of all $n \times n$ magic squares is a cone since adding two magic squares gives another magic square and multiplying all the entries in a magic square by a positive constant also gives another magic square. For this example, we will compute the extreme rays of the cone of magic squares; that is, we will compute a minimal set of magic squares such that every other magic square may be written as a conic combination of the set of magic squares.

More explicitly, the set of $n \times n$ magic squares is the set of all non-negative $n \times n$ matrices $X \in \mathbb{R}_+^{n \times n}$ subject to

$$\sum_{k=1}^{n} X_{1k} = \sum_{j=1}^{n} X_{ij} \ \forall i = 2, ..., n \text{ and } \sum_{k=1}^{n} X_{1k} = \sum_{i=1}^{n} X_{ij} \ \forall j = 1, ..., n,$$

which says that the sum of each row equals the sum of the first row and the sum of each column equals the sum of the first row, and also subject to

$$\sum_{k=1}^{n} X_{1k} = \sum_{i=1}^{n} X_{ii} \text{ and } \sum_{k=1}^{n} X_{1k} = \sum_{i=1}^{n} X_{i(n-i+1)},$$

which says that sum of the two diagonals equals the sum of the first row.

We will compute the extreme rays for the 3×3 case. First, we formulate the set of 3×3 magic squares in a more familiar form. We write the magic square $X \in \mathbb{R}^{3 \times 3}$ in vector form $x \in \mathbb{R}^9$ where $x = (X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32}, X_{33})$. Then, we can formulate the set of 3×3 cones in the following more familiar form: $\mathcal{C} = \{x \in \mathbb{R}^9 : Ax = \mathbf{0}, x \geq \mathbf{0}\}$ where

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

Note that the first row of the matrix A is the equality constraint that says that the first row sum equals the second row sum. We can write the cone $\mathcal{C}$ as $\mathcal{C} = \mathcal{C}_\sigma(A') =$

$\{x \in \mathbb{R}^9 : A'_{\bar{\sigma}}x = \mathbf{0}, A'_\sigma x \geq \mathbf{0}\}$ *where* $\sigma = \{1, ..., 9\}$ *and* $A' \in \mathbb{R}^{(7+9)\times 9}$ *where* $A'_{\bar{\sigma}} = A$ *and* $A'_\sigma = I$. *Note that, for any ray* $r \in \mathcal{C} = \mathcal{C}_\sigma(A')$, *the support of* r *is* $\text{supp}_{A'}(r) = \text{supp}_I(r) = \{i \in \{1, ..., 9\} : r_i \neq \mathbf{0}\}$.

We will use the combinatorial check for adjacency (see Lemma 8.1.4). For this particular cone $\mathcal{C} = \mathcal{C}_\sigma(A')$, *two extreme rays* r^1 *and* r^2 *of* $\mathcal{C}_\sigma(A')$ *are adjacent if and only if there does not exist another distinct extreme ray* r *of* $\mathcal{C}_\sigma(A')$ *such that* $\text{supp}_I(r) \subseteq \text{supp}_I(r^1+r^2)$. *Also, for a relaxation* $\mathcal{C}^\tau_\sigma(A') = \{x \in \mathbb{R}^9 : Ax = \mathbf{0}, x_{\bar{\tau}} \geq \mathbf{0}\}$ *for some* $\tau \subseteq \sigma$, *two extreme rays* r^1 *and* r^2 *of* $\mathcal{C}^\tau_\sigma(A')$ *are adjacent if and only if there does not exist another distinct extreme ray* r *of* $\mathcal{C}^\tau_\sigma(A')$ *such that* $\text{supp}_I(r) \setminus \tau \subseteq \text{supp}_I(r^1 + r^2) \setminus \tau$.

We now proceed with the extreme ray algorithm (Algorithm 11).

Let $\tau = \{4, 5, 6, 7, 8, 9\} \subseteq \sigma$. *The following three rays are all the extreme rays of the cone* $\mathcal{C} = \mathcal{C}^\tau_\sigma(A') = \{x \in \mathbb{R}^9 : Ax = \mathbf{0}, x_{\bar{\tau}} \geq \mathbf{0}\}$ *where* $\bar{\tau} = \{1, 2, 3\}$:

$$R = \left\{ \begin{array}{rrrrrrrrr} (3, & 0, & 0, & -2, & 1, & 4, & 2, & 2, & -1) \\ (0, & 3, & 0, & 1, & 1, & 1, & 2, & -1, & 2) \\ (0, & 0, & 3, & 4, & 1, & -2, & -1, & 2, & 2) \end{array} \right\}.$$

We have scaled the extreme rays so that all data is integral, which we will do for all data in this example. Let r^i *be the* i*th row of* R: *that is,* $r^i = R_i$. *Note that* $\text{supp}_I(r^1) \setminus \tau = \{1\}$, $\text{supp}_I(r^2) \setminus \tau = \{2\}$, *and* $\text{supp}_I(r^3) \setminus \tau = \{3\}$, *so* r^1, r^2, *and* r^2 *have a support size of* 1; *thus, they are all the extreme rays of* $\mathcal{C}$.

(i). Note that $\tau = \{4, 5, 6, 7, 8, 9\}$. *Select* $i = 4$. *So,* $R^+ = \{r^2, r^3\}$ *and* $R^- = \{r^1\}$. *We now process all pairs of extreme rays from* R^+ *and* R^- *and check for adjacency.*

(a) Select $r^1 = (3, 0, 0, -2, 1, 4, 2, 2, -1)$ *and* $r^2 = (0, 3, 0, 1, 1, 1, 2, -1, 2)$. *The two extreme rays* r^1 *and* r^2 *are adjacent since* $\text{supp}_I(r^3) \setminus \tau = \{3\} \not\subseteq \text{supp}_I(r^1 + r^2) \setminus \tau = \{1, 2\}$ *and* r^3 *is the only extreme ray distinct from* r^1 *and* r^2. *We then create the new extreme ray*

$$r = (-A'_i r^1)r^2 + (A'_i r^2)r^1 = (-r^1_i)r^2 + (r^2_i)r^1 = (3, 6, 0, 0, 3, 6, 6, 0, 3).$$

In practice, since we use only integral data, it is computationally worthwhile to remove common divisors from extreme rays to avoid numerical problems. So, we divide r *by* 3 *giving* $(1, 2, 0, 0, 1, 2, 2, 0, 1)$, *which is effectively the same extreme ray.*

(b) Select $r^1 = (3, 0, 0, -2, 1, 4, 2, 2, -1)$ *and* $r^3 = (0, 0, 3, 4, 1, -2, -1, 2, 2)$. *These two extreme rays are also adjacent. We then create the new extreme ray*

$$r = (-r^1_i)r^3 + (r^3_i)r^1 = (12, 0, 6, 0, 6, 12, 6, 12, 0).$$

Again, we remove common divisors from r *giving* $(2, 0, 1, 0, 1, 2, 1, 2, 0)$.

We now add the new extreme rays to R *and remove the extreme rays in* R^- *from* R:

$$R = \left\{ \begin{array}{rrrrrrrrr} (0, & 3, & 0, & 1, & 1, & 1, & 2, & -1, & 2) \\ (0, & 0, & 3, & 4, & 1, & -2, & -1, & 2, & 2) \\ (1, & 2, & 0, & 0, & 1, & 2, & 2, & 0, & 1) \\ (2, & 0, & 1, & 0, & 1, & 2, & 1, & 2, & 0) \end{array} \right\}.$$

Set $\tau = \tau \setminus i$.

(ii). Note that $\tau = \{5, 6, 7, 8, 9\}$. *Select* $i = 8$. *So,* $R^+ = \{r^2, r^4\}$ *and* $R^- = \{r^1\}$.

(a) Select $r^1 = (0, 3, 0, 1, 1, 1, 2, \text{-}1, 2)$ *and* $r^2 = (0, 0, 3, 4, 1, \text{-}2, \text{-}1, 2, 2)$. *These two extreme rays are adjacent, and they create the following new extreme ray:* $(0, 2, 1, 2, 1, 0, 1, 0, 2)$.

(b) Select $r^1 = (0, 3, 0, 1, 1, 1, 2, \text{-}1, 2)$ *and* $r^4 = (2, 0, 1, 0, 1, 2, 1, 2, 0)$. *These two extreme rays are not adjacent because* $\text{supp}_I(r^2) \setminus \tau = \{1, 2\} \subseteq \text{supp}_I(r^1 + r^4) \setminus \tau = \{1, 2, 3, 4\}$.

We now add the new extreme rays to R *and remove the extreme rays in* R^- *from* R:

$$R = \left\{ \begin{array}{rrrrrrrrr} (1, & 2, & 0, & 0, & 1, & 2, & 2, & 0, & 1) \\ (0, & 0, & 3, & 4, & 1, & \text{-}2, & \text{-}1, & 2, & 2) \\ (2, & 0, & 1, & 0, & 1, & 2, & 1, & 2, & 0) \\ (0, & 2, & 1, & 2, & 1, & 0, & 1, & 0, & 2) \end{array} \right\}.$$

Set $\tau = \tau \setminus i$.

(iii). Note that $\tau = \{5, 6, 7, 9\}$. *Select* $i = 6$. *So,* $R^+ = \{r^1, r^3\}$ *and* $R^- = \{r^2\}$.

(a) Select $r^2 = (0, 0, 3, 4, 1, \text{-}2, \text{-}1, 2, 2)$ *and* $r^1 = (2, 0, 1, 0, 1, 2, 1, 2, 0)$. *These two extreme rays are adjacent, and they create the following new extreme ray:* $(1, 0, 2, 2, 1, 0, 0, 2, 1)$.

(b) Select $r^2 = (0, 0, 3, 4, 1, \text{-}2, \text{-}1, 2, 2)$ *and* $r^3 = (1, 2, 0, 0, 1, 2, 2, 0, 1)$. *These two extreme rays are not adjacent because* $\text{supp}_I(r^1) \setminus \tau = \{1, 2\} \subseteq \text{supp}_I(r^2 + r^3) \setminus \tau = \{1, 2, 3, 4, 8\}$.

We now add the new extreme rays to R *and remove the extreme rays in* R^- *from* R:

$$R = \left\{ \begin{array}{rrrrrrrrr} (0, & 2, & 1, & 2, & 1, & 0, & 1, & 0, & 2) \\ (2, & 0, & 1, & 0, & 1, & 2, & 1, & 2, & 0) \\ (1, & 2, & 0, & 0, & 1, & 2, & 2, & 0, & 1) \\ (1, & 0, & 2, & 2, & 1, & 0, & 0, & 2, & 1) \end{array} \right\}.$$

Set $\tau = \tau \setminus i$.

(iv). Note that $\tau = \{5, 7, 9\}$. *Select* $i = 7$. *So,* $R^+ = \{r^1, r^3\}$ *and* $R^- = \emptyset$. *Since* $R^- = \emptyset$, *there are no extreme rays pairs to check for adjacency, so we can immediately proceed to the next iteration. The set* R *remains the same. This reason is that the non-negativity constraint on* x_i *is redundant. Set* $\tau = \tau \setminus i$.

(v). Note that $\tau = \{5, 9\}$. *Select* $i = 9$. *So,* $R^+ = \{r^1, r^3, r^4\}$ *and* $R^- = \emptyset$. *Again, there is nothing to do for this iteration, and* R *remains the same. Set* $\tau = \tau \setminus i$.

(vi). Note that $\tau = \{5\}$. *Select* $i = 5$. *So,* $R^+ = \{r^1, r^2, r^3, r^4\}$ *and* $R^- = \emptyset$. *Yet again, there is nothing to do, and* R *remains the same. Set* $\tau = \tau \setminus i$.

(vii). Note that $\tau = \emptyset$, *and the algorithm terminates.*

Thus, there are four extreme rays of the cone of magic squares:

$$R = \left\{ \begin{array}{ccccccccc} (0, & 2, & 1, & 2, & 1, & 0, & 1, & 0, & 2) \\ (2, & 0, & 1, & 0, & 1, & 2, & 1, & 2, & 0) \\ (1, & 2, & 0, & 0, & 1, & 2, & 2, & 0, & 1) \\ (1, & 0, & 2, & 2, & 1, & 0, & 0, & 2, & 1) \end{array} \right\}.$$

We can write these four vectors as 3×3 *magic squares as follows:*

0	2	1	2	0	1	1	2	0	1	0	2
2	1	0	0	1	2	0	1	2	2	1	0
1	0	2	1	2	0	2	0	1	0	2	1

So, every possible 3×3 *magic square is a conic combination of these squares.*

8.2 Optimisations

How well the double description method performs depends to an enormous extent on the efficiency of the implementation. Often algorithms are presented without mentioning the necessary optimisations to make them efficient, which makes it difficult if not impossible to reproduce the computational results. For this reason, we present the most important optimisations of the algorithm. Almost all of the optimisations presented in this section are our own.

When running Algorithm 11, almost all of the computation time is spent in checking whether two extreme rays are adjacent for both the algebraic and combinatorial approach. This is largely due to the enormous number of adjacency checks we need to perform. For example, consider the cone of 6×6 magic squares. This cone is a 24-dimensional pointed cone with a total of 97,548 extreme rays, and during the double description method, we need to perform the adjacency check 1,277,014,866 many times. Despite this, using the optimisations in this section, we are able to compute all of the 97,548 extreme rays and perform all of the 1,277,014,866 adjacency checks in just 8.0 seconds on a Pentium 4 3.0 GHz machine with 1.0Gb of RAM running Linux. All of the optimisations we present in this section are designed to speed-up the procedure for checking whether two extreme rays are adjacent or not.

8.2.1 Eliminating extreme ray pairs

In this section, we describe quick sufficient checks for when two extreme rays are or are not adjacent. These quick checks are absolutely necessary for any practical implementation of the double description method. The checks are not enough, so we will still need to rely on the algebraic and combinatorial checks if all of the quick checks fail to determine adjacency. The first check of Lemma 8.2.1 below is well-known (see for example [37]), but the rest of the checks is this section are novel.

Let $\mathcal{C}_\sigma(A)$ be a pointed cone, and let r^1 and r^2 be two extreme rays of $\mathcal{C}_\sigma(A)$. We have seen that r^1 and r^2 are adjacent if and only if $n - \text{rank}(A_{\bar{\tau}}) = 2$ where

$\tau = \text{supp}_A(r^1 + r^2)$. The rank of the matrix $A_{\bar{\tau}}$ is clearly at most $|\bar{\tau}|$; this fact suggests the following lemma, which gives the most important optimisation for the double description method. Note that, for a ray $r \in \mathcal{C}_\sigma(A)$, we write $\overline{\text{supp}}_A(r)$ meaning the complement of $\text{supp}_A(r)$ in the set $\{1, ..., m\}$, or more explicitly, we have $\overline{\text{supp}}_A(r) := \{i \in \{1, ..., m\} : A_i r = 0\}$.

Lemma 8.2.1 ([37]). *Let $\mathcal{C}_\sigma(A)$ be a pointed cone and let r^1 and r^2 be two extreme rays of $\mathcal{C}_\sigma(A)$. The rays r^1 and r^2* ***are not*** *adjacent if $|\overline{\text{supp}}_A(r^1 + r^2)| < n - 2$.*

Observe that $|\overline{\text{supp}}_A(r^1 + r^2)| = m - |\text{supp}_A(r^1 + r^2)|$, so Lemma 8.2.1 equivalently says that r^1 and r^2 are not adjacent if $|\text{supp}_A(r^1 + r^2)| > m - n + 2$. The intuition behind this result is that if the support of $r^1 + r^2$ is too large – the vector $r^1 + r^2$ satisfies too few inequalities at equality, then the vector $r^1 + r^2$ cannot lie on a two dimensional face of $\mathcal{C}_\sigma(A)$ and it must lie somewhere more towards the interior of $\mathcal{C}_\sigma(A)$.

This result appears rather innocuous, but in practice, it is surprisingly useful. For example, for the cone of 6×6 magic squares, recall that we need to perform the adjacency check 1,277,014,866 many times. In this case, the above quick check determines that two extreme rays are not adjacent 1,276,593,070 many times. So, 99.97% of the time, the quick check was useful! That leaves 421,796 extreme ray adjacency checks. Of these, 271,464 extreme ray pairs were not adjacent, and 150,332 extreme ray pairs were adjacent thus creating new extreme rays. This number of adjacent extreme rays is larger than 97,548 because many of the 150,332 new extreme rays were created in an intermediate relaxation and were not feasible for the original cone.

The above lemma gives a sufficient condition for when r^1 and r^2 are not adjacent. We now present a sufficient condition for when r^1 and r^2 are adjacent.

Lemma 8.2.2. *Let $\mathcal{C}_\sigma(A)$ be a pointed cone and let r^1 and r^2 be two distinct extreme rays of $\mathcal{C}_\sigma(A)$. The rays r^1 and r^2* ***are*** *adjacent if $|\text{supp}_A(r^2) \setminus \text{supp}_A(r^1)| = 1$ or $|\text{supp}_A(r^1) \setminus \text{supp}_A(r^2)| = 1$.*

Proof. Assume that $|\text{supp}_A(r^2) \setminus \text{supp}_A(r^1)| = 1$. Since r^1 is an extreme ray of $\mathcal{C}_\sigma(A)$, we must have $n - \text{rank}(A_{\bar{\tau}}) = 1$ where $\tau = \text{supp}_A(r^1)$. This implies that $n - \text{rank}(A_{\bar{\rho}}) \leq 2$ where $\rho = \text{supp}_A(r^1 + r^2)$ since $|\bar{\rho}| = |\bar{\tau}| - 1$ by assumption. However, we must have $n - \text{rank}(A_{\bar{\tau}}) \geq 2$ since r^1 and r^2 are distinct extreme rays. Therefore, $n - \text{rank}(A_{\bar{\rho}}) = 2$, and r^1 and r^2 are adjacent extreme rays.

The case where $|\text{supp}_A(r^1) \setminus \text{supp}_A(r^2)| = 1$ is the same as before after switching the roles of r^1 and r^2. □

Using Lemma 8.2.2 above after applying Lemma 8.2.1, we can further reduce the number of times that we need to perform more expensive algebraic or combinatorial checks for adjacency. For the cone of 6×6 magic squares, where 150,332 extreme ray pairs created new extreme rays, we find that 81,398 of the 150,332 pairs satisfied Lemma 8.2.2 leaving 68,934 that didn't. Hence, we reduce the number of expensive adjacency checks from 421,796 to 340,398.

There is a special case where Lemmas 8.2.1 and 8.2.2 both apply giving a necessary and sufficient condition for when two extreme rays are adjacent.

Lemma 8.2.3. *Let $\mathcal{C}_\sigma(A)$ be a pointed cone and let r^1 and r^2 be two extreme rays of $\mathcal{C}_\sigma(A)$ where $|\overline{\text{supp}}_A(r^1)| = n-1$. The rays r^1 and r^2 are adjacent if and only if $|\operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)| = 1$.*

Proof. If $|\operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)| > 1$, then $|\overline{\text{supp}}_A(r^1 + r^2)| < n-2$ since by assumption $|\overline{\text{supp}}(r^1)| = n-1$; therefore, r^1 and r^2 are not adjacent by Lemma 8.2.1. On the other hand, if $|\operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)| = 1$, then r^1 and r^2 are adjacent by Lemma 8.2.2. □

Using this lemma, we can decrease the number of times we need to perform the check from Lemma 8.2.1.

During an iteration of the double description method, we must check whether every $r^1 \in R^+$ is adjacent to every $r^2 \in R^-$. So, at the start of an iteration, we check whether $|\overline{\text{supp}}_A(r^1)| = n-1$ for all $r^1 \in R^+$ and whether $|\overline{\text{supp}}_A(r^2)| = n-1$ for all $r^2 \in R^-$. Then, for every pair $r^1 \in R^+$ and $r^2 \in R^-$ where $|\overline{\text{supp}}_A(r^1)| = n-1$, r^1 and r^2 are adjacent if and only if $|\operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)| = 1$, and similarly, for every pair $r^1 \in R^+$ and $r^2 \in R^-$ where $|\overline{\text{supp}}_A(r^2)| = n-1$, r^1 and r^2 are adjacent if and only if $|\operatorname{supp}_A(r^1) \setminus \operatorname{supp}_A(r^2)| = 1$. So, if $|\overline{\text{supp}}_A(r^1)| = n-1$ or $|\overline{\text{supp}}_A(r^2)| = n-1$, then we do not have to compute compute $|\overline{\text{supp}}_A(r^1 + r^2)|$ for every r^2, which we would normally do as part of the quick check from 8.2.1. For our cone of magic squares example, by doing this check, we reduce the number of times we need to check Lemma 8.2.1 from 1,277,014,866 to 168,216,044 while we need to perform the check of Lemma 8.2.3 only 154,041 times (i.e. checking whether $|\overline{\text{supp}}_A(r^1)| = n-1$ for all $r^1 \in R^+$ and whether $|\overline{\text{supp}}_A(r^2)| = n-1$ for all $r^2 \in R^-$). We must still check Lemma 8.2.2 the same number of times.

There is one final optimisation that we present in this section, which is also a very important one. It is simple but effective. Recall from Lemma 8.1.4 that two distinct extreme rays r^1 and r^2 of a cone $\mathcal{C}_\sigma(A)$ are adjacent if and only if there does not exist another distinct extreme ray r such that $\operatorname{supp}_A(r) \subseteq \operatorname{supp}_A(r^1 + r^2)$. We can express the condition $\operatorname{supp}_A(r) \subseteq \operatorname{supp}_A(r^1 + r^2)$ equivalently as $\operatorname{supp}_A(r) \setminus \operatorname{supp}_A(r^1) \subseteq \operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)$, and this is true only if we have $|\operatorname{supp}_A(r) \setminus \operatorname{supp}_A(r^1)| \leq |\operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)|$. Now, in the double description method, as we mentioned before, we check whether one extreme ray r^1 of $\mathcal{C}_\sigma(A)$ is adjacent to many other extreme rays of $\mathcal{C}_\sigma(A)$. So, given an extreme ray r of $\mathcal{C}_\sigma(A)$ where $|\operatorname{supp}_A(r) \setminus \operatorname{supp}_A(r^1)| = 1$, we would expect that the condition $\operatorname{supp}_A(r) \setminus \operatorname{supp}_A(r^1) \subseteq \operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)$ is often true as r^2 varies over the extreme rays of $\mathcal{C}_\sigma(A)$.

So, during the double description method, when checking for adjacent extreme rays of some $r^1 \in R^+$, we first find all extreme rays r such that $|\operatorname{supp}_A(r) \setminus \operatorname{supp}_A(r^1)| = 1$, and then we use these extreme rays as a first check when determining if $r^2 \in R^-$ is not adjacent to r^1. Let R be the set of extreme rays of $\mathcal{C}_\sigma(A)$, and let

$$I := \{i \in \{1, ..., m\} : \exists r \in R, |\operatorname{supp}_A(r) \setminus \operatorname{supp}_A(r^1)| = \{i\}\}.$$

Then, r^2 is not adjacent to r^1 if $\operatorname{supp}_A(r^2) \cap I \neq \emptyset$ and $|\operatorname{supp}_A(r^2) \setminus \operatorname{supp}_A(r^1)| \neq 1$. We show in the following sections that it is possible to compute the set I very quickly using either the algebraic approach or the combinatorial approach thus making it

computationally worthwhile to perform this check. This check is surprisingly useful in practice. For our cone of magic squares example, if we apply this check, without any other checks, then out of the 1,277,014,866 many times that we need to check the adjacency of two extreme rays, this checks proves that 1,255,820,367 many extreme ray pairs are not adjacent. So, 98.3% of the time, this check was useful. Moreover, if we apply all of the checks of this section, then we find that we only need to perform an expensive adjacency check only 156,410 many times, and out of this, a pair of extreme rays was found to be adjacent 68,934 many times. Although, this check is not as effective as the check of Lemma 8.2.2, we found that we could implement it faster in practice, so we perform this check first before any of the other checks.

In Algorithm 12, we have added all the above checks to Algorithm 11 in a way that closely resembles the actual implementation in software.

8.2.2 Combinatorial approach

In this section, we show how to optimise the "combinatorial" approach for checking whether two extreme rays of a pointed cone are adjacent. Recall that two extreme rays r^1 and r^2 of a pointed cone $\mathcal{C}_\sigma(A)$ are adjacent if and only if there does not exist another extreme ray r^3 of C (not r^1 nor r^2) such that $\text{supp}_A(r^3) \subseteq \text{supp}_A(r^1 + r^2)$.

The most straight-forward approach to determine whether two distinct extreme rays r^1 and r^2 are adjacent is to store a list of all supports of all extreme rays of $\mathcal{C}_\sigma(A)$ and to iterate through the list from start to finish searching for a different extreme ray r^3 such that $\text{supp}_A(r^3) \subseteq \text{supp}_A(r^1 + r^2)$. More formally, we proceed as follows. Let $R := \{r^1, r^2, ..., r^k\}$ be the set of extreme rays of $\mathcal{C}_\sigma(A)$, and let $S := \{s^1, s^2, ..., s^k\}$ be the supports of the extreme rays: $s^i = \text{supp}_A(r^i)$ for $i = 1, ..., k$. So, for a given pair of distinct extreme rays r^{j_1} and r^{j_2} for some $j_1, j_2 \in \{1, ..., k\}$, we check whether $s^i \subseteq s^{j_1} \cup s^{j_2}$ for each $i = 1, ..., k$ where $i \neq j_1, j_2$. We can improve upon this simple method a little by noting that $s^i \subseteq s^{j_1} \cup s^{j_2}$ only if $|s^i| \leq |s^{j_1} \cup s^{j_2}|$. So, we sort the list of extreme rays R in order of increasing support size. In this way, we only have to iterate through S until either $s^i \subseteq s^{j_1} \cup s^{j_2}$ and $i \neq j_1, j_2$, or $|s^i| > |s^{j_1} \cup s^{j_2}|$.

This straight-forward approach is efficient when the number of extreme rays is a less than a thousand, but otherwise we need a more sophisticated approach. Instead, we use a tree structure as opposed to a list to store the support of all the extreme rays. This tree structure has proven in practice to be a very efficient mechanism for determining adjacency of extreme rays. The tree is constructed in such a way that each leaf of the tree will correspond to the support of one extreme ray.

Let $T = (V, E)$ be a tree where V are the vertices or nodes and E are the edges. Each node $v \in V$ of the tree T except the root node has a label $l_v \in \{1, ..., m\}$, and by convention, we give the root node the label 0. The label of each node is always strictly greater than the label of its parent node, and we require that the label of a node is unique amongst its siblings, although the label might not be unique amongst all nodes. For any leaf node $v \in V$, the list of labels of its ancestors including its own label but excluding the root node's label is a subset of $\{1, ..., m\}$. Moreover, this subset is unique for every leaf of the tree since labels are increasing as we move down the tree and labels are unique amongst the siblings of a node. So, given a list

Algorithm 12 Optimised Ray Algorithm

Input: a pointed cone $\mathcal{C}_\sigma(A)$
Output: the set R of extreme rays of $\mathcal{C}_\sigma(A)$.

Find a set $\tau \subseteq \sigma$ such that $\mathcal{C}^\tau_\sigma(A)$ is pointed.
Compute the minimal set R of extreme rays $\mathcal{C}^\tau_\sigma(A)$.
while $\tau \neq \emptyset$ **do**
  Select $i \in \tau$.
  $R^+ := \{r \in R : A_i r > 0\}$.
  $R^- := \{r \in R : A_i r < 0\}$.
  $R := R \setminus R^-$.
  $R1^+ := \{r \in R^+ : |\overline{\text{supp}}_A(r) \setminus \tau| = n - 1\}$.
  $R1^- := \{r \in R^- : |\overline{\text{supp}}_A(r) \setminus \tau| = n - 1\}$.
  for all $r^1 \in R1^+$ and $r^2 \in R^-$ **do**
    if $|(\text{supp}_A(r^2) \setminus \tau) \setminus (\text{supp}_A(r^1) \setminus \tau)| = 1$ **then** $R := R \cup \{(-A_i r^2) r^1 + (A_i r^1) r^2\}$
    end if
  end for
  for all $r^1 \in R^+ \setminus R1^+$ and $r^2 \in R1^-$ **do**
    if $|(\text{supp}_A(r^1) \setminus \tau) \setminus (\text{supp}_A(r^2) \setminus \tau)| = 1$ **then** $R := R \cup \{(-A_i r^2) r^1 + (A_i r^1) r^2\}$
    end if
  end for
  for all $r^1 \in R^+ \setminus R1^+$ and $r^2 \in R^- \setminus R1^-$ **do**
    $I := \{i \in \{1, ..., m\} : (\text{supp}_A(r) \setminus \tau) \setminus (\text{supp}_A(r^1) \setminus \tau) = \{i\}, r \in R\}$
    if $\text{supp}_A(r^2) \setminus \tau \cap I = \emptyset$ **then**
      if $|\overline{\text{supp}}_{A_{\bar{\tau}}}(r^1 + r^2)| < n - 2$ **then**
        if $|(\text{supp}_A(r^1) \setminus \tau) \setminus (\text{supp}_A(r^2) \setminus \tau)| = 1$ **then**
          $R := R \cup \{(-A_i r^2) r^1 + (A_i r^1) r^2\}$.
        else if r^1 and r^2 are adjacent in $\mathcal{C}^\tau_\sigma(A)$ **then**
          $R := R \cup \{(-A_i r^2) r^1 + (A_i r^1) r^2\}$.
        end if
      end if
    else if $|(\text{supp}_A(r^2) \setminus \tau) \setminus (\text{supp}_A(r^1) \setminus \tau)| = 1$ **then**
      $R := R \cup \{(-A_i r^2) r^1 + (A_i r^1) r^2\}$.
    end if
  end for
  $\tau := \tau \setminus i$.
end while
return R

of supports of extreme rays, we can construct a tree T such that the support of each extreme ray is associated with exactly one leaf of the tree.

Now, as above, let $R := \{r^1, r^2, ..., r^k\}$ be the set of extreme rays of $\mathcal{C}_\sigma(A)$, and let $S := \{s^1, s^2, ..., s^k\}$ be the supports of the extreme rays. Also, let $T = (V, E)$ be the tree of the supports in S as described above. So, if we wish to check that a given pair of distinct extreme rays r^{j_1} and r^{j_2} for some $j_1, j_2 \in \{1, ..., k\}$ are adjacent, we must check whether there exists $i \in \{1, ..., k\}$ such that $s^i \subseteq s^{j_1} \cup s^{j_2}$ and $i \neq j_1, j_2$. We then proceed as follows. Starting from the root node, we traverse the tree T is a depth first search fashion where we only visit a vertex $v \in V$ if $l_v \in s^{j_1} \cup s^{j_2}$. If we reach a leaf $v \in V$ of the tree associated with the support set $s^i \in \{1, ..., m\}$, then we must have $s^i \subseteq s^{j_1} \cup s^{j_2}$ since $l_{v'} \in s^{j_1} \cup s^{j_2}$ for every ancestor v' of v excluding the root node. So, if $i \neq j_1, j_2$, then we have shown that r^{j_1} and r^{j_2} are not adjacent and we can stop searching, otherwise we continue to traverse the tree. If we have traversed the entire tree without finding a support set s^i where $s^i \subseteq s^{j_1} \cup s^{j_2}$ and $i \neq j_1, j_2$, then r^{j_1} and r^{j_2} are adjacent.

Example 8.2.4. *In Table 8.1, we list the supports of 20 extreme rays of the cone of* 4×4 *magic squares. Figure 8.2 is the tree associated with these 20 extreme rays. In this diagram, we have numbered the leaves of the tree, and the number of a leaf corresponds to the number of the support it represents in Table 8.1. Consider for instance, the leaf labelled* 7 *in Figure 8.2. Its label is 14, and its ancestors are labelled 12, 7, 1, and 0 moving up the tree. Therefore, the set associated with this leaf is* $\{1, 7, 12, 14\}$*, which is exactly the same as support number 7 in Table 8.1.*

#	support	#	support
1	{1,2,5,7,10,12,15,16}	11	{2,5,7,11,12,13,16}
2	{1,2,6,7,12,13,15}	12	{2,7,9,16}
3	{1,3,7,8,9,10,14,16}	13	{2,8,11,13}
4	{1,3,8,10,11,13,14}	14	{3,4,6,7,9,11,13,14}
5	{1,4,5,6,10,13,15}	15	{3,4,6,8,9,11,13,14}
6	{1,7,8,9,11,14}	16	{3,5,10,16}
7	{1,7,12,14}	17	{3,6,8,9,10,13,16}
8	{1,8,10,15}	18	{3,6,12,13}
9	{2,4,5,6,11,12,13,15}	19	{4,5,11,14}
10	{2,4,5,10,11,15,16}	20	{4,6,9,15}

Table 8.1: Extreme ray supports.

Consider $j_1 = 12$ *and* $j_2 = 19$*; hence,* $s^{j_1} = \{2, 7, 9, 16\}$ *and* $s^{j_2} = \{4, 5, 11, 14\}$*, and* $s^{j_1} \cup s^{j_2} = \{2, 4, 5, 7, 9, 11, 14, 16\}$*. In this case, the extreme rays* r^{j_1} *and* r^{j_2} *are adjacent. In Figure 8.3, we show the part of tree of supports that we would traverse using a depth first search approach. Note that only the leaves 12 and 19 are visited thus proving that* r^{j_1} *and* r^{j_2} *are adjacent.*

In Section 8.2.1, the last quick check for deducing that two extreme rays are not adjacent requires that, given an extreme ray r of the cone $\mathcal{C}_\sigma(A)$, we find the set $I \subseteq \{1, ..., m\}$ where $i \in I$ if $\text{supp}_A(r') \setminus \text{supp}_A(r) = \{i\}$ for some extreme ray r'

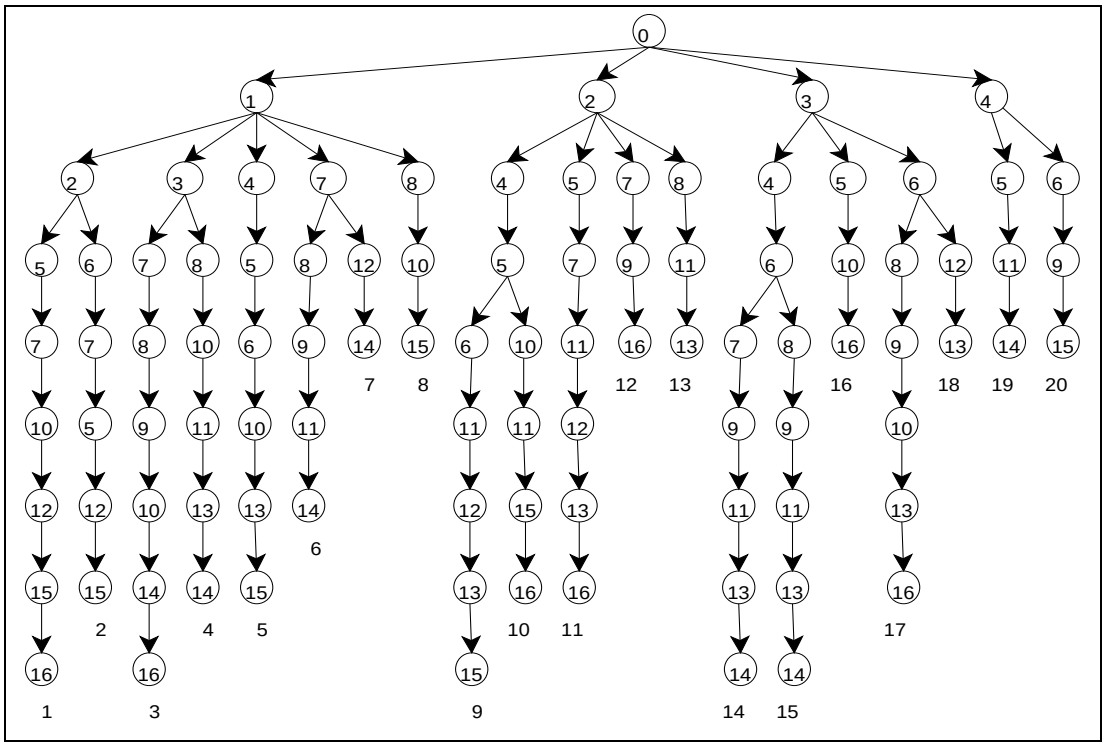

Figure 8.2: A support tree.

of $\mathcal{C}_\sigma(A)$. We can find this set efficiently by traversing the support tree $T = (V, E)$ as follows. We traverse the tree in a depth first search fashion and we visit a node $v \in V$ if $l_v \in \text{supp}_A(r)$ with the exception that we may visit v where $l_v \notin \text{supp}_A(r)$ if v does not have an ancestor v' such that $l_{v'} \notin \text{supp}_A(r)$. In this way, if we visit a leaf associated with extreme ray r', we know $|\text{supp}_A(r') \setminus \text{supp}_A(r)| = 1$ or $r' = r$. So, we can compute the set I from the set of leaves that we visited.

In the example tree of Figure 8.2, there are many vertices with only one child and often there is a sequence of vertices with only one child ending at a leaf node. Traversing such structures is not very efficient. One improvement of the current tree structure would be to avoid such sequence of vertices. If a vertex has only one child, then we combine it with its child node. In this way, a sequence of vertices each with just one child node becomes one vertex, and the label of this new vertex would be the set of indices consisting of the labels of each of the vertices that this new vertex replaces. We have not yet implemented such a tree; it would be interesting to see if there is a significant improvement in performance.

Independently, Terzer and Stelling wrote an article [84] about a different tree structure for speeding up the combinatorial approach. The tree they suggest is a binary tree where every node is labelled either 0 or 1 (and the root node is not labelled). So, for a given leaf node v, the list of labels of its ancestors from the root node downwards is sequence of 0's and 1's of length k (i.e. a subset of $\{1, 0\}^k$) where k is depth of the leaf node. A sequence $q \in \{1, 0\}^k$ represents a set $s \subseteq \{1, ..., k\}$ where $i \in s$ if $q_i = 1$. Similarly to our tree structure, they associate a support set with every leaf of the tree, so, given a support set s, they can search the binary tree for a

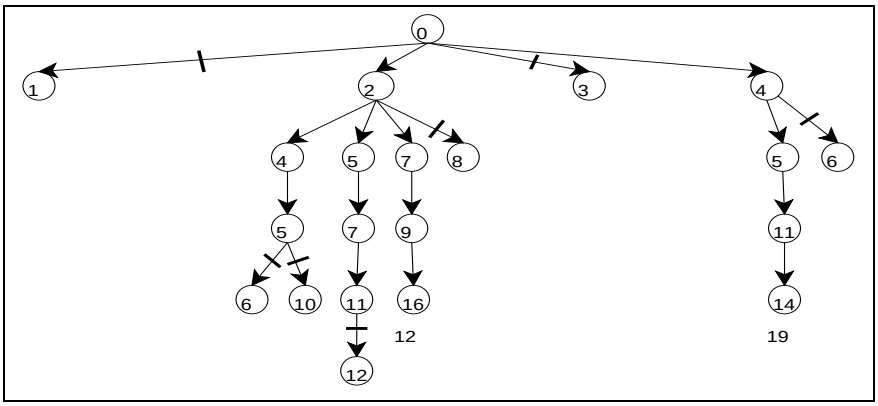

Figure 8.3: Traversing the support tree.

support set s' such that $s' \subseteq s$. Independently, we also implemented such a binary tree structure, but we found that the tree as described above lead to faster computations, which we believe is because the binary tree has a larger depth. However, they do suggest to limit the depth of the tree so a leaf corresponds only to a subset of a support, so there may be more than one support set associated with a leaf, and they also suggest another optimisation for the tree for which we refer the reader to [84].

More research is needed here to determine a good tree structure.

8.2.3 Algebraic approach

In this section, we describe important new optimisations to speed-up the algebraic adjacency check, which involves computing the rank of a matrix. Recall that for a cone $\mathcal{C}_\sigma(A)$, two distinct extreme rays r^1 and r^2 of $\mathcal{C}_\sigma(A)$ are adjacent if and only if $\text{rank}(A_{\bar{\tau}}) = n-2$ where $\tau = \text{supp}_A(r^1+r^2)$. So, to determine the adjacency of r^1 and r^2, we must compute the rank of the matrix $A_{\bar{\tau}}$, which is an $|\overline{\text{supp}}_A(r^1 + r^2)|$ by n matrix. Fukuda and Prodon in [37] claim that this approach is not as efficient as the combinatorial approach, but we found the opposite, and as did Gagneur and Klamt in [39]. The algebraic approach has an advantage over the combinatorial approach in that the complexity of performing the algebraic adjacency check does not depend on how many extreme rays there are, whereas the combinatorial adjacency check does.

Clearly, we need an efficient method for computing the rank of a matrix, and note that it may be possible to stop the rank computation early when it becomes clear that $\text{rank}(A_{\bar{\tau}}) < n-2$. We will not discuss the details of computing ranks of matrices any further here, so we assume that we have an efficient method. Instead, we present methods for reducing the size of the matrix by using different formulations of the cone $\mathcal{C}_\sigma(A)$ since reducing the size of the matrix usually results in a quicker rank computation in practice. We will consider three different possible formulations of a cone: $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$, $\mathcal{C} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x \geq \mathbf{0}\}$, and lastly $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}, x \geq \mathbf{0}\}$. We show that any cone $\mathcal{C}_\sigma(A)$ can be transformed

into an equivalent cone in one of the above three forms. We examine the effect of the transformation on the size of the matrix needed to perform the rank check.

We show that it might be beneficial to reformulate the cone $\mathcal{C}_\sigma(A)$ where $A \in \mathbb{R}^{m\times n}$ into the form $\mathcal{C}(A') = \{x \in \mathbb{R}^n : A'x \geq \mathbf{0}\}$ for some matrix $A' \in \mathbb{R}^{m\times k}$ where $k \leq n$. We show that performing this transformation means that we need to compute the rank of a smaller matrix. First, we show how to perform the transformation. We can assume that the matrix $A_{\bar\sigma}$ is full row rank (i.e. $\text{rank}(A_{\bar\sigma}) = |\bar\sigma|$) since we can remove any linearly dependent rows of $A_{\bar\sigma}$ and still have the same cone. Let $\tau \subseteq \{1, ..., m\}$ be $|\bar\sigma|$ many linearly independent columns of $A_{\bar\sigma}$. Let $T \in \mathbb{R}^{|\bar\sigma|\times|\bar\sigma|}$ be the inverse matrix of $A_{\bar\sigma\tau}$ (i.e. $TA_{\bar\sigma\tau} = I$). Note that $A_{\bar\sigma\tau}$ is the submatrix of A consisting of the rows of A index by $\bar\sigma$ and the columns of A indexed by τ and $A_{\bar\sigma\tau} \in \mathbb{R}^{|\bar\sigma|\times|\bar\sigma|}$. Then,

$$\begin{aligned}
\mathcal{C}_\sigma(A) :=& \{x \in \mathbb{R}^n : A_{\bar\sigma}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}\\
=& \{x \in \mathbb{R}^n : TA_{\bar\sigma}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}\\
=& \{x \in \mathbb{R}^n : TA_{\bar\sigma\tau}x_\tau + TA_{\bar\sigma\bar\tau}x_{\bar\tau} = \mathbf{0}, A_{\sigma\tau}x_\tau + A_{\sigma\bar\tau}x_{\bar\tau} \geq \mathbf{0}\}\\
=& \{x \in \mathbb{R}^n : x_\tau = -TA_{\bar\sigma\bar\tau}x_{\bar\tau}, (-A_{\sigma\tau}TA_{\bar\sigma\bar\tau} + A_{\sigma\bar\tau})x_{\bar\tau} \geq \mathbf{0}\}.
\end{aligned}$$

Finally, it follows that if we wish to compute the generators of $\mathcal{C}_\sigma(A)$, then it suffices to compute the generators of the cone $\mathcal{C}(A') \subseteq \mathbb{R}^{|\bar\tau|}$ where $A' = -A_{\sigma\tau}TA_{\bar\sigma\bar\tau} + A_{\sigma\bar\tau}$ since there are no inequality constraints involving the variables x_τ and thus we can effectively ignore them. We can reconstruct the generators of $\mathcal{C}_\sigma(A)$ from the generators of $\mathcal{C}(A')$ by using the equation $x_\tau = -TA_{\bar\sigma\bar\tau}x_{\bar\tau}$. Note that the matrix A' is an $(m - |\bar\sigma|)$ by $(n - |\bar\sigma|)$ matrix. Intuitively, we have eliminated the x_τ variables from $\mathcal{C}_\sigma(A)$ to arrive at $\mathcal{C}(A')$, or in other words, we have projected $\mathcal{C}_\sigma(A)$ onto the $x_{\bar\tau}$ variables.

We now discuss what effect this transformation has on the rank check for adjacency. The set $\text{supp}_A(r)$ is the same as $\text{supp}_{A'}(r')$ since the cones $\mathcal{C}(A')$ and $\mathcal{C}_\sigma(A)$ have essentially the same set of inequalities, but $|\overline{\text{supp}}_{A'}(r')| = |\overline{\text{supp}}_A(r)| - |\bar\sigma|$. Now, let $\tilde r^1, \tilde r^2 \in \mathcal{C}(A')$ where $r^1_{\bar\tau} = \tilde r^1$ and $r^2_{\bar\tau} = \tilde r^2$. Then, determining whether r^1 is adjacent to r^2 in $\mathcal{C}_\sigma(A)$ is equivalent to determining whether $\tilde r^1$ is adjacent to $\tilde r^2$ in $\mathcal{C}(A')$ where we only need to determine the rank of a matrix of size $|\overline{\text{supp}}_{A'}(\tilde r^1 + \tilde r^2)|$ by $(n - |\bar\sigma|)$ where $|\overline{\text{supp}}_{A'}(\tilde r^1 + \tilde r^2)| = |\overline{\text{supp}}_A(r^1 + r^2)| - |\bar\sigma|$. Hence, we have decreased the numbers of columns by $|\bar\sigma|$ and the number of rows by $|\bar\sigma|$ of the matrix needed to perform the rank check. So, it is possibly worth performing the transformation from $\mathcal{C}_\sigma(A)$ to $\mathcal{C}(A')$.

However, in general, $\mathcal{C}(A')$ is not always the best formulation for performing the rank check. Consider the cone $\mathcal{C}_\sigma(A)$ for some $A \in \mathbb{R}^{m\times n}$. Let r^1 and r^2 be two distinct extreme rays of $\mathcal{C}_\sigma(A)$. We have observed in practice that, for some cones, the size of the set $\text{supp}_A(r^1 + r^2)$ can be much smaller than the size of $\overline{\text{supp}}_A(r^1 + r^2)$. If the pair of extreme rays passes the test from Lemma 8.2.1, then $\overline{\text{supp}}_A(r^1 + r^2) \geq n - 2$ or equivalently $\text{supp}_A(r^1 + r^2) \leq m - n + 2$. So, if n is large and m is not much larger than n, then $\text{supp}_A(r^1 + r^2)$ must be small compared to $\overline{\text{supp}}_A(r^1 + r^2)$. It is also true that $\text{supp}_A(r^1 + r^2)$ is often much smaller than $\overline{\text{supp}}_A(r^1 + r^2)$ when the cone is quite degenerate. For these cases, we would prefer to compute the rank of a matrix whose size depends more on $\text{supp}_A(r^1 + r^2)$ rather than on $\overline{\text{supp}}_A(r^1 + r^2)$ as

it does for cones in the form $\mathcal{C}_\sigma(A)$ and $\mathcal{C}(A)$. Next, we give a formulation for which this is the case.

Consider a cone in the form $\mathcal{C} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x \geq \mathbf{0}\}$ for some matrix $A \in \mathbb{R}^{m\times n}$. In our implementation, we only use cones in this form. Note that $\mathcal{C} = \mathcal{C}_\sigma(A') := \{x \in \mathbb{R}^n : A'_{\bar{\sigma}}x = \mathbf{0}, A'_\sigma x \geq \mathbf{0}\}$ where

$$A' = \begin{bmatrix} I \\ A \end{bmatrix}$$

and $\sigma = \{1, ..., m\}$ (i.e. $A'_{\bar{\sigma}} = A$ and $A'_\sigma = I$). The matrix $A' \in \mathbb{R}^{(m+n)\times n}$ has special structure that we can take advantage of when performing rank computations with A'. From Corollary 8.1.3, two distinct extreme rays r^1 and r^2 of $\mathcal{C}(A')$ are adjacent if $\operatorname{rank}(A'_{\bar{\rho}}) = n - 2$ where $\rho = \operatorname{supp}_{A'}(r^1 + r^2)$. Furthermore,

$$A'_{\bar{\rho}} = \begin{bmatrix} I_{\bar{\tau}} \\ A \end{bmatrix}$$

where $\tau = \operatorname{supp}_I(r^1 + r^2)$ because $\rho = \operatorname{supp}_{A'}(r^1 + r^2) = \operatorname{supp}_I(r^1 + r^2) = \tau$. It then follows from linear algebra that $\operatorname{rank}(A'_{\bar{\rho}}) = \operatorname{rank}(A_{*\tau}) + |\bar{\tau}|$, and therefore, $\operatorname{rank}(A'_{\bar{\rho}}) = n - 2$ when $\operatorname{rank}(A_{*\tau}) = n - 2 - |\bar{\tau}| = |\tau| - 2$. Thus, we must compute the rank of a m by $|\tau|$ matrix. Note that the size of the matrix only depends on $|\operatorname{supp}_{A'}(r^1 + r^2)|$ and not on $|\overline{\operatorname{supp}}_{A'}(r^1 + r^2)|$. Recall that if the pair r^1 and r^2 of extreme rays passes the test for non-adjacency from Lemma 8.2.1, then we have $|\overline{\operatorname{supp}}_{A'}(r^1 + r^2)| \geq n - 2$ implying that $|\bar{\tau}| = |\overline{\operatorname{supp}}_I(r^1 + r^2)| \geq n - m - 2$ and $|\tau| = |\operatorname{supp}_I(r^1 + r^2)| \leq m + 2$ since $|\overline{\operatorname{supp}}_{A'}(r^1 + r^2)| = |\overline{\operatorname{supp}}_I(r^1 + r^2)| + m$ and $|\operatorname{supp}_I(r^1 + r^2)| = n - |\overline{\operatorname{supp}}_I(r^1 + r^2)|$. So, if m is small and n is large or $\mathcal{C}$ is quite degenerate, then the matrix $A_{*\tau}$ is small in comparison with the matrix A.

Now, we show how to transform any pointed cone into the very convenient form $\mathcal{C} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x \geq \mathbf{0}\}$ and what effect this has on the rank computation. Consider the pointed cone $\mathcal{C}(A) := \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$. We assume that any equality constraints have been removed using the transformation mentioned above, which only reduces the number of variables and constraints. We will show that we can transform $\mathcal{C}(A)$ into a cone $\mathcal{C} = \{x \in \mathbb{R}^n : A'x = \mathbf{0}, x \geq \mathbf{0}\}$ where A' is an $(m-n)$ by m matrix. We perform the transformation by introducing slack variables to change inequality constraints into equality constraints, and then, we eliminate the x variables. Now, since $\mathcal{C}(A)$ is pointed, $\operatorname{rank}(A) = n$, and then, by linear algebra, there exists a matrix $T \in \mathbb{R}^{m\times m}$ such that $(TA)^\intercal = (I, \mathbf{0})^\intercal$; that is, the first n rows of TA are the n by n identity matrix and the last $m-n$ rows of TA are zero. Then,

$$\begin{aligned}
\mathcal{C}(A) :=& \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\} \\
=& \{x \in \mathbb{R}^n : Ax - Is = \mathbf{0}, s \geq \mathbf{0}\} \\
=& \{x \in \mathbb{R}^n : TAx - Ts = \mathbf{0}, s \geq \mathbf{0}\} \\
=& \{x \in \mathbb{R}^n : x = T_\rho s, T_{\bar{\rho}} s = \mathbf{0}, s \geq \mathbf{0}\}
\end{aligned}$$

where $\rho = \{1, ..., n\}$, so T_ρ is the first n rows of T and $T_{\bar{\rho}}$ is the last $m - n$ rows of T. It follows that if we wish to compute the extreme rays of $\mathcal{C}(A)$, then it suffices to compute the extreme rays of $\mathcal{C} = \{s \in \mathbb{R}^m : A's = \mathbf{0}, s \geq \mathbf{0}\}$ where $A' = T_{\bar{\rho}}$,

which is an $(m-n)$ by m matrix. We can reconstruct the generators of $\mathcal{C}(A)$ from the generators of $\mathcal{C}$ using the equation $x = T_\rho s$. Moreover, note that if $x \in \mathcal{C}(A)$ and $s \in \mathcal{C}$ where $s = Ax$ (or equivalently $x = T_\rho s$), then $A_i x > 0$ if and only if $s_i > 0$, which means that we have not changed the support of extreme rays during the transformation: $\operatorname{supp}_A(x) = \operatorname{supp}_I(s)$. Let $r^1, r^2 \in \mathcal{C}(A)$ and let $s^1, s^2 \in \mathcal{C}$ such that $Ar^1 = s^1$ and $Ar^2 = s^2$. Performing the adjacency rank check in $\mathcal{C}(A)$ means computing $\operatorname{rank}(A_{\bar{\tau}})$ where $\tau = \operatorname{supp}_A(r^1 + r^2)$. The matrix $A_{\bar{\tau}}$ is a $|\bar{\tau}|$ by n matrix. On the other hand, performing the adjacency rank check in $\mathcal{C}$ means computing $\operatorname{rank}(A'_{*\tau})$ as we saw above. Here, the matrix $A'_{*\tau}$ is a $(m-n)$ by $|\tau|$ matrix. Hence, if n is large and m is not much larger than n or $\mathcal{C}(A)$ is quite degenerate, then $|\tau|$ is small compared to $|\bar{\tau}|$ and the matrix $A'_{*\tau}$ is probably much smaller than the matrix $A_{\bar{\tau}}$, in which case, we prefer the $\mathcal{C}$ formulation. However, if n is small and m is large, then $|\bar{\tau}|$ is small compared to $|\tau|$ and $A'_{*\tau}$ is probably much larger than the matrix $A_{\bar{\tau}}$, in which case, we prefer the $\mathcal{C}(A)$ formulation.

There is another benefit of the formulation $\mathcal{C} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x \geq \mathbf{0}\}$: it is easy to find all extreme rays $r \in \mathcal{C}$ such that $|\operatorname{supp}_A(r)| = 1$. Note that if $|\operatorname{supp}_A(r)| = 1$ for $r \in \mathcal{C}$, then r must be an extreme ray since it is definitely support-minimal. Firstly, if the ith column of A is zero (i.e. $A_i = \mathbf{0}$), then $r = \mathbf{e}^i$ is an extreme ray of $\mathcal{C}$ since $r \in \mathcal{C}$ and $|\operatorname{supp}_A(r)| = 1$. Secondly, if r is an extreme of $\mathcal{C}$ such that $|\operatorname{supp}_A(r)| = 1$, then $r = \mathbf{e}^i$ for some $i \in \{1, ..., n\}$ and the ith column of A must be zero.

Fortunately, there is a formulation of a cone that has the best of both of the previous formulations. We will argue that the form $\mathcal{C} := \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}, x \geq \mathbf{0}\}$ for some $A \in \mathbb{R}^{m \times n}$ is the best formulation for cones for performing the algebraic check. Importantly, we will show below that any pointed cone can be written in this form without increasing the size of the constraint matrix. Note that we have $\mathcal{C} = \mathcal{C}(A') := \{x \in \mathbb{R}^n : A'x \geq \mathbf{0}\}$ where

$$A' = \begin{bmatrix} I \\ A \end{bmatrix}.$$

Then, from Corollary 8.1.3, two distinct extreme rays r^1 and r^2 of $\mathcal{C}(A')$ are adjacent if $\operatorname{rank}(A'_{\bar{\rho}}) = n - 2$ where $\rho = \operatorname{supp}_{A'}(r^1 + r^2)$. Furthermore,

$$A'_{\bar{\rho}} = \begin{bmatrix} I_{\bar{\tau}} \\ A_{\bar{\sigma}} \end{bmatrix}$$

where $\sigma = \operatorname{supp}_A(r^1 + r^2)$ and $\tau = \operatorname{supp}_I(r^1 + r^2)$. It then follows from linear algebra that $\operatorname{rank}(A'_{\bar{\rho}}) = \operatorname{rank}(A_{\bar{\sigma}\tau}) + |\bar{\tau}|$, and therefore, $\operatorname{rank}(A'_{\bar{\rho}}) = n - 2$ when $\operatorname{rank}(A_{\bar{\sigma}\tau}) = n - 2 - |\bar{\tau}| = |\tau| - 2$. So, two distinct extreme rays r^1 and r^2 of $\mathcal{C}$ are adjacent if and only if $\operatorname{rank}(A_{\bar{\sigma}\tau}) = |\tau| - 2$ Thus, we must compute the rank of a $|\bar{\sigma}|$ by $|\tau|$ matrix. The matrix $A_{\bar{\sigma}\tau}$ is potentially much smaller than $A'_{\bar{\rho}}$ and it is certainly never larger. Moreover, note that $|\rho| = |\operatorname{supp}_{A'}(r^1 + r^2)| = |\operatorname{supp}_A(r^1 + r^2)| + |\operatorname{supp}_I(r^1 + r^2)| = |\sigma| + |\tau|$, and similarly, $|\bar{\rho}| = |\bar{\sigma}| + |\bar{\tau}|$; therefore, $|\tau| \leq |\operatorname{supp}_{A'}(r^1 + r^2)|$ and $|\bar{\sigma}| \leq |\overline{\operatorname{supp}}_{A'}(r^1 + r^2)|$. Hence, if either $|\operatorname{supp}_{A'}(r^1 + r^2)|$ is small or $|\overline{\operatorname{supp}}_{A'}(r^1 + r^2)|$ is small then the size of the matrix $A_{\bar{\sigma}\tau}$ is also small. We next show how to transform cones into the form of $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}, x \geq \mathbf{0}\}$

without increasing the size of the constraint matrix, and subsequently, we show rank computations for $\mathcal{C}$ are potentially much smaller than for the original formulation; it is certainly not a larger computation.

Consider the pointed cone $\mathcal{C}(A) := \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$. We assume that any equality constraints have been removed using the transformation mentioned above. We will show that we can transform $\mathcal{C}(A)$ into a cone $\mathcal{C} = \{x \in \mathbb{R}^n : A'x \geq \mathbf{0}, x \geq \mathbf{0}\}$ where A' is an $m - n$ by n matrix. Note that the number of variables and the number of constraints does not change. Since $\mathcal{C}(A)$ is pointed, $\text{rank}(A) = n$. We assume without loss of generality that the first n rows of A are linearly independent. Let $T \in \mathbb{R}^{n \times n}$ be the matrix inverse of the square matrix consisting of the first n rows of A. Then,

$$AT = \begin{bmatrix} I \\ A' \end{bmatrix}.$$

Now, substitute Ty for x, and then,

$$\mathcal{C}(A) := \{Ty \in \mathbb{R}^n : ATy \geq \mathbf{0}\} = \{Ty \in \mathbb{R}^n : A'y \geq \mathbf{0}, y \geq \mathbf{0}\}.$$

We thus can compute the extreme rays of the cone $\mathcal{C} = \{y \in \mathbb{R}^n : A'y \geq \mathbf{0}, y \geq \mathbf{0}\}$, and then, if y is an extreme ray of $\mathcal{C}$, then Ty is an extreme ray of $\mathcal{C}(A)$ since the support of y in $\mathcal{C}$ is exactly the same as the support of $Ty = x$ in $\mathcal{C}(A)$ (i.e. $\text{supp}_{AT}(y) = \text{supp}_A(Ty)$). Now, let r^1 and r^2 be two distinct extreme rays of $\mathcal{C}(A)$, and let $\tilde{r}^1$ and $\tilde{r}^2$ be the corresponding distinct extreme rays of $\mathcal{C}$. To determine the adjacency of r^1 and r^2 in $\mathcal{C}(A)$, we need to compute the rank of a $|\overline{\text{supp}}_A(r^1 + r^2)|$ by n matrix. However, to determine the adjacency of $\tilde{r}^1$ and $\tilde{r}^2$ in $\mathcal{C}$, we need to compute the rank of a $|\bar{\sigma}|$ by $|\tau|$ matrix where $\sigma = \text{supp}_{A'}(\tilde{r}^1 + \tilde{r}^2)$ and $\tau = \text{supp}_I(\tilde{r}^1 + \tilde{r}^2)$. Crucially, $|\sigma| + |\tau| = |\,\text{supp}_A(r^1 + r^2)|$ and $|\bar{\sigma}| + |\bar{\tau}| = |\overline{\text{supp}}_A(r^1 + r^2)|$, so a $|\bar{\sigma}|$ by $|\tau|$ matrix is never larger than a $|\overline{\text{supp}}_A(r^1 + r^2)|$ by n matrix.

As per the previous formulation, there is another benefit of formulating cones in the form $\mathcal{C}(A) = \{x \in \mathbb{R}^n : A'x \geq \mathbf{0}, x \geq \mathbf{0}\}$: it is easy to find all extreme rays $r \in \mathcal{C}(A)$ such that $|\,\text{supp}_A(r)| = 1$. Note that if $|\,\text{supp}_A(r)| = 1$ for $r \in \mathcal{C}(A)$, then r must be an extreme ray since it is definitely support-minimal. Firstly, if the ith column of A' is zero (i.e. $A'_i = \mathbf{0}$), then $r = \mathbf{e}^i$ is an extreme ray of $\mathcal{C}(A)$ since $r \in \mathcal{C}(A)$ and $|\,\text{supp}_A(r)| = 1$. Secondly, if r is an extreme of $\mathcal{C}(A)$ such that $|\,\text{supp}_A(r)| = 1$, then we show below that $r = \mathbf{e}^i$ for some $i \in \{1, ..., n\}$ and the ith column of A' is zero. Note that $|\,\text{supp}_A(r)| = 1$ implies $|\,\text{supp}_I(r)| = 1$ and $|\,\text{supp}_{A'}(r)| = 0$, since $|\,\text{supp}_I(r)| = 0$ means that $r = \mathbf{0}$ implying $|\,\text{supp}'_A(r)| = 0$ as well. Finally, if $|\,\text{supp}_A(r)| = 1$ and $|\,\text{supp}_{A'}(r)| = 0$, then $r = \lambda\mathbf{e}^i$ for some $i \in \{1, ..., n\}$ and $\lambda \in \mathbb{R}_+$ and $A'r = \mathbf{0}$ implying that the ith column of A' must be zero. Thus, we can easily check for zero columns of A' to find all the extreme rays r of $\mathcal{C}(A)$ such that $|\,\text{supp}_A(r)| = 1$.

During the double description method, even after applying all of the optimisations from Section 8.2.1 to avoid performing the algebraic adjacency check, we observed that we must determine whether one given extreme ray r of $\mathcal{C}_\sigma(A)$ is adjacent to potentially many other extreme rays $r^1, ..., r^k$ of $\mathcal{C}_\sigma(A)$. Hence, we must check for adjacency of many extreme rays pairs where one extreme ray of the pair is the same. We next show how we exploit this fact.

Consider the cone $\mathcal{C} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x \geq \mathbf{0}\}$ where $A \in \mathbb{R}^{m \times n}$. Let r and $r^1, ..., r^k$ be extreme rays of $\mathcal{C}$. Recall from previously that r is adjacent to r^i if and only if $\text{rank}(A_{*\tau^i}) = |\tau^i| - 2$ where $\tau^i = \text{supp}_I(r + r^i)$. Here, $A_{*\tau^i}$ is an m by $|\tau^i|$ matrix. The important point here is that $\text{supp}_I(r) \subseteq \tau^i$ for all $i = 1, ..., k$, so each matrix $A_{*\tau^i}$ has a common submatrix $A_{*\tau}$ where $\tau = \text{supp}_I(r)$. We can use this fact to speed up the rank check. By linear algebra, there exists a non-singular matrix $T \in \mathbb{R}^{m \times m}$ such that TA is in row echelon form; that is,

$$TA_{*\tau} = \begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix}$$

where $\tilde{A}$ is a $\text{rank}(A_{*\tau})$ by $|\tau|$ matrix. Note that since r is an extreme ray of $\mathcal{C}$, we have $\text{rank}(A_{*\tau}) = |\tau| - 1$ from Corollary 2.2.12. So, $\tilde{A}$ is a $(|\tau| - 1)$ by $|\tau|$ matrix. Now, let

$$TA = \begin{bmatrix} C \\ D \end{bmatrix}$$

where C is a $(|\tau| - 1)$ by n matrix and D is a $(m - |\tau| + 1)$ by n matrix. Note that $C_{*\tau} = \tilde{A}$ and $D_{*\tau} = \mathbf{0}$ by construction of T. Then, since T is non-singular, $\text{rank}(A_{*\tau^i}) = \text{rank}(TA_{*\tau^i})$. Moreover, by construction, $\text{rank}(TA_{*\tau^i}) = |\tau| - 1 + \text{rank}(D_{*\rho^i})$ where $\rho^i = \text{supp}_I(r^i) \setminus \text{supp}_I(r)$, and thus, $\text{rank}(TA_{*\tau^i}) = |\tau^i| - 2$ if and only if $\text{rank}(D_{*\rho^i}) = |\tau^i| - 2 - |\tau| + 1 = |\rho^i| - 1$. Note that $D_{*\rho^i}$ is a $(m - |\tau| + 1)$ by $|\rho^i| = |\tau^i| - |\tau|$ matrix. Thus, for all $i = 1, ..., k$, the extreme rays r and r^i are adjacent if and only if $\text{rank}(D_{*\rho^i}) = |\rho^i| - 1$. Thus, after some initial work computing D, we have reduced the size of the rank computation for checking the adjacency of r and r^i for every $i = 1, ..., k$ from an m by $|\tau^i|$ matrix to an $(m - |\tau| + 1)$ by $(|\tau^i| - |\tau|)$ matrix.

There is another added benefit of computing D that makes performing this computation even more appealing. The benefit is that we can easily determine the set $I \subseteq \{1, ..., n\}$ where $i \in I$ if $\text{supp}_I(r') \setminus \text{supp}_I(r) = \{i\}$ for some extreme ray r' of $\mathcal{C}$. Recall from the end of Section 8.2.3 that it is desirable to find the set I for proving non-adjacency of extreme rays. The set I is precisely the index set of the zero columns of the matrix $D_{*\bar{\tau}}$ from above where $\tau = \text{supp}_I(r)$. From our discussion above, a ray r' is adjacent to r if and only if $\text{rank}(D_{*\rho'}) = |\rho'| - 1$ where $\rho' = \text{supp}_I(r') \setminus \text{supp}_I(r)$. Recall from Lemma 8.2.2 that if $\rho' = \{i\}$, then r' is adjacent to r, and consequently, $\text{rank}(D_{*\rho'}) = |\rho'| - 1 = 0$ implying that $D_{*\rho'} = D_{*i} = \mathbf{0}$. Conversely, assume that the ith column of D is zero: $D_{*i} = \mathbf{0}$. This implies that $\text{rank}(TA_{*\tau'}) = |\tau'| - 2$ where $\tau' = \tau \cup i$ because $\text{rank}(TA_{*\tau}) = |\tau| - 1$ since r is an extreme ray of $\mathcal{C}$. Let $F = \mathcal{C} \cap \{x \in \mathbb{R}^n : A_{*\bar{\tau}'}x = \mathbf{0}\}$, which is a face of $\mathcal{C}$. Then, by construction, $r \in F$ and F is a 2-dimensional face of $\mathcal{C}$, and therefore, there is another extreme ray r' of $\mathcal{C}$ in F, which is adjacent to r. The rays r and r' generate F; thus, $\text{supp}(r) \cup \text{supp}(r') = \tau'$. Moreover, since $\tau' = \tau \cup i$, $\text{supp}_I(r') \setminus \text{supp}_I(r) = \{i\}$ as required.

Example 8.2.5. *We will show how the above procedure for reducing the size of a rank computation works when applied to the cone of* 4×4 *magic squares as an example. Recall that we can write the cone of* 4×4 *magic squares in the form*

$\mathcal{C} = \{x \in \mathbb{R}^{16} : Ax = \mathbf{0}, x \geq \mathbf{0}\}$ *where*

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

The vector $r^1 = (0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0)$ *is an extreme ray of* $\mathcal{C}$ *and so is* $r^2 = (0, 0, 1, 1, 0, 1, 1, 0, 2, 0, 0, 0, 0, 1, 0, 1)$. *In this case, these extreme rays are adjacent. Recall that for cones in the form* $\mathcal{C} = \{x \in \mathbb{R}^{16} : Ax = \mathbf{0}, x \geq \mathbf{0}\}$, *the extreme rays* r^1 *and* r^2 *are adjacent if and only if* $\operatorname{rank}(A_{*\tau}) = |\tau| - 2 = 7$ *where* $\tau = \operatorname{supp}_I(r^1 + r^2) = \{3, 4, 6, 7, 9, 11, 13, 14, 16\}$. *Recall that* $A_{*\tau}$ *is the submatrix of* A *consisting of the columns of* A *indexed by* τ. *Thus, we could check whether* r^1 *was adjacent to* r^2 *by computing the rank of the matrix*

$$A_{*\tau} = \begin{bmatrix} 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \\ 1 & 1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}.$$

This matrix has rank $|\tau| - 2 = 7$, *proving that* r^1 *and* r^2 *are adjacent.*

We next show that using the method described above, the size of the rank computation significantly decreases. Let $\tau^1 = \operatorname{supp}_I(r^1) = \{3, 4, 6, 8, 9, 11, 13, 14\}$. *First, we compute the matrix* TA *such that the matrix* $TA_{*\tau^1}$ *is in row echelon form and* $T \in \mathbb{R}^{9\times 9}$ *is a transformation matrix. We have highlighted the columns of* TA *below that correspond to the columns of* $TA_{*\tau^1}$. *Note that* $TA_{*\tau^1}$ *is indeed in row echelon form and that* $\operatorname{rank}(TA_{*\tau^1}) = |\tau^1| - 1 = 7$ *since* r^1 *is an extreme ray of* $\mathcal{C}$.

$$TA = \begin{bmatrix} 1 & 0 & \mathbf{1} & \mathbf{0} & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & -1 & \mathbf{0} & -1 & \mathbf{0} & \mathbf{-1} & 0 & -1 \\ 2 & 0 & \mathbf{0} & \mathbf{1} & 0 & \mathbf{0} & -1 & \mathbf{0} & \mathbf{0} & -1 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{-1} & -1 & 1 \\ 2 & 0 & \mathbf{0} & \mathbf{0} & 1 & \mathbf{1} & 0 & \mathbf{0} & \mathbf{0} & -1 & \mathbf{0} & -1 & \mathbf{0} & \mathbf{-1} & -1 & 0 \\ 0 & -1 & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 1 & \mathbf{1} & \mathbf{0} & -1 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{-1} & 0 & 0 \\ 1 & 0 & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{1} & 0 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{-1} & -1 & -1 \\ 1 & -1 & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & -1 & \mathbf{1} & 0 & \mathbf{0} & \mathbf{-1} & 0 & 1 \\ 2 & -1 & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & -2 & \mathbf{0} & -1 & \mathbf{1} & \mathbf{-1} & 0 & 1 \\ -2 & 0 & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & 2 & \mathbf{0} & \mathbf{0} & 2 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & 0 & -2 \\ 0 & 0 & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & 0 & 0 \end{bmatrix}.$$

Note that last row is all zeros since there was a linearly dependent row in the matrix A. We can thus remove this row from the matrix TA.

Let $\tau^2 = \text{supp}_I(r^2) = \{3, 4, 6, 7, 9, 14, 16\}$. Then, the matrix D from above is the last $m - |\tau^1| + 1 = 2$ rows of TA, and after omitting the all zero last row, we have

$$D = \begin{bmatrix} \text{-2} & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & \text{-2} \end{bmatrix}.$$

Now, to check that r^2 is adjacent to r^1, we need to check whether $\text{rank}(D_{*\rho'}) = |\rho'| - 1 = 1$ *where $\rho' = \text{supp}_I(r^2) \setminus \text{supp}_I(r^1) = \{7, 16\}$, which indeed is the case as $D_{*\rho'} = [2\ \text{-2}]$. This matrix is certainly much smaller than $A_{*\tau}$, and indeed, checking the adjacency of r^1 and several other extreme rays would also be very easy in this case since we only need to compute ranks of submatrices of D.*

Furthermore, we have $I := \{2, 5, 12, 15\}$ where I is the index set of zero columns of $D_{*\tilde{r}^1}$. From our discussion above, for every $i \in I$, there exists an extreme ray r' of $\mathcal{C}$ such that $\text{supp}_I(r') \setminus \text{supp}_I(r^1) = \{i\}$. *Indeed, for the extreme ray of $\mathcal{C}$, $r' = (0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0)$, we have* $\text{supp}_I(r') \setminus \text{supp}_I(r^1) = \{2\} \in I$.

In practice, we found this transformation very useful indeed.

There is a very nice geometric interpretation of the above procedure for reducing the size of rank computations, which we now present at length. The new fundamental idea behind the geometric interpretation is that, given an extreme ray $r \in \mathcal{C}_\sigma(A)$, we can construct a pointed *smaller* cone $\mathcal{C}$ such that there is a one-to-one correspondence between the extreme rays of $\mathcal{C}$ and the extreme rays of $\mathcal{C}_\sigma(A)$ that are adjacent to r. So, instead of checking whether an extreme ray $r' \in \mathcal{C}_\sigma(A)$ is adjacent to r, we check whether there exists an extreme ray $\tilde{r}'$ of $\mathcal{C}$ that corresponds to r'. The expectation is that checking for extreme rays of $\mathcal{C}$ is *faster* than checking for adjacent extreme rays of $\mathcal{C}_\sigma(A)$. We will see that the above procedure for reducing the size of rank computations equates to constructing the cone $\mathcal{C}$ and checking for extreme rays of $\mathcal{C}$.

Additionally, we will show that the cone $\mathcal{C}$ that we are interested in is actually the projection of the cone $\mathcal{C}_\sigma(A)$ onto the hyperplane $\{x \in \mathbb{R}^n : rx = 0\}$. First, let us say exactly what we mean by projection onto a hyperplane. Given a point $x \in \mathbb{R}^n$ and a linear space $\mathcal{S} \subseteq \mathbb{R}^n$, the **projection** of x onto $\mathcal{S}$, written as $\text{proj}_\mathcal{S}(x)$, is the point $y \in \mathcal{S}$ such that $x - y \in \mathcal{S}^*$. So, the projection of $\mathcal{C}_\sigma(A)$ onto the linear space $\mathcal{S} = \{x \in \mathbb{R}^n : rx = 0\}$, written as $\text{proj}_\mathcal{S}(\mathcal{C}_\sigma(A))$, is the set of vectors $y \in \mathcal{S}$ such that $x - y \in \mathcal{S}^* = \{\lambda r : \lambda \in \mathbb{R}^n\}$ for some $x \in \mathcal{C}_\sigma(A)$.

We now demonstrate that there is a one-to-one correspondence between the extreme rays of $\text{proj}_\mathcal{S}(\mathcal{C}_\sigma(A))$ where $\mathcal{S} = \{x \in \mathbb{R}^n : rx = 0\}$ and the extreme rays of $\mathcal{C}_\sigma(A)$ that are adjacent to r. Consider the 3-dimensional cone in Figure 8.4 with extreme rays r, a, b, c, and d. Note that there are two adjacent extreme rays of r: ray a and d. We have also drawn in Figure 8.4 the projection of this cone onto the hyperplane $\mathcal{S} = \{x \in \mathbb{R}^n : rx = 0\}$, labelled as $\mathcal{C}$, and the projection of the extreme rays a, b, c, and d onto the hyperplane $\mathcal{S}$, labelled as a', b', c', and d' respectively. Observe that the two extreme rays of the projected cone $\mathcal{C}$ are a' and d', which correspond to

precisely the two adjacent extreme rays of r. The other extreme rays b and c project into the interior of $\mathcal{C}$, and the extreme ray r projects to the origin. Next, we show why this is true.

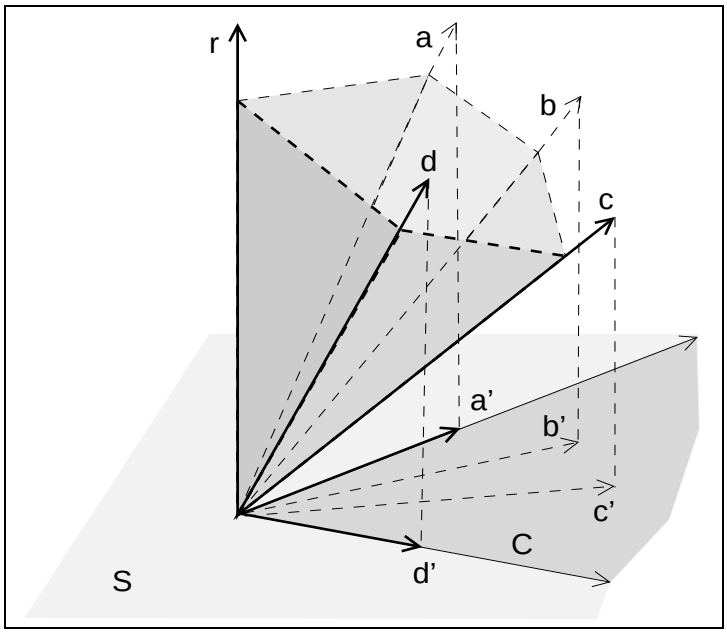

Figure 8.4: The projection of a cone onto $\{x \in \mathbb{R}^n : rx = \mathbf{0}\}$.

Recall from Lemma 8.1.4 that an extreme ray $r \in \mathcal{C}_\sigma(A)$ is adjacent to another distinct extreme ray $r' \in \mathcal{C}_\sigma(A)$ if and only if there does not exist another distinct extreme ray $r'' \in \mathcal{C}_\sigma(A)$ such that $\text{supp}_A(r'') \subseteq \text{supp}_A(r + r')$. Or equivalently, an extreme ray $r \in \mathcal{C}_\sigma(A)$ is adjacent to another distinct extreme ray $r' \in \mathcal{C}_\sigma(A)$ if and only if there does not exist another distinct extreme ray $r'' \in \mathcal{C}_\sigma(A)$ such that $\text{supp}_A(r'') \backslash \text{supp}_A(r) \subseteq \text{supp}_A(r') \backslash \text{supp}_A(r)$. In other words, r' is adjacent to r if and only if $\text{supp}_A(r') \backslash \text{supp}_A(r)$ is inclusion-minimal amongst all sets $\text{supp}_A(r'') \backslash \text{supp}_A(r)$ for all extreme rays r'' of $\mathcal{C}_\sigma(A)$ except r. This is very similar to the result that the extreme rays of a cone are the support-minimal rays of the cone as in Lemma 2.2.13. We will use this observation as a motivation to find the cone $\mathcal{C}$ that we want.

Now, consider the cone $\mathcal{C}_\sigma^\tau(A)$ where $\tau = \text{supp}_A(r)$. So, we have relaxed all the constraints which are not satisfied at equality by r. Observe that for every $r' \in \mathcal{C}_\sigma(A)$, we have $r' \in \mathcal{C}_\sigma^\tau(A)$ and the support of r' in $\mathcal{C}_\sigma^\tau(A)$ is $\text{supp}_A(r') \setminus \tau$. Hence, from the previous paragraph, for every extreme ray $r' \in \mathcal{C}_\sigma(A)$, r' is adjacent to r if r' has minimal support in $\mathcal{C}_\sigma^\tau(A)$. The cone $\mathcal{C}_\sigma^\tau(A)$ is not pointed because it has a non-trivial lineality space $\text{lin}(\mathcal{C}_\sigma^\tau(A)) = \{x \in \mathbb{R}^n : A_{\bar{\tau}} x = \mathbf{0}\}$. The ray r is in the lineality space, and moreover, the ray r actually generates the lineality space: $\text{lin}(\mathcal{C}_\sigma^\tau(A)) = \{\lambda r : \lambda \in \mathbb{R}\}$. This follows since the inclusion-minimal face of $\mathcal{C}_\sigma(A)$ containing the extreme ray r is $F = \mathcal{C}_\tau(A) = \{x \in \mathbb{R}^n : A_{\bar{\tau}} x = \mathbf{0}, A_\tau x \geq \mathbf{0}\} = \{\lambda r : \lambda \in \mathbb{R}_+\}$. We can make $\mathcal{C}_\sigma^\tau(A)$ pointed by intersecting it with the dual of its lineality space, which is simply $\{\lambda r : \lambda \in \mathbb{R}\}^* = \{x \in \mathbb{R}^n : rx = 0\}$. We will show next that the cone $\mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}$ is precisely the projection of $\mathcal{C}_\sigma^\tau(A)$ onto the hyperplane $\{x \in \mathbb{R}^n : rx = 0\}$, and that there is a one-to-one correspondence between extreme

rays of $\mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}$ and extreme rays of $\mathcal{C}_\sigma(A)$ that are adjacent to r. Note that $\mathcal{C}_\sigma^\tau(\tilde{A}) = \mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}$ where $\tilde{A}$ is the matrix A with the vector r added as an extra row after all the other rows, and the support of a ray $\tilde{r}$ in $\mathcal{C}_\sigma^\tau(\tilde{A})$ is $\text{supp}_{\tilde{A}}(\tilde{r}) \setminus \tau = \text{supp}_A(\tilde{r}) \setminus \tau$ since the extra equality constraint $rx = 0$ does not change the support.

We now show that $\mathcal{C}_\sigma^\tau(\tilde{A}) = \mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}$ is the projection of $\mathcal{C}_\sigma^\tau(A)$ onto the hyperplane $\{x \in \mathbb{R}^n : rx = 0\}$; that is, $\mathcal{C}_\sigma^\tau(\tilde{A}) = \text{proj}_{\mathcal{S}}(\mathcal{C}_\sigma(A))$. First, recall that $\text{proj}_{\mathcal{S}}(\mathcal{C}_\sigma(A))$ is the set of vectors $y \in \mathcal{S}$ such that $x - y \in \mathcal{S}^* = \{\lambda r : \lambda \in \mathbb{R}^n\}$ for some $x \in \mathcal{C}_\sigma(A)$. Then,

$$\begin{aligned}
\text{proj}_{\mathcal{S}}(\mathcal{C}_\sigma(A)) :=& \{y \in \mathbb{R}^n : x \in \mathcal{C}_\sigma(A), x - y \in \mathcal{S}^*, y \in \mathcal{S}\} \\
=& \{y \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_\sigma x \geq \mathbf{0}, x - y = \lambda r, \lambda \in \mathbb{R}, ry = 0\} \\
=& \{y \in \mathbb{R}^n : A_{\bar{\sigma}}(y + \lambda r) = \mathbf{0}, A_\sigma(y + \lambda r) \geq \mathbf{0}, \lambda \in \mathbb{R}, ry = 0\} \\
=& \{y \in \mathbb{R}^n : A_{\bar{\sigma}} y = \mathbf{0}, A_{\sigma\setminus\tau} y \geq \mathbf{0}, ry = 0\} \\
=& \mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}
\end{aligned}$$

where $\tau = \text{supp}_A(r)$. Note that $A_i r = 0$ for every $i \in \bar{\tau}$, so $A_{\sigma\setminus\tau}(y + \lambda r) \geq \mathbf{0}$ is equivalent to $A_{\sigma\setminus\tau} y \geq \mathbf{0}$. Also, note that $A_i r > 0$ for every $i \in \tau$ and thus the inequalities $A_\tau(y + \lambda r) \geq \mathbf{0}$ are always satisfied for every y after choosing an appropriate value for λ.

Next, in Lemma 8.2.6, we will show that there is a one-to-one correspondence between extreme rays of $\mathcal{C}_\sigma^\tau(\tilde{A}) = \mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}$ and extreme rays of $\mathcal{C}_\sigma(A)$ that are adjacent to r. But first, we describe how to map rays of $\mathcal{C}_\sigma(A)$ to rays of $\mathcal{C}_\sigma^\tau(\tilde{A})$. We show in Lemma 8.2.6 that this map from rays of $\mathcal{C}_\sigma(A)$ to rays of $\mathcal{C}_\sigma^\tau(\tilde{A})$ maps extreme rays of $\mathcal{C}_\sigma(A)$ that are adjacent to r to extreme rays of $\mathcal{C}_\sigma^\tau(\tilde{A})$. Firstly, we can map any ray $r' \in \mathcal{C}_\sigma(A)$ to a ray $\tilde{r}'$ in the cone $\mathcal{C}_\sigma^\tau(\tilde{A})$ by projecting r' onto the linear space $\{x \in \mathbb{R}^n : rx = 0\}$. Let $\tilde{r}' = r' - \lambda r$ where $\lambda = \frac{r'r}{rr}$. Then, $\tilde{r}' r = 0$ and $\tilde{r}' \in \mathcal{C}_\sigma^\tau(\tilde{A})$, and moreover, $\text{supp}_A(\tilde{r}') \setminus \tau = \text{supp}_A(r') \setminus \tau$. Secondly, we can map any ray $\tilde{r}' \in \mathcal{C}_\sigma^\tau(\tilde{A})$ to a ray r' in the cone $\mathcal{C}_\sigma(A)$ as follows. Let $r' = \tilde{r}' + \lambda r$ where $\lambda \in \mathbb{R}_+$ such that $A_\tau r' \geq \mathbf{0}$. This is always possible since $A_\tau r > \mathbf{0}$. Furthermore, we also choose the smallest possible λ; that is, we choose $\lambda \in \mathbb{R}_+$ such that $A_\tau r' \geq \mathbf{0}$ and $A_i r' = 0$ for some $i \in \tau$. Geometrically speaking, we lift $\tilde{r}'$ to r' by computing the first point of intersection of the line $\tilde{r}' + \lambda r$ where $\lambda \in \mathbb{R}_+$ with the cone $\mathcal{C}_\sigma(A)$. Choosing the smallest λ makes the mapping from $\mathcal{C}_\sigma^\tau(\tilde{A})$ to $\mathcal{C}_\sigma(A)$ unique. Then, $r' \in \mathcal{C}_\sigma(A)$ and $\text{supp}_A(\tilde{r}') \setminus \tau = \text{supp}_A(r') \setminus \tau$. Importantly, $\tau \subsetneq \text{supp}_A(r')$ since $A_i r' = 0$ for some $i \in \tau$. This is important for the proof of Lemma 8.2.6 because otherwise r' cannot be an extreme ray of $\mathcal{C}_\sigma(A)$.

We show in the proof of Lemma 8.2.6 below that the map between $\mathcal{C}_\sigma^\tau(\tilde{A})$ and $\mathcal{C}_\sigma(A)$ above is a bijection between the extreme rays of $\mathcal{C}_\sigma(A)$ that are adjacent to r and the extreme rays of $\mathcal{C}_\sigma^\tau(\tilde{A})$.

Lemma 8.2.6. *Let r be an extreme ray of $\mathcal{C}_\sigma(A)$. There exists an extreme ray of $\mathcal{C}_\sigma(A)$ with support ρ that is adjacent to r if and only if there exists an extreme ray of $\mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}$ where $\tau = \text{supp}_A(r)$ with support $\rho \setminus \tau$.*

Proof. Let $\mathcal{C}_\sigma^\tau(\tilde{A}) = \mathcal{C}_\sigma^\tau(A) \cap \{x \in \mathbb{R}^n : rx = 0\}$ where $\tilde{A}$ is the matrix A with the vector r added as an extra row after all the other rows.

Let $\tilde{r}'$ be an extreme ray of $\mathcal{C}_\sigma^\tau(\tilde{A})$, and let $r' \in \mathcal{C}_\sigma(A)$ such that $\tilde{r}'$ maps onto r' as described above. Then, $\text{supp}_A(r') \setminus \tau = \text{supp}_A(\tilde{r}') \setminus \tau$. We first show that r' is an extreme ray of $\mathcal{C}_\sigma(A)$. Let r'' be an extreme ray of $\mathcal{C}_\sigma(A)$ such that $\text{supp}_A(r'') \subsetneq \text{supp}_A(r')$ (implying that r' is not an extreme ray). Note that $r' \neq r$ and $r'' \neq r$ by construction of the map. Let $\tilde{r}'' \in \mathcal{C}_\sigma^\tau(\tilde{A})$ such that r'' maps onto $\tilde{r}''$ as described above. Then, $\text{supp}_A(r'') \subsetneq \text{supp}_A(r')$ implies that $\text{supp}_A(\tilde{r}'') \setminus \tau \subsetneq \text{supp}_A(\tilde{r}') \setminus \tau$ contradicting that $\tilde{r}'$ is an extreme ray of $\mathcal{C}_\sigma^\tau(\tilde{A})$. Next, we show that r' is adjacent to r. Now, let $r''' \in \mathcal{C}_\sigma(A)$ be an extreme ray of $\mathcal{C}_\sigma(A)$ such that $\text{supp}_A(r''') \setminus \tau \subsetneq \text{supp}_A(r') \setminus \tau$ or equivalently $\text{supp}_A(r''') \subsetneq \text{supp}_A(r' + r)$ (implying that r' and r are not adjacent by Lemma 8.1.4), and let $\tilde{r}''' \in \mathcal{C}_\sigma^\tau(\tilde{A})$ such that r''' maps onto $\tilde{r}'''$ as described above. Now, $\text{supp}_A(r''') \setminus \tau \subsetneq \text{supp}_A(r') \setminus \tau$ implies that $\text{supp}_A(\tilde{r}''') \setminus \tau \subsetneq \text{supp}_A(\tilde{r}') \setminus \tau$, which contradicts that $\tilde{r}'$ is an extreme ray of $\mathcal{C}_\sigma^\tau(\tilde{A})$.

Conversely, let r' be an extreme ray of $\mathcal{C}_\sigma(A)$ that is adjacent to r, and let $\tilde{r}' \in \mathcal{C}_\sigma^\tau(\tilde{A})$ such that r' maps onto $\tilde{r}'$ as described above. Then, $\text{supp}_A(r') \setminus \tau = \text{supp}_A(\tilde{r}') \setminus \tau$. We must show that $\tilde{r}'$ is an extreme ray of $\mathcal{C}_\sigma^\tau(\tilde{A})$. Let $\tilde{r}''$ be an extreme ray of $\mathcal{C}_\sigma^\tau(\tilde{A})$ such that $\text{supp}_A(\tilde{r}'') \setminus \tau \subsetneq \text{supp}_A(\tilde{r}') \setminus \tau$ (implying that $\tilde{r}'$ is not an extreme ray), and let $r'' \in \mathcal{C}_\sigma(A)$ such that $\tilde{r}''$ maps onto r'' as described above. From above, r'' is an extreme ray of $\mathcal{C}_\sigma(A)$. Now, $\text{supp}_A(\tilde{r}'') \setminus \tau \subsetneq \text{supp}_A(\tilde{r}') \setminus \tau$ implies that $\text{supp}_A(r'') \setminus \tau \subsetneq \text{supp}_A(r') \setminus \tau$ or equivalently $\text{supp}_A(r'') \subsetneq \text{supp}_A(r' + r)$, which contradicts that r' and r are adjacent by Lemma 8.1.4.

□

So, combining Lemma 8.2.6 with the observation that the cone $\mathcal{C}_\sigma^\tau(\tilde{A})$ is the cone $\mathcal{C}_\sigma(A)$ projected onto the plane $\mathcal{S} = \{x \in \mathbb{R}^n : rx = 0\}$, we arrive at the nice and new result that an extreme ray r' is adjacent to r if and only if the projection of r' onto $\mathcal{S}$ is an extreme ray of the projection of $\mathcal{C}_\sigma(A)$ onto $\mathcal{S}$.

We now show that the previous procedure for reducing the size of rank computations for determining adjacency for the cone $\mathcal{C} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x \geq \mathbf{0}\}$ is actually equivalent to projecting the cone $\mathcal{C}$ onto a hyperplane and performing rank computations in the projected cone. Let r be an extreme ray of $\mathcal{C}$, and let $\tau = \text{supp}_I(r)$. Then, from our discussion above, the projection of $\mathcal{C}$ onto $\mathcal{S} = \{x \in \mathbb{R}^n : rx = 0\}$ is the cone $\mathcal{C}' = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, rx = 0, x_{\bar{\tau}} \geq \mathbf{0}\}$. Note that we have relaxed the constraints $x_\tau \geq \mathbf{0}$. Next, we reformulate the cone $\mathcal{C}'$ by eliminating the x_τ variables. Let $\tilde{A} \in \mathbb{R}^{(m+1)\times n}$ be the matrix A with r added as the last row, so $\mathcal{C}' = \{x \in \mathbb{R}^n : \tilde{A}x = \mathbf{0}, x_{\bar{\tau}} \geq \mathbf{0}\}$. Now, note that $\text{rank}(\tilde{A}_{*\tau}) = |\tau|$ since $\mathcal{C}'$ is a pointed cone (a projection of a pointed cone is pointed). Let T be a transformation matrix such that $T\tilde{A}_{*\tau}$ is in row echelon form; that is,

$$T\tilde{A}_{*\tau} = \begin{bmatrix} I \\ \mathbf{0} \end{bmatrix} \text{ and } T\tilde{A} = \begin{bmatrix} C \\ D \end{bmatrix}$$

where C is a $|\tau|$ by n matrix and D is a $(m - |\tau| + 1)$ by n matrix such that $C_{*\tau} = I$ and $D_{*\tau} = \mathbf{0}$. Then,

$$\begin{aligned} \mathcal{C}' &= \{x \in \mathbb{R}^n : \tilde{A}x = \mathbf{0}, x_{\bar{\tau}} \geq \mathbf{0}\} \\ &= \{x \in \mathbb{R}^n : T\tilde{A}x = \mathbf{0}, x_{\bar{\tau}} \geq \mathbf{0}\} \\ &= \{x \in \mathbb{R}^n : x_\tau = -C_{*\bar{\tau}}x_{\bar{\tau}}, D_{*\bar{\tau}}x_{\bar{\tau}} = \mathbf{0}, x_{\bar{\tau}} \geq \mathbf{0}\} \end{aligned}$$

It follows that if we wish to check for extreme rays of $\mathcal{C}'$ it suffices to just consider the cone $\mathcal{C}'' = \{x \in \mathbb{R}^{|\bar{\tau}|} : D_{*\bar{\tau}}x = \mathbf{0}, x \geq \mathbf{0}\}$ since the variables x_τ can effectively be ignored. Observe that $D_{*\bar{\tau}}$ is a $(m - |\tau| + 1)$ by $|\bar{\tau}|$ matrix. Now, let r and r' be extreme rays of $\mathcal{C}$. It follows from Lemma 8.2.6 that r' is adjacent to r if and only if there exists an extreme ray of $\mathcal{C}'$ with support $\rho = \text{supp}_I(r') \setminus \tau$ or equivalently there exists an extreme ray of $\mathcal{C}''$ with support ρ. Moreover, there is an extreme ray of $\mathcal{C}''$ with support ρ if and only if $\text{rank}(D_{*\rho}) = |\rho| - 1$ from Corollary 2.2.12. This rank check for adjacency is the same rank check as for the previous procedure; indeed, the matrix D defined here is basically the same matrix as the matrix D defined in the previous procedure.

Furthermore, from our discussion earlier in this section (before Example 8.2.5), an extreme ray r'' of the cone $\mathcal{C}'' = \{x \in \mathbb{R}^{|\bar{\tau}|} : D_{*\bar{\tau}}x = \mathbf{0}, x \geq \mathbf{0}\}$ has $\text{supp}_I(r'') = \{i\}$ if and only if $D_{*i} = \mathbf{0}$ (i.e. the ith column of D is zero). This means that the set of rays r' of $\mathcal{C}''$ where $\text{supp}_I(r') \setminus \tau = \{i\}$ for some $i \in \bar{\tau}$ corresponds precisely to set of zero columns of the matrix $D_{*\bar{\tau}}$. This is exactly the result we found earlier for the previous procedure.

It is also possible to perform a similar procedure as above that reduces the size of rank computations for cones in the form $\mathcal{C} = \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}, x \geq \mathbf{0}\}$; we leave the details to the reader.

Using the different formulations presented in this section is not necessarily a good idea since the transformations involved may mean losing specific structure of the matrix A that could have otherwise been used to speed up rank computations and the transformations may cause numerical problems.

8.2.4 Adjacency inference

In this section, we examine what we can infer about the adjacencies of the extreme rays of $\mathcal{C}_\sigma(A)$ from our knowledge of the adjacencies of the extreme rays of $\mathcal{C}^i_\sigma(A)$. We investigate whether this information would be useful for the double description method. Fukuda and Prodon in [37] claim that it is useful, but this is not what we found in our experience. Moreover, they only show how to use this information when applying a fixed order for inserting constraints (independent of the input). We show that it might be used even when the order is not fixed.

Consider again the polytope $\mathcal{P}$, half-space $\mathcal{H}$, affine hyperplane H as defined above from Figure 8.1, which is repeated as Figure 8.5(i). In the context of polytopes, the concept of the adjacency of two extreme rays translates into the adjacency of extreme points where two extreme points are adjacent if there is an edge between them. Firstly, (i) observe that the vertices a and d are adjacent in $\mathcal{P}$ and that they are also adjacent in $\mathcal{P} \cap \mathcal{H}$. Secondly, (ii) observe that the vertices a and c both lie in H and they are not adjacent in $\mathcal{P}$, but they are adjacent in $\mathcal{P} \cap \mathcal{H}$. Thirdly, (iii) observe that the new vertex i of $\mathcal{P} \cap \mathcal{H}$, which is created from the intersection of H with the edge between e and f, is adjacent to e in $\mathcal{P} \cap \mathcal{H}$, and also, observe that the new vertex j of $\mathcal{P} \cap \mathcal{H}$, which is created from the intersection of H with the edge between f and g, is adjacent to g in $\mathcal{P} \cap \mathcal{H}$. Lastly, (iv) observe that the new vertex

i is not adjacent to d, g, or h which all do not lie on H, and that the new vertex j is not adjacent to d, e, or h which, again, all do not lie on H.

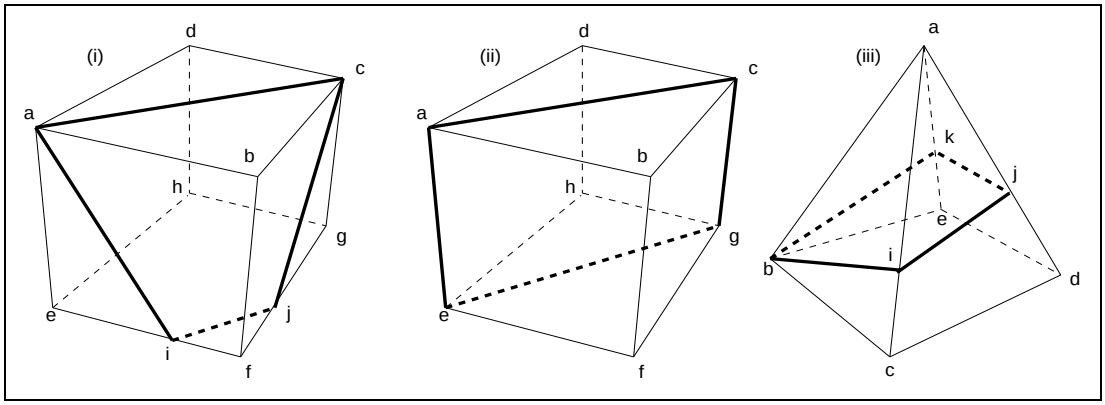

Figure 8.5: Adjacency Inference.

Generalising these observations, we induce the following for any polytope $\mathcal{P}$ and any half-space $\mathcal{H}$ with affine hyperplane H: (i) if two vertices of $\mathcal{P}$ are adjacent in $\mathcal{P}$ and lie in $\mathcal{H}$, they are also adjacent in $\mathcal{P} \cap \mathcal{H}$; (ii) if two vertices of $\mathcal{P}$ that lie in $\mathcal{H}$ are not adjacent in $\mathcal{P}$ but are adjacent in $\mathcal{P} \cap \mathcal{H}$, then they must lie on the hyperplane H; (iii) a new vertex of $\mathcal{P} \cap \mathcal{H}$, which is always created from the intersection of H with an edge of $\mathcal{P}$, is adjacent to the vertex of the edge lying in $\mathcal{H}$; (iv) and lastly, a new vertex of $\mathcal{P} \cap \mathcal{H}$ cannot be adjacent to any vertex of $\mathcal{P}$ that lies in $\mathcal{H}$ but not on H except where the previous rule (iii) applies.

There are a couple of things that are true for Figure 8.5(i), but that are not true in general. Firstly, it is *not* true that two non-adjacent vertices of $\mathcal{P}$ that lie on H are *always* adjacent in $\mathcal{P} \cap \mathcal{H}$. This is demonstrated in Figure 8.5(ii). The vertices a and g are not adjacent in $\mathcal{P}$ and they lie on the hyperplane H defined by a, c, e, and g, but they are not adjacent in $\mathcal{P} \cap \mathcal{H}$ where $\mathcal{H}$ is the halfspace defined by H extending into the page. Secondly, in Figure 8.5(i), the new vertices i and j are adjacent in $\mathcal{P} \cap \mathcal{H}$, but it is not true that vertices created from edges that have a common vertex are always adjacent. This is demonstrated in Figure 8.5(iii). Here, $\mathcal{P}$ is the pyramid with vertices a, b, c, and d. The hyperplane H intersects $\mathcal{P}$ at b, i, j, and k, and $\mathcal{H}$ is defined by H and does not include the vertex a. Vertices i and k are created from edges with the vertex a in common, but they are not adjacent.

Without any additional information other than adjacencies of $\mathcal{P}$, we really cannot deduce anything about the adjacencies of $\mathcal{P} \cap \mathcal{H}$ other than the four points listed above. More concisely, if we wish to know all the adjacencies of $\mathcal{P} \cap \mathcal{H}$, we must check all pairs of extreme rays of $\mathcal{P} \cap \mathcal{H}$ that lie on H that where either non-adjacent in $\mathcal{P}$ or at least one extreme ray in the pair is new. All the other adjacencies are determined by the adjacencies of $\mathcal{P}$.

We can transpose the three observations above to apply to cones.

Lemma 8.2.7. *Let r^1 and r^2 be two distinct extreme rays of the cone $\mathcal{C}^i_\sigma(A)$.*

(i). If r^1 and r^2 are adjacent in $\mathcal{C}^i_\sigma(A)$, $A_i r^1 \geq 0$, and $A_i r^2 \geq 0$, then r^1 and r^2 are adjacent in $\mathcal{C}_\sigma(A)$.

(ii). If r^1 and r^2 are not adjacent in $\mathcal{C}^i_\sigma(A)$ but are adjacent in $\mathcal{C}_\sigma(A)$, then $A_i r^1 = 0$ and $A_i r^2 = 0$.

(iii). If r^1 and r^2 are adjacent in $\mathcal{C}^i_\sigma(A)$, $A_i r^1 > 0$, and $A_i r^2 < 0$, then $r = (-A_i r^2) r^1 + (A_i r^1) r^2$ is adjacent to r^1 in $\mathcal{C}_\sigma(A)$.

(iv). If r is an extreme ray of $\mathcal{C}_\sigma(A)$ but not an extreme ray of $\mathcal{C}^i_\sigma(A)$ and $A_i r^1 > 0$, then r is not adjacent to r^1 in $\mathcal{C}_\sigma(A)$ unless the previous rule (iii) applies.

Proof. (i) If F is a face of $\mathcal{C}^i_\sigma(A)$, then $F \cap \{x \in \mathbb{R}^n : A_i x \geq 0\}$ is also a face of $\mathcal{C}_\sigma(A)$. So, if r^1 and r^2 generate a face of $\mathcal{C}^i_\sigma(A)$, $A_i r^1 \geq 0$, and $A_i r^2 \geq 0$, then r^1 and r^2 must also generate a face of $\mathcal{C}_\sigma(A)$, so by definition, r^1 and r^2 are adjacent in $\mathcal{C}_\sigma(A)$.

(ii) Assume r^1 and r^2 are adjacent in $\mathcal{C}_\sigma(A)$ and that $A_i r^1 > 0$ and/or $A_i r^2 > 0$. By Corollary 8.1.3, $\operatorname{rank}(A_{\bar{\tau}}) = n - 2$ where $\tau = \operatorname{supp}_A(r^1 + r^2)$ since r^1 and r^2 are adjacent in $\mathcal{C}_\sigma(A)$. Crucially, $i \notin \overline{\operatorname{supp}}_A(r^1 + r^2)$, and thus, $\bar{\tau} = \overline{\operatorname{supp}}_A(r^1 + r^2) = \overline{\operatorname{supp}}_A(r^1 + r^2) \setminus i$ implying that r^1 and r^2 are adjacent in $\mathcal{C}^i_\sigma(A)$ by Corollary 8.1.3 since $\operatorname{rank}(A_{\bar{\tau}}) = n - 2$ and $i \notin \bar{\tau}$.

(iii) Suppose r^1 and r^2 are adjacent in $\mathcal{C}^i_\sigma(A)$. The rays r^1 and r^2 generate a 2-dimensional face F of $\mathcal{C}^i_\sigma(A)$ since they are adjacent in $\mathcal{C}^i_\sigma(A)$. The ray $r = (-A_i r^2) r^1 + (A_i r^1) r^2$ is an extreme ray of $\mathcal{C}_\sigma(A)$ by Lemma 8.1.5. The set $F' = F \cap \{x \in \mathbb{R}^n : A_i x \geq 0\}$ is a 2-dimensional face of $\mathcal{C}_\sigma(A)$. Moreover, $r^1, r \in F'$, and r^1 and r are extreme rays of F'; thus, r^1 and r generate F' and are, therefore, adjacent in $\mathcal{C}_\sigma(A)$.

(iv) Assume r is an extreme ray of $\mathcal{C}_\sigma(A)$ but not an extreme ray of $\mathcal{C}^i_\sigma(A)$. Let r^3 and r^4 be extreme rays of $\mathcal{C}^i_\sigma(A)$ such that $A_i r^3 > 0$, $A_i r^4 < 0$, and $r = (-A_i r^4) r^3 + (A_i r^3) r^4$. Also, assume that $A_i r^1 > 0$ but $r^1 \neq r^3$. Observe that $\operatorname{supp}_A(r^3) \subseteq \operatorname{supp}_A(r + r^1)$ since $\operatorname{supp}_A(r^3) \setminus i \subseteq \operatorname{supp}_A(r) \setminus i$ and $i \in \operatorname{supp}_A(r^1)$, but $\operatorname{supp}_A(r^3) \not\subseteq \operatorname{supp}_A(r)$ and $\operatorname{supp}_A(r^3) \not\subseteq \operatorname{supp}_A(r^1)$. Therefore, by Lemma 8.1.4, r and r^1 are not adjacent in $\mathcal{C}_\sigma(A)$. □

All of the four points of Lemma 8.2.7 put together imply that if we know all the adjacencies for the cone $\mathcal{C}^i_\sigma(A)$ and if we wish to know all the adjacencies for the cone $\mathcal{C}_\sigma(A)$, we only need to check between extreme rays r^1 and r^2 of $\mathcal{C}_\sigma(A)$ in the following situation: $A_i r^1 = 0$, $A_i r^2 = 0$, r^1 and r^2 are not adjacent extreme rays in $\mathcal{C}^i_\sigma(A)$, and r^1 was not created from r^2 and neither was r^2 created from r^1.

It might seem at this point then that it would be a good idea to store all the adjacencies from iteration to iteration during the double description method; however, we argue that this is not the case (Fukuda and Prodon agree with this view in [37]). There are several problems with this approach. Firstly, many pairs of adjacent extreme rays are never used to create a new extreme ray, and hence, we perform additional unnecessary work computing all adjacencies. If r^1 and r^2 are adjacent extreme rays of $\mathcal{C}^i_\sigma(A)$, then only if $A_i r^1 > 0$ and $A_i r^2 < 0$ (or vice-versa) is this pair useful. We should really only concern ourselves with the adjacency of extreme pairs that are used at some point in the algorithm to create a new extreme ray. Secondly, for a given pair of non-adjacent extreme rays, we might recompute that they are

non-adjacent many times, because of Lemma 8.2.7 part (ii), when we only really need to compute this once or perhaps never. We now elaborate on why this is. Let $\tau \subseteq \sigma$ such that $\mathcal{C}_\sigma^\tau(A)$ is pointed, and let r^1 and r^2 be non-adjacent extreme rays of $\mathcal{C}_\sigma^\tau(A)$ that are also non-adjacent extreme rays of $\mathcal{C}_\sigma(A)$. At every iteration of the double description method to compute the extreme rays of $\mathcal{C}_\sigma(A)$ from the extreme rays of $\mathcal{C}_\sigma^\tau(A)$, we would need to re-check if r^1 and r^2 were adjacent for every $i \in \tau$ where $A_i r^1 = 0$ and $A_i r^2 = 0$, but we only really need to check whether r^1 and r^2 are adjacent at most once.

The solution around these problems is to perform the double description method as described in previous sections, which only computes an adjacency when necessary, and additionally use Lemma 8.2.7 to infer adjacency or non-adjacency if possible.

We now show how we can use Lemma 8.2.7 parts (i) and (iii) to imply that two extreme rays are adjacent during an iteration of the double description method. Assume that we are at some iteration of the double description method with some $\tau \subseteq \sigma$ and we have selected some $i \in \tau$. So, we are trying to compute the extreme rays of $\mathcal{C}_\sigma^{\tau\text{-}i}(A)$ from the extreme rays of $\mathcal{C}_\sigma^\tau(A)$. Let r^1 and r^2 be adjacent extreme rays of $\mathcal{C}_\sigma^\tau(A)$ such that $A_i r^1 > 0$ and $A_i r^2 < 0$, and let $r = (-A_i r^2) r^1 + (A_i r^1) r^2$. In this situation, we say that r^1 is the **father** of r, and we write $f(r) := r^1$. By Lemma 8.2.7(iii), r and r^1 are adjacent extreme rays of $\mathcal{C}_\sigma^{\tau\text{-}i}(A)$ (i.e. r and r^1 are adjacent in the next iteration). Moreover, in any subsequent iteration, if r and r^1 are still feasible, they are still adjacent extreme rays by Lemma 8.2.7(i). The ray r^2 is also a *parent* of r, but it is not an extreme ray of $\mathcal{C}_\sigma^{\tau\text{-}i}(A)$, so we do not need to keep track of r^2 as a parent of r. Some extreme rays r exist from the start of the algorithm and so have no father, in which case, we set $f(r) := \mathbf{0}$. Then, in general, at an iteration of the double description method with some $\tau \subseteq \sigma$, two extreme rays r^1 and r^2 of $\mathcal{C}_\sigma^\tau(A)$ are adjacent in $\mathcal{C}_\sigma^\tau(A)$ if either $f(r^1) = r^2$ or $f(r^2) = r^1$. If we apply this check without any of the other optimisations mentioned above to our running example, we find that out of the 150,332 adjacent extreme ray pairs, only 2,712 if these pairs were shown to be adjacent by applying this *father* check. Moreover, if we apply this check after applying the quick checks for adjacency mentioned in Section 8.2.1, we find that out of the 68,934 adjacent extreme ray pairs, only 28 times did this check apply where the other checks did not. Thus, in our experience, this check does not apply often enough for it to be useful. It is quick to check though, and perhaps, for some cones that we have yet to try, it might be very useful, but we doubt this since it applies in very special cases.

We now show how we can use Lemma 8.2.7 parts (ii) and (iv) to imply that two extreme rays are not adjacent during the double description method. Let τ^j be the value of τ in the jth iteration of Algorithm 11, and let $i^j \in \tau^j$ be the i selected in the jth iteration, so $\tau^{j+1} = \tau^j \setminus i^j$. Assume that we are currently in the kth iteration of Algorithm 11. Let r^1 be an extreme ray of $\mathcal{C}_\sigma^{\tau^k}(A)$, and let j_1 be the iteration in which r^1 was created. This means that r^1 is not an extreme ray of $\mathcal{C}_\sigma^{\tau^{j_1}}(A)$, but r^1 is an extreme ray for the next iteration and every subsequent iteration until the current kth iteration; that is, r^1 is an extreme ray of $\mathcal{C}_\sigma^{\tau^j}(A)$ for all $j = j^1 + 1, ..., k$. And, similarly, let r^2 be an extreme ray of $\mathcal{C}_\sigma^{\tau^k}(A)$, and let j_2 be the iteration in which r^2 was created.

Consider the set $(\overline{\text{supp}}_A(r^1) \setminus \tau^k) \cap \tau^{j_1}$. This is the set of indices $i \in \{i^{j_1}, i^{j_1+1}, ..., i^k\}$

such that $A_i r^1 = 0$. Thus, the set $((\overline{\text{supp}}_A(r^1) \setminus \tau^k) \cap \tau^{j_1}) \cap ((\overline{\text{supp}}_A(r^2) \setminus \tau^k) \cap \tau^{j_2})$ is the set of indices $i \in \{\max\{i^{j_1}, i^{j_2}\}, ..., i^k\}$ where $A_i r^1 = 0$ and $A_i r^2 = 0$. Thus, if $((\overline{\text{supp}}_A(r^1) \setminus \tau^k) \cap \tau^{j_1}) \cap ((\overline{\text{supp}}_A(r^2) \setminus \tau^k) \cap \tau^{j_2}) = \emptyset$, then if r^1 and r^2 were initially non-adjacent (when r^1 and r^2 were first both extreme rays), then they are still non-adjacent in the current iteration by Lemma 8.2.7(ii). Finally, r^1 and r^2 are initially non-adjacent if $f(r^1) \neq r^2$ and $f(r^2) \neq r^1$, and either $A_{j^2} r^1 > 0$ and $j^2 > j^1$ (i.e. r^2 was created after r^1), or $A_{j^1} r^2 > 0$ and $j^1 > j^2$ (i.e. r^1 was created after r^2) by Lemma 8.2.7(iv). We now arrive at the following result. The extreme rays r^1 and r^2 are not adjacent in $\mathcal{C}_\sigma^{\tau^k}(A)$ if $f(r^1) \neq r^2$, $f(r^2) \neq r^1$, and $((\overline{\text{supp}}_A(r^1) \setminus \tau^k) \cap \tau^{j_1}) \cap ((\overline{\text{supp}}_A(r^2) \setminus \tau^k) \cap \tau^{j_2}) = \emptyset$. Note that if $j^2 > j^1$ and $A_{j^2} r^1 > 0$ or $j^1 > j^2$ and $A_{j^1} r^2 > 0$, then $((\overline{\text{supp}}_A(r^1) \setminus \tau^k) \cap \tau^{j_1}) \cap ((\overline{\text{supp}}_A(r^2) \setminus \tau^k) \cap \tau^{j_2}) \neq \emptyset$.

To implement this check, at each iteration $k = 1, ..., n$, for every extreme ray r, we store the set $(\overline{\text{supp}}_A(r) \setminus \tau^k) \cap \tau^j$ where the extreme ray r was created in the jth iteration. These sets are easy to compute initially and update each iteration for each extreme ray.

If we apply this check without any of the other optimisations mentioned above in Section 8.2.1 to our running example, we find that out of the 1,277,014,866 adjacency checks, the above check determines that two extreme rays are not adjacent 578,003,758 many times. So, 45.3% of the time, this quick check was useful. The quick check is thus effective but not nearly as effective as the other quick checks for non-adjacency in Section 8.2.1 on this example. We then tried this check after applying the other optimisation mentioned in Section 8.2.1, and we found that out of the 87,476 extreme ray pairs that were not shown to be adjacent, this check showed that 6,769 many extreme rays pairs were not adjacent. This is not bad, but it did not result in an overall speed-up of the algorithm due to the time spent performing the check.

In conclusion, we have shown that it is possible to perform checks based upon adjacency inference without a fixed insertion ordering, which is a slight extension of the results of Fukuda and Prodon in [37]. In practice, we did not find either of the checks in this section useful; however, Fukuda and Prodon in [37] did, and we think that it is certainly plausible that this check could be useful for some classes of cones particularly non-degenerate cones. Also, as we mentioned before, the performance of the algorithm greatly depends upon its implementation, so perhaps, if we had implemented our algorithm in a different manner we might have also found these checks useful.

8.3 Alternative approach

Before presenting some computational results, we wish to describe briefly a variant of the double description method that is very similar to the previous approach of Algorithm 11. We have implemented this variant, and we did not find that it was competitive with the double description method as described above. However, perhaps if we had a better implementation of the variant or if we tested it on a different set of cones, this variant would have been competitive. For this reason, we feel it necessary to describe this variant in order to give a more complete picture

of the possible variants of the double description algorithms for computing extreme rays of cones.

The variant is fundamentally the same in that it computes the set of extreme rays of the cone $\mathcal{C}_\sigma(A)$ from the set of extreme rays of $\mathcal{C}^i_\sigma(A)$, but it differs slightly in how it determines new extreme rays of $\mathcal{C}_\sigma(A)$ that were not extreme rays of $\mathcal{C}^i_\sigma(A)$. Let r^1 and r^2 be extreme rays of a pointed cone $\mathcal{C}^i_\sigma(A)$ such that $A_i r^1 > 0$ and $A_i r^2 < 0$, and let $r = (-A_i r^2)r^1 + (A_i r^1)r^2$. The previous approach for determining whether r is an extreme ray of $\mathcal{C}_\sigma(A)$ was to check whether r^1 and r^2 are adjacent extreme rays of $\mathcal{C}^i_\sigma(A)$. For this variant of the double description method, we check directly whether r is an extreme ray of $\mathcal{C}_\sigma(A)$.

As for Algorithm 11 above, there are two possible approaches for checking whether r is an extreme ray of $\mathcal{C}_\sigma(A)$: the algebraic approach and the combinatorial approach.

The algebraic approach is to check whether $\text{rank}(A_{\bar{\tau}}) = n - 1$ where $\tau = \text{supp}_A(r)$ because, from Corollary 2.2.12, r is an extreme ray of $\mathcal{C}_\sigma(A)$ if and only if $\text{rank}(A_{\bar{\tau}}) = n - 1$. This is essentially identical to the algebraic check whether r^1 and r^2 are adjacent because, in that case, we check whether $\text{rank}(A_{\bar{\tau}'}) = n - 2$ where $\tau' = \text{supp}_A(r^1 + r^2) = \tau \cup \{i\}$, so $A_{\bar{\tau}'}$ is the matrix $A_{\bar{\tau}}$ without the ith row. Hence, for the algebraic check there is real no difference between this variant and the above method, but there is a difference for the combinatorial approach, which we describe next.

The combinatorial approach is to check whether there exists another extreme ray $r' \in \mathcal{C}_\sigma(A)$ such that $\text{supp}_A(r') \subsetneq \text{supp}_A(r)$ because, from Lemma 2.2.13, r is an extreme ray of $\mathcal{C}_\sigma(A)$ if and only if there does not exist another extreme ray $r' \in \mathcal{C}_\sigma(A)$ such that $\text{supp}_A(r') \subsetneq \text{supp}_A(r)$. The problem here is that we do not yet know all of the extreme rays of $\mathcal{C}_\sigma(A)$, but we do know a superset of all the extreme rays of $\mathcal{C}_\sigma(A)$. Let R be the set of extreme rays of $\mathcal{C}^i_\sigma(A)$. Let $R^+ := \{r \in R : A_i r > 0\}$, $R^= := \{r \in R : A_i r = 0\}$, $R^- := \{r \in R : A_i r < 0\}$, and $R' := \{(-A_i r^2)r^1 + (A_i r^1)r^2 : r^1 \in R^+, r^2 \in R^-\}$. Then, we know from Lemma 8.1.5 that the set of all extreme rays of $\mathcal{C}_\sigma(A)$ is contained within the set $(R' \cup R^+ \cup R^=)$. Thus, r is an extreme ray of $\mathcal{C}_\sigma(A)$ if and only if there does not exist a vector $r' \in (R' \cup R^+ \cup R^=)$ such that $\text{supp}_A(r') \subsetneq \text{supp}_A(r)$. Thus, the combinatorial approach is to first compute R' and then to check for all $r \in R'$ whether there exists a ray $r' \in (R' \cup R^+ \cup R^=)$ such that $\text{supp}_A(r') \subsetneq \text{supp}_A(r)$. Note that we can use all of the optimisations for eliminating extreme rays pairs to reduce the size of R' from Section 8.2.1, and we can also use the tree structure from Section 8.2.2 to speed up the combinatorial check. Furthermore, note that $A_i r' = 0$ for all $r' \in R'$, but $A_i r'' \neq 0$ for all $r'' \in R^+$, so for all $r' \in R'$, we can never have $\text{supp}_A(r'') \subseteq \text{supp}_A(r')$ for any $r'' \in R^+$; therefore, for all $r \in R'$ we only need to check whether there exists a ray $r' \in R' \cup R^=$ such that $\text{supp}_A(r') \subsetneq \text{supp}_A(r)$. We can now present the variant of the double description method using the combinatorial check in Algorithm 13.

The difference between this alternate approach and the previous is that, in this alternate approach, we perform the support check over the set $(R' \cup R^=)$ but in the previous approach we perform the support over the set R. The problem with this approach that we noticed was that the size of the set $(R' \cup R^=)$ may be much larger than R even after eliminating critical pairs, although it also may be much smaller

Algorithm 13 Alternate Ray Algorithm

Input: a pointed cone $\mathcal{C}_\sigma(A)$
Output: the set R of extreme rays of $\mathcal{C}_\sigma(A)$.
  Find a set $\tau \subseteq \sigma$ such that $\mathcal{C}^\tau_\sigma(A)$ is pointed.
  Compute the minimal set R of extreme rays of $\mathcal{C}^\tau_\sigma(A)$.
  while $\tau \neq \emptyset$ **do**
    Select $i \in \tau$.
    $R^+ := \{r \in R : A_i r > 0\}$.
    $R^= := \{r \in R : A_i r = 0\}$.
    $R^- := \{r \in R : A_i r < 0\}$.
    $R' := \{(-A_i r^2)r^1 + (A_i r^1)r^2 : r^1 \in R^+, r^2 \in R^-\}$
    $R := R^+ \cup R^=$.
    for all $r' \in R'$ **do**
      if r' is support-minimal in $(R' \cup R^=)$ **then**
        $R := R \cup \{r'\}$.
      end if
    end for
    $\tau := \tau \setminus i$.
  end while
  return R

than R as well, which is why we cannot definitively say that this alternative method is inferior to the previous approach.

8.4 Computational results

In this section, we give some computational results of computing extreme rays of cones. All results in this section are listed in seconds and were computed on an Intel XEON 3.2 GHz machine with 4Gb of RAM running Linux Redhat. The table entries with a "$*$" indicate after several hours of computation time the computation was still nowhere near completion.

In Table 8.2, we list times in seconds for computing the extreme rays of various different cones. The first column gives the name of the cone. The example 55 is the cone of 5×5 magic squares and 66 is the cones of 6×6 magic squares. The cone 66*pan* is the cone of 6×6 panmagic squares; a panmagic square is similar to a normal magic square except that there are additional constraints on the diagonals. The other cones were taken from the paper of Fukuda and Prodon [37]. The examples *cube*14 is a hypercube of dimension 14 and *cube*16 is a hypercube of dimension 16. The cones *cross*8, *cross*10, and *cross*12 are the dual cones of a hypercube of dimension 8, 10, and 12 respectively. The cones *ccc*6 and *ccc*7 are complete cut cones, and *ccp*6 and *ccp*7 are cut polytopes. The cone *prodmT*5 is another cone from [37], which is based upon a graph. The second column is the number of extreme rays that were computed for the cone. The third column is the dimension of the cone, and the fourth column is the number of inequality constraints. The next four columns list the times taken to compute the extreme rays using different insertion orders (the order in which

we select constraints); these are the min-cut-off, max-cut-off, max-intersection, and lexicographic insertion orders.

The variant of the double description method that we used here is as described in Algorithm 12 with all the optimisations for checking for adjacency. This algorithm is implemented in the Software package $4ti2$ version 1.3 (with some minor changes) ([1]). We use the algebraic rank check for adjacency here, and we compare the algebraic adjacency check with the combinatorial adjacency check later in this section. We use 64 bit integers with arithmetic overflow checking to perform the rank computations.[3]

Problem	Size	dim	m	mincut	maxcut	maxinter	lex
55	1940	15	25	0.60	0.60	0.04	0.05
66	97548	25	36	13.67	14.27	10.10	12.61
66pan	265536	17	36	201.06	184.28	167.73	172.29
ccc6	210	15	31	0.10	0.12	0.12	0.07
ccc7	38780	21	63	6234.10	*	*	587.04
ccp6	368	15	32	0.21	0.28	0.32	0.15
ccp7	116764	21	64	18743.86	*	*	2152.13
cube14	16384	14	28	0.43	0.42	0.43	0.42
cube16	65536	16	32	2.12	2.05	2.11	2.11
cube18	262144	18	36	10.01	17.44	10.09	10.08
prodmT5	76	19	711	570.03	1.30	1.49	5.09
cross8	16	8	256	1.77	33.42	8.89	1.80
cross10	20	10	1024	276.28	*	7123.98	120.14
cross12	24	12	4096	*	*	*	11259.65

Table 8.2: Running times for computing extreme rays.

The first point that we wish to make from Table 8.2 is the effect that the insertion order can have on the time taken for the computation. It can make the difference between being able to compute the extreme rays and running out of time or memory. The time to compute the extreme rays of some cones did not vary much for different insertions orders, but for other cones, the computation time varied considerably. This is only because the size of the set of extreme rays at intermediate stages of the extreme ray algorithm can differ greatly. For example, for the cone $ccc7$, using the lexicographic order, the maximum number of extreme rays at an intermediate stage of the algorithm was 68930, but for the min-cut-off order, the maximum number was 128994, almost twice as large. Also, for the cone $ccp7$, using the lexicographic order, the maximum number was 123685, but using the min-cut-off order, the maximum number was 291218. Somewhat surprisingly, the lexicographic ordering seems on average better than the other insertion orders. This was also noted by Fukuda and Prodon in [37], but they do not offer an explanation for this, and we cannot offer a solid explanation either.

[3]The computations are certainly much faster if we use only 32 bit integers without arithmetic overflow checking.

The second point that we wish to make from Table 8.2 is that our implementation of the double description method is effective on problems where m is small in comparison to the dimension of the problem. Indeed, although the number of extreme rays for the last four problems is quite small, our implementation struggles to compute the extreme rays for these examples. The probable reason is that we formulate the cones in the form $\mathcal{C} = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x \geq \mathbf{0}\}$ by introducing slack variables for each constraint, so if m is large in comparison with the dimension then the size of the matrix A can be very large making the rank computations slow. We need to implement and test reformulation of the cones into the form $\mathcal{C}' = \{x \in \mathbb{R}^n : A'x \geq \mathbf{0}, x \geq \mathbf{0}\}$ instead because we think that this form is not as sensitive to large numbers of constraints (see Section 8.2.3). It would be very interesting to see how this approach fares in comparison to the above two formulations.

In Table 8.3, we compare the combinatorial check for adjacency with the algebraic check for adjacency, and we also compare our implementation with *cdd* version 0.61 ([35]) and *lrs* version 4.1 with extended precision arithmetic ([5]). The column titled "algeb." lists the times for the algebraic check, and the column title "comb." lists the times for the combinatorial check. We use the same insertion order, the lexicographic order, to compare the different checks for adjacency and to compare our implementation with *cdd*. The algorithm used by *lrs* is a reverse search algorithm; thus, there is no insertion order required. The first four columns are the same as per Table 8.2 above.

Problem	Size	dim	m	algeb.	comb.	cdd	lrs
55	1940	15	25	0.05	0.05	0.20	0.60
66	97548	25	36	12.61	10.97	2277.73	102.41
66pan	265536	17	36	172.29	145.90	9719.00	171.08
ccc6	210	15	31	0.07	0.04	0.64	5.89
ccc7	38780	21	63	587.04	648.48	7078.25	*
ccp6	368	15	32	0.15	0.06	0.13	0.75
ccp7	116764	21	64	2152.13	2593.26	26923.74	*
cube14	16384	14	28	0.42	0.47	3.81	1.78
cube16	65536	16	32	2.11	2.33	76.64	8.33
cube18	262144	18	36	10.08	15.50	1903.27	43.68
prodmT5	76	19	711	5.09	0.58	0.36	*
cross8	16	8	256	1.80	0.23	0.02	62.76
cross10	20	10	1024	120.14	9.82	0.13	*
cross12	24	12	4096	11259.65	508.76	2.34	*

Table 8.3: Running times for computing extreme rays.

We first discuss the differences in the timings for the combinatorial check versus the algebraic adjacency check. For these example problems, neither check clearly dominates the other although the algebraic check seems more robust. Note that the algebraic check is certainly competitive, which is contrary to what Fukuda and Prodon found in [37]. Also, note that the combinatorial check works better for problems with many constraints since it does not involve rank computations of

large matrices.

Our implementation compares favourably with *cdd* and *lrs* on problems where there are not too many constraints. However, when there are many constraints, *cdd* is much better. We attribute this to the fact that *cdd* computes with cones in the form $\mathcal{C}(A) = \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$, but we compute with cones in the form $\mathcal{C}' = \{x \in \mathbb{R}^n : A'x = \mathbf{0}, x \geq \mathbf{0}\}$ as we discussed above. So, we are computing with large constraint matrices and long vectors, but *cdd* computes with much smaller constraint matrices and shorter vectors. Again, as we mentioned above, we need to investigate computing with cones in the form $\mathcal{C}'' = \{x \in \mathbb{R}^n : A''x \geq \mathbf{0}, x \geq \mathbf{0}\}$ as suggested in Section 8.2.3. We listed the times for *lrs* only to highlight that the double description method is faster than the reverse search method for some degenerate cones. Our implementation of the algebraic check is faster than *lrs* for these example problems, but we certainly would not expect this to be true in general.

Chapter 9

Computing circuits of matrices

In this chapter, we present an algorithm for computing circuits of a matrix. This algorithm is based upon the algorithm of the previous section for computing extreme rays of cones. The algorithm presented here is fundamentally the same algorithm as presented by Hemmecke in [48] and later by others in [93, 39, 91]. There is no new theory presented in this chapter. We only wish to describe how we can modify the algorithm for computing extreme rays of cones to construct an algorithm for computing circuits of matrices, and in doing so, we can reuse our optimisations to the extreme ray algorithm that we presented in the previous chapter to the algorithm for computing circuits of matrices.

Definition 9.0.1. *Given a matrix $A \in \mathbb{R}^{m\times n}$, a non-zero vector $v \in \mathbb{R}^n$ is a circuit of A if $Av = \mathbf{0}$ and there does not exist another vector $v' \in \mathbb{R}^n$ such that $Av' = \mathbf{0}$ and $\text{supp}_I(v') \subsetneq \text{supp}_I(v)$.*

Recall that for any vector $v \in \mathbb{R}^n$, $\text{supp}_I(v) = \{i \in \{1, ..., n\} : v_i \neq 0\}$. So, a circuit of a matrix A is a support-minimal vector in the linear space $\{x \in \mathbb{R}^n : Ax = \mathbf{0}\}$. As we shall see, circuits of matrices can be considered as a natural extension of the concept of extreme rays of cones.

Circuits of matrices are used in computational biology for metabolic pathway analysis (see [38]). Very roughly speaking, for this application, each column of the matrix A represents a chemical reaction and each row of the matrix represents a metabolite (a substance involved in the chemical reactions). For a given reaction j and a given metabolite i, the matrix entry A_{ij} of the matrix is positive if the metabolite i is produced in reaction j, negative if it is consumed, and zero if the metabolite plays no role in reaction j. The proportions of the entries in the column A_{*j} for a particular reaction j give the exact proportions of the metabolites involved in reaction j. A solution $v \in \mathbb{R}^n$ to the equation $Av = \mathbf{0}$ represents a steady state of the system where the net metabolite production and consumption is zero. We are interested in solutions of $Av = \mathbf{0}$ involving a minimal number of reactions (which are exactly the circuits of A). These solutions are called elementary modes. Some reactions are reversible meaning that they are feasible in either direction. Hence, an elementary mode v can have negative and positive entries for a reversible reaction representing different directions of the reaction. Also, some reactions are irreversible

and are feasible in only one direction. In this case, an elementary mode is restricted in sign (non-negative by convention). Hence, to analyse the system, we are interested in all solutions of $Av = \mathbf{0}$ involving a minimal number of reactions and $v_j \geq 0$ for all irreversible reactions j. So, we are actually not interested in all circuits v of the matrix A, but only those circuits for which $v_j \geq 0$ for all irreversible reactions j. We address this point later.

Another application of circuits of matrices is in studying gene interactions. We refer the reader to [11] for a description of this application.

The following lemma is a fundamental property of circuits. It says that a circuit is uniquely determined by its support (unique up to multiplication by a constant), and therefore, the set of circuits of a matrix A is finite since there are only finitely many possible supports a vector can have. Furthermore the Lemma says that circuits are unique up to multiplication by a constant, and we shall treat to circuits that are scalar multiples of each other as the same circuit.

Lemma 9.0.2. *Let $v, v' \in \mathbb{R}^n$ be circuits of the matrix $A \in \mathbb{R}^{m\times n}$. If $\operatorname{supp}_I(v') = \operatorname{supp}_I(v)$, then $v' = \lambda v$ for some $\lambda \in \mathbb{R}^n$.*

Proof. Assume that $\operatorname{supp}_I(v') = \operatorname{supp}_I(v)$ and $v' \neq \lambda v$ for some $\lambda \in \mathbb{R}^n$. Let $\lambda = \frac{v_i}{v'_i}$ for any $i \in \operatorname{supp}_I(v')$, and let $v'' = v - \lambda v'$. By assumption, $v'' \neq \mathbf{0}$. Moreover, by construction, $\operatorname{supp}_I(v'') \subsetneq \operatorname{supp}_I(v)$ since $v''_i = 0$ but $v'_i \neq 0$ contradicting that v is support-minimal. □

9.1 Circuit algorithm

In this section, we present the algorithm for computing circuits of matrices. We translate the concepts that we have discussed so far into a more general form so that we can use the results of Chapter 8 more readily.

Definition 9.1.1. *Given a matrix $A \in \mathbb{R}^{m\times n}$ and a set $\sigma \subseteq \{1, ..., m\}$, a non-zero vector $v \in \mathbb{R}^n$ is a σ-circuit of A if $v \in \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}\}$ and there does not exist another vector $v' \in \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}\}$ such that $\operatorname{supp}_A(v') \subsetneq \operatorname{supp}_A(v)$.*

So, a σ-circuit of A is a supp_A-minimal vector in $\{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}\}$. By definition, a circuit of a matrix $A \in \mathbb{R}^{m\times n}$ is equivalent to a σ-circuit of a matrix $A' \in \mathbb{R}^{(m+n)\times n}$ where $\sigma = \{1, ..., m\}$, $A'_{\bar{\sigma}} = A$, and $A'_{\sigma} = I$. We will see later that σ-circuits of matrices are essentially equivalent to circuits of matrices.

Analogously to circuits of matrices, σ-circuits of matrices are also uniquely determined (unique up to multiplication by a constant) by their support, so we shall treat circuits that are scalar multiples of each other as the same circuit, and therefore, the set of σ-circuits of a matrix A is finite since there are only finitely many possible supports a vector may have,

Lemma 9.1.2. *Let $v \in \mathbb{R}^n$ be a σ-circuit of the matrix $A \in \mathbb{R}^{m\times n}$ and let $v' \in \mathbb{R}^n$ such that $A_{\bar{\sigma}}v' = \mathbf{0}$. If $\operatorname{supp}_A(v') \subseteq \operatorname{supp}_A(v)$, then $v' = \lambda v$ for some $\lambda \in \mathbb{R}^n$.*

Proof. Assume that $\text{supp}_A(v') \subseteq \text{supp}_A(v)$ and $v' \neq \lambda v$ for any $\lambda \in \mathbb{R}^n$. Let $\lambda = \frac{A_i v}{A_i v'}$ for some $i \in \text{supp}_A(v')$, and let $v'' = v - \lambda v'$. By assumption, $v'' \neq \mathbf{0}$. Moreover, by construction, $\text{supp}_A(v'') \subsetneq \text{supp}_A(v)$ since $A_i v'' = 0$ but $A_i v' \neq 0$ contradicting that v is support-minimal. □

Crucially, we show in Lemma 9.1.3 below that a vector $v \in \mathbb{R}^n$ where $A_{\bar{\sigma}} v = \mathbf{0}$ and $A_\sigma v \geq \mathbf{0}$ is a σ-circuit of A if and only if v is an extreme ray of the cone $\mathcal{C}_\sigma(A) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$. Moreover, more generally, we show that a vector $v \in \mathbb{R}^n$ is a σ-circuit of A if and only if v is an extreme ray of the cone $\mathcal{C} = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_{\sigma\setminus\rho} x \geq \mathbf{0}, A_\rho x \leq \mathbf{0}\}$ for any $\rho \subseteq \sigma$ such that $A_{\sigma\setminus\rho} v \geq \mathbf{0}$ and $A_\rho v \leq \mathbf{0}$. Here, the rows of A indexed by σ are inequality constraints, the rows of A indexed by $\bar{\sigma}$ are equality constraints, and the rows of A indexed by $\rho \subseteq \sigma$ are less-than-or-equal-to-zero inequality constraints ($\leq \mathbf{0}$), and the rows of A indexed by $\sigma \setminus \rho$ are greater-than-or-equal-to-zero inequality constraints ($\geq \mathbf{0}$). We may reformulate $\mathcal{C}$ in the form $\mathcal{C}_\sigma(A^\rho)$ where $A^\rho_{\bar{\rho}} = A_{\bar{\rho}}$ and $A^\rho_\rho = -A_\rho$, in other words, A^ρ is the matrix A after multiplying the rows of A indexed by ρ by -1 thus turning $\geq \mathbf{0}$ constraints into $\leq \mathbf{0}$ constraints. Note that the set $\mathcal{O}_\rho := \{x \in \mathbb{R}^n : x_{\bar{\rho}} \geq \mathbf{0}, x_\rho \leq \mathbf{0}\}$ defines an orthant of $\mathbb{R}^n$ where $v \in \mathcal{O}_\rho$. So, $\mathcal{C}_\sigma(A^\rho) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_\sigma x \in \mathcal{O}_\rho\}$. Also, there are possibly many such orthants $\mathcal{O}_\rho$ in which v lies because if $A_i v = 0$, then we can choose either $i \in \rho$ or $i \notin \rho$, and in either case, $A_\sigma v \in \mathcal{O}_\rho$.

Lemma 9.1.3. *A vector $v \in \mathbb{R}^n$ is a σ-circuit of A if and only if v is an extreme ray of the cone $\mathcal{C}_\sigma(A^\rho) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_{\sigma\setminus\rho} x \geq \mathbf{0}, A_\rho x \leq \mathbf{0}\}$ for any $\rho \subseteq \sigma$ such that $v \in \mathcal{C}_\sigma(A^\rho)$.*

Proof. Let v be a σ-circuit of A. Let $\rho \subseteq \sigma$ such that $v \in \mathcal{C}_\sigma(A^\rho)$. Then, by definition, v is a support-minimal vector in $\mathcal{C}_\sigma(A^\rho)$, and thus from Lemma 2.2.13, it is an extreme ray of $\mathcal{C}_\sigma(A^\rho)$.

Conversely, let v be an extreme ray of $\mathcal{C}_\sigma(A^\rho)$. We assume without loss of generality that $\rho = \emptyset$ (i.e. $\mathcal{C}_\sigma(A^\rho) = \mathcal{C}_\sigma(A)$). Let $v' \in \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}\}$ such that $\text{supp}_A(v') \subsetneq \text{supp}_A(v)$. We must have $A_i v' < 0$ for some $i \in \sigma$, otherwise $v' \in \mathcal{C}_\sigma(A)$, which contradicts that v is an extreme ray of $\mathcal{C}_\sigma(A)$ and thus support minimal in $\mathcal{C}_\sigma(A)$ by Lemma 2.2.13. Let $\lambda = \min\{-\frac{A_i v}{A_i v'} : A_i v' < 0, i \in \sigma\}$, and let $v'' = v + \lambda v'$. By construction, $v'' \neq \mathbf{0}$, $v'' \in \mathcal{C}_\sigma(A)$, and $\text{supp}_A(v'') \subsetneq \text{supp}_A(v)$ since $v''_i = 0$ but $v'_i \neq 0$ where $i = \text{argmin}\{-\frac{A_i v}{A_i v'} : A_i v' < 0, i \in \sigma\}$. The statement $\text{supp}_A(v'') \subsetneq \text{supp}_A(v)$ contradicts that v is an extreme ray of $\mathcal{C}_\sigma(A)$. Therefore, no such vector v' exists, so v is supp_A-minimal in $\{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}\}$, so v is a circuit of A. □

We discussed earlier that a circuit of a matrix $A \in \mathbb{R}^{m\times n}$ is equivalent to a σ-circuit of a matrix $A' \in \mathbb{R}^{(m+n)\times n}$ where $\sigma = \{1, ..., m\}$, $A'_{\bar{\sigma}} = A$, and $A'_\sigma = I$. Lemma 9.1.3 above says that the circuits of A are the extreme rays of the cones $\mathcal{C}_\sigma(A'^\rho) = \{x \in \mathbb{R}^n : Ax = \mathbf{0}, x_{\sigma\setminus\rho} \geq \mathbf{0}, x_\rho \leq \mathbf{0}\}$ for every $\rho \subseteq \sigma$. Conversely, from Lemma 9.1.3, the σ-circuits of matrix A are the extreme rays of the cones $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \sigma$, and as we saw in Section 8.2.3, every cone $\mathcal{C}_\sigma(A)$ can be reformulated as $\mathcal{C} = \{x \in \mathbb{R}^n : A'x = \mathbf{0}, x \geq \mathbf{0}\}$ for some matrix A', so Lemma 9.1.3 says that the circuits of A' are the σ-circuits of A. So, the set of circuits of a matrix A is essentially equivalent to the set of σ-circuits of another matrix A'.

It follows immediately from Lemma 9.1.3 above that the union of all the sets of extreme rays of each cone $\mathcal{C}_\sigma(A^\rho) = \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_{\sigma\setminus\rho}x \geq \mathbf{0}, A_\rho x \leq \mathbf{0}\}$ for every $\rho \subseteq \{1, ..., n\}$ gives the set of σ-circuits of A. Actually, if R is the set all extreme rays of all cones $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \{1, ..., n\}$ and C is the set of σ-circuits of A, then $R = C \cup$ -C; in other words, for every circuit $v \in C$, there are two distinct extreme rays $v \in R$ and $-v \in R$. The reason for this is that extreme rays are unique up to multiplication by a positive constant and circuits are unique up to multiplication by a constant (positive or negative). Indeed, if v is a σ-circuit of A, then $-v$ is a σ-circuit of A, and we treat them as the same circuit, but they are different extreme rays.

Then, to compute the circuits of A, we could compute the extreme rays of each cone $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \sigma$ individually using any algorithm for computing extreme rays, and then, we take the union of the sets of extreme rays for all the different cones. The obvious problem with this approach is that there are $2^{|\sigma|}$ many different cones and that we would compute the same circuit v many different times because v can lie in many different cones and from Lemma 9.1.3 a circuit is an extreme ray of every cone $\mathcal{C}_\sigma(A^\rho)$ where $v \in \mathcal{C}_\sigma(A^\rho)$. Moreover, during any iteration of a extreme ray computation, we discard extreme rays that are not feasible for the next iteration, but these extreme rays are actually circuits, so we are wasting computation by discarding them. The solution is to compute the extreme rays for all the orthants simultaneously. Importantly, unlike for the extreme ray case, this algorithm is polynomial in input and output size.

The fundamental idea behind the Circuit Algorithm 14 is that the combined set of extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \sigma$ consists of all the extreme rays of $\mathcal{C}^i_\sigma(A^\rho)$ for every $\rho \subseteq \sigma$ and the set of all new extreme rays $r' = (-A_i r^2)r^1 + (A_i r^1)r^2$ where r^1 and r^2 are adjacent extreme rays of $\mathcal{C}^i_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ such that $A_i r^1 > 0$ and $A_i r^2 < 0$. This follows because any extreme ray of some cone $\mathcal{C}^i_\sigma(A^\rho)$ is feasible for at least one cone $\mathcal{C}_\sigma(A^{\rho'})$ for some $\rho' \subseteq \sigma$ and thus an extreme ray of $\mathcal{C}^\sigma_A(\rho')$, so we never throw away any vectors. Extending this idea, the combined set of extreme rays of $\mathcal{C}^{\tau\text{-}i}_\sigma(A^\rho)$ for $\rho \subseteq \sigma$ and some $i \in \tau$ can be computed from the extreme rays of all cones $\mathcal{C}^\tau_\sigma(A^\rho)$ for $\rho \subseteq \sigma$.

The algorithm for computing all the extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \sigma$, Algorithm 14, thus proceeds in much the same way as the extreme ray algorithm. Initially, we choose some relaxation of $\mathcal{C}_\sigma(A)$ given by relaxing some of the inequality constraints such that the relaxation is still a pointed cone and we can easily find the extreme rays of the relaxation $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \sigma$. Then, we iteratively apply the above fundamental idea: we compute the extreme rays of all cones $\mathcal{C}^{\tau\text{-}i}_\sigma(A^\rho)$ for $\rho \subseteq \sigma$ from the extreme rays of all cones $\mathcal{C}^\tau_\sigma(A^\rho)$ for $\rho \subseteq \sigma$, and we set $\tau = \tau \setminus i$, and repeat until $\tau = \emptyset$.

Algorithm 14 is incomplete. Firstly, it does not specify how to find an initial $\tau \subseteq \sigma$ such that $\mathcal{C}^\tau_\sigma(A)$ is pointed and such that we can compute the extreme rays of $\mathcal{C}^\tau_\sigma(A^\rho)$ for all $\rho \subseteq \sigma$. This is done in essentially the same way as in Algorithm 11 for computing extreme rays. Secondly, it does not specify how to select $i \in \tau$. Thirdly, it does not specify how to check that r^1 and r^2 are adjacent extreme rays of $\mathcal{C}^\tau_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$. We can use essentially the same approaches as for computing extreme rays for all these issues; we address these issues next.

Algorithm 14 Circuit Algorithm

Input: a pointed cone $\mathcal{C}_\sigma(A)$

Output: a set R containing all the extreme rays for every $\mathcal{C}_\sigma(A^\rho)$ for all $\rho \in \sigma$.

```
Find a set τ ⊆ σ such that C^τ_σ(A) is pointed.
Compute the minimal set R of extreme rays of C^τ_σ(A^ρ) for all ρ ⊆ σ.
while τ ≠ ∅ do
  Select i ∈ τ.
  R^+ := {r ∈ R : A_i r > 0}.
  R^- := {r ∈ R : A_i r < 0}.
  for all r^1 ∈ R^+ and r^2 ∈ R^- do
    if r^1 and r^2 are adjacent extreme rays of C^τ_σ(A^ρ) for some ρ ⊆ σ then
      R := R ∪ {(-A_i r^2) r^1 + (A_i r^1) r^2}.
    end if
  end for
  τ := τ-i.
end while
return R
```

Next, we show how to find a set $\tau \subseteq \sigma$ such that $\mathcal{C}^\tau_\sigma(A)$ is pointed and such that we can find all the extreme rays of $\mathcal{C}_\tau(\sigma^A)\rho$ for every $\rho \subseteq \sigma$ easily. This is done in essentially the same way as for computing extreme rays.

Firstly, consider the cone $\mathcal{C}(A) := \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$ where $A \in \mathbb{R}^{n\times n}$ and $\text{rank}(A) = n$. We can easily find all extreme rays of $\mathcal{C}(A^\rho)$ for all $\rho \subseteq \{1, ..., m\}$ for cones of this type. Recall from Section 8.1 that we can easily find the extreme rays of $\mathcal{C}(A)$ as follows. Let R be the inverse matrix of A; i.e. $AR = I$. Let $r^i = R_{*i}$, the ith column of the matrix R. Thus, we have $Ar^i = \mathbf{e}^i$, so $r^i \in \mathcal{C}(A)$, and r^i is an extreme ray of $\mathcal{C}(A)$. The columns of R are thus the extreme rays of $\mathcal{C}(A)$. We now show that the set of extreme rays of $\mathcal{C}(A^\rho) = \{x \in \mathbb{R}^n : A_{\bar{\rho}}x \geq \mathbf{0}, A_\rho x \leq \mathbf{0}\}$ for every $\rho \subseteq \{1, ..., m\}$ consists of exactly the columns of R and the columns of -R. First, consider the cone $\mathcal{C}$(-A) := $\{x \in \mathbb{R}^n : Ax \leq \mathbf{0}\}$, which is the case where $\rho = \{1, ..., m\}$. Note that the inverse matrix of -A is -R. Then, analogously to $\mathcal{C}(A)$, the columns of -R are the extreme rays of $\mathcal{C}$(-A). Extending this, it follows that the extreme rays of $\mathcal{C}(A^\rho)$ for some $\rho \subseteq \{1, ..., m\}$ are the vectors R_{*i} where $i \in \bar{\rho}$ and the vectors $-R_{*i}$ where $i \in \rho$. Therefore, the set of extreme rays of $\mathcal{C}(A^\rho)$ for every $\rho \subseteq \{1, ..., m\}$ consists of exactly the columns of R and the columns of -R.

Next, consider the cone $\mathcal{C}_\sigma(A) := \{x \in \mathbb{R}^n : A_{\bar{\sigma}}x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$ for some $\sigma \subseteq \{1, ..., m\}$ where again $A \in \mathbb{R}^{n\times n}$ and $\text{rank}(A) = n$. We can also easily find all extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for cones of this type. Recall from Section 8.1 that we can easily find the extreme rays of $\mathcal{C}_\sigma(A)$ as follows. Again let R be the inverse matrix of A, and let $r^i = R_{*i}$. Then, the extreme rays of $\mathcal{C}_\sigma(A)$ are the vectors r^i where $i \in \sigma$. Similarly to above, it follows that the extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ are the vectors R_{*i} where $i \in \bar{\rho} = \sigma \setminus \rho$ and the vectors -R_{*i} where $i \in \rho$. Therefore, the set of extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \{1, ..., m\}$ consists of exactly the columns of $R_{*\sigma}$ and the columns of -$R_{*\sigma}$.

Lastly, recall from Section 8.1 that we can find a set $\tau \subseteq \sigma$ such that $\mathcal{C}^\tau_\sigma(A)$ is

pointed and $A_{\bar{\tau}}$ is an $|\bar{\tau}|$ by $|\bar{\tau}|$ matrix. Then, we can find the extremes of $\mathcal{C}_\sigma^\tau(A^\rho)$ for all $\rho \subseteq \sigma$ by computing the inverse matrix of $A_{\bar{\tau}}$ as we discussed above.

We now examine approaches for selecting $i \in \tau$ in Algorithm 14. Although the order in which we select $i \in \tau$ was crucial for Algorithm 11, it is not so important for Algorithm 14 because we never throw away any vectors in Algorithm 14, and thus, at intermediate phases of Algorithm 14, the number of vectors in R never exceeds the number of vectors at completion of the algorithm. However, the Algorithm 14 performs better when we try to keep the number of vectors at intermediate phases low leaving as much work as possible for the last few iterations. Most approaches for selecting $i \in \tau$ do not make sense for computing circuits since, for each cone $\mathcal{C}_\sigma^\tau(A^\rho)$ for different $\rho \subseteq \sigma$, they would potentially select different $i \in \tau$. For example, the lexicographic approach is completely different for $\mathcal{C}_\sigma^\tau(A) = \mathcal{C}_\sigma^\tau(A^\rho)$ for $\rho = \emptyset$ and $\mathcal{C}_\sigma^\tau(\text{-}A) = \mathcal{C}_\sigma^\tau(A^\rho)$ for $\rho = \sigma$. So, since we are computing extreme rays of $\mathcal{C}_\sigma^\tau(A^\rho)$ for every $\rho \subseteq \sigma$ simultaneously, there is no clear choice. The only approach for selecting $i \in \tau$ that we used for computing extreme rays that makes sense for computing circuits is the max-intersection approach; in practice, we found that this approach works better than just choosing $i \in \tau$ in increasing order.

We now examine how to check whether two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ in Algorithm 14. We show that r^1 and r^2 are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ if and only if r^1 and r^2 are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for any $\rho \subseteq \sigma$ such that $r^1, r^2 \in \mathcal{C}_\sigma(A^\rho)$. Thus, we only need to perform one adjacency check per pair of extreme rays r^1 and r^2 in R; that is, we only need to check whether r^1 and r^2 are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for only one and any one $\rho \subseteq \sigma$ such that $r^1, r^2 \in \mathcal{C}_\sigma(A^\rho)$. We do not need to check for adjacency for every possible $\rho \subseteq \sigma$ where $r^1, r^2 \in \mathcal{C}_\sigma(A^\rho)$. Essentially, we show that adjacency is independent of ρ.

Before checking adjacency, we must first check that r^1 and r^2 are extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$. Fortunately, this is straight-forward to do. Firstly, r^1 and r^2 are both rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ if and only if $(A_i r^1)(A_i r^2) \geq 0$ for all $i \in \{1, ..., m\}$ meaning that either $A_i r^1 \geq 0$ and $A_i r^2 \geq 0$ or $A_i r^1 \leq 0$ and $A_i r^2 \leq 0$ for all $i \in \sigma$. Secondly, by construction, all vectors in R are extreme rays for at least one cone $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$, so r^1 and r^2 are extreme rays of at least one cone, and also, as shown in Lemma 9.1.4 below, if a ray r is an extreme ray of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$, then r is an extreme ray of $\mathcal{C}_\sigma(A^{\rho'})$ for any $\rho' \subseteq \sigma$ where $r \in \mathcal{C}_\sigma(A^{\rho'})$.

Lemma 9.1.4. *If a ray r is an extreme ray of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$, then r is an extreme ray of $\mathcal{C}_\sigma(A^{\rho'})$ for every $\rho' \subseteq \sigma$ where $r \in \mathcal{C}_\sigma(A^{\rho'})$.*

Proof. From Corollary 2.2.12, a ray r is an extreme ray of $\mathcal{C}_\sigma(A^\rho)$ if and only if $r \in \mathcal{C}_\sigma(A^\rho)$ and $n - \text{rank}(A^\rho_{\bar{\tau}}) = 1$ where $\tau = \text{supp}_{A^\rho}(r)$. Now, $\tau = \text{supp}_{A^\rho}(r) = \text{supp}_A(r)$ and $\text{rank}(A^\rho_{\bar{\tau}}) = \text{rank}(A_{\bar{\tau}})$ since multiplying rows of A by -1 does not affect the support of a vector nor the rank of a matrix. Thus, a ray r is an extreme ray of $\mathcal{C}_\sigma(A^\rho)$ if and only if $r \in \mathcal{C}_\sigma(A^\rho)$ and $n - \text{rank}(A_{\bar{\tau}}) = 1$ where $\tau = \text{supp}_A(r)$. Since the condition $n - \text{rank}(A_{\bar{\tau}}) = 1$ is independent of ρ, the result follows. □

Lastly, as shown in Lemma 9.1.5 below, if two rays r^1 and r^2 are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$, then they are also adjacent extreme rays of $\mathcal{C}_\sigma(A^{\rho'})$

for every $\rho' \subseteq \sigma$ where $r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$. This is important because it means that we only need to check whether r^1 and r^2 are adjacent extreme rays in only one and any one cone $\mathcal{C}_\sigma(A^\rho)$ for some ρ such that $r^1, r^2 \in \mathcal{C}_\sigma(A^\rho)$.

Lemma 9.1.5. *If the ray r^1 and r^2 are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$, then r^1 and r^2 are adjacent extreme rays of $\mathcal{C}_\sigma(A^{\rho'})$ for every $\rho' \subseteq \sigma$ where $r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$.*

Proof. From Lemma 9.1.4 above, if r^1 and r^2 are extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$, then r^1 and r^2 are extreme rays of $\mathcal{C}_\sigma(A^{\rho'})$ for every $\rho' \subseteq \sigma$ where $r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$. Now, $\tau = \text{supp}_{A^\rho}(r^1 + r^2) = \text{supp}_A(r^1 + r^2)$ and $\text{rank}(A^\rho_{\bar{\tau}}) = \text{rank}(A_{\bar{\tau}})$. So, two distinct extreme rays r^1 and r^2 of the pointed cone $\mathcal{C}^\rho_\sigma(A)$ are adjacent if and only if $\text{rank}(A_{\bar{\tau}}) = n - 2$ by Corollary 8.1.3. Since the condition $\text{rank}(A_{\bar{\tau}}) = n - 2$ is independent of ρ, the result follows. □

In summary, two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ if and only if $(A_i r^1)(A_i r^2) \geq 0$ for all $i \in \{1, ..., m\}$ and r^1 and r^2 are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for any $\rho \subseteq \sigma$ such that $r^1, r^2 \in \mathcal{C}_\sigma(A^\rho)$.

Next, we consider the algebraic check for adjacency, which is essentially the same check as per computing extreme rays because the effect on the constraint matrix of ρ is that of multiplying the rows of A by -1, which has no effect on rank computations. The following corollary was proven in the proof of Lemma 9.1.5 above.

Corollary 9.1.6. *Two distinct extreme rays r^1 and r^2 of the pointed cone $\mathcal{C}_\sigma(A^\rho)$ are adjacent if and only if* $\text{rank}(A_{\bar{\tau}}) = n - 2$ *where $\tau = \text{supp}_A(r^1 + r^2)$.*

So, two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ if and only if $(A_i r^1)(A_i r^2) \geq 0$ for all $i \in \{1, ..., m\}$ and $\text{rank}(A_{\bar{\tau}}) = n - 2$ where $\tau = \text{supp}_A(r^1 + r^2)$. The algebraic check for adjacency is thus easy to adapt and all the improvements we suggested in Section 8.2.3 may be used here.

We now consider the combinatorial check for adjacency. Recall from Lemma 8.1.4, that two distinct r^1 and r^2 of the pointed cone $\mathcal{C}_\sigma(A^\rho)$ are adjacent if and only if there does not exist another extreme ray r of $\mathcal{C}^\rho_\sigma(A)$ distinct from r^1 and r^2 where $\text{supp}_{A^\rho}(r) \subseteq \text{supp}_{A^\rho}(r^1 + r^2)$. Also, from before $\text{supp}_{A^\rho}(r) = \text{supp}_A(r)$ and $\text{supp}_{A^\rho}(r^1 + r^2) = \text{supp}_A(r^1 + r^2)$, so we arrive at Corollary 9.1.7 below.

Corollary 9.1.7. *Two distinct extreme rays r^1 and r^2 of the pointed cone $\mathcal{C}_\sigma(A^\rho)$ are adjacent if and only if there does not exist another extreme ray r of $\mathcal{C}_\sigma(A^\rho)$ distinct from r^1 and r^2 where* $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$.

Recall from above that two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ if and only if r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^{\rho'})$ for any $\rho' \subseteq \sigma$. So, two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ if and only if there does not exist another distinct extreme ray $r \in R$ such that $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$ and $r, r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$ for some $\rho' \subseteq \sigma$. Unfortunately, it is insufficient just to check whether $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$ for some $r \in R$; it may occur that r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$,

but $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$ for some ray $r \in R$ and there does not exist $\rho' \subseteq \sigma$ such that $r, r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$.

We can check whether there exists $\rho' \subseteq \sigma$ such that $r, r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$ as follows. Recall from above that $r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$ for some $\rho' \subseteq \sigma$ if and only if $(A_i r^1)(A_i r^2) \geq 0$ for all $i \in \{1, ..., m\}$. Extending this, $r, r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$ for some $\rho' \subseteq \sigma$ if and only if $(A_i r^1)(A_i r^2) \geq 0$, $(A_i r)(A_i r^1) \geq 0$, and $(A_i r)(A_i r^2) \geq 0$ for all $i \in \{1, ..., m\}$. Or more succinctly, $r, r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$ for some $\rho' \subseteq \sigma$ if and only if $(A_i r^1)(A_i r^2) \geq 0$ and $(A_i r)(A_i(r^1 + r^2)) \geq 0$ for all $i \in \{1, ..., m\}$. Thus, we can check whether two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ by first checking whether $(A_i r^1)(A_i r^2) \geq 0$ for all $i \in \{1, ..., m\}$ and then searching through R for another extreme ray r such that $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$ and $(A_i r)(A_i(r^1 + r^2)) \geq 0$ for all $i \in \{1, ..., m\}$. We can use a tree structure as in Section 8.2.2 to search for a ray $r \in R$ such that $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$. However, we found this approach to be inefficient, and fortunately, there is a much better approach.

Before presenting the better approach we need some new notation. For a ray r and a matrix A, we define the **positive support** of r as

$$\text{supp}_A^+(r) := \{i \in \{1, ..., m\} : A_i r > 0\}$$

and the **negative support** of r as

$$\text{supp}_A^-(r) := \{i \in \{1, ..., m\} : A_i r < 0\}.$$

So, $\text{supp}_A(r) = \text{supp}_A^+(r) \cup \text{supp}_A^-(r)$. Then, $r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$ for some $\rho' \subseteq \sigma$ if and only if $\text{supp}_A^+(r^1) \cap \text{supp}_A^-(r^2) = \emptyset$ and $\text{supp}_A^-(r^1) \cap \text{supp}_A^+(r^2) = \emptyset$. Moreover, $r, r^1, r^2 \in \mathcal{C}_\sigma(A^{\rho'})$ for some $\rho' \subseteq \sigma$ if and only if $\text{supp}_A^+(r^1) \cap \text{supp}_A^-(r^2) = \emptyset$ and $\text{supp}_A^-(r^1) \cap \text{supp}_A^+(r^2) = \emptyset$, and additionally, $\text{supp}_A^+(r) \cap \text{supp}_A^-(r^1 + r^2) = \emptyset$ and $\text{supp}_A^-(r) \cap \text{supp}_A^+(r^1 + r^2) = \emptyset$. Finally, observe that $\text{supp}_A(r) \subseteq \text{supp}_A(r^1 + r^2)$ as well as $\text{supp}_A^+(r) \cap \text{supp}_A^-(r^1 + r^2) = \emptyset$ and $\text{supp}_A^-(r) \cap \text{supp}_A^+(r^1 + r^2) = \emptyset$ if and only if $\text{supp}_A^+(r) \subseteq \text{supp}_A^+(r^1 + r^2) = \emptyset$ and $\text{supp}_A^-(r) \subseteq \text{supp}_A^-(r^1 + r^2) = \emptyset$. Therefore, we can check whether two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ by first checking whether $\text{supp}_A^+(r^1) \cap \text{supp}_A^-(r^2) = \emptyset$ and $\text{supp}_A^-(r^1) \cap \text{supp}_A^+(r^2) = \emptyset$ and then searching through R for another extreme ray r such that $\text{supp}_A^+(r) \subseteq \text{supp}_A^+(r^1 + r^2)$ and $\text{supp}_A^-(r) \subseteq \text{supp}_A^-(r^1 + r^2)$. The problem now is then how do we efficiently search for an extreme ray r in R such that $\text{supp}_A^+(r) \subseteq \text{supp}_A^+(r^1 + r^2)$ and $\text{supp}_A^-(r) \subseteq \text{supp}_A^-(r^1 + r^2)$. We can do this efficiently using a tree structure as in Section 8.2.2 using an extended support set.

Given a ray r and a matrix A, we define the **extended support** as

$$\text{xsupp}_A(r) := \{i : A_i r > 0, i \in \{1, ..., m\}\} \cup \{i + m : A_i r < 0, i \in \{1, ..., m\}\}.$$

Note that $\text{xsupp}_A(r) \subseteq \{1, ..., 2m\}$. Thus, $\text{supp}_A^+(r) \subseteq \text{supp}_A^+(r^1 + r^2)$ and $\text{supp}_A^-(r) \subseteq \text{supp}_A^-(r^1 + r^2)$ if and only if $\text{xsupp}_A(r) \subseteq \text{xsupp}_A(r^1 + r^2)$. Therefore, we can check whether two rays r^1 and r^2 in R are adjacent extreme rays of $\mathcal{C}_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ by first checking whether $\text{supp}_A^+(r^1) \cap \text{supp}_A^-(r^2) = \emptyset$ and $\text{supp}_A^-(r^1) \cap \text{supp}_A^+(r^2) = \emptyset$ and then searching through R for another extreme ray r such that $\text{xsupp}_A(r) \subseteq$

$\text{xsupp}_A(r^1 + r^2)$. We can use exactly the same tree structure as in 8.2.2 to perform the search, and since $|\text{xsupp}_A(r)| = |\text{supp}_A(r)|$, the depth of a tree of extended supports is not any larger than a tree of normal supports. This is the approach we use in practice.

There is one last improvement to the circuit algorithm that we present. As we discussed previously, Algorithm 14 computes twice as many vectors as necessary (i.e. it computes the set $C \cup -C$ where C is the set of σ-circuits of A) because, for every circuit r, both r and $-r$ are extreme rays of cones since both r and $-r$ are circuits. We would prefer to avoid computing the duplicates – only computing one representative of r and $-r$ – and thus just compute the set C. This is certainly possible for the following reasons. If r^1 and r^2 are adjacent extreme rays of $\mathcal{C}^i_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ and some $i \in \sigma$, then $-r^1$ and $-r^2$ are adjacent extreme rays of $\mathcal{C}^i_\sigma(A^{\bar{\rho}})$ where $\bar{\rho} = \sigma \setminus \rho$. This follows from Corollary 9.1.7. Furthermore, if $A_i r^1 > 0$ and $A_i r^2 < 0$, then $r = (-A_i r^2)r^1 + (A_i r^1)r^2$ is an extreme ray of $\mathcal{C}_\sigma(A^\rho)$, but also, since $A_i(-r^2) > 0$ and $A_i(-r^1) < 0$, then $-r = (-A_i(-r^1))(-r^2) + (A_i(-r^2)(-r^1)$ is an extreme ray of $\mathcal{C}_\sigma(A^{\bar{\rho}})$. Thus, if $A_i r^1 > 0$ and $A_i r^2 < 0$, we only need to check whether r^1 and r^2 are adjacent extreme rays of $\mathcal{C}^i_\sigma(A^\rho)$ for some $\rho \subseteq \sigma$ – we do not need to consider $-r^1$ and $-r^2$ – and if r^1 and r^2 are adjacent, we only generate $r = (-A_i r^2)r^1 + (A_i r^1)r^2$, and thus, we avoid computing duplicates. So, when computing the σ-circuits of A, we only store one representative of r and $-r$.

Also, we can extend the circuit algorithm for the more general case where, instead of computing supp_A-minimal vectors of $\{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}\}$, we wish to compute supp_A-minimal vectors of $\{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_\tau x \geq \mathbf{0}\}$ for some $\tau \subseteq \sigma$. Note that computing the extreme rays of a cone is a special case of this where $\tau = \sigma$ and computing the σ-circuits of A is also a special case where $\tau = \emptyset$. Here, we wish to compute the combined set of extreme rays of the cones $\mathcal{C}_\sigma(A^\rho)$ for every $\rho \subseteq \sigma \setminus \tau$, so we restrict the values that ρ can take. The algorithm to compute such a set proceeds as per Algorithm 14 except that we may need to throw away some vectors r at intermediate stages of the algorithm if there does not exist a set $\rho \subseteq \sigma \setminus \tau$ such that $r \in \mathcal{C}_\sigma(A^\rho)$, or in other words, $A_\tau x \not\geq \mathbf{0}$.

We could also modify the alternative approach to the double description method described in Section 8.3 to compute circuits. This alternative approach for computing circuits is essentially the same as the approach given in [48, 93, 91] and the above approach is essentially equivalent to the approach in [39]. As in the case for computing extreme rays, our computational experience lead us to the conclusion that the alternative approach was not as effective as the approach we give above.

9.2 Computational results

In this section, we give some computational results of computing circuits of matrices. All results in this section are listed in seconds and were computed on an Intel XEON 3.2 GHz machine with 4Gb of RAM running Linux Redhat.

In Table 9.1, we list the times in seconds for computing the circuits of some matrices. We compare our implementation in 4*ti*2 version 1.3 (with some minor modifications)

using the algebraic adjacency check and the combinatorial adjacency check, and we also compare our implementation with Metatool 5.0 (see for example, [73, 38, 91]).[1] We consider the computations times for four matrices. The first two matrices are the constraint matrices for the cones of magic squares. The third matrix, "Epistasis", corresponds to a problem in computational biology (see [11]).[2] The last matrix, "Glucose", corresponds to a problem from metabolic pathway analysis (see for example [63]).

The second column is the number of circuits of the matrix. The third and fourth column give the dimensions of the matrix. The next two columns are the times for computing the circuits of the matrix with our implementation in $4ti2$ for the algebraic adjacency check and the combinatorial adjacency check respectively. The last column is the time for Metatool 5.0 to compute the circuits. We understand that Metatool 5.0 uses an algorithm based upon the algorithm described in [91], which is similar to the algorithm described in this chapter (without our optimisations) using an algebraic check for adjacency.

Problem	Size	n	m	algeb.	comb.	Metatool
44	459	16	9	0.01	0.01	0.02
55	229563	25	11	46.04	48.13	75.26
Epistasis	772731	37	18	184.91	732.26	653.68
Glucoe	27100	42	25	29.42	6.32	3.50

Table 9.1: Running times for computing circuits of matrices.

In general, the algebraic check is faster than the combinatorial check particularly when computing large numbers of circuits. The time spent performing the algebraic check is not dependent on the number of circuits whereas the time spent performing the combinatorial check does depend on the number of circuits. We believe that it is for this reason that the algebraic check performs significantly better than the combinatorial check when there are many of circuits. For the last example, the combinatorial check is faster than the algebraic check. This can be attributed to the fact that we needed to use arbitrary precision integer arithmetic to solve this problem; thus, the matrix computations are much slower, but the combinatorial check is left unaffected. For the other problems, we used 64 bit integer arithmetic with overflow checking.

In comparison with Metatool 5.0 our implementation compares well. Metatool is faster for the last problem; this may be because, as far as we know, Metatool uses floating point arithmetic instead of arbitrary precision integer arithmetic, so the algebraic adjacency check is much faster. But, this means that Metatool 5.0 cannot guarantee that its output is correct.

[1]The software package Metatool 5.0 is available for download from the following website: `http://pinguin.biologie.uni-jena.de/bioinformatik/networks/`.

[2]We thank N. Beerenwinkel for providing this example.

Chapter 10

Conclusion

In this chapter, we review the results of this thesis and discuss some possible directions of future research.

In Chapter 3 and Chapter 4, we presented the concepts of Markov bases and Gröbner bases. Most of concepts and theory in these chapters are known in some form or another in the literature, but we tried to present the concepts and theory in a coherent and concise way in the context of integer programming. Here, we described Markov bases and Gröbner basis of fibers and of lattices as well as truncated Markov bases and Gröbner bases of lattices. There are no explicit references to Markov bases of one fiber nor to Gröbner bases of one fiber in the literature, but we found these concepts quite useful in the context of integer programming in order to gain a better understanding of Markov bases and Gröbner bases. In the chapter on Gröbner bases, we gave a pseudo-polynomial upper bound on the size of truncated Gröbner bases for equality knapsack problems. Further investigation into the size of Gröbner bases for different classes of integer programs would be interesting.

The next chapter on computing Gröbner bases of lattices and truncated Gröbner bases of lattices contained a description of the well-known completion procedure. Not much is known about the complexity of the completion procedure for computing Gröbner bases of lattices. More research is required to better understand its complexity even for simple cases. In section 4.3, we presented a different approach for truncating Gröbner bases for which we have some promising initial computational results. At the end of the chapter, we presented different approaches for computing Gröbner bases. These algorithms are not as yet implemented, and it would be interesting to compare their performance with the performance of the completion procedure.

In Chapter 6, we compared different algorithms for computing Markov bases. In Section 6.1, we presented a new algorithm, the Project-and-Lift algorithm, for computing Markov basis of lattices, which is the major contribution of this thesis. This method computes a Markov basis incrementally for a hierarchy of relaxations. The Project-and-Lift algorithm outperforms the other main algorithms for computing Markov bases: the Saturation algorithm and the Lift-and-Project algorithm. The reason for this is that the Project-and-Lift algorithm performs computations in sublattices in contrast to the other algorithms that always work in the original lattice

or in an extended lattice. We also showed that we could modify this algorithm to compute a truncated Markov basis without needing to first compute a non-truncated Markov basis of a lattice and then truncate. Instead, the truncated Project-and-Lift algorithm truncates at intermediate stages of the algorithm thus usually avoiding computing the entire non-truncated Markov basis.

We next presented the applications of the Project-and-Lift algorithm. Firstly, we showed how Markov bases can be used in algebraic statistics. Using the Project-and-Lift algorithm, we were able to solve a previously intractable problem in algebraic statistics to compute the minimal Markov basis of a $4 \times 4 \times 4$ contingency table. The Markov basis of a $4 \times 4 \times 4$ contingency table contains a lot of symmetry – many different vectors in the Markov basis are just permutations of each other. This symmetry of the Markov basis is due to the symmetry of contingency tables (e.g. swapping rows and swapping columns gives another contingency table). Ideally, an algorithm for computing Markov bases should be able to take advantage of this symmetry; currently, none of the algorithms described in this thesis for computing Markov bases with the exception of the Graver basis algorithm (see [51]) can handle symmetry. It would be interesting future work to see whether the Project-and-Lift algorithm could be modified to handle symmetry.

In the chapter on applications, we also presented a slightly extended form of the Project-and-Lift algorithm for computing a feasible solution of an integer program; thus, at the same time as computing a Markov basis, we can compute a feasible solution (or perhaps a set of feasible solutions for a set of different integer programs), which is important for Gröbner basis techniques since they require an initial feasible solution. We showed that the feasibility algorithm performs well on equality knapsack problems. A possible subject for future research is to understand why this method works well for the particular equality knapsack problems we examined and to search for other classes of problems for which this method also works well. Another possibility is to look into using this feasibility approach during the truncated completion procedure (Algorithm 3) since, during the truncated completion procedure, we must determine the infeasibility of many different integer feasibility problems in order to determine whether to truncate a vector or not; this feasibility approach is well suited to this situation.

Another application that we presented concerns the solution of integer programs using Gröbner basis methods. Previous Gröbner basis methods for solving an integer program involved first computing an entire truncated Gröbner basis of a lattice (or even a non-truncated Gröbner basis of a lattice). These approaches do not use much information about the structure of the problem, and thus, they can be very inefficient. We demonstrated that it is possible to significantly improve upon the performance of previous Gröbner basis approaches using structure specific to an integer program. We did this by solving a hierarchy of relaxations so as to avoid computing with unnecessary constraints and by adding upper bounds on the objective function in order to strengthen truncation. Gröbner basis approaches have still not yet proven to be competitive with traditional branch-and-bound based approaches to integer programming for industrial applications. However, given the progress shown here, it appears possible that combined with existing methods, Gröbner basis methods will be useful for certain classes of problems. So, it would be interesting to apply

this method to different classes of problems to see how it performs, and it would also be interesting to see whether it is possible to improve the method for particular classes of problems by using problem structure specific to the class of problems. Specifically, because a Gröbner basis approach is an exact local search method, one could apply Gröbner basis approaches to classes of problems for which heuristic local search methods are often used. Also, Gröbner basis approaches could perhaps be used as part of a heuristic local search method. Here, one would not compute the entire Gröbner basis but only some subset of the Gröbner basis – a partial Gröbner basis – and use the vectors in the partial Gröbner basis as part of a heuristic search method such as tabu search. To compute the partial Gröbner basis, one could just terminate the completion procedure prematurely when an upper bound on time or size had been reached. Perhaps using the set of vectors in a partial Gröbner bases to determine the neighbourhood of a feasible solution will result in better performance of heuristic local search methods.

Another possible avenue of future research is to investigate using a Gröbner basis type approach as a heuristic local search method where Gröbner basis methods can be extended to mixed integer programming (see [55]). For mixed integer programming, the number of vectors in a Gröbner basis can actually be infinite. Most industrial applications are mixed integer programs, so for Gröbner basis methods to have a significant impact on integer programming, they must be applied to the mixed case. There is also a linear programming analogue of Gröbner bases for integer programs (see [80]), and interestingly, solving linear programs with these Gröbner bases is analogous to the simplex method. A Gröbner bases for a linear program can be computed using the algorithm for computing circuits of matrices (see [80]).

In Chapter 8, we showed how we can improve the double description method for computing extreme rays of cones and for computing circuits of matrices. The double description method is a well-known and often used algorithm for computing extreme rays of cones. We suggested optimisations to reduce the number of times during the double description method that we need to check whether two extreme rays are adjacent using the expensive algebraic or combinatorial checks. We also discussed optimisations to the algebraic check for adjacency; we can reformulate the cone so that the rank computation is smaller. There is one particular formulation involving equality constraints together with non-negativity constraints of a cone that was shown to work well in practice (see Section 8.4) for cones where the number of constraints of a cone is not much larger than the dimension of the cone. However, if there are many constraints compared to the dimension, then the formulation that we implemented was not very effective. In Section 8.2.3, we suggested an alternative formulation that should not suffer from the same inefficiencies. It has yet to be implemented, but it would be an interesting direction of future research. Lastly, by using a tree structure for storing the supports of extreme rays, we improved the combinatorial check for adjacency. This approach still has room for improvement, and, with further study, we believe it could be significantly improved.

Perhaps the most interesting topic of future research in the subject of computing extreme rays is to find a polynomial time algorithm (in input and output size) for computing extreme rays of cones or show that such an algorithm does not exist. Recall from Chapter 8 that it is impossible that such a polynomial time algorithm

exists for computing the extreme rays of cones in input polynomial time because there are potentially exponentially many extreme rays, so we must take into consideration the output size. Also, in the context of polyhedra, recall that it was shown by Khachiyan et al. in [62] that there is no polynomial time (in input and output size) algorithm to compute just the vertices of a polyhedron but excluding the extreme rays, but here, the output size does not include the extreme rays, so this result does not rule out a polynomial time algorithm in input and output size for computing extreme rays and vertices.

Lastly, in Chapter 9, we presented a known modified version of the algorithm for computing extreme rays of cones in order to compute the circuits of matrices. In this way, all of our optimisations for the extreme ray algorithm also apply for computing circuits of matrices. Circuits of matrices has an application in computational biology for metabolic pathway analysis ([38]) and in studying gene interactions ([11]). We found that our implementation of the double description method for computing circuits of matrices compared favourably to Metatool 5.0 ([73, 38, 91]), which is another implementation of a variant of the double description method designed for metabolic pathway analysis. We do not feel that there is anything more to be done with respect to the computation of circuits of matrices that would not also apply to computing extreme rays of cones, but there may be more applications of circuits of matrices yet to be discovered.

Appendix A

Computational Algebraic Geometry

In this appendix, we give an *extremely brief* synopsis of the relationship between Markov bases and Gröbner bases of integer programs and generating sets and Gröbner bases of lattice ideals in polynomial rings respectively. We only present the two main results. We refer the reader to [28] for a thorough and easy to read introduction to computational algebraic geometry.

We first define the standards concepts in computational algebraic geometry of ideals, monomial orderings, and Gröbner bases.

We define $k[x_1, ..., x_n]$ where k is a field as the *polynomial ring* consisting of all polynomials in $x_1, ..., x_n$ with coefficients in k. In other words, $k[x_1, ..., x_n]$ is the set of all polynomials in the form $\sum_{\alpha \in \mathbb{N}^n} a_\alpha x^\alpha$ where $a_\alpha \in k$ and only a finite number of a_α are non-zero. Note that we write x^α as shorthand for $x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}$, and x^α is called a **monomial**, and $a_\alpha x^\alpha$ is called a **term**.

An **ideal** I is a subset of $k[x_1, ..., x_n]$ such that (i) $\mathbf{0} \in I$, (ii) if $f, g \in I$, then $f + g \in I$, and (iii) if $f \in I$ and $h \in k[x_1, ..., x_n]$, then $hf \in I$. Given a finite set of polynomials $\{f_1, f_2, ..., f_s\} \subset k[x_1, ..., x_n]$, we define

$$\langle f_1, f_2, ..., f_s \rangle := \{\sum_{i=1}^{s} h_i f_i : h_1, ..., h_s \in k[x_1, ..., x_n]\}.$$

The set $\langle f_1, f_2, ..., f_s \rangle$ is an ideal of $k[x_1, ..., x_n]$, and we call $\{f_1, f_2, ..., f_s\}$ a **generating set** of the ideal $\langle f_1, f_2, ..., f_s \rangle$. A fundamental result in algebraic geometry is the *Hilbert Basis Theorem* which says that every ideal in $k[x_1, ..., x_n]$ has a finite generating set. More formally, for every ideal $I \subseteq k[x_1, ..., x_n]$, there exists $\{f_1, ..., f_s\}$ such that $I = \langle f_1, f_2, ..., f_s \rangle$.

A **term order** is any relation $\succ$ on $\mathbb{N}^n$ or equivalently any relation on the set of monomials x^α, $\alpha \in \mathbb{N}^n$ such that (i) $\succ$ is a total order on $\mathbb{N}^n$, (ii) if $\alpha \succ \beta$ and $\gamma \in \mathbb{N}^n$, then $\alpha + \gamma \succ \beta + \gamma$, and (iii) $\succ$ is a well-order on $\mathbb{N}^n$ meaning that every non-emptyset subset of $\mathbb{N}^n$ has a smallest element under $\succ$. Given a polynomial $f = \sum_{\alpha \in \mathbb{N}^n} a_\alpha x^\alpha \in k[x_1, ..., x_n]$ and a term order $\succ$, the **leading term** of f, written $LT(f)$, is the term $a_\alpha x^\alpha$ of f where α is the largest element under $\succ$ such that a_α

is non-zero: i.e. $\alpha = \max_{\succ}\{\alpha \in \mathbb{N}^n : a_\alpha \neq 0\}$. Note that term orders as defined here correspond exactly to term orders as defined in Section 2.10.

A Gröbner basis of an ideal I with respect to $\succ$ is a finite set of polynomials $G = \{g_1, g_2, .., g_t\} \subseteq I$ such that, for every polynomial $f \in I$, there exists a polynomial $g \in G$ such that $LT(g)$ divides $LT(f)$.

We can now introduce concepts and results that relate Gröbner bases and Markov bases of integer programs to Gröbner bases and generating sets of ideals respectively. The two results are known, but the proofs of them are our own.

Given a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$, a **lattice ideal** is $I(\mathcal{L}) := \langle x^{u^+} - x^{u^-} : u \in \mathcal{L}\rangle \subseteq k[x_1, ..., x_n]$. More explicitly, $I(\mathcal{L})$ is the set of all possible polynomials of the form $\sum_{i=1}^{d} f_i(x)(x^{u^{i+}} - x^{u^{i-}})$ where $u^i \in \mathcal{L}$ and $f_i(x) \in k[x_1, ..., x_n]$. A **toric ideal** is a lattice ideal where the lattice is saturated.

We show in Lemma A.0.1 below that, given a set $M \subseteq \mathcal{L}$, M is a Markov basis of $\mathcal{L}$ if and only if the set of polynomials $\{x^{u^+} - x^{u^-} : u \in M\}$ is a generating set of $I(\mathcal{L})$ or equivalently $I(M) = I(\mathcal{L})$ where $I(M) := \langle x^{u^+} - x^{u^-} : u \in M\rangle$.

Lemma A.0.1. *A finite set $M \subseteq \mathcal{L}$ is a Markov basis of $\mathcal{L}$ if and only if $I(M) = I(\mathcal{L})$.*

Proof. Assume that M is a Markov basis of $\mathcal{L}$. We show that, for every $u \in \mathcal{L}$, $x^{u^+} - x^{u^-} = \sum_{i=1}^{d} \delta^i x^{\gamma^i}(x^{v^{i+}} - x^{v^{i-}})$ where $v^i \in M$, $\delta^i \in \{1, -1\} \in k$, and $\gamma^i \in \mathbb{N}^n$, and the result follows. Note that we allow $v^i = v^j$ for $i \neq j$. Let $u \in \mathcal{L}$. Then, $u^+, u^- \in \mathcal{F}_{\mathcal{L}}(\nu)$ where $\nu = u^+$. Since M is a Markov basis, there exists a path from u^+ to u^- in the graph $\mathcal{G}_{\mathcal{L}}(\nu, M)$, or more explicitly, there exists a sequence of points $(p^1, p^2, ..., p^{d+1}) \subseteq \mathcal{F}_{\mathcal{L}}(\nu)$ where $p^1 = u^+$, $p^{d+1} = u^-$, and $p^i - p^{i+1} \in M$ or $p^{i+1} - p^i \in M$ for $i = 1, ..., d$. For every $i = 1, ..., d$, let $v^i = p^i - p^{i+1}$, $\gamma^i = p^i - v^{i+}$, and $\delta^i = 1$ if $p^i - p^{i+1} \in M$, and $v^i = p^{i+1} - p^i$, $\gamma^i = p^i - v^{i-}$, and $\delta^i = -1$ otherwise. So, $v^i \in M$, $\delta^i v^i = p^i - p^{i+1}$, $p^i = (\delta^i v^i)^+ + \gamma^i$, and $p^{i+1} = (\delta^i v^i)^- + \gamma^i$ for every $i = 1, ..., d$. Hence, $x^{p^i} - x^{p^{i+1}} = \delta^i x^{\gamma^i}(x^{v^{i+}} - x^{v^{i-}})$, and therefore, $x^{u^+} - x^{u^-} = x^{p^1} - x^{p^{d+1}} = \sum_{i=1}^{d} \delta^i x^{\gamma^i}(x^{v^{i+}} - x^{v^{i-}})$ as required.

Conversely, assume that $I(M) = I(\mathcal{L})$. Choose $\alpha, \beta \in \mathcal{F}_{\mathcal{L}}(\nu)$ for some $\nu \in \mathbb{Z}^n$ such that α and β are not connected in $\mathcal{G}_{\mathcal{L}}(\nu, M)$. We will derive a contradiction. The binomial $x^\alpha - x^\beta$ is in $I(\mathcal{L}) = I(M)$, so we may write $x^\alpha - x^\beta = \sum_{i=1}^{d} c^i x^{\gamma^i}(x^{v^{i+}} - x^{v^{i-}})$ where $v^i \in M$, $c^i \in k$, and $\gamma^i \in \mathbb{N}^n$. Note that we allow $v^i = v^j$ for $i \neq j$. Now, let $I \subseteq \{1, ..., d\}$ be the set of $i \in \{1, ..., d\}$ such that the point $(\gamma^i + v^{i+})$ is in $\mathcal{F}_{\mathcal{L}}(\nu)$ and $(\gamma^i + v^{i+})$ is connected to α in $\mathcal{G}_{\mathcal{L}}(\nu, M)$. Note that if $(\gamma^i + v^{i+})$ is connected to α in $\mathcal{G}_{\mathcal{L}}(\nu, M)$, then $(\gamma^i + v^{i-})$ is also connected to α in $\mathcal{G}_{\mathcal{L}}(\nu, M)$ since $(\gamma^i + v^{i+}) - (\gamma^i + v^{i-}) = v$. Thus, the set of monomials consisting of $x^{\gamma^i}x^{v^{i+}}$ and $x^{\gamma^i}x^{v^{i-}}$ for all $i \in I$, which includes x^α and not x^β, is disjoint from the set of monomials consisting of $x^{\gamma^i}x^{v^{i+}}$ and $x^{\gamma^i}x^{v^{i-}}$ for all $i \notin I$, which includes x^β and not x^α. Let $f(x) = \sum_{i \in I} c^i x^{\gamma^i}(x^{v^{i+}} - x^{v^{i-}})$ and let $g(x) = -\sum_{i \notin I} c^i x^{\gamma^i}(x^{v^{i+}} - x^{v^{i-}})$. Thus, the polynomials $f(x)$ and $g(x)$ have a disjoint set of monomials, and therefore, $f(x) = x^\alpha$ and $g(x) = x^\beta$ since $x^\alpha - x^\beta = f(x) - g(x)$. However, this is impossible since $f(\mathbf{1}) = 0$ and $g(\mathbf{1}) = 0$ but $\mathbf{1}^\alpha = 1$ and $\mathbf{1}^\beta = 1$. □

Lemma A.0.2. *Given a term order $\succ$, a set $G \subseteq \mathcal{L}_\succ$ is a $\succ$-Gröbner basis of $\mathcal{L}$ if and only if $\{x^{v^+} - x^{v^-} : v \in G\}$ is a $\succ$-Gröbner basis of $I(\mathcal{L})$.*

Proof. Assume that $\{x^{v^+} - x^{v^-} : v \in G\}$ is a $\succ$-Gröbner basis of $I(\mathcal{L})$. Let $u \in \mathcal{L}_\succ$. Then, $x^{u^+} - x^{u^-} \in I(G)$ and the leading term of $x^{u^+} - x^{u^-}$ is x^{u^+} since $u^+ \succ u^-$. There must exist $v \in G$ such that x^{v^+} divides x^{u^+} because G is a $\succ$-Gröbner basis of $I(\mathcal{L})$. This implies that $v^+ \leq u^+$. Thus, $G^+ \leq \mathcal{L}^+_\succ$, and therefore, G is a $\succ$-Gröbner basis of $\mathcal{L}$ by Lemma 4.2.3.

Assume $G \subseteq \mathcal{L}_\succ$ is a $\succ$-Gröbner basis of $\mathcal{L}$. Let $f(x) \in I(\mathcal{L})$ and let x^α be the leading term of $f(x)$. Note that since k is a field, we can always assume that the leading term is a monomial because we can always multiply f by the multiplicative inverse of the leading coefficient. We must show that there exists a vector $v \in G$ such that x^{v^+} divides x^α or equivalently $v^+ \leq \alpha$. Assume that there is no $v \in G$ such that $v^+ \leq \alpha$, which implies that α is the optimal solution of $IP_{\mathcal{L},\succ}(\alpha)$ since G is a Gröbner basis of $\mathcal{L}$. Then, since x^α is the leading monomial of $f(x)$, there are no monomials x^β in $f(x)$ such that $\beta \in \mathcal{F}_\mathcal{L}(\alpha)$. We now derive a contradiction. The polynomial $f(x)$ is in $I(\mathcal{L})$, so we may write $f(x) = \sum_{i=1}^d c^i x^{\gamma^i}(x^{u^{i+}} - x^{u^{i-}})$ where $u^i \in \mathcal{L}$, $c^i \in k$, and $\gamma^i \in \mathbb{N}^n$. Let $I \subseteq \{1, ..., d\}$ be the set of $i \in \{1, ..., d\}$ such that the point $(\gamma^i + u^{i+})$ is in $\mathcal{F}_\mathcal{L}(\alpha)$. Note that if $(\gamma^i + u^{i+}) \in \mathcal{F}_\mathcal{L}(\alpha)$, then $(\gamma^i + u^{i-}) \in \mathcal{F}_\mathcal{L}(\alpha)$. Let $h(x) = \sum_{i \in I} c^i x^{\gamma^i}(x^{u^{i+}} - x^{u^{i-}})$ and let $g(x) = \sum_{i \notin I} c^i x^{\gamma^i}(x^{u^{i+}} - x^{u^{i-}})$. Then, $f(x) = h(x) + g(x)$, and also, $h(x)$ only contains monomials x^β such that $\beta \in \mathcal{F}_\mathcal{L}(\alpha)$, and $g(x)$ only contains monomials x^β such that $\beta \notin \mathcal{F}_\mathcal{L}(\alpha)$. Therefore, $x^\alpha = h(x)$, which is not possible since $h(\mathbf{1}) = 0$ and $\mathbf{1}^\alpha = 1$. □

Appendix B

Notation

$\mathbb{R}$	the set of real numbers.
$\mathbb{R}_+$	the set of non-negative real numbers.
$\mathbb{Q}$	the set of rational numbers.
$\mathbb{Z}$	the set of integers, $\mathbb{Z}$:={...,-2,-1,0,1,2,...}.
$\mathbb{N}$	the set of non-negative integers, $\mathbb{N} := \{0, 1, 2, ...\}$.
$\mathbf{0}$	the vector of all-zeros of appropriate dimension.
$\mathbf{1}$	the vector of all-ones of appropriate dimension.
v^+	the positive part of the vector $v \in \mathbb{Z}^n$, i.e. $v_i^+ = \max\{v_i, 0\}$ for all $i = 1, ..., n$.
v^-	the negative part of $v \in \mathbb{Z}^n$, $v_i^- = \max\{\text{-}v_i, 0\}$ for all $i = 1, ..., n$.
π_σ	the map $\pi_\sigma : \mathbb{Z}^n \to \mathbb{Z}^{\lvert\bar{\sigma}\rvert}$ that maps vectors in $\mathbb{Z}^n$ onto their $\bar{\sigma}$ components.
v_σ	the projection of the vector $v \in \mathbb{Z}^n$ onto its $\sigma \subseteq \{1, ..., n\}$ components.
V_σ	the projection of the set $V \subseteq \mathbb{Z}^n$ onto its $\sigma \subseteq \{1, ..., n\}$ components.
V^+	$:= \{v^+ : v \in V\}$.
V^-	$:= \{v^- : v \in V\}$.
$\mathbf{e}^i$	the ith unit vector of appropriate dimension.
$A^\intercal$	the transpose of the matrix A.
A_i	the ith row of the matrix A.
A_τ	the submatrix of A consisting of the rows of A indexed by the set τ.
A_{*j}	the jth column of the matrix A.
$A_{*\sigma}$	the submatrix of A consisting of the columns of A indexed by the set σ.
A_{ij}	the entry of the matrix A in the ith row and jth column.
$A_{\tau\sigma}$	the submatrix of A consisting of the rows indexed by τ and the columns indexed by σ.
I	the identity matrix of appropriate dimension.
$\lvert\lvert v\rvert\rvert_1$	the l_1-norm of a vector $v \in \mathbb{Z}^n$, $\lvert\lvert v\rvert\rvert_1 = \sum_{i=1}^n \lvert v_i\rvert$.

$\mathcal{S}$	a linear space.
$\mathcal{S}^*$	the dual of the linear space $\mathcal{S}$: $\mathcal{S}^* := \{x \in \mathbb{R}^n : xs = 0 \; \forall s \in \mathcal{S}\}$.
$\mathcal{S}(A)$	the linear space $\mathcal{S}(A) := \{x \in \mathbb{R}^n : Ax = 0\}$.
$\mathcal{C}$	a cone.
$\mathcal{C}^*$	the dual of the cone $\mathcal{C}$: $\mathcal{C}^* := \{x \in \mathbb{R}^n : cx \geq 0 \; \forall c \in \mathcal{C}\}$.
$\mathcal{C}(A)$	$:= \{x \in \mathbb{R}^n : Ax \geq \mathbf{0}\}$.
$\mathcal{C}_\sigma(A)$	$:= \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_\sigma x \geq \mathbf{0}\}$.
$\mathcal{C}_\sigma^\tau(A)$	$:= \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = \mathbf{0}, A_{\sigma \setminus \tau} x \geq \mathbf{0}\}$.
$\mathcal{C}_\sigma(A^\rho)$	$:= \{x \in \mathbb{R}^n : Ax = \mathbf{0}, A_{\sigma \setminus \rho} x \geq \mathbf{0}, A_\rho x \leq \mathbf{0}\}$.
$\mathcal{P}_A(b)$	$:= \{x \in \mathbb{R}^n : Ax \geq b\}$.
$\mathcal{P}_A^\sigma(b)$	$:= \{x \in \mathbb{R}^n : A_{\bar{\sigma}} x = b_{\bar{\sigma}}, A_\sigma x \geq b_\sigma\}$.
$LP_{A,c}(b)$	$:= \min\{cx : x \in \mathcal{P}_A(b)\}$.
$LP_{A,c}^\sigma(b)$	$:= \min\{cx : x \in \mathcal{P}_A^\sigma(b)\}$.
$\mathcal{L}$	a lattice.
$\mathcal{L}^\sigma$	the projection of a lattice $\mathcal{L}$ onto the $\bar{\sigma}$ components: $\mathcal{L}^\sigma := \pi_\sigma(\mathcal{L})$.
$\mathcal{L}_\succ$	$:= \{u \in \mathcal{L} : u^+ \succ u^-\}$.
$\mathcal{L}_\succ$	$:= \{u \in \mathcal{L} : u^+ \succ u^-, u^+ \in \mathcal{B}_\mathcal{L}(\nu)\}$.
$\mathcal{F}_\mathcal{L}(\nu)$	$:= \{x \in \mathbb{Z}^n : x - \nu \in \mathcal{L}, x \geq \mathbf{0}\}$.
$\mathcal{F}_\mathcal{L}^\sigma(\nu)$	$:= \{x \in \mathbb{Z}^n : x - \nu \in \mathcal{L}, x_{\bar{\sigma}} \geq \mathbf{0}\}$.
$IP_{\mathcal{L},c}(\nu)$	$:= \min\{cx : x \in \mathcal{F}_\mathcal{L}(\nu)\}$.
$IP_{\mathcal{L},c}^\sigma(\nu)$	$:= \min\{cx : x \in \mathcal{F}_\mathcal{L}^\sigma(\nu)\}$.
$IP_{\mathcal{L},\succ}(\nu)$	$:= \min_\succ\{x : x \in \mathcal{F}_\mathcal{L}(\nu)\}$.
$IP_{\mathcal{L},\succ}^\sigma(\nu)$	$:= \min_\succ\{x : x \in \mathcal{F}_\mathcal{L}^\sigma(\nu)\}$.
$\mathcal{G}_\mathcal{L}(\nu, S)$	the fiber graph with nodes in $\mathcal{F}_\mathcal{L}(\nu)$ and edges in S.
$\mathcal{G}_\mathcal{L}^\sigma(\nu, S)$	the fiber graph with nodes in $\mathcal{F}_\mathcal{L}^\sigma(\nu)$ and edges in S.
$\mathcal{B}_\mathcal{L}(\nu)$	$:= \{\nu' \in \mathbb{Z}^n : \mathcal{F}_\mathcal{L}(\nu') \neq \emptyset$ and $\mathcal{F}_\mathcal{L}(\nu - \nu') \neq \emptyset\}$.
$\mathcal{B}_\mathcal{L}^\sigma(\nu)$	$:= \{\nu' \in \mathbb{Z}^n : \mathcal{F}_\mathcal{L}^\sigma(\nu') \neq \emptyset$ and $\mathcal{F}_\mathcal{L}^\sigma(\nu - \nu') \neq \emptyset\}$.
$\text{supp}(x)$	$:= \{i \in \{1, ..., n\} : x_i \neq 0\}$.
$\text{supp}_A(x)$	$:= \{i \in \{1, ..., m\} : A_i x \neq 0\}$.
$x \vee y$	component-wise maximum: $(x \wedge y)_i = \max\{x_i, y_i\} \; \forall i = 1, \ldots, n$.
$x \wedge y$	component-wise minimum: $(x \wedge y)_i = \min\{x_i, y_i\} \; \forall i = 1, \ldots, n$.
HNF	Hermite Normal Form.
UHNF	Upper Hermite Normal Form.

Bibliography

[1] 4ti2 team. 4ti2 – a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de.

[2] K. Aardal and A. K. Lenstra. Hard equality constrained integer knapsacks. *Mathematics of operations research*, 29(3):724–738, 2004.

[3] W.W. Adams and P. Loustaunau. *An Introduction to Gröbner Bases*, volume 3. Oxford University Press, 1994.

[4] S. Aoki and A. Takemura. The list of indispensable moves of the unique minimal Markov basis for 3x4xk and 4x4x4 contingency tables with fixed two-dimensional marginals. *METR Technical Report*, pages 03–38, 2003.

[5] D. Avis. lrs: a c implementation of the reverse search vertex enumeration algorithm. Available at http://cgm.cs.mcgill.ca/˜avis/C/lrs.html.

[6] D. Avis and D. Bremner. How Good are Convex Hull Algorithms? In *11th Annual ACM Symposium on Computational Geometry*, pages 20–28, June 1995.

[7] D. Avis and K. Fukuda. A basis enumeration algorithm for linear systems with geometric applications. *Applied Mathematics Letters*, 5:39–42, 1991.

[8] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Computational Geometry*, 8:295–313, 1992.

[9] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 6:21–46, 1996.

[10] T. Becker and V. Weispfenning. *Gröbner Bases: A Computational Approach to Commutative Algebra*. Springer, New York, 1993.

[11] N. Beerenwinkel, L. Pachter, and Bernd Sturmfels. Epistasis and shapes of fitness landscapes. to appear in Statistica Sinica.

[12] A.M. Bigatti, R. LaScala, and L. Robbiano. Computing toric ideals. *Journal of Symbolic Computation*, 27:351–365, 1999.

[13] B. Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner bases. In *Proceedings EUROSAM 79*, volume 72 of *LNCS*, pages 3–21. Springer-Verlag, 1979.

[14] B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, chapter 6, pages 184–232. D. Reidel Publishing Company, 1985.

[15] B. Buchberger. History and basic features of the critical-pair/completion procedure. *Journal of Symbolic Computation*, 2:3–38, 1987.

[16] M. Caboara, M. Kreuzer, and L. Robbiano. Efficiently computing minimal sets of critical pairs. *Journal of Symbolic Computation*, 38(4):1169–1190, 2003.

[17] D.R. Chand and S.S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 17(1):78–86, 1970.

[18] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[19] N.V. Chernikova. An algorithm for finding a general formula for non-negative solutions of system of linear inequalities. *U.S.S.R. Computational Mathematical Physics*, 5:228–233, 1965.

[20] T. Christof and A. Lbel. Porta: Polyhedron representation transformation algorithm. Available at http://www.zib.de/Optimization/Software/Porta/.

[21] K.L. Clarkson and P.W. Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 12–17, New York, NY, USA, 1988. ACM Press.

[22] G.F. Clements. Multiset antichains having minimal downsets. *Journal of Combinatorial Theory*, 48:255–258, 1988.

[23] CoCoATeam. CoCoa: a system for doing Computations in Commutative Algebra, 2005. Available at `http://cocoa.dima.unige.it`.

[24] S. Collart, M. Kalkbrener, and D. Mall. Converting Bases with the Gröbner Walk. *J. Symbolic Computation*, 24:456–469, 1997.

[25] P. Conti and C. Traverso. Buchberger algorithm and integer programming. In *Proceedings AAECC-9 (New Orleans)*, volume 539 of *LNCS*, pages 130–139. Springer Verlag, 1991.

[26] G. Cornuéjols and M. Dawande. A class of hard small 0-1 programs. In R.E. Bixby, E.A. Boyd, and R.Z. Rios-Mercado, editors, *6th International IPCO Conference on Integer Programming and Combinatorial Optimization*, volume 1412 of *LNCS*, pages 284–293. Springer Verlag, 1998.

[27] G. Cornuéjols, R. Urbaniak, R. Weismantel, and L.A. Wolsey. Decomposition of integer programs and of generating sets. In *European Symposium on Algorithms 1997*, volume 1284 of *LNCS*, pages 92–103. Springer-Verlag, 1997.

[28] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer-Verlag, 1992.

[29] G. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[30] P. Diaconis and B. Sturmfels. Algebraic algorithms for sampling from conditional distributions. *Annals of Statistics*, 26:363–397, 1998.

[31] T. Dubé, B. Mishra, and C. K. Yap. Admissible orderings and bounds for Gröbner bases normal form algorithm. Report 88, NYU-Courant Institute Robotics Lab Report, 1986.

[32] H. Edelsbrunner. *Algorithms in Ccomputational Geometry*. Springer-Verlag, 1987.

[33] N. Eriksson. Toric ideals of homogeneous phylogenetic models. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, pages 149–154. ACM Press, 2004.

[34] J.-C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero dimensional gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16:329–344, 1993.

[35] K. Fukuda. cdd – an implementation of the double description method for computing all vertices and extremal rays of a convex polydedron given by a system of linear inequalities. Available at http://www.ifor.math.ethz.ch/~fukuda/cdd_home/index.html.

[36] K. Fukuda, A. N. Jensen, N. Lauritzen, and R.R. Thomas. The generic Gröbner walk. e-print: arXiv:math.AC/0501345, 2005.

[37] K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and Computer Science*, pages 91–111, 1995.

[38] J. Gagneur and S. Klamt. Computation of elementary modes: a unifying framework and the new binary approach. *BMC Bioinformatics*, 5(175), 2004.

[39] J. Gagneur, A. von Kamp, and S. Klamt. Algorithmic approaches for computing elementary modes in large biochemical reaction networks. *IEE Proc. Systems Biology*, 152(4), December 2005.

[40] D. Gamerman and H.F. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman & Hall/CRC, second edition, 2006.

[41] R. Gebauer and H. M. Möller. On an installation of Buchberger's algorithm. *Journal of Symbolic Computation*, 6:275–286, 1988.

[42] A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. "one sugar cube, please" or selection strategies in the buchberger algorithm. In *ISSAC '91: Proceedings of the 1991 international symposium on Symbolic and algebraic computation*, pages 49–54, New York, NY, USA, 1991. ACM Press.

[43] R.E. Gomory. On the relation between integer and noninteger solutions to linear programs. In *Proceedings of the National Academy of Sciences*, volume 53, pages 260–265, 1965.

[44] R.E. Gomory. Some polyhedra related to combinatorial problems. *Linear Algebra and its Applications*, 2:451–558, 1969.

[45] J.E. Graver. On the foundation of linear and integer linear programming I. *Mathematical Programming*, 9:207–226, 1975.

[46] G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 3.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2005. `http://www.singular.uni-kl.de`.

[47] B. Grünbaum. *Convex Polytopes*. John Wiley & Sons, 1967.

[48] R. Hemmecke. On the computation of Hilbert bases and extreme rays of cones. e-print: arXiv:math.CO/0203105, 2002.

[49] R. Hemmecke. On the computation of Hilbert bases of cones. In A. M. Cohen, X.-S. Gao, and N. Takayama, editors, *International Congress on Mathematical Software*. World Scientific, 2002.

[50] R. Hemmecke. On the positive sum property and the computation of graver test sets. *Mathematical Programming*, 96(2):247–269, 2003.

[51] R. Hemmecke. Exploiting symmetries in the computation of Graver bases. e-print: arXiv:math.CO/0410334, 2004.

[52] R. Hemmecke and P.N. Malkin. Computing generating sets of lattice ideals. e-print: arXiv:math.CO/0508359, 2006.

[53] R. Hemmecke, J. Morton, A. Shiu, B. Sturmfels, and O. Wienand. Three counterexamples on semigraphoids. e-print arXiv:math.CO/0610451, 2006.

[54] R. Hemmecke, A. Takemura, and R. Yoshida. Computing holes in semi-groups. e-print arXiv.org:math/0607599, 2006.

[55] M. Henk, M. Köppe, and R. Weismantel. Integral decomposition of polyhedra and some applications in mixed integer programming. *Mathenatical Programming, Series B*, 94(2-3):193–206, 2003.

[56] S. Hosten and J. Shapiro. Primary decomposition of lattice basis ideals. *Journal of Symbolic Computation*, 29:625–639, 2000.

[57] S. Hosten and B. Sturmfels. GRIN: An implementation of Gröbner bases for integer programming. In E. Balas and J.Clausen, editors, *Integer Programming and Combinatorial Optimization*, volume 920 of *LNCS*, pages 267–276. Springer Verlag, 1995.

[58] S. Hosten and R.R. Thomas. Standard pairs and group relaxations in integer programming. *Journal of Pure and Applied Algebra*, 139:133–157, 1999.

[59] S. Hosten and R.R. Thomas. Gomory Integer Programs. *Mathematical Programming Series B*, 96:271–272, 2003.

[60] D.T. Huynh. A Superexponential Lower Bound for Gröbner Bases and Church-Rosser Commutative Thue Systems. *Information and Control*, 86:196–206, 1986.

[61] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[62] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich. Generating all vertices of a polyhedron is hard. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 758–765, New York, NY, USA, 2006. ACM Press.

[63] S. Klamt and J. Stelling. Combinatorial complexity of pathway analysis in metabolic networks. *Molecular Biology Reports*, 29:233–236, 2002.

[64] V. Klee and G.J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.

[65] E.W. Mayr. Some complexity results for polynomial ideals. *Journal of Complexity*, 13(3):303–325, 1997.

[66] P. McMullen. The maximum number of faces of a convex polytope. *Mathematika*, XVII:179–184, 1970.

[67] P. McMullen and G.C. Shephard. *Convex polytopes and the upper bound conjecture*. Cambridge University Press, 1971.

[68] D.S. Moore and G.W. Cobb. Statistics and mathematics: Tension and cooperation. *The American Mathematical Monthly*, 107(7):615–630, 2000.

[69] T.S. Motzkin, H. Raiffa, G.L. Thompson, and R.M. Thrall. The double description method. In H.W. Kuhn A.W. Tucker, editor, *Contributions to theory of games*, volume 2. Princeton University Press, Princeton, RI, 1952.

[70] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice-Hall, 1994.

[71] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

[72] H. Ohsugi and T. Hibi. Toric ideals arising from contingency tables. *Proceedings of the Ramanujan Mathematical Society's Lecture Notes Series*, 2005. to appear.

[73] T. Pfeiffer, I. Sánchez-Valdenebro, J. C. Nuño, F. Montero, and S. Schuster. Metatool: For studying metabolic networks. *Bioinformatics*, 15:251–257, 1999.

[74] B.H. Roune. Solving thousand-digit frobenius problems using grobner bases, 2007.

[75] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 404–413, 1986.

[76] M. Sniedovich. *Dynamic Programming*. Marcel Dekker, Inc., New York, 1991.

[77] A. Storjohann and G. Labahn. Asymptotically Fast Computation of Hermite Normal Forms of Integer Matrices. In Y. N. Lakshman, editor, *Internat. Symp. on Symbolic and Algebraic Computation: ISSAC '96*, pages 259–266, New York, 1996. ACM Press.

[78] B. Sturmfels. Gröbner bases of toric varieties. *Tôhoku Math. Journal*, 43:249–261, 1991.

[79] B. Sturmfels. *Gröbner bases and convex polytopes*, volume 8 of *University Lecture Notes Series*. American Mathematical Society, Providence, Rhode Island, 1996.

[80] B. Sturmfels and R.R. Thomas. Variation of cost functions in integer programming. *Mathematical Programming*, 77:357–387, 1997.

[81] B. Sturmfels, R. Weismantel, and G.M. Ziegler. Gröbner bases of lattices, corner polyhedra, and integer programming. *Contributions to Algebra and Geometry*, 36(2):281–298, 1995.

[82] A. Takemura and R. Yoshida. A generalization of the integer linear infeasibility problem. e-print arXiv.org:math/0603108, 2006.

[83] S.R. Tayur, R.R. Thomas, and N.R.Natraj. An algebraic geometry algorithm for scheduling in the presence of setups and correlated demands. *Mathematical Programming*, 69:369–401, 1995.

[84] M. Terzer and J. Stelling. Accelerating the computation of elementary modes using pattern trees. In *Algorithms in Bioinformatics*, WABI, page 333ff, Zurich, 2006.

[85] R. Thomas and R. Weismantel. Truncated Gröbner bases for integer programming. *Applicable Algebra in Engineering, Communication and Computing*, 8:241–257, 1997.

[86] R.R. Thomas. A geometric Buchberger algorithm for integer programming. *Mathematics of Operations Research*, 20:864–884, 1995.

[87] R.R. Thomas. The structure of group relaxations. In Karen Aardal, George Nemhauser, and Robert Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*. Elsevier, 2005.

[88] R.R. Thomas and R. Weismantel. Test sets and inequalities for integer programs. In *Proceedings of the 5th International IPCO conference*, volume 1084 of *LNCS*, pages 16–30, Vancouver, 1996.

[89] C. Traverso. Hilbert functions and the Buchberger algorithm. *Journal of Symbolic Computation*, 22:355–376, 1997.

[90] L.E. Trotter. Lecture notes on integer programming. Workshop sponsored by the E.P.F.-Lausanne, Switzerland, 2004.

[91] R. Urbanczik and C. Wagner. An improved algorithm for stoichiometric network analysis: theory and applications. *Bioinformatics*, 21:1203–1210, 2005.

[92] R. Urbaniak, R. Weismantel, and G.M. Ziegler. A variant of Buchberger's algorithm for integer programming. *SIAM J. on Discrete Mathematics*, 10:96–108, 1997.

[93] C. Wagneur. Nullspace approach to determine elementary modes of chemical reaction systems. *Journal of Physical Chemistry B*, 108(7):2425–2431, 2004.

[94] L.A. Wolsey. Extensions of the group theoretic approach in integer programming. *Management Science*, 18(1):74–83, 1971.

[95] L.A. Wolsey. *Integer Programming*. John Wiley & Sons, Inc, 1998.

[96] S. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1996.