Parameterised Integer Programming, Integer Cones, and Related Problems

Dissertation

zur Erlangung des Doktorgrades
der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn

vorgelegt von

Gennady Shmonin

Paderborn, den 21. Juni 2007

Gutachter der Dissertation:

Prof. Dr. Friedrich Eisenbrand (Betreuer der Dissertation, Universität Paderborn)

Prof. Dr. András Sebő (Laboratoire G-SCOP, Grenoble, France)

Abstract

Given an $m \times n$-matrix A and a polyhedron Q in $\mathbb{R}^m$, we want to find a vector $b \in Q$ such that the system of linear inequalities $Ax \leqslant b$ has no integral solution. We refer to this problem as a *parameterised integer (linear) programming* problem. This is a generalisation of ordinary integer linear programming, as Q can be chosen to contain only a single vector in $\mathbb{R}^m$. Motivated by the celebrated algorithm of Lenstra (1983) for integer programming in fixed dimension, we restrict ourselves to the case when n is fixed and develop a polynomial-time algorithm for parameterised integer programming in fixed dimension. As an application of this result, we provide an algorithm that computes the *integer programming gap* of a family of integer programs, i.e., the maximum value of the difference

$$\max\{cx : Ax \leqslant b\} - \max\{cx : Ax \leqslant b,\ x \in \mathbb{Z}^n\}$$

over all b for which the integer program is feasible.

Then, we consider integer programs in *standard form*,

$$\min\{cx : Ax = b,\ x \in \mathbb{Z}^n_+\},$$

and establish several bounds on the number of non-zero components in an optimum solution. It turns out that there is always an optimum solution with the number of non-zero entries bounded by a *polynomial* in the number of constraints and the maximum size of an entry in A. This fact follows from the *integer* analogue of Carathéodory's theorem, which is proved in this thesis. Such a bound is especially beneficial when the integer program is derived from some combinatorial optimisation problem and contains exponentially many variables — nonetheless, we still can guarantee the existence of an optimum solution of polynomial size.

One of such applications is the *cutting stock problem*. The columns of the matrix A in the integer programming formulation for this problem are exactly the integral non-negative solutions of the knapsack inequality $ax \leqslant 1$; hence, their number is exponential in the input size. However, we prove that an optimum solution of polynomial size exists, and therefore, the cutting stock problem belongs to NP, which was not known so far. We continue investigating this integer program and derive some results on the strength of its linear programming relaxation, i.e., integer programming gap. Finally, we describe *polynomial-size* integer programming formulations for the cutting stock problem.

Kurzzusammenfassung

Sei eine $m \times n$ Matrix A und ein Polyeder Q im $\mathbb{R}^m$ gegeben, dann suchen wir einen Vektor $b \in Q$, so dass das Ungleichungssystem $Ax \leqslant b$ keine ganzzahlige Lösung besitzt. Wir bezeichnen dieses Problem als *parametrisierte ganzzahlige Programmierung*. Dies ist eine Verallgemeinerung der gewöhnlichen ganzzahligen Programmierung, denn Q kann als ein einzelner Vektor des $\mathbb{R}^m$ gewählt werden. Motiviert durch den vielgepriesenen Algorithmus von Lenstra (1983) für ganzzahlige Programmierung in fester Dimension, beschränken wir uns auf konstantes n und entwickeln für diesen Fall einen Polynomialzeit-Algorithmus für parametrisierte ganzzahlige Programmierung in fester Dimension. Als eine Anwendung dieses Resultats liefern wir einen Algorithmus, welcher den Integrality Gap einer Familie von ganzzahligen Programmen berechnet; das bedeutet die maximale Differenz

$$\max\{cx : Ax \leqslant b\} - \max\{cx : Ax \leqslant b,\, x \in \mathbb{Z}^n\}$$

über alle Vektoren b, für die das ganzzahlige Programm zulässig ist.

Dann betrachten wir ganzzahlige Programme in *Standardform*,

$$\min\{cx : Ax = b,\, x \in \mathbb{Z}^n_+\}$$

und beweisen mehrere Schranken an die Anzahl der von Null verschiedenen Komponenten in einer optimalen Lösung. Es wird sich herausstellen, dass es stets eine optimale Lösung gibt, deren Anzahl der von Null verschiedenen Einträge sich durch ein *Polynom* in der Anzahl der Ungleichungen und der maximalen Größe der Einträge in A beschränken lässt. Dieses Ergebnis folgt aus dem *ganzzahligen* Analogon des Satzes von Carathéodory, welches in dieser Dissertation bewiesen wird. Diese Schranke ist besonders nützlich, wenn das ganzzahlige Programm aus bestimmten kombinatorischen Optimierungsproblemen abgeleitet ist und exponentiell viele Variablen enthält, denn nichtsdestotrotz können wir in diesem Fall die Existenz einer optimalen Lösung polynomieller Größe zeigen.

Eine solche Anwendung ist das *Cutting Stock-Problem*. Die Spalten der Matrix A in der IP-Formulierung des Problems sind exakt die nicht-negativen ganzzahligen Lösungen der Knapsack-Ungleichung $ax \leqslant 1$, womit ihre Anzahl exponentiell in der Eingabe ist. Dennoch können wir beweisen, dass eine optimale Lösung polynomieller Größe existiert und damit das Cutting Stock-Problem in NP liegt, was bis dato nicht bekannt war. Wir setzen die Untersuchung dieses ganzzahligen Programms fort und leiten einige Resultate für die Güte der LP-Relaxation bzw. des Integrality Gaps ab. Schließlich geben wir eine IP-Formulierung *polynomieller Größe* für das Cutting Stock-Problem an.

Acknowledgements

First of all, I thank my supervisor Professor Dr. Friedrich Eisenbrand for introducing me to the subject of integer programming and combinatorial optimisation, his support and encouragement, invaluable help, and for many other things. I cannot imagine a better adviser and, due to this reason, had to follow him all the way during his practical study of the "travelling researcher problem." Almost all results presented in this thesis were obtained in collaboration with him.

I am grateful to András Sebő for inviting me to visit his research group in Grenoble and many valuable discussions during my stay there, which gave rise to some results presented in Chapter 6. Yet, I thank him for his willingness to take the second assessment for this thesis.

And of course, I thank my parents, my grandmother, all of my friends in Germany and Russia and, specially, Svetlana.

Table of Contents

Chapter 1

Introduction

"The secret of being boring is to say everything."

—VOLTAIRE

The epigraph came to my mind almost immediately after I had finished writing this introductory chapter. "You are boring yourself," such was the first impression. And indeed, I should have said something about the importance of integer programming in nowadays life, in particular, in computer science and management. I should have given some impressive practical examples and tried to convince the reader that his life would be much worse if integer programming never existed. But I am sure that all these things, if I said them, would sound very artificial and insincere, because my real motivation to study this subject has nothing to do with them. Integer programming is just interesting itself, and this is the only thing I can tell to argue my personal interest in it.

The term integer linear programming, or simply *integer programming*, refers to the problem of optimising a linear function over the integral vectors of a polyhedron. Many combinatorial optimisation problems can be modelled as integer programs in a very straightforward way, that motivated the intensive study of different aspects of integer programming in both theory and practice. Our contribution concerns rather the *theoretical* side of this research. We consider a number of complexity issues related to integer programming, like polynomial-time solvability of certain problems and bounds on the size of solutions.

The first well-known fact to be mentioned about the complexity of integer programming is that it is NP-complete in general; hence, it is very unlikely that there is

an efficient algorithm for it. On the other hand, there are several subclasses of the problem for which such an algorithm *does* exist. Thus, for example, if the underlying polyhedron is integral, i.e., equal to the convex hull of the integral vectors contained in it, we can drop the integrality requirement and apply linear programming techniques to solve the problem.

Yet, integer programming problems with a fixed number of variables are solvable in polynomial time, as was shown by Lenstra (1983). Later, Kannan (1992) considered a generalisation of the problem: given an $m \times n$-matrix A and a polyhedron Q in $\mathbb{R}^m$, test the following statement:

$$\forall b \in Q \quad \exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \ ? \tag{1.1}$$

Essentially, this is a *parameterised* version of integer programming, with b being a parameter. Kannan described an algorithm that answers this question in polynomial time if n and the affine dimension of the polyhedron Q are fixed. It is clear that the requirement for n to be fixed is necessary, since for Q, containing only one vector, we have just an ordinary integer programming problem. But can we make the affine dimension of Q variable? In other words, does there exist a polynomial-time algorithm for the problem (1.1) in fixed dimension? In this thesis we establish such an algorithm.

This result gives rise to an algorithm that computes the so-called *integer programming gap* for a family of integer programs

$$\max\{cx : Ax \leqslant b,\ x \text{ integral}\}, \tag{1.2}$$

that is, the maximum difference between the optimum value of its linear programming relaxation and the optimum value of the integer program (1.2) itself, over all vectors b for which the integer program is feasible.

Combinatorial optimisation problems often admit integer programming formulations of the form

$$\min\{cx : Ax = b,\ x \geqslant 0 \text{ integral}\}, \tag{1.3}$$

where the columns of the matrix A are derived from the input to the problem, and sometimes their number is exponential in the input size; especially, this is the case for various partitioning, covering, and packing problems. Such integer programs are usually tackled by means of column generation methods. The common idea of these methods is to generate the columns of A in run-time — during the solution process — when needed.

Perhaps, the most popular combinatorial optimisation problem leading to an integer program of the form (1.3) is the *cutting stock problem*. In this case, the

columns of A are all integral non-negative solutions of the linear inequality $ax \leqslant 1$, for some vector a, while c is the all-one vector. The problem is known to be NP-hard, but the existence of an optimum solution of polynomial size has not yet been proved. We show that such a solution *exists*; more generally, there always exists an optimum solution of the integer program (1.3) whose size is polynomially bounded in the number of constraints, the size of the optimum value, and the maximum size of an entry in A. Like existence of a basic optimum solution for linear programming problems follows from Carathéodory's theorem, the bound mentioned above appears to be just a corollary of the Carathéodory-type theorem for *integer cones*.

It was observed that the integer programming gap for integer programs derived from the cutting stock problem is usually small. We describe the bound, which follows directly from the algorithm of Karmarkar and Karp (1982). Although the proof is just a straightforward application of the algorithm, the bound was never stated explicitly. At last, we propose other integer programming formulations for the cutting stock problem, which appear to have *polynomial size*. These rely on the Carathéodory-type bounds for integer cones.

1.1 Outline

In Chapter 2 we review some preliminaries on linear algebra, algorithms and complexity theory, and theory of linear and integer programming, which are necessary for understanding the rest of the thesis. We assume, however, that the reader is already familiar with most of the facts and definition presented in this chapter — it was mostly added for reference.

Chapter 3 is a crucial preparation step towards a solution of the problem (1.1). We begin this chapter by describing — very briefly — the Lenstra's algorithm for integer programming in fixed dimension. The following sections are devoted to the adaptation of this approach to the case of the varying right-hand side in the system of linear inequalities $Ax \leqslant b$, that finally results in the important *structural theorem*.

The applications of the structural theorem, including the algorithm for parameterised integer programming and computation of the integer programming gap, are discussed in Chapter 4.

In Chapter 5 we turn to integer programs in standard form (1.3), to derive a desired bound on the size of an optimum solution. We introduce the notion of an integer cone and prove a number of theorems, which can be seen analogous to the well-known Carathéodory's theorem. These theorems immediately imply the corresponding bounds on the size of an optimum solution.

Chapter 6 is devoted to the cutting stock problem or, more precisely, to the inte-

ger programming formulation of this problem described earlier. First, we apply the results of Chapter 5 to show the existence of an optimum solution of polynomial size. Then we consider the integer programming gap for these integer programs: we show that the algorithm of Karmarkar and Karp (1982) implies that the integer programming gap is of order $O(\log^2 d)$, where d is the number of constraints in the integer program. Finally, we describe two different integer programming formulations, both of polynomial size in the input.

The last chapter summarises our basic results once again, and lists some interesting open problems related to the subject of this thesis. We remark that each of the chapters begins with a more detailed description of its content and main results presented there.

Chapters 3 and 4 are mostly borrowed from Eisenbrand and Shmonin (2007). Chapter 5 and parts of Chapter 6 are from Eisenbrand and Shmonin (2006). Some parts of Chapter 6 are done in collaboration with András Sebő.

Chapter 2

Preliminaries

We expect the reader to be familiar with basic set theory, linear algebra, algorithms and complexity theory, and theory of linear and integer programming. In this chapter we summarise some definitions and results from these fields, and describe the notation to be used throughout the whole thesis.

2.1 Basic definitions and notation

Given a set X, we write $x \in X$ if x is an element of X, and $x \notin X$ otherwise. If X is a subset of a set Y, we write $X \subseteq Y$. If, in addition, $X \neq Y$, we write $X \subset Y$ and say that X is a *proper subset* of Y. The intersection, union, difference and the Cartesian product of sets X and Y are denoted by $X \cap Y$, $X \cup Y$, $X \setminus Y$, and $X \times Y$, respectively. The symbol $\emptyset$ refers to the empty set. The cardinality of a set X is denoted by $|X|$.

Sets $X_1, \ldots, X_n$ are called *disjoint* if they are *pairwise* disjoint, i.e., $X_i \cap X_j = \emptyset$ for all pairs of distinct indices i and j. A *partition* of a set X is a collection of disjoint sets $X_1, \ldots, X_n$ such that

$$X = \bigcup_{i=1}^{n} X_i.$$

The symbols $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{Z}$ stand for the sets of real numbers, rational numbers and integers, respectively. Their restrictions to the non-negative numbers are, respectively, $\mathbb{R}_+$, $\mathbb{Q}_+$, and $\mathbb{Z}_+$. For any real number α, the symbol $\lfloor \alpha \rfloor$ denotes the largest integer not exceeding α. Similarly, $\lceil \alpha \rceil$ is the smallest integer that is greater than or equal to α. Both $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are called *rounding* operations. The absolute value of a number α is denoted by $|\alpha|$.

A number α is said to *divide* a number β if there exists an integer γ such that $\beta = \gamma\alpha$. For rational numbers $\alpha_1, \ldots, \alpha_n$, not all equal to 0, there always exists the largest

rational number α dividing each of $\alpha_1, \ldots, \alpha_n$; this number is called the *greatest common divisor* of $\alpha_1, \ldots, \alpha_n$ and denoted by $\gcd(\alpha_1, \ldots, \alpha_n)$. If $\gcd(\alpha_1, \ldots, \alpha_n) = 1$, then the numbers $\alpha_1, \ldots, \alpha_n$ are called *relatively prime*.

For two functions $f, g : \mathbb{Z}_+ \to \mathbb{R}_+$, we write $f = O(g)$, or equivalently, $g = \Omega(f)$, if there exists a constant C and an index n_0 such that $f(n) \leqslant C\, g(n)$ for all $n \geqslant n_0$. If both $f = O(g)$ and $f = \Omega(g)$ hold, we write $f = \Theta(g)$.

2.2 Matrices and linear algebra

Let $\mathbb{F}$ be a set. The symbol $\mathbb{F}^n$ denotes the set of all n-tuples, or n-vectors, of elements from $\mathbb{F}$. The set of $m \times n$-matrices with all entries taken from $\mathbb{F}$ is denoted by $\mathbb{F}^{m\times n}$. We shall use this notation for the sets $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$ and their restrictions $\mathbb{R}_+$, $\mathbb{Q}_+$, and $\mathbb{Z}_+$ only. Vectors in $\mathbb{Q}^n$ and matrices in $\mathbb{Q}^{m\times n}$ are called *rational*, while vectors in $\mathbb{Z}^n$ and matrices in $\mathbb{Z}^{m\times n}$ are *integral*.

Given a vector $x \in \mathbb{R}^n$, we write x_i to refer to the i-th component of x ($i = 1, \ldots, n$). For a number α, the *all-α vector* is a vector with all components equal to α. The symbols $\lceil x \rceil$ and $\lfloor x \rfloor$ denote the vectors obtained by the component-wise application of the operations $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$, respectively. For two vectors x and y in $\mathbb{R}^n$, we write $x \leqslant y$ if $x_i \leqslant y_i$ for each $i = 1, \ldots, n$. Similarly, we write $x < y$ if $x_i < y_i$ for each $i = 1, \ldots, n$.

In order to simplify our notation, we distinguish between row-vectors and column-vectors. Furthermore, suppose that A is a matrix and x, y, b, and c are vectors. Following Schrijver (1986), we agree that whenever we use notation like

$$Ax = b, \quad Ax \leqslant b, \quad yA = c,$$

we implicitly assume compatibility of sizes of A, x, y, b, and c. In particular, b and x are column-vectors, while y and c are row-vectors. Similarly, if c and x are vectors and we write cx, then c is a row-vector and x is a column-vector, both having the same number of components.

The rank of the matrix $A \in \mathbb{R}^{m\times n}$ is denoted by $\operatorname{rank}(A)$. If all entries of A are equal to 0, we write $A = 0$. If A is a square matrix ($m = n$), then $\det(A)$ is the determinant of A. We say that A has *full column rank* if $\operatorname{rank}(A) = n$. If $\operatorname{rank}(A) = m$, then A is said to be of *full row rank*. A square matrix A is called *non-singular* if $\det(A) \neq 0$. In this case there exists the unique *inverse matrix* A^{-1} with the property $AA^{-1} = A^{-1}A = I$.

The sets $\mathbb{R}^n$ and $\mathbb{Q}^n$ are essentially linear spaces over the fields $\mathbb{R}$ and $\mathbb{Q}$, respectively, with addition of vectors and multiplication of vectors with scalars defined as

usual. By e_i ($i = 1,\dots,n$) we denote the *i-th unit vector* in $\mathbb{R}^n$; thus, the i-th component of e_i is equal to 1, while all other components are zeros. For two sets $X, Y \subseteq \mathbb{R}^n$ and a number α, we define

$$X + Y := \{x + y : x \in X,\, y \in Y\},$$

and

$$\alpha X := \{\alpha x : x \in X\}.$$

If Y consists of only one vector y, we simply write $X + y$ instead of $X + \{y\}$ and say that $X + y$ is the *translate* of X along the vector y.

We need two different norms defined on $\mathbb{R}^n$; namely, the *l_∞-norm*

$$\|x\|_\infty := \max\{|x_i| : i = 1,\dots,n\}$$

and the *Euclidean norm*, or *l_2-norm*,

$$\|x\|_2 := \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}.$$

It is easy to see that

$$\|x + y\|_2 = \|x\|_2 + \|y\|_2$$

if and only if $x = \lambda y$ for some number λ, i.e, vectors x and y are collinear.

It is well-known that any linear transformation $L : \mathbb{R}^n \to \mathbb{R}^m$ can be represented by a matrix $A \in \mathbb{R}^{m\times n}$, so that $Ax = L(x)$ for all $x \in \mathbb{R}^n$. Similarly, any affine transformation $T : \mathbb{R}^n \to \mathbb{R}^m$ is completely determined by a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $b \in \mathbb{R}^m$, with $Ax + b = T(x)$ for all $x \in \mathbb{R}^n$. We say that a linear transformation is *rational* if it is defined by a rational matrix. Analogously, an affine transformation is *rational* if it is defined by a rational matrix and a rational vector. Finally, we remark that usually we do not distinguish between transformations and the matrices representing them.

Let X be a set of vectors in $\mathbb{R}^n$. The *linear hull* of X, denoted by $\mathrm{lin}(X)$, is just the subspace of $\mathbb{R}^n$ spanned by X. The *affine hull* of X is the set

$$\mathrm{aff}(X) := \mathrm{lin}(X - x) + x,$$

where x is an arbitrary vector from X.[1]

[1]The set $\mathrm{aff}(X)$ appears to be independent of the choice of x. Similarly, $\mathrm{aff}(X) - x$ is the same linear space for all $x \in \mathrm{aff}(X)$.

We say that a set X is *convex* if $\lambda x + (1-\lambda) y \in X$ for any $x, y \in X$ and any $0 < \lambda < 1$. The *convex hull* of X, $\mathrm{conv}(X)$, is the minimal convex set containing X. Equivalently,

$$\mathrm{conv}(X) := \left\{ \sum_{i=1}^{t} \lambda_i x_i : t \geqslant 1,\, x_1, \ldots, x_t \in X,\, \lambda_1, \ldots, \lambda_t \geqslant 0,\, \sum_{i=1}^{t} \lambda_i = 1 \right\}.$$

Theorem 2.1 (Carathéodory's theorem)**.** *For any set $X \subseteq \mathbb{R}^n$ and any $x \in \mathrm{conv}(X)$, there exist affinely independent vectors $x_1, \ldots, x_k$ in X such that $x \in \mathrm{conv}(\{x_1, \ldots, x_k\})$.*

A set X in $\mathbb{R}^n$ is a *convex cone*, or simply a *cone*, if $X \neq \emptyset$ and $\lambda x + \mu y \in X$ for any $x, y \in X$ and any $\lambda, \mu \geqslant 0$. The *cone generated by* X, $\mathrm{cone}(X)$, is the smallest convex cone containing X; equivalently, it is the set of all non-negative linear combinations of vectors from X:

$$\mathrm{cone}(X) := \left\{ \sum_{i=1}^{t} \lambda_i x_i : t \geqslant 0,\, x_1, \ldots, x_t \in X,\, \lambda_1, \ldots, \lambda_t \geqslant 0 \right\}.$$

If X is finite, we say that $\mathrm{cone}(X)$ is *finitely generated*. A cone in $\mathbb{R}^n$ is called *simplicial* if it is generated by n linearly independent vectors.

Theorem 2.2 (Carathéodory's theorem)**.** *For any set $X \subseteq \mathbb{R}^n$ and any $x \in \mathrm{cone}(X)$, there exist linearly independent vectors $x_1, \ldots, x_k$ in X such that $x \in \mathrm{cone}(\{x_1, \ldots, x_k\})$.*

The *affine dimension* of a set X in $\mathbb{R}^n$, denoted by $\dim(X)$, is the dimension of the linear subspace $\mathrm{aff}(X) - x$, where x is an arbitrary vector from X.

2.3 Algorithms and complexity

Unless explicitly stated otherwise, we always assume that the data is stored in *binary encoding* as finite $\{0,1\}$-strings. The *size* of the data is the total length of these strings. For a rational number $\alpha = p/q$, where p and q are relatively prime integers with $q \geqslant 1$, we have

$$\mathrm{size}(\alpha) := 1 + \lceil \log(|p| + 1) \rceil + \lceil \log(q + 1) \rceil.$$

The size of a rational vector $x \in \mathbb{Q}^n$ is roughly the sum of the sizes of its components:

$$\mathrm{size}(x) := n + \sum_{i=1}^{n} \mathrm{size}(x_i).$$

The size of a rational matrix $A = [\alpha_{ij}] \in \mathbb{Q}^{m\times n}$ is

$$\text{size}(A) := mn + \sum_{i=1}^{m}\sum_{j=1}^{n} \text{size}(\alpha_{ij}).$$

The size of a linear inequality $ax \leqslant \beta$ or equation $ax = \beta$, where a is a rational row-vector and β is a rational number, is equal to

$$1 + \text{size}(a) + \text{size}(\beta).$$

The size of a system of linear inequalities $Ax \leqslant b$ or equations $Ax = b$, where A is a rational matrix and b is a rational vector, is

$$1 + \text{size}(A) + \text{size}(b).$$

As we have agreed in the previous section, linear and affine transformations are identified with the matrices representing them; hence, the size of a rational linear transformation is just the size of the corresponding matrix (and similarly for affine transformations).

We use the computational model of the *random access machine* (*RAM*) operating on $\{0,1\}$-strings. For a formal description of the RAM computational model and basic complexity issues we refer to Garey and Johnson (1979) and Papadimitriou (1994). Here we assume that the following *arithmetic operations* are available: addition, subtraction, multiplication, division, comparison, rounding, and taking logarithm.

Let Σ be an alphabet (in our case $\Sigma = \{0,1\}$). A *decision problem* is a subset $\Pi \subseteq \Sigma^*$, where Σ^* denotes the set of all strings of symbols from the alphabet Σ. Informally, it is a problem that can be answered by 'yes' or 'no', with Π representing the set of inputs for which the answer is 'yes'.

A *polynomial-time algorithm* is an algorithm that terminates after a number of steps bounded by a polynomial in the size of the input data. Such algorithms are also called *efficient*. A decision problem Π is *polynomial-time solvable* if it can be solved by a polynomial-time algorithm. The class of the decision problems solvable in polynomial time is denoted by P.

The class NP comprises the decision problems whose solutions can be verified in polynomial time. Formally, $\Pi \in \text{NP}$ if there exists a decision problem $\Pi' \in \text{P}$ and a polynomial $p : \mathbb{Z}_+ \to \mathbb{Z}_+$ such that for each string $\sigma \in \Sigma^*$, σ is in Π if and only if there exists a string $\sigma' \in \Sigma^*$ of size at most $p(\text{size}(\sigma))$ with $\sigma\sigma' \in \Pi'$. The string σ' is called a *certificate* for σ. Trivially, $\text{P} \subseteq \text{NP}$. One of the most fundamental questions

in complexity theory is whether P $\neq$ NP. Although it is widely believed to be true, nobody has been able to prove this so far.

A decision problem $\Pi \subseteq \Sigma^*$ is *Karp-reducible*, or simply *reducible*, to a problem $\Pi' \subseteq \Sigma^*$ if there exists a polynomial-time algorithm that returns, for any $\sigma \in \Sigma^*$, a string $\sigma' \in \Sigma^*$ such that $\sigma \in \Pi$ if and only if $\sigma' \in \Pi'$. This definition implicitly requires size(σ') to be bounded by a polynomial in size(σ). A problem Π is NP-*hard* if each problem in NP is reducible to Π. If, in addition, $\Pi \in$ NP, it is called NP-*complete*. It is the theorem of Cook (1971) that there exist NP-complete problems.

A problem $\Pi \subseteq \Sigma^*$ is *Turing-reducible*, or *polynomially reducible*, to a problem $\Pi' \subseteq \Sigma^*$, if there exists an algorithm such that, given an input $\sigma \in \Sigma^*$ and an algorithm $\mathscr{A}$ for the problem Π', solves the problem Π for the input σ in time bounded by a polynomial in the size of σ and the running time function of $\mathscr{A}$. If Π is polynomially reducible to Π' and Π' is polynomially reducible to Π, the problems Π and Π' are called *polynomially equivalent*.

An *optimisation problem* is usually stated as follows (maximisation problem is analogous):

$$\min\{f(x) : x \in X_\sigma\}, \tag{2.1}$$

where X_σ is a collection of elements derived from the input σ of the problem, f is a rational-valued function. The associated decision problem is: Given a rational number α, is there an $x \in X_\sigma$ with $f(x) \leqslant \alpha$? If γ is an upper bound on the size of the minimum value, we can find the optimum by solving $O(\gamma)$ associated decision problems (via binary search). Thus, if γ is bounded by a polynomial in the input size, the optimisation problem is polynomially equivalent to its decision version.

Karp (1972) showed that several fundamental combinatorial optimisation problems (the travelling salesman problem, the maximum clique problem, the maximum cut problem) are NP-complete. Since then almost all combinatorial optimisation problems have been proved to be either solvable in polynomial time, or NP-complete.

In this thesis we study a number of optimisation problems, assuming some of their parameters to be *fixed*. By "fixing" a parameter, we mean that it does not belong to the input of the problem, and hence, does not contribute its size. Particularly, when computing the running time of the algorithm, we may treat this parameter as a constant.

Apparently, we mention some approximation algorithms. For the sake of completeness, we define that a polynomial-time algorithm $\mathscr{A}$ is an *α-approximation algorithm* ($\alpha > 1$) for the optimisation problem (2.1) if for any input σ, it yields a solution $x \in X_\sigma$ of value $f(x) \leqslant \alpha \cdot \mathrm{OPT}(\sigma)$, where $\mathrm{OPT}(\sigma)$ denotes the optimum value. A *polynomial-time approximation scheme* is a family of algorithms $\{\mathscr{A}_\varepsilon : \varepsilon > 0\}$ such

that each $\mathscr{A}_\varepsilon$ is an $(1+\varepsilon)$-approximation algorithm. It is a *fully polynomial-time approximation scheme* if, additionally, each algorithm $\mathscr{A}_\varepsilon$ runs in time polynomial in $1/\varepsilon^2$.

2.4 Polyhedra and linear programming

An excellent reference on theory of polyhedra and linear programming is Schrijver (1986). All facts and algorithms we mention in this section are perfectly treated in this book.

Let $a \in \mathbb{R}^n$ be a row-vector and β a number. The sets

$$H_< := \{x : ax < \beta\}, \quad H_\leqslant := \{x : ax \leqslant \beta\}, \quad H_= := \{x : ax = \beta\}$$

are called, respectively, an *open half-space*, a *closed half-space*, and a *hyper-plane* in $\mathbb{R}^n$. A *partially open polyhedron* P is the intersection of finitely many closed or open half-spaces. If P can be defined by means of closed half-spaces only, we say that it is a *closed polyhedron*, or simply a *polyhedron*.[1] A bounded (partially open) polyhedron is called a (*partially open*) *polytope*. It is known that P is a polytope if and only if it is the convex hull of finitely many vectors. The polyhedron P in $\mathbb{R}^n$ is said to be *full-dimensional* if $\dim(P) = n$.

A (closed or open) half-space is called *rational* if it can be defined by an inequality with rational coefficients and a rational right-hand side. The corresponding hyper-plane is then *rational*, too. Finally, a partially open polyhedron is *rational* if it can be defined by means of rational half-spaces.

A polyhedron of the form

$$C = \{x : Ax \leqslant 0\},$$

is called a *polyhedral cone*. It is well-known that a cone C is polyhedral if and only if it is finitely generated. We shall need the following lemma (directly follows from Theorem 10.2 in Schrijver (1986)).

Lemma 2.3. *Let $x_1, \ldots, x_n$ be linearly independent vectors in $\mathbb{R}^n$. Then*

$$\mathrm{cone}(\{x_1, \ldots, x_n\}) = \{x : Ax \leqslant 0\},$$

[1] Traditionally, linear and integer programming deals only with closed polyhedra. We need the notion of partially open polyhedra in Chapters 3 and 4 to be able to partition the space $\mathbb{R}^m$, which is certainly impossible by means of closed polyhedra only. We remark that many results of linear and integer programming are easily transformed to the case of partially open polyhedra.

for some matrix $A \in \mathbb{R}^{n\times n}$. Moreover, the size of each row in A is at most $4n^2\phi$, where ϕ is the largest size of a vector in $\{x_1, \ldots, x_n\}$.

Each polyhedron can be decomposed into a polytope and a polyhedral cone in the following sense.

Theorem 2.4 (Decomposition of polyhedra)**.** *A set $P \subseteq \mathbb{R}^n$ is a polyhedron if and only if $P = Q + C$ for some polytope Q and some polyhedral cone C.*

In fact, in this decomposition

$$C = \{y : y + x \in P \text{ for all } x \in P\}$$

and is called the *characteristic cone* of P. It is a common agreement that the characteristic cone of the empty polyhedron is $C = \{0\}$. The non-zero vectors in C are called the *infinite directions* of P. The characteristic cone of a rational polyhedron is also rational. The *linearity space* of $P = \{x : Ax \leqslant b\}$ is the set

$$C \cup (-C) = \{y : Ay = 0\}.$$

If the linearity space of P is equal to $\{0\}$, the polyhedron P is called *pointed*. It can be shown that if P is pointed, then there exist hyper-planes that intersect P at exactly one vector; this vector is then called a *vertex* of P. We remark that any simplicial cone is pointed and has exactly one vertex, which is 0.

Let $A \in \mathbb{R}^{m\times n}$ be a matrix. The set of the right-hand sides $b \in \mathbb{R}^m$ for which the system $Ax \leqslant b$ has a solution is a polyhedron in $\mathbb{R}^m$ and can be computed by exploiting the well-known *Fourier–Motzkin elimination* procedure. This procedure runs in polynomial time if n is fixed.

Linear programming problem is formulated as follows: Given a matrix A and vectors b and c, compute

$$\max\{cx : Ax \leqslant b\}. \tag{2.2}$$

Geometrically it can be interpreted as the problem of finding a furthest point of the polyhedron

$$P = \{x : Ax \leqslant b\}$$

with respect to the direction c. Indeed, if we consider the family of hyper-planes

$$H_\delta = \{x : cx = \delta\},$$

then the optimum value of the problem (2.2) is the largest δ such that the intersection $P \cap H_\delta$ is non-empty, see Figure 2.1. There are several polynomially equivalent forms of a linear programming problem; for example,

$$\min\{cx : Ax = b,\ x \geqslant 0\}$$

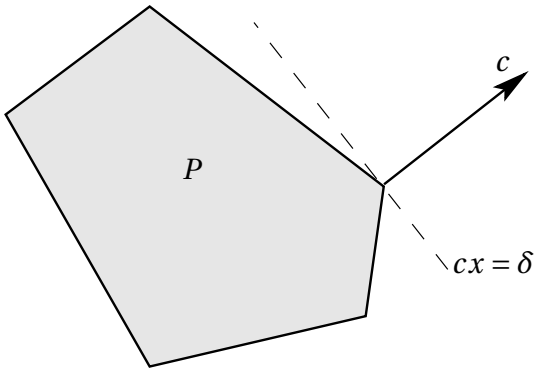

Figure 2.1: Linear programming geometrically.

is a linear program in *standard form*.

We say that the linear program (2.2) is *feasible* if there exists a vector x satisfying $Ax \leqslant b$; otherwise, it is *infeasible*. It is *bounded* if its optimum is finite. It follows that the linear program (2.2) is unbounded if and only if there exists a vector y in the characteristic cone of P such that $cy > 0$. A "certificate" for optimality of a given solution follows from the following theorem.

Theorem 2.5 (Duality theorem)**.** *Let A be a matrix and b, c vectors. Then*

$$\max\{cx : Ax \leqslant b\} = \min\{yb : yA = c,\ y \geqslant 0\}, \tag{2.3}$$

if at least one of these optima is finite.

Consequently, if at least one of the optima in (2.3) is finite, then both are finite. Let x and y be feasible solutions for linear programs in (2.3), i.e.,

$$Ax \leqslant b, \quad yA = c, \quad y \geqslant 0.$$

Complementary slackness states that x and y are optimum solutions if and only if

$$y(b - Ax) = 0.$$

For a matrix A and an index set $N \subseteq \{1, \ldots, m\}$, let A_N denote the matrix consisting of the rows of A whose index is in N. We say that I is a *basis* for A if

$$\operatorname{rank}(A) = \operatorname{rank}(A_N) = |N|.$$

Complementary slackness together with Carathéodory's theorem (see Theorem 2.2) imply the following important result.

Theorem 2.6. *If the optima in* (2.3) *are finite, then there exists a basis N of A such that*

(a) the minimum is attained by a row-vector y such that $y_i = 0$ for all $i \notin N$;

(b) the maximum is attained by a vector x such that $A_N x = b_N$.

In particular, if A has full column rank, then both x and y are uniquely determined by N and called *basic solutions*. In fact, basic solutions of the system $Ax \leqslant b$ are exactly the vertices of the polyhedron $P = \{x : Ax \leqslant b\}$.

Perhaps, the most efficient method for solving linear programming problems in practice is the *simplex method*, introduced by Dantzig (1951). Nonetheless, most of its variants have been proved to take exponential time in the worst case and none of them is shown to run in polynomial time.

The first polynomial-time algorithm for solving linear programming problems—the *ellipsoid method*—was proposed by Khachiyan (1979). Although practical usage of this algorithm is very limited (or even infeasible), its theoretical importance in combinatorial optimisation is difficult to overestimate. We consider some implications of this algorithm in Section 2.5.

Karmarkar (1984) showed that *interior-point methods* can also be used to solve linear programming problems in polynomial time. Furthermore, these methods turned out to be efficient in practice, too.

We conclude with the remark that all these methods can be implemented in such a way that—for matrices of full column rank—they yield basic optimum solutions for both primal and dual linear programs.

2.5 The ellipsoid method

As we have already mentioned in the previous section, the ellipsoid algorithm became a very powerful tool in theoretical study of combinatorial optimisation problems. Grötschel et al. (1981) observed that for the ellipsoid method to work, we do not need to list all constraints of a linear program explicitly; in fact, it suffices to provide a tool for generating them when needed. A good description of this approach, together with its various applications in combinatorial optimisation, can be found in Schrijver (1986) and Grötschel et al. (1993). In this section we only formulate the main result.

Let Σ be an alphabet and let $\Pi \subseteq \Sigma^*$ be a family of words (inputs to the problem). Suppose that for each input $\sigma \in \Pi$, there is an associated rational polyhedron P_σ in $\mathbb{R}^{n_\sigma}$. We assume that n_σ is known in advance and that P_σ is defined by a system of linear inequalities, each of size bounded by a polynomial in size(σ).

The *separation problem* for the family of polyhedra $\{P_\sigma : \sigma \in \Pi\}$ is the following:

> Given $\sigma \in \Pi$ and a vector $z \in \mathbb{R}^{n_\sigma}$, decide whether z belongs to P_σ, and if not, find a vector $a \in \mathbb{R}^{n_\sigma}$ such that $ax < az$ for all $x \in P_\sigma$.

We say that the separation problem is polynomial-time solvable if it is solvable in time polynomial in size(σ) and size(z).

The corresponding *optimisation problem* is:

> Given $\sigma \in \Pi$ and a row-vector $c \in \mathbb{R}^{n_\sigma}$, find a vector $x \in P_\sigma$ maximising cx over P_σ or an infinite direction y of P_σ with $cy > 0$, if either of them exists.

This problem is said to be polynomial-time solvable if it is solvable in time polynomial in size(σ) and size(c).

The following theorem states that these two problems are, in fact, polynomially equivalent.

Theorem 2.7 (Equivalence of separation and optimisation)**.** *The separation problem for a family of polyhedra $\{P_\sigma : \sigma \in \Pi\}$ is solvable in polynomial time if and only if the optimisation problem for $\{P_\sigma : \sigma \in \Pi\}$ is solvable in polynomial time.*

Finally, we remark that the ellipsoid algorithm is not restricted to closed polyhedra only—it can also be used to find a vector in a given *partially open* polyhedron.

2.6 Integer programming and lattices

Many combinatorial optimisation problems can be formulated as maximising (or minimising) a linear function over the *integral* vectors in a polyhedron:

$$\max\{cx : Ax \leqslant b,\ x \text{ integral}\}, \tag{2.4}$$

where A is a rational matrix, b and c are rational vectors. Problems of this form are called *integer linear programming* problems. The corresponding linear program

$$\max\{cx : Ax \leqslant b\} \tag{2.5}$$

is called the *linear programming relaxation* of (2.4).

It is not surprising that integer programming is NP-complete in general. However, there are several classes of integer programs for which polynomial-time algorithms are known to exist.

A polyhedron P is an *integer polyhedron* if it is the convex hull of the integral vectors contained in P. In particular, if P is pointed, then all its vertices are integral.

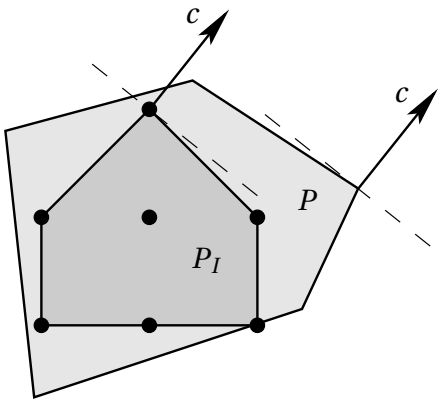

Figure 2.2: Integer hull of a polyhedron.

Theorem 2.8. *Let P be a rational polyhedron in $\mathbb{R}^n$. Then P is integer if and only if for each $c \in \mathbb{Q}^n$, the linear program* (2.5) *has an integral optimum solution whenever it is finite.*

Moreover, if P is an integer polyhedron, we can find an integral optimum solution of the linear programming problem (2.5), if it exists, in polynomial time. In other words, integer programming over integer polyhedra is polynomial-time solvable.

A system of linear inequalities $Ax \leqslant b$, with $A \in \mathbb{Q}^{m\times n}$ and $b \in \mathbb{Q}^m$, is called *totally dual integral* if for each $c \in \mathbb{Z}^n$, the linear program

$$\min\{yb : yA = c,\ y \geqslant 0\}$$

has an integral optimum solution y whenever it is finite.

Theorem 2.9. *If $Ax \leqslant b$ is a totally dual integral system and b is an integral vector, then the polyhedron $\{x : Ax \leqslant b\}$ is integer.*

Given a rational polyhedron

$$P = \{x : Ax \leqslant b\} \subseteq \mathbb{R}^n,$$

we define the *integer hull* of P as the convex hull of all integral vectors lying in P:

$$P_I := \mathrm{conv}(P \cap \mathbb{Z}^n).$$

P_I is an integer polyhedron, and therefore, the integer program (2.4) is equivalent to the linear program

$$\max\{cx : x \in P_I\};$$

see Figure 2.2. Usually, the integer hull P_I is the intersection of exponentially many half-spaces. Nevertheless, it is often the case that the separation problem for P_I can be solved in polynomial time, which yields a polynomial-time optimisation algorithm, see Theorem 2.7.

The problem (2.4) is polynomially equivalent to the following decision problem

$$\exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \ ? \tag{2.6}$$

Indeed, it can be shown that if the optimum (2.4) is finite, then there exists an optimum solution whose size is bounded by a polynomial in the input size. Therefore, we can use binary search with polynomially many calls to the subroutine solving (2.6) to find the optimum (2.4).

Lenstra (1983) was the first to develop a polynomial-time algorithm for integer programming with a fixed number of variables. The fastest algorithm so far, which is due to Eisenbrand (2003), combines Lenstra's ideas together with randomised sampling techniques of Clarkson (1995). We also mention the algorithm of Barvinok (1994) for counting integral vectors in a given polyhedron in fixed dimension, which exploits rational functions to encode all integral vectors of a polyhedron. Obviously, this algorithm is also able to decide (2.6), and therefore, to solve integer programming problems in fixed dimension.

It turns out that the number of vertices of the integer hull of a polyhedron in fixed dimension is bounded by a polynomial in the input size. The best bound so far is given in the following theorem, which is due to Cook et al. (1992).

Theorem 2.10. *Let $P = \{x : Ax \leqslant b\}$ be a polyhedron, where $A \in \mathbb{Q}^{m\times n}$ and $b \in \mathbb{Q}^m$. Suppose that the size of each inequality in the system $Ax \leqslant b$ is at most ϕ. Then the number of vertices of P_I is at most $2m^n(6n^2\phi)^{n-1}$.*

Combining the bound of Theorem 2.10 with the Lenstra's algorithm, we can derive an algorithm to *compute* all vertices of P_I in polynomial time, if the dimension is fixed; see Hartmann (1989) for details.

A slightly modified version of the Lenstra's algorithm will be discussed in the beginning of Chapter 3, since we shall use essentially the same framework in the rest of that chapter. But before doing this, we need to introduce some basics of lattice theory, which is extensively used by Lenstra (1983) in his algorithm. We just mention that the Lenstra's algorithm can be adapted to solve integer programs with a fixed number of constraints and *mixed-integer linear programming* problems

$$\exists (x, y) \in \mathbb{Z}^n \times \mathbb{R}^k : \quad Ax + By \leqslant b \ ? \tag{2.7}$$

where A and B are rational matrices and b is a rational vector, if the number of integer variables is fixed. Moreover, it can also be adapted to the case when some of the inequalities in (2.7) are strict.

Let B be a rational non-singular square matrix with columns $b_1, \ldots, b_n \in \mathbb{Q}^n$. The *lattice* generated by B is the set

$$\Lambda(B) := \left\{ \sum_{i=1}^{n} \lambda_i b_i : \lambda_i \in \mathbb{Z},\ i = 1, \ldots, n \right\}.$$

The vectors $b_1, \ldots, b_n$ are called the *basis* of the lattice $\Lambda(B)$. The lattice generated by the unit vectors $e_1, \ldots, e_n$ is just $\mathbb{Z}^n$ and called the *standard lattice*.

An integral square matrix U is called *unimodular* if $|\det(A)| = 1$. In this case, we have $|\det(U^{-1})| = 1$ and the inverse matrix U^{-1} is also integral. A linear transformation defined by a unimodular matrix is called *unimodular*, too.

Theorem 2.11. *Let $B, B' \in \mathbb{Q}^{n\times n}$ be two rational non-singular square matrices. Then $\Lambda(B) = \Lambda(B')$ if and only if $B' = BU$ for some unimodular matrix $U \in \mathbb{Z}^{n\times n}$.*

Particularly, any unimodular matrix $U \in \mathbb{Z}^{n\times n}$ is a basis of the standard lattice $\mathbb{Z}^n$.

A matrix of full row rank is said to be in *Hermite normal form* if it has the form $\begin{bmatrix} B & 0 \end{bmatrix}$, where B is a non-singular, lower triangular, non-negative matrix, in which each row has a unique maximum entry located on the main diagonal of B. It turns out that each matrix of full column rank can be transformed into Hermite normal form by multiplying from the right with an appropriate unimodular matrix. Moreover, the Hermite normal form of a matrix is unique and can be computed in polynomial time; see, for example, Kannan and Bachem (1979). Finally, we mention that the Hermite normal form of a row-vector a is the vector $[\gamma, 0, \ldots, 0]$, where $\gamma = \gcd(\alpha_1, \ldots, \alpha_n)$.

Chapter 3

Integral Vectors in a Parameterised Polyhedron

Let us consider integer linear programs

$$\max\{cx : Ax \leqslant b\},$$

where $A \in \mathbb{Q}^{m\times n}$ is a rational matrix, $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^n$ are rational vectors, under the assumption that the number n of variables is fixed. The corresponding, polynomially equivalent, decision problem is the following:

$$\exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \ ? \tag{3.1}$$

In words, we are given a polyhedron in fixed dimension, and the question is to find an integral vector in this polyhedron. As was shown by Lenstra (1983), the problem (3.1) is polynomial-time solvable for fixed n, and we shall consider his algorithm in Section 3.1.

Kannan (1990) examined a generalisation of the problem (3.1). In his settings, the right-hand side b in the inequality system is allowed to vary over some partially open polyhedron Q in $\mathbb{R}^m$ and the question is to decide the following:

$$\forall b \in Q \quad \exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \ ? \tag{3.2}$$

We shall refer to this problem as a *parameterised integer linear programming* problem, with the right-hand side b being a *parameter*. The family of polyhedra

$$P_b := \{x \in \mathbb{R}^n : Ax \leqslant b\},$$

is called a *parameterised polyhedron* P. The question (3.2) is then equivalent to

$$\exists b \in Q : \quad P_b \cap \mathbb{Z}^n = \emptyset \ ?$$

Kannan gave an algorithm that solves the problem (3.2) in polynomial time under the assumption that both n and the affine dimension of Q are fixed. The main techniques used in his algorithm were actually developed in Kannan (1992) to tackle the famous *Frobenius problem*—given n relatively prime integers $\alpha_1, \ldots, \alpha_n$, find the largest integer that is not representable as an integral non-negative combination of $\alpha_1, \ldots, \alpha_n$—in the case when n is fixed. This problem is polynomial-time solvable if we can decide, in polynomial time, if there exists an integer $b > b_0$ such that the system

$$a_1 x_1 + \ldots + a_n x_n = b, \quad x_i \geqslant 0, \, i = 1, \ldots, n$$

has no integral solution $x_1, \ldots, x_n$; here b_0 is a given number. In this statement, the right-hand side b is required to be integer, too; however, as we shall see further, this is not a crucial restriction.

We shall also adopt many techniques from Kannan (1992), but improve the algorithm to run in time polynomial in $\dim(Q)$. In other words, we assume only n to be fixed, proving that the problem (3.2) is polynomial-time solvable in fixed dimension.

Our algorithm will be based on a *structural theorem*, which provides a description of candidate integral solutions of the system $Ax \leqslant b$ via affine transformations of the right-hand side b. Formally, we partition the partially open polyhedron Q into polynomially many regions Q_i, and for each i, find a constant number of unimodular transformations $U_{ij} : \mathbb{R}^n \to \mathbb{R}^n$ and affine transformations $T_{ij} : \mathbb{R}^m \to \mathbb{R}^n$ such that, for any $b \in Q_i$, the polyhedron P_b contains an integral vector if and only if it contains a vector $U_{ij}\lceil T_{ij}(b)\rceil$ for some index j. Thus, given a right-hand side $b \in Q_i$, we need to try only a *constant number* of candidate solutions in order to check whether the polyhedron P_b contains an integral vector!

Although the regions Q_i are no more partially open polyhedra, we still have a good description for them in terms of *integer projections*—the notion we define in Section 3.3—of partially open polyhedra. It turns out that in fixed dimension these integer projections are easy to deal with; in particular, we can efficiently test whether a given vector b belongs to Q_i, optimise a linear function over Q_i, etc.

This chapter is mostly devoted to the proof of the structural theorem; applications of this theorem, including the algorithm for parameterised integer programming, are left until Chapter 4.

3.1 Integer programming in fixed dimension

We begin by describing the algorithm of Lenstra (1983) for integer programming in fixed dimension. Basic concepts of this algorithm will then be used in the proof of

the structural theorem.

The main idea of the algorithm can be explained, intuitively, in few words: if a polyhedron contains no integral vector, then it must be "flat" along some integral direction. In order to make this formal, we introduce the notion of "lattice width." The *width* of a closed convex set K in $\mathbb{R}^n$ along a direction $c \in \mathbb{R}^n$ is defined as

$$w_c(K) := \max\{cx : x \in K\} - \min\{cx : x \in K\}.$$

The *lattice width* of K (with respect to the standard lattice $\mathbb{Z}^n$) is the minimum of its widths along all non-zero integral directions:

$$w(K) := \min\{w_c(K) : c \in \mathbb{Z}^n \setminus \{0\}\}.$$

An integral direction c attaining the above minimum is called a *width direction* of K. Observe that if c is a width direction of K, then $(-1)c$ is also a width direction of K. Also, we have

$$w(v + \alpha K) = \alpha\, w(K) \tag{3.3}$$

for any vector v and any number α; moreover, both sets K and $v + \alpha K$ have the same width directions.

Usage of the concept of lattice width in integer linear programming, as well as in algorithmic number theory relies upon the celebrated *flatness theorem*, which goes back to Khinchin (1948) who first proved it for ellipsoids in $\mathbb{R}^n$. Here we state it for *convex bodies*, i.e., bounded closed convex sets of non-zero volume.

Theorem 3.1 (Flatness theorem)**.** *There exists a constant $\omega(n)$, depending only on n, such that any convex body $K \subseteq \mathbb{R}^n$ with $w(K) \geqslant \omega(n)$ contains an integral vector.*

The constant $\omega(n)$ in Theorem 3.1 is referred to as the *flatness constant*. The best known estimate for the flatness constant $\omega(n)$ so far is $O(n^{3/2})$, which is due to Banaszczyk et al. (1999), although a linear dependence on n is conjectured; see, for example, Kannan and Lovász (1988).

Further on we shall deal with rational polyhedra rather than general convex bodies. It is easy to see that for the particular case of rational polyhedra, assumptions of non-zero volume and boundedness can safely be removed from the theorem's statement.

- *Rational polyhedra of zero volume.* If $P \subseteq \mathbb{R}^n$ is a rational polyhedron of zero volume, then it has width 0 along an integral direction orthogonal to its affine hull.[1]

[1] This is not necessarily true for polyhedra which are not rational. For a counter-example, we can consider polyhedra on the line in $\mathbb{R}^2$ defined by the equation $x_1 + \sqrt{2}x_2 = 0$: they do not contain any integral point but their width along any non-zero integral direction can be made arbitrarily large.

- *Unbounded rational polyhedra.* Let C be the characteristic cone of a rational polyhedron $P \subseteq \mathbb{R}^n$; then C is also rational. If $C = \{0\}$, then P is already bounded. If C is full-dimensional, then the set $y + C$, where y is an arbitrary vector from P, trivially contains an integral vector, as we can always allocate a unit box
$$B := \{x \in \mathbb{R}^n : 0 \leqslant x_i \leqslant 1,\ i = 1, \ldots, n\}$$
inside a full-dimensional cone, i.e., find $v \in \mathbb{R}^n$ such that $v + B \subseteq C$. Finally, if C is not full-dimensional, then $w(P)$ is finite, since it is finite along any direction orthogonal to the affine hull of C. Then we can choose a sufficiently large box
$$B_\delta := \{x \in \mathbb{R}^n : |x_i| \leqslant \delta,\ i = 1, \ldots, n\}$$
such that $w(P) = w(P \cap B_\delta)$. If $w(P) \geqslant \omega(n)$, then $P \cap B_\delta$ (and hence P) contains an integral vector by Theorem 3.1.

Thus, we have proved the following statement.

Corollary 3.2. *There exists a constant $\omega(n)$, depending only on n, such that any rational polyhedron $P \subseteq \mathbb{R}^n$ with $w(P) \geqslant \omega(n)$ contains an integral vector.*

Corollary 3.2 assumes a rational polyhedron to be closed. However, this assumption can also be dropped. To see this, consider a non-empty partially open polyhedron P. It can be represented as the sum $P = y + Q$, where y is an arbitrary vector in P and $Q := P - y$ is a partially open polyhedron; then $0 \in Q$. If P contains no integral vector, then the polyhedron $y + (1-\varepsilon)\overline{Q}$ contains no integral vector for any small $\varepsilon > 0$ (here $\overline{Q}$ denotes the *closure* of Q). Applying Corollary 3.2, we obtain
$$(1-\varepsilon) \cdot w(\overline{P}) = (1-\varepsilon) \cdot w(\overline{Q}) = w\big(y + (1-\varepsilon) \cdot \overline{Q}\big) < \omega(n)$$
for any $\varepsilon > 0$, which is equivalent to
$$w(\overline{P}) \leqslant \omega(n).$$
Obviously, we can make the above inequality strict by adding a positive number to the original value of $\omega(n)$. This gives us the following claim.

Corollary 3.3. *There exists a constant $\omega(n)$, depending only on n, such that any rational partially open polyhedron $P \subseteq \mathbb{R}^n$ with $w(\overline{P}) \geqslant \omega(n)$ contains an integral vector.*

In what follows we shall always assume that the polyhedron in question is closed and exploit Corollary 3.2 for our arguments. On the other hand, Corollary 3.3 can be

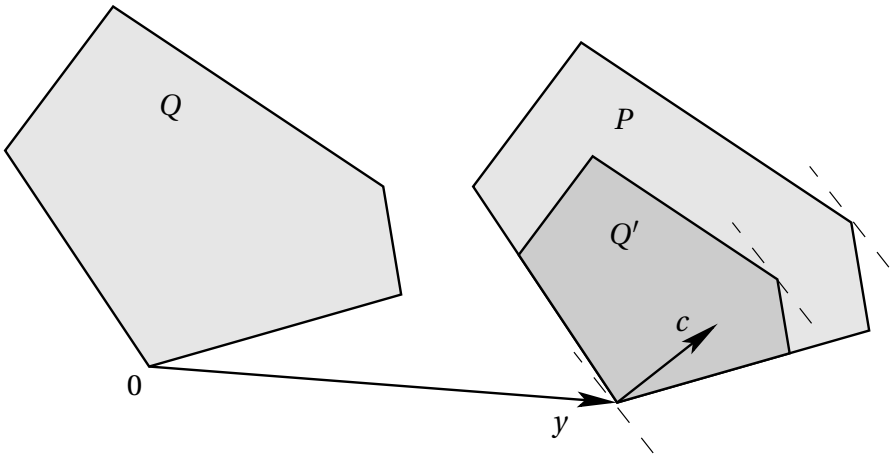

Figure 3.1: Construction of polyhedra from the proof of Lemma 3.4.

used to derive essentially the same conclusions for partially open polyhedra. In particular, the algorithm we shall describe is also applicable for the problem of finding an integral vector in a partially open polyhedron.

The following lemma is almost a direct implication of the flatness theorem for rational polyhedra.

Lemma 3.4. *Let P be a rational polyhedron in $\mathbb{R}^n$ of finite lattice width and let c be its width direction. We define*

$$\beta := \min\{cx : x \in P\}. \tag{3.4}$$

Then P contains an integral vector if and only if the polyhedron

$$P \cap \{x \in \mathbb{R}^n : \beta \leqslant cx \leqslant \beta + \omega(n)\}$$

contains an integral vector.

Proof. If $w(P) < \omega(n)$, there is nothing to prove, as

$$P \subseteq \{x \in \mathbb{R}^n : \beta \leqslant cx \leqslant \beta + \omega(n)\}.$$

Suppose that $w(P) \geqslant \omega(n)$. We can express P as the sum $P = y + Q$, with y being an optimum solution of the linear program (3.4) and $Q := P - y$. Consider the polyhedron

$$Q' := y + \frac{\omega(n)}{w(P)} Q.$$

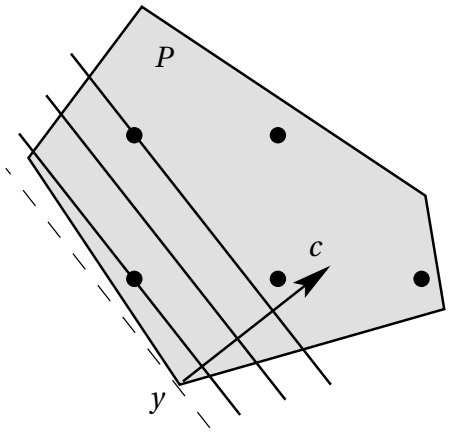

Figure 3.2: Recursion step of the Lenstra's algorithm.

Since P is a convex set and $\omega(n) \leqslant w(P)$, we have $Q' \subseteq P$ (see Figure 3.1). Furthermore, equation (3.3) implies that

$$w(Q') = \frac{\omega(n)}{w(P)}\, w(Q) = \frac{\omega(n)}{w(P)}\, w(P) = \omega(n),$$

and c is also a width direction of Q'. Applying Corollary 3.2, we conclude that Q' contains an integral vector, say z. But then $z \in P$ and

$$cz \leqslant cy + w(Q') = \beta + \omega(n).$$

This completes the proof. □

Now, suppose that we know a width direction c of a rational polyhedron

$$P = \big\{x \in \mathbb{R}^n : Ax \leqslant b\big\}.$$

Since c is an integral vector, for any integral vector $x \in P$ the inner product cx must be an integer. Together with Lemma 3.4, it allows us to split the original problem into $\omega(n)+1$ integer programming problems on lower-dimensional polyhedra

$$P \cap \big\{x \in \mathbb{R}^n : cx = \lceil \beta \rceil + j\big\}, \quad j = 0, \ldots, \omega(n),$$

where β is defined by (3.4), see Figure 3.2. The rounding operation $\lceil \beta \rceil$ is important here, as β is not necessarily an integer.

The components of c must be relatively prime integers, as otherwise we could scale c by the greatest common divisor of its components, to obtain a smaller lattice

width of P. Therefore, Hermite normal form of c is just the unit row-vector e_1 in $\mathbb{R}^n$. We can easily find a unimodular matrix U such that $cU = e_1$, introduce new variables

$$y = U^{-1}x$$

and rewrite the original system of linear inequalities $Ax \leqslant b$ in the form

$$(AU)\,y \leqslant b.$$

Since U is unimodular, the system $Ax \leqslant b$ has an integral solution if and only if the system $(AU)\,y \leqslant b$ has an integral solution: indeed, x is an integral vector if and only if y is an integral vector. But the equation

$$cx = \lceil \beta \rceil + j$$

is then transformed into

$$y_1 = \lceil \beta \rceil + j;$$

and hence, the variable y_1 can be eliminated. All together, we can proceed with a constant number of integer programming problems, each having a smaller number of variables. If n is fixed, this procedure will terminate after a polynomial number of steps.

In the above description we have omitted a question of computing a width direction. This question is actually out of scope of this thesis, since we shall use the resulting algorithm for integer programming in fixed dimension only as a "black box," while the recursion step described by Lemma 3.4 will be needed explicitly. Due to this reason, we just mention that in order to find a width direction for the polyhedron P, the algorithm of Lenstra (1983) exploits the well-known *LLL-algorithm*, proposed by Lenstra et al. (1982).

We also remark that, in its original description, the Lenstra's algorithm either finds an integral vector in P, or provides an integral direction along which P is flat. In other words, the Lenstra's algorithm uses a recursion call only for "flat" polyhedra. In contrast, the algorithm described here *always* proceeds with a recursion call, even if the actual lattice width of P is large enough to conclude that P does contain an integral vector. Such an approach proved to be more suitable for the case when the right-hand side of the inequality system defining P is allowed to vary.

To conclude this section, let us briefly summarise the issues arising when trying to generalise the above algorithm for the case of parameterised integer linear programming.

(a) The described algorithm is very sensitive to the fact that c is a width direction of the polyhedron. However, width directions of the polyhedron $\{x : Ax \leqslant b\}$ may change if we change b.

(b) Even if the width direction c remains the same, as b varies, it is not a trivial task to apply recursion: the value (3.4) also depends on b and may happen to be fractional. As a consequence, the hyper-planes

$$\{x : cx = \lceil \beta \rceil + j\}$$

are not easy to deal with.

In the following section we address the first problem and consider the width directions of a parameterised polyhedron.

3.2 Lattice width of a parameterised polyhedron

Let P be a parameterised polyhedron defined by a rational matrix $A \in \mathbb{Q}^{m\times n}$:

$$P_b = \{x : Ax \leqslant b\},$$

where the parameter b is allowed to vary over $\mathbb{R}^m$. We restrict our attention only to those b, for which P_b is non-empty, and aim to find a small set C of non-zero integral directions such that

$$w(P_b) = \min\{w_c(P_b) : c \in C\}$$

for any of these vectors b. Further on, the elements of the set C are referred to as *width directions* of the parameterised polyhedron P. It turns out that such a set can be computed in polynomial time if n is fixed. In particular, the set C itself contains only polynomially many vectors!

Without loss of generality, we can assume that A has full column rank. Indeed, if $r := \operatorname{rank}(A) < n$, then we can find a unimodular matrix U such that AU is in Hermite normal form, say $AU = [H\ 0]$. If C is the set of width directions of the parameterised polyhedron

$$\{y \in \mathbb{R}^n : [H\ 0]\, y \leqslant b\},$$

then $\{cU^{-1} : c \in C\}$ is the set of width directions of the parameterised polyhedron P_b, since for any vector c, we have

$$\begin{aligned}\max\{cx : Ax \leqslant b\} &= \max\{cUU^{-1}x : AUU^{-1}x \leqslant b\}\\ &= \max\{c'y : [H\ 0]\, y \leqslant b\},\end{aligned}$$

where $c' := cU$, and the same line for minima. The latter optimum is infinite if $c'_i \neq 0$ for some i, $r < i \leqslant n$; therefore, we can assume that all width directions c' satisfy

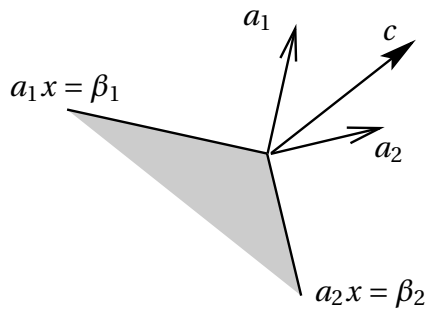

Figure 3.3: Bounded linear program.

$c'_i = 0$ for all $i = r+1, \ldots, n$. But then the problem is equivalent to that of finding width directions of the parameterised polyhedron

$$P'_b := \{y \in \mathbb{R}^r : Hy \leqslant b\},$$

which is defined by the matrix H of full column rank.

By Theorem 2.5, any feasible linear program

$$\max\{cx : Ax \leqslant b\} \tag{3.5}$$

is bounded if and only if there exists a feasible solution for the dual program

$$\min\{yb : yA = c,\ y \geqslant 0\}.$$

Furthermore, if the dual program is feasible, then there exists a basic feasible solution to it, i.e., a basis $N \subseteq \{1, \ldots, m\}$ of A such that $c = y_N A_N$ for some vector $y \geqslant 0$, see Theorem 2.6. In other words, the vector c belongs to the cone generated by the rows of matrix A_N, as in Figure 3.3. This simple observation yields the following lemma.

Lemma 3.5. *Let P be a parameterised polyhedron defined by a rational matrix A. If $P_{b'}$ has infinite lattice width for some b', then P_b has infinite lattice width for all b.*

Proof. Suppose that the lattice width of P_b is finite for some b and let $c \neq 0$ be a width direction. Then both linear programs

$$\max\{cx : Ax \leqslant b\} \quad \text{and} \quad \min\{cx : Ax \leqslant b\}$$

are bounded, and therefore, there exist bases N_1 and N_2 of A such that c belongs to the cones

$$C_1 = \{yA_{N_1} : y \geqslant 0\} \quad \text{and} \quad C_2 = \{-yA_{N_2} : y \geqslant 0\}$$

generated by the rows of matrices A_{N_1} and $-A_{N_2}$, respectively. But then the linear programs

$$\max\{cx : Ax \leqslant b'\} \quad \text{and} \quad \min\{cx : Ax \leqslant b'\}$$

must also be bounded, whence $w_c(P_{b'})$ is finite. □

The above lemma shows that finite lattice width is a property of the matrix A. In particular P_0 has finite lattice width if and only if P_b has finite lattice width for all b, and if P_0 has infinite lattice width, then P_b contains an integral vector for all b. Since we can easily recognise whether P_0 has infinite lattice width, we shall further consider only those parameterised polyhedra, for which $w(P_0)$ is finite, and therefore, $w(P_b)$ is finite for any b. We say in this case that the parameterised polyhedron P, defined by A, has *finite lattice width*.

Once we agreed that A is of full column rank, each basis $N \subseteq \{1,\dots,m\}$ of A uniquely defines the corresponding basic solution as a rational linear transformation of the right-hand side b:

$$F_N : \mathbb{R}^m \to \mathbb{R}^n, \quad F_N b = A_N^{-1} b_N. \tag{3.6}$$

Moreover, if the optimum (3.5) if finite, then there exists a basis N such that the optimum value is attained by vector $F_N b$ (again, Theorem 2.6).

Now, suppose that c is a width direction of P_b. Then there exist two bases N_1 and N_2 such that

$$\max\{cx : Ax \leqslant b\} = cF_{N_1} b \quad \text{and} \quad \min\{cx : Ax \leqslant b\} = cF_{N_2} b \tag{3.7}$$

and c belongs to both cones

$$C_1 := \{yA_{N_1} : y \geqslant 0\} \quad \text{and} \quad C_2 = \{-yA_{N_1} : y \geqslant 0\}$$

generated by the rows of the matrices A_{N_1} and $-A_{N_2}$, respectively. In fact, equations (3.7) hold for any vector c from $C_1 \cap C_2$. Thus, the lattice width of P_b is equal to the optimum value of the following optimisation problem:

$$\min\{c(F_{N_1} - F_{N_2})b : c \in C_1 \cap C_2 \cap \mathbb{Z}^n \setminus \{0\}\}, \tag{3.8}$$

while an optimum solution provides a width direction. The latter is an integer programming problem, since the (simplicial) cones C_1 and C_2 can be represented by some systems of inequalities, $cD_1 \leqslant 0$ and $cD_2 \leqslant 0$, respectively, while the origin can be cut off by a single inequality, for example, $cD_1\mathbf{1} \leqslant -1$, where $\mathbf{1}$ denotes the n-

dimensional all-one vector. Hence, the optimum value of (3.8) is attained at some vertex of the integer hull of the pointed polyhedron

$$\{c : cD_1 \leqslant 0,\ cD_2 \leqslant 0,\ cD_1\mathbf{1} \leqslant -1\}. \tag{3.9}$$

For fixed n, the number of these vertices is polynomial in the input size and they all can be computed in polynomial time, see Theorem 2.10. This is summarised in the following lemma.

Lemma 3.6. *There exists an algorithm that takes as input a rational matrix $A \in \mathbb{Q}^{m\times n}$ of full column rank, defining a parameterised polyhedron P of finite lattice width, and computes a set of triples (F_i, G_i, c_i) of linear transformations $F_i, G_i : \mathbb{R}^m \to \mathbb{R}^n$ and a non-zero row-vector $c_i \in \mathbb{Z}^n$, $i = 1,\dots,M$, such that for all b, for which P_b is non-empty,*

(a) F_i and G_i provide, respectively, an upper and lower bound on the value of the linear function $c_i x$ in P_b, i.e., for all i,

$$c_i G_i b \leqslant \min\{c_i x : x \in P_b\} \leqslant \max\{c_i x : x \in P_b\} \leqslant c_i F_i b,$$

(b) the lattice width of P_b is attained along the direction c_i for some $i \in \{1,\dots,M\}$ and can be expressed as

$$w(P_b) = \min_i c_i(F_i - G_i)b.$$

The number M satisfies the bound

$$M \leqslant m^n(2n+1)^n(24n^5\phi)^{n-1}, \tag{3.10}$$

where ϕ is the maximum size of a column in A. The algorithm runs in polynomial time if n is fixed.

Proof. In the first step of the algorithm we enumerate all possible bases of A; since A is of full column rank, there exists at least one basis, but the total number of possible bases is at most $m^n/2$. The algorithm iterates over all unordered pairs of bases, and for each such a pair $\{N_1, N_2\}$ does the following.

Let

$$C_1 = \{yA_{N_1} : y \geqslant 0\} \quad \text{and} \quad C_2 = \{-yA_{N_2} : y \geqslant 0\}$$

be the simplicial cones generated by the rows of matrices A_{N_1} and $-A_{N_2}$ respectively. By Lemma 2.3, these cones can be represented by systems of linear inequalities,

$cD_1 \leqslant 0$ and $cD_2 \leqslant 0$, respectively, each of which consists of n inequalities and the size of each inequality is bounded by $4n^2\phi$. As the cone $C_1 \cap C_2$ is pointed, the origin can be cut off by a single inequality; for example,

$$cD_1\mathbf{1} \leqslant -1,$$

where $\mathbf{1}$ stands for the n-dimensional all-one vector. The size of the latter inequality is bounded by $4n^3\phi$.

All together, there are $2n+1$ inequalities in the integer program (3.9) and the size of each is bounded by $4n^3\phi$. This implies that the number of vertices of the integer hull of (3.9) is at most $2\big(2n+1\big)^n\big(24n^5\phi\big)^{n-1}$, see Theorem 2.10, and they all can be computed in polynomial time if n is fixed, by exploiting the algorithm for integer programming in fixed dimension; see Hartmann (1989) for details.

The algorithm then outputs the triple (F_{N_1}, F_{N_2}, c) for each vertex c of the integer hull of (3.9), where F_{N_1} and F_{N_2} are the linear transformations defined by (3.6).

Since there are at most $m^n/2$ unordered pairs of bases and, for each pair, the algorithm returns at most $2\big(2n+1\big)^n\big(24n^5\phi\big)^{n-1}$ triples, the total number of triples satisfies (3.10), as required. Both parts of the theorem follow directly from our previous explanation. □

The bound (3.10) can be rewritten for fixed n as

$$M = O\big(m^n\phi^{n-1}\big).$$

Clearly, the greatest common divisor of the components of any direction c_i obtained by the algorithm must be equal to 1, as otherwise it would not be a vertex of (3.9). This implies, in particular, that the Hermite normal form of any of these vectors is just the first unit vector e_1 in $\mathbb{R}^n$.

It is also worth mentioning that if (F_i, G_i, c_i) is a triple attaining the minimum in Part (b) of Lemma 3.6, then we have

$$w(P_b) \leqslant \max\big\{c_i x : x \in P_b\big\} - \min\big\{c_i x : x \in P_b\big\} \leqslant c_i F_i b - c_i G_i b = w(P_b),$$

hence Part (a) of the lemma, when applied to this triple, turns into the equations

$$\min\big\{c_i x : x \in P_b\big\} = c_i G_i b \quad \text{and} \quad \max\big\{c_i x : x \in P_b\big\} = c_i F_i b.$$

For our further purposes, however, it is more suitable to have a *unique* width direction for all polyhedra P_b with varying b. In fact, using Lemma 3.6, we can partition the set of the right-hand sides into a number of partially open polyhedra, such that the width direction remains the same for all b belonging to the same region of

the partition. To see this, observe that the sets Q_i $(i = 1,\dots,M)$ defined by the inequalities

$$c_i(F_i - G_i)b \leqslant c_j(F_j - G_j)b \quad \text{for all } j \neq i,$$

where the triples (c_i, F_i, G_i) are those returned by the algorithm of Lemma 3.6, are polyhedra (even polyhedral cones) over b. Then Part (b) of Lemma 3.6 implies that

$$w(P_b) = \min_j c_j(F_j - G_j)b = c_i(F_i - G_i)b$$

for all $b \in Q_i$. Moreover, the polyhedra $Q_i \cap Q$ $(i = 1,\dots,M)$ is "almost" a partition of Q, in a sense that they can intersect only on the hyper-planes

$$\{b : c_i(F_i - G_i)b = c_j(F_j - G_j)b\},$$

i.e., the boundaries of the polyhedra Q_i. Thus, the only thing we need is to exclude such a boundary from all but one polyhedra. We remark that this was essentially the main reason for introducing the notion of partially open polyhedra in Section 2.4, which is not very common in the literature related to linear and integer programming.

Theorem 3.7. *There exists an algorithm that, given a rational matrix $A \in \mathbb{Q}^{m\times n}$ of full column rank, defining a parameterised polyhedron P of finite lattice width, and a rational partially open polyhedron*[1] *$Q \subseteq \mathbb{R}^m$ such that P_b is non-empty for all $b \in Q$, partitions Q into a number of partially open polyhedra $Q_1,\dots,Q_M$ and finds, for each i, a triple (F_i, G_i, c_i) of linear transformations $F_i, G_i : \mathbb{R}^m \to \mathbb{R}^n$ and a non-zero row-vector $c_i \in \mathbb{Z}^n$, such that*

$$\min\{c_i x : x \in P_b\} = c_i G_i b, \quad \max\{c_i x : x \in P_b\} = c_i F_i b,$$

and

$$w(P_b) = w_{c_i}(P_b) = c_i(F_i - G_i)b$$

for all $b \in Q_i$. If n is fixed, the algorithm runs in polynomial time and

$$M = O\big(m^n \phi^{n-1}\big),$$

where ϕ is the maximum size of a column in A.

[1] Here and in what follows, the phrase "given a polyhedron" means that we are given a system of linear inequalities defining the polyhedron. Particularly, the size of the polyhedron is the size of this system of linear inequalities.

Proof. First, we exploit the algorithm of Lemma 3.7 to obtain the triples (F_i, G_i, c_i), $i = 1, \ldots, M$, with $M = O\big(m^n \phi^{n-1}\big)$, providing us the width directions of the parameterised polyhedron P. For each $i = 1, \ldots, M$, we define a partially open polyhedron Q_i by the inequalities

$$\begin{aligned} c_i(F_i - G_i)b &< c_j(F_j - G_j)b, \quad j = 1, \ldots, i-1, \\ c_i(F_i - G_i)b &\leqslant c_j(F_j - G_j)b, \quad j = i+1, \ldots, M. \end{aligned}$$

Thus,

$$\min_j c_j(F_j - G_j)b = c_i(F_i - G_i)b$$

for all $b \in Q_i$. We claim that the intersections of the partially open polyhedra Q_i with Q give the required partition.

Indeed, let $b \in Q$ and let μ be the minimal value of $c_i(F_i - G_i)b$, $i = 1, \ldots, M$. Let I denote the set of indices i with $c_i(F_i - G_i)b = \mu$. Then $b \in Q_{i_0}$, where i_0 is the smallest index in I. Yet, suppose that $b \in Q$ belongs to two partially open polyhedra, say Q_i and Q_j. Without loss of generality, we can assume $i < j$. But then we have

$$c_i(F_i - G_i)b \leqslant c_j(F_j - G_j)b < c_i(F_i - G_i)b,$$

where the first inequality is due to the fact $b \in Q_i$ and the second inequality follows from $b \in Q_j$; both together are a contradiction.

For the width directions, Lemma 3.6 implies that

$$w(P_b) = \min_j c_j(F_j - G_j)b = c_i(F_i - G_i)b$$

for all $b \in Q_i \cap Q$. This completes the proof. □

The result analogous to Theorem 3.7 first appeared in Kannan (1992). However, the bound on the number of regions in the partition was exponential not only in n, but also in the affine dimension of Q, $\dim(Q)$. Consequently, the algorithm to compute this partition ran in time exponential in $\dim(Q)$. Actually, this is exactly the point, where the algorithm of Kannan (1990) for parameterised integer programming became exponential in $\dim(Q)$.

Another improvement over the result of Kannan — although not so important from the algorithmic point of view — is that we compute the *exact* width directions of a parameterised polyhedron P, while Kannan's algorithm outputs for each region Q_i, a direction $c_i \in \mathbb{Z}^n$ such that for all $b \in Q_i$,

$$\text{either } w_c(P_b) \leqslant 1, \quad \text{or } w_c(P_b) \leqslant 2w(P_b);$$

in other words, it computes *approximate* width directions of P_b.

3.3 Integer projections

Notice that the algorithm of Theorem 3.7 not only delivers, for each region Q_i of the partition, a width direction of the polyhedron P_b, but also expresses the lattice width $w(P_b)$ as a linear transformation of b. Furthermore, looking back to the Lenstra's algorithm for integer programming in fixed dimension (Section 3.1), we can see that the value (3.4) is also defined as a linear transformation of b; namely,

$$\beta = c_i G_i b$$

for each $b \in Q_i$. Following the Lenstra's algorithm, we can now proceed with a recursion call, as P_b contains an integral vector if and only if its intersection with the hyper-plane

$$\{x : c_i x = \lceil \beta \rceil + j\}$$

contains an integral vector for some $j \in \{0, \dots, \omega(n)\}$. With b varying, these hyper-planes take the form

$$\{x : c_i x = \lceil c_i G_i b \rceil + j\}$$

and, while still being hyper-planes in variables x, are no more polyhedra in the space of variables (x, b), i.e,

$$\{(x, b) : c_i x = \lceil c_i G_i b \rceil + j\} \tag{3.11}$$

are *not* polyhedra.

Nevertheless, we can still express these sets in the form suitable for our purposes. Indeed, the set (3.11) is, in fact, the projection of the set of all feasible solutions of the *mixed*-integer linear program

$$c_i x = z + j, \quad c_i G_i b \leqslant z < c_i G_i b + 1, \quad z \in \mathbb{Z}$$

onto the space of variables (x, b).

More generally, we define the *integer projection* $W/\mathbb{Z}^l$ of a set $W \subseteq \mathbb{R}^{n+l}$ as

$$W/\mathbb{Z}^l := \{x \in \mathbb{R}^n : (x, z) \in W \text{ for some } z \in \mathbb{Z}^l\}.$$

In words, it is a set of vectors x, for which there exists an integral vector $z \in \mathbb{Z}^l$ such that (x, z) belongs to W. We shall mostly deal with integer projections of rational partially open polyhedra.

It is easy to see that the integer projection $P/\mathbb{Z}^l$ of any polyhedron $P \subseteq \mathbb{R}^{n+l}$ is just the union of (possibly infinite number of) partially open polyhedra in $\mathbb{R}^n$. Furthermore, for any polyhedron P we have

$$P = P/\mathbb{Z}^0, \quad P \cap \mathbb{Z}^n = \{(x, x) \in \mathbb{R}^n \times \mathbb{R}^n : x \in P\}/\mathbb{Z}^n. \tag{3.12}$$

Thus, integer projections of polyhedra can be viewed as a generalisation of both polyhedra and integral vectors in polyhedra.

Let P be a partially open polyhedron in $\mathbb{R}^{n+l}$. If l is fixed, we can apply the Lenstra's algorithm for mixed-integer programming to check whether $P/\mathbb{Z}^l$ is empty, and if not, to find a vector x in $P/\mathbb{Z}^l$. Indeed, $P/\mathbb{Z}^l$ is empty if and only if the mixed-integer program

$$(x,z) \in P, \quad x \in \mathbb{R}^n, \quad z \in \mathbb{Z}^l$$

is infeasible. Yet, we can optimise a linear function over a set $P/\mathbb{Z}^l$ in polynomial time, if l is fixed. Barvinok and Woods (2003) studied the set of integral vectors in $P/\mathbb{Z}^l$, assuming n and l to be fixed. They were able to develop an algorithm, which computes a short rational generating function for $(P/\mathbb{Z}^l) \cap \mathbb{Z}^n$. As a consequence, it is possible, for example, to compute the number of integral vectors in $P/\mathbb{Z}^l$. We remark that the algorithm of Barvinok and Woods (2003) combines the approach of Barvinok (1994) with the result of Kannan (1992) on width directions of a parameterised polyhedron, which we have improved in the previous section.

In our proof of the structural theorem we must be able to check feasibility for an integer projection $P/\mathbb{Z}^l$. As we have already mentioned, it can be done with the Lenstra's algorithm, if l is bounded by a constant. Therefore, we must carefully estimate the growth of l in the algorithm. The following lemma, although very simple, is helpful for this.

Lemma 3.8. *Let $V, W \subseteq \mathbb{R}^n$ be integer projections of some partially open polyhedra in $\mathbb{R}^{n+l}$ and $\mathbb{R}^{n+k}$, respectively. Then $V \cap W$ is the integer projection of a partially open polyhedron in $\mathbb{R}^{n+l+k}$.*

Proof. Let $V = P_1/\mathbb{Z}^l$ and $W = P_2/\mathbb{Z}^k$, where $P_1 \subseteq \mathbb{R}^{n+l}$ and $P_2 \subseteq \mathbb{R}^{n+k}$ are partially open polyhedra. A vector $x \in \mathbb{R}^n$ belongs to the intersection $V \cap W$ if and only if there exist integral vectors $z_1 \in \mathbb{Z}^l$ and $z_2 \in \mathbb{Z}^k$ such that $(x, z_1) \in P_1$ and $(x, z_2) \in P_2$. Equivalently, there exists an integral vector $(z_1, z_2) \in \mathbb{Z}^{l+k}$ such that the vector (x, z_1, z_2) belongs to the polyhedron defined by

$$(x, z_1) \in P_1 \quad \text{and} \quad (x, z_2) \in P_2.$$

This completes the proof. □

3.4 Structural theorem

Now, we have all necessary tools to establish the main result of this chapter — the structural theorem. We must remark that the theorem itself, as well as its proof, is

mostly taken from Kannan (1992). However, our stronger result on width directions of a parameterised polyhedron (see Section 3.2) leads to the stronger result in this structural theorem: again, the parameter b is allowed to vary over a polyhedron of a variable dimension.

Theorem 3.9. *There exists an algorithm that, given a rational matrix $A \in \mathbb{Q}^{m\times n}$ of full column rank, defining a parameterised polyhedron P of finite lattice width, and a rational partially open polyhedron $Q \subseteq \mathbb{R}^m$ such that P_b is non-empty for all $b \in Q$, computes a partition of Q into sets $S_1, \dots, S_M$, each being the integer projection of a partially open polyhedron, $S_i = S_i'/\mathbb{Z}^{l_i}$, and finds, for each i, a number of unimodular transformations $U_{ij} : \mathbb{R}^n \to \mathbb{R}^n$ and affine transformations $T_{ij} : \mathbb{R}^m \to \mathbb{R}^n$, $j = 1, \dots, K_i$, such that, for any $b \in S_i$, $P_b \cap \mathbb{Z}^n \neq \emptyset$ if and only if P_b contains $U_{ij}\lceil T_{ij}(b)\rceil$ for some index j.*

If n is fixed, then the algorithm runs in polynomial time and the following bounds hold:

$$M = O\big((m^n\phi^{n-1})^{n\chi(n)}\big), \quad l_i = O\big(\chi(n)\big), \quad K_i = O\big(2^{n^2/2}\chi(n)\big), \quad i = 1, \dots, M,$$

where ϕ denotes the maximum size of a column in A and

$$\chi(n) = \prod_{i=1}^{n} \omega(n)$$

is a constant.

Before we present the proof of this theorem, we informally discuss why it is useful. Suppose n is fixed and we want to decide whether there exists a $b \in Q$ such that the system $Ax \leqslant b$ has no integral solution. The algorithm of Theorem 3.9 returns us a partition of Q into polynomially many sets S_i, each being the integer projection of some partially open polyhedron $S_i' \subseteq \mathbb{R}^{n+l_i}$. If n is fixed, then l_i is bounded by a constant, see the statement of Theorem 3.9. This means that S_i can be modelled in an extended space as the solutions of a mixed-integer program with a fixed number of integer variables. Then the theorem states further that in order to find an integral vector in P_b, we need to consider K_i (a *fixed* number of) candidate solutions

$$x_{ij} = U_{ij}\lceil T_{ij}(b)\rceil.$$

Notice that each of these candidate solutions for a given b can be modelled with a fixed number of integer variables too, as $T_{ij}(b) \in \mathbb{R}^n$ and n is fixed. We want to check whether each of these candidate solutions does not satisfy $Ax \leqslant b$. In this case, each of the candidate solutions violates at least one constraint of $Ax \leqslant b$. Since the number of candidate solutions K_i is fixed, we can check the $\binom{m}{K_i}$ many ways, in which

a candidate solution is associated with a constraint to be violated. All together we can answer the question whether there exists a $b \in Q$ with $Ax \leqslant b$ having no integral solution, by solving a *polynomial number* of mixed-integer programs with a fixed number of integer variables. In Chapter 4 we describe this again in more detail and in a slightly more general form.

Proof of Theorem 3.9. The proof is by induction on n. First, suppose that $n = 1$. The algorithm of Theorem 3.7 partitions Q into $M = O(m)$ partially open polyhedra $Q_1, \ldots, Q_M$ and computes, for each $i = 1, \ldots, M$, a triple (F_i, G_i, c_i) such that

$$\min\{c_i x : x \in P_b\} = c_i G_i b \quad \text{and} \quad \max\{c_i x : x \in P_b\} = c_i F_i b.$$

Notice that c_i is necessarily 1. Thus, we set $S_i = Q_i$ and assign to each S_i one transformation $T_{i1} : \mathbb{R}^m \to \mathbb{R}$ defined by $T_{i1}(b) = G_i b$. Indeed, for any $b \in S_i$, the polyhedron P_b contains an integral vector if and only if $\lceil G_i b \rceil$ is contained in P_b. The unimodular transformation U_{i1} is just the identity.

Now, we consider a general n. Again, by applying the algorithm of Theorem 3.7, we obtain a partition of Q into partially open polyhedra Q_i and the corresponding triples (F_i, G_i, c_i) such that

$$w(P_b) = c_i(F_i - G_i)b$$

for all $b \in Q_i$; moreover, $c_i G_i b$ gives the minimal value of the linear function $c_i x$ in P_b, while $c_i F_i b$ is its maximum value. Further on, we restrict our attention onto one particular cell of this partition, and (we hope it does not confuse the reader) denote it by Q. Let (F, G, c) be the corresponding triple. After applying an appropriate unimodular transformation U, we may assume that c is the first unit vector e_1. This is feasible, since we can transform the candidate solutions $U_{ij}\lceil T_{ij}(b) \rceil$ back with the inverse of this unimodular transformation from the left: indeed, $Ax \leqslant b$ if and only if $AU(U^{-1}x) \leqslant b$, and x is integral if and only if $U^{-1}x$ is integral.

Let P' denote the lower-dimensional parameterised polyhedron, derived from P by moving the variable x_1 to the right-hand side:

$$P'_{b - a_1 x_1} = \{x' : A'x' \leqslant b - a_1 x_1\},$$

where A' stands for the matrix A after removing the first column a_1, x_i is the i-th component of x and $x' = [x_2, \ldots, x_n]$. The polyhedron P_b contains an integral vector if and only if $P'_{b - a_1 x_1}$ contains an integral vector for some integral value of x_1. On the other hand, it follows from Lemma 3.4 that we need to consider only those values of x_1 that satisfy

$$e_1 Gb \leqslant x_1 \leqslant e_1 Gb + \omega(n).$$

As b varies over Q and x_1 varies from e_1Gb to $e_1Gb+\omega(n)$, the vector $b-a_1x_1$ varies over the polyhedron

$$Q' = \{b-a_1x_1 : b\in Q,\ e_1Gb \leqslant x_1 \leqslant e_1Gb+\omega(n),\ x_1 \leqslant e_1Fb\},$$

where the last inequality ensures that we do not leave the feasible region.

As P' has a smaller dimension than P, we can use the induction hypothesis to obtain a partition of Q' into sets $R_1,\dots,R_{M'}$, where $R_i = R_i'/\mathbb{Z}^{l_i}$ is an integer projections of a partially open polyhedron R_i' in $\mathbb{R}^{n+l_i}$ $(i=1,\dots,M')$, and the corresponding collections of unimodular transformations U_{ij} and affine transformations T_{ij} such that $P'_{b'}$, with $b'\in R_i$, contains an integral vector if and only if it contains $U_{ij}\lceil T_{ij}(b')\rceil$ for some j. By induction, we also have the bounds

$$M' = O\big((m^{n-1}\phi^{n-2})^{(n-1)\chi(n-1)}\big),\quad l_i = O(\chi(n-1)),\quad i=1,\dots,M',$$

and, for each i, the number of unimodular transformations U_{ij} and affine transformations T_{ij} is

$$O\big(2^{(n-1)^2/2}\chi(n-1)\big).$$

Recall that we are interested in integral vectors of the polyhedra $P'_{b-a_1x_1}$ for at most $\omega(n)+1$ different values of x_1, namely,

$$x_1 = \lceil e_1Gb\rceil + j,\quad j=0,\dots,\omega(n).$$

Consequently, we need to consider transformations U_{ij} and T_{ij} corresponding to these particular values of x_1. However, the vectors

$$b-\big(\lceil e_1Gb\rceil + j\big)a_1,\quad j=0,\dots,\omega(n),$$

may happen to lie in different parts of the partition of Q'. We define our partition as follows. For every ordered tuple $I=\langle i_0,\dots,i_{|I|-1}\rangle$ of at most $\omega(n)+1$ indices from $\{1,\dots,M'\}$, we define S_I as the set of all $b\in Q$ such that

$$\begin{aligned} &b-\big(\lceil e_1Gb\rceil + j\big)a_1 \in R_{i_j},\quad && j=0,\dots,|I|-1\\ &e_1Gb+j > e_1Fb, && j\geqslant |I|. \end{aligned}$$

The second constraint is equivalent to $b-\big(\lceil e_1Gb\rceil + j\big)a_1 \notin Q'$.

These sets S_I are integer projections of some higher-dimensional partially open polyhedra, $S_I = S_I'/\mathbb{Z}^{l_I}$. Indeed, $\lceil e_1Gb\rceil$ can be expressed as an integer variable z_0 satisfying the constraint

$$e_1Gb \leqslant z_0 < e_1Gb+1,$$

while each of the conditions $b - \big(\lceil e_1 Gb \rceil + j\big) a_1 \in R_{i_j}$ is equivalent to

$$(b, z_0, z) \in R'_{i_j}$$

for some integral vector $z \in \mathbb{Z}^{l_{i_j}}$. By Lemma 3.8, we also have

$$l_I \leqslant 1 + (\omega(n) + 1) O(\chi(n-1)) = O(\chi(n)).$$

The number of the regions is roughly

$$O\big((m^n \phi^{n-1})\big((m^{n-1}\phi^{n-2})^{(n-1)\chi(n-1)}\big)^{\omega(n)}\big) = O\big((m^n \phi^{n-1})^{n\chi(n)}\big).$$

At last, they do form a partition of Q. Indeed, for each $b \in Q$, its translate

$$b - \big(\lceil e_1 Gb \rceil + j\big) a_1$$

belongs to Q' and lies in some set R_i, unless

$$e_1 Gb + j > e_1 Fb \quad \text{or} \quad j > \omega(n).$$

Consequently, there exists a tuple I such that $b \in S_I$. Similarly, b cannot lie in several sets S_I, as in this case

$$b - \big(\lceil e_1 Gb \rceil + j\big) a_1,$$

for some j, would belong to several sets R_i, which is impossible, since the R_i formed a partition of Q'.

Now, we need to construct the appropriate transformations for each set S_I. Let $I = \langle i_0, \ldots, i_{N-1} \rangle$ and let S_I be the corresponding set in our partition. Let $b \in S_I$. If P_b contains an integral vector x, then it contains one with

$$x_1 = \lceil e_1 Gb \rceil + j \quad \text{for some } j = 0, \ldots, N-1.$$

For this x_1, the polyhedron $P'_{b - a_1 x_1}$ contains an integral vector x', defined by

$$x' = U_{i_j k} \big\lceil T_{i_j k}(b - a_1 x_1) \big\rceil$$

for some index k. Equivalently,

$$U_{i_j k}^{-1} x' = \big\lceil T_{i_j k}(b) - T_{i_j k}(a_1) x_1 \big\rceil.$$

To prove the induction step, we need to move x_1 to the left-hand side of the above equation. First, since x_1 is an integer, the product $\lfloor T_{i_j k}(a_1) \rfloor x_1$ is also an integer and rounding will not affect it. Hence, we get

$$\big\lfloor T_{i_j k}(a_1) \big\rfloor x_1 + U_{i_j k}^{-1} x' = \big\lceil T_{i_j k}(b) - \big\{T_{i_j k}(a_1)\big\} x_1 \big\rceil,$$

where $\{T_{i_jk}(a_1)\}$ denotes the fractional part of the vector $T_{i_jk}(a_1)$. Recall that we consider

$$x_1 = \lceil e_1 Gb \rceil + j. \tag{3.13}$$

Therefore,

$$\lfloor T_{i_jk}(a_1) \rfloor x_1 + U_{i_jk}^{-1} x' = \lceil T_{i_jk}(b) - (e_1 Gb + j)\{T_{i_jk}(a_1)\} - \gamma\{T_{i_jk}(a_1)\} \rceil \tag{3.14}$$

for some $0 \leqslant \gamma < 1$. Observe that $T_{i_jk}(b) - (e_1 Gb + j)\{T_{i_jk}(a_1)\}$ is an affine transformation of b only; let us denote it by $T_{Ijk}(b)$. Furthermore, each component of the vector $\gamma\{T_{i_jk}(a_1)\}$ lies between 0 and 1; thus each component of the right-hand side vector in (3.14) can actually take only two values. Therefore, we can try all possibilities; namely, we have to consider equations

$$\lfloor T_{i_jk}(a_1) \rfloor x_1 + U_{i_jk}^{-1} x' = \lceil T_{Ijk}(b) - v \rceil, \tag{3.15}$$

where $v \in \mathbb{Z}^{n-1}$ satisfies the bounds $0 \leqslant v \leqslant 1$. Obviously, there are only 2^{n-1} such vectors, which is a constant if n is fixed. The right-hand side in these equations depends now only on b. Combining them with (3.13), we obtain the required formula for an integral vector in P_b, since the transformation

$$U_{Ijk}^{-1} = \begin{bmatrix} 1 & 0 \\ \lfloor T_{i_jk}(a_1) \rfloor & U_{i_jk}^{-1} \end{bmatrix}$$

is obviously unimodular.

The above construction must be repeated for all $j = 0, \ldots, N-1$ and all indices k, thus we obtain at most

$$O\big(2^{(n-1)^2/2}\chi(n-1)\big)\big(\omega(n)+1\big)2^{n-1} = O\big(2^{n^2/2}\chi(n)\big)$$

pairs of transformations for each set S_I. As explained earlier, if there is an integral vector in P_b, then there is one satisfying (3.13), for some j, and (3.14), for some k, which is equivalent to (3.15), for some v. This completes the proof. □

Chapter 4

Application of the Structural Theorem

In the previous chapter, we defined a parameterised integer linear programming problem as follows: Given a rational matrix $A \in \mathbb{Q}^{m\times n}$ and a partially open polyhedron $Q \subseteq \mathbb{R}^m$, decide

$$\forall b \in Q \quad \exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \ ? \tag{4.1}$$

Equivalently, we can ask if there exists $b \in Q$ such that the system $Ax \leqslant b$ has no integral solution. The structural theorem proved in Section 3.4 provides a tool for solving parameterised integer programming problems when n is fixed. In fact, we can do even more. But before we state the generalisation of the problem, let us consider the following question:

$$\exists b \in Q \quad \exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \ ? \tag{4.2}$$

Clearly, this problem is polynomial-time solvable when n is fixed, since it is just a mixed-integer programming problem with a fixed number of integer variables:

$$\exists (x,b) \in \mathbb{Z}^n \times \mathbb{R}^m : \quad Ax - b \leqslant 0, \quad b \in Q \ ?$$

Our general question combines (4.1) and (4.2): now, we are given two systems of linear inequalities, $Ax \leqslant \Phi(b)$ and $Bx \leqslant \Psi(b)$, with the right-hand sides depending on the parameter b via rational affine transformations Φ and Ψ, and the question is to *find* $b \in Q$ such that $Ax \leqslant \Phi(b)$ has an integral solution, while the system $Bx \leqslant \Psi(b)$ is infeasible in integer variables.[1]

[1] The affine transformations Φ and Ψ were introduced into the statement to treat systems with different number of inequalities (the systems $Ax \leqslant b$ and $Bx \leqslant b$ have the same number of inequalities).

4.1 Parameterised integer programming

Let $A \in \mathbb{Q}^{m\times n}$ and $B \in \mathbb{Q}^{k\times n}$ be rational matrices and let $\Phi : \mathbb{R}^l \to \mathbb{R}^m$ and $\Psi : \mathbb{R}^l \to \mathbb{R}^k$ be rational affine transformations. Yet, let $Q \subseteq \mathbb{R}^m$ be a partially open polyhedron. We aim to find a vector $b \in Q$ such that the system $Ax \leqslant \Phi(b)$ has an integral solution, but the system $Bx \leqslant \Psi(b)$ has no integral solution.

The idea of the algorithm is now simple: First we run the algorithm of Theorem 3.9 on input B and $\Psi(Q)$, the image of Q with respect to the transformation Ψ. Then we consider each set S_i returned by the algorithm of Theorem 3.9 independently. For each b such that $\Psi(b) \in S_i$ we have a fixed number of candidate solutions for the system $Bx \leqslant \Psi(b)$, defined via unimodular and affine transformations as $U_{ij}\lceil T_{ij}(\Psi(b))\rceil$. Each rounding operation can be expressed by introducing an integral vector: $z = \lceil T_{ij}(\Psi(b))\rceil$ is equivalent to

$$T_{ij}(\Psi(b)) \leqslant z < T_{ij}(\Psi(b)) + \mathbf{1},$$

where $\mathbf{1}$ denotes the all-one vector. We need only a constant number of integer variables to express all candidate solutions plus a fixed number of integer variables to represent the integer projections $S_i = S_i'/\mathbb{Z}^{l_i}$. It remains to solve a number of mixed-integer programs, to which we also include the constraints $Ax \leqslant \Phi(b)$, in order to check whether there exists $b \in Q$ such that all candidate solutions z violate $Bz \leqslant \Psi(b)$, while the system $Ax \leqslant \Phi(b)$ has an integral solution.

Theorem 4.1. *There exists an algorithm that, given rational matrices $A \in \mathbb{Q}^{m\times n}$ and $B \in \mathbb{Q}^{k\times n}$, rational affine transformations $\Phi : \mathbb{R}^l \to \mathbb{R}^m$ and $\Psi : \mathbb{R}^l \to \mathbb{R}^k$ and a rational polyhedron $Q \subseteq \mathbb{R}^l$, finds $b \in Q$ such that the system $Ax \leqslant \Phi(b)$ has an integral solution, but the system $Bx \leqslant \Psi(b)$ has no integral solution, or asserts that no such b exists. The algorithm runs in polynomial time if n is fixed.*

Proof. Let $r := \operatorname{rank}(B)$. We can compute in polynomial time a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $BU = \begin{bmatrix} B' 0 \end{bmatrix}$ is the Hermite normal form of B, where B' has full column rank; see Kannan and Bachem (1979). Then the system $Bx \leqslant \Psi(b)$ has an integral solution if and only if the system $B'y \leqslant \Psi(b)$ has an integral solution $y \in \mathbb{Z}^r$.

Let P be a parameterised polyhedron defined by the matrix B'. The system $B'y \leqslant \Psi(b)$ has no integral solution if and only if $P_{\Psi(b)} \cap \mathbb{Z}^n$ is empty. First, we exploit the Fourier–Motzkin elimination procedure to construct the polyhedron $Q' \subseteq \mathbb{R}^k$ of the right-hand sides b', for which the system $B'y \leqslant b'$ has a fractional solution. For each inequality $ab' \leqslant \beta$, defining the polyhedron Q', we can solve the following mixed-integer program

$$b \in Q, \quad a\Psi(b) > \beta, \quad Ax - \Phi(b) \leqslant 0, \quad x \in \mathbb{Z}^n,$$

and if any of these problems has a feasible solution (x, b), then b is the answer to the original problem. Hence, we can terminate and output this b.

Now, we can assume that for all $b \in Q$ the system $B'y \leqslant \Psi(b)$ has a fractional solution. By applying the algorithm of Theorem 3.9, we construct a partition of $\Psi(Q)$ into the sets $S_1, \ldots, S_M$, where each S_i is the integer projection of a partially open polyhedron, $S_i = S_i'/\mathbb{Z}^{l_i}$. Since n is fixed, the numbers l_i $(i = 1, \ldots, M)$ are bounded by some constant. Furthermore, for each i, the algorithm constructs unimodular transformations U_{ij} and affine transformations T_{ij}, $j = 1, \ldots, K$, such that $P_{\Psi(b)}$, with $\Psi(b) \in S_i$, contains an integral vector if and only if $U_{ij}\lceil T_{ij}\Psi(b)\rceil \in P_b$ for some j. Again, K is fixed for a fixed n.

The algorithm will consider each index i independently. For a given i, S_i can be described as the set of vectors b such that

$$(\Psi(b), z) \in S_i'$$

has a fractional solution for some integer $z \in \mathbb{Z}^{l_i}$, which can be expressed in terms of linear constraints, as S_i' is a partially open polyhedron. Let $y_j = U_{ij}\lceil T_{ij}\Psi(b)\rceil$. The vectors y_j can be described by linear inequalities as

$$T_{ij}\Psi(b) \leqslant z_j < T_{ij}\Psi(b) + \mathbf{1},$$

$$y_j = U_{ij}z_j,$$

where $\mathbf{1}$ is the all-one vector. Then b is a solution to our problem if and only if $y_j \notin P_{\Psi(b)}$ for all j. In this case, each y_j violates at least one constraint in the system $B'y \leqslant \Psi(b)$. We consider all possible tuples I of K constraints from $B'y \leqslant \Psi(b)$. Obviously, there are only m^K such tuples, that is, polynomially many in the input size. For each such a tuple, we solve the mixed-integer program

$$\begin{aligned}
&Ax \leqslant \Phi(b),\\
&(\Psi(b), z) \in S_i',\\
&T_{ij}\Psi(b) \leqslant z_j < T_{ij}\Psi(b) + \mathbf{1}, \quad j = 1, \ldots, K,\\
&y_j = U_{ij}z_j, \quad j = 1, \ldots, K,\\
&a_{i_j}y_j > \Psi_{i_j}(b), \quad j = 1, \ldots, K,
\end{aligned}$$

where $a_{i_j}y \leqslant \Psi_{i_j}(b)$ is the j-th constraint in the chosen tuple. Each such a mixed-integer program can be solved in polynomial time, since the number of integer variables is fixed (in fact, there are at most $(K+1)n + l_i$ integer variables).

If there exists a feasible solution b to one of these mixed-integer programs, this b is also a solution to our original problem, hence we terminate and output b. If all these mixed-integer programs are infeasible, there is no solution to the problem. □

The theorem analogous to Theorem 4.1 was proved by Kannan (1992). Our main contribution comparing to that theorem is that the numbers l, k, and m do not need to be fixed.

We remark that Theorem 4.1 allows to answer the question (4.1) in the case when Q is the set of *mixed-integral* vectors of a polyhedron, if the number of integer components is fixed. Indeed, for $Q \in \mathbb{R}^{k+m}$ we can ask whether there is a $b = (b_1, b_2) \in Q$, $b_1 \in \mathbb{R}^k$ and $b_2 \in \mathbb{R}^m$, such that the system $x = b_1$ has an integral solution, while the system $Ax \leqslant b$ is infeasible in integer variables. This question is then equivalent to testing

$$\forall b \in Q \cap (\mathbb{Z}^k \times \mathbb{R}^m) \quad \exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \ ?$$

for k and n fixed. The latter was also observed by Kannan (1992), but, again, his algorithm runs in polynomial-time only with an additional assumption that m is fixed.

4.2 Integer programming gaps

The algorithm described in Theorem 4.1 can be used to find the maximum difference between the optimum value of the integer linear program

$$\max\{cx : Ax \leqslant b,\ x \in \mathbb{Z}^n\}, \tag{4.3}$$

and its linear programming relaxation

$$\max\{cx : Ax \leqslant b\}. \tag{4.4}$$

Given a rational matrix $A \in \mathbb{Q}^{m\times n}$ and a rational vector $c \in \mathbb{R}^n$, we define

$$\delta(A, c) := \max_b \{\max\{cx : Ax \leqslant b\} - \max\{cx : Ax \leqslant b,\ x \in \mathbb{Z}^n\}\},$$

where the maximum is taken over all vectors $b \in \mathbb{R}^m$, for which (4.3) is feasible. We shall call $\delta(A, c)$ the *integer programming gap* for the family of integer programs (4.3).

Recently, Hoşten and Sturmfels (2003) established an algorithm to find the integer programming gap for a family of integer programs in standard form,

$$\min\{cx : Ax = b,\ x \geqslant 0\},$$

assuming the number of variables to be fixed. The latter, however, implies that the number of rows is fixed, as we can always assume A to be of full row rank, and the

problem fits into our settings. We develop an algorithm that, provided a matrix A and a vector b, finds the integer programming gap $\delta(A,c)$ and runs in polynomial time if the number of columns of A is fixed.

Consider the following system of inequalities:

$$\begin{aligned} cx &\geqslant \beta, \\ Ax &\leqslant b. \end{aligned}$$

Given a vector b and a number β, there exists a feasible solution of the above system if and only if the linear program (4.4) is feasible and its value is at least β. The set of pairs $(\beta, b) \in \mathbb{R}^{m+1}$, for which the above linear program has a solution, is a polyhedron in $\mathbb{R}^m$ and can be computed by means of Fourier–Motzkin elimination, in polynomial time if n if fixed. Let Q denote this polyhedron.

Suppose that we suspect the maximum integer programming gap to be smaller than γ. This means that, whenever β is an optimum value of (4.4), the integer program (4.3) must have a solution of value at least $\beta - \gamma$. Equivalently, the system

$$\begin{aligned} cx &\geqslant \beta - \gamma, \\ Ax &\leqslant b, \end{aligned} \tag{4.5}$$

must have an integral solution. If there exists $(b, \beta) \in Q$ such that (4.5) has no integral solution, the integer programming gap is bigger than γ. We also need to ensure that, for a given b, the integer program is feasible, i.e., the system $Ax \leqslant b$ has a solution in integer variables.

Now, this is exactly the question for the algorithm of Theorem 4.1: Is there a $(\beta, b) \in Q'$ such that the system (4.5) has no integral solution, but there exists $y \in \mathbb{Z}^n$ such that $Ay \leqslant b$? Here $Q' = Q - \gamma(1,0)$ is the appropriate translate of the set Q. If the algorithm returns some $(\beta - \gamma, b)$, then the integer program (4.3), with the right-hand side b, has no solution of value greater than $\beta - \gamma$, while being feasible. On the other hand, $(\beta, b) \in Q$, thus the corresponding linear solution has optimum value at least β. We can conclude that the maximum integer programming gap is greater than γ. This gives us the following theorem.

Theorem 4.2. *There exists an algorithm that, given a rational matrix $A \in \mathbb{R}^{m \times n}$, a rational row-vector $c \in \mathbb{Q}^n$ and a number γ, checks whether the maximum integer programming gap for the integer programs* (4.3) *defined by A and c is bigger than γ. The algorithm runs in polynomial time if the rank of A is fixed.*

Using binary search, we can also find the *minimum* possible value for γ, hence the maximum integer programming gap.

The algorithm can easily be modified to find an integer programming gap for the family (4.3) of integer programs, restricting the right-hand sides b to vary over some polyhedron in $\mathbb{R}^m$.

Chapter 5

Integer Programs in Standard Form

In this chapter we consider integer linear programs *in standard form*,

$$\min\{cx : Ax = b,\ x \in \mathbb{Z}_+^n\}, \tag{5.1}$$

where $A \in \mathbb{Q}^{d\times n}$ is a matrix, $c \in \mathbb{Q}^n$ is a row-vector and $b \in \mathbb{Q}^d$ is a column-vector. The problem (5.1) is polynomially equivalent to the following decision problem: Given a matrix A and a vector b, decide

$$\exists x \in \mathbb{Z}_+^n : \quad Ax = b\ ? \tag{5.2}$$

In other words, the question is to decide whether a given vector b is an integral non-negative combination of the columns of A. Clearly, we can assume without loss of generality that A, b and c are integral.

Given a finite set $X \subset \mathbb{R}^d$, we define the *integer cone* generated by X to be the set

$$\text{int.cone}(X) := \left\{ \sum_{i=1}^{t} \lambda_i x_i : t \geqslant 0;\ x_1,\dots,x_t \in X;\ \lambda_1,\dots,\lambda_t \in \mathbb{Z}_+ \right\}.$$

An example of the integer cone is shown on Figure 5.1: it is a discrete set of points inside the "rational" cone $\text{cone}(X)$. The set X is called a *Hilbert basis* if

$$\text{int.cone}(X) = \text{cone}(X) \cap \mathbb{Z}^d.$$

It is well-known (see Schrijver (1986)) that every rational polyhedral cone C is generated by an *integral* Hilbert basis, that is, a Hilbert basis consisting of integral vectors only. Moreover, if C is pointed, there is a unique minimal (with respect to inclusion) integral Hilbert basis generating C.

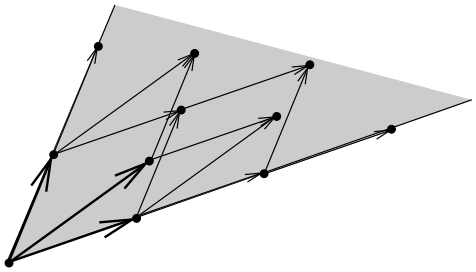

Figure 5.1: Integer cone.

Carathéodory's theorem (see Theorem 2.2) states that for a finite set $X \subset \mathbb{R}^d$ and a vector $b \in \text{cone}(X)$, there exists a subset $Y \subseteq X$ of linearly independent vectors such that $b \in \text{cone}(Y)$; in particular, $|Y| \leqslant d$. We consider the analogous question for integer cones:

> Given a finite set X of vectors in $\mathbb{Q}^d$ and a vector $b \in \text{int.cone}(X)$, how large is the smallest subset Y of X such that $b \in \text{int.cone}(Y)$?

Cook et al. (1986) showed that if X is an integral Hilbert basis and $\text{cone}(X)$ is pointed, then there exists a subset Y of X such that $b \in \text{int.cone}(Y)$ and $|Y| \leqslant 2d-1$. Later, Sebő (1990) improved this bound to $|Y| \leqslant 2d-2$. On the other hand, Bruns et al. (1999) showed that the bound d is not valid. A tight bound, however, is still an open question.

In Section 5.1 we shall investigate *general* integer cones generated by integral vectors. We show that the desired bound cannot be given in terms of d only, and propose a valid bound which is polynomial in d and the maximum size of a component among vectors in X. For a special case, when $\text{conv}(X) \cap \mathbb{Z}^d \subseteq X$, we prove the bound 2^d, which is exponential in d, but does not depend on the size of vectors in X.

In Section 5.2 we apply these results to find the number of non-zero components in an optimum solution of the integer program (5.1). It turns out that this number does not depend on the number of variables, which is especially useful for integer programs with exponentially many columns in the constraint matrix, since we can guarantee the existence of an optimum solution of polynomial size. One of the examples — the cutting stock problem — will be considered in Chapter 6.

Another interesting question concerns complexity of integer programs (5.1) under the assumption that the number d of the equality constraints is fixed. The algorithm of Lenstra is not applicable here, as the total number of constraints is not

fixed — we have to count non-negativity constraints, too. In fact, the problem remains NP-hard: the case $d = 1$ corresponds to the *knapsack problem*. However, Papadimitriou (1981) showed that there exists a *pseudo-polynomial* algorithm for the problems (5.1) when d is fixed, that is, an algorithm that runs in time polynomial in n and the maximum absolute value of the entries of A, b and c. Similar as in Chapter 4, we can ask whether the *integer programming gap* for the family of integer programs (5.1), with b varying over $\mathbb{Q}^d$, can also be found in pseudo-polynomial time.

It is worth mentioning that if the columns of the matrix

$$\begin{bmatrix} c & 1 \\ A & 0 \end{bmatrix}$$

form a Hilbert basis, the integer programming gap for (5.1) is less than 1: for any integral vector b, the optimum value of an integer program is always equal to the optimum value of the corresponding linear programming relaxation rounded up. Moreover, the optimum solution can then be found in polynomial time. At last, Cook et al. (1984) showed that if d is fixed, there exists a polynomial-time algorithm to test whether a given set of integral vectors constitute a Hilbert basis. For a variable d, it is easy to see that Hilbert basis testing belongs to coNP, but it is still not known whether it is in NP.

5.1 Carathéodory-type theorems

Let X be a finite set of vectors in $\mathbb{Z}^d$ and let $b \in \text{int.cone}(X)$. We aim to find an upper bound on the size of a smallest subset Y of X such that $b \in \text{int.cone}(Y)$. It is easy to see that we cannot give such a bound in terms of the dimension d only. For example, if the set X consists of the vectors

$$x_{ij} = 2^{i-1} e_j + e_d, \quad i = 1, \ldots, n,\ j = 1, \ldots, d-1,$$

where e_j denotes the j-th unit vector in $\mathbb{R}^d$, then all these vectors are needed to express the vector $b \in \mathbb{Z}^d$,

$$\begin{aligned} b_j &= 2^n - 1, \quad j = 1, \ldots, d-1, \\ b_d &= n\,(d-1), \end{aligned}$$

as an integral non-negative combination of the elements of X. Indeed,

$$b = \sum_{i=1}^{n} \sum_{j=1}^{d-1} x_{ij}$$

is the unique representation. Since size$(b) = O(dn)$, we conclude that $\Omega(\text{size}(b))$ of vectors are needed to represent b.

In the above example all vectors in X were, in fact, non-negative integral vectors. The following theorem provides an upper bound for such sets, which is $O(\text{size}(b))$.

Theorem 5.1. *Let X be a finite set of non-negative integral vectors in $\mathbb{R}^d$ and let $b \in \mathbb{R}^d$ belong to* int.cone(X). *Then there exists a subset Y of X such that $b \in$* int.cone(Y) *and*

$$|Y| \leqslant \text{size}(b).$$

Proof. Suppose that

$$b = \sum_{x \in X} \lambda_x x,$$

with $\lambda_x > 0$ integer for all $x \in X$. It suffices to show that if $|X| > \text{size}(b)$, then we can find a proper subset Y of X such that $b \in$ int.cone(Y).

Clearly, $\sum_{x \in Y} x \leqslant b$ for all subsets Y of X. This implies that the number of different vectors of the form $\sum_{x \in Y} x$ is bounded by $\prod_{i=1}^{d}(b_i + 1)$. Inequality $|X| > \text{size}(b)$ implies

$$2^{|X|} > \prod_{i=1}^{d}(b_i + 1),$$

and therefore, there exist two subsets A and B of X, $A \neq B$, such that

$$\sum_{x \in A} x = \sum_{x \in B} x.$$

Consequently, there exist two disjoint subsets of X, namely $A' := A \setminus B$ and $B' := B \setminus A$, satisfying

$$\sum_{x \in A'} x = \sum_{x \in B'} x.$$

Without loss of generality we can assume $A' \neq \emptyset$. Let

$$\lambda^* := \min\{\lambda_x : x \in A'\}$$

and let $x^* \in A'$ be a vector for which this minimum is attained, i.e., $\lambda_x = \lambda^*$. Now we can rewrite

$$\begin{aligned}
\sum_{x \in X} \lambda_x x &= \sum_{x \in X \setminus A'} \lambda_x x + \sum_{x \in A'} \lambda_x x \\
&= \sum_{x \in X \setminus A'} \lambda_x x + \sum_{x \in A'} (\lambda_x - \lambda^*) x + \lambda^* \sum_{x \in A'} x \\
&= \sum_{x \in X \setminus A'} \lambda_x x + \sum_{x \in A'} (\lambda_x - \lambda^*) x + \lambda^* \sum_{x \in B'} x \\
&= \sum_{x \in X} \mu_x x
\end{aligned}$$

where

$$\mu_x := \begin{cases} \lambda_x + \lambda^* & \text{if } x \in B', \\ \lambda_x - \lambda^* & \text{if } x \in A', \\ \lambda_x & \text{otherwise.} \end{cases}$$

Thus, we have $\mu_x \geqslant 0$ for all $x \in X$ and $\mu_{x^*} = 0$. But then $b \in \text{int.cone}(X \setminus \{x^*\})$ and the claim follows. □

Suppose now that vectors in X are not all non-negative. In this case the bound of Theorem 5.1 is no more valid. In order to see this, we consider the set X consisting of vectors

$$\begin{aligned} x_{ij} &= -2^{i-1} e_j + e_d, \quad i = 1, \ldots, n-1,\ j = 1, \ldots, d-1, \\ x_{nj} &= 2^{n-1} e_j + e_d, \quad j = 1, \ldots, d-1, \end{aligned}$$

and the vector b, defined by

$$\begin{aligned} b_j &= 1, \quad j = 1, \ldots, d-1, \\ b_d &= n\,(d-1), \end{aligned}$$

Again,

$$b = \sum_{i=1}^{n} \sum_{j=1}^{d-1} x_{ij}$$

is the unique representation of b as an integral non-negative combination of the vectors from X. In other words, all of these vectors are needed to express b. But $|X| = n(d-1)$, while $\text{size}(b) = O(d \log(dn))$. We can notice, however, that n is roughly the largest size of a vector in X.

Theorem 5.2. *Let $X \subset \mathbb{Z}^d$ be a finite set of integral vectors, such that $X \neq \{\mathbf{0}\}$, and let $b \in \text{int.cone}(X)$. Then there exists a subset Y of X such that $b \in \text{int.cone}(Y)$ and*

$$|Y| \leqslant 2d \log(4dM), \tag{5.3}$$

where

$$M = \max_{x \in X} \|x\|_\infty.$$

Proof. Let b be an integral vector in $\mathbb{R}^d$ and let

$$b = \sum_{x \in X} \lambda_x x,$$

with $\lambda_x > 0$ integer for all $x \in X$. First, we show that if

$$|X| > d \log(2|X|M+1), \tag{5.4}$$

then there exists a proper subset Y of X such that $b \in \text{int.cone}(Y)$.

Indeed, for every subset Y of X, we have

$$\Big\| \sum_{x \in Y} x \Big\|_\infty \leqslant |X|M.$$

This implies that the number of different vectors that are representable as the sum of vectors of a subset Y of X is bounded by $(2|X|M+1)^d$. By (5.4), we have

$$2^{|X|} > (2|X|M+1)^d.$$

Therefore, there exist two subsets A and B of X, with $A \neq B$, such that

$$\sum_{x \in A} x = \sum_{x \in B} x.$$

and we can proceed as in the proof of Theorem 5.1.

It remains to show that the inequality

$$|X| > 2d \log(4dM) \tag{5.5}$$

implies (5.4). Indeed, inequality (5.5) yields

$$M < \frac{2^{|X|/(2d)}}{4d},$$

whence

$$\begin{aligned} d \log(2|X|M+1) &< d \log\left(\frac{|X|}{2d} 2^{|X|/(2d)} + 1\right) \\ &\leqslant d \log\left(2^{|X|/(2d)} \left(\frac{|X|}{2d} + 1\right)\right) \\ &= \frac{|X|}{2} + d \log\left(\frac{|X|}{2d} + 1\right) \\ &\leqslant \frac{|X|}{2} + \frac{|X|}{2} = |X|. \end{aligned}$$

This completes the proof. □

We remark that the bound (5.3) can be written as

$$|Y| \leqslant 2d(2 + s + \log d),$$

where s denotes the maximum size of a component of a vector in X.

Finally, we consider sets X of the very special structure: we suppose that X is *closed under convex combinations*. The latter means that every integral vector in $\mathrm{conv}(X)$ also belongs to X. The theorem provides then a bound, which is independent of the size of the vectors but exponential in the dimension.

Theorem 5.3. *Let $X \subset \mathbb{Z}^d$ be a finite set of integral vectors that is closed under convex combinations and let $b \in \mathrm{int.cone}(X)$. Then there exists a subset Y of X such that $b \in \mathrm{int.cone}(Y)$ and*

$$|Y| \leqslant 2^d.$$

Furthermore, if

$$b = \sum_{x \in X} \lambda_x x,$$

where $\lambda_x \geqslant 0$ integer for all $x \in X$, then there exist integers $\mu_x \geqslant 0$ for all $x \in Y$, such that

$$b = \sum_{x \in Y} \mu_x x \quad \textit{and} \quad \sum_{x \in X} \lambda_x = \sum_{x \in Y} \mu_x.$$

Proof. We say that

$$b = \sum_{x \in X} \lambda_x x,$$

with $\lambda_x \geqslant 0$ integer for all $x \in X$, is a *representation* of b of *value* $\sum_{x \in X} \lambda_x$ with the *potential*

$$\sum_{x \in X} \lambda_x \left\| \left[\begin{smallmatrix} 1 \\ x \end{smallmatrix} \right] \right\|,$$

where $\|\cdot\|$ stands for the Euclidean norm in $\mathbb{R}^{d+1}$. We show that, if there exists a representation of b, then there exists a representation with the same value and at most 2^d non-zero coefficients.

Let

$$b = \sum_{x \in X} \lambda_x x$$

be the representation of b of value γ with the smallest potential. If the number of non-zero coefficients is greater than 2^d, then there exist two different vectors x_1 and x_2 in X such that $\lambda_{x_1} > 0$, $\lambda_{x_2} > 0$ and

$$x_1 \equiv x_2 \mod 2.$$

Since X is closed under convex combinations, $1/2(x_1 + x_2)$ belongs to X. Suppose without loss of generality that $\lambda_{x_1} \geqslant \lambda_{x_2}$. Then

$$\lambda_{x_1} x_1 + \lambda_{x_2} x_2 = (\lambda_{x_1} - \lambda_{x_2}) x_1 + 2\lambda_{x_2} \left(\frac{1}{2}(x_1 + x_2) \right).$$

Since the vectors $\left[\begin{smallmatrix} 1 \\ x_1 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} 1 \\ x_2 \end{smallmatrix}\right]$ are not collinear, we have

$$\begin{aligned}(\lambda_{x_1} - \lambda_{x_2}) \left\| \left[\begin{smallmatrix} 1 \\ x_1 \end{smallmatrix}\right] \right\| + 2\lambda_{x_2} \left\| \left[\begin{smallmatrix} 1 \\ 1/2(x_1+x_2) \end{smallmatrix}\right] \right\| &= (\lambda_{x_1} - \lambda_{x_2}) \left\| \left[\begin{smallmatrix} 1 \\ x_1 \end{smallmatrix}\right] \right\| + \lambda_{x_2} \left\| \left[\begin{smallmatrix} 1 \\ x_1 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 1 \\ x_2 \end{smallmatrix}\right] \right\| \\ &< (\lambda_{x_1} - \lambda_{x_2}) \left\| \left[\begin{smallmatrix} 1 \\ x_1 \end{smallmatrix}\right] \right\| + \lambda_{x_2} \left(\left\| \left[\begin{smallmatrix} 1 \\ x_1 \end{smallmatrix}\right] \right\| + \left\| \left[\begin{smallmatrix} 1 \\ x_2 \end{smallmatrix}\right] \right\| \right) \\ &= \lambda_{x_1} \left\| \left[\begin{smallmatrix} 1 \\ x_1 \end{smallmatrix}\right] \right\| + \lambda_{x_2} \left\| \left[\begin{smallmatrix} 1 \\ x_2 \end{smallmatrix}\right] \right\|\end{aligned}$$

Thus, replacing

- $\lambda_{1/2(x_1+x_2)}$ by $\lambda_{1/2(x1+x2)} + 2\lambda_{x_2}$,
- λ_{x_1} by $\lambda_{x_1} - \lambda_{x_2}$, and
- λ_{x_2} by 0

we obtain a representation of b with the same value and smaller potential, which is a contradiction. □

5.2 Integer programming problems in standard form

Motivated by their integer analogue of Carathéodory's theorem, Cook et al. (1986) considered integer programs

$$\min \{cx : Ax = b,\ x \in \mathbb{Z}^n_+\}, \tag{5.6}$$

where $A \in \mathbb{Z}^{d \times n}$ is a matrix, $c \in \mathbb{Z}^n$ is a row-vector, and $b \in \mathbb{Z}^d$ is a column-vector, under the condition that $yA \leqslant c$ is a *totally dual integral* system. They showed that if the polyhedron $\{y : yA \leqslant c\}$ is full-dimensional and the minimum (5.6) exists, then there exists an optimum solution with at most $2d - 1$ non-zero variables. Using the result of Sebő (1990), this bound can be improved to $2d - 2$.

We can also apply our results, obtained in the previous section, to prove similar bounds for *general* integer programs in standard form. Let x^* be an integral optimum solution of (5.6) and let $\gamma := cx^*$; then γ is an integer. The vector $\left[\begin{smallmatrix} \gamma \\ b \end{smallmatrix}\right]$ is, therefore, an integral non-negative combination of the columns of the matrix $\left[\begin{smallmatrix} c \\ A \end{smallmatrix}\right]$. Obviously, any other integral non-negative combination for $\left[\begin{smallmatrix} \gamma \\ b \end{smallmatrix}\right]$ also yields an integral optimum solution. Thus, Theorems 5.1 and 5.2 imply the following.

Corollary 5.4. *Let A be a non-negative integral matrix and c a non-negative integral row-vector. If the minimum* (5.6) *exists, then there is an optimum solution with at most*

$$2\,\mathrm{size}(b)$$

non-zero variables.

Proof. Since A and c are integral, we have

$$\gamma \leqslant \prod_{i=1}^{d} b_i \leqslant \prod_{i=1}^{d} (b_i + 1) - 1,$$

and therefore,

$$\log(\gamma + 1) \leqslant \sum_{i=1}^{d} \log(b_i + 1) = \mathrm{size}(b).$$

The claim follows from Theorem 5.1. □

Corollary 5.5. *Let A be an arbitrary integral matrix. If the minimum* (5.6) *exists, then there is an optimum solution with at most*

$$2(d+1)(2 + s + \log(d+1))$$

non-zero variables, where s is the largest size of a coefficient in A and c.

These corollaries are very important for integer programs, where the columns of the matrix A cannot be written explicitly. Such integer programs are usually derived from combinatorial optimisation problems, e.g., packing, covering, partitioning, and tackled with column generation approach. Corollaries 5.4 and 5.5 then ensure that there exists an optimum solution of polynomial size. Chapter 6 is mostly devoted to one particular example of such integer programs, which is derived from the *cutting stock problem*.

Now, we switch to integer programs of the form

$$\min\left\{\mathbf{1}x : Ax = b,\, x \in \mathbb{Z}^n_+\right\}. \tag{5.7}$$

Cook et al. (1986) proved that for matrices $A \in \{0,1\}^{d\times n}$ such that $yA \leqslant \mathbf{1}$ has the *integer rounding property*, i.e.,

$$\min\left\{\mathbf{1}x : Ax = b,\, x \in \mathbb{Z}^n_+\right\} = \left\lceil \min\left\{\mathbf{1}x : Ax = b,\, x \in \mathbb{R}^n_+\right\}\right\rceil$$

for all vectors b for which the integer program is feasible, if the minimum (5.7) exists, then there is an optimum solution with at most $2d-1$ non-zero components.

We consider the case when the columns of A are closed under convex combinations; for example, all integral vectors in a given polyhedron in $\mathbb{R}^d$. Theorem 5.3 then yields an analogue of a "basic solution" for integer programs.

Corollary 5.6. *Suppose that the columns of a matrix A are closed under convex combination. If the minimum* (5.7) *exists, then there is an optimum solution with at most 2^d non-zero variables.*

The above statement implies that, if d is fixed and the matrix A is given explicitly (all n columns are in the input), then the integer program (5.7) can be solved in polynomial time. Indeed, we can simply enumerate all possible "bases", each consisting of 2^d columns of A, and solve an integer programming problem in fixed dimension for each such a basis, using the Lenstra's algorithm.

Chapter 6

Cutting Stock Problem

In this chapter we investigate integer programs derived from the so-called *cutting stock problem*. These are the integer programs in standard form, with the constraint matrix composed of all integral solutions of the *knapsack inequality* $ax \leqslant 1$. Clearly, the number of these columns is exponential in the input size.

6.1 Introduction

The classical one-dimensional *bin packing problem* is stated as follows: Given n items of sizes $s_1, \ldots, s_n \in (0,1]$, find the minimum number of bins needed to pack all these items, under the condition that the total size of the items packed into a bin does not exceed 1. A closely related problem is that of stock cutting. In the one-dimensional *cutting stock problem* the input is represented in a *compact form*: for each index $i = 1, \ldots, d$, we are given two numbers, $a_i \in (0,1]$ and $b_i \in \mathbb{Z}_+$, meaning that there are b_i items of size a_i. The question is the same as for bin packing: we need to partition all items into sets such that the total size of the items in each set is at most 1 and the number of sets is minimised. Clearly, we can assume, without loss of generality, that all sizes a_i $(i = 1, \ldots, d)$ are different and all numbers b_i $(i = 1, \ldots, d)$ are positive.

We denote by (n, s) an instance of the bin packing problem, where $s \in (0,1]^n$ is the vector of items' sizes. The instance of the cutting stock is denoted by (d, a, b); here $a \in (0,1]^n$ is the vector of (different) item sizes, while the vector $b \in \mathbb{Z}_+^n$ specifies the *multiplicity* of each item.

So, bin packing and cutting stock are essentially the same problem, the only difference being in the input representation. While any polynomial-time algorithm for the cutting stock problem runs also in polynomial time for bin packing, the con-

verse is not necessarily true: the size of the input to the bin packing problem is in general exponential in the size of the input to the corresponding cutting stock problem. In this sense, the cutting stock problem is *harder* than that of bin packing. In particular, the bin packing problem is NP-complete, see Garey and Johnson (1979), that immediately implies that cutting stock is NP-hard; however, it was not known so far, whether it admits a solution whose size is polynomial in the size of the input; see, for example, Marcotte (1986). Our results from the previous chapter allow to answer this question positively, since we can formulate the problem as an integer program in standard form—we describe the details in Section 6.2.

For the integer program itself, it was observed that its linear programming relaxation is very strong in practice. In fact, no instance of the cutting stock problem is known with the integer programming gap greater than or equal to 2. This motivates study of the integer programming gap of these integer programs, and Sections 6.3 and 6.4 are devoted to this question. We show that the approximation algorithm of Karmarkar and Karp (1982) implies that the integer programming gap for an instance $I = (d, a, b)$ is bounded by $O(\log^2 d)$.

Finally, in Section 6.5 we present two other integer programming formulations, which appear to have polynomial size. To the best of our knowledge, polynomial-size integer programming formulation for the cutting stock was not known so far.

For the rest of this introductory section, we summarise the most important results concerning bin packing and cutting stock. As we have already mentioned, the problems are NP-hard in general. Moreover, no approximation algorithm of factor better than 3/2 is possible, unless P = NP. Consequently, no polynomial-time approximation scheme can exist for bin packing, unless P = NP. Nonetheless, Fernandez de la Vega and Lueker (1981) established the so-called *asymptotic polynomial-time approximation scheme*: for any given $\varepsilon > 0$, there is an algorithm that runs in time polynomial in the input size and yields a solution of value at most

$$(1+\varepsilon)\mathrm{OPT}(I) + 1,$$

where $\mathrm{OPT}(I)$ denotes the optimum value of the instance I of the bin packing problem, and runs in time polynomial in the input size. We remark that it can also be implemented to run in polynomial time with respect to the compact input, as for the cutting stock problem.

Karmarkar and Karp (1982) presented an *asymptotic fully polynomial-time approximation scheme* that, given $\varepsilon > 0$, computes a solution of value at most

$$(1+\varepsilon)\mathrm{OPT}(I) + O(\varepsilon^{-2}),$$

and runs in time polynomial in the input size and $1/\varepsilon$. They also described a polynomial-time near-optimal approximation algorithm, that computes a solution of

value at most

$$\mathrm{OPT}(I) + O(\log^2 \mathrm{OPT}(I)).$$

As we shall see in Section 6.4, the bound is actually

$$\mathrm{LIN}(I) + O(\log^2 d),$$

where $\mathrm{LIN}(I)$ denotes the optimum value of the linear programming relaxation and d is the number of different item sizes in the instance I. Again, we remark that these algorithms can be implemented to run in polynomial time with respect to the compact input, too.

Yet, the bin packing problem is, perhaps, a leader among combinatorial optimisation problems in what concerns the number of different *heuristics* proposed for its solution. We shall apparently use the heuristic called *first fit decreasing*: it takes the items in non-increasing order of their sizes and packs an item into the first available bin, opening a new bin only if necessary. Johnson (1973) showed that the the first fit decreasing algorithm computes a solution of value at most

$$\frac{11}{9}\mathrm{OPT}(I) + 4.$$

We refer to Coffman et al. (1997) for a good survey on the approximation algorithms for bin packing.

Finally, we mention an interesting *open problem* concerning cutting stock. It arises if we assume the number of different item sizes to be fixed: Is the problem still NP-hard or polynomial-time solvable? McCormick et al. (2001) proved polynomial-time solvability in case $d = 2$, but for $d \geqslant 3$ the problem remains unsolved.

6.2 Integer programming formulation

Let $I = (d, a, b)$ be an instance of the cutting stock problem. Any feasible packing of a single bin can be represented by an integral vector $v \in \mathbb{Z}_+^d$, where v_i $(i = 1, \ldots, d)$ is the number of items of size a_i packed into the bin, satisfying the constraint

$$av \leqslant 1;$$

such a vector is called a *pattern*. If we enumerate all possible patterns $v_1, \ldots, v_N$, then the cutting stock problem I can be formulated as the integer linear program,

which was first introduced by Eisemann (1957):

$$\begin{aligned} \min \quad & \sum_{j=1}^{N} \lambda_j \\ \text{s. t.} \quad & \sum_{j=1}^{N} \lambda_j v_j = b, \\ & \lambda_j \geqslant 0 \text{ integer}, \quad j = 1, \ldots, N; \end{aligned} \tag{6.1}$$

here the variable λ_j $(j = 1, \ldots, N)$ represents the *frequency* of the pattern v_j in a solution. We denote by OPT(I) the optimum value of (6.1); clearly, it is also the optimum value of the original cutting stock problem I. The corresponding linear programming relaxation has the form

$$\begin{aligned} \min \quad & \sum_{j=1}^{N} \lambda_j \\ \text{s.t.} \quad & \sum_{j=1}^{N} \lambda_j v_j = b, \\ & \lambda_j \geqslant 0, \quad j = 1, \ldots, N. \end{aligned} \tag{6.2}$$

We denote the optimum value of this linear program by LIN(I).

Both problems (6.1) and (6.2) have an enormous number of variables, namely, equal to the number of non-negative integral solutions of the inequality $av \leqslant 1$, and therefore, cannot be written explicitly. Furthermore, even the linear programming relaxation (6.2) is NP-hard. To see this, consider its dual

$$\max\{yb : yv_j \leqslant 1,\ j = 1, \ldots, N\}. \tag{6.3}$$

The corresponding separation problem is: Given a row-vector y^*, check if it is feasible for (6.3), and if not, find a vector $v \in \mathbb{Z}^d$ such that $y^* v > yv$ for all feasible vectors y. This is equivalent to the *knapsack problem*

$$\max\{y^* v : av \leqslant 1,\ v \in \mathbb{Z}_+^d\}. \tag{6.4}$$

Indeed, y^* is a feasible solution of (6.3) if and only if the optimum value of (6.4) does not exceed 1. This means, by Theorem 2.7, that the linear program (6.2) is as hard as the knapsack problem, and the latter is known to be NP-hard.

However, if d is fixed, the problem (6.4) is just an integer programming problem with a fixed number of variables, and hence, solvable in polynomial time by the algorithm of Lenstra (1983). Consequently, the linear program (6.3), and therefore, (6.2), are also polynomial-time solvable if d is fixed. We formulate this as a lemma.

Lemma 6.1. *If d is fixed, then the linear program* (6.2) *is polynomial-time solvable.*

Gilmore and Gomory (1961) (see also Gilmore and Gomory (1963)) described an efficient practical method to solve the linear program (6.2), nowadays known as the *column generation method*. Essentially, it can be viewed as the simplex method, where a column to be added into the current basis is derived by solving the knapsack problem (6.4). Karmarkar and Karp (1982) described an algorithm that solves the linear program (6.2) approximately — within any given tolerance — in polynomial time; formally, they showed that for any $t > 0$, there exists a polynomial-time algorithm that computes a feasible solution to (6.2) of value at most $\mathrm{LIN}(I) + t$. Their algorithm is basically the ellipsoid method, but the separation problem is solved approximately, using a fully polynomial-time approximation scheme for the knapsack problem.

We conclude this section with a bound on the size of an optimum solution for the cutting stock problem. As we have said, it was not even known before whether it admits an optimal solution whose encoding length is polynomial in the input size.

The following is an immediate consequence of Corollaries 5.4 and 5.6.

Theorem 6.2. *Let $I = (d, a, b)$ be an instance of the cutting stock problem. Then there exists an optimal solution of* (6.1) *with at most*

$$\min\left\{2\,\mathrm{size}(b),\, 2^d\right\}$$

non-zero components.

6.3 Residual instances and small items

The linear program (6.2) has an important advantage over other relaxations of the cutting stock problem: its optimum value is very close to the integral optimum value. Surprisingly, the vast majority of cutting stock instances in practice satisfy the property

$$\mathrm{OPT}(I) = \lceil \mathrm{LIN}(I) \rceil;$$

some classes of such instances were investigated by Marcotte (1985). There are also instances for which this property does not hold; for example, see Marcotte (1986), Rietz et al. (2002). However, no instance is known to violate the property

$$\mathrm{OPT}(I) \leqslant \lceil \mathrm{LIN}(I) \rceil + 1, \tag{6.5}$$

and it was conjectured by Scheithauer and Terno (1997) that (6.5) holds in general, for all instances of the cutting stock problem.

In the next section we prove that

$$\mathrm{OPT}(I) - \mathrm{LIN}(I) \leqslant O(\log^2 d)$$

for all instances of the cutting stock problem. Fairly, this bound follows directly from the near-optimal approximation algorithm of Karmarkar and Karp (1982) and it is not clear to us why it has not been shown explicitly so far.

First, let us agree on some notation. Since there is no crucial difference between cutting stock and bin packing except their input representation, and the latter applies only when we talk about algorithmic aspects of the problem, we often switch from one to another, depending on which input representation is more suitable for us in this particular moment. Thus, the same instance I can be understood as the bin packing instance (n, s), as the cutting stock instance (d, a, b). The bin packing input can also be viewed as the multi-set of items, that justifies the notation, like $I \cup J$ or $I \setminus J$ for instances I and J. The total size of the items in I is sometimes denoted by $s(I)$, hence $s(I) = ab$.

Lemma 6.3. *For any instance I of the bin packing problem, we have*

$$\mathrm{OPT}(I) \leqslant 2s(I) + 1.$$

Proof. There exists a packing such that all but one bins are more than half-full. If k is the number of bins used, then

$$(k-1)\frac{1}{2} \leqslant s(I),$$

and hence,

$$k \leqslant 2s(I) + 1.$$

The claim follows. □

Let $I = (d, a, b)$ be an instance of the cutting stock problem and let $\lambda \in \mathbb{R}^N_+$ be an optimal solution of the corresponding linear program (6.2). We can assume that λ is a *basic* optimum solution (Theorem 2.6), and therefore, has at most d non-zero components. Without loss of generality, let $\lambda_1, \ldots, \lambda_d$ be these components. Then

$$b = \sum_{i=1}^{d} \lambda_i v_i = \sum_{i=1}^{d} \lfloor \lambda_i \rfloor v_i + \sum_{i=1}^{d} \{\lambda_i\} v_i,$$

where $\{\lambda_i\} := \lambda_i - \lfloor \lambda_i \rfloor$ denotes the fractional part of the number λ_i. We say that an instance $I' = (d, a, b')$, where

$$b' = \sum_{i=1}^{d} \{\lambda_i\} v_i,$$

is a *residual instance* for I. In other words, an instance of the cutting stock problem is called residual if there exists a basic optimum solution of the linear program (6.2) with all variables being smaller than 1.

Lemma 6.4. *Let $I = (d, a, b)$ be an instance of the cutting stock problem, and let $I' = (d, a, b')$ be its residual instance. Then*

$$\mathrm{OPT}(I) - \mathrm{LIN}(I) \leqslant \mathrm{OPT}(I') - \mathrm{LIN}(I').$$

Proof. The proof is straightforward. For some basic optimum solution λ of the linear program for the instance I, we have

$$b = \sum_{i=1}^{d} \lfloor \lambda_i \rfloor v_i + \sum_{i=1}^{d} \{\lambda_i\} v_i = \sum_{i=1}^{d} \lfloor \lambda_i \rfloor v_i + b'.$$

Then

$$\mathrm{LIN}(I) = \sum_{i=1}^{d} \lfloor \lambda_i \rfloor + \mathrm{LIN}(I')$$

and

$$\mathrm{OPT}(I) \leqslant \sum_{i=1}^{d} \lfloor \lambda_i \rfloor + \mathrm{OPT}(I')$$

and the claim follows. □

Lemma 6.4 implies that the maximum of the difference $\mathrm{OPT}(I) - \mathrm{LIN}(I)$ over all instances of the cutting stock problem is attained for some residual instance. Thus, we can restrict our attention on residual instances only. It is easy to see that for any residual instance $I = (d, a, b)$, we have

$$s(I) = ab \leqslant \mathrm{LIN}(I) \leqslant \lceil \mathrm{LIN}(I) \rceil \leqslant \mathrm{OPT}(I) \leqslant d.$$

Lemma 6.5. *Let $I = (d, a, b)$ be a residual instance of the cutting stock problem and let I' be the instance obtained from I by eliminating all items of size less than or equal to $1/s(I)$. Then*

$$\mathrm{OPT}(I) - \lceil \mathrm{LIN}(I) \rceil \leqslant \max\left\{\mathrm{OPT}(I') - \lceil \mathrm{LIN}(I') \rceil, 1\right\}.$$

Proof. Having an optimum packing for I', we apply the first fit decreasing heuristic to pack the remaining items. If the heuristic does not open a new bin, we are done. Otherwise, we obtain a packing with, say p, bins:

$$I = v_1 + v_2 + \ldots + v_p,$$

where each, except possibly the last, bin satisfy

$$av_j > 1 - 1/s(I) \quad (j = 1, \ldots, p-1)$$

Now, if $p \geqslant \lceil s(I) \rceil + 1$, then

$$\sum_{j=1}^{\lceil s(I) \rceil} av_j > \sum_{j=1}^{\lceil s(I) \rceil} (1 - 1/s(I)) \geqslant \lceil s(I) \rceil - 1.$$

Then the size of the items which are not packed within $v_1, \ldots, v_s$ is

$$a\left(b - \sum_{j=1}^{\lceil s(I) \rceil} v_j\right) = ab - \sum_{j=1}^{\lceil s(I) \rceil} av_j < s(I) - (\lceil s(I) \rceil - 1) \leqslant 1,$$

and therefore, fits into one bin. Hence, $\mathrm{OPT}(I) \leqslant \lceil s(I) \rceil + 1$, and the claim follows. □

Therefore, we can restrict our attention only to residual instances I with all items bigger than $1/s(I)$. It gives immediately the following result.

Lemma 6.6. *Let $I = (d, a, b)$ be an instance of the cutting stock problem, where all sizes are of the form $a_i = 1/k_i$, $k_i \in \mathbb{Z}_+$ $(i = 1, \ldots, d)$. Then* (6.5) *holds.*

Proof. The proof is by induction on d. Let $I = (d, a, b)$ be a residual instance with $1/a_i \in \mathbb{Z}_+$ for all $i = 1, \ldots, d$. Without loss of generality, we can assume that $b > 0$; if $b_i = 0$ for some i, then the size a_i does not occur in the instance and we have, in fact, an instance with $d - 1$ different item sizes. But then there is an item of size $\leqslant 1/d$, and it can be eliminated, due to Lemma 6.5. □

6.4 Integer programming gaps

Let $I = (n, s)$ and $I' = (n', s')$ be two instances of the bin packing problem. Without loss of generality, we may assume that

$$s_1 \geqslant \ldots \geqslant s_n \quad \text{and} \quad s'_1 \geqslant \ldots \geqslant s'_{n'}.$$

We say that instance I *dominates* instance I', and write $I \geqslant I'$, if $n \geqslant n'$ and $s_i \geqslant s'_i$ for all $i = 1, \ldots, n'$. Clearly, if $I \geqslant I'$, then

$$\mathrm{LIN}(I) \geqslant \mathrm{LIN}(I') \quad \text{and} \quad \mathrm{OPT}(I) \geqslant \mathrm{OPT}(I'),$$

since any (linear) solution for I' can be transformed to a (linear) solution for I, by using essentially the same set of patterns.

Theorem 6.7. *For any instance $I = (d, a, b)$ of the cutting stock problem,*

$$\mathrm{OPT}(I) \leqslant \mathrm{LIN}(I) + C \log d \ln d,$$

where C is a constant independent of I ($C \leqslant 30$).

Proof. The proof is by induction on d. Since the case $d = 1$ is trivial, suppose that $d > 1$ and consider a residual instance $I = (d, a, b)$ of the cutting stock problem. By Lemma 6.5 we can also assume that $a_i > 1/d$ for all $i = 1, \ldots, d$. Let (n, s) be the equivalent bin packing representation; without loss of generality, $s_1 \geqslant \ldots \geqslant s_n$. We split the set of items into groups as follows.

We fill the first group, G_1, with first l_1 largest items, $s_1, \ldots, s_{l_1}$, such that

$$\sum_{i=1}^{l_1} s_i > 2, \quad \text{but} \quad \sum_{i=1}^{l_1-1} s_i \leqslant 2.$$

Then we fill the group G_2, in a similar way, with items $s_{l_1+1}, \ldots, s_{l_1+l_2}$, until their total size exceeds 2. We continue this process until all items are considered, obtaining the sets $G_1, \ldots, G_k$, with $|G_i| = l_i$ $(i = 1, \ldots, k)$. Then we have

$$k \leqslant s(I)/2 + 1 \leqslant d/2 + 1,$$

since the total size of the items in each group G_i $(i = 1, \ldots, k-1)$ is bigger than 2. Moreover, since all items have size bigger than $1/d$, we have $l_i < 2d$ for all $i = 1, \ldots, k-1$.

Clearly, $l_1 \leqslant \ldots \leqslant l_{k-1}$. Let G_i' be the set of items obtained from G_i by removing smallest $l_i - l_{i-1}$ items $(i = 2, \ldots, k-1)$; then $G_{i-1} \geqslant G_i'$. Let H_i be the set of items obtained from G_i' by setting the size of all its items equal to the size of the largest item in G_i' $(i = 2, \ldots, k-1)$. Now, $G_{i-1} \geqslant H_i \geqslant G_i'$ for all $i = 2, \ldots, k-1$.

We define

$$J := \bigcup_{i=2}^{k} H_i, \quad J' := G_1 \cup G_k \cup \bigcup_{i=2}^{k} (G_i \setminus G_i'),$$

Then

$$J \leqslant I \leqslant J \cup J',$$

and therefore,

$$\mathrm{OPT}(J) \leqslant \mathrm{OPT}(I) \leqslant \mathrm{OPT}(J \cup J') \leqslant \mathrm{OPT}(J) + \mathrm{OPT}(J'),$$

and

$$\mathrm{LIN}(J) \leqslant \mathrm{LIN}(I) \leqslant \mathrm{LIN}(J \cup J') \leqslant \mathrm{LIN}(J) + \mathrm{LIN}(J').$$

It follows that

$$\mathrm{OPT}(I) - \mathrm{LIN}(I) \leqslant \mathrm{OPT}(J) - \mathrm{LIN}(J) + \mathrm{OPT}(J'). \tag{6.6}$$

It remains to estimate $\mathrm{OPT}(J')$. For this purpose, we give an upper bound on the total size of the items in J' and apply Lemma 6.3. First, observe that $s(G_1 \cup G_k) \leqslant 6$. For each $i = 2, \ldots, k-1$, the set J' contains $l_i - l_{i-1}$ *smallest* items from G_i. Since $l_i - 1$ items from G_i have size at most 2 (by construction), we get

$$s(G_i \setminus G_i') \leqslant 2\frac{l_i - l_{i-1}}{l_i - 1}.$$

All together, we obtain

$$\begin{aligned} s(J') &\leqslant 6 + 2\sum_{i=2}^{k-1} \frac{l_i - l_{i-1}}{l_i - 1} \leqslant 6 + 2\sum_{i=2}^{k-1} \sum_{j=l_{i+1}}^{l_i - 1} \frac{1}{j} = 6 + 2\sum_{j=1}^{l_{k-1}-1} \frac{1}{j} \\ &\leqslant 6 + 2\ln l_{k-1} \leqslant 6 + 2\ln(2d). \end{aligned}$$

With Lemma 6.3, we conclude that

$$\mathrm{OPT}(J') \leqslant 13 + 4\ln(2d) \leqslant C\ln d. \tag{6.7}$$

By induction hypothesis, we also have

$$\mathrm{OPT}(J) - \mathrm{LIN}(J) + \mathrm{OPT}(J') \leqslant C\log(d/2)\ln(d/2) + C\ln d,$$

that together with (6.7) and (6.6) yields

$$\begin{aligned} \mathrm{OPT}(I) - \mathrm{LIN}(I) &\leqslant \mathrm{OPT}(J) - \mathrm{LIN}(J) + \mathrm{OPT}(J') \leqslant C\log(d/2)\ln(d/2) + C\ln d \\ &= C(\log d - 1)(\ln d - 1) + C\ln d \leqslant C\log d \ln d. \end{aligned}$$

This completes the proof. □

6.5 Polynomial-size integer programs

Integer programming formulation described in Section 6.2 is not the only possible for the cutting stock problem. For the survey on various integer programming models for cutting stock, we refer to Valério de Carvalho (2002). We mention, however, that like the formulation we used, all other integer programs discussed there involve exponentially many variables or even constraints. For example, the assignment formulation of Kantorovich (1960) has a binary variable x_{ij} for each pair of an item s_i

and a bin B_j, showing whether the item s_i is to be packed into the bin B_j. For a cutting stock instance $I = (d, a, b)$, this already gives $O(d\|b\|_\infty)$ variables.

Perhaps, the first integer programming formulation of polynomial size was proposed by Belov and Weismantel (2003). The idea is that, given a cutting stock problem $I = (m, a, b)$, each pattern can be represented as the sum of a small number of *sub-patterns*. Let us denote

$$u_{ij} = 2^{j-1} e_i, \quad i = 1, \dots, d,\ j = 1, \dots, K_i,$$

where $K_i = \lceil \log(1/a_i) \rceil$. Then the number of these vectors u_{ij} is polynomial in the input size, and each valid pattern $v \in \mathbb{Z}_+^d$, $av \leqslant 1$, can be written as the sum

$$v = \sum_{i=1}^{d} \sum_{j=1}^{K_i} \mu_{ij} u_{ij},$$

where $\mu_{ij} \in \{0, 1\}$ for all i, j.

Now, suppose that we have an upper bound D on the number of patterns to be used in an optimum solution. In other words, an optimum solution can be written in the form

$$b = \sum_{k=1}^{D} \lambda_k v_k$$

for some patterns v_k and some non-negative integers $\lambda_1, \dots, \lambda_D$ ($k = 1, \dots, D$); or, in terms of sub-patterns,

$$b = \sum_{k=1}^{D} \sum_{i=1}^{n} \sum_{j=1}^{K_i} \lambda_k \mu_{kij} u_{ij}$$

for some non-negative integers λ_k and binaries μ_{kij} ($k = 1, \dots, D$, $i = 1, \dots, d$ and $j = 1, \dots, K_i$). This expression is non-linear, but variables μ_{kij} are binary, which allows to use a technique of "switching" and express the product $\lambda_k \mu_{kij}$ as a single integer

variable, say η_{kij}. This results in the following integer programming formulation:

$$\begin{aligned}
\min \quad & \sum_{k=1}^{D} \lambda_k \\
\text{s.t.} \quad & \sum_{k=1}^{D} \sum_{i=1}^{d} \sum_{j=1}^{K_i} \eta_{kij} u_{ij} = b, \\
& \sum_{i=1}^{d} \sum_{j=1}^{K_i} \mu_{kij} u_{ij} \leqslant 1 \quad \text{for all } k, \\
& \eta_{kij} \leqslant \eta_k^* \mu_{kij} \quad \text{for all } k, i, j, \\
& \lambda_k \geqslant \eta_{kij} \quad \text{for all } k, i, j, \\
& \lambda_k \in \mathbb{Z}_+, \eta_{kij} \in \mathbb{Z}_+, \mu_{kij} \in \{0,1\} \quad \text{for all } k, i\, j,
\end{aligned}$$

where η_k^* is some robust upper bound on the value of $\eta_{kij} = \lambda_k \mu_{kij}$ and can be taken to be, for example,

$$\eta_k^* = \sum_{i=1}^{n} b_i$$

for all k.

We remark that the authors did not state explicitly that their integer program is of polynomial size. However, it follows immediately from Theorem 6.2, since D can be chosen to be polynomial in the input size.

The second polynomial-size integer programming formulation was suggested by András Sebő and explicitly exploits Catathéodory-type bounds from Section 5.2. Let $I = (d, a, b)$ be an instance of the cutting stock problem, and suppose we want to decide whether there is a solution of value less than m. If so, then the vector $\left[\begin{smallmatrix} m \\ b \end{smallmatrix}\right]$ belongs to the integer cone of the vectors $\left[\begin{smallmatrix} 1 \\ v_i \end{smallmatrix}\right]$, where $v_1, \ldots, v_N$ are all patterns of I,[1] hence

$$\left[\begin{smallmatrix} m \\ b \end{smallmatrix}\right] = \sum_{i=1}^{N} \lambda_i \left[\begin{smallmatrix} 1 \\ v_i \end{smallmatrix}\right]$$

for some non-negative integers $\lambda_1, \ldots, \lambda_N$. If D is an upper bound on the number of patterns to be used in a solution, then at least one coefficient λ_i in that solution must be greater than or equal to $\lceil m/D \rceil$. In other words,

$$\left[\begin{smallmatrix} m \\ b \end{smallmatrix}\right] - \lceil m/D \rceil \left[\begin{smallmatrix} 1 \\ v \end{smallmatrix}\right] = \sum_{i=1}^{N} \lambda_i \left[\begin{smallmatrix} 1 \\ v_i \end{smallmatrix}\right] \tag{6.8}$$

[1] Notice that $v = 0$ is also a valid pattern for the cutting stock problem.

for some pattern v and some non-negative integers $\lambda_1, \ldots, \lambda_N$. Now, we see that the vector $\begin{bmatrix} m-\lceil m/D \rceil \\ b' \end{bmatrix}$ also belongs to the integer cone of vectors $\begin{bmatrix} 1 \\ v_i \end{bmatrix}$, $i = 1, \ldots, N$, where

$$b' = b - \lceil m/L \rceil v,$$

and therefore, is also expressible as the sum of D vectors. Thus, at least one coefficients λ_i in (6.8) must be greater than or equal to $\lceil (m - \lceil m/D \rceil)/D \rceil$. Repeating this procedure, we conclude that $\begin{bmatrix} m \\ b \end{bmatrix}$ is of the form

$$\begin{bmatrix} m \\ b \end{bmatrix} = \sum_{i=1}^{M} \lambda_i^* \begin{bmatrix} 1 \\ v_i \end{bmatrix}$$

for some (unknown) patterns v_i and coefficients λ_i^*, recursively defined as follows:

$$\lambda_i^* = \left\lceil \frac{1}{D} \left(m - \sum_{j=1}^{i-1} \lambda_j^* \right) \right\rceil \tag{6.9}$$

for $i = 1, \ldots, M$. For the bound M, we get

$$M = O(D \log m). \tag{6.10}$$

The latter can easily be shown by induction, since the number of patterns for a vector $\begin{bmatrix} m \\ b \end{bmatrix}$ with the first component equal to m is equal to the number of patterns we used for some vector with the first component equal to $m - \lceil m/D \rceil$ plus 1. The bound (6.10) is polynomial, by Theorem 5.2, that gives us the polynomial-size formulation in terms of integer programming:

$$\begin{aligned} \begin{bmatrix} m \\ b \end{bmatrix} &= \sum_{i=1}^{M} \lambda_i^* \begin{bmatrix} 1 \\ v_i \end{bmatrix} \\ a v_i &\leqslant 1 \quad \text{for all } i, \\ v_i &\in \mathbb{Z}_+ \quad \text{for all } i, \end{aligned}$$

where the numbers λ_i^* are defined by (6.9).

Chapter 7

Conclusions and Open Questions

At the end, let us summarise the main contributions of the present thesis once again.

It has been known for a long time that the set of vectors $b \in \mathbb{R}^m$, for which the system of linear inequalities $Ax \leqslant b$ has a solution, is a polyhedron, and this polyhedron can be found using Fourier–Motzkin elimination — in polynomial time if the number n of variables x is fixed. Kannan (1992) asked an analogous question for the case of *integer* variables x, and provided a algorithm that computes a decomposition of the space of right-hand sides into *integer projections* of polyhedra such that, for each particular region of this decomposition, we need to try only constantly many candidate solutions, all defined in terms of affine transformations of b, in order to check whether the system $Ax \leqslant b$ has an integer solution. However, his decomposition, as well as the running time of the algorithm, was exponential in the dimension m of the b-space. We improved the algorithm so that it runs in polynomial time under the only assumption that n is fixed. It seems to be best possible, since answering whether the system $Ax \leqslant b$ has an integer solution in case of variable n is just an ordinary integer programming problem, and therefore, NP-complete.

A related question concerns the complexity of so-called parameterised integer programming problems: the question here is to decide the statement

$$\forall b \in Q \quad \exists x \in \mathbb{Z}^n : \quad Ax \leqslant b \,?$$

where Q is a polyhedron in $\mathbb{R}^m$ and A is an $m \times n$-matrix. Again, the algorithm of Kannan (1992) was able to solve this problem but took exponential time in m. We made his algorithm running in time polynomial in m. The number n of variables x needs to be fixed, but this assumption is absolutely reasonable, since integer programming is just a special case of parameterised integer programming.

Hoşten and Sturmfels (2003) considered integer programs of the form

$$\min\{cx : Ax = b,\ x \geqslant 0 \text{ integer}\}$$

and developed an algorithm to find the so-called integer programming gap: the maximum difference between the optimum value of the integer program and the optimum value of its linear programming relaxation, for b varying over all vectors for which the integer program is feasible. Their algorithm ran in polynomial time if the number of variables is fixed. Implicitly, however, it also implies that the number of constraints is fixed. We considered a more general case and asked the same question for integer programs of the form

$$\max\{cx : Ax \leqslant b,\ x \text{ integer}\},$$

assuming the number of variables to be fixed. Our result — a polynomial-time algorithm — generalises that of Hoşten and Sturmfels (2003), since each equality constraint can be replaced by two inequality constraints.

It is often the case in combinatorial optimisation, that a problem can be formulated as an integer program in standard form

$$\min\{cx : Ax = b,\ x \geqslant 0 \text{ integer}\}$$

with exponentially many variables. Apart from a solution method (these problems are mostly NP-hard), we might be interested in the size of an (optimum) solution. We give a bound which is polynomial in the number of rows in the matrix A and the maximum size of an entry in A. If all entries of A are positive, then a bound in terms of the size of the right-hand side b can be given. Finally, for the case when the columns of A are, for example, all vectors of a given polyhedron, we provide a bound which depends on the number of rows of A only, although exponential in it. As a consequence, we prove that the cutting stock problem is NP-complete.

7.1 Open problems

There are some rumours that any research can be represented as a directed rooted tree. A researcher starts with a problem at the root node, but its solution gives rise to a number of other problems. Whatever direction (s)he chooses, solution of the next problem creates several new problems again. The more problems you solve, the more open questions you get!

Among the problems that arose during the work on the present thesis, we would like to mention the following.

Problem 1. We discussed a polynomial-time algorithm for finding the integer programming gap for integer programs of the form

$$\max\{cx : Ax \leqslant b,\ x \text{ integer}\},$$

where the right-hand side varies over all vectors for which the integer program is feasible, when the number of variables is fixed. Integer programs in standard form with a fixed number of variables is a sub-case. But we can consider integer programs in standard form where the number of *rows* is fixed. Integer programming itself becomes NP-hard in this case, but there exists a pseudo-polynomial algorithm for it. Hence, the question: Can we compute the integer programming gap for these integer programs in pseudo-polynomial time?

Problem 2. It is known that

$$\min\{cx : Ax = b,\ x \geqslant 0 \text{ integer}\} = \lceil \min\{cx : Ax = b,\ x \geqslant 0\} \rceil$$

for all b for which the integer program is feasible if and only if the columns of the matrix

$$\begin{bmatrix} c & 1 \\ A & 0 \end{bmatrix}$$

form a Hilbert basis. Furthermore, in fixed dimension we can test in polynomial time whether given vectors constitute a Hilbert basis. But can we test, in polynomial time in fixed dimension, whether

$$\min\{cx : Ax = b,\ x \geqslant 0 \text{ integer}\} \leqslant \lceil \min\{cx : Ax = b,\ x \geqslant 0\} \rceil + 1$$

holds for all b for which the integer program is feasible?

Problem 3. The cutting stock problem is known to be NP-complete in general. Can it be solved in polynomial time under the assumption that the number of different items sizes is fixed?

Problem 4. What is the tight bound on the maximum integer programming gap for the cutting stock problem? As we mentioned, it was conjectured that $\mathrm{OPT}(I) - \lceil \mathrm{LIN}(I) \rceil \leqslant 1$ holds for all instances I. This conjecture needs to be proved (or disproved).

Bibliography

Banaszczyk, W., Litvak, A. E., Pajor, A., and Szarek, S. J. (1999). The flatness theorem for nonsymmetric convex bodies via the local theory of Banach spaces. *Mathematics of Operations Research*, 24(3):728–750.

Barvinok, A. I. (1994). A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Mathematics of Operations Research*, 19(4):769–779.

Barvinok, A. I. and Woods, K. M. (2003). Short rational generating functions for lattice point problems. *Journal of the American Mathematical Society*, 16(4):957–979.

Belov, G. and Weismantel, R. (2003). A class of subpattern formulations for one-dimensional stock cutting.

Bruns, W., Gubeladze, J. S., Henk, M., Martin, A., and Weismantel, R. (1999). A counterexample to an integer analogue of Carathéodory's theorem. *Journal für die reine und angewandte Mathematik*, 510:179–185.

Clarkson, K. L. (1995). Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499.

Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S. (1997). Approximation algorithms for bin packing: A survey. In Hochbaum, D. S. R., editor, *Approximation Algorithms for NP-Hard Problems*, chapter 2, pages 46–93. PWS Publishing Company, Boston.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York. The Association for Computing Machinery.

Cook, W. J., Fonlupt, J., and Schrijver, A. (1986). An integer analogue of Carathéodory's theorem. *Journal of Combinatorial Theory, Series B*, 40(1):63–70.

Cook, W. J., Hartmann, M. E., Kannan, R., and McDiarmid, C. J. H. (1992). On integer points in polyhedra. *Combinatorica*, 12(1):27–37.

Cook, W. J., Lovász, L., and Schrijver, A. (1984). A polynomial-time test for total dual integrality in fixed dimension. In Korte, B. H. and Ritter, K., editors, *Mathematical programming at Oberwolfach II*, volume 22 of *Mathematical Programming Study*. North-Holland, Amsterdam.

Dantzig, G. B. (1951). Maximization of a linear function subject to linear inequalities. In Koopmans, T. C., editor, *Activity Analysis of Production and Allocation—Proceedings of a Conference*, pages 339–347, New York. Wiley.

Eisemann, K. (1957). The trim problem. *Management Science*, 3(3):279–284.

Eisenbrand, F. (2003). Fast integer programming in fixed dimension. In Di Battista, G. and Zwick, U., editors, *Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, volume 2832 of *Lecture Notes in Computer Science*, pages 196–207, Budapest, Hungary, September 16–19.

Eisenbrand, F. and Shmonin, G. (2006). Carathéodory bounds for integer cones. *Operations Research Letters*, 34:564–568.

Eisenbrand, F. and Shmonin, G. (2007). Integer points in a parameterised polyhedron. Submitted to *Mathematics of Operations Research*.

Fernandez de la Vega, W. and Lueker, G. S. (1981). Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. W. H. Freeman, San Francisco.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Gilmore, P. C. and Gomory, R. E. (1963). A linear programming approach to the cutting-stock problem–Part II. *Operations Research*, 11(6):863–888.

Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197.

Grötschel, M., Lovász, L., and Schrijver, A. (1993). *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, Berlin.

Hartmann, M. E. (1989). *Cutting Planes and the Complexity of the Integer Hull*. PhD thesis, Department of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

Hoşten, S. and Sturmfels, B. (2003). Computing the integer programming gap. To appear in *Combinatorica*.

Johnson, D. S. (1973). *Near-Optimal Bin Packing Algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.

Kannan, R. (1990). Test sets for integer programs, $\forall\exists$ sentences. In Cook, W. J. and Seymour, P. D., editors, *Polyhedral Combinatorics*, volume 1 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 39–47. American Mathematical Society, Providence, RI. Proceedings of a workshop on polyhedral combinatorics, held in the Headquarters Plaza Hotel, Morristown, NJ, June 12–16, 1989.

Kannan, R. (1992). Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12(2):161–177.

Kannan, R. and Bachem, A. (1979). Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8(4):499–507.

Kannan, R. and Lovász, L. (1988). Covering minima and lattice-point-free convex bodies. *The Annals of Mathematics, 2nd Series*, 128(3):577–602.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396.

Karmarkar, N. and Karp, R. M. (1982). An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science, November 3–5, 1982, Chicago, Illinois*, pages 312–320, Los Angeles. IEEE Computer Society Press.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations, Proceedings of a Symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Center, Yorktown Heights, New York*, pages 85–103. Plenum Press, New York.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096.

Khinchin, A. Y. (1948). A quantitative formulation of Kronecker's theory of approximation. *Izvestiya Akademii Nauk SSSR. Seriya Matematicheskaya*, 12:113–122.

Lenstra, A. K., Lenstra, Jr., H. W., and Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534.

Lenstra, Jr., H. W. (1983). Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548.

Marcotte, O. (1985). The cutting stock problem and integer rounding. *Mathematical Programming*, 33(1):82–92.

Marcotte, O. (1986). An instance of the cutting stock problem for which the rounding property does not hold. *Operations Research Letters*, 4(5):239–243.

McCormick, S. T., Smallwood, S. R., and Spieksma, F. C. R. (2001). A polynomial algorithm for multiprocessor scheduling with two job lengths. *Mathematics of Operations Research*, 26(1):31–49.

Papadimitriou, C. H. (1981). On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA.

Rietz, J., Scheithauer, G., and Terno, J. (2002). Families of non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics*, 121(1–3):229–245.

Scheithauer, G. and Terno, J. (1997). Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *European Journal of Operational Research*, 20(2):93–100.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, Chichester. A Wiley-Interscience publication.

Sebő, A. (1990). Hilbert bases, Carathéodory's theorem and combinatorial optimization. In Kannan, R. and Pulleyblank, W. R., editors, *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, pages 431–455, Waterloo, Ontario. University of Waterloo Press.

Valério de Carvalho, J. M. V. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273.

Index