Chapter 9

Enumeration

The goal of this chapter is threefold. First we present a polynomial algorithm for integer programming in fixed dimension. This algorithm is based on elegant ideas such as basis reduction and the flatness theorem. Second we revisit branch-and-cut, the most successful approach in practice for a wide range of applications. In particular we address a number of implementation issues related to the enumerative aspects of branch-and-cut. Finally we present an approach for dealing with integer programs that have a high degree of symmetry.

9.1 Integer Programming in Fixed Dimension

The *integer feasibility problem* "Does there exist $x \in \mathbb{Z}^n$ such that $Ax \leq b$?" is NP-complete in general. However, Lenstra [256] showed that this problem can be solved in polynomial time when n is a fixed constant. The key idea is that one can find in polynomial time either an integer point in $P := \{x : Ax \leq b\}$, or a direction d in which the polyhedron is *flat*, meaning that there is at most a fixed number $k(n)$ of parallel hyperplanes $dx = \delta_1, \dots, dx = \delta_{k(n)}$ that contain all the integral points in P (if any). Applying this idea recursively to each of the $(n-1)$-dimensional polyhedra $P \cap \{x : dx = \delta_j\}$, the algorithm enumerates a fixed number of polyhedra overall, since n is fixed. It is not obvious that a flat direction always exists for a polyhedron that does not contain integral points and that it can be found in polynomial time. The proof is based on two ingredients,

M. Conforti et al., *Integer Programming*, Graduate Texts in Mathematics 271, DOI 10.1007/978-3-319-11008-0_9

Lovász's basis reduction algorithm and a result stating that for every full-dimensional polytope P on can compute in polynomial time an ellipsoid E such that $E \subset P \subset (n+1)E$.

9.1.1 Basis Reduction

A subset Λ of $\mathbb{R}^n$ is a *lattice* if there exists a basis $a^1, \dots, a^n$ of $\mathbb{R}^n$ such that $\Lambda = \{\sum_{i=1}^n \lambda_i a^i \, : \, \lambda \in \mathbb{Z}^n\}$. The set of vectors $a^1, \dots, a^n$ is called a *basis* of Λ, and Λ is said to be *generated by* the basis $a^1, \dots, a^n$. For example, the lattice generated by the n unit vectors of $\mathbb{R}^n$ is $\mathbb{Z}^n$.

For ease of notation, throughout the section we identify a basis $a^1, \dots, a^n$ with the square matrix $A = (a^1, \dots, a^n)$. Given a square nonsingular matrix A, when we say "the lattice generated by A," we refer to the lattice generated by the columns of A.

Theorem 9.1. *Two nonsingular square matrices A and B generate the same lattice if and only if there exists a unimodular matrix U such that $B = AU$.*

Proof. Assume $B = AU$ for some unimodular matrix U. Since U is integral, by definition every column of B is in the lattice generated by A. On the other hand, $A = BU^{-1}$ and, by Lemma 1.16, U^{-1} is unimodular. So every column of A is in the lattice generated by B. It follows that A and B generate the same lattice.

Conversely, assume that A and B generate the same lattice. Then $B = AX$ for some integral matrix X and $A = BY$ for some integral matrix Y. Setting $U = X$, it follows that $U^{-1} = Y$. Since U and U^{-1} are integral matrices, U is unimodular by Lemma 1.16. $\square$

Given a lattice $\Lambda \subseteq \mathbb{R}^n$, Theorem 9.1 implies that all the bases of Λ have the same determinant. This determinant is therefore called the *determinant of the lattice* Λ, and is denoted by $\det(\Lambda)$.

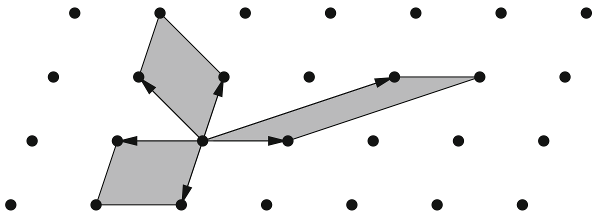


Figure 9.1: Three different bases in the plane; the three corresponding areas are equal

Any basis $B = (b^1, \dots, b^n)$ of the lattice Λ satisfies *Hadamard's inequality*:

$$\det(\Lambda) \le \|b^1\| \dots \|b^n\|, \tag{9.1}$$

where $\|.\|$ denotes the Euclidean norm. Indeed, a classical result in linear algebra states that $|\det(B)|$ is the volume of the parallelepiped in $\mathbb{R}^n$ generated by the n vectors $b^1, \dots, b^n$ (see [39] for example). Figure 9.1 illustrates this for three different bases of a lattice in the plane. Note that $\det(\Lambda) = \|b^1\| \dots \|b^n\|$ if and only if the vectors b^i are pairwise orthogonal. Not every lattice has an orthogonal basis. However Hermite [200] showed that there always exists a basis that is fairly orthogonal in the sense that

$$\|b^1\| \dots \|b^n\| \le c(n)\det(\Lambda) \tag{9.2}$$

where the *orthogonality defect* $c(n)$ is a constant that only depends on the dimension n but not on the lattice Λ. Hermite's result implies that (9.2) holds if we choose $c(n) \ge (\frac{4}{3})^{n(n-1)/4}$.

Lovász showed that, for rational bases B, if one chooses $c(n) = 2^{n(n-1)/4}$, one can actually compute such an approximation to an orthogonal basis in polynomial time. The algorithm, attributed to Lovász by Lenstra in [256], was presented in a paper of Lenstra, Lenstra and Lovász [255], and is sometimes called the *LLL algorithm*. In this book we will call it *Lovász' basis reduction algorithm* or simply the *basis reduction algorithm*. The main objective of this section is to present this algorithm.

Consider a lattice Λ generated by n linearly independent vectors in $\mathbb{R}^n$. Lovász introduced the notion of a reduced basis, using a Gram–Schmidt orthogonal basis as a reference. The *Gram–Schmidt procedure* is as follows. Starting from a basis $B = (b^1, \dots, b^n)$ of vectors in $\mathbb{R}^n$, define $g^1 := b^1$ and, recursively, for $j = 2, \dots, n$, define g^j as the projection of b^j onto the orthogonal complement of the vector space spanned by $b^1, \dots, b^{j-1}$. In other words,

$$\begin{aligned} &g^1 := b^1 \\ &g^j := b^j - \textstyle\sum_{k=1}^{j-1} \mu_{jk} g^k \quad \text{for } 2 \le j \le n \end{aligned} \tag{9.3}$$

where

$$\mu_{jk} := \frac{(b^j)^T g^k}{\|g^k\|^2} \qquad \text{for } 1 \le k < j \le n.$$

By construction, the *Gram–Schmidt basis* $G := (g^1, \dots, g^n)$ is an orthogonal basis of $\mathbb{R}^n$ with the property that, for $j = 1, \dots, n$, the vector spaces spanned by $b^1, \dots, b^j$ and by $g^1, \dots, g^j$ coincide. The coefficient μ_{jk}

is the kth coordinate of vector b^j relative to the orthogonal basis G. Note that $B = GR$ where $R = (r_{ij})_{i,j=1,\dots,n}$ is the upper-triangular matrix whose diagonal elements are all 1 and where $r_{ij} = \mu_{ji}$ for $1 \le i < j \le n$. In particular, since all diagonal entries of R are equal to 1, it follows that $\det(R) = 1$ and $\det(B) = \det(G)$. Note that, since the vectors $g^1, \dots, g^n$ are orthogonal, G^TG is the diagonal matrix with entries $\|g^1\|^2, \dots, \|g^n\|^2$. Since $\det(G^TG) = \det(G)^2$, we have $\|g^1\| \cdots \|g^n\| = |\det(G)| = |\det(B)| = \det(\Lambda)$. By construction, $\|b^j\| \ge \|g^j\|$ for $j = 1, \dots, n$, thus Hadamard's inequality (9.1) holds.

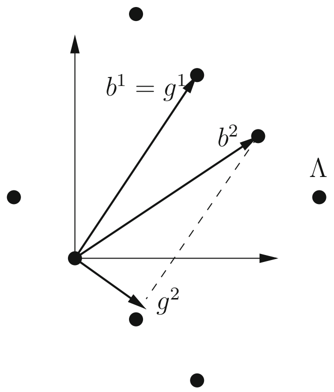


Figure 9.2: A basis (b^1, b^2) in $\mathbb{R}^2$ and its Gram–Schmidt orthogonal basis (g^1, g^2)

A basis $b^1, \dots, b^n$ of the lattice Λ is said to be *reduced* if it satisfies the following two conditions

$$\begin{array}{lll} (i) & |\mu_{jk}| \le \frac{1}{2} & \text{for } 1 \le k < j \le n, \\ (ii) & \|g^j + \mu_{j,j-1}g^{j-1}\|^2 \ge \frac{3}{4}\|g^{j-1}\|^2 & \text{for } 2 \le j \le n. \end{array} \tag{9.4}$$

where $g^1, \dots, g^n$ is the associated Gram–Schmidt orthogonal basis. Note that the basis (b^1, b^2) of Fig. 9.2 is not reduced because condition (i) is violated.

The next theorem shows that, if a basis of Λ is reduced, then it is "nearly orthogonal," in the sense that it satisfies (9.2) for $c(n) = 2^{n(n-1)/4}$. We will then explain Lovász' basis reduction algorithm, which given any basis B of Λ, produces a reduced basis in polynomial time.

Theorem 9.2. *Let $B = (b^1, \dots, b^n)$ be a square nonsingular matrix. If B is reduced, then $\|b^1\| \cdots \|b^n\| \le 2^{n(n-1)/4} \det(B)$.*

Proof. Let $g^1, \dots, g^n$ be the Gram–Schmidt orthogonalization of $b^1, \dots, b^n$. Since $g^1, \dots, g^n$ are pairwise orthogonal, it follows that $\|g^j + \mu_{j,j-1}g^{j-1}\|^2 = \|g^j\|^2 + |\mu_{j,j-1}|^2\|g^{j-1}\|^2$ for $j = 2, \dots, n$. Since B is reduced, from (9.4) and the above equation we have that

$$\|g^j\|^2 \geq (\frac{3}{4} - |\mu_{j,j-1}|^2)\|g^{j-1}\|^2 \geq \frac{1}{2}\|g^{j-1}\|^2$$

for $j = 2, \dots, n$. By induction, it follows that, for $1 \leq k < j \leq n$,

$$\|g^j\|^2 \geq 2^{k-j}\|g^k\|^2. \tag{9.5}$$

Furthermore, since $b^j = g^j + \sum_{k=1}^{j-1} \mu_{jk}g^k$, $j = 1, \dots, n$,

$$\begin{array}{rcll}
\|b^j\|^2 & = & \|g^j + \sum_{k=1}^{j-1} \mu_{jk}g^k\|^2 & \\
& = & \|g^j\|^2 + \sum_{k=1}^{j-1} |\mu_{jk}|^2\|g^k\|^2 & \text{(because } g^1, \dots, g^n \text{ are orthogonal)} \\
& \leq & \|g^j\|^2 + \frac{1}{4}\sum_{k=1}^{j-1} \|g^k\|^2 & \text{(because } |\mu_{jk}| \leq \frac{1}{2}) \\
& \leq & \|g^j\|^2(1 + \frac{1}{4}\sum_{k=1}^{j-1} 2^{j-k}) & \text{(by (9.5))} \\
& \leq & 2^{j-1}\|g^j\|^2 &
\end{array}$$

This implies $\prod_{j=1}^n \|b^j\| \leq \prod_{j=1}^n 2^{(j-1)/2}\|g^j\| = 2^{n(n-1)/4}\prod_{j=1}^n \|g^j\| = 2^{n(n-1)/4}\det(B)$. □

For $a \in \mathbb{R}$, let $\lfloor a \rceil$ denote the closest integer to a. Each iteration of the basis reduction algorithm consists of two steps, a "normalization step" and a "swapping step."

Lovász' Basis Reduction Algorithm

Input: A rational basis B of a lattice Λ.
Output: A reduced basis B of Λ.

Step 1. (Normalization)
For $j := 2$ to n and for $k := j-1$ down to 1, replace b^j by $b^j - \lfloor \mu_{jk} \rceil b^k$.

Step 2. (Swapping)
If Condition (9.4)(ii) holds, output the basis $b^1, \dots, b^n$ and stop.
Else, find an index j, $2 \leq j \leq n$, that violates Condition (9.4)(ii) and interchange vectors b^{j-1} and b^j in the basis.

Note that the operations involved in the algorithm are unimodular operations (see Sect. 1.5.2), thus they do not change the lattice generated by B. This follows from Lemma 1.16 and Theorem 9.1. Let us now analyze the effect of the normalization step.

Lemma 9.3. *The Gram–Schmidt basis remains unchanged in Step 1 of the basis reduction algorithm.*

Proof. Let $\tilde{B}$ be the basis obtained from B after Step 1. Since $\tilde{b}^j$ equals b^j plus a linear combination of $b^1, \dots, b^{j-1}$, it follows that $b^1, \dots, b^j$ and $\tilde{b}^1, \dots, \tilde{b}^j$ generate the same vector space L_j, $j = 1, \dots, n$, and that the projections of b^j and $\tilde{b}^j$ into the orthogonal complement of L_{j-1} are identical. This implies that the Gram–Schmidt basis associated with B and $\tilde{B}$ is the same. $\square$

Theorem 9.4. *When the basis reduction algorithm terminates, the basis $b^1, \dots, b^n$ is reduced.*

Proof. The algorithm stops in Step 2 when Condition (9.4)(ii) is satisfied. To show that the basis is reduced, it suffices to show that, at each iteration, at the end of Step 1 the current basis always satisfies Condition (9.4)(i).

By Lemma 9.3, the Gram–Schmidt basis $g^1, \dots, g^n$ remains unchanged in Step 1. Thus, if $\tilde{B}$ denotes the basis obtained from B after Step 1, we can express $\tilde{b}^j$ at the end of Step 1 as $\tilde{b}^j = g^j + \sum_{k=1}^{j-1} \tilde{\mu}_{jk} g^k$, $j = 1, \dots, n$. We claim that $|\tilde{\mu}_{jk}| \le \frac{1}{2}$, $1 \le k < j \le n$. Indeed, in a given iteration j^*, k^* in Step 1, among the several coefficients μ_{jk} that are modified, note that $\mu_{j^*k^*}$ is modified to $\mu_{j^*k^*} - \lfloor \mu_{j^*k^*} \rceil \le \frac{1}{2}$. Furthermore, this new coefficient $\mu_{j^*k^*}$ remains unchanged in all subsequent iterations, since they involve $k < k^*$ or $j > j^*$. $\square$

It is not obvious that this algorithm terminates, let alone that it terminates in polynomial time. Before giving the proof, we illustrate the algorithm on an example (Fig. 9.3).

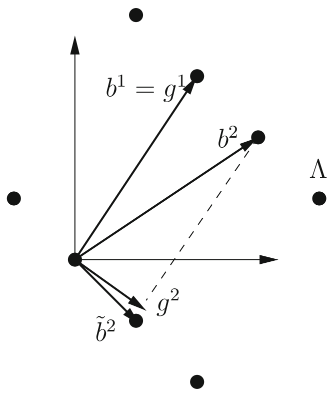


Figure 9.3: A basis (b^1, b^2) in $\mathbb{R}^2$ and a reduced basis $(\tilde{b}^2, b^1)$

Example 9.5. Consider the lattice Λ in $\mathbb{R}^2$ generated by the two vectors $b^1 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$ and $b^2 = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$. The Gram–Schmidt basis is $g^1 = b^1$ and $g^2 = b^2 - \mu_{21} g^1$. We compute $\mu_{21} = \frac{(b^2)^T g^1}{\|g^1\|^2} = \frac{12}{13}$. Since Condition (9.4)(i) is not satisfied we replace b^2 by $\tilde{b}^2 := b^2 - b^1$ (Step 1 of Lovász' algorithm). $\tilde{b}^2$ is shorter than b^2, as suggested by the name "reduced basis." For the new basis $(b^1, \tilde{b}^2)$ we have $\mu_{21} = \frac{-1}{13}$, therefore Condition (9.4)(i) is satisfied. However Condition (9.4)(ii) is violated: Since $\|\tilde{b}^2\|^2 = 2$ and $\|b^1\|^2 = 13$, we have $\|g^2 + \mu_{21} g^1\|^2 = \|\tilde{b}^2\|^2 < \frac{3}{4}\|b^1\|^2$. Hence we interchange b^1 and $\tilde{b}^2$ putting the shortest vector in front (Step 2 of Lovász's algorithm). The reader can check that now both Conditions (9.4)(i) and (ii) are satisfied. Therefore the basis $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $\begin{pmatrix} 2 \\ 3 \end{pmatrix}$ is a reduced basis for the lattice Λ. ■

Theorem 9.6. *Lovász' basis reduction algorithm terminates, and it runs in polynomial time in the input size of the original basis.*

Proof. We will prove that the algorithm terminates in a polynomial number of iterations. To prove that it runs in polynomial time, one should also show that the encoding size of the numbers remains polynomially bounded at every iteration; however we will not do it here (see [325] for a proof).

We may assume that B is an integral matrix. Indeed, given an integer number δ such that δB is integral, the basis reduction algorithm applied to δB executes the same iterations as for B, only the matrix produced at each iteration is multiplied by δ. Note that, if we apply the algorithm to an integral matrix, the basis remains integral at every iteration.

To prove finiteness of the algorithm, we use the following potential function associated with a basis $B = (b^1, \ldots, b^n)$, expressed in terms of the Gram–Schmidt basis associated with B:

$$\Phi(B) := \|g^1\|^{2n} \|g^2\|^{2n-2} \cdots \|g^n\|^2.$$

We observed that the Gram–Schmidt basis remains unchanged in Step 1 of the basis reduction algorithm (Lemma 9.3). Therefore the potential function Φ only changes in Step 2 when vectors b^j and b^{j-1} are interchanged.

Let B be the basis at a given iteration, and let us denote by $\tilde{B} = (\tilde{b}^1, \ldots, \tilde{b}^n)$ the basis obtained from B by interchanging b^j and b^{j-1}. We will prove that $\Phi(\tilde{B})/\Phi(B) \leq 3/4$ and that the potential is always integer. These two facts imply that the algorithm terminates after $O(\log \Phi(B))$ steps. In particular, the number of iterations is polynomial in the encoding size of the input matrix B. Indeed, if M is the largest absolute value of an entry of B, then $\Phi(B) \le \det(B)^{2n} \le (n^n M^n)^{2n} = (nM)^{2n^2}$, hence the number of iterations is $O(n^2 \log(nM))$.

Let $\tilde{g}^1, \dots, \tilde{g}^n$ be the Gram–Schmidt orthogonalization of $\tilde{B}$. Since $b^h = \tilde{b}^h$ for all $h \neq j-1, j$, $1 \le h \le n$, it follows that $g^h = \tilde{g}^h$ for all $h \neq j-1, j$. Furthermore, since $b^j = g^j + \sum_{k=1}^{j-1} \mu_{jk} g^k$, it follows that $g^j + \mu_{j,j-1} g^{j-1}$ is the projection of b^j onto the orthogonal complement of the space generated by $b^1, \dots, b^{j-2}$, thus $\tilde{g}^{j-1} = g^j + \mu_{j,j-1} g^{j-1}$. We also have that $\|g^1\| \cdots \|g^n\| = |\det(G)| = |\det(\tilde{G})| = \|\tilde{g}^1\| \cdots \|\tilde{g}^n\|$, therefore $\|g^{j-1}\| \|g^j\| = \|\tilde{g}^{j-1}\| \|\tilde{g}^j\|$. It then follows that

$$\begin{aligned}
\frac{\Phi(\tilde{B})}{\Phi(B)} &= \frac{\|\tilde{g}^{j-1}\|^{2(n-j+2)} \|\tilde{g}^j\|^{2(n-j+1)}}{\|g^{j-1}\|^{2(n-j+2)} \|g^j\|^{2(n-j+1)}} \\
&= \frac{(\|\tilde{g}^{j-1}\| \|\tilde{g}^j\|)^{2(n-j+1)}}{(\|g^{j-1}\| \|g^j\|)^{2(n-j+1)}} \cdot \frac{\|\tilde{g}^{j-1}\|^2}{\|g^{j-1}\|^2} \\
&= \frac{\|g^j + \mu_{j,j-1} g^{j-1}\|^2}{\|g^{j-1}\|^2} < \frac{3}{4},
\end{aligned}$$

where the last inequality follows from the fact that b^{j-1} and b^j are interchanged at Step 2 if they violate Condition (9.4)(ii).

We finally show that $\Phi(B)$ is an integer for every integral matrix B. Let us denote by B_i the matrix with columns $b^1, \dots, b^i$, and G_i the matrix with columns $g^1, \dots, g^i$, $i = 1, \dots, n$. By (9.3), $B_i = G_i R_i$, where R_i is an $i \times i$ upper-triangular matrix with all diagonal elements equal to 1. It follows that $\det(B_i^T B_i) = \det(G_i^T G_i) = \|g^1\|^2 \cdots \|g^i\|^2$, where the last equality follows from the fact that $g^1, \dots, g^n$ are pairwise orthogonal. This shows that $\|g^1\|^2 \cdots \|g^i\|^2$ is integer for $i = 1, \dots, n$. Thus $\Phi(B) = \|g^1\|^{2n} \|g^2\|^{2n-2} \cdots \|g^n\|^2 = \prod_{i=1}^n (\|g^1\|^2 \cdots \|g^i\|^2)$ is integer as well. $\square$

9.1.2 The Flatness Theorem and Rounding Polytopes

Let $K \subseteq \mathbb{R}^n$ be a *convex body*, that is, a closed bounded convex set. Given a vector $d \in \mathbb{R}^n$, we define the *width of K along d* to be

$$w_d(K) = \max_{x \in K} d^T x - \min_{x \in K} d^T x.$$

The *lattice width of K* is defined as the minimum width along any **integral** vector d, that is

$$w(K) = \min_{d \in \mathbb{Z}^n} w_d(K).$$

The fundamental result used in Lenstra's algorithm for integer programming in fixed dimension is Khinchine's *flatness theorem* [237]. The theorem states that any full-dimensional convex body, either contains an integral point, or is "fairly flat," in the sense that its lattice width is bounded by some constant that depends only on the dimension. Figure 9.4 shows that such a constant is greater than 2 in the plane (Hurkens [208] showed that $1+\frac{2}{\sqrt{3}} \approx 2.155$ is tight).

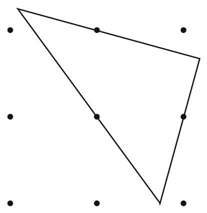


Figure 9.4: A lattice-free triangle with lattice width greater than 2

Theorem 9.7 (Flatness Theorem). *Let $K \subseteq \mathbb{R}^n$ be a full-dimensional convex body. If K does not contain any integral point, then $w(K) \le k(n)$, where $k(n)$ is a constant depending only on n.*

For rational polytopes, the proof will lead to a polynomial time algorithm that, for any full-dimensional polytope P expressed as a system of rational linear inequalities, outputs either an integral point in P or a *flat direction* for P, that is a vector $d \in \mathbb{Z}^n$ such that $w_d(P) \le n(n+1)2^{n(n-1)/4}$.

The first step is proving the flatness theorem for ellipsoids. An ellipsoid in $\mathbb{R}^n$ is an affine transformation of the unit ball. That is, an *ellipsoid centered at a* is a set of the form $E(C,a) = \{x \in \mathbb{R}^n : \|C(x-a)\| \le 1\}$, where $a \in \mathbb{R}^n$ and C is an $n \times n$ nonsingular matrix.

Theorem 9.8 (Flatness Theorem for Ellipsoids). *Let $E \subseteq \mathbb{R}^n$ be an ellipsoid. If E does not contain any integral point, then $w(E) \le n2^{n(n-1)/4}$.*

Proof. Let $a \in \mathbb{R}^n$ and $C \in \mathbb{R}^{n\times n}$ be a nonsingular matrix such that $E = E(C,a)$. For any $d \in \mathbb{R}^n$, we first compute $w_d(E)$. Let us view d as a row vector. We have $\max\{dx : \|C(x-a)\| \le 1\} = da + \max\{dC^{-1}y : \|y\| \le 1\} = da + \|dC^{-1}\|$, where we have applied the change of variables $y = C(x-a)$, and the maximum is achieved by $y = (dC^{-1})^T/\|dC^{-1}\|$. Therefore, for every $d \in \mathbb{R}^n$,

$$w_d(E) = \max_{x\in E} dx - \min_{x\in E} dx = 2\|dC^{-1}\|. \tag{9.6}$$

Let Λ be the lattice generated by C, and let B be a basis of Λ satisfying $\|b^1\| \cdots \|b^n\| \leq 2^{n(n-1)/4} |\det(B)|$. (While the existence of such a basis B is implied by the basis reduction algorithm when C is rational, the general case follows from a result of Hermite [200] mentioned in Sect. 9.1.1. See inequality (9.2).) Since $|\det(B)|$ is invariant under permuting the columns of B, we may assume that b^n is the element of maximum norm in B, that is $\|b^n\| = \max_{j=1,\ldots,n} \|b^j\|$.

Since C and B are bases of the same lattice, by Theorem 9.1 there exists a unimodular matrix U such that $C = BU$. Let $\lambda = Ua$, and define $\bar{x} = U^{-1}\lfloor\lambda\rceil$ where $\lfloor\lambda\rceil = (\lfloor\lambda_1\rceil, \ldots, \lfloor\lambda_n\rceil)$. Note that $\bar{x}$ is integral. Define the vector $d \in \mathbb{Z}^n$ to be the last row of U. We will show that, if $\bar{x} \notin E$, then $w_d(E) \leq n2^{n(n-1)/4}$. This will conclude the proof of the theorem.

Assume that, $\bar{x} \notin E$. Then $\|C(a - \bar{x})\| > 1$, that is, $\|B(\lambda - \lfloor\lambda\rceil)\| > 1$. Hence

$$1 < \|\sum_{j=1}^{n}(\lambda_j - \lfloor\lambda_j\rceil)b^j\| \leq \sum_{j=1}^{n} |\lambda_j - \lfloor\lambda_j\rceil| \, \|b^j\| \leq \frac{n}{2}\|b^n\|. \tag{9.7}$$

Consider the Gram–Schmidt orthogonal basis $g^1, \ldots, g^n$ obtained from $b^1, \ldots, b^n$. We have $|\det(B)| = \|g^1\| \cdots \|g^n\|$. Since $\|b^1\| \cdots \|b^n\| \leq 2^{n(n-1)/4}\| g^1\| \cdots \|g^n\|$ and $\|b^j\| \geq \|g^j\|$ for $j = 1, \ldots, n$, it follows that $\|b^n\| \leq 2^{n(n-1)/4}\|g^n\|$. The latter and (9.7) imply

$$\|g^n\| > 2/(n2^{n(n-1)/4}). \tag{9.8}$$

We now evaluate $w_d(E)$. Let us denote by v the last row of B^{-1}. Then $w_d(E) = 2\|dC^{-1}\| = 2\|v\|$, since $C^{-1} = U^{-1}B^{-1}$ and d is the last row of U. Since g^n is orthogonal to $g^1, \ldots, g^{n-1}$, it follows from (9.3) that $(g^n)^T b^j = 0$ for $j = 1, \ldots, n-1$ and $(g^n)^T b^n = \|g^n\|^2$. In particular, $v = (g^n)^T/\|g^n\|^2$. Thus, by (9.8), $w_d(E) = 2\|g^n\|/\|g^n\|^2 \leq n2^{n(n-1)/4}$. $\square$

Note that, when C and a are rational, the elements $\bar{x}$ and d defined in the proof of Theorem 9.8 can be computed in polynomial time, since this amounts to computing a reduced basis of Λ and solving systems of linear equations. The proof shows that $\bar{x} \in E(C, a)$ or $w_d(E(C, a)) \leq n2^{n(n-1)/4}$. This proves the following.

Remark 9.9. *There is a polynomial-time algorithm that, given $a \in \mathbb{Q}^n$ and a nonsingular matrix $C \in \mathbb{Q}^{n\times n}$, either finds an integral point in the ellipsoid $E(C, a)$, or finds a vector $d \in \mathbb{Z}^n$ such that $w_d(E(C, a)) \leq n2^{n(n-1)/4}$.*

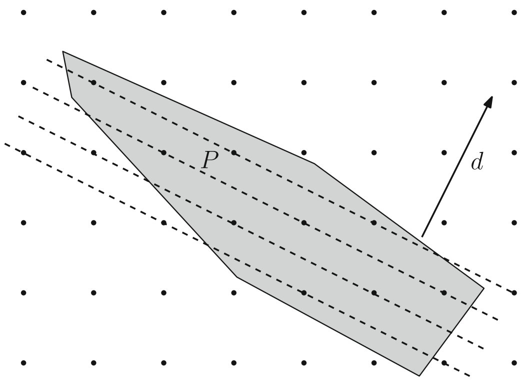


Figure 9.5: The *dashed lines* represent the hyperplanes $d^T x = k$ that intersect the polytope P (where $d = (1, 2)$ and k is an integer). Every integral point in P must lie in one of the four *dashed lines*

Proof of the Flatness Theorem (Theorem 9.7). The proof relies on the flatness theorem for ellipsoids and the following theorem of Löner (reported by Danzer, Grünbaum, and Klee [105]) and John [214].

> *For every full-dimensional convex body K, there exists an ellipsoid $E(C, a)$ such that $E(nC, a) \subseteq K \subseteq E(C, a)$.*

Note that $E(nC, a)$ is obtained by scaling $E(C, a)$ by $1/n$ around its center. Given such an ellipsoid $E(C, a)$, we know from Theorem 9.8 that either $E(nC, a)$ contains an integral point, and so does K because $E(nC, a) \subseteq K$, or there exists $d \in \mathbb{Z}^n$ such that $w_d(E(nC, a)) \leq n2^{n(n-1)/4}$. It then follows that $w_d(K) \leq w_d(E(C, a)) = nw_d(E(nC, a)) \leq n^2 2^{n(n-1)/4}$. The statement of Theorem 9.7 then follows if we choose $k(n) = n^2 2^{n(n-1)/4}$. $\square$

The scaling factor of $1/n$ in Löner and John's theorem is the best possible. A result of Goffin [174] implies that there exists a polynomial time algorithm which, given a full-dimensional polytope $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ expressed by a rational system of linear inequalities, computes an ellipsoid $E(C, a)$ such that $E((n+1)C, a) \subseteq P \subseteq E(C, a)$. This improves an earlier algorithm of Lenstra [256] guaranteeing a scaling factor of $\frac{1}{2}n^{-3/2}$. Following the proof of the flatness theorem, the above discussion and Remark 9.9 imply the following.

Theorem 9.10. *There exists a polynomial time algorithm that, given $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$ such that $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is a full-dimensional polytope, outputs either an integral point in P, or a vector $d \in \mathbb{Z}^n$ such that $w_d(P) \leq n(n+1)2^{n(n-1)/4}$.*

9.1.3 Lenstra's Algorithm

Theorem 9.10 leads immediately to the basic idea of Lenstra's algorithm: given a rational system $Ax \leq b$ in n variables, either find an integral point in $P := \{x \in \mathbb{R}^n : Ax \leq b\}$, or find a flat direction for P, namely a direction $d \in \mathbb{Z}^n$ such that $w_d(P) \leq n(n+1)2^{n(n-1)/4}$. Since d is integral, every point in $P \cap \mathbb{Z}^n$ must lie in one of the $(n-1)$-dimensional polytopes $P \cap \{x : d^T x = k\}$, for $k = \lceil \min_{x \in P} d^T x \rceil, \ldots, \lfloor \max_{x \in P} d^T x \rfloor$. Since there are at most $n(n+1)2^{n(n-1)/4} + 1$ such polytopes and n is a constant, we need to apply the algorithm recursively to a constant number of polytopes of lower dimension (see Fig. 9.5).

However, Theorem 9.10 applies to *full-dimensional*, *bounded* polyhedra. We need to address the following two technical issues: what to do if P is not bounded, and what to do if P is not full-dimensional (which will be the case at every iteration, since polytopes of the form $P \cap \{x : d^T x = k\}$ are not full-dimensional).

We first address the issue of non-boundedness of P. By Corollary 4.37, if we denote by L the maximum among the encoding sizes of the coefficients of (A, b), there exists an integer valued function f of n and L such that the encoding size of $f(n, L)$ is polynomially bounded by n and L and such that P has an integral point if and only if $P' := \{x \in P : -f(n, L) \leq x \leq f(n, L)\}$ has an integral point. Thus we only need to check if P' contains an integral point. We have therefore reduced to the case that the input system $Ax \leq b$ defines a polytope.

If P is not full-dimensional, then by Theorem 3.17 the system $Ax \leq b$ must include some implicit equality. An implicit equality can be determined in polynomial time by solving the m linear programs $\beta_i := \min\{a_i x : Ax \leq b\}$, $i = 1, \ldots, m$, where $a_1, \ldots, a_m$ are the rows of A, and checking if $\beta_i = b_i$. Thus, in polynomial time, we can determine a rational equation $\alpha x = \beta$ such that $P \subseteq \{x : \alpha x = \beta\}$. Possibly by multiplying the equation by the greatest denominators of the entries of α, we may assume that α is an integral vector with relatively prime entries. If β is not integer, then P does not contain any integral point and we are done. If β is integer, the next lemma shows how we can reduce to a problem with $n-1$ variables.

Lemma 9.11. *Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $\alpha \in \mathbb{Z}^n$, $\beta \in \mathbb{Z}$ be such that the entries of α are relatively prime. There exists a matrix $D \in \mathbb{Z}^{n \times (n-1)}$ and a vector $b' \in \mathbb{Q}^m$ such that the system $Ax \leq b$, $\alpha x = \beta$ has an integral solution if and only if the system $ADy \leq b'$ has an integral solution.*

Proof. Since all entries of α are relatively prime, by Corollary 1.9, $\alpha x = \beta$ has an integral solution $\bar{x}$, and there exists a unimodular matrix U such that $\alpha U = e^1$, where e^1 denotes the first unit vector in $\mathbb{R}^n$. If we define D as the $n \times (n-1)$ matrix formed by the last $n-1$ columns of U, we have that $\{x \in \mathbb{Z}^n : \alpha x = \beta\} = \{\bar{x} + Dy : y \in \mathbb{Z}^{n-1}\}$. Therefore, if we let $b' = b - A\bar{x}$, we have

$$\{x \in \mathbb{Z}^n : Ax \leq b, \alpha x = \beta\} = \{\bar{x} + Dy : ADy \leq b', y \in \mathbb{Z}^{n-1}\}. \tag{9.9}$$

In particular $Ax \leq b$, $\alpha x = \beta$ has an integral solution if and only if $ADy \leq b'$ has an integral solution. □

Note that the matrix D and vector b' in the statement of Lemma 9.11 can be computed in polynomial time by Remark 1.13. It thus follows from the above discussion that one can reduce to the case where P is full-dimensional. We are now ready to formally present Lenstra's algorithm.

Lenstra's Algorithm

Input: A matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $b \in \mathbb{Q}^m$ such that $P := \{x \in \mathbb{R}^n : Ax \leq b\}$ is a full-dimensional polytope.

Output: "Yes" if P contains an integral point, "No" otherwise.

```
Apply the algorithm of Theorem 9.10.

If the outcome is an integral point in $P$, then output "Yes."

If the outcome is a vector $d \in \mathbb{Z}^n$ such that $w_d(P) \leq n(n+1)2^{(n(n-1)/4)}$, do the following;

    If $\lceil \min_{x\in P} d^T x \rceil > \lfloor \max_{x\in P} d^T x \rfloor$, output "No."
    Else, for $k = \lceil \min_{x\in P} d^T x \rceil, \dots, \lfloor \max_{x\in P} d^T x \rfloor$,
        Compute a matrix $D \in \mathbb{Z}^{n\times(n-1)}$ and a vector $b' \in \mathbb{Q}^m$ such that the system $Ax \leq b$, $d^T x = k$ has an integral solution if and only if the system $ADy \leq b'$ has an integral solution.
        Apply Lenstra's algorithm recursively to the system $ADy \leq b'$.
```

Lenstra [256] observed that the above algorithm can be modified to solve mixed integer linear programming problems with a fixed number of integer variables (and arbitrarily many continuous variables) in polynomial time.

9.2 Implementing Branch-and-Cut

Lenstra's algorithm is of great theoretical interest for solving integer programs. However, in practice, the most successful solvers are currently based on the branch-and-cut approach introduced in Chap. 1. This is because many applications have a combinatorial flavor (a large number of 0,1 variables) rather than a number theoretic origin (a small number of interconnected variables that must take integer values, possibly in a wide range). The complexity of Lenstra's algorithm explodes as the number of variables exceeds a few dozens. On the other hand, integer programs with thousands of 0,1 variables are solved routinely in several application areas using software based on the branch-and-cut method.

In this section we return to branch-and-cut. Many implementation issues were left open in Chap. 1: branching strategy, node selection strategy, heuristics for getting feasible solutions, and many others. To better understand the range of questions that arise, consider a mixed integer linear program (MILP) and suppose that we just solved its linear programming relaxation; let $\bar{z}$ be the optimum value and $\bar{x}$ the optimal solution of this linear programming relaxation. What should one do if $\bar{x}_j$ is fractional for at least one of the variables that are required to be integer in the MILP that we are trying to solve? Should one generate cutting planes in the hope of improving the linear programming relaxation, or should one branch? When branching, should one use a strategy in the spirit of Lenstra's algorithm, creating smaller subproblems in parallel hyperplanes along a thin direction, or should one simply branch on one of the integer variables x_j for which $\bar{x}_j$ is fractional, setting $x_j \leq \lfloor \bar{x}_j \rfloor$ on one branch, and $x_j \geq \lceil \bar{x}_j \rceil$ on the other. In this case, which variable should we choose for branching? We should favor choices that help pruning the enumeration tree faster: infeasibility or integrality of the subproblems that we create, and pruning by bounds. The goal of pruning by bounds is best achieved by strategies that generate good upper and lower bounds on the optimal value of the MILP. To obtain these bounds, heuristics and cutting plane generation need to be integrated in the solver. These components are all interconnected. By now, the reader should have realized that building an efficient branch-and-cut solver is a sophisticated affair. Let us discuss some of the key issues.

Consider the MILP

$$\begin{aligned} z_I = \quad & \max cx \\ & Ax \leq b \\ & x \geq 0 \\ & x_j \text{ integer for } j = 1, \dots, p. \end{aligned} \tag{9.10}$$

The data are a vector $c \in \mathbb{Q}^n$, an $m \times n$ rational matrix A, a vector $b \in \mathbb{Q}^m$, and an integer p such that $1 \leq p \leq n$. The set $I := \{1, \ldots, p\}$ indexes the integer variables whereas the set $C := \{p+1, \ldots, n\}$ indexes the continuous variables.

The branch-and-cut algorithm keeps a list of linear programming problems obtained by relaxing the integrality requirements on the variables and imposing linear constraints such as bounds on the variables: $x_j \leq u_j$ or $x_j \geq l_j$, and/or cutting planes. Each such linear program corresponds to a *node* of the branch-and-cut tree. For a node N_i, let z_i denote the value of the corresponding linear program LP_i. Let $\mathcal{L}$ denote the list of nodes that must still be solved (i.e., that have not been pruned nor branched on). Initially, $\mathcal{L}$ just contains node N_0 corresponding to (MILP). Let $\underline{z}$ denote a lower bound on the optimum value z_I (initially, the bound $\underline{z}$ can be derived from a heuristic solution of (MILP), or it can be set to $-\infty$).

Branch-and-Cut Algorithm

0. **Initialize**

   $\mathcal{L} = \{N_0\}$, $\underline{z} = -\infty$, $x^* = \emptyset$. One may also apply preprocessing (to improve the formulation LP$_0$) and heuristics (to improve $\underline{z}$ and to obtain a feasible solution x^*).

1. **Terminate?**

   If $\mathcal{L} = \emptyset$, the solution x^* is optimal (If $x^* = \emptyset$, (MILP) is infeasible or unbounded).

2. **Select node**

   Choose a node N_i in $\mathcal{L}$ and delete it from $\mathcal{L}$.

3. **Bound**

   Solve LP$_i$. If it is unbounded, (MILP) is unbounded, stop. If it is infeasible, go to Step 1. Else, let x^i be an optimal solution of LP$_i$ and z_i its objective value.

4. **Prune**

   If $z_i \leq \underline{z}$, go to Step 1.

   If x^i is feasible to (MILP), let $\underline{z} = z_i$, $x^* = x^i$. Delete from $\mathcal{L}$ any node N_j for which a bound z_j is known and satisfies $z_j \leq \underline{z}$. Go to Step 1.

   If x^i is not feasible to (MILP), go to Step 5.

5. **Add Cuts?**

   Decide whether to strengthen the formulation LP_i or to branch.

   In the first case, strengthen LP_i by adding cutting planes and go back to Step 3.

   In the second case, go to Step 6.

6. **Branch**

   From LP_i, construct $k \geq 2$ linear programs $\mathrm{LP}_{i_1}, \dots, \mathrm{LP}_{i_k}$ with smaller feasible regions whose union does not contain (x^i, y^i), but contains all the solutions of LP_i with $x \in \mathbb{Z}^n$. Add the corresponding new nodes $N_{i_1}, \dots, N_{i_k}$ to $\mathcal{L}$ and go to Step 1.

Various choices have been left open in this algorithm. In particular, five issues need special attention when solving integer programs by branch-and-cut.

- Preprocessing (to decrease the size of the instance whenever possible),
- Heuristics (to find a good lower bound $\underline{z}$ on z_I),
- Cutting plane generation (to reduce the upper bound $\bar{z}$ obtained when solving the linear programming relaxation),
- Branching,
- Node selection.

We discuss branching strategies first, followed by node selection strategies, heuristics, preprocessing, and cutting plane generation.

Branching

Although Step 6 of the branch-and-cut algorithm provides the flexibility of branching into any number $k \geq 2$ of subproblems, the most popular choice is to generate $k = 2$ disjoint subproblems (Exercise 9.14 helps to understand why this is a good strategy). A natural way of generating two disjoint subproblems is by using a split disjunction $\sum_{j=1}^{p} \pi_j x_j \leq \pi_0$ or $\sum_{j=1}^{p} \pi_j x_j \geq \pi_0 + 1$ where $(\pi, \pi_0) \in \mathbb{Z}^{p+1}$ is chosen such that x^i satisfies $\pi_0 < \sum_{j=1}^{p} \pi_j x^i_j < \pi_0 + 1$. The simplest such disjunction is obtained by choosing π to be a unit vector. This branching strategy is called *variable branching* and it is by far the most widely used in integer programming solvers.

Specifically, let x^i_j be one of the fractional values for $j = 1, \ldots, p$, in the optimal solution x^i of LP$_i$ (we know that there is such a j, since otherwise N_i would have been pruned in Step 4 on account of x^i being feasible to (MILP)). From problem LP$_i$, we can construct two linear programs LP$^-_{ij}$ and LP$^+_{ij}$ that satisfy the requirements of Step 6 by adding the constraints $x_j \le \lfloor x^i_j \rfloor$ and $x_j \ge \lceil x^i_j \rceil$ respectively to LP$_i$. An advantage of variable branching is that the number of constraints in the linear programs does not increase, since linear programming solvers treat bounds on variables implicitly.

An important question is: On which variable x_j should the algorithm branch, among the $j = 1, \ldots, p$ such that x^i_j is fractional? To answer this question, it would be very helpful to know the decrease D^-_{ij} in objective value between LP$_i$ and LP$^-_{ij}$, and D^+_{ij} between LP$_i$ and LP$^+_{ij}$. A good branching variable x_j at node N_i is one for which both D^-_{ij} and D^+_{ij} are relatively large (thus tightening the upper bound z_i, which is useful for pruning). For example, a reasonable strategy is to choose j such that the product $D^-_{ij} \times D^+_{ij}$ is the largest.

The strategy which consists of computing D^-_{ij} and D^+_{ij} explicitly for each j is called *strong branching*. It involves solving linear programs that are small variations of LP$_i$ by performing dual simplex pivots, for each $j = 1, \ldots, p$ such that x^i_j is fractional. Experiments indicate that strong branching reduces the size of the enumeration tree by one or more orders of magnitude in most cases, relative to a simple branching rule such as branching on the most fractional variable. Thus there is a clear benefit to spending time on strong branching. But the computing time of doing it at each node N_i, for every fractional variable x^i_j, may be too high. This suggests the following idea, based on the notion of *pseudocosts* that are initialized at the root node and then updated throughout the branch-and-bound tree.

Let $f^i_j = x^i_j - \lfloor x^i_j \rfloor$ be the fractional part of x^i_j, for $j = 1, \ldots p$. For an index j such that $f^i_j > 0$, define the *down pseudocost* and *up pseudocost* as

$$P^-_j = \frac{D^-_{ij}}{f^i_j} \quad \text{and} \quad P^+_j = \frac{D^+_{ij}}{1 - f^i_j}$$

respectively. Benichou et al. [48] observed that the pseudocosts tend to remain fairly constant throughout the branch-and-bound tree. Therefore the pseudocosts need not be computed at each node of the tree. They can be estimated instead. How are they initialized and how are they updated in the tree? A good way of initializing the pseudocosts is through strong branching at the root node or other nodes of the tree when new variables become fractional for the first time (or the first r times, where $r \geq 2$, to get more reliable initial estimates). To update the estimate $\hat{P}_j^-$ of the pseudocost P_j^-, the algorithm averages the observations $\frac{D_{ij}^-}{f_j^i}$ over all the nodes N_i of the tree in which x_j was branched on or in which D_{ij}^- was computed through strong branching. Similarly for the estimate $\hat{P}_j^+$ of the up pseudocost. The decision of which variable to branch on at the current node N_i of the tree is then made based on these estimated pseudocosts $\hat{P}_j^-$ and $\hat{P}_j^+$ as follows. For each $j = 1, \dots, p$ such that $f_j^i > 0$, compute estimates of D_{ij}^- and D_{ij}^+ by the formula $\hat{D}_{ij}^- = \hat{P}_j^- f_j^i$ and $\hat{D}_{ij}^+ = \hat{P}_j^+(1 - f_j^i)$. Branch on the variable x_j with the largest value of the product $\hat{D}_{ij}^- \times \hat{D}_{ij}^+$.

Variable branching is not the only branching strategy that is implemented in state-of-the-art solvers. Many integer programs contain constraints of the form

$$\sum_{t=1}^{k} x_{j_t} = 1$$

with $x_{j_t} = 0$ or 1, for $t = 1, \dots k$. If one or more of these variables is fractional in the current solution x^i, variable branching would pick one such variable, say x_{j^*}, and set $x_{j^*} = 0$ on one branch and $x_{j^*} = 1$ on the other. This leads to unbalanced trees since only one variable is fixed on the first branch but all variables x_{j_t}, $t = 1, \dots k$, are fixed on the other. A strategy that better balances the tree is to partition the set $\{j_1, \dots, j_k\}$ into $J' \cup J''$ so that both $\sum_{j_t \in J'} x_{j_t}^i > 0$ and $\sum_{j_t \in J''} x_{j_t}^i > 0$ and the cardinalities of J' and J'' are roughly the same. One then fixes $x_{j_i} = 0$ for all $j_i \in J'$ on one branch and $x_{j_i} = 0$ for all $j_i \in J''$ on the other. This branching scheme is known to reduce the size of the enumeration tree significantly in practice. It is often called GUB branching (GUB stands for Generalized Upper Bound). SOS branching is a variation on this idea (SOS stands for Special Ordered Sets).

Node Selection

How does one choose among the different problems N_i available in Step 2 of the algorithm? Two goals need to be considered: finding a better feasible solution (thus increasing the lower bound $\underline{z}$) and proving optimality of the current best feasible solution (by decreasing the upper bound as quickly as possible).

For the first goal, we estimate the value of the best feasible solution in each node N_i. For example, we could use the following estimate:

$$E_i = z_i - \sum_{j=1}^{p} \min(\hat{P}_j^- f_j^i, \hat{P}_j^+ (1 - f_j^i))$$

based on the pseudocost estimates defined earlier. This corresponds to rounding the noninteger solution x^i to a nearby integer solution and using the pseudocosts to estimate the degradation in objective value. We then select a node N_i with the largest E_i. This is the so-called best estimate criterion node selection strategy. A good feasible solution may be found by "diving" from node N_i, computing estimates E_j for its possible sons N_j, selecting a son with the largest estimate, and then repeating from the selected node N_j. Diving heuristics will be revisited in the next section.

For the second goal, the best strategy depends on whether the first goal has been achieved already. If we currently have a very good lower bound $\underline{z}$, it is reasonable to adopt a depth-first search strategy. This is because the linear programs encountered in a depth-first search are small variations of one another. As a result they can be solved faster in sequence, using the dual simplex method initialized with the optimal solution of the father node (they are solved about ten times faster, based on empirical evidence). On the other hand, if no good lower bound is available, depth-first search tends to be wasteful: it might explore many nodes N_i with a value z_i smaller than the optimum z_I. Assuming that we have a good lower bound $\underline{z}$, and that we adopt a depth-first search strategy whenever possible, what should we do when we reach a node of the tree that can be pruned? An alternate to the "best estimate criterion" is the "best bound" node selection strategy, which consists in picking a node N_i with the largest bound z_i. No matter how good a solution of (MILP) is found in other nodes of the branch-and-bound tree, the node with the largest bound z_i cannot be pruned by bounds (assuming no ties) and therefore it will have to be explored eventually. So we might as well explore it first.

The most successful node selection strategy may differ depending on the application. For this reason, most integer programming solvers have several node selection strategies available as options to the user. The default strategy is usually a combination of the "best estimate criterion" (or a variation) and depth-first search. Specifically, the algorithm may dive using depth-first search until it reaches an infeasible node N_i or it finds a feasible solution of (MILP). At this point, the next node might be chosen using the "best estimate criterion" strategy, and so on, alternating between dives in a depth-first search fashion to get feasible solutions at the bottom of the tree and the "best estimate criterion" to select the next most promising node.

Heuristics

Several types of heuristics are routinely applied in integer programming solvers. Heuristics help to improve the bound $\underline{z}$, which is used in Step 4 for pruning the enumeration tree. Heuristic solutions are even more important when the branch-and-cut algorithm has to be terminated before completion, returning a solution of value $\underline{z}$ without a proof of its optimality. We present two successful ideas, *diving* and *local branching*. Each can be applied at any node N_i of the branch-and-cut tree and can be implemented in many different variations. In some applications, even finding a feasible solution might be an issue, in which case heuristics such as the *feasibility pump* are essential.

Diving heuristics can be viewed as depth-first searches in the context of the node selection strategy presented above. One chooses an integer variable x_j that is fractional in the current linear programming solution $\bar{x}$ and adds the constraint $x_j \leq \lfloor \bar{x}_j \rfloor$ or $x_j \geq \lceil \bar{x}_j \rceil$; one then solves the new linear program; the process is repeated until a solution of (MILP) is found or infeasibility is reached. Solvers usually contain several different heuristic rules for choosing the variable x_j and the constraint to add in this procedure. One option is to choose a variable with smallest fractionality $\min(\bar{f}_j, 1-\bar{f}_j)$ among the integer variables x_j that have nonzero fractionality at $\bar{x}$, where $\bar{f}_j = \bar{x}_j - \lfloor \bar{x}_j \rfloor$, and to add the constraint $x_j \leq \lfloor \bar{x}_j \rfloor$ if $\bar{f}_j < \frac{1}{2}$, and $x_j \geq \lceil \bar{x}_j \rceil$ otherwise. Other more sophisticated rules use strong branching or pseudocost information to choose the variable x_j to branch on, and move down the branch that corresponds to the largest estimate of the objective value of the integer program at the children nodes, using the function E_i introduced earlier. Diving heuristics can be repeated from a variety of starting points N_i to improve the chance of getting good solutions.

Once a feasible solution x^* is available, *local branching* [140] can be applied to try to improve upon it. For simplicity of exposition, assume that all the integer variables are 0,1 valued. The idea is to define a neighborhood of x^* as follows:

$$\sum_{j=1}^{p} |x_j - x_j^*| \leq k$$

where k is an integer chosen by the user (for example $k = 20$ seems to work well), to then add this constraint to (MILP) and to apply your favorite integer programming solver. Instead of getting lost in a huge enumeration tree, the search is restricted to the neighborhood of x^* by this constraint. Note that the constraint should be linearized before adding it to the formulation, which is easy to do:

$$\sum_{j \in I : x_j^* = 0} x_j + \sum_{j \in I : x_j^* = 1} (1 - x_j) \leq k.$$

If a better solution than x^* is found, the neighborhood is redefined relatively to this new solution, and the procedure is repeated until no better solution is found. This heuristic can be modified into appealing variations, by changing the definition of the neighborhood. One such example is called RINS (which stands for relaxation induced neighborhood search) [101]. RINS needs a feasible solution x^* and a solution $\bar{x}$ of some linear program in the branch-and-cut tree. It fixes the variables that have the same value in x^* and $\bar{x}$. The resulting smaller integer program is then processed by the integer programming solver (for a limited time or a limited number of nodes) in the hope of finding a better feasible solution than x^*. This recursive use of the integer programming solver is a clever feature of local branching heuristics.

For some instances of (MILP), just finding a feasible solution might be an issue. Heuristics such as the *feasibility pump* are specifically designed for this purpose [139]. The main idea is to construct two sequences of points that hopefully converge to a feasible solution of (MILP). Let P denote the polyhedron defined by the linear constraints of (MILP). One sequence consists of points in P, possibly integer infeasible, the other one of integer feasible points, but possibly not in P. These two sequences are produced by alternately rounding a point $\bar{x}^i \in P$ to an integer feasible point x^i, and finding a point $\bar{x}^{i+1}$ in the polyhedron P that is closest to x^i (using the ℓ_1-norm) by solving a linear program. All current integer programming solvers incorporate some variant of this basic idea (see [6] for an improved feasibility pump).

Preprocessing

Integer programming solvers try to tighten and simplify the formulation before launching into a full-fledged branch-and-cut. These preprocessing steps can also be performed when subproblems are created in the course of the enumeration process. They involve simple cleaning operations: identifying infeasibilities or redundancies in the constraints, improving bounds on the variables, improving constraint coefficients, fixing variables, and identifying logical implications. We give a sample of such steps. Let

$$\sum_{j \in B} a_j x_j + \sum_{j \in C} g_j y_j \leq b \tag{9.11}$$

be a constraint of the integer programming formulation where B denotes the set of 0,1 variables and C denotes the set of remaining variables (both the continuous variables and the general integer variables). Let $B_+ := \{j \in B : a_j > 0\}$, $B_- := \{j \in B : a_j < 0\}$, $C_+ := \{j \in C : g_j > 0\}$ and $C_- := \{j \in C : g_j < 0\}$. Assume that the variables in C are bounded:

$$\ell_j \le y_j \le u_j \quad \text{for } j \in C.$$

The smallest and largest values that the LHS of (9.11) can take are, respectively,

$$L_{\min} := \sum_{j \in B_-} a_j + \sum_{j \in C_-} g_j u_j + \sum_{j \in C_+} g_j l_j$$

$$L_{\max} := \sum_{j \in B_+} a_j + \sum_{j \in C_-} g_j l_j + \sum_{j \in C_+} g_j u_j.$$

We can now perform the following checks:

- If $L_{\min} > b$, infeasibility has been identified,
- If $L_{\max} \le b$, redundancy has been identified,
- If $u_k > \frac{b - L_{\min} + g_k \ell_k}{g_k}$ for some $k \in C_+$, then the bound u_k can be improved to the RHS value,
- If $l_k < \frac{b - L_{\min} + g_k u_k}{g_k}$ for some $k \in C_-$, then the bound ℓ_k can be improved to the RHS value,
- If $L_{\min} + a_k > b$ for some $k \in B_+$, then $x_k = 0$ (variable fixing),
- If $L_{\min} - a_k > b$ for some $k \in B_-$, then $x_k = 1$ (variable fixing),
- If $a_k > L_{\max} - b$, for some $k \in B_+$, then the constraint coefficient a_k and the RHS b can both be reduced by $a_k - (L_{\max} - b)$,
- If $a_k < b - L_{\max}$, for some $k \in B_-$, then the constraint coefficient a_k can be increased to the RHS value $b - L_{\max}$.

Performing these checks on all the constraints in the formulation only takes linear time. Whenever some improvement is discovered, an additional pass through the constraints is performed. Other standard preprocessing steps are typically applied, such as reduced cost fixing (Exercise 9.18), implication based fixing, and identification of clique inequalities (Exercise 9.19). We refer the reader to the presentation of Savelsbergh [321] for additional details. Preprocessing is surprisingly effective in practice as can be seen from the table given in Sect. 5.3.

Cut Pool

When a node N_i is explored, cuts may be generated to strengthen the formulation, thus improving the bound z_i. Some cuts may be local (i.e., valid only at node N_i and its descendants) or global (valid at all the nodes of the branch-and-bound tree). A good strategy is to store cuts in a cut pool instead of adding them permanently to the formulation. The reason is that solving the linear programs tends to be slowed significantly when a large number of cuts are present. Thus, if a cut has become inactive in the linear program at a node N_i, it is moved to the cut pool. At later stages, the algorithm searches the cut pool for those inequalities that are violated by the current fractional point $\bar{x}$, and moves these cuts back into the linear program.

Software

Currently, Gurobi, Cplex, and Xpress are excellent commercial branch-and-cut codes. `Scip` and `cbc` are open source.

9.3 Dealing with Symmetries

Some problems in integer programming have a highly symmetric structure. This means that there are ways of permuting the variables and constraints that leave the formulation invariant. This implies that the feasible region of the linear programming relaxation is also symmetric. This can be a major issue in branch-and-cut algorithms, since they may end up solving many subproblems that are essentially identical. For example, in the operating room scheduling problem (2.17) in Sect. 2.8 with n identical operating rooms, permuting the indices in $\{1, \dots, n\}$ does not change the structure of the constraints. So, for each solution y of the linear programming relaxation, there are $n!$ equivalent solutions, one for each permutation π, obtained by setting $y'_{ij} = y_{i\pi(j)}$.

Various approaches have been tried to break the symmetry in branch-and-cut, such as perturbing the coefficients in the formulation or adding symmetry breaking constraints. The most successful approach to date is to ensure that only one isomorphic copy of each node is kept in the enumeration tree. A way of efficiently constructing such a tree was presented by Margot [267] in the context of integer programming. We will present some of these ideas in this section. To simplify the treatment, we will only present it in the context of pure 0, 1 programs.

Before doing this we need to introduce some basic terminology about permutations. A *permutation* on n-elements is a bijection $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$. We denote by Σ_n the set of all permutations on n elements.

We will represent π by the n-vector $(\pi(1), \dots, \pi(n))$. Given $S \subseteq \{1, \dots, n\}$, we denote by $\pi(S) = \{\pi(i) : i \in S\}$ the image of S under π. Given $v \in \mathbb{R}^n$, we denote by $\pi(v)$ the vector $(v_{\pi(1)}, \dots, v_{\pi(n)})$. Given a matrix $A \in \mathbb{R}^{m \times n}$ and permutations $\pi \in \Sigma_n$, $\sigma \in \Sigma_m$, we denote by $A(\pi, \sigma)$ the matrix obtained by permuting the columns of A according to π and the rows according to σ.

Consider a pure $0, 1$ linear program (BIP)

$$\begin{array}{ll} \max & cx \\ & Ax \leq b \\ & x \in \{0,1\}^n. \end{array} \tag{9.12}$$

The set of permutations

$$\Gamma = \{\pi \in \Sigma_n : \exists\, \sigma \in \Sigma_m \text{ such that } \pi(c) = c,\ \sigma(b) = b,\ A(\pi, \sigma) = A\} \tag{9.13}$$

is called the *symmetry group of (BIP)*. It is not difficult to show that Γ is indeed a group, that is, (i) the identity is in Γ, (ii) if $\pi \in \Gamma$, then $\pi^{-1} \in \Gamma$, (iii) if $\pi^1, \pi^2 \in \Gamma$, then $\pi^1 \circ \pi^2 \in \Gamma$. Note that, for every feasible solution $\bar{x}$ of the linear programming relaxation of (9.12), and for every $\pi \in \Gamma$, $\pi(\bar{x})$ is also feasible and it has the same objective value. The vectors $\bar{x}$ and $\pi(\bar{x})$ are said to be *isomorphic solutions*. Observe that the definition of Γ depends on the choice of formulation, and not only on the of geometry of the linear programming relaxation. For example, multiplying a constraint by a positive number, adding or removing redundant constraints may change the symmetry group.

Isomorphism Pruning

Consider a branch-and-bound algorithm for solving (BIP), in which we perform branching on the variables. At any node N_a of the enumeration tree, let F_a^0 and F_a^1 be the set of indices of the variables that have been fixed to 0 and to 1 respectively. Two nodes N_a and N_b of the enumeration tree are *isomorphic* if there exists some permutation $\pi \in \Gamma$ in the symmetry group of (BIP) such that $\pi(F_b^1) = F_a^1$ and $\pi(F_b^0) = F_a^0$. We are interested in strategies that avoid enumerating isomorphic subproblems. One obvious possibility is to check, every time a new node is generated, if there is an isomorphic node already in the tree, but this is computationally impractical because checking isomorphism is too expensive to be performed multiple times at every node.

We now present *isomorphism pruning*, which can be performed locally at each node without explicitly checking isomorphism to other nodes of the tree. Despite its simplicity, isomorphism pruning guarantees that we never solve two isomorphic subproblems.

Let N_a be a node of the enumeration tree, and let $p_1, \dots, p_\ell$ be the sequence of indices of the variables that have been branched on, on the path from the root to node N_a.

Isomorphism Pruning Rule. *If there exists $\pi \in \Gamma$ and $t \in \{1, \dots, \ell\}$ such that*

$$\begin{array}{l} p_i \in F_a^1 \Rightarrow \pi(p_i) \in F_a^1, \quad i = 1, \dots, t-1; \\ p_t \in F_a^0,\ \pi(p_t) \in F_a^1; \end{array} \tag{9.14}$$

then prune node N_a.

Example 9.12. Consider a 0,1 linear program (BIP) with three variables and assume that $\pi = (3, 1, 2)$ belongs to the symmetry group of (BIP). This implies that, whenever $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$ is feasible to (BIP), $(\bar{x}_3, \bar{x}_1, \bar{x}_2)$ is also feasible to (BIP) and has the same objective value. Consider the enumeration tree of Fig. 9.6. We can apply the isomorphic pruning rule with the above permutation π and $t = 2$: indeed $1 \in F_a^1$ and $\pi(1) = 3 \in F_a^1$; and $2 \in F_a^0$ and $\pi(2) = 1 \in F_a^1$. The isomorphic pruning rule tells us to prune node N_a. Note that it makes sense in this case because node N_b must still be solved and it contains the solution $(1, 1, 0)$ which is isomorphic to the solution $(1, 0, 1)$ that we are pruning in node N_a. ■

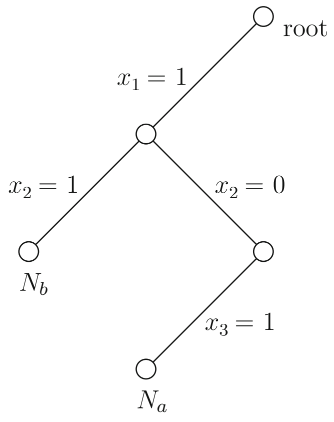


Figure 9.6: Example of isomorphic pruning

Next we explain why the isomorphism pruning rule ensures that we never consider isomorphic subproblems, and that we always keep at least one optimal solution of (BIP). We assume that, when branching on a variable x_k at a given node in the enumeration tree, we create two children, the *left child* obtained by fixing $x_k = 1$, and the *right child* obtained by fixing $x_k = 0$. Given two nodes N_a and N_b in the enumeration tree, we say that N_b is *to the left* of N_a if N_b is a descendant of the left child of the common ancestor N_d of N_a and N_b, and N_a is a descendant of the right child of N_d. We say that a vector $\bar{x}$ is a *solution in node* N_a if it is feasible for (BIP) and $\bar{x}_i = 1$ for all $i \in F^1_a$, $\bar{x}_i = 0$ for all $i \in F^0_a$ (in particular $\bar{x}$ must be a $0, 1$ vector).

Proposition 9.13. *If node N_a has an isomorphic node to its left in the enumeration tree, then node N_a is pruned by isomorphism.*

Conversely, if node N_a is pruned by isomorphism, then for every solution $\bar{x}$ in node N_a there is a node N_b to the left of node N_a containing a solution isomorphic to $\bar{x}$.

Proof. Suppose N_a has an isomorphic node N_b to its left. Then there exists a permutation $\pi \in \Gamma$ such that $\pi(F^1_b) = F^1_a$ and $\pi(F^0_b) = F^0_a$. Let N_d be the common ancestor of N_a and N_b and p_t be the index of the branching variable at node N_d. Thus, for $i = 1, \dots, t-1$, if $p_i \in F^1_a$, then $p_i \in F^1_b$, because the paths from the root to N_a and N_b coincide up to node N_d. By definition of π, this implies that $\pi(p_i) \in F^1_a$ for $i = 1, \dots, t-1$. Since N_b is a descendant of the left child of N_d, and N_a is a descendant of the right child of N_d, it follows that $p_t \in F^0_a$ and $p_t \in F^1_b$. Again by definition of π, this implies that $\pi(p_t) \in F^1_a$. This shows that π and t satisfy the conditions (9.14). Therefore N_a is pruned by isomorphism.

For the second part of the statement, assume that N_a is pruned by isomorphism. Let π and t satisfy conditions (9.14). Given a solution $\bar{x}$ in N_a, let $\tilde{x} = \pi(\bar{x})$. Consider the minimum index k such that $\bar{x}_{p_k} \neq \tilde{x}_{p_k}$. It follows from the conditions (9.14) that $k \leq t$ and $\bar{x}_{p_k} = 0$, $\tilde{x}_{p_k} = 1$. Thus, if N_d is the kth node on the path from the root to N_a (that is, N_d is the node where we branched on x_{p_k}), it follows that $\tilde{x}$ is feasible for the left child N_b of N_d. Hence $\bar{x}$ is isomorphic to a feasible solution of N_b, which is a node to the left of N_a. □

The above proposition shows that a branch-and-bound algorithm that implements isomorphism pruning will produce an optimal solution of (BIP).

Various procedures are often incorporated in branch-and-bound to fix variables at the nodes, such as reduced cost fixing or implication based fixing. Let x^* be the current best feasible solution found. A variable x_j can be fixed to 0 (resp. to 1) at a node N_a of the enumeration tree when it is determined that no better solution $\bar{x}$ than x^* can exist in N_a with $\bar{x}_j = 1$ (resp. $\bar{x}_j = 0$). We call any such procedure *variable fixing by bounds.* We can describe such a fixing in the enumeration tree by branching at node N_a on variable x_j, and pruning one of the two children by bound. Namely, we prune the left child if no better solution than x^* can exist in N_a with $x_j = 1$, and similarly for the right child. With this convention, all variables fixed in the enumeration tree are branching variables. Therefore these fixing procedures are still valid even when isomorphism pruning is performed.

We also remark that the proof of Proposition 9.13 depends on the set of solutions of (BIP) in a give node N_a, and not on the specific linear programming relaxation at that node. In particular, one can add cutting planes to the formulation at node N_a as long as they are valid for the set of 0,1 solutions in N_a.

The above discussion implies that one can incorporate isomorphism pruning in a general branch-and-cut framework for (BIP).

In order to implement the isomorphism pruning rule, one needs to compute the symmetry group Γ defined in (9.13). Furthermore, at each node of the enumeration tree one needs to compute a permutation π satisfying (9.14), if any exists. The computation of Γ needs to be performed only once. Note that applying isomorphism pruning using a subgroup of Γ instead of Γ itself will still produce a correct algorithm, although in this case there may be isomorphic nodes in the enumeration tree. It is often the case that the user has knowledge of the group Γ, or at least a large subgroup of Γ, as for example in the operating room scheduling problem. The group Γ can be also generated automatically using tools from computational group theory. While the status of computing a set of generators for Γ in polynomial time is an open problem, there is software that runs efficiently in practice, such as `nauty` by McKay [275]. Furthermore, permutation groups can be represented in a compact form, called the Schreier–Sims representation (see for example [328]). Algorithms that are based on this representation, to detect a permutation π satisfying (9.14) if any exists, are typically efficient in practice [267].

In principle, one would like to work with a larger group of permutations than Γ, namely the group Γ' of all permutations π such that, for any $\bar{x} \in \mathbb{R}^n$, $\bar{x}$ is an optimal solution for (BIP) if and only if $\pi(\bar{x})$ is optimal for (BIP). Using Γ' for isomorphism pruning would still produce a correct algorithm (as this would guarantee that at least one optimal solution is kept in the enumeration tree), while resulting in smaller trees. However, computing Γ' is not possible without "a priori" knowledge of the symmetries of the optimal solutions, and in practice one is able to detect symmetries only if they are "displayed" by the formulation.

Orbital Fixing

We consider a branch-and-cut algorithm that performs the following operations on the enumeration tree: addition of valid cutting planes at nodes of the tree, branching on variables, pruning by bound, by infeasibility, by integrality or by isomorphism, and variable fixing by bound as introduced earlier. Recall that the latter operation can be represented in the enumeration tree by branching on variables and pruning by bound.

In addition to these operations, we describe how symmetry in the formulation can be exploited to fix additional variables.

Given a group G of permutations in Σ_n and an element $i \in \{1, \dots, n\}$, the *orbit of i under G* is the set $\{\pi(i) : \pi \in G\}$. Given $i, j \in \{1, \dots, n\}$, i is in the orbit of j if and only if j is in the orbit of i (indeed, $j = \pi(i)$ for $\pi \in G$ if and only if $i = \pi^{-1}(j)$ for $\pi^{-1} \in G$). Therefore the orbits of the elements $1, \dots, n$ under G form a partition. Given a set $S \subset \{1, \dots, n\}$, the *stabilizer of S in G* is the set $\text{stabil}(S, G) := \{\pi \in G : \pi(S) = S\}$. One can show that $\text{stabil}(S, G)$ is also a group.

Let N_a be a node of the enumeration tree and let $\mathcal{O}_a$ be the family of orbits of the elements $1, \dots, n$ under $\text{stabil}(F_a^1, \Gamma)$.

Orbital Fixing Rule. *Let Z be the union of all orbits $O \in \mathcal{O}_a$ such that $O \cap F_a^0 \neq \emptyset$ and $O \setminus F_a^0 \neq \emptyset$. If $Z \neq \emptyset$, create a child $N_{a'}$ of N_a and set $F_{a'}^0 := F_a^0 \cup Z$.*

We say that the variables in $Z \setminus F_a^0$ have been *fixed to zero by orbital fixing* at node N_a.

Example 9.14. We illustrate orbital fixing on the following 0,1 linear program, where $n \geq 3$.

$$\begin{array}{ll} \max & \sum_{j=1}^n x_j \\ & x_i + x_j \leq 1 \quad \text{for all } i, j \\ & x \in \{0,1\}^n. \end{array} \tag{9.15}$$

A branch-and-bound algorithm first solves the linear programming relaxation. The optimum of this linear program puts all variables at $\frac{1}{2}$ for an objective value of $\frac{n}{2}$. Branching is done on variable x_1, say. The branch where x_1 is fixed to 1 can be pruned by integrality (in this subproblem, the linear program sets $x_j = 0$ for all $j \neq 1$ and it has objective value 1). Let us now concentrate on the branch where x_1 is fixed to 0. See Fig. 9.7.

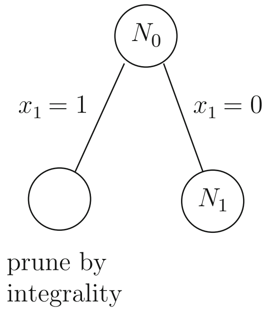


Figure 9.7: Branch-and-bound tree when orbital fixing is applied

Let N_1 denote the corresponding node in the branch-and-bound tree. No variable has been fixed to 1 on the path from N_0 to N_1, therefore $F_1^1 = \emptyset$. In our example, the symmetry group of the formulation (9.15) is Σ_n, the set of all permutations on $1, \ldots, n$. The stabilizer stabil(F_1^1, Σ_n) is Σ_n itself. The family $\mathcal{O}_1$ of orbits of the elements $1, \ldots, n$ under stabil(F_1^1, Σ_n) consists of the single orbit $O = \{1, \ldots, n\}$. Since $F_1^0 = \{1\}$, it follows that $O \cap F_1^0 \neq \emptyset$ and $O \setminus F_1^0 \neq \emptyset$. Therefore the set Z defined in the orbital fixing rule is $Z = O = \{1, \ldots, n\}$. As a consequence we can fix $x_2 = x_3 = \ldots = x_n = 0$ by orbital fixing in node N_1. Since all variables are now fixed in N_1, this node can be pruned and the branch-and-bound algorithm terminates. ■

Proposition 9.15. *Consider a node N_a of the enumeration tree. If $\bar{x}$ is a solution in node N_a such that there exists an orbit $O \in \mathcal{O}_a$ satisfying $O \cap F_a^0 \neq \emptyset$ and an index $i \in O$ such that $\bar{x}_i = 1$, then there exists a node N_b to the left of node N_a containing a solution isomorphic to $\bar{x}$.*

Proof. Let P be the path of the enumeration tree from the root N_0 to node N_a. We will prove the statement by induction on the length of P. The statement is true when $N_a = N_0$. Now consider a general node N_a. Let $\bar{x}$ satisfy the hypothesis. Let S be the union of all orbits $O \in \mathcal{O}_a$ such that $O \cap F_a^0 \neq \emptyset$ and $\bar{x}_i = 1$ for some $i \in O$. Among all indices in $S \cap F_a^0$, choose h to be the index of a variable x_h that was first fixed to zero going from N_0 to N_a in P. Let O be the orbit in $\mathcal{O}_a$ that contains h. By definition of S, there exists $i \in O$ such that $\bar{x}_i = 1$. Since i, h belong to O, it follows that there exists $\pi \in \text{stabil}(F_a^1, \Gamma)$ such that $\pi(h) = i$. Let $\tilde{x} = \pi(\bar{x})$. Note that $\tilde{x}_h = 1$.

Let N_d be the last node on P where variable x_h was not yet fixed to zero. We next show that $\tilde{x}$ is a solution in N_d. Since $\pi \in \text{stabil}(F_a^1, \Gamma)$, it follows that $\tilde{x}_j = 1$ for every $j \in F_a^1$, and thus $\tilde{x}_j = 1$ for every $j \in F_d^1$ because $F_d^1 \subseteq F_a^1$. Thus, if $\tilde{x}$ is not a solution in node N_d, there exists $j \in F_d^0$ such that $\tilde{x}_j = 1$. This implies that $\bar{x}_{\pi(j)} = 1$. But then j is an index in $S \cap F_a^0$ such that the variable x_j was fixed before x_h, contradicting the choice of h.

Now, variable x_h has been fixed along the path P either by branching or by orbital fixing. Suppose x_h was fixed by branching. By the choice of node N_d, x_h must be the branching variable at node N_d, and N_a is a descendant of the right child of N_d. It follows that $\tilde{x}$ is a solution in the left child N_b of N_d, because $\tilde{x}_h = 1$ and $\tilde{x}$ is a solution in N_d. Thus $\tilde{x}$ is a solution isomorphic to $\bar{x}$ in a node to the left of N_a.

Henceforth, we may assume that x_h was fixed to 0 by orbital fixing at node N_d. Let O' be the orbit of h under $\text{stabil}(F_d^1, \Gamma)$. Since h is fixed by orbital fixing at N_d, it follows that $O' \cap F_d^0 \neq \emptyset$. Since $\tilde{x}_h = 1$ and the length of the path from the root N_0 to N_d is less than the length of P, it follows by induction that there exists a node N_b to the left of N_d containing a solution isomorphic to $\tilde{x}$. Since $\tilde{x}$ is isomorphic to $\bar{x}$, N_b contains a solution isomorphic to $\bar{x}$ and the statement follows. □

It is important to note that Proposition 9.13 remains true even when isomorphism pruning is used in conjunction with orbital fixing. Therefore the two rules can both be added to a general branch-and-cut algorithm for (BIP).

9.4 Further Readings

Integer Programming in Fixed Dimension

Surveys on lattice basis reduction and integer programming in fixed dimension are given by Aardal, Weismantel and Wolsey [4] and by Eisenbrand (in [217, pp. 505–560]).

The first papers to apply basis reduction for solving integer programs were those of Hirschberg and Wong [202], who gave a polynomial algorithm for the knapsack problem in two variables, and Kannan [225], who gave a polynomial algorithm for the two-variable integer programming problem.

A question related to basis reduction is to find the shortest nonzero vector in a lattice. Ajtai [9] shows that the shortest vector problem in L2 is NP-hard for randomized reductions, and Micciancio [277] shows that the shortest vector in a lattice is hard to approximate to within some constant.

Kannan [226, 227] provides a variant of Lenstra's algorithm, based on a basis reduction algorithm different from the Lovász' algorithm, which improves the number of hyperplanes to be considered in the recursion to $O(n^{5/2})$. Lovász and Scarf [263] describe an algorithm for integer programming in fixed dimension that avoids computing a rounding ellipsoid in order to find a flat direction. Cook, Rutherford, Scarf and Shallcross [91] report a successful application of Lovász and Scarf's algorithm to a class of small but difficult mixed integer linear programming problems with up to 100 integer variables, arising from a network design application, that could not otherwise be solved by regular branch-and-bound techniques.

Banaszczyk, Litvak, Pajor, and Szarek [35] show that the flatness theorem holds for $k(n) = Cn^{3/2}$, where C is a universal constant. The best choice for $k(n)$ is not known. Note that $k(n) \geq n$ since the simplex $\text{conv}\{0, ne^1, \dots, ne^n\}$ has width n and its interior does not contain any integer point (e^i denotes the ith unit vector).

John [214] shows that every full-dimensional convex body $K \subseteq \mathbb{R}^n$ contains a unique ellipsoid of maximum volume E^*, and that $K \subseteq nE^*$. If $P = \{x : Ax \leq b\}$ is a full-dimensional polytope and E^* is the maximum-volume inscribed ellipsoid in P, one can compute $E \subseteq P$ such that $(1 + \varepsilon)\text{vol}(E) \geq \text{vol}(E^*)$ in time polynomial in the encoding size of (A, b) and in $\log(\varepsilon^{-1})$ using the shallow cut ellipsoid method. Nesterov and Nemirovsky [291] show how to solve such a problem in polynomial time using interior point methods.

It is NP-hard to compute the volume of a polytope, and of a convex body more generally. However, Dyer, Frieze, and Kannan [122] give a random polynomial-time algorithm for approximating these volumes.

A more general problem than the integer feasibility problem is the question of counting integral points in a polyhedron. Barvinok [38] gave a polynomial-time algorithm for counting integral points in polyhedra when the dimension is fixed.

A topic that we do not cover in this book is that of strongly polynomial algorithms. We just mention two papers: Frank and Tardos [149], and Tardos [334].

Computational Aspects of Branch-and-Bound

Much work has been devoted to branching. Related to the topic of basis reduction, we cite Aardal, Bixby, Hurkens, Lenstra, and Smeltink [1], Aardal and Lenstra [2], Aardal, Hurkens, and Lenstra [3].

Strong branching was proposed by Applegate, Bixby, Chvátal, and Cook [13] in the context of the traveling salesman problem.

Computational studies on branching rules for mixed integer linear programming were performed by Linderoth and Savelsbergh [257], Achterberg, Koch, and Martin [7, 8], Achterberg [5], Patel and Chinneck [306], and Kilinc–Karzan, Nemhauser, and Savelsbergh [238].

One aspect that deserves more attention is the design of experiments to evaluate empirical aspects of algorithms. Hooker [205] makes this case in his paper "Needed: An empirical science of algorithms." We also mention "Experimental analysis of algorithms" by McGeogh [274].

Dealing with Symmetry

While the concept of isomorphism pruning in the context of integer programming was first introduced by Margot [267], similar search algorithms have been independently developed in the computer science and constraint programming communities by, among others, Brown, Finkelstein and Purdom [65] and Fahle, Shamberger, and Sellmann [134].

In the context of packing and covering problems, Ostrowski, Linderoth, Rossi, and Smriglio [293] introduced the idea of *orbital branching.* Consider a node a of the enumeration tree and let Γ^a be the symmetry group of the formulation at node a. Suppose we want to branch on the variable x_k. Orbital branching creates two subproblems, the left problem where x_k is set equal to 1, and the right problem where all variables with index in the orbit of k under the group Γ^a are set to 0. This branching strategy excludes many isomorphic solutions, while guaranteeing that an optimal solution is always present in the enumeration tree. However, there is no guarantee that the enumeration tree will not contain isomorphic subproblems.

A common approach to deal with symmetries in the problem is adding symmetry-breaking inequalities. An interesting example is given by Kaibel and Pfetsch in [222], where they characterize the inequalities defining the *partitioning orbitope*, that is the convex hull of all $0, 1$ $m \times n$ matrices with at most one nonzero entry in each row and whose columns are sorted lexicographically. These inequalities can be used to break symmetries in certain integer programming formulations, such as the graph coloring formulation in which a variable x_{ij} indicates if node i is given color j.

9.5 Exercises

Exercise 9.1. Let $k_1, \dots, k_n$ be relatively prime integers, and let p be a positive integer. Let $\Lambda \subseteq \mathbb{Z}^n$ be the set of points $x = (x_1, \dots, x_n)$ satisfying $\sum_{i=1}^n k_i x_i \equiv 0 \pmod{p}$. Prove that Λ is a lattice and that $\det(\Lambda) = p$.

Exercise 9.2. Let $b^1, \dots, b^n$ be an orthogonal basis of $\mathbb{R}^n$. Prove that the shortest vector in the lattice generated by $b^1, \dots, b^n$ is one of the vectors b^i, for $i = 1, \dots, n$.

Exercise 9.3. Let A, B be two $n \times n$ nonsingular integral matrices. Assume that the lattices Λ_A and Λ_B generated by these matrices satisfy $\Lambda_A \subseteq \Lambda_B$.

(i) Prove that $|\det B|$ divides $|\det A|$.

(ii) Prove that there exist bases $u^1, \dots, u^n$ of Λ_A and $v^1, \dots, v^n$ of Λ_B such that $u^i = k_i v^i$ for some positive integers k_i, $i = 1, \dots, n$.

Exercise 9.4. Prove that any lattice $\Lambda \subset \mathbb{R}^2$ has a basis b^1, b^2 such that the angle between b^1 and b^2 is between 60° and 90°.

Exercise 9.5. Let $\Lambda \subset \mathbb{R}^n$ be a lattice and $b^1, \dots, b^n$ a reduced basis of Λ. Prove that

$$\|b^1\| \leq 2^{\frac{n-1}{2}} \min_{u \in \Lambda \setminus \{0\}} \|u\|.$$

Exercise 9.6. Let $\Lambda \subset \mathbb{R}^n$ be a lattice and $b^1, \dots, b^n$ a reduced basis of Λ. For $u \in \Lambda$, consider $\lambda \in \mathbb{Z}^n$ such that $u = \sum_{i=1}^n \lambda_i b^i$. Prove that, if u is a shortest nonzero vector in Λ, then $|\lambda_i| \leq 3^n$ for $i = 1, \dots, n$.

Exercise 9.7. Let $K \subseteq \mathbb{R}^n$ be a convex body and let $w_d(K)$ denote its width along direction $d \in \mathbb{R}^n$. Prove that $\inf_{d \in \mathbb{Z}^n} w_d(K) = \min_{d \in \mathbb{Z}^n} w_d(K)$, i.e., there exists $d^* \in \mathbb{Z}^n$ such that $w_{d^*}(K) = \inf_{d \in \mathbb{Z}^n} w_d(K)$.

Exercise 9.8. For a matrix $B \in \mathbb{R}^{n \times n}$, let M be the absolute value of the largest entry in B. Prove that $|\det(B)| \leq n^{n/2} M^n$.

Exercise 9.9. Let $K \in \mathbb{R}^n$ be a full-dimensional convex body that does not contain any integral point. Let us call an *open split set* any set of the form $\{x \in \mathbb{R}^n : \pi_0 < \pi x < \pi_0 + 1\}$ where $\pi \in \mathbb{Z}^n$, $\pi_0 \in \mathbb{Z}$. Prove that K is contained in the union of $C(n)$ open split sets, where $C(n)$ is a constant depending only on n.

Exercise 9.10. Modify Lenstra's algorithm so that it computes an integral solution to $Ax \leq b$ if one exists (hint: use (9.9)).

Exercise 9.11. Let $\Lambda \subset \mathbb{Q}^n$ be the lattice generated by a basis $b^1, \dots, b^n \in \mathbb{Q}^n$. Prove that, for fixed n, the shortest vector in Λ can be found in polynomial time.

Exercise 9.12. Apply Lenstra's algorithm to find a solution to

$$\begin{aligned}&243243x_1 + 244223x_2 + 243334x_3 = 8539262753\\ &x_1, x_2, x_3 \geq 0 \text{ integer.}\end{aligned}$$

Exercise 9.13. Consider a mixed integer linear program with one integer variable. Assume that its linear programming relaxation has a unique optimal solution. Prove that any branch-and-bound algorithm using variable branching and the LP relaxation to compute lower bounds has an enumeration tree of size at most three.

Exercise 9.14. Consider the mixed integer linear program (9.10) where all integer variables are bounded, $0 \leq x_j \leq u_j$ with u_j integer for $j = 1, \dots, p$. Specialize the branching rule in the branch-and-cut algorithm as follows. Choose a variable x_j, $j = 1, \dots, p$ such that x^i_j is fractional. From the linear program LP$_i$, construct $u_j + 1$ subproblems LP^t_i obtained by adding the constraint $x_j = t$ for $t = 0, \dots, u_j$. Let z^t_i be the objective value of LP^t_i. Let $t_l = \lfloor x^i_j \rfloor$ and $t_u = \lceil x^i_j \rceil$.

(i) Prove that $z_i^{t_l} = \max_{t=0,\dots,t_l} z^t_i$ and $z_i^{t_u} = \max_{t=t_u,\dots,u_j} z^t_i$.

(ii) Suppose that the above branching rule is used in conjunction with a node selection rule that chooses subproblem LP$_i^{t_l}$ or LP$_i^{t_u}$ before the others. Is there an advantage in using this branching rule rather than the more common $x_j \leq t_l$ or $x_j \geq t_u$?

Exercise 9.15. Consider the symmetric traveling salesman problem on the undirected graph $G = (V, E)$ with costs c_e, $e \in E$. Devise a branch-and-bound algorithm for solving this problem where bounds are obtained by computing Lagrangian bounds based on the 1-tree relaxation (8.7).

Explain each step of your branch-and-bound algorithm, such as branching, and how subproblems are created and solved at the nodes of the enumeration tree.

Exercise 9.16. Let $[l, u] \subset \mathbb{R}$ be an interval. An approach for maximizing a nonlinear function $f : [l, u] \to \mathbb{R}$ is to approximate it by a piecewise linear function g with breakpoints $l = s_1 < s_2 < \dots < s_n = u$. The formulation is

$$\begin{aligned} x = \;& \textstyle\sum_{k=1}^n \lambda_k s_k\\ g(x) = \;& \textstyle\sum_{k=1}^n \lambda_k f(s_k)\\ & \textstyle\sum_{k=1}^n \lambda_k = 1\\ & \lambda_k \geq 0, \text{ for } k = 1, \dots, n\end{aligned}$$

together with the condition that no more than two of the λ_k are positive and these are of the form λ_k, λ_{k+1}.

(i) Prove that a valid separation is given by the constraints

$$\sum_{k=1}^{p-1} \lambda_k = 0 \quad \text{or} \quad \sum_{k=p+1}^{n} \lambda_k = 0$$

for some $p = 2, \ldots, n-1$.

(ii) Based on this separation, develop a branch-and-bound algorithm to find an approximate solution to a separable nonlinear program of the form

$$\begin{array}{ll} \max & \sum_{j=1}^n f_j(x_j) \\ & Ax \le b \\ & l \le x \le u \end{array}$$

where $f_j : [l_j, u_j] \to \mathbb{R}$ are nonlinear functions for $j = 1, \ldots, n$.

Exercise 9.17. Denote by (LP) the linear programming relaxation of (9.10). Let $\bar{x}$ denote an optimal solution of (LP). For a split disjunction $\sum_{j=1}^p \pi_j x_j \le \pi_0$ or $\sum_{j=1}^p \pi_j x_j \ge \pi_0 + 1$ where $(\pi, \pi_0) \in \mathbb{Z}^{p+1}$ is chosen such that $\bar{x}$ satisfies $\pi_0 < \sum_{j=1}^p \pi_j \bar{x}_j < \pi_0 + 1$, let (LP$^-$) and (LP$^+$) be the linear programs obtained from (LP) by adding the constraints $\sum_{j=1}^p \pi_j x_j \le \pi_0$ and $\sum_{j=1}^p \pi_j x_j \ge \pi_0 + 1$ respectively. Let z^- and z^+ be the optimum objective values of (LP$^-$) and (LP$^+$) respectively. Compare z^-, z^+ with the value $\hat{z}$ obtained by adding to (LP) a split cut from the above disjunction: Construct an example showing that the differences $\hat{z} - z^-$ and $\hat{z} - z^+$ can be arbitrarily large. Compare z^-, z^+ with the value z^* obtained by adding to (LP) all the split inequalities from the above disjunction.

Exercise 9.18. Consider a pure integer linear program, with bounded variables $0 \le x_j \le u_j$ for $j = 1, \ldots, n$, where the objective function z is to be maximized. Assume that we know a lower bound $\underline{z}$ and that the linear programming relaxation has been solved to optimality with the objective function represented as $z = \overline{z} + \sum_{j \in N_0} \overline{c}_j x_j + \sum_{j \in N_u} \overline{c}_j (x_j - u_j)$ where N_0 indexes the nonbasic variables at 0, N_u indexes the nonbasic variables at their upper bound, $\overline{c}_j \le 0$ for $j \in N_0$, and $\overline{c}_j \ge 0$ for $j \in N_u$. Prove that in any optimal solution of the integer program

$$x_j \le \lfloor \frac{\overline{z} - \underline{z}}{-\overline{c}_j} \rfloor \quad \text{for } j \in N_0 \text{ such that } \overline{c}_j \ne 0, \quad \text{and}$$

$$x_j \geq u_j - \lceil \frac{\overline{z} - \underline{z}}{\overline{c}_j} \rceil \quad \text{for } j \in N_u \text{ such that } \overline{c}_j \neq 0.$$

Exercise 9.19. Let B index the set of binary variables in a mixed 0,1 linear program. For $j \in B$, let $\bar{x}_j := 1 - x_j$ and let $\bar{B}$ index the set of binary variables $\bar{x}_j$, $j \in B$. Construct the graph $G = (B \cup \bar{B}, E)$ where two nodes are joined by an edge if and only if the two corresponding variables cannot be 1 at the same time in a feasible solution.

1. Show that, for any clique C of G, the inequality

$$\sum_{C \cap B} x_j - \sum_{C \cap \bar{B}} x_j \leq 1 - |C \cap \bar{B}|$$

   is a valid inequality for the mixed 0,1 linear program.

2. What can you deduce when G has a clique C of cardinality greater than two such that $C \cap B$ and $C \cap \bar{B}$ contain nodes corresponding to variables x_k and $\bar{x}_k$ respectively, for some index k.

3. Construct the graph $(B \cup \bar{B}, E)$ for the 0,1 program

$$\begin{array}{rl} 4x_1 + x_2 - 3x_4 & \leq 2 \\ 3x_1 + 2x_2 + 5x_3 + 3x_4 & \leq 7 \\ x_2 + x_3 - x_4 & \leq 0 \\ x_1, x_2, x_3, x_4 & \in \{0,1\}. \end{array}$$

   Deduce all the possible maximal clique inequalities and variable fixings.

Exercise 9.20. Consider the following integer linear program

$$\begin{array}{lrrl} \min & & z & \\ & 2x_1 + \ldots + 2x_n & +z & = n \\ & x_1, \ldots, x_n & & = 0 \text{ or } 1 \\ & & z & \geq 0 \end{array}$$

where n is an odd integer.

(i) Prove that any branch-and-bound algorithm using variable branching and the linear programming relaxation to compute lower bounds, but no isomorphism pruning, has an enumeration tree of size at least $2^{n/2}$.

(ii) What is the size of the enumeration tree when isomorphism pruning is used?

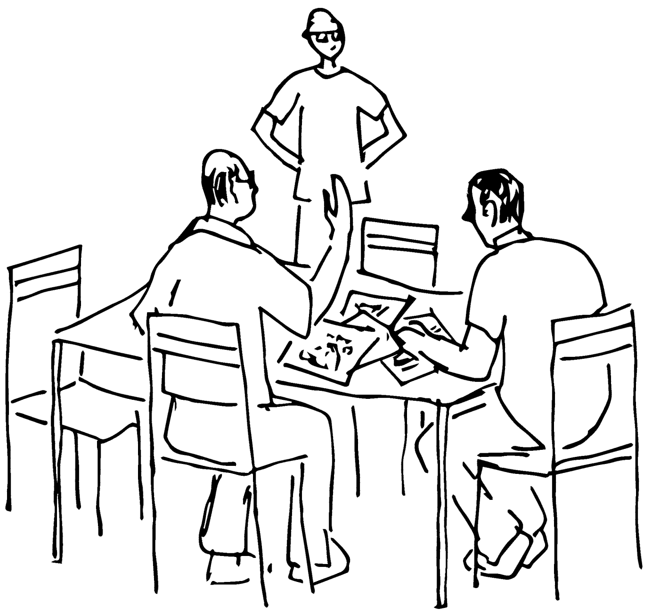
Exercise 9.21. Given a group G of permutations in Σ_n and a set $S \subset \{1, \ldots, n\}$, show that the stabilizer stabil(S, G) is a group.

Exercise 9.22. Consider the following 0,1 linear program, where $n \geq 3$.

$$\begin{array}{ll} \max & \sum_{j=1}^{n} x_j \\ & x_i + x_j + x_k \leq 2 \quad \text{for all } i, j, k \\ & x \in \{0,1\}^n. \end{array} \tag{9.16}$$

Provide the complete enumeration tree for a branch-and-bound algorithm that uses isomorphism pruning and orbital fixing.

Exercise 9.23. Formulate as a pure 0,1 linear program the question of whether there exists a $v \times b$ binary matrix with exactly r ones per row, k ones per column, and with a scalar product of λ between any pair of distinct rows. Solve the problem for $(v, b, r, k, \lambda) = (7, 7, 3, 3, 1)$.



The authors working on Chap. 9

One of the authors enjoying a drink under a fig tree