4 N-Fold Integer Programming

In this chapter we develop the theory of n-fold integer programming, which, incorporating the results of Chapters 2 and 3, leads to the first polynomial time solution of many extremely important fundamental linear and nonlinear integer programming problems in variable dimension. In fact, as will be explained in Chapter 5, this class of problems is universal for integer programming in a well defined sense, and every (linear or nonlinear) integer program is a suitable n-fold program.

Let us start right away with the definition of this class of problems. An \((r + s) \times t\) bimatrix is a matrix \(A\) consisting of two blocks \(A_1, A_2\), with \(A_1\) being its \(r \times t\) submatrix consisting of the first \(r\) rows and \(A_2\) being its \(s \times t\) submatrix consisting of the last \(s\) rows. Note that \(r = 0\) or \(s = 0\) are allowed. The \(n\)-fold product of an \((r + s) \times t\) bimatrix \(A\) is then defined to be the following \((r + ns) \times nt\) matrix,

\[
A^{(n)} := (I_r^n \otimes A_1) \oplus (I_r^n \otimes A_2) = \begin{pmatrix}
A_1 & A_1 & A_1 & \cdots & A_1 \\
A_2 & 0 & 0 & \cdots & 0 \\
0 & A_2 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & A_2
\end{pmatrix}
\]

Note that the \(n\)-fold product depends on \(r\) and \(s\) and not only on the entries of \(A\).

We often consider \(n\)-fold products of a bimatrix whose first block is the identity matrix. We therefore introduce the following notation. Given an \(s \times t\) matrix \(A\), let

\[
\Box A := \begin{pmatrix}
I_t \\
A
\end{pmatrix}
\]

be the \((t + s) \times t\) bimatrix whose first block is the \(t \times t\) identity matrix and whose second block is \(A\). The \(n\)-fold product of \(\Box A\) is then the \((t + ns) \times nt\) matrix

\[
(\Box A)^{(n)} = (I_t^n \otimes I_t) \oplus (I_t^n \otimes A) = \begin{pmatrix}
I_t & I_t & I_t & \cdots & I_t \\
A & 0 & 0 & \cdots & 0 \\
0 & A & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & A
\end{pmatrix}
\]

In this chapter we study the general nonlinear n-fold integer programming problem

\[
\min / \max \{ f(Wx) : x \in \mathbb{Z}^N, \ A^{(n)}x = b, \ l \leq x \leq u \}, \tag{22}
\]

where \(A\) is an integer \((r + s) \times t\) bimatrix, \(n\) is a positive integer, \(N := nt\) is the (variable) dimension of the set of feasible points, \(W\) is an integer \(d \times N\) matrix (with the useful special case of \(W = I_N\) the identity matrix), \(b \in \mathbb{Z}^{r+ns}\) is a suitable right-hand side, \(l, u \in \mathbb{Z}^N_\infty\) are lower and upper bound vectors, and \(f : \mathbb{Z}^d \rightarrow \mathbb{R}\) is a function which is typically given by a comparison oracle. Note that the problem looks like a generic nonlinear integer programming problem, except that the matrix defining the system of inequalities is the \(n\)-fold product of a smaller bimatrix.
In §4.1 we study Graver bases of \( n \)-fold products of integer bimatrices and show that they can be computed in polynomial time. In §4.2, incorporating the results of Chapters 2 and 3, we show that linear optimization, separable convex minimization, and convex maximization over \( n \)-fold integer programs can be done in polynomial time. In §4.3 we discuss (two-stage) stochastic integer programming, which is a certain dual of \( n \)-fold integer programming, and extend our theory to enable polynomial time linear and nonlinear stochastic integer programming as well. We conclude in §4.4 with some of the numerous important applications of this theory.

### 4.1 Graver Bases of N-Fold Products

In §4.1.1 we define the Graver complexity of a bimatrix and use it to establish a polynomial time algorithm for computing Graver bases of \( n \)-fold products. In §4.1.2 we discuss an algorithmic scheme that allows to gradually build up increasingly better approximations of \( G(A^{(n)}) \), which will be used in §4.2 to design a certain Graver approximation scheme for linear and nonlinear \( n \)-fold integer programming. In §4.1.3 we introduce the Graver complexity of a graph and a digraph, which is an important invariant that controls the computational complexity of linear and nonlinear multicommodity flow problems, and use it to demonstrate the algorithm of §4.1.2 for computing Graver bases of \( n \)-fold products.

#### 4.1.1 Computing N-Fold Graver Bases in Polynomial Time

Let \( A \) be a fixed integer \((r + s) \times t\) bimatrix with blocks \( A_1, A_2 \). For each positive integer \( n \) we index vectors in \( \mathbb{Z}^{nt} \) as \( x = (x^1, \ldots, x^n) \) with each brick \( x^k \) lying in \( \mathbb{Z}^t \). The type of vector \( x \) is the number \( \text{type}(x) := |\{k : x^k \neq 0\}| \) of nonzero bricks of \( x \).

The following definition plays an important role in the sequel.

**Definition 4.1** The Graver complexity \( g(A) \) of an integer bimatrix \( A \) is the smallest \( g \in \mathbb{Z}_+ \cup \{\infty\} \) such that every element \( x \) in \( G(A^{(n)}) \) for every \( n \) has type at most \( g \).

We proceed to establish a result of [66] and its extension in [45] which shows that, in fact, the Graver complexity of any integer bimatrix \( A \) is finite.

Consider \( n \)-fold products \( A^{(n)} \) of \( A \). By definition of the \( n \)-fold product, \( A^{(n)} x = 0 \) if and only if \( A_1 \sum_{k=1}^n x^k = 0 \) and \( A_2 x^k = 0 \) for all \( k \). In particular, a necessary condition for \( x \) to lie in \( \mathcal{L}(A^{(n)}) \), and in particular in \( G(A^{(n)}) \), is that \( x^k \in \mathcal{L}(A_2) \) for all \( k \). Call a vector \( x = (x^1, \ldots, x^n) \) full if, in fact, \( x^k \in \mathcal{L}^*(A_2) \) for all \( k \), in which case \( \text{type}(x) = n \), and pure if, moreover, \( x^k \in G(A_2) \) for all \( k \). Full vectors, and in particular pure vectors, are natural candidates for lying in the Graver basis \( G(A^{(n)}) \) of \( A^{(n)} \), and will indeed play an important role in its construction.

Consider any full vector \( y = (y^1, \ldots, y^n) \). By definition, each brick of \( y \) satisfies \( y^i \in \mathcal{L}^*(A_2) \) and is therefore a conformal sum \( y^i = \sum_{j=1}^{k_i} x^{i,j} \) of some elements \( x^{i,j} \in G(A_2) \) for all \( i, j \). Let \( n := k_1 + \cdots + k_m \geq m \) and let \( x \) be the pure vector

\[
  x = (x^1, \ldots, x^n) := (x_1^{1,k_1}, \ldots, x_m^{1,k_m})
\]
We call the pure vector $x$ an \textit{expansion} of the full vector $y$, and we call the full vector $y$ a \textit{compression} of the pure vector $x$. Note that $A_1 \sum y^i = A_1 \sum x^{i,j}$ and therefore $y \in \mathcal{L}(A^{(m)})$ if and only if $x \in \mathcal{L}(A^{(n)})$. Note also that each full $y$ may have many different expansions and each pure $x$ may have many different compressions.

\textbf{Lemma 4.2} Consider any full $y = (y^1, \ldots, y^m)$ and any expansion $x = (x^1, \ldots, x^n)$ of $y$. If $y$ is in the Graver basis $\mathcal{G}(A^{(m)})$ then $x$ is in the Graver basis $\mathcal{G}(A^{(n)})$.

\textit{Proof.} Let $x = (x^{1,1}, \ldots, x^{m,km}) = (x^1, \ldots, x^n)$ be an expansion of $y = (y^1, \ldots, y^m)$ with $y^i = \sum_{j=1}^{k_i} x^{i,j}$ for each $i$. Suppose indirectly $y \in \mathcal{G}(A^{(m)})$ but $x \notin \mathcal{G}(A^{(n)})$. Since $y \in \mathcal{L}^{*}(A^{(m)})$ we have $x \in \mathcal{L}^{*}(A^{(n)})$. Since $x \notin \mathcal{G}(A^{(n)})$, there exists an element $g = (g^{1,1}, \ldots, g^{m,km})$ in $\mathcal{G}(A^{(n)})$ satisfying $g \sqsupseteq x$. Let $h = (h^1, \ldots, h^m)$ be the compression of $g$ defined by $h^i := \sum_{j=1}^{k_i} g^{i,j}$. Since $g \in \mathcal{L}^{*}(A^{(n)})$ we have $h \in \mathcal{L}^{*}(A^{(m)})$. But $h \sqsupseteq y$ and $h \neq y$, contradicting $y \in \mathcal{G}(A^{(m)})$. This completes the proof. \hfill $\Box$

\textbf{Lemma 4.3} The Graver complexity $g(A)$ of every integer bimatrix $A$ is finite.

\textit{Proof.} We need to bound the type of any element in the Graver basis of the $l$-fold product of $A$ for any $l$. Suppose there is an element $z$ of type $m$ in some $\mathcal{G}(A^{(l)})$. Then its restriction $y = (y^1, \ldots, y^m)$ to its $m$ nonzero bricks is a full vector and is in the Graver basis $\mathcal{G}(A^{(m)})$. Let $x = (x^1, \ldots, x^n)$ be any expansion of $y$. Then type($z$) = $m \leq n = $ type($x$), and by Lemma 4.2, the pure vector $x$ is in $\mathcal{G}(A^{(n)})$.

Therefore, it suffices to bound the type of any pure element in the Graver basis of the $n$-fold product of $A$ for any $n$. Suppose $x = (x^1, \ldots, x^n)$ is a pure element in $\mathcal{G}(A^{(n)})$ for some $n$. Let $\mathcal{G}(A_2) = \{g^1, \ldots, g^p\}$ be the Graver basis of $A_2$ and let $G_2$ be the $t \times p$ matrix whose columns are the $g^i$. Let $v \in \mathbb{Z}_+^p$ be the vector with $v_i := |\{k : x^k = g_i^1\}|$ counting the number of bricks of $x$ which are equal to $g_i^1$ for each $i$. Then $\sum_{i=1}^{p} v_i = $ type($x$) = $n$. Now, note that $A_1 G_2 v = A_1 \sum_{k=1}^{n} x^k = 0$ and hence $v \in \mathcal{L}^{*}(A_1 G_2)$. We claim that, moreover, $v$ is in $\mathcal{G}(A_1 G_2)$. Suppose indirectly it is not. Then there is a $\hat{v} \in \mathcal{G}(A_1 G_2)$ with $\hat{v} \sqsubset v$, and it is easy to obtain a nonzero $\hat{x} \sqsubset x$ from $x$ by zeroing out some bricks so that $\hat{v}_i = |\{k : \hat{x}^k = g_i^1\}|$ for all $i$. Then $A_1 \sum_{k=1}^{n} \hat{x}^k = A_1 G_2 \hat{v} = 0$ and hence $\hat{x} \in \mathcal{L}^{*}(A^{(n)})$, contradicting $x \in \mathcal{G}(A^{(n)})$.

So the type of any pure vector, and hence the Graver complexity of $A$, is at most the largest value $\sum_{i=1}^{p} v_i$ of any nonnegative vector $v$ in the Graver basis $\mathcal{G}(A_1 G_2)$. \hfill $\Box$

We proceed to establish the following theorem from [20] which asserts that Graver bases of $n$-fold products can be computed in polynomial time. An $n$-\textit{lifting} of a vector $y = (y^1, \ldots, y^m)$ consisting of $m$ bricks is any vector $z = (z^1, \ldots, z^n)$ consisting of $n$ bricks such that for some $1 \leq k_1 < \cdots < k_m \leq n$ we have $z^{k_i} = y^i$ for $i = 1, \ldots, m$, and all other bricks of $z$ are zero; in particular, $n \geq m$ and type($z$) = type($y$).

\textbf{Theorem 4.4} For every fixed integer bimatrix $A$ there is an algorithm that, given any positive integer $n$, computes the Graver basis $\mathcal{G}(A^{(n)})$ of the $n$-fold product of $A$, in time which is polynomial in $n$. In particular, the cardinality $|\mathcal{G}(A^{(n)})|$ and the binary-encoding length $\langle \mathcal{G}(A^{(n)}) \rangle$ of the Graver basis of $A^{(n)}$ are polynomial in $n$. 

Proof. Let $g := g(A)$ be the Graver complexity of $A$. Since $A$ is fixed, so is $g$. Therefore, for every $n \leq g$, the Graver basis $G := G(A^{(n)})$ of the $g$-fold product of $A$, can be computed in constant time.

Now, consider any $n > g$. Let $L$ be the set of all $n$-liftings of vectors in $G$. We claim that $G(A^{(n)})$ is the set of $\subseteq$-minimal vectors in $L$. To prove this, it suffices to show that $G(A^{(n)}) \subseteq L \subseteq L^*(A^{(n)})$. First, any $n$-lifting $z$ of any $y \in L^*(A^{(n)})$ is clearly in $L^*(A^{(n)})$ and therefore $L \subseteq L^*(A^{(n)})$. Second, consider any $z \in G(A^{(n)})$. Then type$(z) \leq g$ and hence $z$ is the $n$-lifting of some vector $y$ consisting of $g$ bricks. Suppose indirectly that $y$ is not in $G = G(A^{(n)})$. Then there exists a $y' \in G$ such that $y' \supseteq y$. But then there is a suitable $n$-lifting $z'$ of $y'$ such that $z' \supseteq z$ and $z' \neq z$, contradicting $z \in G(A^{(n)})$. Therefore, $y \in G$ and hence its lifting $z$ is in $L$.

This shows $G(A^{(n)}) \subseteq L$. So the proof of the claim is complete.

Now, the number of $n$-liftings of each $y \in G$ is at most $(\binom{n}{g})$, and so we obtain

$$G(A^{(n)}) \leq |L| \leq (\binom{n}{g}) |G| = O(n^g).$$

So the set $L$ of all $n$-liftings of vectors in $G$ can be computed, and the Graver basis $G(A^{(n)})$ of $\subseteq$-minimal vectors in $L$ be distilled out of $L$, in time polynomial in $n$. \hfill \qed

Note that the algorithm of Theorem 4.4 relies on the mere existence of a finite Graver complexity $g(A)$ for every $A$. Then $g(A)$ and the Graver bases $G(A^{(k)})$ for $1 \leq k \leq g(A)$ can be built into the algorithm. However, it is useful to note the following simple finite procedure for computing the Graver complexity. For an integer $r \times p$ matrix $B$ we use $H(B) := G(B) \cap \mathbb{Z}_+^p$ to denote the Hilbert basis of $B$ which is often easier to compute than the entire Graver basis (such as by a suitable restriction of the simple generic Algorithm 3.5 to the nonnegative orthant).

Algorithm 4.5 (computing the Graver complexity of a bimatrix).

1. Compute the Graver basis $G(A_2)$.

2. If $G(A_2) = \emptyset$ then output $g(A) := 0$. Otherwise form the matrix $G_2$ with columns the elements of $G(A_2)$, compute $H(A_1G_2)$ or $G(A_1G_2)$, and output

$$g(A) := \max \{1v : v \in H(A_1G_2)\} = \max \{\|v\|_1 : v \in G(A_1G_2)\}. $$

To justify this procedure, first note that if $G(A_2)$ is empty, which holds if and only if $A_2$ has linearly independent columns, then $G(A^{(n)})$ is also empty for all $n$ and hence indeed $g(A) = 0$. Next note that if $G(A_2) = \{g^1, \ldots, g^p\}$ is nonempty then the proof of Lemma 4.3 shows that if $x = (x^1, \ldots, x^n)$ is a pure vector in $G(A^{(n)})$ then the vector $v \in \mathbb{Z}_+^p$ with $v_i = |\{k : x^k = g^i\}|$ is an element of $H(A_1G_2) = G(A_1G_2) \cap \mathbb{Z}_+^p$ with $1v = \sum_{i=1}^p v_i = n = \text{type}(x)$; but it is also easy to see that, conversely, each vector $v \in H(A_1G_2)$ with $1v = n$ gives rise to a pure element $x \in G(A^{(n)})$ and therefore indeed $g(A) = \max \{1v : v \in H(A_1G_2)\}$. Finally, note that a vector $g$ is a column of $G_2$ if and only if $-g$ is, and therefore the same property holds for $A_1G_2$, which implies that $g(A) = \max \{\|v\|_1 : v \in G(A_1G_2)\}$ holds as well.
### 4.1.2 Gradual Construction of N-Fold Graver Bases

The algorithm of Theorem 4.4 makes use of the Graver basis $G(A^{g(A)})$. While it can be computed in constant time for each fixed bimatrix $A$, this can be a very heavy task, even for small $A$, if one uses a generic procedure for computing Graver bases, such as the simple Algorithm 3.5 noted in Chapter 3. We therefore proceed to discuss a variant of the algorithm of Theorem 4.4, which is in fact an algorithmic scheme parameterized by an integer $1 \leq k \leq g(A)$, that allows to gradually build up increasingly better approximations of $G(A^{(n)})$. (If $g(A) = 0$ then $G(A^{(n)}) = \emptyset$ for all $n$ and this scheme is not needed.) In §4.2 we will discuss this scheme further and use it to design a certain \textit{Graver approximation scheme} for nonlinear $n$-fold integer programming. This scheme will be extended in §5.6 through the universality theorem to provide a heuristic for arbitrary nonlinear integer programming.

**Algorithm 4.6 (approximation scheme for $n$-fold Graver bases).**

1. Construct the set $P_k \subset \mathcal{L}^*(A^{(k)})$ of pure vectors of type $k$, given by
   \[
   P_k := \left\{ x = (x^1, \ldots, x^k) : x^1, \ldots, x^k \in G(A_2), A_1 \sum_{i=1}^{k} x^i = 0 \right\}.
   \]

2. Construct the set $C_k \subset \bigcup_{m=1}^{k} \mathcal{L}^*(A^{(m)})$ of all compressions of vectors in $P_k$.

3. Construct the set $L^n_k \subset \mathcal{L}^*(A^{(n)})$ of all $n$-liftings of vectors in $C_k$.

4. Set $H^n_k := H^n_{k-1} \cup L^n_k$ where $H^n_{k-1}$ is obtained recursively and $H^n_0 := \emptyset$.

5. Distill and output $G^n_k$ as the set of all $\sqsubseteq$-minimal elements in $H^n_k$.

Several explanatory notes about this algorithmic scheme and its complexity are in order. Let $g := g(A)$. First note that $G(A_2)$ and $g$ are fixed since $A$ is, and hence the set $P_k$ produced in step 1 is fixed and independent of $n$ and has constant cardinality $|P_k| \leq |G(A_2)|^k \leq |G(A_2)|^g$. Likewise, the set $C_k$ produced in step 2, consisting of vectors with various numbers $1 \leq m \leq k$ of bricks, is fixed: indeed, each compression of a pure $x \in P_k$ is determined by some $m$-tuple $(k_1, \ldots, k_m)$ of positive integers summing to $k$, and so $|C_k| \leq k^m |P_k| \leq g^g |P_k|$. Now, the number of $n$-liftings of any vector of type $m$ in $C_k$ is $\binom{n}{m}$ and therefore the set $L^n_k$ produced in step 3 satisfies $|L^n_k| = O(n^k)$. This implies by induction on $k$ that the cardinality of $H^n_k$ produced in step 4 and that of the $k$-th approximation $G^n_k$ of the Graver basis $G(A^{(n)})$ output in step 5 also obey the upper bound $O(n^k)$ and hence are polynomial in $n$.

Next, note that $G^n_k$ is a set of $\sqsubseteq$-incomparable elements in $\mathcal{L}^*(A^{(n)})$ and hence can indeed be regarded as a certain approximation of $G(A^{(n)})$. Note also that the $k$-th approximation refines the $(k-1)$-th approximation in the sense that for every $x \in G^n_{k-1}$ there exists some $y \in G^n_k$ satisfying $y \sqsubseteq x$. Finally, we claim that for $k = g = g(A)$, the algorithm outputs the true Graver basis $G(A^{(n)}) = G^n_g$ of the $n$-fold product. To see this, consider any $z \in G(A^{(n)})$ and let $m := \text{type}(z)$. Then
z is the $n$-lifting of some full vector $y \in \mathcal{G}(A^{(m)})$, which by Lemma 4.2, is in turn the compression of some pure $x \in \mathcal{G}(A^{(k)})$ with $k \leq g$. Therefore $x \in P_k$, $y \in C_k$, $z \in L^n_k$ and $z \in H^n_g = \bigcup_{i=1}^g L^n_i$. Since $z$ is in $\mathcal{G}(A^{(n)})$ it is $\sqsubseteq$-minimal in $\mathcal{L}^*(A^{(n)})$ and will therefore be included in $G^n_g$. Thus, $G^n_g$ is a set of $\sqsubseteq$-incomparable vectors that satisfies $\mathcal{G}(A^{(n)}) \subseteq G^n_g \subset \mathcal{L}^*(A^{(n)})$ and hence indeed $G(A^{(n)}) = C^n_g$.

We conclude with a short informal discussion of the complexity of Algorithm 4.6 and some implementational issues. It is clear from the discussion above about the cardinalities of $P_k$, $C_k$, $L^n_k$, $H^n_k$ and $G^n_k$, that steps 1 and 2 can be done in constant time, steps 3 and 4 in $O(n^k)$ time, and step 5 which involves comparing pairs of elements, in $O(n^{2k})$ time. It should be noted that there is much room for experimentation with better and more efficient ways of organizing this scheme. For instance, if the Graver bases $\mathcal{G}(A^{(n)})$ for various increasing values of $n$ are to be approximated, then steps 1 and 2 which are independent of $n$ can be done for some desired values of $k$ ahead of time, as a preprocessing preparation. Also, it is possible to eliminate step 5 and simply output $H^n_k$, which is a superset of $G^n_k$ and hence also an approximation of the Graver basis, avoiding the pairwise comparisons in step 5 and reducing the time complexity to $O(n^{k\epsilon})$. But, when using the Graver basis for iterative augmentation in the optimization algorithms of Chapter 3 and those to be described later in this chapter, it may be advantageous to use the smaller set $G^n_k$.

### 4.1.3 Graver Complexity of Graphs and Digraphs

We now introduce the Graver complexity of a graph and a digraph. This is an important invariant which will be discussed in more detail in §4.4 and §5.7, where we will show that it controls the complexity of linear and nonlinear multicommodity flow problems, and is important in connection with the universality theorem for multiway tables. Here we only give the definition and use it to demonstrate the computation of Graver bases of $n$-fold products by Algorithm 4.6 above.

Let $G$ be a graph or a digraph with $s$ vertices and $t$ edges and let $D$ be its $s \times t$ incidence matrix. (For a graph, $D_{v,e} = 1$ if edge $e$ contains vertex $v$ and $D_{v,e} = 0$ if not; for a digraph, $D_{v,e} = 1$ if edge $e$ leaves vertex $v$, $D_{v,e} = 1$ if $e$ enters $v$, and $D_{v,e} = 0$ otherwise.) The following graph and digraph invariants, the importance and difficulty of which will be explained in §4.4 and §5.7, were introduced in [10].

**Definition 4.7** The Graver complexity $g(G)$ of a graph or a digraph $G$ with $s$ vertices and $t$ edges is the Graver complexity $g(\Xi D)$ of the $(t + s) \times t$ bimatrix with first block the $t \times t$ identity matrix and second block the incidence matrix $D$ of $G$.

**Example 4.8 (directed triangle).** Here we represent Graver bases as sets of rows of suitable matrices. Let $A := (\Xi D)$ be the $(3 + 3) \times 3$ bimatrix with first block $A_1 = I_3$ and second block $A_2 = D$ the incidence matrix of the directed triangle $T$,

$$ D = \begin{pmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}. $$
Then it is easy to see that
\[ G^T_2 := \mathcal{G}(A_2) = \begin{pmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{pmatrix}, \]
\[ \mathcal{H}(A_1G_2) = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}, \quad \mathcal{G}(A_1G_2) = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}. \]

Therefore, by Algorithm 4.5, the Graver complexity of the directed triangle \( T \) is
\[ g(T) = g(A) = \max \{1v : v \in \mathcal{H}(A_1G_2)\} = \max \{\|v\|_1 : v \in \mathcal{G}(A_1G_2)\} = 2. \]

We now demonstrate Algorithm 4.6 for \( k = 2 = g(A) \) and \( n = 4 \), and construct the Graver basis \( \mathcal{G}(A^{(4)}) \) of the following, already quite large, \( 15 \times 12 \) matrix,
\[
A^{(4)} = \begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1
\end{pmatrix}.
\]

For \( k = 2 = g(A) \), Algorithm 4.6 computes the following sets (consisting of the rows of the corresponding matrices), and outputs the true Graver basis \( G^4_2 = \mathcal{G}(A^{(4)}) \):
\[ C_2 = P_2 = \begin{pmatrix} 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{pmatrix}, \]
\[ G^4_2 = H^4_2 = L^5_2 = \begin{pmatrix}
1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & -1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\
-1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \\
-1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & -1 & -1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 \\
0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 1 & 1 & 1
\end{pmatrix}.\]
4.2 Linear and Nonlinear N-Fold Integer Programming

4.2.1 Deciding Feasibility of N-Fold Programs

Combining the results of §3.2 and §4.1 we now show that we can find a feasible point in a given $n$-fold integer program or assert that none exists in polynomial time.

**Theorem 4.9** For every fixed integer $(r+s)\times t$ bimatrix $A$, there is an algorithm that, given $n, l, u \in \mathbb{Z}_\infty^t$, and $b \in \mathbb{Z}^{r+ns}$, either finds an $x \in \mathbb{Z}^n$ satisfying $l \leq x \leq u$ and $A^{(n)}x = b$ or asserts that none exists, in time polynomial in $n$ and $(l, u, b)$.

**Proof.** If $l \not\leq u$ then assert that there is no feasible point and stop. Assume then that $l \leq u$ and determine some $x \in \mathbb{Z}^n$ satisfying $l \leq x \leq u$. Now, introduce $n(2r+2s)$ auxiliary variables and denote by $\hat{x}$ the resulting vector of $n(t+2r+2s)$ variables. Suitably extend the lower and upper bound vectors to $\hat{l}, \hat{u}$ by setting $\hat{l}_j := 0$ and $\hat{u}_j := \infty$ for each auxiliary variable $\hat{x}_j$. Consider the auxiliary integer program of finding an integer vector $\hat{x}$ that minimizes the sum of auxiliary variables subject to the lower and upper bounds $l \leq \hat{x} \leq u$ and the following system of equations, with $I_r$ and $I_s$ the $r \times r$ and $s \times s$ identity matrices respectively,

$$
\begin{pmatrix}
A_1 & I_r & -I_r & 0 & 0 & A_1 & I_r & -I_r & 0 & 0 & \cdots & A_1 & I_r & -I_r & 0 & 0 \\
A_2 & 0 & 0 & I_s & -I_s & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_2 & 0 & 0 & I_s & -I_s & \cdots & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & A_2 & 0 & 0 & I_s & -I_s \\
\end{pmatrix}
\hat{x} = b.
$$

This is again an $n$-fold integer program, with an $(r+s)\times (t+2r+2s)$ bimatrix $\hat{A}$, where $\hat{A}_1 = (A_1, I_r, -I_r, 0, 0)$ and $\hat{A}_2 = (A_2, 0, 0, I_s, -I_s)$. Since $A$ is fixed, so is $\hat{A}$. It is now easy to extend the vector $x \in \mathbb{Z}^n$ determined above to a feasible point $\hat{x}$ of the auxiliary program. Indeed, put $\hat{b} := b - A^{(n)}x \in \mathbb{Z}^{r+ns}$; now, for $i = 1, \ldots, r+ns$, simply choose an auxiliary variable $\hat{x}_j$ appearing only in the $i$-th equation, whose coefficient equals the sign sign($\hat{b}_i$) of the corresponding entry of $\hat{b}$, and set $\hat{x}_j := |\hat{b}_i|$. Define $\hat{w} \in \mathbb{Z}^{n(t+2r+2s)}$ by setting $\hat{w} := 0$ for each original variable and $\hat{w} := 1$ for each auxiliary variable, so that minimizing $\hat{w}\hat{x}$ is equivalent to minimizing the sum of auxiliary variables. Consider the following auxiliary linear $n$-fold integer program,

$$
\min \{\hat{w}\hat{y} : \hat{y} \in \mathbb{Z}^N, \hat{A}^{(n)}\hat{y} = b, \hat{l} \leq \hat{y} \leq \hat{u}\}, \ N := n(t+2r+2s).
$$

Using the algorithm of Theorem 4.4 compute the Graver basis $\mathcal{G}(\hat{A}^{(n)})$ in polynomial time. Using this Graver basis and the feasible point $\hat{x}$, solve the auxiliary program using the algorithm of Theorem 3.15 in polynomial time. Since the auxiliary objective $\hat{w}\hat{y}$ is bounded below by zero, the algorithm will output an optimal solution $\hat{x}^\ast$. If the optimal objective value is positive, then the original $n$-fold program is infeasible, whereas if the optimal value is zero, then the restriction of $\hat{x}^\ast$ to the original variables is a feasible point $x^\ast$ of the original $n$-fold integer program. $\square$
4.2.2 Linear and Separable Convex N-Fold Minimization

We are now in position to show that separable convex $n$-fold integer programming can be done in polynomial time, a result established quite recently in [42].

**Theorem 4.10** For every fixed integer $(r + s) \times t$ bimatrix $A$, there is an algorithm that, given $n$, lower and upper bounds $l, u \in \mathbb{Z}_{\infty}^{nt}$, $b \in \mathbb{Z}^{r + ns}$, and separable convex function $f : \mathbb{Z}^{nt} \to \mathbb{Z}$ presented by comparison oracle, solves in time polynomial in $n$ and $\langle l, u, b, f \rangle$, with $f$ the maximum value of $|f(x)|$ over the feasible set, the problem

$$\min \{ f(x) : x \in \mathbb{Z}^N, A^{(n)}x = b, l \leq x \leq u \}, \quad N := nt.$$  

*Proof.* First, apply the algorithm of Theorem 4.4 and compute the Graver basis $\mathcal{G}(A^{(n)})$. Next apply the algorithm of Theorem 4.9 (which uses this Graver basis) and either conclude that the program is infeasible or obtain a feasible point $x$. Now apply the algorithm of Theorem 3.14 using $\mathcal{G}(A^{(n)})$ and $x$ and solve the problem, namely, either detect that the feasible set is infinite or obtain an optimal solution. \qed

In particular, we conclude as a special case the following fundamental result of [20], which is the article that inaugurated the theory of $n$-fold integer programming.

**Theorem 4.11** For every fixed $(r + s) \times t$ integer bimatrix $A$ there is an algorithm that, given $n$, lower and upper bounds $l, u \in \mathbb{Z}_{\infty}^{nt}$, $w \in \mathbb{Z}^{nt}$, and $b \in \mathbb{Z}^{r + ns}$, solves in time polynomial in $n$ and $\langle l, u, w, b \rangle$, the linear $n$-fold integer programming problem

$$\max \{ wx : x \in \mathbb{Z}^N, A^{(n)}x = b, l \leq x \leq u \}, \quad N := nt.$$  

We proceed to show that $n$-fold separable convex functions can be minimized over $n$-fold systems in polynomial time. This is a strong and important result which will be used in §4.4 to solve congestion-avoiding multicommodity flow problems.

We need the following simple lemma.

**Lemma 4.12** For every fixed integer $(r+s) \times t$ bimatrix $A$ and $(p+q) \times t$ bimatrix $W$, there is an algorithm that, given any positive integer $n$, computes in time polynomial in $n$, the Graver basis $\mathcal{G}(B)$ of the following $(r + ns + p + nq) \times (nt + p + nq)$ matrix,

$$B := \begin{pmatrix} A^{(n)} & 0 \\ W^{(n)} & I \end{pmatrix}.$$  

**Theorem 4.13** For every fixed integer $(r + s) \times t$ bimatrix $A$ and integer $(p + q) \times t$ bimatrix $W$, there is an algorithm that, given $n$, lower and upper bounds $l, u \in \mathbb{Z}_{\infty}^{nt}$ and $\hat{l}, \hat{u} \in \mathbb{Z}_{\infty}^{p + nq}$, $b \in \mathbb{Z}^{r + ns}$, and separable convex function $f : \mathbb{Z}^{p + nq} \to \mathbb{Z}$ presented by a comparison oracle, solves in time polynomial in $n$ and $\langle l, u, \hat{l}, \hat{u}, b, f \rangle$, with $f$ the maximum value of $|f(W^{(n)}x)|$ over the feasible set, the convex minimization problem

$$\min \{ f(W^{(n)}x) : x \in \mathbb{Z}^N, A^{(n)}x = b, l \leq x \leq u, \hat{l} \leq W^{(n)}x \leq \hat{u} \}, \quad N := nt.$$
Proof. First, apply the algorithm of Lemma 4.12 and compute the Graver basis \( G(B) \) of the extended matrix

\[
B := \begin{pmatrix}
A^{(n)} & 0 \\
W^{(n)} & I
\end{pmatrix}.
\]

Next apply the algorithm of Theorem 4.9 (which uses this Graver basis) and either conclude that the program is infeasible or obtain a feasible point \( x \). Now apply the algorithm of Theorem 3.16 using \( G(B) \) and \( x \) and solve the problem, namely, either detect that the feasible set is infinite or obtain an optimal solution. 

4.2.3 Convex N-Fold Maximization

Finally, we can also obtain the following result of [21] on convex \( n \)-fold maximization.

**Theorem 4.14** For every fixed \( d \) and \((r + s) \times t\) integer bimatrix \( A \) there is an algorithm that, given \( n \), lower and upper bounds \( l, u \in \mathbb{Z}_\infty \), integer \( d \times nt \) matrix \( W \), \( b \in \mathbb{Z}^{r+ns} \), and convex function \( f: \mathbb{Z}^d \rightarrow \mathbb{R} \) presented by a comparison oracle, solves in time polynomial in \( \langle l, u, W, b \rangle \), the convex \( n \)-fold integer maximization problem

\[
\max \{ f(Wx) : x \in \mathbb{Z}^N, A^{(n)}x = b, l \leq x \leq u \}, \quad N := nt.
\]

**Proof.** First, apply the algorithm of Theorem 4.4 and compute the Graver basis \( G(A^{(n)}) \). Next apply the algorithm of Theorem 4.9 (which uses this Graver basis) and either conclude that the program is infeasible or obtain a feasible point \( x \). Now apply the algorithm of Theorem 3.19 using \( G(A^{(n)}) \) and \( x \) and solve the problem, namely, either detect that the feasible set is infinite or obtain an optimal solution. 

4.2.4 Graver Approximation Scheme

The algorithms for nonlinear \( n \)-fold integer programming provided in the previous subsections rely on the construction of the Graver basis of \( A^{(n)} \). Although the time needed for constructing \( G(A^{(n)}) \) is polynomial, it can be as large as \( n^{2g(A)} \) with a possibly quite large exponent \( g(A) \). We now suggest a certain Graver approximation scheme, parameterized by an integer \( 1 \leq k \leq g(A) \), which gradually uses the increasingly better approximations of \( G(A^{(n)}) \) constructed by the scheme of Algorithm 4.6. It provides increasingly better approximations at increasingly longer running times, ending up with the true optimal solution in polynomial time. For simplicity we consider the separable convex minimization problem of Theorem 4.10, that is,

\[
\min \{ f(x) : x \in \mathbb{Z}^N, A^{(n)}x = b, l \leq x \leq u \}, \quad N := nt,
\]

and we assume that we are given an initial feasible solution which we denote by \( x^*_0 \).

**Algorithm 4.15** (Graver approximation scheme for \( n \)-fold programming).
1. Use Algorithm 4.6 to obtain the $k$-th approximation $G^n_k$ of $\mathcal{G}(A^{(n)})$.

2. Use the algorithm of Theorem 3.14 with $G^n_k$ instead of $\mathcal{G}(A^{(n)})$ to iteratively augment the initial solution $x^*_{k-1}$ to the best attainable one $x^*_k$ and output it.

To estimate the quality of the approximation $x^*_k$, one could compute in polynomial time the (approximated) optimal objective value $f^*$ of the continuous relaxation

$$
\min \left\{ f(x) : x \in \mathbb{R}^N, A^{(n)}x = b, l \leq x \leq u \right\}, \quad N := nt,
$$

using one of various suitable available algorithms for minimizing a convex function over a polytope (see e.g. [36] and references therein). Then, depending on the ratio $f(x^*_k)/f^*$, a decision could be made of whether to settle for $x^*_k$ or increment $k$ and continue to obtain the better approximation $x^*_{k+1}$. Note that $f(x^*_k) \leq f(x^*_{k+1})$ for all $k$, and for $k = g(A)$ the Graver complexity of $A$, the solution $x^*_g(A)$ is a true optimal solution of the $n$-fold integer programming problem. Note also that, when settling for the $k$-th approximation $x^*_k$, the time needed for computing $G^n_k$ is $O(n^k)$ and can be much smaller than the time $O(n^{2g(A)})$ needed to construct the true Graver basis $\mathcal{G}(A^{(n)}) = G^n_{g(A)}$ and output the true optimal solution $x^* = x^*_g(A)$.

4.3 Stochastic Integer Programming

4.4 Some Applications

In this section we discuss some of the numerous applications of $n$-fold integer programming. We begin in §4.4.1 and §4.4.2 with linear and nonlinear multicommodity flows. In §4.4.1 we solve the nonlinear many-commodity transshipment problem of §1.3.3, which is easier to formulate, and its undirected analog. In §4.4.2 we solve the congestion-avoiding multicommodity transportation problem of §1.3.2. In §4.4.4 we discuss applications to packing problems and in particular to the classical cutting stock problem of [31]. We conclude in §4.4.5 by showing that the vector partitioning problems of §3.4 can also be formulated as $n$-fold integer programming problems.

4.4.1 Nonlinear Many-Commodity Transshipment in Polynomial Time

We now show that the many-commodity transshipment problem of §1.3.3 and its undirected analog are polynomial time solvable. Here the number $n$ of commodities is large and variable and the digraph or graph are small and fixed. We assume that all commodity types have the same volume per unit flow. The objective function is separable convex which, as explained in the introduction, better reflects the increase of the cost of flow over a channel due to congestion. In particular, we obtain the first polynomial time algorithm for the problem with a linear objective function.

**Corollary 4.16** For every fixed digraph $G$ with $s$ vertices and $t$ edges, there is an algorithm that, given $n$ commodity types, demand $d^k \in \mathbb{Z}^s$ for each commodity, edge
For every fixed number \( n \), and separable convex function \( f : \mathbb{Z}^t \to \mathbb{Z} \) presented by a comparison oracle, solves in time polynomial in \( n \) and \( \langle u, d^k, \hat{f} \rangle \), with \( \hat{f} \) the maximum value of \( |f(\sum_k x^k)| \) over the feasible set, the many-commodity transshipment problem,

\[
\min \left\{ f \left( \sum_{k=1}^{n} x^k \right) : \ x^k \in \mathbb{Z}^t, \ 0 \leq \sum_{k=1}^{n} x^k \leq u, \ \sum_{e \in \delta^+(v)} x^k_e - \sum_{e \in \delta^-(v)} x^k_e = d^k_v \right\}.
\]

Proof. We represent each many-commodity transshipment as \( x = (x^0, x^1, \ldots, x^n) \), consisting of \( n \) bricks corresponding to the given commodity types with an extra brick \( x^0 := u - \sum_{k=1}^{n} x^k \) representing a “slack” commodity, where each brick is indexed as \( x^k = (x^k_1, \ldots, x^k_t) \) with \( x^k_j \) the number of units of flow of commodity of type \( k \) routed along edge \( j \). Let \( D \) be the \( s \times t \) incidence matrix of \( G \) and \( A := \emptyset D \) the corresponding \((t+1) \times t\) bimatrix. Define a new separable convex function \( g : \mathbb{Z}^t \to \mathbb{Z} \) by \( g(y) := f(u - y) \). Let \( d^0 := Du - \sum_{k=1}^{n} d^k \) and let \( b = (d^0, d^1, \ldots, d^n) \). Then the problem becomes the \((n+1)\)-fold integer programming problem

\[
\min \left\{ g(x^0) : \ x = (x^0, \ldots, x^n) \in \mathbb{Z}^N, \ A^{(n+1)} x = b, \ x \geq 0 \right\}, \quad N := (n+1)t.
\]

By Theorem 4.10 this problem can be solved in polynomial time as claimed. \( \square \)

### 4.4.2 Nonlinear Multicommodity Transportation in Polynomial Time

We can finally provide a polynomial time solution for the congestion-avoiding multicommodity transportation problem discussed in §1.3.2 of the introduction. As explained therein, following [65], due to route congestion when subject to heavy traffic or communication load, the transportation delay or cost on a route are more realistically modeled by a nonlinear convex function of the volume \( y_{i,j} \) of flow over it, such as \( f_{i,j}(y_{i,j}) = w_{i,j}|y_{i,j}|^{\alpha_{i,j}} \) for suitable \( \alpha_{i,j} > 1 \), resulting in nonlinear, separable convex, objective function \( f(y) := \sum_{i,j} f_{i,j}(y_{i,j}) \) to be minimized. We have the following appealing and important consequence of our theory.

**Corollary 4.17** For every fixed number \( m \) of suppliers, \( t \) of commodity types, and \( v_k \) of volume per unit commodity of type \( k \), there is an algorithm that, given \( n \), supply vectors \( r^k \in \mathbb{Z}^m \), consumption vectors \( c^k \in \mathbb{Z}^n \), capacity \( u \in \mathbb{Z}^m \times \mathbb{Z}^n \), and separable convex function \( f : \mathbb{Z}^{m \times n} \to \mathbb{Z} \) presented by a comparison oracle, solves in time polynomial in \( n \) and \( \langle u, r^k, c^k, \hat{f} \rangle \), with \( \hat{f} \) the maximum value of \( |f(\sum_k v_k y^k)| \) over the feasible set, the congestion-avoiding multicommodity transportation problem,

\[
\min \left\{ f \left( \sum_{k} v_k y^k \right) : \ y^k \in \mathbb{Z}^{m \times n}, \ \sum_{j} y_{i,j}^k = r_i^k, \ \sum_{i} y_{i,j}^k = c_j^k, \ 0 \leq \sum_{k} v_k y^k \leq u \right\}.
\]

Proof. We represent a multicommodity transportation as a vector \( x = (x^1, \ldots, x^n) \), consisting of \( n \) bricks corresponding to consumers, where each brick is indexed as

\[
x^j = (x^j_{i,1}, \ldots, x^j_{i,t}, \ldots, x^j_{m,1}, \ldots, x^j_{m,t})
\]
with $x^j_{i,k}$ the number of units of flow of commodity of type $k$ from supplier $i$ to consumer $j$. We now construct bimatrices $A$ and $W$ as follows. Let $D$ be the $(t + 0) \times t$ bimatrix with $D_1 := I_t$ and $D_2$ empty. Let $V$ be the $(0 + 1) \times t$ bimatrix with $V_1$ empty and $V_2 := (v_1, \ldots, v_t)$. Let $A := \square(D^{(m)})$ be the $(mt + t) \times mt$ bimatrix with $A_1 := I_{mt}$ and $A_2 := D^{(m)}$. Finally, let $W$ be the $(0 + m) \times mt$ bimatrix with $W_1$ empty and $W_2 := V^{(m)}$. Then $A^{(n)}x$ is an $(mt + nt)$-vector, whose first $mt$ entries give the amount of flow of each commodity type from each supplier, and whose last $nt$ entries give the amount of flow of each commodity type to each consumer. Thus, arranging the $t$ supply vectors $r^k \in \mathbb{Z}^m$ and $t$ consumption vectors $c^k \in \mathbb{Z}^n$ in a suitable $(mt + nt)$-vector $b$, the supply and consumption equations become precisely $A^{(n)}x = b$. Next, $W^{(n)}x$ is an $nm$-vector whose $(j, i)$-th entry gives the total volume of flow of all commodities from supplier $i$ to consumer $j$. Identifying $\mathbb{Z}^{m \times n} \cong \mathbb{Z}^{nm}$ (by ordering the entries column after column), we find that the capacity constraints $0 \leq \sum_k v_k y^k \leq u$ become $0 \leq W^{(n)}x \leq u$, and the objective function $f(\sum_k v_k y^k)$ becomes $f(W^{(n)}x)$. Thus, the congestion-avoiding multicommodity transportation problem becomes the following separable convex $n$-fold integer program,

$$\min \left\{ f(W^{(n)}x) : \ x \in \mathbb{Z}^N, \ A^{(n)}x = b, \ 0 \leq W^{(n)}x \leq u \right\}, \quad N := nt.$$

By Theorem 4.13 this problem can be solved in polynomial time as claimed. □

4.4.3 Three-Way Line-Sum Transportation Problems

4.4.4 Packing Problems and Cutting-Stock

4.4.5 Vector Partitioning and Clustering Revisited