COMBINATORICA 7 (1) (1987) 49—65

AN APPLICATION OF SIMULTANEOUS DIOPHANTINE APPROXIMATION IN COMBINATORIAL OPTIMIZATION

ANDRÁS FRANK and ÉVA TARDOS

Received 10 June 1985
Revised 3 June 1986

We present a preprocessing algorithm to make certain polynomial time algorithms strongly polynomial time. The running time of some of the known combinatorial optimization algorithms depends on the size of the objective function w. Our preprocessing algorithm replaces w by an integral valued $\bar{w}$ whose size is polynomially bounded in the size of the combinatorial structure and which yields the same set of optimal solutions as w.

As applications we show how existing polynomial time algorithms for finding the maximum weight clique in a perfect graph and for the minimum cost submodular flow problem can be made strongly polynomial.

Further we apply the preprocessing technique to make H. W. Lenstra's and R. Kannan's Integer Linear Programming algorithms run in polynomial space. This also reduces the number of arithmetic operations used.

The method relies on simultaneous Diophantine approximation.

1. Introduction

The input of a combinatorial optimization problem consists of two parts. The first describes the combinatorial structure in question, the second is a list of numerical data. The *length of an input* is the number of bits occurring in it. The *dimension of an input* is the number of data plus the number of bits in the description of the combinatorial structure. An algorithm runs in *polynomial time* if the number of bit operations is bounded by a polynomial in the input length.

Every algorithm in this paper will have rational numerical data. The *length of a rational number* p/q is $(\lceil \log (p+1)\rceil + \lceil \log (q+1)\rceil + 1)$ where $\lceil \alpha \rceil$ denotes the smallest integer N for which $N \geqq \alpha$. The *length of a vector* is the sum of the sizes of its coordinates.

By an (elementary) *arithmetic operation* we mean addition, multiplication, division, and comparison. An algorithm is *strongly polynomial* if it runs in polynomial time and consists of arithmetic operations and data transfers, and the number of these operations is polynomially bounded in the dimension of the input. Equivalently, an algorithm is strongly polynomial if it consists of arithmetic operations and data transfers, the number of these operations is polynomially bounded in the

This research was done while the authors were visiting the Institute for Operations Research, University of Bonn, West Germany (1984—85), and while the second author was visiting MSRI, Berkeley. Her research was supported in part by NSF Grant 8120790.

AMS subject classification (1980): 68 E 10

dimension of the input, and the length of the numbers occurring during the algorithm is bounded by a polynomial in the length of the input.

Let us fix some notation. The l_∞-norm of a vector $x=(x(1), \dots, x(n))$ is $\|x\|_\infty = \max(|x(i)|$ for $i=1, \dots, n)$. The l_1-norm of vector x is $\|x\|_1 = \sum(|x(i)|: i=1, \dots, n)$. For a vector w and a subset X of its coordinates let $w(X)=\sum(w(i): i\in X)$. The sign of a real number x will be denoted by sign (x) (sign (x) takes values 0, $+1$, and -1). Let $\lfloor\alpha\rfloor$ denote the integer part of α, that is, the largest integer N for which $N \leqq \alpha$.

By definition a strongly polynomial algorithm only consists of elementary arithmetic operations and data transfers. Thus the scaling algorithm, first introduced by Edmonds and Karp [9] for the minimum cost flow problem, is, *a priori*, not strongly polynomial since it uses the operation of taking the integer part of a number; Moreover, the number of arithmetic operations used depends on the size of the numerical data.

Note that strongly polynomial algorithms are not allowed to take the integer part of an arbitrary number α but if α is a rational number such that $\lfloor\alpha\rfloor$ is of polynomial length, then $\lfloor\alpha\rfloor$ can be computed via binary search by using a polynomial number of elementary operations. We will exploit this option.

Another, more significant class of polynomial time but not strongly polynomial algorithms was developed via the ellipsoid method [12].

The purpose of the present paper is to develop a device by which certain kinds of polynomial time algorithms can be made strongly polynomial. A significant feature of the procedure is that it is fairly independent of the algorithm to be made strongly polynomial.

To explain the idea, let us consider the problem of finding a maximum weight clique in a perfect graph $G=(V, E)$. As a nice application of the ellipsoid method, Grötschel, Lovász and Schrijver [12] gave a polynomial time algorithm for this problem. The algorithm is not strongly polynomial since the number of arithmetic operations used depends on the length of the weight function w. To overcome this difficulty we shall construct another weight function $\bar{w}$ such that

(i) $\bar{w}(X) \leqq \bar{w}(Y)$ if and only if $w(X) \leqq w(Y)$ for every $X, Y \subseteq V$, and
(ii) the length of $\bar{w}$ is polynomial in $|V|$.

By (i) a clique of G is optimal with respect to w if and only if it is optimal with respect to $\bar{w}$. Furthermore, because of (ii), the number of arithmetic operations used by the Grötschel, Lovász, Schrijver algorithm when it is applied to $\bar{w}$, can be bounded by a polynomial in $|V|$. Consequently, the algorithm, becomes strongly polynomial. (Notice that this idea has nothing to do with specific properties of perfect graphs, nor with the Grötschel, Lovász, Schrijver algorithm. Whenever a polynomial time algorithm is available to find the maximum weight member of a hypergraph, by applying the above idea, we can make it strongly polynomial).

Besides the above application, the preprocessing algorithm will make it possible to turn an existing polynomial time algorithm for the submodular flow problem [4] into a strongly polynomial algorithm. One special case of this provides a strongly polynomial algorithm for the minimum cost circulation problem. This latter problem has been solved earlier by the second author [17]. Her minimum cost circulation algorithm has the advantage that it does not rely on simultaneous approximation.

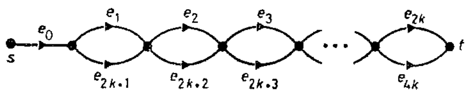
See also [18]. The present method also applies to the problem of testing membership in a matroid polyhedron. A strongly polynomial algorithm for this last problem had earlier been developed by W. H. Cunningham [3].

In Section 5 we show how the preprocessing technique applies to improve the running time of some Integer Linear Programming algorithms and to make these algorithms run in polynomial space. H. W. Lenstra, Jr. [16] invented an algorithm to solve Integer Linear Programming. The algorithm runs in polynomial time if the dimension is fixed. The algorithm was later improved by R. Kannan [14]. The running time of these algorithms is a great improvement over complete enumeration even when the dimension is not fixed. On the other hand, these algorithms have a drawback: complete enumeration can be performed in polynomial space and both of the above algorithms require exponential space. Both algorithms reduce a problem in n variables to a finite numbers of problems in less than n variables. Unfortunately, the length of the number occurring in the new problem can only be bounded by a polynomial in the input length. Thus applying this reduction n times we might get numbers whose length is exponential in the length of the description of the original problem. That is, the algorithm does not run in polynomial space. Applying the preprocessing technique we can keep the lengths of all numbers occurring throughout the algorithms polynomially bounded in the length of the description of the problem (even if the number of variables is not fixed). Applying this to R. Kannan's algorithm reduces the number of arithmetic operations used from $O(n^{4.5n})$ to $O(n^{2.5n})$.

Remark. The max flow-min cut algorithm of Ford and Fulkerson is known to be not polynomial and for irrational capacities it is not even finite. The strongly polynomial algorithms of Dinits [5] and of Edmonds and Karp [9] overcome both difficulties. In view of this, the concept we are using here for strong polynomiality is seemingly weaker since it assumes rational numbers in the input.

Since a strongly polynomial algorithm uses only elementary operations, it works just as well with real input and in this case the number of elementary operations continues to be polynomially bounded. This does not mean, however, that the algorithm is a polynomial time algorithm since the numbers occuring in the course of the algorithm may be too large. This phenomenon may occur with the Dinits and Edmonds—Karp algorithms as is illustrated by the following example.

Let us consider a network with capacities given in the form $\log a$ ($a > 1$ rational). The above maximum flow algorithms remain well defined for the maximum flow problem on such a network since they do not use multiplication or division of capacities, and the difference of two numbers $\log a$ and $\log b$ has the same form, namely $\log (a/b)$. Yet, the algorithms do not run in polynomial time for this problem. The length of the numbers occurring throughout the algorithm might be exponential in the length of the problem. (This can be shown on the following network:



Let us define the capacity of e_0 and of every lower edge to be $\log M$ where M is a big number but its length is polynomial ($M=16^{1+k}$ will do). The capacities of the upper edges will be smaller such that e_0 and the lower edges define a maximum flow of value $\log M$.

We are going to define (recursively) the capacities of the upper edges in such a way that all augmenting paths contain e_0 and have exactly three upper edges. Namely the $(2i+1)$'th augmenting path ($i=0, 1, \dots, k-2$) has $e_{2i+1}, e_{2i+3}, e_{2i+4}$ as upper edges, the $(2i+2)$'th augmenting path has $e_{2i+2}, e_{2i+3}, e_{2i+4}$ as upper edges.

The flow increment during the $(2i+1)$'th and the $(2i+2)$'th augmentation is the same: $\log \Delta_i$. The capacity of the edges e_{2i+1} and e_{2i+2} is the same and denoted by $\log b_i$. These values can be chosen such as to satisfy: $b_0=2$, $\Delta_1=b_0$, $\Delta_i=b_{i-1}/\Delta_{i-1}^2$ where $1<\Delta_i<3$ for $i=2, 4, 6, \dots$ and $1<\Delta_i<2$ for $i=3, 5, \dots$ and each of b_2, $b_4, b_6, \dots$ is one of 2, 4, 8, 16 and each of $b_1, b_3, b_5 \dots$ is one of 3, 9.

With such a choice the increment of every augmenting path is determined by its first upper edge. One can easilly see that the length of Δ_{i+1} is the double of the length of Δ_i.)

2. Simultaneous Diophantine approximation

The main ingredient of the preprocessing algorithm is Lovász's simultaneous (Diophantine) approximation algorithm in [15].

Dirichlet's fundamental theorem on simultaneous approximation is as follows.

Theorem. [2, Sect. 1.10] *Given a positive integer N and n real numbers $\alpha(1), \dots, \alpha(n)$; there are integers $p(1), \dots, p(n)$ and q such that $1 \leqq q \leqq N^n$ and*

$$|q\alpha(i)-p(i)| < \frac{1}{N} \quad \textit{for} \quad i=1, \dots, n. \blacksquare$$

In [15], a polynomial time algorithm is developed which for rational $\alpha(1), \dots$ $\dots, a(n)$ finds a slightly weaker approximation:

Theorem. [15] *There exists an algorithm which uses at most n^6B arithmetic operations and, for a given integer N and rational numbers $\alpha(1), \dots, \alpha(n)$, finds integers $p(1), \dots, p(n)$ and q such that $1 \leqq q \leqq 2^{n^2}N^n$, and*

$$|q\alpha(i)-p(i)| < \frac{1}{N} \quad \textit{for} \quad i=1, \dots, n.$$

Here B is an upper bound on the sizes of the $\alpha(i)$'s and N. Moreover, the numbers occurring in the course of the computation have size at most $O(n^3B)$. $\blacksquare$

The algorithm is not strongly polynomial since the number of arithmetic operations used depends on B. However, we shall need it only in the special case when $-1 \leqq \alpha(i) \leqq 1$ ($i=1, \dots, n$). For this case a slight modification will make the number of arithmetic operations performed by the algorithm bounded by a polynomial in n and the length of N (and independent of the lengths of the $\alpha(i)$'s) at the expense of further weakening the bound on q. In the applications for turning polynomial time algorithms into strongly polynomial ones we will have $N \leqq (n+1)!+1$.

In this case (since $\log N \leqq n^2$) the modified algorithms are strongly polynomial. The modification is done as follows.

First find an individual approximation of each $\alpha(i)$ with precision $2^{-n^2-n-1}N^{-n-1}$. That is, let

$$\alpha'(i) = \frac{\lfloor \alpha(i)2^{n^2+n+1}N^{n+1}\rfloor}{2^{n^2+n+1}N^{n+1}}.$$

Then choose $N'=2N$ and apply Lovász's approximation algorithm to N' and $\alpha'(1), \dots, \alpha'(n)$. Since the size of these numbers is polynomially bounded in n and the size of N (namely, at most $(n+1)^2+(n+1)\log N$) the number of arithmetic operations performed by Lovász's algorithm in this case depends only on n and the length of N. Suppose the algorithm produces integers $p(1), \dots, p(n)$, and q.

Claim. $|q\alpha(i)-p(i)|<N^{-1}$ for $i=1, \dots, n$ *and* $1 \leqq q \leqq 2^{n^2+n}N^n$.

Proof. Clearly $1 \leqq q \leqq 2^{n^2}(N')^n = 2^{n^2+n}N^n$.
To prove the first inequality we have

$$|q\alpha(i)-p(i)| \leqq |q\alpha(i)-q\alpha'(i)|+|q\alpha'(i)-p(i)|$$

$$< q|\alpha(i)-\alpha'(i)|+\frac{1}{2N} \leqq \frac{2^{n^2+n}N^n}{2^{n^2+n+1}N^{n+1}}+\frac{1}{2N} = \frac{1}{N}. \blacksquare$$

We refer to this version of the algorithm as the *revised simultaneous approximation algorithm*. As we have noted, the number of arithmetic operations performed by Lovász's algorithm when applied to $\alpha'(i)$ $(i=1, \dots, n)$ and N' is polynomially bounded in n and $\log N$. The same is true for the revised simultaneous approximation algorithm since, by the remark made on integer parts of "small" numbers in the Introduction, $\alpha'(i)$ can be calculated in strongly polynomial time. (Here we exploit the fact that $-1 \leqq \alpha(i) \leqq 1$.)

3. The preprocessing algorithm

The basis of our algorithm is the following theorem concerning the structure of a vector. The theorem states that every vector w can be expressed as a positive linear combination of at most n "small" integer vectors v_i (that is, $w=\sum \lambda_i v_i$ with $\lambda_i>0$) such that the coefficients λ_i are "quickly" decreasing.

Theorem 3.1. *For every vector $w \in \mathbf{R}^n$ and positive integer N there exist integer vectors $v_1, \dots, v_k$ $(k \leqq n)$ and positive scalars $\lambda_1, \dots, \lambda_k$ such that*

(i) $$w = \sum_{i=1}^{k} \lambda_i v_i.$$

(ii) $$\|v_i\|_\infty \leqq N^n \quad i = 1, \dots, k, \quad \textit{and}$$

(iii) $$\frac{\lambda_i}{\lambda_{i-1}} \leqq \frac{1}{N\|v_i\|_\infty} \quad i = 2, \dots, k.$$

Proof. If all coordinates of w are zero, then the statement is trivial (the empty sum is considered to give the zero vector). Otherwise let $w_0=w$ and define

$$w_0' = \frac{w_0}{\|w_0\|_\infty}.$$

Apply simultaneous Diophantine approximation to the integer N and the coordinates of w_0'. Let $p_1(1), \ldots, p_1(n)$ and q_1 be the resulting integers. Define $v_1=(p_1(1), \ldots, p_1(n))$,

$$\lambda_1 = \frac{\|w_0\|_\infty}{q_1}$$

and $w_1=w_0-\lambda_1 v_1$.

Proceeding the same way with w_1 (defining w_1', v_2, λ_2 and w_2 etc.) after at most n iterations we get $w_k=0$. Indeed, $w_i=w_{i-1}-\lambda_i v_i$ has strictly more zero coordinates than w_{i-1} has: For each index j such that $w_i(j)=0$ we have also $p_i(j)=0$ and so $w_i(j)=0$. Moreover, if $|w_i(j)|=\|w_i\|_\infty$, then $w_i'(j)=\pm 1$. This implies that $p_{i+1}(j)=\pm q_{i+1}$ and so $w_{i+1}(j)=0$.

Now we have

$$w = \sum_{i=1}^{k} \lambda_i v_i,$$

where $\lambda_i>0$ $(i=1, \ldots, k)$ and $k \leqq n$. By Dirichlet's theorem $\|v_i\|_\infty \leqq N^n$ for $i=1, \ldots, k$.

Finally we have to prove that condition (iii) is satisfied. By the simultaneous Diophantine approximation we have

$$\frac{1}{\lambda_i}\|w_i\|_\infty = \left\|\frac{1}{\lambda_i} w_{i-1} - v_i\right\|_\infty = \|q_i w_{i-1}' - v_i\|_\infty < \frac{1}{N} \quad (i=1, \ldots, k).$$

Further, as $\|w_i'\|_\infty = 1$ we have $\|v_i\|_\infty = q_i$. So

$$\frac{\lambda_i}{\lambda_{i-1}} = \frac{\|w_{i-1}\|_\infty}{\lambda_{i-1} q_i} < \frac{1}{N q_i} = \frac{1}{N\|v_i\|_\infty} \quad (i=2, \ldots, k),$$

as required. ∎

We can use the revised simultaneous approximation algorithm to turn the proof of the theorem into a *decomposing algorithm* (whose number of arithmetic operations is polynomially bounded in n and $\log N$) at the expense of weakening condition (ii) to

(ii)′ $$\|v_i\|_\infty \leqq 2^{n^2+n} N^n \quad (i = 1, \ldots, k).$$

We shall need the following lemma about the decomposition. The lemma expresses the fact that, since the coefficients λ_i are very quickly decreasing, the vectors v_i do not "interact" with each other when w is multiplied by a "small" integer vector b.

Lemma 3.2. *Let* $w=\sum_{i=1}^{k} \lambda_i v_i$ *such that* $\lambda_i>0$, *the vectors* v_i *are integral and the decomposition satisfies condition* (iii). *Let* b *be an integer vector with* $\|b\|_1 \leqq N-1$.

Then $\operatorname{sign}(b\cdot w)=\operatorname{sign}(b\cdot v_j)$ *where j is the smallest index for which* $b\cdot v_j\neq 0$. *Observe that, if* $b\cdot v_i=0$ *for every* $i=1,\dots,k$ *then* $b\cdot w=0$ *as well.*

Proof. Let j be the minimal index such that $b\cdot v_j\neq 0$. We may assume without loss of generality that $b\cdot v_j>0$. We proceed by induction on $l=j,\dots,k$ to prove that

$$\sum_{i=1}^{l}\lambda_i bv_i \geqq \lambda_l .$$

The case $l=j$ is trivial since $b\cdot v_i=0$ for $i<j$ and both b and v_j are integral (thus $b\cdot v_j>0$ implies $b\cdot v_j\geqq 1$). Let $j<l\leqq k$.

$$\begin{aligned} b\Big(\sum_{i=1}^{l}\lambda_i v_i\Big) &= b\Big(\sum_{i=1}^{l-1}\lambda_i v_i\Big)+\lambda_l bv_l \geqq \lambda_{l-1}+\lambda_l bv_l \\ &\geqq \lambda_{l-1}-\lambda_l\|b\|_1\|v_l\|_\infty \geqq \lambda_{l-1}-\lambda_l(N-1)\|v_l\|_\infty \\ &\geqq \lambda_l, \end{aligned}$$

as required. Here for the first inequality we used the induction hypothesis, for the last one condition (iii). This proves the lemma. ∎

We proceed to describe an algorithm which, given a rational vector $w= =w(1),\dots,w(n)$ and an integer N, finds an integral vector $\bar w$ such that the size of $\bar w$ is polynomial in n and $\log N$, and no hyperplane of the form $\{x\colon b\cdot x=0\}$ with $\|b\|_1\leqq N-1$ separates w and $\bar w$. In particular, taking $N=n+1$, we obtain an integral vector $\bar w$ such that the size of $\|\bar w\|_\infty$ (that is, $\log\|\bar w\|_\infty$) is $O(n^3)$ and $\bar w(X)\leqq\bar w(Y)$ if and only if $w(X)\leqq w(Y)$ for every subset X and Y of the coordinates.

Preprocessing algorithm

Input. *A rational vector* $w=(w(1),\dots,w(n))$ *and an integer* N.

Output. *An integral vector* $\bar w=(\bar w(1),\dots,\bar w(n))$ *such that* $\|\bar w\|_\infty\leqq 2^{4n^3}N^{n(n+2)}$ and $\operatorname{sign}(w\cdot b)=\operatorname{sign}(\bar w\cdot b)$ *whenever b is an integer vector with* $\|b\|_1\leqq N-1$.

Step 1. Apply the above decomposing algorithm to w and N. It terminates with

$$w=\sum_{i=1}^{k}\lambda_i v_i$$

such that $\lambda_i>0$ and vector v_i is integral $(i=1,\dots,k)$, $k\leqq n$ and the decomposition satisfies conditions (ii)' and (iii) with the integer N.

Step 2. Let $M=2^{n^2+n}N^{n+1}$ and put

$$\bar w=\sum_{i=1}^{k}M^{k-i}v_i .$$

End.

Theorem 3.3. *The vector $\bar{w}$ provided by the preprocessing algorithm satisfies the output criteria.*

Proof. First consider the upper bound on $\|\bar{w}\|_\infty$.

$$\|\bar{w}\|_\infty \leqq \sum_{i=1}^{k} M^{k-i}\|v_i\|_\infty \leqq \sum_{i=1}^{k} M^{k-i}2^{n^2+n}N^n$$

$$\leqq M^k 2^{n^2+n}N^n \leqq 2^{n^3+2n^2+n}N^{n(n+2)} \leqq 2^{4n^3}N^{n(n+2)},$$

as required.

Next we observe that the decomposition

$$\bar{w} = \sum_{i=1}^{k} M^{k-i}v_i$$

satisfies condition (iii). Indeed,

$$\frac{1}{N}\frac{1}{\|v_i\|_\infty} \geqq \frac{1}{N}\frac{1}{2^{n^2+n}N^n} = \frac{1}{M} = \frac{M^{k-i}}{M^{k-(i-1)}}.$$

(Here the inequality is due to condition (ii)′.) So by Lemma 3.2, for any integer vector b with $\|b\|_1 \leqq N-1$ we have

$$\operatorname{sign}(bw) = \operatorname{sign}(bv_j) = \operatorname{sign}(b\bar{w})$$

where j is the smallest index such that $b\cdot v_j \neq 0$. ∎

4. Turning polynomial time algorithms into strongly polynomial

In this section we show how the preprocessing algorithm can be used to turn certain polynomial time linear programming algorithms into strongly polynomial ones. In [18] a strongly polynomial algorithm is developed to solve linear programs whose constraint matrix has size polynomial in the dimension of the problem (for example flow and multi-commodity flow problems). The method used in [18] applies to explicitly given linear programs only. For most combinatorial optimization problems the constraint matrix of the corresponding linear programming problem has 0, +1 and −1 coordinates. However, the number of inequalities in the linear program is often exponential in the length of the combinatorial description of the problem, and so the method used in [18] does not apply. The preprocessing algorithm applies also to these linear programs.

Let A be a matrix with 0 and ± 1 entries and let us consider the linear program $\max w\cdot x$ over the polyhedron $P=\{x\in \mathbf{R}^n\colon Ax\leqq b\}$. We are going to apply the preprocessing algorithm to show that whenever an algorithm is available to solve $\max(w\cdot x\colon x\in P)$ for a class of polyhedra P, whose running time is polynomially bounded in n and the length of w, it can be made strongly polynomial. The idea is that an arbitrary objective function w can be replaced by an integral function $\bar{w}$ whose size is polynomial in n and which yields the same set of optimal solutions and the same set of optimal dual bases as w. Having computed such a $\bar{w}$ we may apply the existing algorithm to the objective function $\bar{w}$ instead of w.

We shall consider the following primal and dual linear programs.

$$(1) \qquad \begin{array}{ll} \text{(a) } Ax \leqq b & \text{(b) } yA = w, \quad y \geqq 0 \\ \max wx & \min yb \end{array}$$

Let n and m denote the number of columns and rows of A, respectively. A vector $\bar{x}$ in $P=\{x\in\mathbf{R}^n\colon Ax\leqq b\}$ is called *w-maximal* if $w\cdot\bar{x}=\max(w\cdot x\colon x\in P)$. We call a maximal set of linearly independent rows of A a *dual basis*. Any dual basis B uniquely determines a vector y for which $yA=w$, provided that such a y exists at all, that is, w is in the rowspace of A. If $y\geqq 0$, then B is called a *feasible dual basis* and y is the *basic dual solution to* (1b) *corresponding to basis B*. If y is an optimal solution to (1b) then B is called an *optimal dual basis*.

Lemma 4.1. *Let w' and w'' be vectors such that* $\operatorname{sign}(w'\cdot h)=\operatorname{sign}(w''\cdot h)$ *for every integral vector h with* $\|h\|_1\leqq(n+1)!$. *Consider* (1a) *and* (1b) *with* $w=w'$ *and* $w=w''$.
(i) *A solution to* (1a) *is w'-maximal if and only if it is w''-maximal.*
(ii) *A dual basis B is optimal for* $w=w'$ *if and only if it is optimal for* $w=w''$.

Proof. Because of the symmetric role of w' and w'' it suffices to prove the "only if" part of both statements. Let B be an optimal dual basis and x an optimal solution to (1a) with $w=w'$. Further, let y' be the basic dual solution to (1b) corresponding to basis B. Let r denote the rank of A. Let B' be a non-singular r by r submatrix of the matrix formed by the rows of B. By elementary linear algebra, y' can be written in the form $y'=(y'_B, y'_N)$ with $y'_B=w'_{B'}(B')^{-1}, y'_N=0$. Here y'_B are those coefficients of y' corresponding to rows of B, y'_N are those coefficients of y' corresponding to rows not in B, and $w'_{B'}$, are those coefficients of w' corresponding to columns of B'. Let $q=|\det(B')|$. Multiply each entry of $(B')^{-1}$ by q and complete the resulting integer matrix with zero rows and columns to an n by m matrix H. Now we have that $y'=(1/q)w'H$. Since A is a 0, ± 1 matrix $q\leqq n!$ and the entries of H have absolute value at most $(n-1)!$.

Let $y''=(1/q)w''H$. We will show that $y''\geqq 0$ and $y''A=w''$. These two statements yield that B is a feasible dual basis of (1b) with $w=w''$. The corresponding basic solution is y''. Finally we shall prove that x and y'' satisfy the complementary slackness conditions. Thus x and y'' are optimal solutions to (1a) and (1b) with $w=w''$. This will complete the proof.

Let us start with proving that $y''A=w''$. We know that $y'A=w'$, that is $w'(HA-qI)=0$. The rows of the integer matrix $HA-qI$ have l_1-norm at most $n\cdot n!+n!=(n+1)!$. Thus, by the assumption, $w'(HA-qI)=0$ implies $w''(HA-qI)= =0$. This latter is equivalent to $y''A=w''$.

Next we show $y''\geqq 0$. We know that $y'\geqq 0$, that is, $(1/q)w'H\geqq 0$, or equivalently, $w'H\geqq 0$. The entries of the integer matrix H have absolute value at most $(n-1)!$. Thus, by the assumption again, $w'H\geqq 0$ implies $w''H\geqq 0$. This latter is equivalent to $y''\geqq 0$.

Finally we show that x and y'' satisfy the complementary slackness conditions; that is, if a coordinate of y'' is positive then x satisfies the corresponding inequality of (1a) as an equality. As in the proof that $y''\geqq 0$, we can see that a coordinate of y'' is positive if and only if the same coordinate of y' is positive. So complementary slackness applied to x and y' proves the required equalities for x. ∎

Let $\bar{w}$ be the output of the preprocessing algorithm when it is applied to w and $N=(n+1)!+1$. Theorem 3.1 and Lemma 4.1 give the following.

Theorem 4.2. (i) *$x\in P$ is w-maximal if and only if it is $\bar{w}$-maximal.*

(ii) *A set of rows of A is an optimal dual basis for* $\max(wx\colon Ax\leqq b)$ *if and only if it is an optimal basis for* $\max(\bar{w}x\colon Ax\leqq b)$. ∎

Note that Theorem 4.2 can be extended to the case where the entries of A are "small" integers: Let α be the maximal length of the entries of A. Theorem 4.2 holds for linear programs with constraint matrix A if we replace $N=(n+1)!+1$ by $N=(n+1)!2^{n\alpha}+1$. With this definition of N, the number of arithmetic operations used by the preprocessing algorithm and the size of the resulting $\bar{w}$ will be polynomially bounded in n and α. Thus, if the size of the entries of the matrix A is polynomially bounded in the number of variables, every algorithm which is polynomial in the size of the objective functions can be made strongly polynomial.

Now we discuss three combinatorial consequences of the above theorem.

(A) Let $G=(V,E)$ be a perfect graph with n nodes, and let w be a weight-function on the nodes. Grötschel, Lovász and Schrijver [12] described an algorithm whose running time is polynomial in the length of w and which finds the maximum weight clique in G. Since G is perfect the convex hull of the characteristic vectors of the cliques is

$$P=\{x\in\mathbf{R}^V\colon\ x\geqq 0,\ x(I)\leqq 1\ \textit{for all independent sets } I \textit{ of } G\}.$$

Theorem 4.2 applies to P. Thus, we can turn the Grötschel, Lovász, Schrijver algorithm into a strongly polynomial algorithm.

(B) As an application of the ellipsoid method, Grötschel, Lovász and Schrijver [12] developed a polynomial time algorithm to maximize a linear objective function over the submodular flow polyhedron of Edmonds and Giles [8]. In this algorithm the number of arithmetic operations depends on the length of both the costs and the capacities (that is, w and b). In [4] a combinatorial algorithm is described for the same problem. Both algorithms use a subroutine to minimize certain submodular functions. The second algorithm has the advantage that the number of calls to the subroutine and the number of arithmetic operations used depends only on the length of the costs (and not on the length of the capacities). (See [4] for details.)

A polynomial time algorithm for minimizing submodular functions was developed by Grötschel, Lovász and Schrijver [12] via the ellipsoid method. In [13] the same authors give a strongly polynomial algorithm. (Surprisingly, this strongly polynomial algorithm also relies on the ellipsoid method.)

Since the submodular flow polyhedron is defined by a matrix with 0 and ± 1 entries the preprocessing algorithm applies. Using the algorithm in [13] for minimizing submodular functions as the subroutine and the preprocessing algorithm to change the costs, the algorithm given in [4] becomes strongly polynomial.

(C) Let r be the rank function of a matroid on a finite set S. The convex hull of the characteristic vectors of the independent sets is

$$P=\{x\in\mathbf{R}^S\colon\ x\geqq 0\quad\text{and}\quad x(A)\leqq r(A)\quad\text{for}\quad A\subseteq S\}.$$

The membership problem is to test whether a certain vector w is in P, and if w is in P, then to express w as a convex combination of at most $n+1$ vertices of P.

The first polynomial time algorithm for the membership problem was found by Grötschel, Lovász and Schrijver [12] using the ellipsoid method. Then Cunningham [4] developed a sophisticated strongly polynomial algorithm which is purely combinatorial. Later Bixby (see in [1]) observed that Edmonds' matroid partition algorithm [6] and a scaling technique together yield a simple polynomial time algorithm (which, however, is not strongly polynomial).

Here we show that the preprocessing algorithm can be applied to turn Bixby's algorithm into a strongly polynomial algorithm. Let A be the matrix whose rows are the characteristic vectors of the independent sets. The vector w is in P if and only if w is the convex combination of the rows of A, that is, if there exists a vector $y \geqq 0$ such that $y \cdot \mathbf{1} = 1$ and $yA = w$. We apply the preprocessing algorithm to the linear programming dual of $\min(y \cdot \mathbf{O} : y \geqq \mathbf{O}, y \cdot \mathbf{1} = 1$ and $yA = w)$, that is, to the problem $\max(w \cdot x + t : Ax + t\mathbf{1} \leqq \mathbf{O})$. This maximum is zero or plus infinity according to whether or not w is in P. Thus Bixby's polynomial time algorithm can be applied to solve it. This results in a strongly polynomial algorithm to decide whether w is in P. The algorithm also finds an optimal dual basis to the maximization problem if w is in P. The basic dual solution corresponding to this basis is an expression of w as a convex combination of at most $n+1$ independent sets.

Remark. In applications (A) and (B) we may be interested in finding an optimal primal solution only, and not looking for the dual solution. In such a case it suffices to take a smaller $\bar{N}$ than the one used in Theorem 4.2. Let $\bar{x}$ be a vector in P and define $C(\bar{x}) = \{x : \exists \lambda > 0$ such that $\bar{x} + \lambda x \in P\}$.

Define $\bar{N}$ to be an integer such that all of the cones $C(\bar{x})$ for $\bar{x}$ in P can be generated by integer vectors with l_1-norm smaller than $\bar{N}$. A vector $\bar{x}$ in P is w-maximal if and only if $w \cdot x \leqq 0$ for all x in $C(\bar{x})$. This observation together with Theorem 3.3 gives the following.

Theorem 4.3. *Let $\bar{w}$ be the output of the preprocessing algorithm when applied to w and $N = \bar{N}$ ($\bar{N}$ defined in the above paragraph). Then a vector x in P is w-maximal if and only if it is $\bar{w}$-maximal.* ▮

In application (A) it is easy to see that $\bar{N} = n+1$ can be taken. In application (B) we can take $\bar{N} = |E| + 1$. This follows from a theorem of Zimmermann [19, Theorem 2.4] asserting that the difference of two submodular flows is a non-negative combination of circuits in the corresponding auxiliary digraph.

In application (C) we could use the preprocessing algorithm with $N = 2 \cdot |S| +$ $+1$: Let $(\bar{w}, t)$, a vector with $|S|+1$ coordinates, be the output of the preprocessing algorithm when applied to the vector $(w, 1)$ and $N = 2 \cdot |S| + 1$. Theorem 3.3 implies that w is in P if and only if $(1/t)\bar{w}$ is in P. In this way we can decide whether w is in P, but we cannot directly express w as a convex combination of vertices of P.

5. Integer linear programming

Finally, we turn to integer linear programming algorithms. The Integer Linear (feasibility) Problem is to determine whether or not there exists a vector of integers satisfying a given system of linear inequalities. H. W. Lenstra Jr. [16] invented an algorithm to solve this problem. The algorithm runs in polynomial time if the number of variables is bounded. For a polyhedron P the algorithm either finds an integer point in P or finds an integer vector c such that the maximum value and the minimum value of $c \cdot x$ over the polyhedron P differ by at most β^{n^2} where β is a constant independent of n. Every integer point must lie on a hyperplane $c \cdot x = d$ for some integer d. Thus this reduces the n-dimensional problem to at most β^{n^2} $(n-1)$-dimensional problems.

This algorithm was improved by R. Kannan [14]. Kannan's algorithm either finds an integer point in P or finds a set of linearly independent vectors $c_1, \ldots, c_k$ such that there are at most $O(n^{2.5k})$ vectors $(d_1, \ldots, d_k)$ with $P \cap \{x : c_i \cdot x = d_i \; i = 1, \ldots, k\}$ not empty. (Here $1 \leqq k \leqq n$ is an appropriate integer chosen by the algorithm.) This reduces the n-dimensional problem to $O(n^{2.5k})$ $(n-k)$-dimensional problems. The reduction uses $O(n^n s)$ arithmetic operations, where s is the length of the description of the polyhedron. The resulting $(n-k)$-dimensional problems have length at most $O(n^2 s)$. Thus the number of arithmetic operations used by the whole algorithm satisfies the following recursion

$$T(n, s) \leqq O(n^{2.5k}) T(n-k, n^2 s) + O(n^n s)$$

for some $1 \leqq k < n$. So we have $T(n, s) \leqq O(n^{4.5n} s)$.

In both algorithms, the length of the reduced problems can only be bounded by a polynomial in the length of the original problem. Thus applying the reduction $O(n)$ times we might get numbers whose length is exponential in the length of the description of the original problem. That is, these algorithms do not run in polynomial space.

We can use the preprocessing technique to keep the length of the numbers bounded by a polynomial in the length of the input. This is done as follows.

Let us apply one of the above two algorithms to the n variable problem given by the linear inequalities $Ax \leqq b$. The reduction results in some $(n-k)$-dimensional problems. Let us keep these $(n-k)$-dimensional problems in the n-variable representation, as

$$Ax \leqq b$$

$$c_i x = d_i \quad i = 1, \ldots, k$$

with k linearly independent vectors c_i, $i = 1, \ldots, k$. We are going to use the preprocessing technique to replace the vectors c_i $(i = 1, \ldots, k)$ by small integral vectors. We have to be a little careful to maintain the linear independence of the equalities.

We may assume that matrix A and vector b are integral. Let α be the maximum length of the coefficients in A and b.

Further, the polyhedron $P = \{x : Ax \leqq b\}$ contains an integer point if and only if it contains an integer point with coordinates at most $(n+1)n^{n/2}2^{n\alpha}$ in absolute value. (See [16].) Thus, we may add the inequalities $-(n+1)n^{n/2}2^{\alpha n} \leqq x_i \leqq (n+1)n^{n/2}2^{\alpha n}$

for all coordinates x_i $(i=1, \ldots, n)$ of x. This increases the length of the description only polynomially. Thus we may assume in the following that P is bounded.

Whenever one of the equalities $c_i \cdot x = d_i$ contains numbers with length larger than $2n^3 \log(\alpha+1)$ throughout the algorithm we apply the following subroutine.

Subroutine for changing $c_i \cdot x = d_i$

(1) Let $N = n \cdot n! 2^{n\alpha}+1$, where α is the maximum length of the coefficients in A and b.

(2) By applying the decomposition algorithm to the $(n+1)$-dimensional vector $w=(c_j, d_j)$ we obtain

$$(c_i, d_i) = \sum_{l=1}^{k(i)} \lambda_l (c_{il}, d_{il})$$

such that $k(i) \leqq n+1$, $\lambda_l > 0$, the vectors $(c_{il} \cdot d_{il})$ are integral (d_{il} denotes the last, c_{il} the first n coordinates) for $l=1, \ldots, k(i)$, and the decomposition satisfies (ii)′ and (iii) with $(n+1)$ in place of n, and N as given above.

(3) Let $I \subseteqq \{1, \ldots, k(i)\}$ be a maximal subset such that the equalities

$$c_j x = d_j \quad j = 1, \ldots, k, j \neq i$$

and

$$c_{il} x = d_{il} \quad l \in I$$

are linearly independent.

Let the new Integer Programming Problem be defined by the linear inequality system

$$Ax \leqq b$$

$$c_j x = d_j \quad j = 1, \ldots, k, \quad j \neq i,$$

and

$$c_{il} x = d_{il} \quad l \in I.$$

End.

First we have to prove that applying the above algorithm to change the equalities after each reduction gives a valid Integer Linear Programming algorithm. This is the content of the following lemma.

Lemma 5.1. *Let $Ax \leqq b$ define a bounded polyhedron, and let $c_j \cdot x = d_j$ $j=1, \ldots, k$ be linearly independent equalities. Apply the above subroutine to the equality $c_i \cdot x = d_i$. Let $c_{il} x = d_{il}$ $l \in I \subseteqq \{1, \ldots, k(i)\}$ be the resulting equalities.*

(i) *An integral vector z satisfies $Az \leqq b$ and $c_j \cdot z = d_j$ $(j=1, \ldots, k)$ if and only if it satisfies $Az \leqq b$, $c_j \cdot z = d_j$ $(j=1, \ldots, k, j \neq i)$ and $c_{i,l} \cdot z = d_{i,l}$ $l \in I$.*

(ii) *The coefficients in the equalities $c_{il} \cdot x = d_{il}$ have length at most* $2n^3 \log(\alpha+$ $+1)$.

(iii) $I \neq \emptyset$.

By the first statement, replacing $c_i \cdot x = d_i$ using the above subroutine does not affect the set of integral solutions. By the third statement, the new problem can be expressed with at most $n-k$ variables.

Proof. By our assumption, $P=\{x: Ax \leqq b\}$ is bounded. Thus the maximum absolute value of a coordinate will be achieved at a vertex, and so by Cramer's rule, $z \in P$ implies $\|z\|_\infty \leqq n!2^{n\alpha}$ (where α is the same as in Step 1). So $\|z\|_1 \leqq n \cdot n!2^{n\alpha}$, and therefore by Lemma 3.2, z satisfies $c_i \cdot z = d_i$ (that is, in the $(n+1)$-dimensional space $(c_j, d_j) \cdot (z-1) = 0$) if and only if $c_{il} \cdot z = d_{il}$ $(l=1, \dots, k(i))$. Thus, by the maximal choice of I,

$$Az \leqq b, \ c_j z = d_j \quad j = 1, \dots, k, \quad j \neq i$$

and

$$c_{il} z = d_{il} \quad l = 1, \dots, k(i)$$

is equivalent to

$$Az \leqq b, \quad c_j z = d_j \quad j = 1, \dots, k, \quad j \neq i$$

and

$$c_{il} z = d_{il} \quad l \in I.$$

This proves (i).

By condition (ii)′ of the decomposition algorithm we have $\|(c_{il}, d_{il})\|_\infty \leqq \leqq 2^{(n+1)^2+(n+1)} N^{(n+1)}$ (for $l=1, \dots, k(i)$). That is, the lengths of the coefficients of the equality $c_{il}x = d_{il}$ is at most $(n+1)^2+(n+1)+(n+1)\log N \leqq (n+1)(n+2+ + n^2 \log n + n \log \alpha) \leqq 2n^3 \log(\alpha+1)$. This proves (ii).

The equality $c_i \cdot z = d_j$ is a linear combination of the equalities $c_{il} \cdot z = d_{il}$ $(l=1, \dots, k(i))$. By our assumption, the equality $c_i \cdot z = d_i$ is linearly independent of the other equalities $c_j \cdot z = d_j$ $(j=1, \dots, k, j \neq i)$. This proves $I \neq \emptyset$. ▮

Theorem 5.2. *Applying the above subroutine before each reduction of either Lenstra's or Kannan's algorithm makes the algorithms run in polynomial space.*

Proof. First note that one reduction in either Lenstra's or Kannan's algorithm runs in polynomial space: Indeed, one reduction of Lenstra's algorithms runs even in polynomial time (irrespective of the number of variables). Kannan has shown that all numbers occurring during the computation of one of his reductions have length at most $O(n^2 s)$ where s is the input length.

The algorithm is organized as follows:

We are going to keep all reduced problems in the form

$$Ax \leqq b \qquad (1)$$
$$c_i x = d_i \quad i = 1, \dots, k.$$

Before applying the reduction again, we run the above subroutine on the equalities $c_i x = d_i$. As a result we get a problem

$$Ax \leqq b$$
$$c_i' x = d_i' \quad i = 1, \dots, k' \qquad (2)$$

with $k' \geqq k$ linearly independent equalities, such that an integral vector satisfies (1) if and only if it satisfies (2). Next we express (2) in $n-k'$ variables. This can be done using Edmonds' [7] version of Gaussian elimination. In this version of Gaussian elimination all numbers occurring throughout the computation have length at most $n(\alpha'+\log n)$, where α' is the maximum length of the coefficients in (2).

Now we can apply either Lenstra's or Kannan's reduction. The reduction results in problems in the form

$$\begin{aligned} &Ax \leqq b \\ (3) \qquad &c_i'x = d_i' \quad i = 1, \ldots, k' \\ &e_jx = f_j \quad j = 1, \ldots, l \end{aligned}$$

with the same e_j $j=1, \ldots, l$ and all possible integers f_j such that the linear system (3) has a (not necessarily integral) solution.

First we show that the length of all numbers occurring throughout the entire course of this integral programming algorithm is polynomially bounded in the input length. Let α denote the maximum length of the coefficients in $Ax \leqq b$. By Lemma 5.1, the length of the coefficients in (2) is at most $2n^3\alpha$. Thus the length of the coefficients of the $(n-k')$ variable problem (and the length of all numbers occurring throughout the computation of that problem) is at most $O(n^4\alpha)$. So the reduction will always be applied to problems with length at most $O(n^6\alpha)$. Thus the lengths of all numbers occurring throughout the algorithm is polynomially bounded in n and α.

The algorithms have to search through exponentially many reduced problems. Both Lenstra's and Kannan's algorithm reduces the n variable problem to exponentially many problems in less than n variables. We cannot list all of these problems. However, all the reduced problems are of the form described in (3). Thus, knowing e_j $j=1, \ldots, l$, we can generate the equalities in the lexicographic order of $(f_1, \ldots, f_l)$ using a linear programming subroutine. ∎

Applying the changing subroutine also improves the running time.

Theorem 5.3. *Kannan's algorithm with the above subroutine applied before each reduction uses $s \cdot n^{2.5n+o(n)}$ arithmetic operations.*

Proof. One reduction takes $O(n^n s)$ time on problems with input length s (see [14]). As we have seen above the reduction will be applied to problems with input length at most $O(n^6 s)$. Thus the number of arithmetic operations needed to solve an m dimensional problem in n variables satisfies the following recursion

$$T(m) \leqq O(m^{2.5k})T(m-k)+O(m^m n^6 s).$$

Thus, by induction $T(m)=O(m^{2.5m}n^6 s)$. ∎

6. Concluding Remarks

Grötschel, Lovász and Schrijver [12 and 13] showed how the ellipsoid method can be used for proving polynomial solvability of various combinatorial problems. For the time being, this method is widely considered to be a proof technique rather than an efficient algorithm useful in practice. Once one knows the existence of a polynomial algorithm for a certain problem one may try to construct an efficient problem-specific polynomial time algorithm.

Parallel to this approach, here we have developed a method to show that certain combinatorial optimization problems can be solved in strongly polynomial time. We consider our method to be a proof technique rather than an efficient algorithm useful in practice. Once one knows the existence of a strongly polynomial algorithm for a certain problem one may try to construct an efficient, problem-specific strongly polynomial algorithm. Three problem-specific strongly polynomial algorithms for problems, which can be solved in strongly polynomial time with the technique presented in this paper, are Cunningham's method [4] for testing membership in a matroid polyhedron, S. Fujishige's [10] combinatorial strongly polynomial algorithm for the submodular flow problem, and the minimum cost circulation algorithm in [11].

Acknowledgments. We wish to thank László Lovász for many stimulating discussions on the results presented in this paper. We are indebted to Ravi Kannan for drawing our attention to possible application of the preprocessing technique to Integer Linear Programming.

References

[1] R. E. Bixby, O. M.-C. Marcotte and L. E. Trotter, Jr., Packing and covering with integral flows in integral supply-demand networks, *Report No.* 84327—OR, Institut für Ökonometrie und Operations Research, University Bonn, Bonn, West Germany.

[2] J. W. S. Cassels, *An Introduction to the Theory of Numbers*, Berlin, Heidelberg, New York, Springer, 1971.

[3] W. H. Cunningham, Testing membership in matroid polyhedra, *Journal of Combinatorial Theory B*, **36** (1984), 161—188.

[4] W. H. Cunningham and A. Frank, A primal dual algorithm for submodular flows, *Mathematics of Operations Research.*, **10** (1985).

[5] E. A. Dinits, Algorithm for solution of a problem of maximum flow in a network with power estimation, *Soviet Math. Dokl.*, **11** (1970). 1277—1280.

[6] J. Edmonds, Minimum partition of a matroid into independent subsets, *Research of the Nat. Bureau of Standards* **69** B (1965), 67—72.

[7] J. Edmonds, System of distinct representatives and linear algebra, *J. Res. Nat. Bur. Standards*, **71** B (1967), 241—245.

[8] J. Edmonds and R. Giles, A min-max relation for submodular functions on graphs, *Annals of Discrete Mathematics*, **1** (1977), 185—204.

[9] J. Edmonds and R. M. Karp, Theoretical improvements in the algorithmic efficiency for network flow problems, *J. ACM*, **19** (1972), 248—264.

[10] S. Fujishige, A capacity rounding algorithm for the minimum-cost circulation problem: a dual framework of the Tardos algorithm, *Mathematical Programming*, **35** (1986), 298—309.

[11] Z. Galil and É. Tardos, An $O(n^2(m+n\log n)\cdot\log n)$ minimum cost flow algorithm, *in: Proc., 27th Annual Symposium on Foundations of Computer Science* (1986), 1—9.

[12] M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, **1** (1981), 169—197.

[13] M. Grötschel, L. Lovász and A. Schrijver, *The ellipsoid method and combinatorial optimization*, Springer Verlag, *to appear.*

[14] R. Kannan, Improved algorithms for integer programming and related lattice problems, *in: Proc.*, **15***th Annual ACM Symposium on the Theory of Computing* (1983), 193—206. *Final version:* Minkowski's Convex Body Theorem and Integer Programming, Carnegie-Mellon University, *Report* No. 86—105.

[15] A. K. Lenstra, H. W. Lenstra, Jr. and L. Lovász, Factoring polynomials with rational coefficients, *Math. Ann.*, **261** (1982), 515—534.

[16] H. W. Lenstra, Jr., Integer programming with a fixed number of variables, *Math. of Operations Research*, **8** (1983), 538—548.
[17] É. Tardos, A strongly polynomial minimum cost circulation algorithm, *Combinatorica,* **5** (1985), 247—255.
[18] É. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, *Operations Research*, (1986), No. 2
[19] U. Zimmermann, Minimization of submodular flows, *Discrete Applied Math.*, **4** (1982), 303—323.

András Frank

Institute of Mathematics
Eötvös University, Budapest
1445, P. O. B. 323
Hungary

Éva Tardos

Institute of Mathematics
Eötvös University, Budapest
1445, P. O. B. 323
Hungary