SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NY 14853-7501

TECHNICAL REPORT NO. 866

October 1989

USING SEPARATION ALGORITHMS
IN FIXED DIMENSION

By

Carolyn Haibt Norton¹
Serge A. Plotkin²
Eva Tardos³

¹Department of Mathematics, MIT. Research supported by an NSF Graduate Fellowship.

²Computer Science Department, Stanford University. Research partially supported by ONR Contract N00014-88-K-0166.

³School of Operations Research and Industrial Engineering, Cornell University. This work was done in part while the author was visiting the Department of Mathematics at M.I.T. Research partially supported by Air Force contract AFOSR-89-0271 and DARPA contract N00014-89-J-1988.

Abstract

Consider a convex set in d dimensions. Assume that we are given a separation subroutine which, given a point, tells us whether this point is in the set. Moreover, if the point is not in the set, the subroutine separates the point from the set by a hyperplane. We show that if d is fixed and the separation subroutine is linear¹ in the input vector, this implies that one can optimize a linear objective function over the convex set in time polynomial in the number of arithmetic operations used by the separation subroutine. We apply this result to extend the class of linear programs solvable in strongly polynomial time. We show that a problem can be solved in strongly polynomial time if, by deleting a constant number of rows and columns, it can be converted to a problem which is already known to be solvable in strongly polynomial time. For example, this yields a strongly polynomial algorithm for the concurrent multi-commodity flow problem.

1 Introduction

One of the most important results of the last ten years is Khachiyan's [15] proof that the linear programming problem can be solved in polynomial time. Since then, several other polynomial time algorithms have been developed. The most important of these is Karmarkar's [13] algorithm, which is also efficient in practice. None of these algorithms, however, is *strongly polynomial*; the number of arithmetic operations depends not only on the number of variables and inequalities, but also on the size of the numbers involved. It is an important open question whether a strongly polynomial linear programming algorithm exists.

If the dimension is fixed, then it is easy to solve the linear programming problem in strongly polynomial time. A polyhedron defined by m inequalities in d variables has at most m^d vertices, so the simplex algorithm runs in $m^{O(d)}$ time. This is strongly polynomial when d is fixed. Megiddo [20] greatly improved this by giving a linear programming algorithm whose running time is $O(2^{2^d} m)$. The dependence on the dimension has been improved to singly-exponential independently by Clarkson [3] and Dyer [7]. Further improvements are possible using randomization [4,8].

Strongly polynomial algorithms have also been developed for other special classes of linear programs. Megiddo [19] gave a strongly polynomial algorithm to solve a system of linear inequalities with at most two variables per inequality. In fact, Megiddo [personal communication] has extended this result to optimization when the objective function has at most two non-zero coefficients. Tardos [25] gave a strongly polynomial algorithm for the minimum-cost flow problem. The constraint matrix of the corresponding linear program is the arc-node adjacency matrix of the network, which is $(0, +1, -1)$ -matrix in this case. Tardos [26] extended this result and described a linear programming algorithm that runs in time polynomial in the dimension of the constraint matrix and the size of the numbers in this matrix. In particular, her results imply a strongly polynomial algorithm for the minimum-cost multi-commodity flow problem. Cohen and Megiddo [5] gave a strongly polynomial algorithm for finding a zero-cost circulation

¹By "linear" we mean that each comparison made by the subroutine is between two expressions that can be written as linear functions of the input vector.

in a network with costs being constant-dimension vectors, and used this algorithm to detect cycles in dynamic graphs.

For most combinatorial optimization problems, the corresponding linear programs have an exponential number of inequalities, so linear programming cannot be used directly. In some cases, it is possible to give a polynomial time *separating subroutine*. That is, a subroutine which decides if a given vector is feasible, and if not, gives a hyperplane which separates it from the feasible region. Grötschel, Lovász and Schrijver [11], Karp and Papadimitriou [14] and Padberg and Rao [21] independently observed that Khachiyan's ellipsoid method can be extended to work for problems defined by separating subroutines, if the vertices of the polyhedron are rational vectors with known binary length. (See also the book by Grötschel, Lovász and Schrijver [12].) Frank and Tardos [9] strengthened this result by proving that the algorithm can be made strongly polynomial if the bound on the length of the binary description of the vertices is polynomial in the number of variables. This is the case, for example, in linear programs with 0-1 vertices.

In this paper, we consider polyhedra given by separating subroutines, without assuming that the complexity of the vertices is polynomially bounded, or even known. On the other hand, we make fairly strong assumptions about the separating subroutine. This is in contrast with the above mentioned versions of the ellipsoid method which, in fact, use separating oracles rather than algorithms. In other words, they do not use any information about the internal structure of the separating algorithm. In this paper we consider separating algorithms that are "linear" in the input vector, where by linear we mean that all internal computations can be represented as functions which are linear in the input vector. For example, if a polyhedron is given by a system of linear inequalities, then the separating subroutine that simply substitutes the vector into the inequalities satisfies this assumption. We prove that in *fixed* dimension one can optimize in time polynomial in the running time of the separating subroutine. Moreover, we show that if the separating subroutine is a parallel algorithm that runs in polylogarithmic time using a polynomial number of processors, then there is a parallel optimization algorithm which also runs in polylogarithmic time and uses a polynomial number of processors. Note, that a statement that corresponds to the first one in variable dimension would imply that the linear programming problem can be solved in strongly polynomial time; a statement which corresponds to the second one in variable dimension would imply that linear programming is in NC.

We show how to use the results in fixed dimension to extend the class of linear programs that can be solved in strongly polynomial time. Consider any of the classes of linear programs mentioned above. We prove that any linear program that can be obtained from these by adding a constant number of additional variables can be solved in strongly polynomial time as well. This linear program is reduced to a fixed dimensional problem by projecting it onto the additional variables and a new variable corresponding to the value of the objective function. This projection can increase the number of faces exponentially, so the inequalities defining the projection cannot be given in polynomial time. Instead we shall use the known strongly polynomial algorithm to separate from the projection. Using linear programming duality, this result can be further extended to linear programs with a constant number of additional variables and inequalities.

The heart of our algorithm is a parametrization technique that was used in Megiddo [17,18,19]. The algorithm executes the separation subroutine on a point parameterized to represent a feasible point. Additions and multiplications are performed symbolically, and comparisons are answered by a recursive call to a smaller-dimensional subroutine. If several comparisons are done simultaneously, the geometric search scheme of Megiddo [20], as improved by Clarkson [3] and Dyer [7], is used to reduce the number of recursive calls, thereby improving the running time. Similar techniques were recently used by Cohen and Megiddo [5] to design an algorithm that detects cycles in dynamic graphs. In the forthcoming journal version of the paper they generalize their algorithm and achieve some of the results presented in this paper.

We mention two special cases for which strongly polynomial algorithms have not been known before. Shahrokhi and Matula [23] have considered an optimization version of the multi-commodity flow problem called the *concurrent multi-commodity flow* problem. A feasible flow for this problem is one that satisfies the same percentage of each commodity; the problem is to maximize this percentage. To turn this into a linear program, we introduce a variable representing the satisfied percentage of the demands. Because the coefficients of this variable in the constraint matrix are the demands, this problem does not belong to any one of the classes of linear programs which were known to be solvable in strongly polynomial time. Our results yield a strongly polynomial algorithm for this problem.

The linear programming dual of the generalized transshipment problem (transshipment in networks with gains and losses) is a linear program with at most two variables per inequality (the nonnegativity of the reduced costs). Using Megiddo's result [19] one can check the feasibility of this dual problem in strongly polynomial time. Adler and Cosares [1] recently observed that if all the demands have the same sign, then Megiddo's algorithm can be used to solve this problem in strongly polynomial time. Results in this paper extend the class of generalized transshipment problems solvable in strongly polynomial time to include problems where all but a constant number of the demands have the same sign. It is an important open problem whether there exists a strongly polynomial algorithm for the generalized transshipment problem, and more generally, for the problem of optimizing an arbitrary linear objective function over a set of inequalities with at most two variables per inequality.

Section 2 describes how to optimize a linear objective function over a convex set which is given by a separation algorithm. Section 3 uses the results of Section 2 to extend the class of linear programs solvable in strongly polynomial time. Section 4 describes how to apply our results to linear programs with 2 variables per inequality.

2 From Separation to Optimization

In this section we consider the problem of optimizing a linear objective function over a fixed-dimensional convex set which is given to us by a *linear* separation algorithm. We start by presenting several definitions, then we describe an algorithm which finds a point in the set and show how to convert this algorithm to an optimization algorithm.

A *separation algorithm* for a closed convex body $\mathcal{P} \subseteq \mathbf{R}^d$ is an algorithm that takes a point

$z \in \mathbf{R}^d$ and either concludes that $z \in \mathcal{P}$ or finds a direction given by a row vector² a , such that $ax < az$ for every point $x \in \mathcal{P}$. The separation algorithm is *linear* if each comparison it makes is between a constant and an expression that can be written as a linear function of the input vector z .

It is possible to add the inequalities output by the separation algorithm to deduce other inequalities. A *derivation certificate* for the inequality $ax \leq \beta$ with respect to the system $A_1x \leq b_1, A_2x < b_2$ is a pair of row vectors $y_1, y_2 \geq 0$ such that $y_1A_1 + y_2A_2 = a$ and $y_1b_1 + y_2b_2 \leq \beta$. A *derivation certificate* for the strict inequality $ax < \beta$ is defined similarly, except that if $y_2 = 0$, then we require that $y_1b_1 < \beta$. Observe that if we have a derivation certificate for $ax < \beta$ with respect to $A_1x \leq b_1, A_2x < b_2$ and derivation certificates for the inequalities $A_1x \leq b_1$ and $A_2x < b_2$ with respect to some $A'_1x \leq b'_1, A'_2x < b'_2$, then substituting one into the other provides a valid derivation certificate for $ax < \beta$ with respect to $A'_1x \leq b'_1, A'_2x < b'_2$. If the inequality $0 < 0$ can be derived from a system of inequalities, then the system is clearly contradictory. Thus, a *certificate of emptiness* for a system is a derivation certificate for $0 < 0$ (with respect to the system). A *certificate of optimality* for a feasible solution to a linear program is a feasible dual solution with the same objective value.

Lemma 2.1 A certificate of emptiness for the system of inequalities $A_1x \leq b_1, A_2x < b_2$ and $ax = \beta$ can be converted to either a certificate of emptiness for $A_1x \leq b_1, A_2x < b_2$, or a derivation certificate for one of the inequalities $ax < \beta$ or $ax > \beta$ with respect to $A_1x \leq b_1, A_2x < b_2$.

Proof: Let (y_1, y_2, y_3) be a certificate of emptiness. If $y_3 = 0$, then (y_1, y_2) is a certificate of emptiness for $A_1x \leq b_1, A_2x < b_2$. Otherwise the vector $(y_1/|y_3|, y_2/|y_3|)$ is a derivation certificate for $ax < \beta$ if $y_3 > 0$ and for $-ax < -\beta$ if $y_3 < 0$. ■

Because we are working in a d -dimensional space, any derivation certificate with respect to m inequalities can be reduced to another derivation certificate with respect to only d inequalities in $O(d^3m)$ time. Hence, we can assume that each derivation certificate is with respect to a system of at most d inequalities. (Certificates of emptiness require $d + 1$ inequalities.)

Lemma 2.2 Given a polyhedron in $\mathbf{R}^d$ defined by the m inequalities $A_1x \leq b_1$ and $A_2x < b_2$, it is possible to find either a point satisfying these inequalities or a certificate of emptiness for the system in $O(m)$ time.

Proof: Consider the linear program $\max\{\epsilon : A_1z \leq b_1, A_2z + \epsilon e \leq b_2\}$. This is a linear program in $d + 1$ variables, and therefore can be solved in $O(m)$ time, using Megiddo's [20] fixed-dimensional linear programming algorithms. (See also the improved versions due to Clarkson [3] and Dyer [7] and the book by Schrijver [22].) If the system is infeasible, then linear programming returns a certificate of emptiness. If the optimal value, ϵ^* , is greater than 0, then this gives a feasible point. If it is less than or equal to 0, then the original inequality system has no solution. The dual solution (y_1, y_2) satisfies $y_1A_1 + y_2A_2 = 0$, $y_2e = 1$, and $y_1b_1 + y_2b_2 = \epsilon^* \leq 0$, and hence it is a certificate of emptiness for the system given in the lemma³. ■

²We will use both row and column vectors. Points will be column vectors; objective functions and solutions to dual linear programs will be row vectors.

³Here e denotes the column vector all of whose coordinates are 1.

```

procedure COMPARE( $a, \beta, B, b, k$ );
  We assume that  $a$  and the rows of  $(B|I)$  are linearly independent.

  Use  $(B|I)x = b$  to compute  $ax$  in terms of the first  $k$  coordinates;
  Assume, for notational convenience, that the last is non-zero;
  Express the equation system  $(B|I)x = b, ax = \beta$  in the form  $(B'|I)x = b'$ ;
  Call FIND-POINT( $B', b', k - 1$ );

  If FIND-POINT returns a point in  $\mathcal{P}$ , then RETURN this point.

  Otherwise, FIND-POINT returns a set of inequalities  $A_1x \leq b_1, A_2x < b_2$ , which are satisfied by all of  $\mathcal{P}$ ,
  and a certificate of emptiness for the system  $A_1x \leq b_1, A_2x < b_2$ , and  $(B'|I)x = b'$ .

  Find either a certificate of emptiness for  $A_1x \leq b_1, A_2x < b_2, (B|I)x = b$ ,
  or a derivation certificate for one of the two inequalities  $ax < \beta$  or  $ax > \beta$ , as in the proof of Lemma 2.1.
  RETURN this system of inequalities together with the certificate of emptiness or derivation.

end.

```

Figure 1: The algorithm to compare ax and β .

To facilitate a recursive algorithm, we consider the following modified version of the problem: Given a linear separation algorithm for $\mathcal{P}$ and a system $(B|I)x = b$ of k linearly independent equations,⁴ find a point in the set $\mathcal{P} \cap \{x : (B|I)x = b\}$ or find a “proof” that no such point exists. This “proof” consists of a set of strict and non-strict inequalities satisfied by *all* points of $\mathcal{P}$, along with a certificate of emptiness for this system combined with the equations $(B|I)x = b$.

We recursively define the algorithm FIND-POINT to solve this problem. The idea of FIND-POINT is to simulate the separation algorithm symbolically. The result of each comparison is either computed directly from the equations $(B|I)x = b$ or by a call to a procedure named COMPARE. COMPARE looks for a point in $\mathcal{P}$ which satisfies both $(B|I)x = b$ and $ax = \beta$. If such point is found, we have a point of $\mathcal{P}$, so we are done. Otherwise, COMPARE determines the outcome of the comparison, and returns a proof of its decision. Using this, FIND-POINT continues to simulate the separation algorithm.

COMPARE is built around a recursive call to FIND-POINT (see Figure 1). If FIND-POINT fails to find a point in the set $\mathcal{P} \cap \{x : (B|I)x = b, ax = \beta\}$, COMPARE uses the “proof of emptiness” returned by FIND-POINT and Lemma 2.1 to decide on which side of the hyperplane $ax = \beta$ the space $\mathcal{P} \cap \{x : (B|I)x = b, ax = \beta\}$ must lie. COMPARE also returns a set of inequalities, $A_1x \leq b_1, A_2x < b_2$, which hold for all of $\mathcal{P}$. Finally, it returns either a certificate of emptiness for $A_1x \leq b_1, A_2x < b_2, (B|I)x = b$, or a derivation certificate for $ax < \beta$ (or $ax > \beta$) with respect to these inequalities and the system $(B|I)x = b$.

FIND-POINT (see Figure 2) continues to simulate the separation algorithm as long as the comparisons can be either deduced directly from $(B|I)x = b$ or decided by COMPARE. If one of the calls to COMPARE returns either a point in $\mathcal{P}$ or a proof of emptiness of $\mathcal{P} \cap \{x : (B|I)x = b\}$, then FIND-POINT is done and returns this result. Somewhat more work must be done if FIND-POINT reaches the end of the original separation algorithm. In this case, FIND-POINT considers the system $Ax < \hat{b}$ of inequalities decided by COMPARE, together with their derivation

⁴Here I denotes the $(d - k) \times (d - k)$ identity matrix.

certificates. Let $\hat{\mathcal{P}}$ denote the region $\{x : Ax < \hat{b}, (B|I)x = b\}$. The derivation certificates prove that $\mathcal{P} \cap \{x : (B|I)x = b\} \subseteq \hat{\mathcal{P}}$.

FIND-POINT uses Lemma 2.2 to test whether $\hat{\mathcal{P}}$ is empty. If so, the intersection of $\mathcal{P}$ and $\{x : (B|I)x = b\}$ must also be empty, so FIND-POINT translates the certificate for $\hat{\mathcal{P}}$ into a certificate of emptiness for $A_1x \leq b_1, A_2x < b_2, (B|I)x = b$ and returns it to the caller.

If FIND-POINT finds a point $x_0 \in \hat{\mathcal{P}}$ and the simulated separation algorithm terminates with an “accept”, then we return this point to the caller.⁵ If, instead, the simulated separation algorithm terminates with “reject”, then $\mathcal{P} \cap \{x : (B|I)x = b\}$ is empty, and we need to produce a “proof” of this. Let a denote the separating direction returned by the algorithm. In this case, FIND-POINT uses fixed-dimensional linear programming to minimize ax over the closure of $\hat{\mathcal{P}}$. Let γ be this minimum, and let (y_1, y_2) denote an optimal dual solution. Then $y_1A + y_2(B|I) = -a$ and $y_1\hat{b} + y_2b = -\gamma$. Since a is separating we have that $ax \leq \gamma$ for every $x \in \mathcal{P}$ (if $y_1 = 0$, then $ax < \gamma$, because in this case $ax = \gamma$ for any point in $\{x : (B|I)x = b\}$). On the other hand, the vector $(y_1, y_2, 1)$ is a certificate of emptiness of $Ax < \hat{b}, (B|I)x = b$ and $ax \leq \gamma$ (or $ax < \gamma$). We add this new inequality to the system $A_1x \leq b_1, A_2x < b_2$ and translate $(y_1, y_2, 1)$ into a certificate of emptiness for this new system.

Theorem 2.3 Given a separation algorithm for a closed convex set $\mathcal{P} \subseteq R^d$, that runs in time t and makes at most q comparisons, we can either find a point in $\mathcal{P}$ or find a certificate of its emptiness in $O(q^d t)$ time.

Proof: Let $T(m)$ and $T'(m)$ be the running times of FIND-POINT(B, b, m) and COMPARE(a, β, B, b, m). Lemma 2.2 implies that $T'(m) = O(q) + T(m - 1)$. The claim follows from the fact that $T(m) = qT'(m) + t$. ■

Next we show how to optimize a linear objective function over a polyhedron $\mathcal{P}$ which is given by a linear separation algorithm. The basic idea is to modify FIXED-POINT so that it finds a point which is not only in $\mathcal{P}$, but is also on the optimal face. As before, we define a more general problem to facilitate recursion. Given a set of hyperplanes $(B|I)x = b$ and an objective function c , OPTIMIZE returns either that cx is unbounded over $\mathcal{P}$, or a set of inequalities $A_1x \leq b_1, A_2x < b_2$ which are satisfied by *all optimal points* of $\mathcal{P}$. In addition, if the system is feasible, then OPTIMIZE returns a point $z \in \mathcal{P}$ which maximizes cx over $\{x : A_1x \leq b_1, A_2x < b_2, (B|I)x = b\}$, together with a certificate of optimality. Otherwise, OPTIMIZE returns a certificate of emptiness for this system. In either case, we can assume that the certificate uses at most $d + 1$ inequalities.

The only difference between OPTIMIZE and FIND-POINT is the routine used to decide comparisons. COMPARE-OPT is very similar to COMPARE, but it makes decisions about the optimal face of $\mathcal{P}$, rather than the whole set. To do this, it makes recursive calls to OPTIMIZE, rather than to FIND-POINT. If no point is found by the recursive call, then the comparison between ax and β is decided as before. If an optimal point z is found, however, we no longer can simply return this point. We need to check if it is optimal over $\mathcal{P} \cap \{x : (B|I)x = b\}$, as well as the

⁵Note that in this case we have that $\hat{\mathcal{P}} = \mathcal{P} \cap \{x : (B|I)x = b\}$. The right hand side is clearly closed, but the left hand side is open unless COMPARE was not called during the run. Hence we must have that $\{x : (B|I)x = b\} \subseteq \mathcal{P}$ in this case.

procedure FIND-POINT(B, b, k);

Solve the system $(B|I) = b$ for the last $d - k$ coordinates, and substitute for all occurrences of these variables in the separation algorithm.

Run the Separation algorithm symbolically using the first k coordinates:

If ax and $(B|I)$ are linearly dependent, then the comparison is trivial. Make the comparison, and continue.
 If ax and $(B|I)$ are linearly independent, then call COMPARE(a, β, B, b, k).
 If COMPARE returns a point, then RETURN this point. This point is in $\mathcal{P} \cap \{x : (B|I)x = b\}$.
 If COMPARE returns a certificate of emptiness of $A_1 x \leq b_1, A_2 x < b_2, (B|I)x = b$,
 where $\mathcal{P} \subseteq \{x : A_1 x \leq b_1, A_2 x < b_2, (B|I)x = b\}$,
 then RETURN this certificate; $\mathcal{P} \cap \{x : (B|I)x = b\}$ is empty.
 If COMPARE returns an inequality, then continue to simulate the separation algorithm symbolically.

When the separation algorithm terminates, consider the set of inequalities $Ax < \hat{b}$ returned by the calls to COMPARE and their derivation certificates with respect to $A_1 x \leq b_1, A_2 x < b_2$,
 where $\mathcal{P} \subseteq \{x : A_1 x \leq b_1, A_2 x < b_2\}$;

Use fixed-dimensional linear programming (see Lemma 2.2) to check if $\hat{\mathcal{P}} = \{x : Ax < \hat{b}, (B|I)x = b\} = \emptyset$.

Case 1: $\hat{\mathcal{P}} = \emptyset$.

Translate its certificate of emptiness into a certificate for $(B|I)x = b, A_1 x \leq b_1, A_2 x < b_2$;
 RETURN this system of inequalities with the certificate.

Case 2: $\hat{\mathcal{P}} \neq \emptyset$, and the separator returned “accept”.

Find a point x_0 in $\hat{\mathcal{P}}$;
 RETURN x_0 .

Case 3: $\hat{\mathcal{P}} \neq \emptyset$, and the separator returned “reject”.

In this case the separator also outputs a separating direction a ,
 such that for any $x \in \mathcal{P}$ and $x' \in \hat{\mathcal{P}}$, we have $ax < ax'$;
 Use fixed-dimensional LP to minimize ax over the closure of $\hat{\mathcal{P}}$, i.e. $\{x : (B|I)x = b, Ax \leq \hat{b}\}$;
 Let γ be this minimum and let (y_1, y_2) be an optimal dual solution,
 i.e., $y_1 A + y_2 (B|I) = -a$; $y_1 \hat{b} + y_2 b = -\gamma$;

Case 3.1: $y_2 \neq 0$.

This implies that $ax \leq \gamma$ for $x \in \mathcal{P}$;
 Add row a to A_1 and γ to b_1 , and denote the result by A'_1, b'_1 ;
 $(y_1, y_2, 1)$ is a certificate of emptiness of $Ax < \hat{b}, (B|I)x = b, ax \leq \gamma$;
 Translate it into a certificate of emptiness of $(B|I)x = b, A'_1 x \leq b'_1, A_2 x < b_2$;
 RETURN this certificate together with (A'_1, A_2, b'_1, b_2) ;

Case 3.2: $y_2 = 0$.

This implies that $ax < \gamma$ for every $x \in \mathcal{P}$;
 Proceed as in Case 3.1, only use $ax < \gamma$;

end.

Figure 2: The algorithm to find a point in $\mathcal{P} \cap \{x : (B|I)x = b\}$.

smaller space $\mathcal{P} \cap \{x : (B|I)x = b, ax = \beta\}$. Look at the coefficient y which corresponds to the equation $ax = \beta$ in the certificate of optimality for z . If $y = 0$, then the extra equation has not changed the problem, and we are done. If not, then assume that $y < 0$ (the case $y > 0$ is treated similarly). In this case, we know that all optimal points of $\mathcal{P}$ satisfy $ax \leq \beta$. We have to decide whether z is optimal or whether the strict inequality $ax < \beta$ can be deduced. We invoke the OPTIMIZE procedure to maximize cx over $\mathcal{P} \cap \{x : (B|I)x = b, ax = \beta - \epsilon\}$, where we regard ϵ as infinitesimally small, computing symbolically with it. If this problem has the same optimal value or is infeasible, then we conclude that z is optimal over $\mathcal{P} \cap \{x : (B|I)x = b\}$; the required proof of its optimality can be computed directly from certificates that we have. Otherwise, we conclude that all optimal points satisfy $ax < \beta$.

Theorem 2.4 Given an objective function c and a separation algorithm for a nonempty closed convex set $\mathcal{P} \subseteq \mathbb{R}^d$, which runs in time t and makes q comparisons, then there is an algorithm which in $O(q^d t)$ time either finds a point $x \in \mathcal{P}$ maximizing cx , or concludes that $\max\{cx : x \in \mathcal{P}\}$ is unbounded.

Proof: Similar to the proof of Theorem 2.3. ■

By using the geometric search scheme of Megiddo [20] (or the improved version due to Clarkson [3] and Dyer [7]), the running time of the algorithm can be improved if, instead of making q sequential comparisons, the separation algorithm makes q_1 sets of comparisons, with at most q_2 comparisons in each set, *i.e.* if the separation algorithm can be represented as a decision tree of depth q_1 with at most q_2 comparisons at every node.

Theorem 2.5 Given an objective function $c \in \mathbb{R}^d$ and a separation algorithm for a nonempty closed convex set $\mathcal{P} \subseteq \mathbb{R}^d$ which runs in time t and makes q_1 sets of at most q_2 comparisons, there is an algorithm which in $O((q_1 \log q_2)^d t)$ time either finds a point $x \in \mathcal{P}$ maximizing cx , or concludes that $\max\{cx : x \in \mathcal{P}\}$ is unbounded.

Proof: Use the algorithm described above with one modification. Each time the COMPARE procedure is called with q_2 hyperplanes, it does not simply call the $(m-1)$ dimensional procedure q_2 times. Instead, it uses the geometric search scheme of Megiddo [20] (or the improved version due to Clarkson [3] and Dyer [7]) to decide all q_2 comparisons by calling COMPARE only $O(\log q_2)$ times. ■

Remark: The time bound in this theorem can be improved in certain cases. One does not have to use the same separation algorithm parametrically as for the real separation. More precisely, in the above theorem it suffices to assume that there exists a separation algorithm that runs in t time and a (potentially different) algorithm that makes q_1 sets of at most q_2 comparisons.

Remark: Megiddo in [17,18] gives strongly polynomial and $\mathcal{NC}$ algorithms for several combinatorial ratio minimization problems. Notice that the existence of such algorithms follows from Theorem 2.4 and 2.5 (with $d = 1$) applied to the set of feasible ratios.

3 Applications to Linear Programming

In this section, we describe how to use the results of the previous section to extend the class of linear programs solvable in strongly polynomial time. In particular, we show how to construct a strongly polynomial algorithm for any linear program that can be obtained by adding a constant number of variables and inequalities to a linear program already known to be solvable in strongly polynomial time.

All known strongly polynomial algorithms for special classes of linear programs [1,5,19,26] are linear in both the objective function and the right hand side. In other words, all intermediate values computed by these algorithms can be written as functions which are linear in the objective function and the right hand side. We shall consider only strongly polynomial algorithms that have this property. Furthermore, we assume that these algorithms provide optimal dual solutions (certificates of optimality) and certificates of emptiness, when appropriate.

Theorem 3.1 Suppose that there exists a strongly polynomial algorithm to solve the linear program $\max\{ax : Ax \leq b\}$ for arbitrary b , that runs in time t , uses q comparisons, and is linear in a and b . Then for any fixed d , there is a strongly polynomial algorithm which runs in $O(q^{d+1}t)$ time and solves the linear program $\max\{ax + cz : Ax + Cz \leq b\}$ for any matrix C with d columns.

Proof: Consider the problem $\max\{cz + \alpha : Ax + Cz \leq b, ax - \alpha \geq 0\}$. This problem and the one we have to solve clearly have the same set of optimal solutions. Let $\mathcal{P} = \{(z^T, \alpha)^T : \exists x, Ax + Cz \leq b, ax - \alpha \geq 0\}$ be the projection of the feasible region onto the variables $(z^T, \alpha)^T$. We will show that there is a separating subroutine for $\mathcal{P}$ that runs in time polynomial in the number of rows and columns of A . The claim will then follow directly from Theorem 2.4.

To test whether a point $(z_0^T, \alpha_0)^T$ is in $\mathcal{P}$, use the assumed algorithm to solve $\max\{ax : Ax \leq b - Cz_0\}$. If this problem is feasible and the maximum is at least α_0 (or is unbounded) then $(z_0^T, \alpha_0)^T$ is in $\mathcal{P}$. If there is a finite maximum less than α_0 , then $(z_0^T, \alpha_0)^T$ is not in $\mathcal{P}$. To find a separating hyperplane, let y be an optimal dual solution. The row vector y satisfies $yA = a$ and $y(b - Cz_0) < \alpha_0$ and the row vector $\tilde{y} = (yC, 1)$ defines a separating hyperplane. Indeed,

$$\begin{aligned} \tilde{y}(z_0^T, \alpha_0)^T &= yCz_0 + \alpha_0 > yb \quad \text{and} \\ \tilde{y}(z^T, \alpha)^T &= yCz + \alpha \leq yb - yAx + \alpha \leq yb - ax + \alpha \leq yb \quad \text{for any } (z^T, \alpha)^T \in \mathcal{P} \end{aligned}$$

Finally, if the problem is infeasible, then no solution to $Ax + Cz \leq b$ has $z = z_0$ and hence $(z_0^T, \alpha_0)^T$ is not in $\mathcal{P}$. If y denotes the certificate of emptiness, then the row vector $\tilde{y} = (yC, 0)$ defines a separating hyperplane for $\mathcal{P}$. ■

Corollary 3.2 Suppose that there exists a strongly polynomial algorithm to solve the linear program $\max\{ax : Ax \leq b\}$ for arbitrary b , which runs in time t , uses q comparisons, and is linear in a and b . Then for any fixed d , there exists a strongly polynomial algorithm that solves the linear program $\max\{ax + a'z : Ax + Bz \leq b, A'x + Cz \leq b'\}$ for any $d \times d$ matrix C , any matrix B with at most d columns and any matrix A' with at most d rows.

Proof: By Theorem 3.1, there is a strongly polynomial algorithm that solves $\max\{ax + a'z : Ax + Bz \leq b\}$ for arbitrary a, a' and b . The claim follows by applying Theorem 3.1 to the dual of this problem. ■

A nice corollary of Theorem 3.1 is that the concurrent multi-commodity flow problem, introduced by Shahrokhi and Matula [23], can be solved in strongly polynomial time. In this version of the multi-commodity flow problem, we are given a network $G = (V, E)$ with capacities c on the arcs, and a network of required demands $R = (V, F)$ with demands r on the arcs; there are no costs. A feasible solution in this problem is a collection of non-negative flows, f_{ij} for $(i, j) \in F$, all satisfying the same percentage of the corresponding demands. The objective of the *concurrent multi-commodity flow problem* is to maximize this percentage. Formally, the problem is defined as follows:

$$\begin{array}{ll} \text{maximize } z & \\ \text{such that: } & \left. \begin{array}{ll} \sum_{e=(i,*)} f_{ij}(e) - \sum_{e=(*,i)} f_{ij}(e) = zr(i, j) & \forall i, j \in F; \\ \sum_{e=(k,*)} f_{ij}(e) - \sum_{e=(*,k)} f_{ij}(e) = 0 & \forall i, j \in V, k \in V/\{i, j\}; \\ \sum_{(i,j)} f_{ij}(e) \leq c(e) & \forall e \in E; \\ f_{ij}(e) \geq 0 & \forall e \in E, (i, j) \in F. \end{array} \right\} \end{array}$$

Corollary 3.3 The concurrent multi-commodity flow problem can be solved in strongly polynomial time.

Proof: In this linear program, only the variable z has coefficients which are not 0, +1 or -1. Tardos [26] gave a strongly polynomial algorithm to solve linear programs with $(0, +1, -1)$ -constraint matrices. The claim follows from Theorem 3.1, with $d = 1$. ■

4 Linear Programming with Two Variables Per Inequality

In this section we consider linear programs $\max\{cx : Ax \leq b\}$ where no more than two variables have non-zero coefficients in each constraint. By modifying a polynomial algorithm due to Aspvall and Shiloach [2], Megiddo [19] has given a strongly polynomial algorithm to find a feasible solution to such a system. His algorithm also computes the projection of the feasible region on each of the variables and hence can maximize or minimize a single variable. In fact, Megiddo [personal communication] has extended this algorithm to optimize an objective function with two non-zero entries. Megiddo's result, together with Theorem 2.3, implies a strongly polynomial algorithm to optimize a linear objective function with any constant number of entries over a region defined by set of constraints. In this section we show how to use the special structure of this problem to improve the running time.

We start with a sketch of Megiddo's algorithm (see Figure 3). Consider the graph whose nodes correspond to the variables of the linear program; let two nodes be connected by an edge if there exists an inequality containing them both. If a variable appears with opposite signs in two inequalities, then these two inequalities can be combined to eliminate the common variable,

Stage 1. For every variable x_i compute the tightest bounds u_i and l_i which can be obtained by eliminations along a cycle. If $\exists i$ such that $l_i > u_i$ then return: “The problem is infeasible”.

Stage 2. For every variable x_i compute the tightest bounds u'_i and l'_i which can be obtained from l_j and u_j for $j = 1, \dots, n$ by eliminations along paths in G . If $\exists i$ such that $l'_i > u'_i$ then RETURN: “The problem is infeasible”. Otherwise there exists a feasible solution.

Stage 3. Find a feasible solution.

Figure 3: An outline of Megiddo’s algorithm.

obtaining an inequality relating the two other variables. By repeating this for the inequalities along a cycle in the graph, we obtain an inequality in a single variable, *i.e.*, a bound on one of the variables in the cycle. Similarly, given a bound on a variable and an inequality containing that variable, we can use elimination to obtain a bound on an adjacent variable.

The first stage of Megiddo’s algorithm computes the tightest upper and lower bounds that can be obtained for a variable by eliminations along cycles. Let u_i and l_i be these bounds. Shostak [24], who suggested the general scheme of the algorithm, gave an implementation which requires exponential time. The first polynomial implementation is due to Aspvall and Shiloach [2]. Using symbolic computation ideas, Megiddo [19] turned the Aspvall – Shiloach algorithm into a strongly polynomial one, requiring $O(n^3 m \log m)$ time.

The second stage of Megiddo’s algorithm computes the bounds u'_j and l'_j , the tightest upper and lower bounds on x_j that can be obtained by repeated eliminations along simple paths, starting with the bounds l_i and u_i at x_i . This stage can be implemented as a simple dynamic program in $O(mn)$ time. Note that this dynamic program can be regarded as a decision tree of depth $O(n)$ with at most m comparisons at each node. If any of the upper bounds is less than the corresponding lower bound, then the problem is clearly infeasible. Shostak [24] proved the converse: if $l'_i \leq u'_i$ for every i , then a feasible solution exists. Moreover, in this case each interval $[l'_i, u'_i]$ is the projection of the feasible region onto the variable x_i .

If there is a feasible solution, then one is found in the third stage. The idea is to fix x_1 to some value between l'_1 and u'_1 and run the algorithm recursively in a smaller dimension. Observe that the values l_i and u_i do not need to be recomputed in the recursive calls. Therefore, the third stage can be implemented as a sequence of n dynamic programs and takes $O(n^2 m)$ time.

This algorithm, together with Theorem 3.1, implies a strongly polynomial algorithm to optimize a linear objective function with a constant number of non-zero coefficients over a region defined by a set of inequalities with at most two variables per inequality. Because of the special structure of Megiddo’s algorithm, the running time can be improved significantly. Assume that the objective function c uses the first d variables and define the convex set

$$\mathcal{P} = \{z \in \mathbf{R}^d : \exists x \text{ such that } Ax \leq b \text{ and } x_i = z_i \text{ for } i = 1, \dots, d\}$$

as in the proof of Theorem 3.1. We also precompute the bounds l_i and u_i for every variable x_i .

Lemma 4.1 Given the bounds l_i and u_i , there is a linear separation algorithm for $\mathcal{P}$ which runs in $O(nm)$ time and can be described by a decision tree of depth $O(n)$ with at most m comparisons at every node.

Proof: Testing whether $z \in \mathcal{P}$ is equivalent to checking the feasibility of a linear inequality system with at most two variables per inequality. Observe that the first two stages of the algorithm are enough to decide whether the system is feasible, so there is no need to run the third stage. Furthermore, the first stage uses only inequalities with exactly two non-zero coefficients; the coordinates of z appear only in inequalities with one variable. Therefore, the precomputed bounds l_i and u_i can be used instead of running the first stage. The second stage is a dynamic program which runs in the claimed time. ■

Lemma 4.1, together with Theorem 2.5 and Theorem 3.1, can be used to obtain the following result.

Theorem 4.2 If A is an $n \times m$ matrix with at most two non-zero entries per row and c is a vector with at most d non-zero coefficients, then for fixed d the linear program $\max\{cx : Ax \leq b\}$ can be solved in $O(n^3m \log n + n^{d+1}m \log^d m)$ time.

The most important problem of this form is the dual of the (minimum-cost) generalized transshipment problem [10,16,27]. In this generalization of the transshipment problem we are given a network G , gain-factors γ and costs c on the arcs, and demands b on the nodes; there are no capacities. A *generalized transshipment* is a non-negative flow vector f on the arcs, such that if flow $f(v, w)$ enters the arc (v, w) at node v then $\gamma(v, w)f(v, w)$ leaves at w . An example is the currency conversion problem: the flow represents money, the nodes represent different currencies, and the gain factors are conversion rates. A generalized transshipment is *feasible* if it satisfies the demands (*i.e.*, for every node v we have that $\sum_{(w,v)} \gamma(w, v)f(w, v) - \sum_{(v,w)} f(v, w) = b(v)$). The *(minimum-cost) generalized transshipment problem* is to find a feasible transshipment of minimum cost. The dual of the generalized transshipment problem is a problem with two variables per inequality. The variables are prices on the nodes, the inequalities require that the reduced costs be non-negative, and the coefficients of the objective function are the demands.

Adler and Cosares [1] extended Megiddo's strongly polynomial algorithm to the special case of the generalized transshipment problem where all demands have the same sign (e.g., all demands are non-negative). Theorem 4.2 implies that the generalized transshipment problem can be solved in strongly polynomial time if there are at most a constant number of non-zero demands. We show how to combine these ideas in order to construct a strongly polynomial algorithm for the generalized transshipment problem where all but a constant number of the demands have the same sign.

The algorithm of Adler and Cosares is based on the fact, proved by Cottle and Veinott [6], that in the dual of the generalized transshipment problem, all variables can be simultaneously minimized (or maximized). The idea behind the proof is that if all the demands are of the same sign, then optimizing with respect to one can be done independently of the optimization with respect to another. The optimum flow satisfying all of the demands is equal to the sum of the optimal flows with one non-zero demand each, since there are no capacities.

Consequently, when all of the demands have the same sign, an optimal solution can be found after the second stage of the feasibility testing algorithm (see Figure 3). If the network has n nodes and m edges, then the resulting algorithm takes $O(n^3 m \log m)$ time to compute l_i and u_i for all of the variables, and then uses dynamic programming to compute an optimal solution. It is a straightforward application of Theorem 3.1 to extend this to the case when all but a constant number of the excesses are non-negative. As in Theorem 4.2, we can use Theorem 2.5 to improve the time bound.

Theorem 4.3 The generalized transshipment problem in a network with n nodes and m arcs, where all but d demands have the same sign and d is fixed, can be solved in $O(n^3 m \log m + mn^{d+2} \log^{d+1} m)$ time.

Proof: Consider the linear programming dual of the problem. The variables are the node prices. Assume that the first d nodes have demands with different sign. First we precompute the bounds l_i and u_i for every node i . This takes $O(n^3 m \log m)$ time. Next we consider the polyhedron $\mathcal{P}$ in $d+1$ dimensions as defined in the proof of Theorem 3.1. The first d coordinates of this polyhedron are the dual variables corresponding to the d nodes whose demand have different sign and the $d+1$ st variable is the corresponding objective function value. Using the precomputed values u_i and l_i , the algorithm of Adler and Cosares can be used to separate from $\mathcal{P}$. This takes $O(nm)$ time. Moreover, the resulting algorithm can be written as a decision tree of depth $O(n)$ with at most m inequalities at each node. Since the resulting algorithm is linear in the input, the claim follows by applying Theorem 2.5. ■

References

- [1] I. Adler and S. Cosares. Strongly polynomial algorithms for linear programming problems with special structure. Unpublished manuscript, 1989.
- [2] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 9:827–845, 1980.
- [3] K. L. Clarkson. Linear programming in $O(3^{d^2})$ time. *Information Processing Letters*, 22:21–24, 1986.
- [4] K. L. Clarkson. Las Vegas algorithms for linear programming and integer programming when the dimension is small. In *Proc. 29th IEEE Annual Symposium on Foundations of Computer Science*, pages 452–456, 1988.
- [5] E. Cohen and N. Megiddo. Strongly polynomial-time and NC algorithms for detecting cycles in dynamic graphs. *Proc. of the 21st annual ACM Symposium on the Theory of Computing*, pages 523–534, May 1989.
- [6] R. W. Cottle and Jr. A. F. Veinott. Polyhedral sets having a least element. *Mathematical Programming*, 3:238–249, 1972.

- [7] M. E. Dyer. On a multidimensional search technique and its application to the euclidean one-centre problem. *SIAM Journal on Computing*, 15:725–738, 1986.
- [8] M. E. Dyer and A. M. Frieze. A randomized algorithm for fixed-dimensional linear programming. Unpublished manuscript, 1987.
- [9] A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.
- [10] A. Goldberg, S. Plotkin, and É. Tardos. Combinatorial algorithms for the generalized circulation problem. In *Proc. 29th IEEE Annual Symposium on Foundations of Computer Science*, pages 432–443, October 1988.
- [11] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.
- [12] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1988.
- [13] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4:373–395, 1984.
- [14] R. M. Karp and C. H. Papadimitriou. On linear characterization of combinatorial problems. *SIAM Journal on Computing*, 11:620–632, 1982.
- [15] L. G. Khachiyan. Polynomial Algorithms in Linear Programming. *Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki*, 20:53–72, 1980.
- [16] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Reinhart, and Winston, New York, NY., 1976.
- [17] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.
- [18] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.
- [19] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12:347–353, 1983.
- [20] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31:114–127, 1984.
- [21] M. W. Padberg and M. R. Rao. The Russian method for linear programming III: Bounded integer programming. Technical Report Research Report 81-39, New York University, Graduate School of Business Administration, 1981.
- [22] A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley & Sons, 1986.

- [23] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. Technical Report CSR-183, Department of Computer Science, New Mexico Tech., 1988.
- [24] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28:769–779, 1981.
- [25] É. Tardos. A Strongly Polynomial Minimum Cost Circulation Algorithm. *Combinatorica*, 5(3):247–255, 1985.
- [26] É. Tardos. A strongly polynomial algorithm for solving combinatorial linear programs. *Operations Research*, pages 250–256, 1986.
- [27] K. Truemper. On Max Flows with Gains and Pure Min-Cost Flows. *SIAM J. Appl. Math.*, 32:450–456, 1977.