A Dual Version of Tardos's Algorithm for Linear Programming

by
James B. Orlin

Sloan W.P. No. 1686-85 July 1985
Revised June 1986

Abstract

Recently, Eva Tardos developed an algorithm for solving the linear program min(cx: Ax = b, x $\geq$ 0) whose solution time is polynomial in the size of A, independent of the sizes of c and b. Her algorithm focuses on the dual LP and employs an approximation of the cost coefficients. Here we adopt what may be called a 'dual approach' in that it focuses on the primal LP. This dual approach has some significant differences from Tardos's approach which make the dual approach conceptually simpler.

Subject Classification:

742. A Dual Version of Tardos's Algorithm for Linear Programming.

Introduction

Recently, Eva Tardos [1986] developed a polynomial time algorithm for the linear program min(cx: Ax = b, x $\geq$ 0) that runs in time polynomial in m, n and $\log(A_{max} + 1)$, assuming that the four basic arithmetic operations are each counted as one step. Indeed, for linear programs developed from combinatorial problems such as the multi-commodity flow problem the algorithm is strongly polynomial, i.e., the number of arithmetic steps is polynomial in the dimension of the problem rather than in the size of the input.

Tardos's algorithm is based on a generalization of her strongly polynomial algorithm for minimum cost network flows (Tardos [1985]). Subsequently, Frank and Tardos [1985] have generalized Tardos's approach so that it is valid for linear programs (with potentially an exponential number of constraints) solved by the ellipsoidal algorithm.

Here we present a 'dual' variant of her algorithm. The major algorithmic ideas are the same, viz., both algorithms solve a sequence of approximated problems and in each instance identify at least one variable that must be zero in an optimal solution. Nevertheless, our algorithm does differ from Tardos's algorithm in more significant ways than is indicated by the dualization. On the negative side, the approximated problems here have m times as many bits as do the approximations used in Tardos's algorithm. Thus under the standard models of computation, our algorithm is less efficient than Tardos's algorithm. On the positive side, our algorithm is a "single phase" algorithm and is more direct and conceptually simpler than Tardos's "two-phase" algorithm. The first

phase of Tardos's algorithm finds the optimizer face of the LP. The second phase constructs a basic feasible solution to the optimizer face. Our algorithm simultaneously constructs a feasible solution and an optimum basic feasible solution.

Recently, Fujishige [1986] has developed a similar dual approach to Tardos to solve the capacitated minimum cost flow problem. His algorithm results in an improvement over Tardos's algorithm in the theoretical worst case running time, and provides a worst case time comparable to that of Orlin [1985]. Fujishige's algorithm has subsequently been improved upon by Galil and Tardos [1986].

The algorithm described below runs in $O(m^{6.5} \log(A_{max} + m + 1))$ arithmetic steps. The arithmetic operations of the algorithm include: addition, subtraction, integer division by a constant, integer multiplication by a constant and comparisons. In the case that $\log(A_{max})$ is polynomially bounded in m and n (as, for example, in multi-commodity network flow problems and in generalized covering and set packing problems), then the number of arithmetic operations is polynomially bounded in m and n. In such a case the algorithm is called strongly polynomial as per Tardos [1986].

One naturally may question the real significance of the existence of a strongly polynomial algorithm for linear programs. The significance of the concept does not lie in its practicality, since one would not in practice wish to solve an LP in which $\log(A_{max})$, $\log(b_{max})$ or $\log(c_{max})$ is exponential in m and n. Rather, strongly polynomial algorithms are significant in the theoretical domain.

A key theoretical motivation for studying the question of 'strong polynomiality' deals with the geometry of linear programs. If we compare the linear programs in which $\log(A_{max})$ is polynomially bounded to those in which $\log(A_{max})$ is exponential in m and n, the difference in terms of the structure of the polyhedra is subtle. It is of theoretical interest whether the facial structure of linear programs can grow continually more complex as $\log(A_{max})$ increases exponentially.

Similarly, one may wish to know if the complexity of solving linear programs stems in part from the large numbers involved. Such is apparently the case in integer programming with a bounded number of variables. Indeed, there is probably no strongly polynomial algorithm for 2-variable singly constrained integer programs since such an algorithm would imply that one could compute whether two numbers a and b are relatively prime in O(1) arithmetic steps. (Consider the integer program min (0x: ax-by=1; x, y $\geq$ 0 integer).)

1. The Problem and the Algorithm

Consider the linear programming problem

$$\begin{aligned} &\text{Minimize} && cx \\ &\text{Subject to} && Ax \geq b \\ & && x \geq 0 \ , \end{aligned} \qquad (1)$$

where c is an integral n-vector, A is an integral m x n matrix, and b is an integral m-vector. We let $A_{max} = \max(|a_{ij}| : i \in [1..m], j \in [1..n])$. In general, for any vector or matrix v, v_{max} denotes $\max(|v_i| : v_i \text{ is a component of } v)$.

The approximation method described below relies on the following two properties of linear programs:

(1) if there is an optimal solution then there is an optimal solution that is basic;

(2) the non-zero components of each basic solution may be obtained by premultiplication of the right hand side vector b by the inverse of some basis B of [A,-I]. Moreover, using Cramer's rule, we may bound the coefficients of B^{-1}.

FACT 1. Let $M = (A_{max})^{m+1}(m)(m!)$. Let B be any m x m invertible submatrix of [A,-I] and let $\bar{A} = B^{-1}[A,-I]$. Then each coefficient of $\bar{A}$ has both its numerator and denominator bounded by M.

PROOF. Let $M' = (A_{max})^m m!$. Then M' is an upper bound on the determinants of submatrices of A. By Cramer's rule M' is an upper bound on the numerator and denominator of each coefficient of B^{-1}. Multiplying B^{-1} by A does not increase the largest denominator and increases the largest numerator by a factor of at most mA_{max}. □

Rather than prove a number of additional facts and lemmas prior to the algorithm, we will first describe the essential aspects of the algorithm and subsequently prove that the total number of arithmetic operations for the algorithm is $O(m^{6.5} \log (A_{max} + m + 1))$.

The following "algorithm" is really an outline of our dual approach to Tardos's algorithm. Some of the key implementation details are discussed in the Lemmas and Theorem that follow. The value M refers to the upper bound on $\bar{A}_{max}$ defined in Fact 1.

Although it is not yet apparent, it is very important that the algorithm will solve both the LP and its dual.

ALGORITHM I.

STEP 1. Form an approximation to (1) as follows.

Choose an m-vector b' and an integer $k \geq 1$ so that

(i) $b'_j = \lfloor b_j / k \rfloor$ for $j \in [1..m]$, and

(ii) either $2(Mm)^{2m} \leq b'_{max} \leq 4(Mm)^{2m}$ or else

$b = b'$ and $b_{max} < 2(Mm)^{2m}$.

STEP 2. Solve the linear program

$$\begin{aligned} &\text{Minimize} && cx \\ &\text{Subject to} && Ax - Is = b' \qquad (2) \\ & && x, s \geq 0 \ . \end{aligned}$$

Case 1. Suppose there is no feasible solution to (2). In this case, there is also no feasible solution to (1) and the algorithm terminates. (A feasible solution x' to (1) would imply that x'/k is feasible for (2), a contradiction.)

Case 2. Suppose that the LP (2) is unbounded. In this case there is a solution $y \geq 0$ such that $Ay \geq 0$ and $cy < 0$. If (1) is feasible, then this vector y will also show that (1) is unbounded. In this case, we test (1) for feasibility by replacing c by 0 and returning to Step 2.

Case 3. Suppose that there is an optimal solution. In this case we let B be an optimal basis, and we go to Step 3.

STEP 3. If $B^{-1}b \geq 0$, then B is also an optimal basis for (1). Otherwise partition B into non-empty subbases D and E so that the 'final tableau' with respect to (1) is as follows.

$$\begin{aligned} \text{Minimize} \quad & z = \bar{c}_N x_N \\ \text{Subject to} \quad & x_D + \bar{N}_1 x_N = \bar{b}_D \\ & x_E + \bar{N}_2 x_N = \bar{b}_E \\ & x_D,\ x_E,\ x_N \geq 0, \end{aligned} \tag{3}$$

where $\bar{b}_D \geq 0$, and in addition, for $i \in D$, $j \in E$

$$\bar{b}_i \geq (Mm)2|\bar{b}_j| \quad . \tag{4}$$

STEP 4. (Recursively) find a basic optimal solution x_E^*, x_N^* to

$$\begin{aligned} \text{Minimize} \quad & q\left(\bar{c}_N x_N\right) \\ \text{Subject to} \quad & q\left(x_E + \bar{N}_2 x_N\right) = q\,\bar{b}_E \\ & x_E,\ x_N \geq 0. \end{aligned} \tag{5}$$

where q is $|DET(B)|$, and thus all coefficients of (5) are integral. If there is no feasible solution to (5), then there is also no feasible solution to (1). Otherwise, the following solution (6) is an optimal basic feasible solution for (1).

$$\begin{aligned} x_E &= x_E^* \\ x_N &= x_N^* \\ x_D &= \bar{b}_D - \bar{N}_1 x_N^*. \end{aligned} \tag{6}$$

Moreover, if B is the optimal basis of the basic solution in (6), then $c_B B^{-1}$ is an optimal dual solution.

To prove that the above 'algorithm' is polynomial in m, n and $\log(A_{max})$ (or equivalently, in n and log M), we still need to fill in a number of details; however, first let us look at the essential ideas.

If one were very optimistic, one might expect that one could solve the approximated problem in Step 2 and obtain the optimal basis. Unfortunately, it is conceivable that an optimal basis of (2) is not primal feasible for (1). In such a case, we can show that a splitting such as in (3) is possible. Moreover, the splitting of basic variables is such that $\bar{b}_D$ is so large relative to $\bar{b}_E$ that we can be assured that $x_D > 0$ in some optimal solution. Once we can be so assured, we may remove the variables x_D from consideration and thus reduce the original m constraint problem to a problem with fewer constraints. We then recursively apply the same algorithm to the smaller problem, and within m iterations we are through. (Similarly, we could have, in the spirit of Tardos's algorithm, replaced the dual inequality constraints $yA_D \leq c_D$ by equality constraints.)

LEMMA 1. We may determine the integer k and the vector b' in Step 1 in O(m log M) arithmetic steps.

PROOF. (This proof uses the same idea as Tardos, and is included for completeness.) We first determine b_{max} in at most m-1 comparisons. If $b_{max} \leq 2(Mm)^{2m}$ then we let k=1 and b'=b. Otherwise, we let $k = \lfloor b_{max}/2(Mm)^{2m} \rfloor$, and we let $b'_j = \lfloor b_j/k \rfloor$. In the case that $k \geq (Mm)^{2m}$, we may determine b' by using binary search in the interval $[0..4(Mm)^{2m}]$ to find the greatest integer d_j such that

$$b_j \geq d_j k.$$

In this way, we reduce the problem of computing $\lfloor b_j/k \rfloor$ to O(m log M) multiplication problems such that in each of these multiplications the smaller multiplier is less than $4(Mm)^{2m}$. Moreover, to divide by

a number less than $4(Mm)2m$, we may use standard algorithms which reduce such a division to a sequence of $\log(4(Mm)^{2m})$ arithmetic operations that are limited to the basic ones allowed in our description of strongly polynomial algorithms.

We also note that if $b_j = b_{max}$ and if $k \neq 1$, then

$$b'_j = \lfloor b_j/k \rfloor \geq b_j \left\lfloor \frac{2(Mm)^{2m}}{b_{max}} \right\rfloor = 2(Mm)^{2m}.$$

Also it is easy to show that $b'_i \leq b'_j \leq 4(Mm)^{2m}$. □

LEMMA 2. The optimal objective values of the linear programs in (3) and (5) are not unbounded.

PROOF. $\bar{c}_N \geq 0$. □

LEMMA 3. Either $B^{-1}b \geq 0$, or else there is a partition of B into non-empty subbases D and E as described in Step 3.

PROOF. If $B^{-1}b \geq 0$ then the result is trivially true. Suppose now that $B^{-1}b \not\geq 0$. In particular, $k > 1$ and $b'_{max} \geq (Mm)^{2m}$. And suppose without loss of generality that $|\bar{b}_1| \geq |\bar{b}_2| \geq \ldots \geq |\bar{b}_m|$. To prove the Lemma we will show that

$$\bar{b}_1 \geq (Mm)^{2m-1} k \tag{7}$$

and we will also show

$$\text{if } \bar{b}_j < 0 \text{ then } |\bar{b}_j| < Mmk\ , \tag{8}$$

where k is the value determined in Step 1.

From (7) and (8) and by our assumption that $\bar{b}_j < 0$ for some j,

$$|\bar{b}_1|/|\bar{b}_m| > (Mm)^{2m-2} \quad . \tag{9}$$

Since $|\bar{b}_1|/|\bar{b}_m| = \prod_{j=1}^{m-1} |\bar{b}_j|/|\bar{b}_{j+1}| > (Mm)^{2m-2}$,

it follows that there is some index r such that

$$|\bar{b}_r|/|\bar{b}_{r+1}| > (Mm)^2 \quad . \tag{10}$$

If there is more than one index satisfying (10), then choose r to be the least such index. We may then select D to consist of basic variables 1,...,r and select E to consist of basic variables r+1,...,m. From our choice of r, $\bar{b}_r > \bar{b}_1(Mm)^{-2m+4} > k(Mm)^3$. By (8), $\bar{b}_j > 0$ for $j \leq r$.

To complete the proof of the Lemma it suffices to prove that (7) and (8) are true. We now prove (7). Since $b = B\bar{b}$,

$$b_{max} \leq mB_{max}\ \bar{b}_{max} \leq mA_{max}\ \bar{b}_1$$

and thus

$$\bar{b}_1 \geq b_{max}/mA_{max} \geq k(Mm)^{2m-1} \quad .$$

This latter inequality is true since $b_{max}/k \geq (Mm)^{2m}$.

We now prove (8). Let $D = (d_{ij}) = B^{-1}$. From Step 1, we have chosen b' so that $b_j - k \leq kb'_j \leq b_j$.
Moreover, $B^{-1}b' \geq 0$. Suppose $(B^{-1}b)_i < 0$. Then

$$(B^{-1}b)_i = \sum_{j=1}^{m} d_{ij}\, b_j \geq \sum_{j=1}^{m} d_{ij}\, b_j - k \sum_{j=1}^{m} d_{ij}\, b'_j \qquad (11)$$

$$\geq -\sum_{j=1}^{m} |d_{ij}|\, |b_j - kb'_j| \geq -kMm.$$

The last inequality is valid since $|d_{ij}| \leq M$. This completes the proof of Lemma 3. □

LEMMA 4. Any of the linear programs (5) as recursively obtained by the algorithm will have a constraint matrix qC for which $qC_{max} \leq M$. Moreover, for any basis F of C, $(F^{-1})_{max} \leq M$.

PROOF. The coefficients of qC may be obtained by multiplying A by $Det(B)B^{-1}$ for some basis B. Thus $D_{max} \leq M$ by Fact 1. Assume now that the columns and rows have been permuted so that $A = [B, N_1, N_2]$ and that

$$B^{-1}A = \begin{bmatrix} I_1 & 0 & N_{11} & N_{12} \\ 0 & I_2 & C & N_{22} \end{bmatrix},$$

where I_1 and I_2 are identity matrices of appropriate dimension. (We may assume without loss of generality that F is disjoint from I_2 by making duplicates of the columns in $F \cap I_2$.) If we now pivot so that $\begin{bmatrix} I_1 & N_{11} \\ 0 & F \end{bmatrix}$ is transformed into the identity matrix, then □ I_2 is transformed into C^{-1}. Thus $(C^{-1})_{max} \leq M$ by Fact 1.

LEMMA 5. Algorithm A will either find some subproblem in Step 2 that has no feasible solution or else it will determine the optimal solution via equation (6).

PROOF. If the algorithm finds any subproblem (2) that is not feasible, then there is no feasible solution to (1) because we have not created any infeasibilities in the recursion (5) nor have we created any new infeasibilities by rounding down b in Step 1.

Next we consider the case in which there is a feasible solution to (1), and we prove our result inductively. Assume that our recursive algorithm has found an optimal solution (x_E^*, x_N^*) to the LP(5). We assume without loss of generality that the solution is basic. Thus there is some basis F of (5) such that

$$x_F = F^{-1} \bar{b}_E \quad .$$

By Lemma 4, if $t = (\bar{b}_E)_{max}$, and if i is a basic variable of F, then $x_i \leq Mmt$. Also by Lemma 4, $(N_1)_{max} < M$. Thus $(N_1 x_N^*)_{max} < M^2m^2t$. By our partition in Step 3, each coefficient $\bar{b}_i$ of $\bar{b}_D$ is such that $\bar{b}_i \geq M^2m^2t$. Thus

$$\bar{b}_D - N_1 x_N^* > 0 \quad ,$$

and the solution in (6) is primal feasible. Since it is also dual feasible, it is optimal. □

The key detail remaining is the implementation of Step 2. We refine Step 2 as follows. Let $\|.\|$ denote the sup norm.

STEP 2. If $\|c\| \leq 4(Mm)^{2m}$, then solve (2) using a minor modification of Karmarkar's algorithm as described below. If $\|c\| > 4(Mm)^{2m}$, then solve the dual of (2) using this algorithm recursively.

We will observe that the solution time in (2) is polynomial if $\|c\| \leq 4(Mm)^{2m}$ using our slight modification of Karmarkar's [1984] algorithm, and thus we may solve (1) in polynomial time if

$\|c\| \leq 4(Mm)^{2m}$. If $\|c\| > 4(Mm)^{2m}$ then the dual of (2) is a special case of (1) in which the cost coefficients are bounded by $4(Mm)^{2m}$. As such we may apply our algorithm recursively.

One cannot use Karmarkar's original algorithm for the following reason: in its original form Karmarkar's algorithm solves only the primal problem, whereas in Step 2 above we need to solve both the primal and dual problems. One easy method of resolving the difficulty is to replace b' by b' + ε^*, where the j-th component of ε^* is ε^j for some sufficiently small positive number ε. This is the well known perturbation method which guarantees that any basis is non-degenerate. As such, we can use Karmarkar's algorithm to identify an optimal primal solution and then use standard techniques to move to an optimal basic solution. Because of non-degeneracy this optimal basic solution has a basis B that is also dual feasible, and thus B induces both primal and dual optimal solutions.

As for ensuring that the algorithm is still polynomial, we may choose $\varepsilon = 1/M^2$. Cramer's rule shows that the fractional part of $B^{-1}[b' + \varepsilon^*]$ is non-zero and thus the solution is non-degenerate. This perturbation leads to an increase in the description of the right hand side by O(m log M) bits. However, in the case that $b'_{max} \geq (Mm)^{2m}$, this increase in the number of bits due to the perturbation method is only a constant multiplicative factor of the number of bits of b'.

An alternative method of resolving the difficulty in Step 2 is to avoid the need for obtaining an optimal dual solution. Instead, one can use the Adler and Hochbaum [1985] extension of the ellipsoid algorithm that runs in time polynomial in n, $\log(A_{max})$ and $\log(c_{max})$ independent of the size of b_{max}.

A plausible method that does not seem to work is solving the primal and dual problems simultaneously, e.g., one could reformulate (1) as

$$\begin{aligned} \text{minimize} \quad & c^t x - b^t y \\ \text{subject to} \quad & Ax \geq b \\ & A^t y \leq c \\ & x, y \geq 0 \quad . \end{aligned}$$

However, we cannot make the same reductions as stated in Step 2 in the case that c_{max} is large, since c appears both in the objective function and in the right-hand-side.

THEOREM 1. Algorithm 1 solves the linear program (1) in $O(m^{6.5} \log(m + A_{max} + 1))$ arithmetic operations.

PROOF. First, the optimality of the algorithm was shown in Lemma 5. Next, each of the approximated linear programs solved by Karmarkar's algorithm has an input size $L = O(m \log M)$ and thus the number of arithmetic steps per linear program is $O(m^{3.5} \log M) = O(m^{4.5} \log (m + A_{max} + 1))$. (We have $m + n$ variables rather than n because of the slack variables. Moreover, $m + n \leq 2m$ by assumption, and so the running time of Karmarkar's algorithm is $O(m^{3.5} \log M)$ arithmetic operations. Karmarkar includes another factor of L to handle the precision. We do not include this factor of L since we are counting only the number of arithmetic operations.) If $\|c\| \leq 4(Mm)^{2m}$, then the number of linear programs solved in Step 2 is at most m because at least one basic variable is eliminated in Step 3 at each iteration. If $\|c\| > 4(Mm)^{2m}$, then to solve (2) requires the solution of at most m linear programs. Thus there are $O(m^2)$ linear programs all together for a total running time of

$O(m^{5.5} \log M) = O(m^{6.5} \log (m + A_{max} + 1))$ for solving the linear programs. It is easy to verify that the other arithmetic operations also have time bounded in total by $O(m^{5.5} \log M)$. □

Conclusions

As noted by Tardos, her algorithm treats a problem of theoretical interest, i.e., LP's in which $\|c\| \geq (mM)^{2m}$ or $\|b\| \geq (mM)^{2m}$. Within this domain, Tardos's algorithm is a significant achievement.

Here we have shown that a dual version of Tardos's algorithm has a notable conceptual advantage: viz, the algorithm itself is simpler in that it solves linear programs in one rather than two phases. Moreover, computationally the algorithm is superior in special cases, such as for the transshipment problem. (See Fujishige [1985] and Galil and Tardos [1986].)

A major related theoretical question is: can we find an algorithm for linear programs whose number of arithmetic operations is bounded in m and n and independent of A_{max}? For this more general problem, it is not apparent how one can generalize Tardos's algorithm or the algorithm presented here.

Acknowledgments

I wish to thank Dorit Hochbaum for pointing out that Karmarkar's algorithm does not necessarily solve the dual LP. I also wish to thank Leslie Hall and an anonymous referee for suggestions on how to improve the presentation.

References

I. Adler and D. Hochbaum. (1985). On an implementation of the ellipsoidal algorithm, the complexity of which is independent of the objective function. In preparation.

A. Frank and E. Tardos. (1985). A combinatorial application of the simultaneous approximation algorithm. In preparation.

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

Z. Galil and E. Tardos. (1986). An $O(n^2 \log n(m+n \log n))$ Minimum Cost Flow Problem. Submitted for publication.

N. Karmarkar. (1984). A new polynomial time algorithm for linear programming. Combinatorica, 4, 373-395.

J. Orlin. (1985). Polynomial time simplex and non-simplex algorithms for the minimum cost flow problem. MIT, Sloan School working paper.

E. Tardos. (1985). A strongly polynomial minimum cost circulation algorithm. Combinatorica. To appear.

E. Tardos. (1986). A strongly polynomial algorithm to solve combinatorial linear programs. Operations Research. To appear.