SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0010 $01.25/0

TOWARDS A GENUINELY POLYNOMIAL ALGORITHM FOR LINEAR PROGRAMMING*

NIMROD MEGIDDO†

Abstract. A linear programming algorithm is called genuinely polynomial if it requires no more than $p(m, n)$ arithmetic operations to solve problems of order $m \times n$, where p is a polynomial. It is not known whether such an algorithm exists. We present a genuinely polynomial algorithm for the simpler problem of solving linear inequalities with at most two variables per inequality. The number of operations required is $O(mn^3 \log m)$. The technique used was developed in a previous paper where a novel binary search idea was introduced.

Key words. linear programming, genuinely polynomial-time, convex minimization

1. Introduction.

A major result in computational complexity theory was reported by Khachiyan [6] in 1979, namely, that the feasibility of linear inequalities can be decided in polynomial time. However, many researchers interested in linear programming have not been completely satisfied with Khachiyan's result for the following reasons. First, the fact that Khachiyan's algorithm is polynomial depends on the numbers being given in binary encoding. It is not hard (see [9]) to establish encoding schemes with respect to which Khachiyan's algorithm requires an *exponential* number of operations, although the operations themselves require polynomial time. The number of operations tends to infinity with the magnitude of the coefficients and thus for any given class of problems with fixed numbers of variables and inequalities, the number of arithmetic operations required by Khachiyan's algorithm is unbounded. Secondly, Khachiyan's algorithm has not yet been proven practical, while the simplex algorithm is usually efficient [4].

By solving a set of linear inequalities we mean producing a feasible solution or else recognizing that the set is infeasible. An interesting open question is the following; Do there exist an algorithm and a polynomial $p(m, n)$ such that every set of m linear inequalities with n variables is solved by the algorithm in less than $p(m, n)$ arithmetic operations? We shall call such an algorithm *genuinely polynomial.* It is not even known whether the transportation problem has a genuinely polynomial algorithm. The scaling method of Edmonds and Karp [5] has a polynomial time-bound but, as in Khachiyan's algorithm, the number of arithmetic operations depends on the magnitude of the coefficients.

In this paper we shall be discussing a special type of system of linear inequalities, namely, sets of m inequalities with n variables but no more than two variables per inequality. Previous results were obtained by Chan [3] and Pratt [11]. They solved the special case of inequalities of the form $x - y \leqq c$ (i.e., the dual of a shortest-path problem) in $O(n^3)$ operations. Shostak [12] developed a nice theory, on which we base our results in this paper, but his algorithm is exponential in the worst-case. Nelson [10] gave an $O(mn^{\lceil \log_2 n \rceil + 4} \log n)$ algorithm. Polynomial-time algorithms for this problem were given by Aspvall and Shiloach [2] and by Aspvall [1]. The former requires $O(mn^3 I)$ arithmetic operations, where I is the size of the binary encoding of the input, while the latter requires $O(mn^2 I)$ operations.

We shall present an algorithm which requires $O(mn^3 \log m)$ operations, i.e., a genuinely polynomial algorithm for solving systems of linear inequalities of order

* Received by the editors August 17, 1981, and in revised form July 15, 1982. This research was partially supported by the National Science Foundation under grants ECS7909724 and ECS8121741. The work was done while the author was visiting at Northwestern University.

† Statistics Department, Tel Aviv University, Tel Aviv, Israel.

$m \times n$ with at most two variables per inequality. Our algorithm is based on that of Aspvall and Shiloach [2] and on Shostak's [12] result. A similar construction can be based on Aspvall's [1] algorithm but no better complexity is obtained. Thus, although this paper is intended to be self-contained, the reader may find it helpful to refer to [2] and [12] for further clarifications.

2. Preliminaries. Given is a set S of m linear inequalities involving n variables but no more than two variables per inequality. Suppose $S = S_1 \cup S_2$, where S_i is the set of inequalities involving exactly i distinct variables ($i = 1, 2$). Without loss of generality, assume that S_1 is given in the form $\mathrm{lo}(y) \leqq y \leqq \mathrm{up}(y)$, where $\mathrm{lo}(y)$ and $\mathrm{up}(y)$ are the lower and upper bounds, respectively, on the variable y; these bounds may be infinite. It will be convenient to maintain for every variable y a list of all the inequalities in which y participates.

Throughout the computation there will be derived more and more restrictive lower and upper bounds, $\underline{y}$ and $\bar{y}$ respectively, for each variable y. The basic step of updating such bounds makes use of a single inequality from S_2. Given the current bounds $\underline{y}$, $\bar{y}$ on y and any inequality $ay + bz \leqq c$ in which y participates ($a, b \neq 0$), the bounds on z may be updated in an obvious way. We define the routine FORWARD (y, $ay + bz \leqq c$) to be the updating procedure which operates according to the following case classification:

$$
\begin{array}{ll}
\text{case (i): } a, b > 0, & \bar{z} \leftarrow \min[\bar{z}, (c - a\underline{y})/b], \\
\text{case (ii): } a > 0, b < 0, & \underline{z} \leftarrow \max[\underline{z}, (c - a\underline{y})/b], \\
\text{case (iii): } a < 0, b > 0, & \bar{z} \leftarrow \min[\bar{z}, (c - a\bar{y})/b], \\
\text{case (iv): } a, b < 0, & \underline{z} \leftarrow \max[\underline{z}, (c - a\bar{y})/b].
\end{array}
$$

The routine FORWARD detects infeasibility when $\bar{z} < \underline{z}$.

The routine FORWARD may repeatedly be applied along "chains" of inequalities. Specifically, a sequence of inequalities $a_i y_i + b_i y_{i+1} \leqq c_i$, $i = 1, \cdots, k$, may be used for updating the bounds on y_{k+1} by starting from the bounds on y_1 and updating $\underline{y_{i+1}}$, $\overline{y_{i+1}}$ according to the updated $\underline{y_i}$, $\overline{y_i}$ ($i = 1, \cdots, k$). Consider the case where the initial bounds are $\underline{y} = \mathrm{lo}(y)$, $\bar{y} = \mathrm{up}(y)$ for all $y \neq y_1$ and $\underline{y_1} = \overline{y_1} = g$, where g is any real number. Obviously, the bounds that will be derived with respect to $y_2, \cdots, y_{k+1}$ will be linear functions of g (not excluding the possibility of infinite bounds).

A special case of chains is that of a "loop", i.e., when y_{k+1} and y_1 are the same variable, which we now denote by x. Consider, for example, a case where applying the routine FORWARD around a loop starting and ending at x yields $\underline{x} = \alpha g + \beta$. A necessary condition for feasibility is that $x \geqq \alpha x + \beta$. This is an inequality which is "hidden" in our loop and obviously has the following consequences:

(i) If $\alpha = 1$ and $\beta > 0$ then S is infeasible; in this case we say that the loop is infeasible.

(ii) If $\alpha < 1$ then $x \geqq \beta/(1-\alpha)$ is a necessary condition for feasibility.

(iii) If $\alpha > 1$ then $x \leqq \beta/(1-\alpha)$ is necessary.

Obviously, the number $h = \beta/(1-\alpha)$ (in case $\alpha \neq 1$) is the solution of the equation $g = \alpha g + \beta$. Suppose we apply the routine FORWARD around each simple loop and along every simple chain. If either an infeasible loop is discovered or an infeasibility is detected by FORWARD (in the form $\underline{z} > \bar{z}$) then the problem is infeasible; otherwise, we may adjoin all the necessary conditions so obtained to our set of inequalities and that of course will not restrict the set of solutions. By doing this we obtain what Shostak [12] calls a *closure* S' of our set of inequalities. Shostak's main theorem states that S is feasible if and only if S' does not have any infeasible simple loop nor a simple chain along which FORWARD detects infeasibility. This is the essence of Shostak's algorithm. That algorithm is exponential since it needs to consider all simple loops.

Aspvall and Shiloach obtained a polynomial-time algorithm by considering another extension S^* of S. Specifically, $S^* = S_1^* \cup S_2$ where S_1^* is the set of the most restrictive inequalities in S' with respect to a single variable and S_2 is the original set of inequalities involving exactly two variables. Following Aspvall and Shiloach we denote those most restrictive bounds for a variable x by $x$low and $x$high, i.e., S_1^* consists of the inequalities $x\text{low} \leqq x \leqq x\text{high}$. Once $x$low and $x$high have been found, Aspvall and Shiloach can find a solution, or else recognize infeasibility, in $O(mn^2)$ operations. We shall develop an $O(mn^2 \log m)$ algorithm for finding $x$low and $x$high for a single variable x.

3. The functions $r(g)$ and $r'(g)$. It has already been noted that the bounds obtained at the end of a fixed chain are themselves linear functions of the value g which is assigned to the variable at the start of the chain. Let x be an arbitrary variable. We define $r(g)$ to be the largest lower-bound on x which may be obtained in one of the following ways: (i) Apply FORWARD along any chain of length not greater than n, with the initial bounds $\underline{y} = \text{lo}(y)$, $\bar{y} = \text{up}(y)$ for all y, (ii) Apply FORWARD around any loop of length not greater than n, starting and ending at x (where x is the selected variable) with the same initial bounds except for $\underline{x} = \bar{x} = g$. Analogously, $r'(g)$ is defined to be the least upper bound on x that may be obtained in such a way. It follows that $r(g)$ is convex piecewise-linear function of g while $r'(g)$ is concave and piecewise linear.

By definition, if g is a feasible value of x (i.e., there is a solution of S in which $x = g$) then, necessarily, $r(g) \leqq g \leqq r'(g)$. The properties of the functions r, r' imply that the set of the values of g such that $r(g) \leqq g \leqq r'(g)$ is convex, i.e., there exist (possibly infinite) numbers a, b such that $r(g) \leqq g \leqq r'(g)$ if and only if $a \leqq g \leqq b$. If this set is empty we take $a = \infty$, $b = -\infty$. On the other hand, if h is either a lower or an upper bound which is hidden in a loop then there exist $\alpha \neq 1$ and β such that $h = \alpha h + \beta$ and either $\alpha g + \beta \leqq r(g)$ for all $g \in [a, b]$ or $\alpha g + \beta \geqq r'(g)$ for all $g \in [a, b]$. Moreover, if h is a bound obtained from a chain then either $h \leqq r(g)$ or $h \geqq r'(g)$ for every g. It thus follows (see Fig. 1) that the endpoints a, b are precisely the most

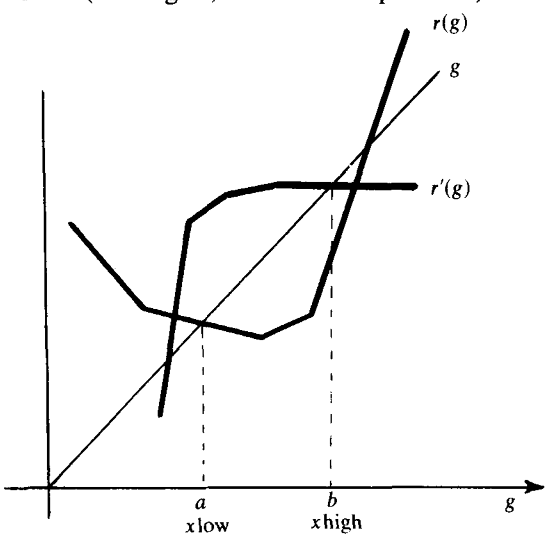

FIG. 1

restrictive bounds that may be obtained either along chains or around loops (all of length no greater than n), i.e., $a = x\text{low}$ and $b = x\text{high}$. In other words, $x\text{low} = \min\{g : r(g) \leqq g \leqq r'(g)\}$ and $x\text{high} = \max\{g : r(g) \leqq g \leqq r'(g)\}$. We shall develop a search algorithm for $x$low and $x$high.

4. A useful generalization. As a matter of fact, we can handle a more general situation which is more convenient to describe. Consider the function $R(g) = \min[r'(g) - g, g - r(g)]$. Note that this function is defined with respect to a variable x. Obviously, $r(g) \leqq g \leqq r'(g)$ if and only if $R(g) \geqq 0$, while R is concave and piecewise linear. We are interested in finding $a = \min\{g: R(g) \geqq 0\}$ and $b = \max\{g: R(g) \geqq 0\}$. Let $R_+(g)$ and $R_-(g)$ denote the slopes of R at g on the right-hand side and on the left-hand side, respectively. Thus, $R_-(g) \geqq R_+(g)$ and this inequality is strict if and only if g is a breakpoint of R. If $R(g)$, $R_+(g)$ and $R_-(g)$ are known at a certain g, then the location of g relative to a and b can be decided according to the following table:

$R(g) \geqq 0$	$a \leqq g \leqq b$
$R(g) > 0,\ \ R_-(g) \geqq 0$	$g < a$
$R(g) < 0,\ \ R_+(g) \leqq 0$	$g > b$

Note that this table exhausts all possible cases since $R_-(g) \geqq R_+(g)$. Furthermore, if $R_-(g) \geqq 0 \geqq R_+(g)$ and $R(g) < 0$, then R takes on only negative values ($a = \infty$, $b = -\infty$).

An algorithm for evaluating $r(g)$ and $r'(g)$ (with respect to a variable x) was given by Aspvall and Shiloach [2]. To conform with the notation used in the present paper, we state the following algorithm which is essentially the same as Algorithm 1 in [2].

```
procedure EVAL(g);
  begin
    for each variable y   [ȳ ← up(y); y̲ ← lo(y)];
    x̄ ← min(x̄, g); x̲ ← max(x̲, g);
    for i ← 1 until n do
      begin
        for each y and each ay + bz ≦ c   FORWARD (y, ay + bz ≦ c);
      end
    r ← x̲; r' ← x̄;
  end
```

Clearly, EVAL(g) requires $O(mn)$ arithmetic operations. For our purposes we need to know not only $r(g)$ and $r'(g)$ but also the one-sided slopes of r and r' at g. Thus, we have to modify EVAL a little. Imagine all the quantities $\underline{x}$, $\bar{y}$ (including $\underline{x}$ and $\bar{x}$) to be themselves functions of g in some neighborhood of a given value. There exists a neighborhood over which all these functions consist of at most two linear pieces with the given g being the unique breakpoint. It is fairly simple to keep track of the slopes of these linear pieces. At the start, every y has both $\bar{y}$ and $\underline{y}$ with slope zero on both sides. The next step is $\bar{x} \leftarrow \min(\bar{x}, g)$. Here we have one of the following cases: (i) If $g < \text{up}(x)$ then $\bar{x}$ has slope unity on both sides. (ii) If $g = \text{up}(x)$ then $\bar{x}$ has slope zero on the right-hand side and slope unity on the left-hand side. (iii) If $g > \text{up}(x)$ then $\bar{x}$ has both slopes equal to zero. Later, when functions are multiplied by constants (see the routine FORWARD), the slopes are multiplied by the same constants. Adding a constant does not affect the slope. The effect of the "min" operation is also straightforward. Without loss of generality assume we perform $f_3 \leftarrow \min(f_1, f_2)$, where $f_1 \leqq f_2$. The solution is as follows. If $f_1(g) < f_2(g)$ then f_3 inherits its slopes from f_1; otherwise, if $f_1(g) = f_2(g)$ then f_3 inherits the minimum slope on either side of g. Thus, in general, as long as in the evaluation of $R(g)$ the variable g is involved only in comparisons, additions and multiplications by constants, we can evaluate the slopes $R_+(g)$ and $R_-(g)$ with the same computational complexity as that of $R(g)$. In our particular case this is $O(mn)$.

5. Solving $R(g) \geqq 0$. We shall now develop an algorithm for finding a and b. Assume that we have an algorithm for evaluating $R(g)$ such that g itself is involved only in comparisons, additions and multiplications by constants (and R is a concave function of g). In view of the discussion in the preceding section, we assume without loss of generality that this algorithm computes not only $R(g)$ but also the slopes $R_+(g)$ and $R_-(g)$.

We maintain bounds $\underline{a}$, $\bar{a}$, $\underline{b}$, $\bar{b}$ which are repeatedly updated and always satisfy $\underline{a} \leqq a \leqq \bar{a}$ and $\underline{b} \leqq b \leqq \bar{b}$. The initial values are $\underline{a} = \underline{b} = -\infty$ and $\bar{a} = \bar{b} = \infty$. The basic idea is to follow the known algorithm for evaluating R with g being indeterminate; however, g will always be confined to $D = [\underline{a}, \bar{a}] \cup [\underline{b}, \bar{b}]$. Whenever the result of the succeeding step depends on the value of g within D, a test which amounts to one R-evaluation (i.e., with a specific argument g) is performed, in order to update D appropriately. The fundamental principle used here was first introduced in [7] and later applied in [8].

The details are as follows. At the start, the available quantities are the indeterminate g together with several constants, while $D = [-\infty, \infty]$. We distinguish two phases in the computation: Phase 1 lasts as long as $\underline{a} = \underline{b}$ and $\bar{a} = \bar{b}$; when this does not hold any more then we are in Phase 2. Consider a typical point at Phase 1. Assume, by induction on the number of steps since the start, that all the "program variables" are linear functions of g over D, possibly constants. If the next operation is an addition or a multiplication by a constant, then it can be carried out with the indeterminate g over the entire D. Suppose the next operation is a comparison, $f_3 \leftarrow \min(f_1, f_2)$, say. If the linear functions f_1 and f_2 do not intersect over D, or if they coincide over D, then the assignment can be carried out symbolically and f_3 is a linear function of g over D; otherwise, denote the intersection point by g' and assume, without loss of generality, that $f_1(g) < f_2(g)$ for $g < g'$ while $f_2(g) < f_1(g)$ for $g > g'$ $(g \in D)$. At this point we test the value g', i.e., we evaluate $R(g'), R_+(g')$ and $R_-(g')$ and update D as follows:

$R(g') \geqq 0$	(enter[1] Phase 2) $\bar{a} \leftarrow g'$; $\underline{b} \leftarrow g'$; $f_3 \leftarrow \min(f_1, f_2)$
$R(g') < 0$ and $R_-(g') \geqq 0$	$\underline{a} \leftarrow g'$; $\underline{b} \leftarrow g'$; $f_3 \leftarrow f_2$
$R(g') < 0$ and $R_+(g') \leqq 0$	$\bar{a} \leftarrow g'$; $\bar{b} \leftarrow g'$; $f_3 \leftarrow f_1$

If Phase 1 continues then all the available quantities remain linear functions of g over the updated D.

[1] Phase 2 will work on the two intervals separately; the assignment will be different but constant over each interval.

When Phase 2 starts we have $\bar{a}=\underline{b}$ and all the quantities consist of at most two linear pieces with the breakpoint occurring at $\bar{a}=\underline{b}$). During Phase 2 we split the computation of a from that of b. Consider, for example, the computation of a. We continue with the evaluation of R, where g is indeterminate but confined to $[a, \bar{a}]$. The situation is very similar to that of Phase 1. If g' and f_1, f_2 and f_3 are as before, then the assignments are according to the following table:

$R(g') \geqq 0$	$\bar{a} \leftarrow g'; f_3 \leftarrow f_1$
$R(g') < 0$ and $R_-(g) \geqq 0$	$\underline{a} \leftarrow g'; f_3 \leftarrow f_2$
$R(g') < 0$ and $R_+(g) \leqq 0$	$\bar{a} \leftarrow g'; f_3 \leftarrow f_1$

As a result we have $R(g)$ as a linear function over $[\underline{a}, \bar{a}]$. It is then straightforward to decide which of the following is the case: (i) There is a unique solution to $R(g)=0$ over $[\underline{a}, \bar{a}]$; this solution is then assigned to a(i.e., $\underline{a}=\bar{a}$). (ii) $R(g) \geqq 0$ for all $g \in [\underline{a}, \bar{a}]$; this is possible only if $\underline{a}=-\infty$, in which case $a \leftarrow -\infty$. (iii) $R(g)<0$ for all $g \in [\underline{a}, \bar{a}]$; this is possible only if $\bar{a}=\infty$, in which case $a \leftarrow \infty$ and $R(g)<0$ for every real g (i.e., infeasible system). The computation of b is analogous.

If the evaluation of R at a single g requires T operations, including C comparisons, then the computation of a and b takes O(CT) operations, since it amounts to O(C) evaluations of R (see [7] for a more detailed discussion of this point).

6. Finding $x$low and $x$high. When we solve $r(g) \leqq g \leqq r'(g)$ (equivalently, $R(g) \geqq 0$) according to the scheme presented in the preceding section, we run the routine EVAL with g being indeterminate. However, here we do not have to test every critical value g' right away. Specifically, consider for example the value of $\underline{z}$ which is obtained at the end of the second loop of a single iteration of EVAL (i.e., while i is fixed). As a function of g over D , this is the maximum envelope of the linear functions corresponding to the different inequalities in which z participates together with the previous function corresponding to $\underline{z}$. If there are m_z such inequalities, then we can can find all the breakpoints of the maximum function in $O(m_z \log m_z)$time (see the Appendix of [7]). Thus, the set of all breakpoints produced during one interation can be found and sorted in $O(m \log m)$ time. Assuming that these breakpoints are $g_1 \leqq \cdots \leqq g_q$ ($q=0(m)$), we may perform a binary search over these q values which amounts to testing only $O(\log q)$ of them. If this occurs during Phase 1, then by testing the number $g_{[q/2]}$ we either enter Phase 2 or discard approximately a half of the set of critical values. During Phase 2 each test cuts the set of critical values (lying in $[\underline{a}, \bar{a}]$, say) in half. Thus, the computation of $x$low and $x$high takes n stages during each of which we have to evaluate $r(g)$ and $r'(g)$ at $O(\log m)$ values of g. This amounts to $O(mn^2 \log m)$ arithmetic operations. This procedure needs to be repeated for every other variable so that the bounds $x$low and $x$high are found for all variables x in $O(mn^3 \log m)$ time.

7. Solving S. Let $y$low and $y$high denote the bounds obtained in the previous section. The following routine (which was essentially given by Aspvall and Shiloach [2]) either discovers that S is infeasible or else produces a feasible solution ($x_j = x_j^*, j = 1, \cdots, n$):

procedure FINAL:
 begin
  for each variable x $[\bar{x} \leftarrow x\text{high}; \underline{x} \leftarrow x\text{low}]$;
  for $j \leftarrow 1$**until** n **do**
   begin
    for $i \leftarrow 1$ **until** n **do**
    begin
     for each y and $(ay + bz \leqq c)$ FORWARD $(y, ay + bz \leqq c)$;
    end
    if there is a finite ξ such that $x_j \leqq \xi \leqq \overline{x_j}$ **then** $[\underline{x_j} \leftarrow \xi$;
    $x_j^* \leftarrow \xi; \overline{x_j} \leftarrow \xi]$ **else return** (INFEASIBLE);
   end
  return$(x_j = x_j^*, j = 1, \cdots, n)$;
 end

The validity of the routine FINAL follows from Shostak's theorem. Since we are now working with the set of inequalities extended so as to include the necessary conditions $x\text{low} \leqq x \leqq x\text{high}$, if no infeasible loops or chains of length n are discovered, then the problem is feasible.

The routine FINAL takes only $O(mn^2)$ operations, i.e., the whole process is dominated by the computation of the bounds $x$low and $x$high for all the variables. The genuinely polynomial algorithm hence runs in $O(mn^3 \log m)$ operations.

Acknowledgment. The author is grateful to the referees for their helpful comments.

REFERENCES

[1] B. ASPVALL, *Efficient algorithms for certain satisfiability and linear programming problems*, Ph.D. dissertation, Dept. Computer Science, Stanford University, Stanford, CA, August 1980.
[2] B. ASPVALL AND Y. SHILOACH, *A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality*, this Journal, 9 (1980), pp. 827–845.
[3] T.-H. CHAN, *An algorithm for checking PL/CV arithmetic inferences*, Report TR-77-326, Dept. Computer Science, Cornell University, Ithaca, NY, 1977.
[4] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, NJ, 1963.
[5] J. EDMONDS AND R. M. KARP, *Theoretical improvements in algorithmic efficiency for network flow problems*, J. Assoc. Comput. Mach., 19 (1972), pp. 248–264.
[6] L. G. KHACHIYAN, *A polynomial algorithm in linear programming*, Soviet Math. Dokl., 20 (1979), pp. 191–194.
[7] N. MEGIDDO, *Combinatorial optimization with rational objective functions*, Math. OR, 4 (1979), pp. 414–424.
[8] ———, *Applying parallel computations algorithms in the design of serial algorithms*, Proc. IEEE 22nd Symposium on Foundations of Computer Science, 1981, pp. 399-408.
[9] ———, *Is binary encoding appropriate for the problem-language relationship*?, Theoret. Comp. Sci., to appear.
[10] C. G. NELSON, *A* $n^{\log n}$ *algorithm for the two-variable-per-constraint linear programming satisfiability problem*, Tech. Rep. AIM-319, Dept. Computer Science, Stanford University, Stanford, CA, 1978.
[11] V. R. PRATT, *Two easy theories whose combination is hard*, unpublished manuscript, 1977.
[12] R. SHOSTAK, *Deciding linear inequalities by computing loop residues*, J. Assoc. Comput. Mach., 28 (1981), pp. 769–779.