Megiddo, Nimrod

Working Paper

Towards a Genuinely Polynomial Algorithm for Linear Programming

Discussion Paper, No. 493

Provided in Cooperation with:
Kellogg School of Management - Center for Mathematical Studies in Economics and Management Science, Northwestern University

Suggested Citation: Megiddo, Nimrod (1981) : Towards a Genuinely Polynomial Algorithm for Linear Programming, Discussion Paper, No. 493, Northwestern University, Kellogg School of Management, Center for Mathematical Studies in Economics and Management Science, Evanston, IL

This Version is available at:
https://hdl.handle.net/10419/220853

DISCUSSION PAPER NO. 493

TOWARDS A GENUINELY POLYNOMIAL ALGORITHM FOR LINEAR PROGRAMMING*

by

Nimrod Megiddo

August 1981

Statistics Department, Tel Aviv University
Tel Aviv, Israel

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 solving linear inequalities with at most two variables per inequality. The number of arithmetic operations which we require is $O(mn^3 \log m)$.

* This work was done while the author was visiting at Northwestern University. The work was supported in part by National Science Foundation under Grant #ECS7909724.

1. INTRODUCTION

A major result in computational complexity theory was reported by Khachiyan [K] in 1979, namely, that the feasibilty 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. This is because the fact, that the bound is polynomial in terms of the input size, depends on the numbers being given in binary encoding. In particular, even for fixed numbers of variables and inequalities, the number of arithmetic operations required by Khachiyan's algorithm is unbounded. This number tends to infinity with the magnitude of the coefficients even though it is bounded by a polynomial in the logarithms of these numbers. It is not hard to establish encoding schemes with respect to which Khachiyan's algorithm requires an exponential number of operations, although the operations themselves require polynomial-time. It should also be mentioned that Khachiyan's algorithm has not yet been proven practical.

An interesting question which is still open is the following: Is there an algorithm, and is there 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 for the min-cost flow problem by Edmonds and Karp [EK] has a polynomial time-bound but, like in Khachiyan's algorithm, the number of arithmetic operations depends on the magnitude of the coefficients.

In this paper we shall be dealing with a special type of linear inequalities,

---

* By solving we mean producing a feasible solution or else deciding that the set is infeasible.

namely, sets of m inequalities with n variables but no more than two variables per inequality. Previous results with respect to this problem are as follows. Pratt [P] solves the special case of inequalities of the form $x - y \leq c$ (i.e., the dual of a shortest-path problem) in $O(n^3)$ operations. Shostak [S] developed a nice theory, on which we base our results in this paper, but his algorithm is exponential in the worst-case. Nelson [N] 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 [AS] and by Aspvall [A]. 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 $m \times n$ with at most two variables per inequality. Our algorithm is based on that of Aspvall and Shiloach [AS] and on Shostak's [S] result. A similar construction can be based on Aspvall's [A] algorithm but no better complexity is obtained. Thus, although this paper is aimed to be self-contained, the reader may find it helpful to refer to [S] and [AS] 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 variables ($i = 1,2$). Without loss of generality, assume that S_1 is given in the form $lo(y) \leq y \leq up(y)$, where $lo(y)$ and $up(y)$ 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 resrictive lower and upper bounds, $\underline{y}$ and $\overline{y}$ 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 $\underline{y}$, $\overline{y}$ on y and any inequality $ay + bz \leq c$ in which y participates ($a,b \neq 0$ and assuming y and z to be distinct) the bounds on z may be updated in an obvious way. We define the routine FORWARD(y , $ay + bz \leq c$) to be the updating procedure which operates according to the following case classification:

case (i) : $a , b > 0$ $\quad \overline{z} \leftarrow \min[\ \overline{z} , (c - a\underline{y})/b\ ]$

case (ii) : $a > 0 , b < 0$ $\quad \underline{z} \leftarrow \max[\ \underline{z} , (c - a\overline{y})/b\ ]$

case (iii) : $a < 0 , b > 0$ $\quad \overline{z} \leftarrow \min[\ \overline{z} , (c - a\overline{y})/b\ ]$

case (iv) : $a , b < 0$ $\quad \underline{z} \leftarrow \max[\ \underline{z} , (c - a\underline{y})/b\ ]$

The routine FORWARD detects infeasibilty when $\overline{z} < \underline{z}$.

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

A special case 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 $\underline{x} = \alpha g + \beta$.

Equivalently, a necessary condition for feasibility is that $x \geq \alpha x + \beta$. 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 $x \geq \beta/(1 - \alpha)$ is a necessary condition for feasibility.

(iii) If $\alpha > 1$ then $x \leq \beta/(1 - \alpha)$ 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 $\underline{z} > \overline{z}$) 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 [S] calls a <u>closure</u> 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 xlow and xhigh, i.e., S_1^* consists of the inequalities $\text{xlow} \leq x \leq \text{xhigh}$. Once xlow and xhigh 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 xlow and xhigh 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 no greater than n, with the initial bounds $\underline{y} = lo(y)$, $\overline{y} = up(y)$. (ii) Apply FORWARD around any loop of length no greater than n, starting and ending at x, with the same initial bounds except for $\underline{x} = \overline{x} = g$. 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 a 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, $r(g) \leq g \leq r'(g)$. The properties of the functions r, r' imply that the set of the values of g such that $r(g) \leq g \leq r'(g)$ is convex, i.e., there exist (possibly infinite) numbers a,b such that $r(g) \leq g \leq r'(g)$ if and only if $a \leq g \leq b$. On the other hand, if h is either a lower or an upper bound which is hidden in a loop then $h = \alpha h + \beta$ $(\alpha \neq 1)$ where either $\alpha g + \beta \leq r(g)$ for all $g \in [a,b]$ or $\alpha g + \beta \geq r'(g)$ for all $g \in [a,b]$. Moreover, if h is a bound obtained from a chain then either $h \leq r(g)$ or $h \geq r'(g)$ for every g. It thus follows (see Fig. 1) that the endpoints a,b are precisely the most restrictive bounds that may be obtained either along chains or around loops (all of length no greater than n), i.e., a = xlow and b = xhigh.

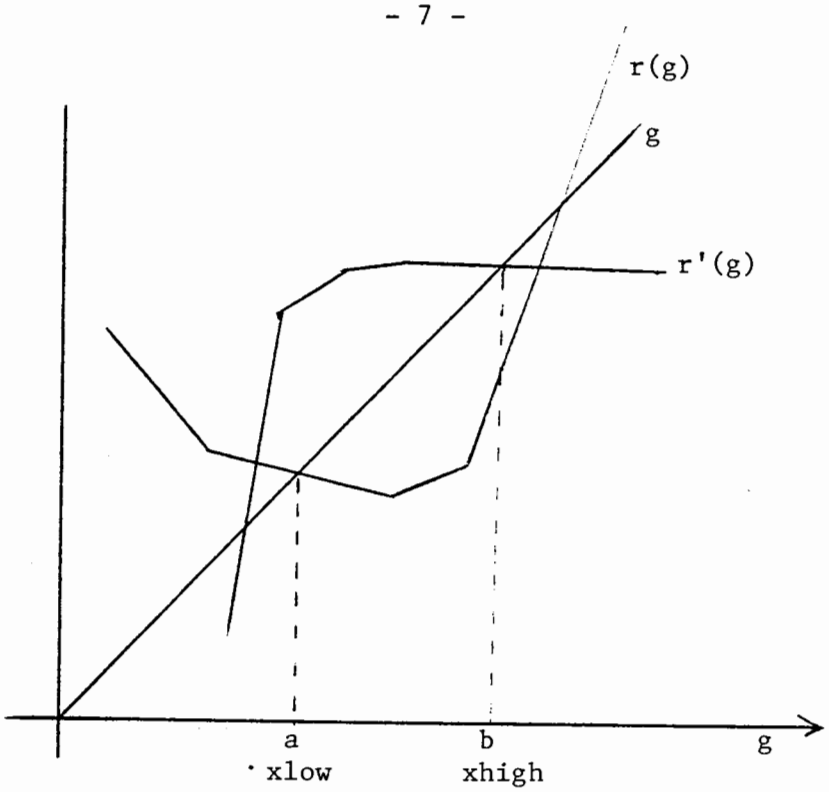

Fig. 1

In other words, xlow = min{ g : r(g) $\leq$ g $\leq$ r'(g)} and xhigh = max { g : r(g) $\leq$ g $\leq$ r'(g) }. We shall develop a search algorithm for xlow and xhigh .

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 $R(g) = \min[\, r'(g) - g \,,\, g - r(g)]$. Obviously, $r(g) \leq g \leq r'(g)$ if and only if $R(g) \geq 0$ while R is concave and piecewise-linear. We are interested in finding $a = \min \{ g : R(g) \geq 0 \}$ and $b = \max \{ g : R(g) \geq 0 \}$. 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, $R_-(g) \geq R_+(g)$ 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:

$R(g) \geq 0$	$a \leq g \leq b$
$R(g) < 0$, $R_-(g) \geq 0$	$g < a$
$R(g) < 0$, $R_+(g) \leq 0$	$g > b$

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

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

```
procedure EVAL(g) ;
  begin
    for each variable y [ ȳ ← up(y) ; y ← lo(y) ] ;
    x̄ ← min(x̄,g) ; x ← max(x,g) ;
    for i ← 1 until n do
      begin
        for each y and each ay+bz ≤ c FORWARD ( y , ay+bz ≤ c );
      end
    r ← x ; 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 $\underline{y}$, $\bar{y}$ (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. Then, when we update $\overline{x} \leftarrow \min(\overline{x},g)$ we have one of the following cases: (i) If $g < up(x)$ then $\overline{x}$ has slope unity on both sides. (ii) If $g = up(x)$ then $\overline{x}$ has slope zero on the right-hand side and slope unity on the left-hand side. (iii) If $g > up(x)$ then $\overline{x}$ 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. Suppose $f_3 \leftarrow \min(f_1,f_2)$. Then the situation 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 $R_-(g)$ at 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 $\underline{a}$, $\overline{a}$, $\underline{b}$, $\overline{b}$ which are repeatedly updated and always satisfy $\underline{a} \leqq a \leqq \overline{a}$ and $\underline{b} \leqq b \leqq \overline{b}$. The initial values are $\underline{a} = \underline{b} = -\infty$ and $\overline{a} = \overline{b} = \infty$. The basic idea is to follow the known algorithm for evaluating R with g being indeterminate; however, g will always be confined

to $D = [\underline{a},\overline{a}] \cup [\underline{b},\overline{b}]$. Whenever the result of the following step depends on the value of g within D , a test which amounts to one R-evaluation is performed, in order to update D appropriately. The fundamental principle used here was first introduced in [M1] and later applied in [M2].

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 $\overline{a} = \overline{b}$; 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 available quantities 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 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:

$R(g') \geq 0$	(enter* Phase 2) $\overline{a} \leftarrow g'$; $\underline{b} \leftarrow g'$; $f_3 \leftarrow \min(f_1,f_2)$
$R(g') < 0$ and $R_-(g') \geq 0$	$\underline{a} \leftarrow g'$; $\underline{b} \leftarrow g'$; $f_3 \leftarrow f_2$
$R(g') < 0$ and $R_+(g') \leq 0$	$\overline{a} \leftarrow g'$; $\overline{b} \leftarrow g'$; $f_3 \leftarrow f_1$

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

* 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 $\overline{a} = \underline{b}$ and all the quantities consist of at most two linear pieces with the breakpoint occurring at $\overline{a} = \underline{b}$. 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 $[\underline{a},\overline{a}]$. 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:

$R(g') \geq 0$	$\overline{a} \leftarrow g'$; $f_3 \leftarrow f_1$
$R(g') < 0$ and $R_-(g) \geq 0$	$\underline{a} \leftarrow g'$; $f_3 \leftarrow f_2$
$R(g') < 0$ and $R_+(g) \leq 0$	$\overline{a} \leftarrow g'$; $f_3 \leftarrow f_1$

At the end we have $R(g)$ as a linear function over $[\underline{a},\overline{a}]$. It is then straightforward to decide which of the following is the case: (i) There is a unique solution to $R(g) = 0$ over $[\underline{a},\overline{a}]$; this solution is then assigned to a. (ii) $R(g) \geq 0$ for all $g \in [\underline{a},\overline{a}]$; this is possible only if $\underline{a} = -\infty$, in which case $a \leftarrow -\infty$. (iii) $R(g) < 0$ for all $g \in [\underline{a},\overline{a}]$; this is possible only if $\overline{a} = \infty$, in which case $a \leftarrow \infty$ and $R(g) < 0$ for every real g. 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 [M1] for a more detailed discussion of this point).

6. FINDING xlow AND xhigh

When we solve $r(g) \leq g \leq r'(g)$ 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 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 find all the breakpoints of the maximum function in $O(m_z \log m_z)$ time (see the Appendix of [M1]). Thus, the set of all breakpoints produced during one iteration can be found and sorted in $O(m \log m)$ time. Assuming that these breakpoints are $g_1 \leq \ldots \leq g_q$ ($q = O(m)$), we may perform a binary search over these q values which amounts to testing only $O(\log q)$ of them. If this happens 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 xlow and xhigh 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 xlow and xhigh are found for all variables x in $O(mn^3 \log m)$ time.

7. SOLVING S

Let ylow and yhigh denote the bounds obtained in the previous section. The following routine (which was essentially given by Aspvall and Shiloach [AS]) either discovers that S is infeasible, or else produces a feasible solution

$(x_j = x_j^* , j=1,\ldots,n)$:

```
procedure  FINAL ;
  begin
    for each variable  x  [ x̄ ← xhigh ; x ← xlow ] ;
    for  j ← 1  until  n  do
      begin
        for  i ← 1  until  n  do
          begin
            for each  y  and  (ay+bz ≤ c)  FORWARD( y , ay+bz ≤ c ) ;
          end
        if there is a finite  ξ  such that  x_j ≤ ξ ≤ x̄_j  then  [ x_j ← ξ ;
        x*_j ← ξ ; x̄_j ← ξ ] else return (INFEASIBLE) ;
      end
    return  ( x_j = x*_j , j=1,...,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 $xlow \leq x \leq xhigh$, 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 xlow and xhigh for all the variables. The genuinely polynomial algorithm hence runs in $O(mn^3 \log m)$ operations.

REFERENCES

[A] B. Aspvall, "Efficient Algorithms for Certain Satisfiability and linear Programming Problems", Ph.D. dissertation, Department of Computer Science, Stanford University, August 1980.

[AS] B. Aspvall and Y. Shiloach, "A Polynomial Time Algorithm for Solving Systems of Linear Inequalities with Two Variables per Inequality", SIAM J. Comput. 9 (1980) 827-845.

[EK] J. Edmonds and R.M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", J. ACM 19 (1972) 248-264.

[K] L.G. Khachiyan, "A Polynomial Algorithm in Linear Programming", Soviet Math. Dokl. 20 (1979) 191-194.

[M1] N. Megiddo, "Combinatorial Optimization with Rational Objective Functions", Math. of OR 4 (1979) 414-424.

[M2] N. Megiddo, "Applying Parallel Computation Algorithms in the Design of Serial Algorithms", Proceedings of the IEEE 22nd Symposium on Foundations of Computer Science (1981),

[N] C.G. Nelson, "An $n^{\log n}$ Algorithm for the Two-Variable-per-Constraint Linear Programming Satisfiability Problem", Technical Report AIM-319, Department of Computer Science, Stanford University, 1978.

[P] V.R. Pratt, "Two Easy Theories whose Combination Is Hard", unpublished manuscript, 1977.

[S] R. Shostak, "Deciding Linear Inequalities by Computing Loop Residues", J. ACM , to appear.