Linear Programming with Two Variables per Inequality in Poly-Log Time*

George S. Lueker† Nimrod Megiddo‡ Vijaya Ramachandran§

Abstract. The parallel time complexity of the linear programming problem with at most two variables per inequality is discussed. Let n and m denote the number of variables and the number of inequalities, respectively, in a linear programming problem. We assume all inequalities are weak. We describe an $O((\log m + \log^2 n)\log^2 n)$-time parallel algorithm for deciding feasibility, under the concurrent-read-exclusive-write PRAM model. It requires $mn^{O(\log n)}$ processors in the worst case, though we do not know whether this bound is tight. When the problem is feasible a solution can be computed within the same complexity. Moreover, linear programming problems with at most two nonzero coefficients in the objective function can be solved in poly-log time on a similar number of processors. Consequently, all these problems can be solved sequentially with only $O((\log m + \log^2 n)^2 \log^2 n)$ space. (These bounds assume that numbers take $O(1)$ space, and arithmetic on them takes $O(1)$ time; the problem can still be solved in poly-log space as a function of the input size even if we instead use a Turing machine model with rational input.) It is also shown that if the underlying graph has bounded tree-width and an underlying tree is given then the feasibility problem is in the class NC.

1. Introduction

Dobkin, Lipton and Reiss [7] first showed that the general linear programming problem was (log-space) hard for P. Combined with Khachiyan's deep result [14] that the problem is in P, this establishes that the problem is P-complete (that is, log-space complete for

*An earlier version of this paper appeared in the *Proceedings of the 18th Annual Symposium on Theory of Computing*, May 1986, pp. 196–205.

†Department of Information and Computer Science, University of California at Irvine, Irvine, CA 92717. Supported by National Science Foundation Grant DCR-8509667.

‡IBM Research, Almaden Research Center, 650 Harry Road, San Jose, CA 95120 and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel

§Department of Computer Science, University of Texas at Austin, Austin, TX 78712. Supported by NSF under Grant ECS-8404866, by Joint Services Electronics Program under Grant N00014-84-CO149 and by an IBM Faculty Development award.

P). A popular specialization of the general linear programming problem is the problem of solving linear inequalities with at most two variables per inequality (see [15] and the references thereof). It is shown in [15] that a system of m linear inequalities in n variables (but at most two nonzero coefficients per inequality) can be solved in $O(mn^3 \log m)$ arithmetic operations and comparisons over any ordered field. It is not known whether the general problem (even only over the rationals) can be solved in less than $p(m,n)$ operations, for any polynomial p.

Throughout the paper we assume that the space to store numbers and the time for arithmetic operations is $O(1)$. Since each expression we compute can be written as an expression tree of height $O(\log n)$ in the input values, the length of numbers only increases by a polynomial factor during the execution of the algorithm, so this assumption does not alter the statements of our results by more than a polynomial factor for the number of processors, or more than a poly-log factor for the time.

In this paper we are interested in the *parallel* computational complexity of the two variables per inequality problem. We first mention some related results which, we hope, shed some light on the parallel complexity of the problem.

Proposition 1.1. *The problem of finding the minimum value of a general linear function subject to linear inequalities with at most two variables per inequality is P-complete.*

Proof: The proof follows from the result that the problem of finding the *value* of the maximum flow through a capacitated network is P-complete [11]. More specifically, every maximum flow problem can be reduced [8] to a transportation problem, that is, a problem of the form

$$\begin{aligned} \text{Minimize} \quad & \sum_{ij} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j} x_{ij} \leq a_i \\ & \sum_{i} x_{ij} \geq b_j \\ & x_{ij} \geq 0 \ . \end{aligned}$$

The dual of the latter has only two variables per inequality. ∎

In fact, there exist much simpler but yet P-complete linear programming problems with a general objective function and only two variables per inequality. For example, consider the following recursive formula

$$x_{k+1} = \max(\alpha_k x_k + \beta_k \ , \ \gamma_k x_k + \delta_k)$$

where α_k, β_k, γ_k and δ_k $(k = 1, \ldots, n)$ are given numbers. Given any value of x_1, we want to determine the resulting value of x_n. Let us call this the *PROPAGATION* problem.

The propagation problem can be solved as the following linear programming problem:

$$\begin{aligned} \text{Minimize} \quad & \sum_i M^i x_i \\ \text{subject to} \quad & x_{k+1} \geq \alpha_k x_k + \beta_k \\ & x_{k+1} \geq \gamma_k x_k + \delta_k \ . \end{aligned}$$

where M is sufficiently large.

Proposition 1.2. *PROPAGATION is P-complete.*

Proof: The proposition follows from the result by Helmbold and Mayr [12] that the 2-processor list scheduling problem is P-complete. The latter amounts to a special case of *PROPAGATION*: $x_k = |x_{k-1} - T_k| = \max\{x_{k-1} - T_k, -x_{k-1} + T_k\}$, $(k = 1, \ldots, n)$, where $T_1, \ldots, T_n$ are given integers and $x_0 = 0$. ∎

Interestingly, the subproblem of *PROPAGATION* where all the α_k's and γ_k's are nonnegative is in NC; this follows immediately from the parallel composition of monotonic piecewise linear functions to be discussed in Section 3.

We discuss three different problems:

(i) **Deciding feasibility.** Here we only need to determine whether there exists a solution to a given set of linear inequalities with at most two variables per inequality.
(ii) **Solving inequalities.** Here we require that if the system is feasible then some point in the feasible space be determined.
(iii) **Optimization.** Here we seek to find a point in the feasible space which maximizes a given linear combination of two of the variables. As in the previous case, one has to distinguish here between the problem of computing the optimum value of the function and the problem of computing optimum values of the variables.

Deciding feasibility is the core of our algorithm. This is accomplished by computing the *projections* of the set of feasible solutions onto the individual coordinate axes. Using a parallelization of the sequential algorithm of [17], we will compute for each variable x an interval $[x_{\text{low}}, x_{\text{high}}]$, $-\infty \leq x_{\text{low}} \leq x_{\text{high}} \leq \infty$, (possibly empty) so that for every $x' \in [x_{\text{low}}, x_{\text{high}}]$ there is a solution to the system of inequalities with $x = x'$. This part of the algorithm suffices of course for determining whether a given system of linear inequalities (with at most two variables per inequality) has a solution. We shall later discuss the problem of finding a feasible solution given the (nonempty) projections onto the axes. Finally, we show how to optimize a linear function with at most two nonzero coefficients subject to such systems of inequalities.

2. Preliminaries

Two characterizations of feasibility of linear inequalities with at most two variables per inequality were given by Nelson [17] and Shostak [19]. Our algorithm is a parallelization of the algorithm in Nelson [17], but our exposition will also make use of the characterization of Shostak [19] and further results by Aspvall and Shiloach [2] which we now describe.[1] A key idea in these papers is the observation that one can combine inequalities to deduce new inequalities. For example, suppose we are given $x + 2y \leq 3$ and $-y + 3z \leq 4$. Eliminating y between these two inequalities, we obtain $x + 6z \leq 11$.

It is convenient to discuss the problem using graph-theoretic terminology. We form what is called the *constraints graph*, denoted G, by creating a vertex for each variable, and an edge for each inequality whose endpoints are the two variables appearing in the inequality. (Note that this is actually a multigraph, since a pair of variables can be involved in many different inequalities.) We henceforth identify variables with the corresponding vertices. To take care of inequalities involving only a single variable, such as $x \leq 4$, [19] also added a dummy vertex v_0 corresponding to a new variable which can only occur with coefficient 0; an edge representing an inequality involving only one variable, say x, runs between x and v_0. Now let L denote a path in G from x to y, that is, a set of inequalities $\alpha_i x_i + \beta_{i+1} x_{i+1} \leq \gamma_i$, $(i = 0, 1, \ldots, k-1)$, where $x = x_0$ and $y = x_k$. (Note that since G is a multigraph, a path should be thought of as a sequence of edges rather than as a sequence of vertices, to avoid ambiguity.) If $\alpha_i \beta_i < 0$ for $i = 1, \ldots, k-1$, we say P is *admissible*, and then by successive elimination of $x_1, x_2, \ldots, x_{k-1}$ one can deduce a new inequality $\alpha x + \beta y \leq \gamma$ that is implied by this path. We will call this the *resultant* of the path. If on the other hand there exists an i $(1 \leq i \leq k-1)$ such that $\alpha_i \beta_i > 0$, we say the path is inadmissible, and then one cannot deduce a resultant.

The case in which the endpoints of the path are the same variable deserves special attention; in this case we call the path a *loop*. Then the resultant involves only a single variable, that is, it is a bound of the form $\alpha x \leq \gamma$. It is possible that the resultant could be an inherent contradiction—this occurs if $\alpha = 0$ and $\gamma < 0$. For instance, the path $-x + 2y \leq -7$ and $-y + x/2 \leq 2$ has the resultant $0x \leq -3$, which is clearly a contradiction. In this case we can deduce that there is no feasible solution. Unfortunately the converse is not true; it is possible to produce an example of an infeasible set of inequalities for which no loop yields a contradiction. One further idea is needed to produce a characterization.

The *closure* G' of G is the multigraph we obtain by adding to G all of the bounds which are resultants of simple cycles.

[1] The correctness of our feasibility test follows directly from the theorems of [17], but the presentation we have chosen enables us to depend only on results appearing in journals.

Theorem 2.1 (Shostak [19]) : *The original system G is infeasible if and only if some simple loop in the closure G' has a resultant which is a contradiction.*

3. Operations on polygons

For parallel computation we will want to be able to manipulate many bounds simultaneously. To discuss this approach it is convenient to represent a set of inequalities involving x and y by a *polygon* P_{xy} consisting of all points in the plane which satisfy all of the inequalities. (We call these objects polygons even though they may sometimes be unbounded.) It is useful to have a term to indicate that membership in a polygon implies some inequality; thus we will say that a polygon P *incorporates* an inequality if all points in the polygon satisfy the inequality. The algorithm of Nelson [17] made use of two operations on polygons, namely intersection and composition. In this section we define these polygons and operations, and indicate how we do them efficiently in parallel. (We have not carefully optimized these algorithms since this would not significantly improve the statement of the overall complexity of our feasibility testing algorithm.) Let R denote the set of reals, and let R^{xy}, for any two variables x and y, denote the two-dimensional space $R \times R$ given by the cross product of the x and y axes.

Suppose S consists of m inequalities in n variables. For $1 \leq k \leq n$, let Q^k_{xy} denote the polygon corresponding to the set of all inequalities which involve no variables other than x and y, and which are resultants of paths (not necessarily simple) of length at most k. Note that Q^1_{xy} is the polygon determined by the original inequalities involving only x and/or y. In particular, the polygon Q^1_{xy} incorporates all of the inequalities corresponding to edges among v_0, x, and y; thus we do not need the vertex v_0 in our graphs. Clearly the polygon Q^k_{xx} incorporates the resultants of all cycles of length at most k which involve x.

The algorithm uses two basic operations on convex polygons, namely, intersection and functional composition. The *intersection* of two polygons is the ordinary set intersection of their point sets, i.e.,

$$P_{xy} \cap P'_{xy} = \{(x,y) \mid (x,y) \in P_{xy} \text{ and } (x,y) \in P'_{xy}\}.$$

The *composition* P_{zx} of two polygons P_{zy} and P_{yx} is

$$P_{zx} = P_{zy} \circ P_{yx} = \{(z,x) \mid \exists y \in R \text{ such that } (z,y) \in P_{zy} \text{ and } (y,x) \in P_{yx}\}.$$

In other words, as noted in [17], $P_{zy} \circ P_{yx}$ is the projection onto R^{zx} of the intersection of the cylinders with bases P_{zy} and P_{yx}.

We now describe the implementation of these two operations. We represent convex polygons by a domain D and two bounds L and H; D is a (possibly infinite) interval,

and $L(x)$ (respectively $H(x)$) is a convex (respectively concave) piecewise linear function. The set of points in the represented polygon P_{yx} is

$$P_{yx} = \{(y, x) \mid x \in D \text{ and } L(x) \leq y \leq H(x)\};$$

for the representation to be considered valid we require that

$$x \in D \implies L(x) \leq H(x).$$

For simplicity we first discuss the basic operations as applied to piecewise linear convex functions rather than polygons. Thus, we first consider functions of the form $y = f(x) = \max_{1 \leq i \leq N}\{\alpha_i x + \beta_i\}$, which we represent by a list of pairs (α_i, β_i) ordered such that $\alpha_i < \alpha_j$ for $i < j$; we will require that there be no extraneous pairs, i.e., that no pair of values (α, β) appears more than once in the list, and that f coincides with each linear function $y = \alpha_i x + \beta_i$ over some interval of positive length.

Consider first the intersection problem. Given two functions $y = f_1(x)$ and $y = f_2(x)$ in the form described above, with N_1 and N_2 linear pieces, respectively, we have to compute the representation of $y = g(x) = \max\{f_1(x), f_2(x)\}$. This problem can be solved in $O(\log N)$ time with $O(N)$ processors, where $N = N_1 + N_2$. Here we briefly sketch the method. First note that we can convert between the representation discussed above and a list of the breakpoints (i.e., coordinates of points of discontinuity in the slope) of each function in constant time. Next, we merge the sets of breakpoints for f_1 and f_2 according to their x-coordinate, but keep track of whether each came from f_1 or f_2; call these respectively type 1 and type 2 breakpoints. The merging can be done efficiently by the algorithm of [4]. Next, using standard pointer doubling techniques, each type 1 (resp. type 2) point can determine the previous and following type 2 (resp. type 1) point. Once this information is available, each point can determine in $O(1)$ time whether it lies below or on $g(x)$. Finally, knowing the type of its neighbors, and whether they lie below or lie on $g(x)$, each point can determine in $O(1)$ time whether f_1 and f_2 intersect between it and its neighbor. Thus we can generate a list of all breakpoints of $g(x)$. This can then be converted back to the representation, as a list of linear functions, described above.

The second operation we need for our algorithm is functional composition. We first demonstrate this operation in a special case. Suppose $y = f(x)$ and $z = g(y)$ are strictly *monotone* piecewise linear functions, each represented as described above. We would like to compute the representation of the composition $z = h(x) = g(f(x))$. Suppose f and g consist of k and l linear pieces, respectively, and let $N = k + l$. The problem can be solved by $O(N)$ processors in $O(\log N)$ time as follows. Let $y_1, \ldots, y_{l-1}$ denote the breakpoints of g. These can be found in constant time from the representation of g. Let $t_i = f^{-1}(y_i)$, $i = 1, \ldots, l-1$. The t_i's can be computed in parallel in $O(\log k)$ time by a binary search. Let $x_1, \ldots, x_{k-1}$ denote the breakpoints of f. Now, the x_i's and t_i's can

be merged and then the linear pieces of h can be constructed as compositions of linear functions.

Obviously, if both f and g are increasing, or if both are decreasing, then h is increasing; otherwise, h is decreasing. As for convexity or concavity properties, it is easy to verify the following:

(i) If g is monotone increasing then h is convex if both f and g are convex, and h is concave if both f and g are concave.
(ii) If g is monotone decreasing then h is convex if f is concave and g is convex, and h is concave if f is convex and g is concave.

We now sketch the construction of P_{zx} with linearly many processors in the total number of edges in P_{yx} and P_{zy}. Let y_h denote the smallest value of y (with $y \in D_{zy}$) at which $H_{zy}(y)$ attains a maximum. Note that $H_{zy}(y)$ is increasing for $y \leq y_h$ (for $y \in D_{zy}$) and nonincreasing for $y \geq y_h$ (again, for $y \in D_{zy}$). The function $H_{zx}(x)$ maps x to the largest value of z such that there is y in $[L_{yx}(x), H_{yx}(x)]$ for which $y \in D_{zy}$ and $L_{zy}(y) \leq z \leq H_{zy}(y)$. Thus, if $x \in D_{yx}$ is such that $H_{yx}(x) \leq y_h$ (and $H_{yx}(x) \in D_{zy}$) then a least upper bound on z is obtained by setting y to $H_{yx}(x)$, that is, $H_{zx}(x) = H_{zy}(H_{yx}(x))$. On the other hand, if x is such that $L_{yx}(x) \geq y_h$, then a least upper bound on z is obtained by setting y to $L_{yx}(x)$. Finally, if x is such that $L_{yx}(x) \leq y_h \leq H_{yx}(x)$, then the least upper bound on z is found by setting y to y_h. Summarizing, we have

$$H_{zx}(x) = \begin{cases} H_{zy}(L_{yx}(x)) & \text{if } y_h \leq L_{yx}(x) \\ H_{zy}(y_h) & \text{if } L_{yx}(x) \leq y_h \leq H_{yx}(x) \\ H_{zy}(H_{yx}(x)) & \text{if } H_{yx}(x) \leq y_h. \end{cases}$$

Similarly, let y_l denote the smallest value of y at which $L_{zy}(y)$ attains a minimum. Then $L_{zy}(y)$ is decreasing for $y \leq y_l$ (with $y \in D_{zy}$) and nondecreasing for $y \geq y_l$ (with $y \in D_{zy}$). This implies that if $x \in D_{yx}$ is such that $H_{yx}(x) \leq y_l$, then a largest lower bound on z is obtained by picking y to be $H_{yx}(x)$, and if x is such that $L_{yx}(x) \geq y_l$, then a largest lower bound on z is obtained by picking y to be $L_{yx}(x)$. Finally, if $L_{yx}(x) \leq y_l \leq H_{yx}(x)$, then we pick $y = y_l$. Thus

$$L_{zx}(x) = \begin{cases} L_{zy}(L_{yx}(x)) & \text{if } y_l \leq L_{yx}(x) \\ L_{zy}(y_l) & \text{if } L_{yx}(x) \leq y_l \leq H_{yx}(x) \\ L_{zy}(H_{yx}(x)) & \text{if } H_{yx}(x) \leq y_l. \end{cases}$$

Obviously, there exist x_{hl} and x_{hh} such that $H_{yx}(x) \geq y_h$ iff $x \in [x_{hl}, x_{hh}]$. Analogously, there exist x_{ll} and x_{lh} such that $L_{yx}(x) \leq y_l$ iff $x \in [x_{ll}, x_{lh}]$. Note that the values of y_l, y_h, x_{hl}, x_{hh}, x_{ll} and x_{lh} can be found in $O(\log N)$ time. It follows that the representations of the functions $H_{zx}(x)$ and $L_{zx}(x)$ can be computed, each over at most three disjoint intervals of x, as compositions of monotone functions. Let us consider the

procedure UpdatePaths(Q);
comment Q is an array of polygons indexed by x and y;
begin
  for $i \leftarrow 1$ **to** $\lceil \lg n \rceil$ **do**
    for all variables z and x **pardo**
      $Q_{zx} \leftarrow Q_{zx} \cap \left(\bigcap_y Q_{zy} \circ Q_{yx} \right)$;
  POINTA: **comment** at this point the resultants of all paths
      are incorporated into the polygons;
  for all variables x **pardo**
    $Q_{xx} \leftarrow Q_{xx} \cap I_{xx}$;
end;

various types of breakpoints of the functions $H_{zx}(x)$ and $L_{zx}(x)$. Obviously, any such breakpoint is of one of the following types: (i) a breakpoint of one of the functions $H_{yx}(x)$ and $L_{yx}(x)$, (ii) an inverse image under one of these functions of a breakpoint of one of the functions $H_{zy}(y)$ and $L_{zy}(y)$, (iii) one of the points x_{hl}, x_{hh}, x_{ll} and x_{lh}. Each of the breakpoints of the functions H_{zy} (respectively L_{zy}) contributes at most two breakpoints to H_{zx} (respectively L_{zx}). Also, each of the breakpoints of the functions H_{yx} and L_{yx} contributes at most one breakpoint to each of the functions H_{zx} and L_{zx}. Thus, the total number of breakpoints of H_{zx} and L_{zx} is at most $2N + 4$, where N is the total number of breakpoints of H_{yx}, L_{yx}, H_{zy} and L_{zy}.

The new domain is given by

$$D_{zx} = \{x \mid x \in D_{yx} \text{ and } [L_{yx}(x), H_{yx}(x)] \cap D_{zy} \neq \emptyset\}$$

We omit the details showing how this can be computed within the stated resource bounds.

4. Deciding feasibility

Using the basic operations of intersection and decomposition, we can now sketch the algorithm for deciding feasibility of a given system of linear inequalities with at most two variables per inequality. The algorithm consists of two iterations of the procedure UpdatePaths, shown here. We define I_{xx} to be the *identity polygon*, i.e., $I = \{(x, x) \mid x \in R\}$. This algorithm for deciding feasibility begins by using an algorithm analogous to the standard parallel transitive closure or shortest path algorithms (see [13] for more information about such algorithms). It is interesting to note that the two operations $\cap$ and $\circ$ do *not* form a closed semiring in the sense defined in [1]. In particular, the distributivity condition $P_{zy} \circ (P_{yx} \cap P'_{yx}) = (P_{zy} \circ P_{yx}) \cap (P_{zy} \circ P'_{yx})$ *fails* to hold in

procedure CheckFeasibility(S,V);
comment S is the set of inequalities, and V is the set of variables;
begin
  for all variables x and y **pardo**
    $Q_{xy} \leftarrow$ the polygon determined by all inequalities
        involving no variables outside $\{x, y\}$;
  for all variables x **pardo**
    $Q_{xx} \leftarrow Q_{xx} \cap I_{xx}$;
  for $i \leftarrow 1$ **to** $2\lceil \lg n \rceil$ **do**
    for all variables x and z **pardo**
      $Q_{zx} \leftarrow Q_{zx} \cap \left(\bigcap_y Q_{zy} \circ Q_{yx} \right)$;
end;

general. (As an example, let P_{zy} be the entire zy plane, let P_{yx} be determined by the one inequality $y \leq -1$, and let P'_{yx} be determined by the one inequality $y \geq 1$. Then the left side is the empty set and the right side is the entire zx plane.) It is not hard to see though that we do have

$$P_{zy} \circ (P_{yx} \cap P'_{yx}) \subseteq (P_{zy} \circ P_{yx}) \cap (P_{zy} \circ P'_{yx}).$$

From this one can easily show that, if for each x and y we initialize Q_{xy} to be the set of all inequalities involving x and y, then at POINTA we will have $Q_{xy} \subseteq Q^n_{xy}$, where Q^n_{xy} is defined as in Section 3. (Note that while we do not claim equality, the Q_{xy} do contain the projection of the feasible space onto R^{xy} since each composition and intersection corresponds to valid deductions that can be made about the feasible space.) In particular, at the end of UpdatePaths each Q_{xx} will be a set of pairs (x, x) where each x obeys the constraints added to G in Section 2 to form the closure G'. Thus by Theorem 2.1, a second application of UpdatePaths will cause at least one of the Q_{xy} to become empty if the original set of inequalities had no feasible solution. Further, if the feasible space is nonempty, it follows from [2, Lemma 9] that after this second application of the procedure the projection of the feasible set onto any axis R^x is the same as the projection of Q_{xx} onto R^x.

A simplification of this description is possible: by initially restricting each Q_{xx} to be contained within I_{xx}, we eliminate the need for the last loop in UpdatePaths. In fact, then the entire feasibility checking procedure becomes the CheckFeasibility procedure shown here. This is nearly the same as the algorithm of [17], and another proof of correctness can be found there.

Let the total number of edges in all polygons constructed during the algorithm be E. As in the sequential algorithm of [17], the polygons Q_{xy} are computed in $O(\log n)$ stages,

and we have $E = mn^{O(\log n)}$. Intersection of n polygons can be computed by n parallel teams of processors in $O(\log n)$ phases, where in each phase each team is computing the intersection of two polygons. It is convenient to think here of a model of computation where the machine does not have to allocate all the processors in advance; it rather invokes processors as they are needed, just like a Turing machine using unlimited tape space. This allows us to talk about the "worst-case processor complexity." Assuming we have $O(E)$ processors, all pairwise intersections and compositions take $O(\log E)$ time. It follows that the entire procedure takes $O(\log E \ \log^2 n)$ time. Thus, the worst-case running time is $O((\log m + \log^2 n) \log^2 n)$.

It is interesting to consider the space complexity implied by our result. We have just established that we can determine feasibility in $T = O((\log m + \log^2 n) \log^2 n)$ parallel time using $P = mn^{O(\log n)}$ write parallel RAM. Using standard simulation relations between parallel models of computation and between parallel time and sequential space [3, 9, 13, 20], this implies that feasibility can be determined by a poly-log space-bounded deterministic Turing machine. This suggests that the problem might not be P-complete.

5. Computing a feasible solution

We now consider the problem of computing a feasible solution, given the projections of the (nonempty) feasible domain P onto the individual axes. Thus, let $[x_{\text{low}}, x_{\text{high}}]$ $(-\infty \leq x_{\text{low}} \leq x_{\text{high}} \leq \infty)$ denote the set of values of variable x that can be completed into a solution of the entire system S. If all the projections are *finite* intervals then a feasible solution is readily available:

Proposition 5.1. *If for every x both x_{low} and x_{high} are finite, then a feasible solution is obtained by setting each variable x to the arithmetic mean $\frac{1}{2}(x_{\text{low}} + x_{\text{high}})$.*

Proof: Suppose, to the contrary, that the vector of the arithmetic means $\frac{1}{2}(x_{\text{low}} + x_{\text{high}})$ is not feasible. Then there is an inequality $\alpha x + \beta y \leq \gamma$ which is violated. In other words,

$$\tfrac{1}{2}\alpha(x_{\text{low}} + x_{\text{high}}) + \tfrac{1}{2}\beta(y_{\text{low}} + y_{\text{high}}) > \gamma \ .$$

Consider the rectangle $[x_{\text{low}}, x_{\text{high}}] \times [y_{\text{low}}, y_{\text{high}}]$. By definition, each edge of this rectangle contains at least one point of the projection P_{xy} . However, we claim that this contradicts the inequality

$$\tfrac{1}{2}\alpha(x_{\text{low}} + x_{\text{high}}) + \tfrac{1}{2}\beta(y_{\text{low}} + y_{\text{high}}) > \gamma \ ,$$

since the center of the rectangle is in the convex hull of any set that intersects all four edges of the rectangle. The proof of this claim is easy. Let L, R, T and B denote points (not necessarily distinct) that lie on the left, right, top and bottom edges of the

rectangle, respectively. Consider the straight line determined by the points L and R. If the center lies on this line then we are done. Otherwise, if the center lies above the line then it is in the triangle determined by T together with L with R, and if it lies below this line then it is in the triangle determined by B together with L and R. ∎

Interestingly, Proposition 5.1 does not hold if there are more than two variables per inequality. To see this, consider the system $x \geq 0$, $y \geq 0$, $z \geq 0$, and $x + y + z \leq 1$. The projection of the feasible space onto the x-, y-, or z-axis is just $[0, 1]$, but the point $(1/2, 1/2, 1/2)$ is not feasible.

The unbounded case is handled as follows. We introduce to the system an additional variable ξ and the $2n$ inequalities $x_j \leq \xi$, $x_j \geq -\xi$ $(j = 1, \ldots, n)$. We find the projection of the augmented problem onto the ξ-axis. In other words, we compute an interval $I = [\xi_{\text{low}}, \xi_{\text{high}}]$ $(0 \leq \xi_{\text{low}} \leq \xi_{\text{high}} = \infty)$, such that for every $\xi \in I$ there exist values for $x_1, \ldots, x_n$ which solve the augmented problem. By setting ξ to any finite number in I we obtain a feasible system of linear inequalities (with at most two variables per inequality) whose set of solutions is bounded. Any solution of the latter yields a solution to the original problem simply by dropping ξ. Thus we have the following:

Proposition 5.2. *If a system of linear inequalities has a nonempty set of solutions, then a solution can be found in poly-log time with $mn^{O(\log n)}$ processors in the worst case.*

6. Optimization problems

We have already shown that, with a general objective function, the optimization problem with at most two variables per inequality is P-complete. In this section we discuss the case where the objective function also has at most two variables with nonzero coefficients.

Intuitively, the optimization problem can be solved by searching for the optimum value, using the feasibility checking algorithm as an "oracle". In the context of sequential computation this yields a polynomial-time (but not strongly polynomial-time) algorithm. In the context of parallel computation this approach does not provide a poly-log algorithm since the number of queries during the search is *linear* in the length of the binary representation of the input.

We can use here a technique presented in [16] to obtain a poly-log algorithm for finding optimum solutions over any ordered field. Here is a sketch of the method; see [16] for more detail. Suppose the problem is to minimize the function

$$f(x_1, \ldots, x_n) = c_1 x_1 + c_2 x_2$$

subject to a system S of linear inequalities in $x_1, \ldots, x_n$ with at most two variables per inequality. Consider the system S' of inequalities, which is obtained by adding to S

an inequality $c_1x_1 + c_2x_2 \leq \lambda$, where λ is a parameter. We need to find the smallest value of λ for which S' is feasible. Denote this optimum value by λ^*. We can run our parallel algorithm for deciding feasibility on S', handling λ as an indeterminate. Thus the "program variables" will be functions of λ rather than field elements. Throughout the execution of the algorithm we maintain an interval of values of λ, guaranteed to contain λ^*, over which the current program variables are all linear functions of λ. Comparisons between two functions of λ have to be resolved according to the function values at λ^*, which is itself not known. However, during each step of the algorithm, each processor that is unable to perform a comparison for which it is responsible simply reports the value of λ which is critical for that comparison, that is, a value λ' such that the comparison between the two functions can be resolved by comparing λ' and λ^*. The comparison between λ' and λ^* can be carried out by setting λ to λ' and checking feasibility of the system. Let p denote a bound on the number of processors required to check feasibility. For the parametric algorithm we can either use p^2 processors, in which case all the critical values of λ can be tested in parallel, or only p processors and run a binary search over the set of critical values. In the latter case we obtain a poly-log algorithm with $mn^{O(\log n)}$ processors for computing λ^* over any ordered field.

Once λ^* is known, we can *solve* the system S' with $\lambda = \lambda^*$.

7. Bounded tree-width

Robertson and Seymour [18] introduced the notion of the *tree-width* of a graph. This notion lends itself via the constraints graph to systems of linear inequalities with at most two variables per inequality.

Definition 7.1. A connected graph G is said to have *tree-width* less than or equal to k if there is a family $\mathbf{V} = \{V_1, \ldots, V_t\}$ of sets V_i of vertices of G with the following properties:

(i) Each V_i contains at most $k+1$ vertices of G.
(ii) For every edge e of G, there exists an i such that e has both its endpoints lying in V_i.
(iii) The intersection graph $\mathbf{T} = (\mathbf{V}, \mathbf{E})$, where $(V_i, V_j) \in \mathbf{E}$ if and only if $V_i \cap V_j \neq \emptyset$, is a tree.

We assume the graph is given together with such a tree and develop an algorithm that relies on the tree. Note that a tree with at most n nodes suffices. It will follow that if the tree-width is bounded then the number of edges remains polynomial in m and n during the execution of the special algorithm.

For our purpose here we may assume, without loss of generality, that our graphs are connected. Also, for simplicity of presentation, assume all the sets V_i are $(k+1)$-cliques

in G; this assumption is also made without loss of generality since redundant inequalities can always be added to the system.

Proposition 7.2. *Suppose U, V, and W are nodes of $\mathbf{T}$ such that V lies on the path connecting U and W. Let $u \in U$ and $w \in W$ be vertices of G. Then on any path in G connecting u and w there is at least one vertex $v \in V$.*

Proof: Consider any such path $u = v_1, \ldots, v_r = w$. Let $V_0 = U$ and $V_r = W$. For every i, $i = 0, \ldots, r-1$, there is a set $V_i \in \mathbf{T}$ such that both v_i and v_{i+1} are in V_i. By definition each (V_i, V_{i+1}) is an arc in $\mathbf{T}$ (if $V_i \neq V_{i+1}$). Thus, $U = V_0, V_1, \ldots, V_{r-1}, V_r = W$ yields a path in $\mathbf{T}$. It follows that one of the V_i's equals V. This implies that one of the v_i's is in V. ∎

Given the underlying tree $\mathbf{T}$, we can decompose the graph G in an efficient way. The decomposition is based on the *centroid* which is often useful in the design of parallel algorithms (as an early reference we might mention [5]). The centroid of a tree T with N nodes is a node c so that there exist two subtrees T_1, T_2 rooted at c (and also c is the only common node), each with no more than $\frac{2}{3}N + 1$ nodes, whose union is T. The *centroid decomposition* of a tree is the iterated partitioning of a tree in this way into two subtrees rooted at the centroid. This decomposition is obtained in $O(\log N)$ iterations, and moreover, it can be computed in poly-log time with a polynomial number of processors.

In view of Proposition 7.2 the centroid decomposition of $\mathbf{T}$ induces a decomposition of G as follows. At the first level of the decomposition we have a set C of $k+1$ vertices of G and two induced subgraphs G_1, G_2, whose vertex sets intersect at C and cover all the vertices of G. Moreover, every edge of G is contained in one of these two graphs. The decomposition is iterated until all the subgraphs consist of not more than $k+1$ vertices. It follows that this decomposition has only $O(\log n)$ levels.

Given the decomposition of G, we produce polygons $Q_{xy}(G)$ as follows. The polygon Q_{xy} computed will incorporate all of the resultants of *simple* paths from x to y. (Recall that a simple path can begin and end at the same point, so x and y may be equal.) Let G_1, G_2, and C be as explained above. We state the algorithm recursively. Thus, assume we have computed polygons $Q_{xy}(G_i)$ for all pairs of vertices $x, y \in G_i$ $(i = 1, 2)$. In particular, if $x, y \in C$ then we have for them *two* polygons $Q_{xy}(G_1)$ and $Q_{xy}(G_2)$.

The recursive step is performed as follows. Let x and y be any two vertices of G for which we compute $Q_{xy}(G)$. For simplicity of notation assume without loss of generality that $x \in G_1$. Any simple path from x to y can be represented as a union of paths $\pi(z_0, z_1), \pi(z_1, z_2), \ldots, \pi(z_{l-1}, z_l)$ where $z_0 = x$, $z_l = y$, $z_i \in C$ for $i = 1, \ldots, l-1$, and $\pi(z, z')$ denotes some simple path from z to z'. Moreover, paths of the form $\pi(z_{2i}, z_{2i+1})$ stay entirely within G_1 while paths of the form $\pi(z_{2i-1}, z_{2i})$ stay entirely within G_2. Thus,

to incorporate the resultants of all simple paths connecting x and y in G, it suffices to intersect all the polygons obtained by compositions of the form

$$Q_{xz_1}(G_1) \circ Q_{z_1 z_2}(G_2) \circ Q_{z_2 z_3}(G_1) \circ \cdots \circ Q_{z_{l-1} y}(G_i)$$

(where $y \in G_i$), so that the z_j's ($1 \leq j \leq l-1$) are pairwise distinct points in C. The number of different choices of the z_j's implies that for each pair x, y, the number of polygons intersected this way is bounded by a constant K depending only on k. For each pair x and y, each composition is of at most $k+2$ polygons. This may multiply the number of breakpoints by at most $O(k)$, since composition of k polygons can be computed in $O(\log k)$ compositions of two polygons (where the number of breakpoints is at most approximately doubled). Since the entire process runs in $O(\log n)$ stages, and there are m inequalities at the beginning, it follows that the number of edges in each of the generated polygons is $m\,(kK)^{O(\log n)}$. This is the same as $mn^{g(k)}$ for some $g(k)$ depending only on k; hence it is polynomial in m and n for any fixed k. The running time on a suitable number of processors is $O(\log^2 n \log m)$ with a coefficient that depends on k. By Theorem 2.1 this algorithm can determine feasibility.

8. Directions for further work

It is interesting to ask whether the algorithms we have described in Sections 4, 5, and 6 can ever in fact require more than polynomially many processors. This is essentially the same as the question asked in [17] of whether the algorithm of [17] can require more than polynomial time.

More generally, resolving whether the linear programming problem with two variables per inequality lies in NC seems like a very interesting question. To provide context, note that Cook fairly recently observed [6, p. 18] "I find it interesting that very few natural problems in [poly-log space] have come to my attention which are not in NC. One notable exception is the problem of determining whether two groups, presented by their multiplication tables, are isomorphic. .... I know of no NC solution to this problem, or even any polynomial time solution." Thus the present status of linear programming with two variables per inequality seems to be rather unusual, particularly since it is known to be solvable in polynomial time (even if we allow that inputs are arbitrary reals and the time bound must be independent of these values [15]).

Acknowledgment. This work was done while the authors were at Mathematical Sciences Research Institute, Berkeley, California.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] 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.

[3] A. Borodin, "On relating time and space to size and depth" *SIAM J. Comput.* **6** (1977) 733–744.

[4] A. Borodin and J. E. Hopcroft, "Routing, merging, and sorting on parallel models of computation," *J. Comput. System Sci.* **30** (1985) 130–145.

[5] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *J. Assoc. Comput. Mach.* **21** (1974) 201–206.

[6] S. A. Cook, "A taxonomy of problems with fast parallel algorithms," *Information and Control* **64** (1985), pp. 2–22.

[7] D. Dobkin, R. J. Lipton and S. Reiss, "Linear programming is log space hard for P," *Information Processing Letters* **8** (1979) 96–97.

[8] L. R. Ford, Jr., and R. D. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[9] S. Fortune and J. Wyllie, "Parallelism in random access machines," *Proc. 10th Annual ACM Symposium on Theory of Computing*, 1978, pp. 114–118.

[10] L. M. Goldschlager, "Synchronous parallel computation," Technical Report No. 114, Department of Computer Science, University of Toronto, December 1977.

[11] L. M. Goldschlager, R. A. Shaw and J. Staples, "The maximum flow problem is log space complete for P," *Theoretical Computer Science* **21** (1982) 105–111.

[12] D. Helmbold and E. Mayr, "Fast scheduling problems on parallel computers," Report No. STAN-CS-84-1025, Computer Science Department, Stanford University, 1984.

[13] R. M. Karp and V. Ramachandran, "Parallel algorithms for shared memory machines," in: *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., North-Holland, 1988, to appear.

[14] L. G. Khachiyan, "A polynomial algorithm in linear programming," *Soviet Math. Dokl.* **20** (1979) 191–194.

[15] N. Megiddo, "Towards a genuinely polynomial algorithm for linear programming," *SIAM Journal on Computing* **12** (1983) 347–353.

[16] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *J. Assoc. Comput. Mach.* **30** (1983) 852–865.

[17] C. G. Nelson, An $n^{O(\log n)}$ algorithm for the two two-variable-per-constraint linear programming satisfiability problem," Report No. STAN-CS-78-689, Department of Computer Science, Stanford University, November 1978.

[18] N. Robertson and P. D. Seymour, "Graph width and well-quasi-ordering: a survey," *Progress in Graph Theory*, Academic Press Canada, 1984, pp. 399–406.

[19] R. Shostak, "Deciding linear inequalities by computing loop residues," *J. Assoc. Comput. Mach.* **28** (1981) 769–779.

[20] L. Stockmeyer and U. Vishkin, "Simulation of parallel random access machines by circuits," *SIAM J. Comput.* **13** (1984) 409–422.