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 describe an $O((\log m + \log^2 n)\log^2 n)$ time parallel algorithm under the concurrent-read-exclusive-write PRAM model for deciding feasibility. 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 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. It is also shown that if the underlying graph has bounded tree-width and an underlying tree is given then the problem is in the class NC.

1. Introduction

Dobkin, Lipton and Reiss [4] first showed that the general linear programming problem was (log-space) hard for P. Combined with Khachiyan's result [10] that the problem is in P, this establishes that the problem is P-complete (that is, log-space complete for 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 [11] and the references thereof). It is shown in [11] that a system of m linear inequalities in n variables (but at most two nonzero coefficients per inequality) can be solved in $O(mn^3 \log n)$ 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 $p(m,n)$

*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

‡Coordinated Science Laboratory, University of Illinois at Urbana-Champaign 1101 West Springfield, Urbana, IL 61801. Supported by NSF under Grant ECS-8404866, by Joint Services Electronics Program under Grant N00014-84-CO149 and by an IBM Faculty Development award.

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 [7]. More specifically, every maximum flow problem can be reduced [5] 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 $(k = 1, ..., n)$ 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:

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

where M is sufficiently large.

Proposition 1.2. *PROPAGATION is P-complete.*

Proof: The proof follows from a result by Helmbold and Mayr [8] that the 2-processor list scheduling problem is P-complete. The latter amounts to a special case of *PROPAGATION*: $x_k = |x_{k-1} - T_k|$ $(k = 1, \cdots, n)$, where $T_1, \cdots, T_n$ 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 from more general results we prove in this paper.

We discuss three different problems

(i) **Deciding feasibility.** Here we only need to recognize 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 solution will be computed.
(iii) **Optimizing a linear function with at most two nonzero coefficients.** As in the previous case, one has to distinguish here between the problem of computing the optimal value of the function and the problem of computing optimal 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 on the individual coordinate axes. Thus, we will compute for each variable x an interval $[xlow, xhigh]$, $-\infty \leq xlow \leq xhigh \leq \infty$, (possibly empty) so that for every $x' \in [xlow, xhigh]$ 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 on 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 [13] and Shostak [15]. Our algorithm is essentially a parallelization of the algorithm in Nelson [13], but our exposition will also make use of the characterization of Shostak [15] and further results by Aspvall and Shiloach [1] which we now describe. Given some bounds on a variable x and an inequality $\alpha x + \beta y \leq \gamma$ (where α and β are nonzero), some bound on y can obviously be derived, depending on the signs of α and β. If bounds on y were available before, we keep the tightest available. Let us call the routine that updates such bounds FORWARD (see [11]). It is convenient

to discuss the problem using graph-theoretic terminology. Thus, we identify variables with vertices, and inequalities with edges of a graph, which we call the *constraints graph*. This graph may have multiple edges and loops. The routine FORWARD can be applied along paths of the graph, using bounds on the vertex at one end of the path for updating bounds on the vertex at the other end. Similarly, if we apply FORWARD around a cycle, starting and ending at some vertex x (using x as an indeterminate), then we may obtain some bounds on x, possibly thereby proving infeasibility.

Suppose we apply FORWARD along all simple paths and around all simple cycles. This procedure may discover that the system is infeasible, but in any case it computes upper and lower bounds on different variables which are implied by the system. By adding all these bounds to the system as explicit inequalities, we obtain the "closure" of the system.

Shostak's theorem states that the system is feasible if and only if by applying the same procedure to the *closure* (that is, applying FORWARD along all simple paths and cycles) no infeasibility is detected.

For parallel computation we need to consider, as in [13], the implied bounds on one variable as *functions* of the values assigned to another variable. A more precise statement is given in the following proposition:

Proposition 2.1. *Let L denote a simple path from x to y, that is, a set of inequalities $\alpha_i x_i + \beta_{i+1} x_{i+1} \leq \gamma_i$, $(i = 0, 1, \cdots, k-1)$, with nonzero coefficients, where $x = x_0$ and $y = x_k$, and the variables are pairwise distinct. Let $P(L) \subseteq R^{k+1}$ denote the set of solutions to the system L of inequalities. Let $P_{xy}(L)$ denote the projection of $P(L)$ on the subspace R^{xy} of x and y. Under these conditions,*

(i) *if $\alpha_i \beta_i < 0$ for $i = 1, \cdots, k-1$, then a single inequality $\alpha x + \beta y \leq \gamma$ is implied by this path, that is, $P_{xy}(L)$ is a half-plane.*
(ii) *if $k \geq 2$ and $\alpha_i \beta_i > 0$ for at least one i $(1 \leq i \leq k-1)$, then $P_{xy}(L)$ equals the entire plane R^{xy}.*

Proof: The proof goes by induction on k. The case $k = 1$ is trivial. The inductive step is essentially the same as the case $k = 2$ which is straightforward. ∎

3. Operations on polygons

The algorithm of Nelson [13] made use of two operations on polygons, namely intersection and composition. In this section we describe the role played by these polygons and operations, and indicate how we do them efficiently in parallel.

Given a general system S of inequalities (with at most two variables per inequality), we would like to compute the projection $P_{xy}(S)$ of the set of solutions $P(S)$ on a two-dimensional subspace R^{xy} corresponding to any two variables x, y. Obviously, $P_{xy}(S)$ is a two-dimensional polyhedron, that is, an intersection of a finite number of half-planes. We call these objects polygons even though they may sometimes be unbounded. These polygons will be computed by considering subsystems of S corresponding to paths of some bounded length.

Suppose S consists of m inequalities in n variables. For any k, $2 \leq k \leq n$, and any two variables x, y, let Q^k_{xy} denote a polygon as follows. Suppose we set x to some value and then repeatedly update bounds on variables along paths (not necessarily simple) of length at most k, until bounds on y are derived. Thus for each x we obtain a certain interval (possibly empty) of feasible values of y. This mapping of x to intervals of y can naturally be represented by a convex polygonal region in the R^{xy} plane, which we denote by Q^k_{xy} . In particular, we allow $y = x$, in which case the polygon Q^k_{xx} represents tightest bounds obtained on x, as a function of the value of x that is sent around a cycle of length at most k. We compute only polygons of the form $Q^{2^i}_{xy}$ $(i = 0, 1, \cdots, \log n)$. This is carried out in $O(\log n)$ parallel steps, where a single step amounts to computing all the polygons Q^{2k}_{xy}, given all the polygons Q^k_{xy} .

The algorithm uses two basic operations on convex polygons, namely, *intersection* and *functional composition*. We now describe these two operations. We represent convex polygons by at most two relations $L(x) \leq y \leq H(x)$, where $L(x)$ and $H(x)$ are, respectively, convex and concave piecewise linear functions. 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 $y = f(x) = \max_{1 \leq i \leq N}\{\alpha_i x + \beta_i\}$. Furthermore, let us assume that $\alpha_i < \alpha_j$ for $i < j$, and 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. 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 $g(x)$.

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 by a sequence of linear functions sorted by the slope. 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 $y_1, \cdots, y_{l-1}$ denote the breakpoints of g. These can be found in constant time from the representation of g. Let $t_i = f^{-1}(y_i)$, $i = 1, \cdots, l-1$. The t_i's can be computed in parallel in $O(\log k)$ time by a binary search. Let $x_1, \cdots, x_{k-1}$ denote the breakpoints of f. Now, the x_i's and t_j'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.

Suppose a polygon P_{yx} is represented by $L_{yx}(x) \leq y \leq H_{yx}(x)$, where $L_{yx}(x)$ and $H_{yx}(x)$ are piecewise linear convex and concave functions, respectively. Without loss of generality, let us restrict the domains of the functions L_{yx} and H_{yx} to the set of x's for which $L_{yx}(x) \leq H_{yx}(x)$. Obviously, such a representation gives us for every x the precise range of values of y, $[L_{yx}(x), H_{yx}(x)]$, for which (x, y) solves the system of inequalities corresponding to P_{yx}. Similarly, suppose P_{zy} is another polygon, represented by $L_{zy}(y) \leq z \leq H_{zy}(y)$ so for every y the range of values of z for which (y, z) solves the system corresponding to P_{zy} is precisely $[L_{zy}(y), H_{zy}(y)]$. Consider both P_{yx} and P_{zy} as systems of inequalities in all the three variables x, y, z. The *composition* of P_{yx} with P_{zy} is a polygon $P_{zx} = P_{zy} \circ P_{yx}$, represented in the form $L_{zx}(x) \leq z \leq H_{zx}(x)$ so that for any x, $[L_{zx}(x), H_{zx}(x)]$ is precisely the range of values of z for which there is y so that (x, y, z) solves both P_{yx} and P_{zy}. In other words, as noted in [13], P_{zx} is the projection on R^{xz} of the intersection of the cylinders with bases P_{yx} and P_{zy}.

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 ($-\infty \leq y \leq \infty$) at which $H_{zy}(y)$ attains a maximum subject to $L_{zy}(y) \leq H_{zy}(y)$. Note that $H_{zy}(y)$ is increasing for $y \leq y_h$ (in the restricted domain where $L_{zy}(y) \leq H_{zy}(y)$) and nonincreasing for $y \geq y_h$ in that domain. 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 $L_{yz}(y) \leq z \leq H_{zy}(y)$. Thus, if x is such that

$H_{yx}(x) \leq y_h$ 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 $L_{yx}(x)$, that is, $H_{zx}(x) = H_{zy}(L_{yx}(x))$. Finally, if x is such that $L_{yx}(x) \leq y_h \leq H_{yx}(x)$, then $H_{zx}(x) = H_{zy}(y_h)$. Analogously, let y_l denote the smallest value of y at which $L_{zy}(y)$ attains a minimum. Then $L_{zy}(y)$ is decreasing for $y \leq y_l$ in the restricted domain and nondecreasing for $y \geq y_l$ in that domain. This implies that if x is such that $H_{yx}(x) \leq y_l$, then a largest lower bound on z is obtained by setting y to $H_{yx}(x)$, that is, $L_{zx}(z) = L_{zy}(H_{yx}(x))$, and if x is such that $L_{yx}(x) \geq y_l$, then a largest lower bound on z is obtained by setting y to $L_{yx}(x)$, that is, $L_{zx}(x) = L_{zy}(L_{yx}(x))$. Finally, if $L_{yx}(x) \leq y_l \leq H_{yx}(x)$, then $L_{zx}(x) = L_{zy}(y_l)$.

Obviously, there exist x_{hl} and x_{hh} such that $H_{yx}(x) < y_h$ for $x < x_{hl}$ and $x > x_{hh}$, and $H_{yx}(x) \geq y_h$ for $x_{hl} \leq x \leq x_{hh}$. Analogously, there exist x_{ll} and x_{lh} such that $L_{yx}(x) > y_l$ for $x < x_{ll}$ and $x > x_{lh}$ and $L_{yx}(x) \leq y_l$ for $x_{ll} \leq x \leq x_{lh}$. 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 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_{yx}, L_{yx}, H_{zy} and L_{zy} 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 $2N + 4$.

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 runs in two phases which are essentially two applications of the same procedure. During the first phase we construct for any pair of variables (x, y) (including (x, x)) a polygon $Q_{xy}(n)$. Assume for simplicity of presentation that n is a power of 2. The polygon $Q_{xy}(n)$ represents tightest bounds that can be derived with respect to y in terms of a value of x, when such a value is sent along any path of length at most n from x to y. The same polygon provides such bounds on x in terms of y. We explain below how these polygons are computed. Polygons of the form $Q_{xx}(n)$ provide specific bounds $xlow$ and $xhigh$ on variables x. These can be derived by minimizing and maximizing x subject to

$$L_{xx}(x) \leq x \leq H_{xx}(x) \ ,$$

where L_{xx} and H_{xx} describe the lower and upper envelopes of the polygon, respectively. To obtain $xlow$ and $xhigh$, suppose X and x are different names of the same variable. The bounds are derived by intersecting the polygon P_{xX} with line $X = x$ (see [11]). The values of $xlow$ and $xhigh$ are then added to the system and the process is repeated. It follows from results in [1] that after the second application of this procedure the projections of the feasible set on all the axes R^x are readily available.

Let the total number of edges in all polygons constructed during the algorithm be E. As in the sequential algorithm of [13], the polygons $Q_{xy}(n)$ are computed in $O(\log n)$ stages, and we have $E = mn^{O(\log n)}$. For each pair (x, y) the polygon $Q_{xy}(2^{s+1})$ is computed by taking the intersection, over all possible z, of the compositions $Q_{xz}(2^s) \circ Q_{zy}(2^s)$. Intersection of n polygons can be computed by n parallel teams of processors in $O(\log n)$ steps, where in each step 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" and discuss classes of problems, within which the worst case is better than the overall worst case. 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 $P = m^2 n^{O(\log n)}$ processors with a concurrent read exclusive write parallel RAM. Using standard simulation relations between parallel models of computation and between parallel time and sequential space [2; 9; 16], this implies that feasibility can be determined by an $O((\log m + \log^2 n)^2 \log^2 n)$ space-bounded read deterministic Turing machine.

5. Computing a feasible solution

We now consider the problem of computing a feasible solution, given the projections of the (nonempty) feasible domain P on the individual axes. Thus, let $[\underline{x}, \bar{x}]$ $(-\infty \leq \underline{x} \leq \bar{x} \leq \infty)$ 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 $\underline{x}$ and $\bar{x}$ are finite, then a feasible solution is obtained by setting each variable x to the arithmetic mean $\frac{1}{2}(\underline{x} + \bar{x})$.*

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

words,

$$\frac{1}{2}\alpha(\underline{x}+\bar{x})+\frac{1}{2}\beta(\underline{y}+\bar{y})>\gamma\ .$$

Consider the rectangle $[\underline{x},\bar{x}]\times[\underline{y},\bar{y}]$. 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

$$\frac{1}{2}\alpha(\underline{x}+\bar{x})+\frac{1}{2}\beta(\underline{y}+\bar{y})>\gamma\ ,$$

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 B together with L with R, and if it lies below this line then it is in the triangle determined by T together with L and R. ∎

Interestingly, the Proposition 5.1 is true only if there are at most two variables per inequality.

The unbounded case is handled as follows. We introduce to the system an additional variable ξ and the $2n$ inequalities $x_j\le\xi$, $x_j\ge-\xi$ $(j=1,\cdots,n)$. We find the projection of the augmented problem on the space of ξ. In other words, we compute an interval $I=[\xi low,\xi high]$ $(-\infty\le\xi low\le\xi high\le\infty)$, such that for every $\xi\in I$ there exist values for $x_1,\cdots,x_n$ which solve the augmented problem. By setting ξ to any finite number in I we obtain a 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. *Over any ordered field, 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 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 optimal 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 [12] to obtain a poly-log algorithm for finding optimal solutions over any ordered field. The idea is roughly as follows (see [12] for more detail). Suppose the problem is to minimize the function

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

subject to a system S of linear inequalities in $x_1, \cdots, x_n$ with at most two variables per inequality. Consider the system S' of inequalities, which is obtained by adding to S an inequality $c_1 x_1 + c_2 x_2 \leq \lambda$, where λ is a parameter. We need to find the smallest value of λ for which S' is feasible. Denote this optimal 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 the 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 the number of processors used by the feasibility checking algorithm. 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 either 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. NC solutions to systems of bounded tree-width

Robertson and Seymour [14] 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 $\mathbf{V} = \{V_1, \cdots, V_t\}$ 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) Every edge of G has both its endpoints lying in at least one of the V_i's.
(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.

It is apparent from the work of Robertson and Seymour that the problem of deciding whether a given graph has tree-width less than or equal to k (and, moreover, constructing a suitable tree $\mathbf{T}$) is difficult. 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 and have no multiple edges, that is, each pair of variables participates in at most one inequality. Obviously, any system is reduced to this form if we replace each inequality $\alpha x + \beta y \leq \gamma$ by a pair of inequalities: $\alpha x - z \leq 0$ and $z + \beta y \leq \gamma$, where z is a new variable. 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 $\mathbf{T}$ so 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 $u = v_1, \cdots, v_r = w$. For every i, $i = 1, \cdots, r-1$, there is a set $V_i \in \mathbf{T}$ such that both v_i and v_{i+1} are in V_i. By definition (V_i, V_{i+1}) is an arc in $\mathbf{T}$. Thus, $V_1, \cdots, V_{i-1}$ is 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 usually useful in the design of parallel algorithms (see [3]). 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 into two subtrees rooted at the centroid. Obviously, the tree decomposition is obtained in $O(\log N)$ iterations, and moreover, it can be computed in poly-log time with a polynomial number of processors [3].

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 the all the vertices of G. Moreover, every edge of G in 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 $Q_{xy}(G)$ represents the tightest bounds that can be obtained on y in terms of x, or vice versa, by updating bounds along paths in some class including all the *simple* paths connecting x to y (or cycles if x and y are the same variable). The paths covered are not necessarily all the paths of length n as in the original algorithm. 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 $\pi(z_0, z_1), \pi(z_1, z_2), \cdots, \pi(z_{l-1}, z_l)$ where $z_0 = x$, $z_l = y$ and $z_i \in C$ for $i = 1, \cdots, l-1$. 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 cover 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 $(k!)^k$. For each pair we compose at most $k+2$ pieces. 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 $O(\log n)$ steps, and there are m inequalities in the beginning, it follows that the number of edges in each of the generated polygons is $m\left(k(k!)^k\right)^{O(\log n)}$. This is the same as $mn^{g(k)}$ where $g(k) = O(k^2 \log k)$, and hence 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.

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

References

[1] B. Aspvall and Y. Shiloach, "A polynomial time algorithm fro solving systems of linear inequalities with two variables per inequality", *SIAM J. Comput.* **9** (1980) 827-845.

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

[3] R. P. Brent, "The parallel evaluation of expressions", *J. Assoc. Comput. Mach.* **21** (1974) 210-208.

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

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

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

[7] 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.

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

[9] 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.

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

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

[12] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms", *J. Assoc. Comput. Mach.* bf 30 (1983) 337-341.

[13] C. G. Nelson, An $n^{O(\log n)}$ algorithm for the two two-variable-per-constraint linear programming satisfiability problem", Report No. STAN-CS-76-689, Department of Computer Science, Stanford University, November 1978.

[14] N. Robertson and P. D. Seymour, "Graph width and well-quasi-ordering: a survey".

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

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