Parallel linear programming in fixed dimension almost surely in constant time

Noga Alon, Nimrod Megiddo

Institutions: IBM, Tel Aviv University

Published on: 01 Mar 1994 - Journal of the ACM (ACM)

Topics: Almost surely, Linear programming, Constant (mathematics), Parallel algorithm and Algorithmics

Parallel Linear Programming in Fixed Dimension Almost Surely in Constant Time

Noga Alon and Nimrod Megiddo

IBM Almaden Research Center
San Jose, California 95120 and
School of Mathematical Sciences
Tel Aviv University, Israel

Revised; January 1992

Abstract. For any fixed dimension d, the linear programming problem with n inequality constraints can be solved on a probabilistic CRCW PRAM with $O(n)$ processors almost surely in constant time. The algorithm always finds the correct solution. With $nd/\log^2 d$ processors, the probability that the algorithm will not finish within $O(d^2 \log^2 d)$ time tends to zero exponentially with n.

1. Introduction

The linear programming problem in fixed dimension is to maximize a linear function of a fixed number, d, of variables, subject to n linear inequality constraints, where n is not fixed. Megiddo [11] showed that for any d, this problem can be solved in $O(n)$ time. Clarkson [4] and Dyer [7] improved the constant of proportionality. Clarkson [5] later developed linear-time probabilistic algorithms with even better complexity. The problem in fixed dimension is interesting from the point of view of parallel computation, since the general linear programming problem is known to be P-complete. The algorithm of [11] can be parallelized efficiently, but the exact parallel complexity of the problem in fixed dimension is still not known.[1] Here we develop a very efficient probabilistic parallel algorithm based on Clarkson's [5] scheme.

In this paper, when we say that a sequence of events $\{E_n\}_{n=1}^{\infty}$ occurs *almost surely*, we mean that there exists an $\epsilon > 0$ such that $\text{prob}(E_n) \geq 1 - e^{-n^{\epsilon}}$. A consequence of this estimate is that with probability 1, only a finite number of the events do not occur. The main result of this paper generalizes a known fact [12; 10] that the maximum of n items can be computed almost surely in constant time.

As mentioned above, the basic idea of the underlying sequential algorithm is due to Clarkson [5]. His beautiful iterative sequential algorithm uses an idea of Welzl [14]. As in Clarkson's algorithm, we also sample constraints repeatedly with variable probabilities. Several additional ideas and some modifications were, however, required in order to achieve the result of this paper. Our probabilistic analysis is also different, and focuses on probabilities of failure to meet time bounds, rather than on expected running times. In particular, a suitable sequential implementation of our algorithm can be shown to terminate almost surely within the best known asymptotic bounds on the expected time.

[1]Ajtai and Megiddo recently developed deterministic algorithms which run on a linear number of processors in $poly(\log\log n)$ time.

In Section 2 we present a special form of the output required from a linear programming problem, which unifies the cases of problems with optimal solutions and unbounded ones. In Section 3 we describe the algorithm and provide the necessary probabilistic analysis.

2. Preliminaries

The purpose of this section is to state the required form of the output of the linear programming problem, which turns out to be useful for our purposes in this paper.

2.1 A special form of the output

Let $N = \{1, \ldots, n\}$ and suppose the linear programming problem is given in the form

$$(LP) \qquad \begin{array}{ll} \text{Minimize} & \boldsymbol{c} \cdot \boldsymbol{x} \\ \text{subject to} & \boldsymbol{a}_i \cdot \boldsymbol{x} \geq b_i \quad (i \in N) \ , \end{array}$$

where $\{\boldsymbol{c}, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \subset R^d$ and $b_1, \ldots, b_n$ are real scalars. An inequality $\boldsymbol{a}_i \cdot \boldsymbol{x} \geq b_i$ is called a *constraint*. We denote by LP_S a similar problem where only a subset $S \subset N$ of the constraints is imposed. If LP is infeasible (*i.e.*, there is no $\boldsymbol{x}$ such that $\boldsymbol{a}_i \cdot \boldsymbol{x} \geq b_i$ for all $i \in N$), then there exists a set $Z \subseteq N$ with $|Z| \leq d + 1$ such that LP_Z is infeasible. In this case we refer to the lexicographically minimal such Z as the *defining subset*.

For any $S \subseteq N$, and for any fixed scalar t, denote by $P_S(t)$ the problem:

$$\text{Minimize} \;\; t\boldsymbol{c} \cdot \boldsymbol{x} + \frac{1}{2}\|\boldsymbol{x}\|^2$$

$$\text{subject to} \;\; \boldsymbol{a}_i \cdot \boldsymbol{x} \geq b_i \quad (i \in S) \; .$$

The objective function of $P_S(t)$ is strictly convex, hence if LP_S is feasible, then $P_S(t)$ has a unique optimal solution $\boldsymbol{x}^S(t)$. It is easy to see that the latter can be characterized as the closest point to the origin, among all points $\boldsymbol{x}$ such that $\boldsymbol{a}_i \cdot \boldsymbol{x} \geq b_i$ $(i \in S)$ and $\boldsymbol{c} \cdot \boldsymbol{x} = \boldsymbol{c} \cdot \boldsymbol{x}^S(t)$. Denote $\operatorname{val}(S,t) = \boldsymbol{c} \cdot \boldsymbol{x}^S(t)$.

Fix t, and let $S \subseteq N$ denote the set of indices i for which $\boldsymbol{a}_i \cdot \boldsymbol{x}^N(t) = b_i$. Obviously, $\boldsymbol{x}^N(t) = \boldsymbol{x}^S(t)$. Moreover, the classical Karush-Kuhn-Tucker optimality conditions imply that $t\boldsymbol{c} + \boldsymbol{x}^N(t)$ is a nonnegative linear combination of the vectors $\boldsymbol{a}_i$ $(i \in S)$, *i.e.*, $t\boldsymbol{c} + \boldsymbol{x}^N(t) \in \operatorname{cone}\{\boldsymbol{a}_i\}_{i \in S}$. By a classical theorem of linear programming, there exists a set $B \subseteq S$ such that $\{\boldsymbol{a}_i\}_{i \in B}$ are linearly independent and $t\boldsymbol{c} + \boldsymbol{x}^N(t) \in \operatorname{cone}\{\boldsymbol{a}_i\}_{i \in B}$. It follows that $\boldsymbol{x}^N(t) = \boldsymbol{x}^B(t)$ and $|B| \leq d$. Moreover, we have $\boldsymbol{a}_i \cdot \boldsymbol{x}^N(t) = b_i$ $(i \in B)$. For this particular value of t, the optimal solution does not change if the inequalities of $P_B(t)$ are replaced by equalities. The importance of this argument about B is that it shows the piecewise linear nature of the parametric solution.

2.2 Analysis of the parametric solution

Denote by $\boldsymbol{B}$ the matrix whose rows are the vectors $\boldsymbol{a}_i$ ($i \in B$), and let $\boldsymbol{b}_B$ denote the vector whose components are the corresponding b_i's. Assuming LP_B is feasible, since $\boldsymbol{x}^B(t)$ minimizes $t\boldsymbol{c} \cdot \boldsymbol{x} + \frac{1}{2}\|\boldsymbol{x}\|^2$ subject to $\boldsymbol{B}\boldsymbol{x} = \boldsymbol{b}_B$, it follows that there exists a $\boldsymbol{y}^B(t) \in R^{|B|}$ such that

$$t\boldsymbol{c} + \boldsymbol{x}^B(t) - \boldsymbol{B}^T\boldsymbol{y}^B(t) = \boldsymbol{0}$$

$$\boldsymbol{B}\boldsymbol{x}^B(t) = \boldsymbol{b}_B \ .$$

Since the rows of $\boldsymbol{B}$ are linearly independent, we can represent the solution in the form:

$$\boldsymbol{y}^B(t) = (\boldsymbol{B}\boldsymbol{B}^T)^{-1}\boldsymbol{b}_B + t(\boldsymbol{B}\boldsymbol{B}^T)^{-1}\boldsymbol{B}\boldsymbol{c}$$

so

$$\boldsymbol{x}^B(t) = \boldsymbol{u}^B + t\boldsymbol{v}^B$$

where

$$\boldsymbol{u}^B = \boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{B}^T)^{-1}\boldsymbol{b}_B$$

and

$$\boldsymbol{v}^B = -(\boldsymbol{I} - \boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{B}^T)^{-1}\boldsymbol{B})\boldsymbol{c} \ .$$

The vector $\boldsymbol{u}^B + t\boldsymbol{v}^B$, however, will be the solution of $P_B(t)$ only for t such that $\boldsymbol{y}^B(t) \geq \boldsymbol{0}$. Denote by I_B the set of all values of t for which $\boldsymbol{y}^B(t) \geq \boldsymbol{0}$, and also $\boldsymbol{a}_i \cdot (\boldsymbol{u}^B + t\boldsymbol{v}^B) \geq b_i$ for all $i \in N$. Obviously, I_B is precisely the interval of t's in which $\boldsymbol{x}^N(t) = \boldsymbol{x}^B(t)$.

We have shown that if LP is feasible, then $\boldsymbol{x}^N(t)$ varies piecewise linearly with t, where each linearly independent set B contributes at most one linear piece. Thus, there exists a "last" set $Z \subset N$ with $|Z| \leq d$, and there exists a t_0, such that for all $t \geq t_0$, $\boldsymbol{x}^N(t) = \boldsymbol{x}^Z(t)$ and $\boldsymbol{a}_i \cdot \boldsymbol{x}^N(t) = b_i$ $(i \in Z)$. Given the correct Z, it is easy to compute $\boldsymbol{u}^Z$, $\boldsymbol{v}^Z$ and the (semi-infinite) interval I_Z in which $\boldsymbol{x}^N(t) = \boldsymbol{x}^Z(t) = \boldsymbol{u}^Z + t\boldsymbol{v}^Z$. It is interesting to distinguish the two possible cases. First, if $\boldsymbol{v}^Z = \mathbf{0}$, then $\boldsymbol{x}^N(t)$ is constant for $t \geq t_0$; this means that the original problem has a minimum, which is the same as if only the constraints corresponding to Z were present. In this case, $\boldsymbol{u}^Z$ is the optimal solution that has the minimum norm among all optimal solutions. Second, if $\boldsymbol{v}^Z \neq \mathbf{0}$, then the original problem is unbounded, and $\{\boldsymbol{u}^Z + t\boldsymbol{v}^Z : t \geq t_0\}$ is a feasible ray along which $\boldsymbol{c} \cdot \boldsymbol{x}$ tends to $-\infty$. Moreover, each point on this ray has the minimum norm among the feasible points with the same value of $\boldsymbol{c} \cdot \boldsymbol{x}$.

In view of the above, we can now define the vectors $\boldsymbol{u}^N$ and $\boldsymbol{v}^N$ to be equal to $\boldsymbol{u}^Z$ and $\boldsymbol{v}^Z$, respectively. Indeed, for any subset $S \subseteq N$ (whose corresponding vectors $\boldsymbol{a}_i$ may be linearly dependent), we can define the appropriate vectors $\boldsymbol{u}^S$ and $\boldsymbol{v}^S$ to describe the output required in the problem LP_S. To summarize, we have proven the following:

Proposition 2.1. *If the ray $\boldsymbol{u}^N + t\boldsymbol{v}^N$ coincides with the optimal solution of $P_N(t)$ for all sufficiently large t, then there exists a subset $Z \subset N$, whose corresponding vectors are* *linearly independent, such that the ray coincides with the optimal solution of $P_Z(t)$ for all such t.*

For every point on a polyhedron, there exists precisely one face of the polyhedron which contains the point in its relative interior. Consider the lexicographically minimal set Z which describes this face. We say that this set Z is the *defining subset* of the solution $(\boldsymbol{u}^N, \boldsymbol{v}^N)$.

2.3 The fundamental property

Denote by $V(\boldsymbol{u}, \boldsymbol{v})$ the set of indices $i \in N$ for which $\boldsymbol{a}_i \cdot (\boldsymbol{u} + t\boldsymbol{v}) < b_i$ for all sufficiently large values of t. If $i \in V(\boldsymbol{u}, \boldsymbol{v})$, we say that the corresponding constraint is asymptotically violated on $(\boldsymbol{u}, \boldsymbol{v})$. Obviously, if $\boldsymbol{v} = \boldsymbol{0}$, then $V(\boldsymbol{u}, \boldsymbol{v})$ is the set of indices $i \in N$ such that $\boldsymbol{a}_i \cdot \boldsymbol{u} < b_i$. If $\boldsymbol{v} \neq \boldsymbol{0}$, then $i \in V(\boldsymbol{u}, \boldsymbol{v})$ if and only if either $\boldsymbol{a}_i \cdot \boldsymbol{v} < 0$ or $\boldsymbol{a}_i \cdot \boldsymbol{v} = 0$ and $\boldsymbol{a}_i \cdot \boldsymbol{u} < b_i$.

The following proposition is essentially due to Clarkson [5]:

Proposition 2.2. *For any $S \subset N$ such that $V(\boldsymbol{u}^S, \boldsymbol{v}^S) \neq \emptyset$, and for any $I \subset N$ such that* $\mathrm{val}(N, t) = \mathrm{val}(I, t)$ *(for all sufficiently large t), $V(\boldsymbol{u}^S, \boldsymbol{v}^S) \cap I \neq \emptyset$.*

Proof: If on the contrary $V(\boldsymbol{u}^S, \boldsymbol{v}^S) \cap I = \emptyset$, then we arrive at the contradiction that for all sufficiently large t,

$$\mathrm{val}(I) \leq \mathrm{val}(S \cup I) = \mathrm{val}(S) < \mathrm{val}(N) ,$$

where the strict inequality follows from the uniqueness of the solution of $P_S(t)$. ∎

The importance of Proposition 2.2 can be explained as follows. If a set S has been found such that at least one constraint is violated at the optimal solution of LP_S, then at least one of these violated constraints must belong to the defining set. Thus, when the probabilistic weight of each violated constraint increases, we know that the weight of at least one constraint from the defining set increases.

3. The algorithm

As mentioned above, the underlying scheme of our algorithm is the same as that of the iterative algorithm in the paper by Clarkson [5], but the adaptation to a parallel machine requires many details to be modified.

During a single iteration, the processors sample a subset S of constraints and solve the subproblem LP_S with "brute force." If the latter is infeasible then so is the original one and we are done. Also, if the solution of the latter is feasible in the original problem, we can terminate. Typically, though, some constraints of the original problem will be violated at the solution of the sampled subproblem. In such a case, the remainder of the iteration is devoted to modifying the sample distribution for the next iteration, so that such violated constraints become more likely to be sampled. The process of modifying the distribution is much more involved in the context of parallel computation. It amounts to replicating violated constraints, so that processors keep sampling from a "uniform" distribution. The replicating procedure is carried out in two steps. First, the set of violated constraints is "compressed" into a smaller area, and only then the processors attempt to replicate.

During the run of the algorithm, the probability that the entire "defining set" is included in the sample increases rapidly. In order to implement the above ideas efficiently on a PRAM, several parameters of the algorithm have to be chosen with care and special mechanisms have to be introduced. The algorithm utilizes $p = p(n,d) = 2nd/\log^2 d$ processors $P_1,\ldots,P_p$. Denote by $k = k(n,d)$ the largest integer such that $\max\{d^3, kd/\log^2 d\}\binom{k}{d} \le p(n,d)$. Note that[2] $k = \Omega(n^{1/(d+1)})$. We first describe the organization of the memory shared by our processors.

3.1 The shared memory

The shared memory consists of four types of cells as follows.

(i) The *Base* B, consisting of k cells, $B[1],\ldots,B[k]$.

[2]The notation $f(n) = \Omega(g(n))$ means that there exists a constant $c > 0$ such that $f(n) \ge cg(n)$.

(ii) The *Sequence* S, consisting of $2n$ cells, $S[1], \ldots, S[2n]$. We also partition the Sequence into $2n^{3/4}$ blocks of length $n^{1/4}$, so these cells are also addressed as $S[I, J]$, $I = 1, \ldots, n^{1/4}$, $J = 1, \ldots, 2n^{3/4}$.

(iii) The *Table* T, consisting of $m = C_d n^{1-1/(2d)}$ cells $T[1], \ldots, T[m]$, where $C_d = \log(16d) + 2$. We also partition the Table into C_d blocks of length $m' = n^{1-1/(2d)}$, so these cells are also addressed as $T[I, J]$, $I = 1, \ldots, m'$, $J = 1, \ldots, C_d$.

(iv) The *Area* R, consisting of $n^{3/4}$ cells, $R[1], \ldots, R[n^{3/4}]$. We also partition the Area into $n^{1/4}$ blocks of size $\sqrt{n}$, so these cells are also addressed as $R[I, J]$, $I = 1, \ldots, \sqrt{n}$, $J = 1, \ldots, n^{1/4}$.

Each memory cell stores either some halfspace $H_i = \{\boldsymbol{x} \in R^d : \boldsymbol{a}_i \cdot \boldsymbol{x} \leq b_i\}$ $(i \in N)$, or the space R^d. Initially, $S[j] = H_j$ for $j = 1, \ldots, n$, and all the other cells store the space R^d. The Base always describes a subproblem LP_K where K is the set of the constraints stored in the Base. By the choice of our parameters, every Base problem can be solved by "brute force" in $O(\log^2 d)$ time as we show in Proposition 3.1 below.

The Sequence is where the sample space of constraints is maintained. Initially, the Sequence stores one copy of each constraint. Throughout the execution of the algorithm, more copies are added, depending on the constraints that are discovered to be violated at solutions of subproblems. The role of the Table and the Area is to facilitate the process of modifying the sample distribution.

3.2 The base problem

As already indicated, the algorithm repeatedly solves by "brute force" subproblems consisting of k constraints

Proposition 3.1. *Using a* $(2nd/\log^2 d)$*-processor* CRCW PRAM*, any subproblem* LP_K *with* $|K| = k(n,d)$ *constraints can be solved deterministically in* $O(\log^2 d)$ *time.*

Proof: Recall that $p = 2nd/\log^2 d$. In order to solve the Base problem, $p/\binom{k}{d}$ processors are allocated to each subset $B \subset K$ such that $|B| = d$. Thus, the number of processors assigned to each B is bounded from below by $\max\{d^3, kd/\log^2 d\}$. It follows that all the subproblems LP_B can be solved in $O(\log^2 d)$ time, as each of them amounts to solving a system of linear equations of order $(2d) \times (2d)$, and we have at least d^3 processors (see [2]). If any of the LP_B's is discovered to be infeasible then LP is infeasible, and the algorithm stops. Otherwise, for each B the algorithm checks whether $\boldsymbol{u}^B + t\boldsymbol{v}^B$ is asymptotically feasible (*i.e.*, feasible for all sufficiently large t) in LP_K. With $d/\log^2 d$ processors, it takes $O(\log^2 d)$ time to evaluate the inner product of two d-vectors.[3] Since there are at least $kd/\log^2 d$ processors assigned to each B,

[3]It is easy to see that the inner product can be evaluated by $d/\log d$ processors in $O(\log d)$ time. Here we can afford $O(\log^2 d)$ time, so we can save on the number of processors.

the asymptotic feasibility of all the $(\boldsymbol{u}^B + t\boldsymbol{v}^B)$'s in LP_K can be checked in $O(\log^2 d)$ time. Finally, the algorithm finds the best among the solutions of the LP_B's which are feasible in LP_K or, in case none of these is feasible in LP_K, the algorithm recognizes that LP_K, and hence also LP, is infeasible. The final step is essentially a computation of the minimum of $\binom{k}{d} = O(n^{d/(d+1)})$ numbers. An algorithm of Valiant[4] [13] (which can be easily implemented on a CRCW PRAM) finds the minimum of m elements, using p processors, in $O(\log(\log m/\log(p/m)))$ time. Here, $m = \binom{k}{d} = O(n^{d/(d+1)})$ and $p = 2nd/\log^2 d$, so the time is $O(\log d)$. ∎

3.3 The iteration

We now describe how the sampling works and how the sample space is maintained.

Sampling a base problem

An iteration of the algorithm starts with sampling a Base problem. As indicated above, the sample space is stored in the S-cells. There are $2n$ such cells and each stores either a constraint or the entire space; one constraint may be stored in more than one S-cell. To perform the sampling, each of the first k processors P_i generates a random integer[5] I_i uniformly between 1 and $2n$, and copies the contents of the S-cell $S[I_i]$ into the B-cell

[4]We note that with a different choice of k, namely, if k is such that $\binom{k}{d} \le \sqrt{p}$, then the use of Valiant's algorithm can be avoided.

[5]We assume each of the processors can generate random numbers of $O(\log n)$ bits in constant time.

$B[\,i\,]$. Next, all the processors jointly solve the subproblem LP_K currently stored in the Base (see Proposition 3.1) in $O(\log^2 d)$ time.

Asymptotically violated constraints

Assuming LP_K is feasible, the algorithm now checks which S-cells store constraints that are violated asymptotically on the ray $\{\boldsymbol{u}^K + t\boldsymbol{v}^K\}$, *i.e.*, for t sufficiently large. This task is accomplished by assigning $d/\log^2 d$ processors to each cell $S[\,i\,]$ as follows. For each i ($i = 1, \ldots, 2n$), the processors P_j with $1 + (i-1)d/\log^2 d \le j \le id/\log^2 d$ are assigned to the cell $S[\,i\,]$; they check the asymptotic feasibility of the constraint stored therein, as explained in Section 2. This step also takes $O(\log^2 d)$ time, since it essentially amounts to an evaluation of inner products of d-vectors. For brevity, we say that an S-cell storing an asymptotically violated constraint is itself *violated.*

Replicating violated constraints

Having identified the violated S-cells, the processors now "replicate" the contents of each such cell $n^{1/(4d)}$ times. The idea is that by repeating this step several times, the members of the "defining set" get a sufficiently large probability to be included in the sample (in which case the problem is solved). Since it is not known in advance which S-cells are violated, and since there are only $O(n)$ processors, the algorithm cannot decide in advance which processors will replicate which cells. For this reason, the replication step is carried out on a probabilistic CRCW PRAM in two parts. First, the violated S-cells are injected into the Table (whose size is only $O(n^{1-1/(2d)})$), and then replications are made from the Table back into the Sequence, using a predetermined assignment of processors to cells. The first part of this step is performed as follows.

Injecting into the Table: Operation 1

First, for any violated cell $S[\,i\,]$, processor P_i generates a random integer I_i^1 between 1 and m', and attempts to copy the contents of $S[\,i\,]$ into $T[I_i^1, 1]$. Next, if P_i has attempted to write and failed, then it generates a random integer I_i^2 between 1 and m' and attempts again to copy the contents of $S[\,i\,]$ into $T[I_i^2, 2]$. In general, each such processor attempts to write at most $C_d - 1$ times, each time into a different block of the Table.

Proposition 3.2. *The conditional probability that at least $n^{1/4}$ processors will fail to write during all the $C_d - 1$ trials, given that at most $n^{1-\frac{1}{2d}-\frac{1}{16d}}$ processors attempt to write during the first trial, is at most $e^{-\Omega(n^{1/4})}$.*

Proof: Let X_i be the random variable representing the number of processors that failed to write during the first i rounds (and therefore attempt to write during the $(i+1)$-st round). Suppose $X_0 \leq n^{1-\frac{1}{2d}-\frac{1}{16d}}$. Note that for each processor attempting to write during the i-th round, the conditional probability that it will be involved in a write conflict, given any information on the success or failure of the other processors during this round, is at most X_{i-1}/m'. Thus, we can apply here estimates for independent Bernoulli variables. By an estimate due to Chernoff [3] (apply Proposition 4.1 part (i) with $n = \xi_{i-1}$ and $p = \xi_{i-1}/m'$),

$$\operatorname{prob}\left\{X_i > 2\xi_{i-1}^2/m' \mid X_{i-1} = \xi_{i-1}\right\} \quad \leq \quad e^{-\Omega(\xi_{i-1}^2/m')} .$$

Let j denote the largest integer such that

$$2^{2^j-1} n^{1-\frac{1}{2d}-\frac{2^j}{16d}} \quad \geq \quad e^{-2} n^{\frac{1}{4}} .$$

Clearly, $j < \log(16d)$ for n sufficiently large. Notice that if indeed $X_i \leq 2X_{i-1}^2/m'$ for all i, $1 \leq i \leq j$, then

$$\frac{2X_i}{m'} \leq \left(\frac{2X_{i-1}}{m'}\right)^2$$

and hence,

$$\frac{2X_j}{m'} \leq \left(\frac{2X_0}{m'}\right)^{2^j} .$$

Thus, $X_j \leq 2^{2^j-1} n^{1-\frac{1}{2d}-\frac{2^j}{16d}}$. The probability that j does not satisfy the latter is at most $e^{-\Omega(n^{1/4})}$. Combining this with Proposition 4.1 part (ii), we get

$$\operatorname{prob}\{X_{j+1} > n^{1/4}\} \quad \leq \quad e^{-\Omega(n^{1/4})} .$$

∎

We note that, as pointed out by one of the referees, the analysis in the last proposition can be somewhat simplified by having the processors try to inject the violated constraints to the same table for some C'_d times (or until they succeed). It is easy to see that for, say, $C'_d = 16d$ the conclusion of Proposition 3.2 will hold. However, this will increase the running time of each phase of our algorithm to $\Omega(d)$ and it is therefore better to perform the first part of the injection as described above. Alternatively, the time increase can be avoided by making all C'_d attempts in parallel and then, in case of more than one success, erase all but the first successful replication. We omit the details of the implementation.

Injecting into the Table: Operation 2

To complete the step of injecting into the Table, one final operation has to be performed. During this operation, the algorithm uses a predetermined assignment of some $q = n^{1/4}$ processors to each of the $2n^{3/4}$ S-blocks, *e.g.*, processor P_j is assigned to the block $S[*, \lceil jn^{-1/4} \rceil]$. An S-cell is said to be *active* at this point if it has failed $C_d - 1$ times to be injected into the Table. An S-block is said to be *active* if it contains at least one active cell.[6] For each active block $S[*, J]$ $(1 \leq J \leq 2n^{3/4})$, all the q processors assigned to $S[*, J]$ attempt to write the symbol J into the Area R as follows. The i-th processor among those assigned to $S[*, J]$ generates a random[7] integer I_i between 1 and $\sqrt{n}$, and attempts to write the symbol J into the cell $R[I_i, i]$.

[6]It takes constant time to reach the situation where each of the q processors knows whether or not it is assigned to an active S-block.

[7]This last step can also be done deterministically with hash functions.

Proposition 3.3. *If there are less than $n^{1/4}$ active S-blocks, then the probability that write conflicts will occur in every single R-block is less than $e^{-\Omega(n^{1/4})}$.*

Proof: At most $n^{1/4}$ processors attempt to write into any R-block (whose length is $\sqrt{n}$), so the probability of a conflict within any fixed R-block is less than $1/2$. Thus, the probability of conflicts in every single R-block is less than $2^{-n^{1/4}}$. ∎

It takes constant time to reach a situation where all the processors recognize that a specific block $R[*, J^*]$ was free of conflict (assuming at least one such block exists). At this point, the names of all the active S-blocks have been moved into one commonly known block $R[*, J^*]$. The role of the area R is to facilitate the organization of the work involved in replicating the remaining active S-blocks into the last T-block.

Next, $n^{1/4}$ processors are assigned to each cell $R[I, J^*]$ ($I = 1, \ldots, \sqrt{n}$). Each such cell is either empty or contains the index J_I of some active S-block $S[*, J_I]$. In the latter case, the processors assigned to $R[I, J^*]$ copy the contents of the active cells of $S[*, J_I]$ into the last T-block, $T[*, C_d]$, according to some predetermined assignment.

From the Table to the Sequence

For the second half of the replication step, the algorithm uses a predetermined (many-to-one) assignment $P_j \rightarrow T[\sigma(j)]$ of the first $C_d n^{1-1/(4d)}$ processors to the $m = C_d n^{1-1/(2d)}$ T-cells, where $n^{1/(4d)}$ processors are assigned to each cell. Each processor P_j copies the contents of $T[\sigma(j)]$ into a cell $S[\ell]$, where

$$\ell = \ell(j, \nu) = n + (\nu - 1)C_d n^{1-1/(4d)} + j$$

depends both on the processor and on the iteration number ν. Later, we will discuss the actual number of iterations. We will show that almost surely $\ell \leq 2n$. In the unlikely event that the number of iterations gets too large, the algorithm simply restarts with $\nu = 1$.

3.4 Probabilistic analysis

In this section we analyze the probability that the algorithm fails to include the defining set in the sample after a certain constant number of iterations.

Estimating the number of violated S-cells

Let $\preceq$ be any fixed weak linear order on N. Given the contents of the Sequence S and the random Base set K, denote by $\mu = \mu(S, K; \preceq)$ the number of S-cells which store halfspaces H_i such that[8] $j \prec i$ for all j such that H_j is in the Base.

Proposition 3.4. *For any possible contents of the Sequence and for every $\epsilon > 0$,*

$$\text{prob}\{\mu > n^{1-\frac{1}{d+1}+\epsilon}\} \quad < \quad e^{-\Omega(n^\epsilon)} \ .$$

[8]We write $j \prec i$ if and only if $j \preceq i$ and $i \not\preceq j$.

Proof: For any $x > 0$,

$$\text{prob}\,\{\mu > x\} \le \left(1 - \frac{x}{2n}\right)^k .$$

In particular, for $x = n^{1-\frac{1}{d+1}+\epsilon}$ this probability is at most

$$\left(1 - \frac{1}{2n^{\frac{1}{d+1}-\epsilon}}\right)^k .$$

Our claim follows from the fact that $k = \Omega(n^{\frac{1}{d+1}})$. ∎

For any $M \subseteq N$ such that LP_M is feasible, denote by $\overset{M}{\preceq}$ the weak linear order induced on N by the asymptotic behavior of the quantities $b_i - \boldsymbol{a}_i \cdot (\boldsymbol{u}^M + t\boldsymbol{v}^M)$ as t tends to infinity. More precisely, $j \overset{M}{\preceq} i$ if and only if for all sufficiently large t,

$$b_j - \boldsymbol{a}_j \cdot (\boldsymbol{u}^M + t\boldsymbol{v}^M) \le b_i - \boldsymbol{a}_i \cdot (\boldsymbol{u}^M + t\boldsymbol{v}^M) .$$

For brevity, denote $\mu'(M) = \mu(S, K; \overset{M}{\preceq})$

Corollary 3.5. *For any $\epsilon > 0$, the conditional probability that there will be more than $n^{1-\frac{1}{d+1}+\epsilon}$ violated S-cells on $\boldsymbol{u}^K + t\boldsymbol{v}^K$, given that LP_K is feasible, is less than $\binom{n}{d}e^{-\Omega(n^\epsilon)}$.*

Proof: Consider the set $\mathcal{L}$ of orders $\overset{B}{\preceq}$, where $B \subset N$ corresponds to a set of linearly independent vectors (and hence $|B| \le d$), and LP_B is feasible. If LP_K is feasible, then by Proposition 2.1 there exists $Z \subseteq K$, whose corresponding constraints are linearly independent, such that $(\boldsymbol{u}^Z, \boldsymbol{v}^Z) = (\boldsymbol{u}^K, \boldsymbol{v}^K)$. Thus, $\overset{K}{\preceq} \in \mathcal{L}$. By Proposition 3.4, for any fixed M,

$$\mathrm{prob}\{\mu'(M) > n^{1-\frac{1}{d+1}+\epsilon}\} < e^{-\Omega(n^{\epsilon})} \ .$$

Since $|\mathcal{L}| \leq \binom{n}{d}$,

$$\mathrm{prob}\{\mu'(K) > n^{1-\frac{1}{d+1}+\epsilon}\} < \binom{n}{d} e^{-\Omega(n^{\epsilon})} \ .$$

▮

Proposition 3.6. *During each iteration, the probability that at least one active S-cell will fail to inject its contents into the Table is at most $e^{-\Omega(n^{1/(16d)})}$.*

Proof: The proof follows immediately from Corollary 3.5 with $\epsilon = 1/(16d)$ together with Propositions 3.2 and 3.3. ▮

Successful iterations

Let LP_K denote the current Base problem. An iteration is considered successful in either of the following cases:

(i) The problem LP_K is discovered to be infeasible, hence so is LP and the algorithm stops.

(ii) The problem LP_K is feasible and its solution $\boldsymbol{u}^K + t\boldsymbol{v}^K$ turns out to be feasible for LP for all sufficiently large t, so it is also the solution of LP and the algorithm stops.

(iii) For at least one i in the defining set Z (see Section 2), H_i is asymptotically violated on $\boldsymbol{u}^K + t\boldsymbol{v}^K$, and all S-cells storing H_i are injected into the Table.

Proposition 3.7. *During any iteration, given any past history, the conditional probability of failure is at most* $e^{-\Omega(n^{1/(16d)})}$.

Proof: By Proposition 2.2, if the solution of LP has not been found, then H_i is violated, for at least one $i \in Z$, and hence every processor checking a copy of H_i will attempt to inject it into the Table. The result now follows from Proposition 3.6. ∎

Proposition 3.8. *For any fixed d, the probability that the algorithm will not finish within $9d^2$ iterations is at most* $e^{-d^2\Omega(n^{1/(16d)})}$.

Proof: Notice that in $9d^2$ iterations, for sufficiently large n, only the first $n + 9d^2 C_d n^{1-1/(4d)} < 2n$ S-cells are possibly accessed. By Proposition 3.7, in each iteration, the conditional probability of failure, given any past history, is at most $e^{-\Omega(n^{1/(16d)})}$. Therefore, the probability of less than $5d^2$ successes in $9d^2$ iterations is less than

$$\binom{9d^2}{5d^2}\left(e^{-\Omega(n^{1/(16d)})}\right)^{4d^2} \leq e^{-d^2\Omega(n^{1/(16d)})}\ .$$

To complete the proof, we show that it is impossible to have $5d^2$ successes. This is because if there are that many successes, then there exists at least one i in the "defining set" Z such that during at least $5d$ of the iterations, the contents of all the S-cells storing the halfspace H_i are successfully injected into the Table.[9] This means that there are at least

$$\left(n^{1/(4d)}\right)^{5d} = n^{1.25} > 2n$$

S-cells storing H_i, whereas the total length of the Sequence is only $2n$. Hence, a contradiction. ∎

Thus, we have proven the following:

Theorem 3.9. *There exists a probabilistic parallel algorithm for the linear programming problem with d variables and n constraints, which runs on a $(2nd/\log^2 d)$-processor CRCW PRAM with performance as follows. The algorithm always finds the correct solution. There exists an $\epsilon > 0$ (e.g., $\epsilon = 1/16$) such that for every fixed d and for all sufficiently large n, the probability that the algorithm takes more than $O(d^2 \log^2 d)$ time is less than $e^{-\Omega(n^{\epsilon/d})}$.*

A further improvement

It is not too difficult to modify the algorithm to obtain one for which there are two constants $C, \epsilon > 0$, *independent of d* with performance as follows. For every fixed dimension d, and for all sufficiently large n, the probability that the running time will exceed

[9] The Table is erased after each iteration.

$Cd^2 \log^2 d$ is at most $2^{-\Omega(n^\epsilon)}$. This is done by choosing the size k of the Base problem so that $k\binom{k}{d} \le \sqrt{n}$. This enables us to solve during each iteration $\sqrt{n}$ random Base problems simultaneously. As before, processors are assigned to S-cells. Each such processor chooses randomly one of the Base problems. The processor then checks whether the constraint in its cell is violated at the solution of the Base problem. With each of the $\sqrt{n}$ Base problems we associate a Table of size $n^{\frac{1}{2}-\frac{1}{2d}+\frac{1}{32d}}$. Next, each processor which has a violated S-cell (with respect to the Base problem i that was chosen by that processor) attempts to inject the contents of its cell into the Table of Base problem i. This is done as in the corresponding steps of the algorithm described above. We call a Base problem *successful* if all the processors attempting to write succeed eventually. Note that if Base problem i is successful, then not too many S-cells (among those whose processors chose the Base problem i) were violated. Therefore, with high probability, not too many S-cells altogether were violated at the solution of this Base problem. The algorithm now chooses a successful Base problem. It then continues as the previous algorithm, i.e., it checks which of all the S-cells are violated, injects these cells into a Table of size $2C_d n^{1-1/(4d)}$, and replicates each of the violated ones $n^{1/5d}$ times. We say that an iteration is *successful* if at least one of its $\sqrt{n}$ Base problems is successful, and the contents of *all* the violated S-cells are injected successfully into the Table. It is not too difficult to check that the conditional probability that an iteration will not be successful, given any information about the success or failure of previous iterations, is at most $e^{-\Omega(n^{\epsilon})}$ for some $\epsilon > 0$ (e.g., $\epsilon = 1/16$). We omit the details.

Remarks

The total work done by all the processors in our algorithm is $O(d^3 n)$, whereas Clarkson's sequential algorithm [5] runs in expected $O(d^2 n)$ time. We can easily modify our algorithm to run on a probabilistic CRCW PRAM with $n/(d \log^2 d)$ processors in $O(d^3 \log^2 d)$ time, so that the total work is $O(d^2 n)$. Moreover, the probability of a longer running time is exponentially small in terms of n. To this end, observe that, using our previous algorithm, we can solve in $O(d^2 \log^2 d)$ time and $n/(d \log^2 d)$ processors a Base problem of size n/d^2. Hence, we can repeat the previous algorithm by choosing Base problems of size n/d^2, solving them, checking all the violated S-cells in $O(d^2 \log^2 d)$ time, and replicating each violated S-cell $\sqrt{n}$ times. Such an algorithm terminates almost surely in $O(d)$ iterations. Hence, the total parallel time is $O(d^3 \log^2 d)$.

4. Appendix

The following proposition summarizes the standard estimates of the binomial distribution which are used in the paper. A random variable X has the binomial distribution with parameters n, p, if it is the sum of n independent $(0, 1)$-variables, each with expectation

p.

Proposition 4.1. *If X is a binomial random variable with parameters n, p, then*

(i) *For every $a > 0$,*

$$\text{prob}\{X - np > a\} < e^{-\frac{a^2}{2pn} + \frac{a^3}{2(pn)^2}} .$$

In particular, for $a = 0.5np$,

$$\text{prob}\{X > 2np\} < \text{prob}\{X > 1.5np\}$$

$$< e^{-\frac{np}{16}}.$$

(ii) *If $a > e^2 np$ then* $\text{prob}\{X > a\} < e^{-a}$.

Proof: Part (i) is due to Chernoff [3]. (see also [1], p. 237). Part (ii) follows immediately from the fact that

$$\text{prob}\{X > a\} < \binom{n}{a} p^a$$
$$\leq \left(\frac{en}{a}\right)^a \left(\frac{a}{e^2 n}\right)^a = e^{-a}.$$

∎

Acknowledgement. We thank Moni Naor for helpful suggestions concerning the step of injection into the Table, that led to a significant improvement in the estimate of the failure probabilities. We also thank two anonymous referees for their helpful comments.

References

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley, New York, 1991.

[2] A. Borodin, J. von zur Gathen and J. E. Hopcroft, "Fast parallel matrix and GCD computations," *Information and Control* **52** (1982) 241–256.

[3] H. Chernoff, "A measure of asymptotic efficiency for tests of hypothesis based on the sum of observations," *Ann. Math. Stat.* **23** (1952) 493–507.

[4] K. L. Clarkson, "Linear programming in $O(n3^{d^2})$ time," *Information Processing Letters* **22** (1986) 21–24.

[5] K. L. Clarkson, "Las Vegas algorithms for linear and integer programming when the dimension is small," unpublished manuscript, 1988; a preliminary version appeared in *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science (1988),* pp. 452–456.

[6] X. Deng, "An optimal parallel algorithm for linear programming in the plane," unpublished manuscript, Dept. of Operations Research, Stanford University and Dept. of Computer Science and Engineering, University of California at San Diego.

[7] M. E. Dyer, "On a multidimensional search technique and its application to the Euclidean one-center problem," *SIAM J. Comput.* **15** (1986) 725–738.

[8] M. E. Dyer and A. M. Freeze, "A randomized algorithm for fixed dimensional linear programming," *Mathematical Programming* **44** (1989) 203–212.

[9] G. S. Lueker, N. Megiddo and V. Ramachandran, "Linear programming with two variables per inequality in poly log time," *SIAM J. Comput.* , to appear.

[10] N. Megiddo, "Parallel algorithms for finding the maximum and the median almost surely in constant-time," Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, October 1982.

[11] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *J. ACM* **31** (1984) 114–127.

[12] R. Reischuk, "A fast probabilistic parallel sorting algorithm," in: *Proceedings of the 22th Annual IEEE Symposium on Foundations of Computer Science (1981),* pp. 212–219.

[13] L. G. Valiant, "Parallelism in comparison algorithms," *SIAM J. Comput.* **4** (1975) 348–355.

[14] E. Welzl, "Partition trees for triangle counting and other range searching problems," in: *Proceedings of the 4th Annual ACM Symposium on Computational Geometry (1988)*, pp. 23–33.