Comp. & Maths. with Appls. Vol. 12A, No. 12, pp. 1217–1227, 1986
Printed in Great Britain.

0886-9553/86 $3.00 + 0.00

FAST AND EFFICIENT LINEAR PROGRAMMING AND LINEAR LEAST-SQUARES COMPUTATIONS†

V. PAN
Computer Science Department, State University of New York at Albany, Albany, NY 12222, U.S.A.

J. REIF
Aiken Computation Laboratory, Division of Applied Sciences, Harvard University, Cambridge, MA 02138, U.S.A.

(*Received April 1986*)

Communicated by E. Y. Rodin

Abstract—We present a new parallel algorithm for computing a least-squares solution to a sparse overdetermined system of linear equations $\mathbb{A}\mathbf{x} = \mathbf{b}$ such that the $m \times n$ matrix $\mathbb{A}$ is sparse and the graph, $G = (V, E)$, of the matrix

$$\mathbb{H} = \begin{bmatrix} \mathbb{I} & \mathbb{A}^T \\ \mathbb{A} & \mathbb{O} \end{bmatrix}$$

has an $s(m+n)$-separator family, i.e. either $|V| < n_0$ for a fixed constant n_0 or, by deleting a separator subset S of vertices of size $\leqslant s(m+n)$, G can be partitioned into two disconnected subgraphs having vertex sets V_1, V_2 of size $\leqslant 2/3(m+n)$, and each of the two resulting subgraphs induced by the vertex sets $S \cup V_i$, $i = 1, 2$, can be recursively $s(|S \cup V_i|)$-separated in a similar way. Our algorithm uses $O(\log(m+n)\log^2 s(m+n))$ steps and $\leqslant s^3(m+n)$ processors; it relies on our recent parallel algorithm for solving sparse linear systems and has several immediate applications of interest, in particular to mathematical programming, to sparse nonsymmetric systems of linear equations and to the path algebra computations. We most closely examine the impact on *the linear programming problem* (LPP) which requires maximizing $\mathbf{c}^T\mathbf{y}$ subject to $\mathbb{A}^T\mathbf{y} \leqslant \mathbf{b}$, $\mathbf{y} \geqslant \mathbf{0}$, where $\mathbb{A}$ is an $m \times n$ matrix. Hereafter it is assumed that $m \geqslant n$. The recent algorithm by Karmarkar gives the best-known upper estimate [$O(m^{3.5}L)$ arithmetic operations, where L is the input size] for the cost of the solution of this problem in the worst case. We prove an asymptotic improvement of that result in the case where the graph of the associated matrix $\mathbb{H}$ has an $s(m+n)$-separator family; then our algorithm can be implemented using $O(mL \log m \log^2 s(m+n))$ parallel arithmetic steps, $s^3(m+n)$ processors and a total of $O(mLs^3(m+n)\log m \log^2 s(m+n))$ arithmetic operations. In many cases of practical importance this is a considerable improvement on the known estimates: for example, $s(m+n) = \sqrt{8}(m+n)$ if G is planar [as occurs in many operations research applications; for instance, in the problem of computing the maximum multicommodity flow with a bounded number of commodities in a network having an $s(m+n)$-separator family], so that the processor bound is only $8\sqrt{8}(m+n)^{1.5}$ and the total number of arithmetic steps is $O(m^{2.5}L)$. Similarly, Karmakar's algorithm and the known algorithms for the solution of overdetermined linear systems are accelerated in the case of dense input matrices via our recent parallel algorithms for the inversion of dense $k \times k$ matrices using $O(\log^2 k)$ steps and k^3 processors. Combined with a modification of Karmarkar's algorithm, this implies solution of the LPP using $O(Lm \log^2 m)$ steps and $m^{2.5}$ processors. The stated results promise some important practical applications. Theoretically, the above processor bounds can be reduced for dense matrix inversion to $o(k^{2.5})$ and for the LPP to $o(m^{2.165})$ in the dense case and to $o(s^{2.5}(m+n))$ in the sparse case (preserving the same number of parallel steps); this also decreases the known sequential time bound for the LPP by a factor of $m^{0.335}$, i.e. to $O(Lm^{3.165})$.

1. INTRODUCTION

Numerous practical computations require finding a least-squares solution to an *overdetermined* system of linear equations, $\mathbb{A}\mathbf{x} = \mathbf{b}$, i.e. finding a vector $\mathbf{x}$ of dimension n that minimizes $\|\mathbb{A}\mathbf{x} - \mathbf{b}\|$ given an $m \times n$ matrix $\mathbb{A}$ and a vector $\mathbf{b}$ of dimension m where $m \geqslant n$. (Here and hereafter we apply the Euclidean vector norm and the associated 2-norm of matrices [1].) Such a problem is called the *linear least-squares problem* (LLSP). In

†The results of this paper were presented at the *12th Int. Symp. on Mathematical Programming*, Boston, Mass., 5–9 Aug. 1985.

particular, solving a linear system $\mathbb{A}\mathbf{x} = \mathbf{b}$ in the usual sense is a simplification of the LLSP where the output is either the answer that

$$\min_{\mathbf{x}} \| \mathbb{A}\mathbf{x} - \mathbf{b} \| > 0$$

or, otherwise, a vector $\mathbf{x}^*$ such that

$$\mathbb{A}\mathbf{x}^* - \mathbf{b} = \mathbf{0}.$$

The first objective of this paper is to reexamine the time complexity of the LLSP and to indicate the possibility of speeding up its solution using the parallel algorithms of Ref. [2] combined with the techniques of *blow-up transformations* and *variable diagonals* and with the Sherman–Morrison–Woodbury formula. As a major consequence [which may become decisive in determining the best algorithm for the *linear programming problem* (LPP), at least over some important classes of instances of that problem], we will substantially speed up Karmarkar's algorithm [3] for the LPP, because solving the LLSP constitutes the most costly part of every iteration of that algorithm. Furthermore, we will modify Karmarkar's algorithm and solve an LPP with a dense $m \times n$ input matrix using $O(Lm \log^2 m)$ parallel arithmetic steps and $m^{2.5}$ processors, where the parameter L (defined in Ref. [3]) represents the input size of the problem. Applying fast matrix multiplication algorithms we may decrease the above processor bound in the dense case, as well as the asymptotic sequential time bound of Ref. [3], by a factor of $m^{0.335}$ (preserving the best asymptotic parallel time). In fact, improving or even combining the known fast matrix multiplication methods, see Refs [4–6] may lead to a further minor decrease in the processor bounds, but the latter decrease, as well as the above improvements of a factor of $m^{0.335}$, would hardly have any practical value due to the huge overhead of the asymptotically fast matrix multiplication methods and their inability to preserve sparsity. Our acceleration of Karmarkar's algorithm, however, is practical and most significant in the important case (arising, for instance, in the optimization of an economy consisting of several branches weakly connected to each other and in the multicommodity flow problem in a planar network for a fixed number of commodities, see Refs [7] or [8, p. 391]), where the input matrix of the LPP is large and sparse and is associated with graphs having a family of small separators (see the formal definitions below, in Section 3).

Our work has several further impacts. Similarly to the case of Karmarkar's algorithm [3], we may immediately improve the performance of several known algorithms, in particular of algorithms for systems of linear inequalities [9], for mathematical programming [10] and for sparse nonsymmetric systems of linear equations, because (as we indicated above) solving a system of linear equations constitutes a particular case of the LLSP where

$$\min_{\mathbf{x}} \| \mathbb{A}\mathbf{x} - \mathbf{b} \| = 0.$$

The latter observation leads to a very wide range of applications of our results, including in particular the acceleration of *the simplex algorithms* for a sparse LPP [cf. 11, 12]. Further applications may include several combinatorial computations. This is demonstrated in Ref. [13], where, relying on the latter improvement of the algorithms for sparse nonsymmetric systems of linear equations, we extend the parallel nested dissection algorithm of Ref. [2] to the path algebra computations.

We organize the paper as follows. In Section 2 we recall two known representations of the LLSP, using normal equations and their blow-up transformations. In Section 3 we reexamine the computational cost of sequential algorithms for the LLSP, in particular, we recall the sequential nested dissection algorithm of Ref. [14] and adjust it to the case of the LLSP. We also describe the variable-diagonal techniques for stabilization in solving the LLSP. In Section 4 we estimate the cost of performing our parallel algorithm for the same problem. In Section 5 we consider one of the major applications of our results, i.e. the acceleration of Karmarkar's [3] algorithm. In the Appendix we will briefly comment on the current estimates for the computational cost of solving the LPP.

2. THE LINEAR LEAST-SQUARES PROBLEM (LLSP)

We will use the known fact (see Ref. [1]) that the LLSP can be reduced to computing the solution $\mathbf{x}$ of the system of normal linear equations

$$\mathbb{A}^T\mathbb{A}\mathbf{x} = \mathbb{A}^T\mathbf{b}, \tag{1}$$

which can be reduced to the following system of linear equations in $\mathbf{s}$ and $\mathbf{y}$:

$$\mathbb{D}_1\mathbb{D}_1^T\mathbf{s} + \mathbb{D}_1\mathbb{A}\mathbb{D}_0\mathbf{y} = \mathbb{D}_1\mathbf{b}$$

and

$$\mathbb{D}_0^T\mathbb{A}^T\mathbb{D}_1^T\mathbf{s} = \mathbf{0},$$

or equivalently

$$\mathbb{H}\mathbf{v} = \mathbf{d}, \tag{2}$$

where

$$\mathbb{H} = \begin{bmatrix} \mathbb{D} & \mathbb{D}_1\mathbb{A}\mathbb{D}_0 \\ \mathbb{D}_0^T\mathbb{A}^T\mathbb{D}_1^T & \mathbb{O} \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \mathbf{s} \\ \mathbf{y} \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} \mathbb{D}_1\mathbf{b} \\ 0 \end{bmatrix}, \quad \mathbf{x} = \mathbb{D}_0\mathbf{y}, \ \mathbb{D} = \mathbb{D}_1\mathbb{D}_1^T;$$

$\mathbb{D}_0$ is an $n \times n$ matrix, $\mathbb{D}_1$ is an $m \times m$ matrix and $\mathbb{D}_0, \mathbb{D}_1$ are nonsingular. Here and hereafter $\mathbb{I}$, $\mathbb{W}^T$, $\mathbf{v}^T$, $\mathbb{O}$ and $\mathbf{0}$ denote the identity matrix, the transposes of a matrix $\mathbb{W}$ and of a vector $\mathbf{v}$, the null matrix and the null vector of appropriate sizes, respectively. Hereafter $\mathbb{W}^{-T}$ will denote the inverse of $\mathbb{W}^T$.

If we need to solve the linear system $\mathbb{A}\mathbf{x} = \mathbf{b}$ in the usual sense, then that system can be equivalently rewritten as $\mathbb{G}\mathbb{A}\mathbf{x} = \mathbb{G}\mathbf{b}$ for any nonsingular matrix $\mathbb{G}$. The latter system is equivalent to system (2), where, in this case, $\mathbb{D}$ can be any $m \times m$ matrix, not necessarily $\mathbb{D}_1\mathbb{D}_1^T$.

Remark 1

Even though the systems (1) and (2) are equivalent to each other, it is more convenient to apply some algorithms to system (2) than to system (1), particularly where $\mathbb{H}$ is more sparse and/or better structured than $\mathbb{A}^T\mathbb{A}$. [We will call the transition from system (1) to system (2) and similar transformations *blow-up transformations* of linear systems.] The simplest and the most customary choice for $\mathbb{D}_0$ and $\mathbb{D}_1$ in system (2) is the identity matrices $\mathbb{I}$; however, choosing appropriate diagonal matrices for $\mathbb{D}_0$ and $\mathbb{D}_1$ we may scale the rows and columns of the three blocks of $\mathbb{H}$ in order to stabilize some special algorithms for sparse linear systems (2), such as the nested dissection algorithm, see below. This stabilization can be combined with the customary techniques of threshold pivoting, used in sparse matrix computations at the stage of determining the elimination ordering [15] (see also Remark 3).

Remark 2

In some cases further equivalent transformations of systems (1) and (2) are effective. In particular the $m \times n$ matrix $\mathbb{A}^T$ of system (1) may take the form $\mathbb{A}^T = [\mathbb{B}^T\mathbb{C}^T]$, where the block $\mathbb{B}^T$ is a readily invertible $n \times n$ matrix and the block $\mathbb{C}^T$ is an $n \times (m - n)$ matrix. Then

$$\mathbb{A}^T\mathbb{A} = \mathbb{B}^T(\mathbb{I} + \mathbb{E}^T\mathbb{E})\mathbb{B}, \quad \mathbb{E} = \mathbb{C}\mathbb{B}^{-1},$$

so solving system (1) can be reduced to computing the vector $\mathbf{x} = \mathbb{B}^{-1}\mathbf{u}$, where $\mathbf{u}$ satisfies the system

$$(\mathbb{I} + \mathbb{E}^T\mathbb{E})\mathbf{u} = \mathbf{g}, \quad \mathbf{g} = \mathbb{B}^{-T}\mathbb{A}^T\mathbf{b}.$$

Solving the latter system can be reduced to computing the vectors $\mathbf{u}$ and $\mathbf{r}$ such that

$$\mathbf{r} = \mathbb{E}\mathbf{u}, \quad \mathbf{u} = -\mathbb{E}^T\mathbf{r} + \mathbf{g}.$$

This is a linear system in $\mathbf{r}$ and $\mathbf{u}$, which can be equivalently rewritten in the following four ways, including two blow-up transformations:

$$\mathbf{u} = (\mathbb{I} + \mathbb{E}^T\mathbb{E})^{-1}\mathbf{g}, \quad \mathbf{r} = \mathbb{E}\mathbf{u};$$

$$\mathbf{r} = (\mathbb{I} + \mathbb{E}\mathbb{E}^T)^{-1}\mathbb{E}\mathbf{g}, \quad \mathbf{u} = -\mathbb{E}^T\mathbf{r} + \mathbf{g};$$

$$\mathbb{G}\begin{bmatrix}\lambda\mathbf{r}\\ \mathbf{u}\end{bmatrix} = \begin{bmatrix}0\\ \lambda\mathbf{g}\end{bmatrix}, \quad \mathbb{G} = \begin{bmatrix}-(1/\lambda)\mathbb{I} & \mathbb{E}\\ \mathbb{E}^T & \lambda\mathbb{I}\end{bmatrix}, \quad \lambda \neq 0 \text{ is a scalar;} \tag{3}$$

and

$$\begin{bmatrix}-\mathbb{I} & \mathbb{C}\\ \mathbb{C}^T & \mathbb{B}^T\mathbb{B}\end{bmatrix}\begin{bmatrix}\mathbf{r}\\ \mathbb{B}^{-1}\mathbf{u}\end{bmatrix} = \begin{bmatrix}0\\ \mathbb{B}^T\mathbf{g}\end{bmatrix}. \tag{4}$$

Thus the solution can be reduced to linear systems with coefficient matrices of sizes $n \times n$ or $(m-n) \times (m-n)$ or $m \times m$ [note that $m-n$ can be much smaller than n, also note the sparsity of the $m \times m$ matrices of systems (3) and (4), provided that $\mathbb{E}$ and/or $\mathbb{C}$ are sparse]. Scaling can be also extended to all four of the above systems [we explicitly showed only scaling by λ in system (3)].

3. SEQUENTIAL COMPUTATIONAL COMPLEXITY OF THE LLSP

For an LLSP with a dense matrix $\mathbb{A}$, its solution can be obtained from system (1) using $O(m/n)M(n)$ arithmetic operations where $M(n)$ is the cost of $n \times n$ matrix multiplication, $M(n) \leqslant 2n^3 - n^3$. Theoretically $M(n)$ is at least as small as $o(n^{2.496})$, but this bound is not practical due to the huge overhead constants hidden in the "o", [6].

If the matrix $\mathbb{A}$ is sparse, the solution can be accelerated using some special methods, see Ref. [16]. In particular, applying the conjugate gradient method or the Lanczos method [16,1] we may reduce the cost of solving both system (I) and (consequently) an LLSP to $O(mN(\mathbb{A}))$ arithmetic operations where $N(\mathbb{A})$ is the number of nonzero entries of $\mathbb{A}$, provided that the multiplication by 0 and the addition of 0 are cost-free operations.

We will single out a more specific case encountered in many practical instances of the LLSP, i.e. in the instances where the matrix $\mathbb{A}$ is sparse and where, furthermore, the graph $G = (V,E)$ associated with the matrix $\mathbb{H}$ has an $s(m+n)$-separator family with $s(m+n) = o(m+n)$. (Hereafter we will assume that $s(k) \geqslant \sqrt{k}$.) Here and hereafter we apply the following definitions, which we reproduce from Section 1.2 of Ref. [2] (cf. also Ref. [14]).

Definition 1

Let $\mathscr{G}$ be a class of undirected graphs closed under the subgraph relation, i.e. if $G \in \mathscr{G}$ and G' is a subgraph of G, then $G' \in \mathscr{G}$. The class $\mathscr{G}$ is said to have a *dense family of $s(n)$-separators* or, simply, *an $s(n)$-separator family* if there exist constants $n_0 > 0$ and $\alpha, 0 < \alpha < 1$, such that for each graph $G \in \mathscr{G}$ with $n \geqslant n_0$ vertices there is a partition V_1, V_2, S of the vertex set of G such that $|V_1| \leqslant \alpha n$, $|V_2| \leqslant \alpha n$, $|S| \leqslant s(n)$ and G has no edge from a vertex of V_1 to V_2 [then S is said to be an *$s(n)$-separator* of G]. An undirected graph is said to *have an $s(n)$-separator family* if the class of all its subgraphs has an $s(n)$-separator family.

Binary trees obviously have a 1-separator family. A d-dimensional grid (of uniform size in each dimension) has an $n^{1-(1/d)}$-separator. Lipton and Tarjan [17] show that the planar graphs have a $\sqrt{8n}$-separator family and that every n-vertex finite-element graph with $\leqslant k$ boundary vertices in every element has a $4\lceil k/2 \rceil \sqrt{n}$-separator.

Definition 2

Given a $k \times k$ symmetric matrix $\mathbb{W} = [w_{ij}]$, we define $G(\mathbb{W}) = (V,E)$ to be the undirected graph with vertex set $V = \{1, \ldots, k\}$ and edge set $E = \{\{i,j\} \mid w_{ij} \neq 0\}$.

The very large linear systems $\mathbb{A}\mathbf{x} = \mathbf{b}$ that arise in practice often have graphs $G(\mathbb{A})$ with small separators. Important examples of such systems can be found in circuit analysis (e.g. in the analysis of the electrical properties of a VLSI circuit), in structural mechanics (e.g. in the stress analysis of large structures) and in fluid mechanics (e.g. in the design of airplane wings and in weather prediction).

When the associated graph G of the matrix $\mathbb{H}$ of system (2) has an $s(m+n)$-separator family, the application of the techniques of nested dissection [16, p. 182; 18, 14] decreases the cost of the solution of system (2), and consequently of the original LLSP, to $O(|E| + M(s(m+n)))$ arithmetic operations where $|E|$ is the cardinality of the edge set of G [14]. This is the cost of computing the $\mathbb{L}\mathbb{D}\mathbb{L}^T$-factorization of $\mathbb{H}$; this cost is much lower than $M(m+n)$, the cost in the case of dense $\mathbb{H}$ [1]. The subsequent evaluation of the vectors $\mathbf{r},\mathbf{x}$ satisfying system (2) costs $O(|E| + [s(m+n)]^2)$ arithmetic operations [14], so the approach is particularly effective for solving several systems (1) with fixed $\mathbb{A}$ and variable $\mathbf{b}$. To see the potential advantage of using the nested dissection algorithm, assume that $m = O(n)$ and that the associated graph G of $\mathbb{H}$ is planar. Then G has $O(\sqrt{n})$-separators [17] and $|E| = O(n)$, so computing the $\mathbb{L}\mathbb{D}\mathbb{L}^T$-factorization of $\mathbb{H}$ costs $O(n^{1.5})$ arithmetic operations, and the subsequent solution of system (1) costs only $O(n)$ for every fixed vector $\mathbf{b}$, compared with the $O(n^3)$ arithmetic operations required for the solution if the sparsity is not exploited and with the $O(n^2)$ arithmetic operations required by the conjugate gradient and Lanczos methods [1, 16].

Remark 3

Systems (2) and (3) are not positive definite, so the nested dissection algorithm, if applied to such systems, may involve destabilizing elimination steps, characterized by the small magnitudes of pivot elements. Scaling and threshold pivoting (see Remark 1) can be partial remedies. We also suggest the following combination of the modified method of *variable diagonals* [19] (see also the end of Section 4 below) with blow-up transformations and the Morrison–Sherman–Woodbury formula, see equation (5) below. Namely, whenever a pivot entry of small magnitude appears during the elimination process, we increase the respective diagonal entry (i,i) of the matrix $\mathbb{H}$ of system (2) [similarly for the matrices of systems (3) and (4)] by adding a large value $k^2(i)$; we then continue the computation. Finally, we compute the $\mathbb{L}\mathbb{D}\mathbb{L}^T$-factorization of the matrix $\mathbb{S} = \mathbb{H} + \mathbb{K}^2$. Here the matrix $\mathbb{K}$ is filled with zeros, except for the j diagonal entries corresponding to the changes in the diagonal pivot entries of $\mathbb{H}$. In these places $\mathbb{K}$ is filled with the values $k(i)$. We will consider the case where j, the total number of corrections to the pivot entries of $\mathbb{H}$, is relatively small. Then the system (2) can be effectively solved using the computed $\mathbb{L}\mathbb{D}\mathbb{L}^T$-factorization of $\mathbb{S} = \mathbb{H} + \mathbb{K}^2$ and the Sherman–Morrison–Woodbury formula [1, p. 3]:

$$(\mathbb{S} - \mathbb{U}\mathbb{V})^{-1} = \mathbb{S}^{-1} + \mathbb{S}^{-1}\mathbb{U}(\mathbb{I} - \mathbb{V}\mathbb{S}^{-1}\mathbb{U})^{-1}\mathbb{V}\mathbb{S}^{-1}, \tag{5}$$

which holds for arbitrary matrices $\mathbb{S}$, $\mathbb{U}$ and $\mathbb{V}$ of appropriate size such that $\mathbb{I} - \mathbb{V}\mathbb{S}^{-1}\mathbb{U}$ is a nonsingular matrix. In our case $\mathbb{U}\mathbb{V}$ is the diagonal matrix $\mathbb{K}^2$; this enables us to simplify the computations. We have $\mathbb{S} = \mathbb{H} + \mathbb{K}^2$ and $\mathbb{H} = \mathbb{S} - \mathbb{K}^2$. Let $\mathbb{U} = \mathbb{V} = \mathbb{K}$, so $\mathbb{H} = \mathbb{S} - \mathbb{U}\mathbb{V}$, and therefore [see equations (2) and (5)]

$$\mathbf{v} = \mathbb{H}^{-1}\mathbf{c} = (\mathbb{S} - \mathbb{U}\mathbb{V})^{-1}\mathbf{c} = \mathbb{S}^{-1}\mathbf{c} + \mathbb{S}^{-1}\mathbb{V}(\mathbb{I} - \mathbb{V}\mathbb{S}^{-1}\mathbb{V})^{-1}\mathbb{V}\mathbb{S}^{-1}\mathbf{c}.$$

Let us examine the evaluation of $\mathbf{v}$, assuming for simplicity that all the nonzero entries of $\mathbb{K}$ lie in the first j rows. Since the $\mathbb{L}\mathbb{D}\mathbb{L}^T$-factors of $\mathbb{S}$ have been computed, solving linear systems with the matrix $\mathbb{S}$ is simple. Computing the $j \times j$ upper-left submatrix $\mathbb{T} = \mathbb{V}\mathbb{S}^{-1}\mathbb{V}$ of $\mathbb{S}^{-1}$ is reduced to computing the first j rows of $\mathbb{V}\mathbb{L}^{-T}$ and to computing the product $(\mathbb{V}\mathbb{L}^{-T})\mathbb{D}^{-1}(\mathbb{L}^{-1}\mathbb{V})$. When j is not large, this computation is simple, as well as the subsequent solution of a linear system with the matrix $\mathbb{I} - \mathbb{T}$. The computation does not require storage of the matrix $\mathbb{V}\mathbb{L}^{-T}$. To confine the pivot corrections to the left-upper block of $\mathbb{H}$ (this should simplify computing $\mathbb{V}\mathbb{L}^{-T}$), we may scale systems (2)–(4) [say by choosing a small positive λ in system (3)].

4. PARALLEL ALGORITHMS FOR LLSP

For large input matrices $\mathbb{A}$, the sequential algorithms for the LLSP can be prohibitively slow. Their dramatic acceleration that preserves their efficiency can be obtained using the recent parallel algorithms of Ref. [2], where in each step every processor may perform one arithmetic operation. Specifically, before Ref. [2] appeared, the best algorithms for solving a linear system with an $n \times n$ dense matrix $\mathbb{A}$ either (i) were unstable and required $O(\log^2 n)$ parallel steps and $\geqslant \sqrt{n}\, M(n)$ processors or (ii) involved $\geqslant n$ steps and n^2 processors. (Here and hereafter the numbers of processors are defined within constant factors for we may save processors using more steps. Practically this means that the user, having, say, k times less processors than in our subsequent estimates, may still use our algorithms; the parallel time, even increased by a factor of k, may still be attractively small for that user.) The stable iterative algorithm of Ref. [2], based on Newton's iteration for the matrix equation $\mathbb{I} - \mathbb{X}\mathbb{A} = 0$, requires only $O(\log^2 n)$ steps and $M(n)/\log n$ processors to compute the solution of such a dense system (with the relative error norm bounded, say by $1/2^{n^{100}}$), provided that the system has a well-conditioned or strongly diagonally dominant matrix. (In fact the algorithm even inverts the matrix of the given system for the above parallel cost.) The algorithm successively computes $t = \|\mathbb{A}\|_1 \|\mathbb{A}\|_\infty$, $\mathbb{B}_0 = (1/t)\mathbb{A}^T$, $\mathbb{B}_{k+1} = 2\mathbb{B}_k - \mathbb{B}_k\mathbb{A}\mathbb{B}_k$, $k = 0, 1, \ldots, q$. $\mathbb{B}_q$ is shown to be a very high precision approximation to $\mathbb{B}^{-1}$ if $q = O(\log n)$ and if $\operatorname{cond}(\mathbb{A})$ is bounded by a polynomial in n (similarly, if $\mathbb{B}$ is strongly diagonally dominant. The desired estimates for the parallel complexity of solving dense linear systems immediately follow. Applying the cited algorithm to system (1), we solve the original LLSP using $O(\log m + \log^2 n)$ steps and $[M(n)/\log n][1 + m/(n \log n)]$ processors. These are the bounds in the case where $\mathbb{A}$ is a general (dense) matrix. Another parallel algorithm of Ref. [2] is applied in cases of practical interest, where $\mathbb{A}$ is sparse and the graph $G = (V,E)$ of $\mathbb{H}$ has an $s(m+n)$-separator family, see Definitions 1 and 2. In this case the parallel nested dissection algorithm of Ref. [2] computes a special recursive $s(m+n)$-factorization of the matrix $\mathbb{H}$ of systems (2) and (3) using $O(\log m \log^2 s(m+n))$ parallel steps and $|E| + M(s(m+n))/\log s(m+n)$ processors, the observations of Remarks 1 and 3 are still applied. Following Definition 4.1 of Ref. [2], we define such a recursive $s(m+n)$-factorization of $\mathbb{H}$ as a sequence of matrices $\mathbb{H}_0, \mathbb{H}_1, \ldots, \mathbb{H}_d$ such that $\mathbb{H}_0 = \mathbb{P}\mathbb{H}\mathbb{P}^T$, $\mathbb{P}$ is an $(m+n) \times (m+n)$ permutation matrix,

$$\mathbb{H}_g = \begin{bmatrix} \mathbb{X}_g & \mathbb{Y}_g^T \\ \mathbb{Y}_g & \mathbb{Z}_g \end{bmatrix} \quad \text{and} \quad \mathbb{Z}_g = \mathbb{H}_{g+1} + \mathbb{Y}_g \mathbb{X}_g^{-1} \mathbb{Y}_g^T \tag{6}$$

for $g = 0, 1, \ldots, d-1$, and $\mathbb{X}_g$ is a block diagonal matrix consisting of square blocks of size $s(\alpha^{d-g}(m+n)) \times s(\alpha^{d-g}(m+n))$ at most, where $\alpha^d(m+n) \leqslant n_0$ for constants n_0 and α of Definition 1. The latter inequality implies that the factorization (6) has length $d = O(\log m)$, so the computation of factorization (6) is reduced to $O(\log m)$ parallel steps of matrix multiplication and inversion vs $m+n$ such steps in the sequential nested dissection algorithms, required to compute the $\mathbb{L}\mathbb{D}\mathbb{L}^T$-factorization. The dense blocks of $\mathbb{X}_g$ [of size $s(\alpha^{d-g}n \times \alpha^{d-g}n)$] are inverted by the cited parallel algorithm of Ref. [2] for matrix inversion. This enables us to keep the total cost of computing the recursive factorization (6) as low as stated.

Observe that the definition of a recursive $s(n)$-factorization implies the following identities for $g = 0, \ldots, d-1$:

$$\mathbb{H}_g = \begin{bmatrix} \mathbb{I} & \mathbb{O} \\ \mathbb{Y}_g \mathbb{X}_g^{-1} & \mathbb{I} \end{bmatrix} \begin{bmatrix} \mathbb{X}_g & \mathbb{O} \\ \mathbb{O} & \mathbb{H}_{g-1} \end{bmatrix} \begin{bmatrix} \mathbb{I} & \mathbb{X}_g^{-1}\mathbb{Y}_g^T \\ \mathbb{O} & \mathbb{I} \end{bmatrix}$$

and, hence

$$\mathbb{H}_g^{-1} = \begin{bmatrix} \mathbb{I} & -\mathbb{X}_g^{-1}\mathbb{Y}_g^T \\ \mathbb{O} & \mathbb{I} \end{bmatrix} \begin{bmatrix} \mathbb{X}_g^{-1} & \mathbb{O} \\ \mathbb{O} & \mathbb{H}_{g-1}^{-1} \end{bmatrix} \begin{bmatrix} \mathbb{I} & \mathbb{O} \\ -\mathbb{Y}_g \mathbb{X}_g^{-1} & \mathbb{I} \end{bmatrix}.$$

This reduces solving linear systems with matrix $\mathbb{H}_g$ to solving linear systems with matrices $\mathbb{X}_g$ and $\mathbb{H}_{g+1}$ and finally implies that, although the recursive factorization (6) is distinct from the more customary $\mathbb{L}\mathbb{D}\mathbb{L}^T$-factorization used in the sequential algorithms, both have similar power, i.e. when the recursive factorization (6) is available, $O((\log m)[\log s(m+n)])$ parallel steps and $|E| + [s(m+n)]^2$ processors suffice in order to solve system (2) and consequently the original LLSP. In Ref. [2], the partition of $\mathbb{H}_g$ in factorization (6) for $g = 0, 1, \ldots, d-1$ is defined by appropriate enumeration of the vertices of the graph G. The enumeration, the study of the block diagonal structure of the matrices $\mathbb{X}_g$ and the complexity estimates rely on extensive exploitation of the properties of the graph G stated in Definitions 1 and 2.

Comparing the cost bounds of Refs [14] and [2], we can see that the parallelization is efficient, i.e. the production of the two upper bounds on the numbers of steps and processors of Ref. [2] is equal (within a polylogarithmic factor) to the bound on the number of arithmetic operations in the current best sequential algorithm of Ref. [14] for the same problem. The same efficiency criterion is satisfied in the algorithms of Ref. [2] inverting an $n \times n$ dense matrix in $O(\log^2 n)$ parallel steps using $M(n)/\log n$ processors. Consequently, all our parallel algorithms for an LLSP are also efficient.

The complexity estimates of Ref. [2] have been established in the case of well-conditioned input matrices; the algorithms of Ref. [2] output the approximate solutions with a sufficiently high precision. On the other hand, all the estimates have been extended to the case of an arbitrary integer input matrix $\mathbb{A}$ in Refs [19, 20] by using some different techniques, in particular, using variable diagonals. In this case the solutions are computed exactly, although that computation generally involves larger numbers, such as the determinant of the input matrix, det $\mathbb{A}$. This algorithm exactly computes, first det $\mathbb{A}$ and adj $\mathbb{A}$ and then $\mathbb{A}^{-1} = \text{adj } \mathbb{A}/\det \mathbb{A}$ and $\mathbb{A}^{-1}\mathbf{b} = (\text{adj } \mathbb{A})\,\mathbf{b}/\det \mathbb{A}$. If only a system $\mathbb{A}\mathbf{x} = \mathbf{b}$ with an $n \times n$ integer matrix $\mathbb{A}$ need be solved, only (adj $\mathbb{A}$) $\mathbf{b}$ (rather than adj $\mathbb{A}$) need be computed. The evaluation of det $\mathbb{A}$ in Refs [19, 20] is reduced to computing the Krylov matrix, $\mathbb{K} = [\mathbf{v}, \mathbb{A}\mathbf{v}, \ldots, \mathbb{A}^{n-1}\mathbf{v}]$, the vector $\mathbb{A}^n\mathbf{v}$, $\mathbf{v} = [1, 0, \ldots, 0]^T$, and the exact solution of the linear system, $\mathbb{K}\mathbf{y} = \mathbb{A}^n\mathbf{v}$. The solution vector $\mathbf{y} = [y_i]$ is the coefficient vector of the characteristic polynomial of $\mathbb{A}$, $\det |\lambda \mathbb{I} - \mathbb{A}|$, so $y_0 = \det \mathbb{A}$ and all the entries of $\mathbf{y}$ are integers. At these stages, $O(\log^2 n)$ parallel steps and $M(n)$ processors suffice, provided that $\mathbb{K}$ is strongly diagonally dominant, because we may use the algorithm of Ref. [2] in order to compute the integer solution vector $\mathbf{y}$ with the absolute error norm bound, say 1/3; then we may obtain $\mathbf{y}$ exactly by rounding-off. To make the matrix $\mathbb{K}$ strongly diagonally dominant, first we replace $\mathbb{A}$, say by $\mathbb{A} + p\mathbb{I}$ or, more generally, by a matrix $\mathbb{W}$ such that $\mathbb{W} = \mathbb{A} \bmod p$, so $\det \mathbb{W} = \det \mathbb{A} \bmod p$ is computed. Then using Newton–Hensel's lifting, we compute $\det \mathbb{W} = \det \mathbb{A} \bmod p^s$, where $2|\det \mathbb{A}| < p^s$, $s = 2^h$, $h = O(\log n)$, so we may recover det $\mathbb{A}$. Similarly, we compute adj $\mathbb{A}$ or (adj $\mathbb{A}$)$\mathbf{b}$. In the worst case this construction requires choosing p as large as n^n. However, with probability $1 - \epsilon(n)$, $\epsilon(n) \to 0$ as $n \to \infty$, it suffices to choose p to be a prime of an order of $O([n \| \mathbb{A} \|]^{3.1})$ and to define both the Krylov matrix and the vector $\mathbb{W}^n\mathbf{v} \bmod p^s$. Another approach leads to slightly inferior (with the time bound increased by a factor of $\log n$) but deterministic estimates. It relies on computing the $\mathbb{L}\mathbb{U}$-factors of $\mathbb{A}$, see Ref. [20].

5. KARMARKAR'S ALGORITHM. PARALLELIZATION. APPLICATION TO SPARSE LP

In this section we will examine the cost of Karmarkar's linear programming algorithm [3] and of its modifications that use blow-up transformations, nested dissection and parallelization. First we will reproduce the algorithm, which solves the problem of the minimization of the linear function $\mathbf{c}^T\mathbf{y}$ subject to the constraints

$$\mathbb{A}^T\mathbf{y} = \mathbf{0}, \quad \sum_j y_j = 1, \quad \mathbf{y} \geqslant \mathbf{0}, \tag{7}$$

where $\mathbf{y} = [y_j, j = 0, 1, \ldots, m-1]$ and $\mathbf{c}$ are m-dimensional vectors, $\mathbb{A}^T$ is an $n \times m$ matrix, $m \geq n$, $\mathbf{y}$ is unknown. This version is equivalent to the canonical LPP of the minimization of $\mathbf{c}^T\mathbf{y}$ subject to $\mathbb{A}^T\mathbf{y} \leq \mathbf{b}, \mathbf{y} \geq \mathbf{0}$, see Ref. [3] and cf. Refs [11, 12]. We will designate

$$\mathbf{e} = [1, 1, \ldots, 1]^T,$$

$$\mathbf{y}(i) = [y_0(i), y_1(i), \ldots, y_{m-1}(i)]^T,$$

$$\mathbb{D}(0) = \mathbb{I}, \quad \mathbb{D}(i) = \operatorname{diag}(y_0(i), y_1(i), \ldots, y_{m-1}(i)),$$

and

$$\mathbb{B}^T = \mathbb{B}^T(i) = \begin{bmatrix} \mathbb{A}^T \mathbb{D}(i) \\ \mathbf{e}^T \end{bmatrix}, \quad i = 0, 1, \ldots. \tag{8}$$

(All the diagonal matrices $\mathbb{D}(i)$ encountered in the algorithm of Ref. [3] are positive definite.) The algorithm proceeds as follows.

Initialize. Choose $\epsilon > 0$ (prescribe tolerance) and a parameter β (in particular, β can be set equal to 1/4). Let $\mathbf{y}(0) = (1/n)\mathbf{e}$, $i = 0$.

Recursive step. While nonoptimal $y(i)(\mathbf{c}^T\mathbf{y}(i) > \epsilon)$ and while the infeasibility tests fail do

Compute the vector $\mathbf{y}(i+1) = \gamma(\mathbf{y}(i))$, increment i.

Given vector $\mathbf{y}(i)$, the vector $\mathbf{y}(i+1)$ is computed as follows:

1. Compute the matrix $\mathbb{B} = \mathbb{B}(i)$, compare (8), i.e. compute the matrix $\mathbb{A}^T\mathbb{D}(i)$ and augment it by appending the row $\mathbf{e}^T$.
2. Compute the vector $\mathbf{c}_p = [\mathbb{I} - \mathbb{B}(\mathbb{B}^T\mathbb{B})^{-1}\mathbb{B}^T]\mathbb{D}(i)\mathbf{c}$.
3. Compute the vector $\mathbf{z}(i) = \mathbf{y}(i) - \beta r \mathbf{c}_p / \|\mathbf{c}_p\|$, where $r = 1/\sqrt{m(m-1)}$.
4. Compute the vector $\mathbf{y}(i+1) = \mathbb{D}(i)\mathbf{z}(i)/[\mathbf{e}^T\mathbb{D}(i)\mathbf{z}(i)]$.

The algorithm includes checks for infeasibility and optimality [3], but it is easy to verify that their computational cost, as well as the computational cost of reducing the problem from the canonical form to that of equations (7), is dominated by the cost of computing the vector $\gamma(\mathbf{y}(i))$ at the recursive steps which is, in turn, dominated by the cost of computing $(\mathbb{B}^T\mathbb{B})^{-1}$ given $\mathbb{B}^T = \mathbb{B}^T(i)$ for all i. Reference [3] shows that $\mathbb{B}^T\mathbb{B}$ can be represented as follows:

$$\mathbb{B}^T\mathbb{B} = \begin{bmatrix} \mathbb{A}^T\mathbb{D}^2(i)\mathbb{A} & \mathbf{0} \\ \mathbf{0}^T & m \end{bmatrix},$$

so the inversion of $\mathbb{B}^T\mathbb{B}$ is reduced to the inversion of $\mathbb{A}^T\mathbb{D}^2(i)\mathbb{A}$ which, in turn, is reduced to the inversion of the matrix $\mathbb{H}$ of system (2) where $\mathbb{A}$ is replaced by $\mathbb{D}(i)\mathbb{A}$. Furthermore, we can see that it suffices to compute the product $(\mathbb{B}^T\mathbb{B})^{-1}\mathbb{B}\mathbb{D}(i)\mathbf{c}$, and this amounts to matrix × vector multiplications and to solving a blown-up linear system of form (2) with the matrix

$$\mathbb{H} = \mathbb{H}(i) = \begin{bmatrix} \mathbb{D}^{-2}(i) & \mathbb{A} \\ \mathbb{A}^T & \mathbb{O} \end{bmatrix}. \tag{9}$$

This algorithm of Ref. [3] requires $O(Lm)$ recursive steps in the worst case, so the total computational cost is $O(Lm\,C)$, where L is the input size of the problem and C is the cost of computing $\gamma(\mathbf{y})$ given $\mathbf{y}$. The algorithm for the incremental computation of the inverse of $\mathbb{B}^T\mathbb{B}$ of Section 6 of Ref. [3] implies that $C = O(m^{2.5})$ for the dense matrix $\mathbb{A}$. It is rather straightforward to perform these $O(m^{2.5})$ arithmetic operations in parallel using $O(\sqrt{m}\log m)$ steps and $m^2/\log m$ processors (and using $O(m)$ steps and m^2 processors for the initial inversion of $\mathbb{A}^T\mathbb{A}$). Applying the matrix inversion algorithms of Ref. [2], we may perform every evaluation of $\gamma(\mathbf{y})$ using $O(\log m + \log^2 n)$ parallel arithmetic steps and $[M(n)/\log n][1 + m/(n \log n)]$ processors, so we arrive at the following trade-off for the estimated total arithmetic cost of Karmarkar's algorithm: $O(m^{1.5}L)$ steps and m^2 processors, i.e. $O(m^{3.5}L)$ arithmetic operations (via the straightforward parallelization); or

$O(mL(\log m + \log^2 n))$ steps and $[M(n)/\log n][1 + m/(n \log n)]$ processors, i.e. $O(mLM(n)(\log m + \log^2 n)[1 + m/(n \log n)]/\log n)$ arithmetic operations (via the parallel matrix inversion algorithms of Ref. [2]).

In both cases the sparsity of $\mathbb{A}$ is not exploited. In particular, the algorithm for the incremental computation of the inverse suggested in Section 6 of Ref. [3] does not preserve the sparsity of the original input matrix. This causes some problems in practical computations, because the storage space increases substantially. Thus, the special methods of solving sparse LLSPs, such as the conjugate gradient [16], the Lanczos [1] and the nested dissection methods (this paper), become competitive with (if not superior to) the latter algorithm of Section 6 of Ref. [3]. If the matrix $\mathbb{A}$ is such that the graph $G = (V,E)$ of the matrices $\mathbb{H}$ of equation (9) has an $s(m+n)$-separator family and $s(m+n) = o(m+n)$, then the nested dissection method can be strongly recommended. Specifically, in this case we arrive at the estimates of $O(Lm[|E| + M(s(m+n))])$ arithmetic operations for solving the LPP by combining Refs [3] and [14], see Section 3, and of $O(Lm \log m \log^2 s(m+n))$ parallel arithmetic steps and $O(|E| + M(s(m+n))/\log s(m+n))$ processors, by combining Ref. [3] and the parallel algorithm of this paper. The reader could better appreciate this improvement, due to the application of nested dissection, if we recall that $s(m+n) = \sqrt{(m+n)}$, where the graph G is planar [as occurs in many operations research applications, for instance, in the problem of computing the maximum flow in a network having an $s(m+n)$-separator family]. Then the processor bound for computing the recursive factorization (6) is less than $2s^3(m+n) = 8\sqrt{8}(m+n)^{1.5}$ and the total number of arithmetic operations is $O(mL(m+n)^{1.5})$. The premultiplications of $\mathbb{A}$ by the nonsingular matrix $\mathbb{D}(i)$ do not change the separator sets for the graph G, so these sets are precomputed once and for all, which is an additional advantage of using nested dissection in this case.

Finally, we apply fast matrix multiplication algorithms in order to decrease the known theoretical upper bounds on the complexity of solving LPPs with a dense input matrix from the $O(m^{3.5}L)$ arithmetic operations of Ref. [3] to $O(m^{\beta}L)$, where $\beta < 3.165$. Recall that iteration i of Ref. [3] can be reduced to inverting the matrix $\mathbb{H}(i)$ of equation (9); furthermore, the diagonal matrix $\Delta(i) = \mathbb{D}^{-2}(i) - \mathbb{D}^{-2}(i-1)$ has at most $j = O(\sqrt{m})$ nonzero entries for each i, $i = 1, 2, \ldots$. Let us apply formula (5) in order to compute $\mathbb{H}^{-1}(i)$ given $\mathbb{H}^{-1}(i-1)$. Similarly to the stabilization process of Remark 3, let $\mathbb{U} = \mathbb{V}$ be a diagonal matrix with, at most, j nonzero entries, such that

$$\mathbb{U}\mathbb{V} = \begin{bmatrix} \Delta(i) & \mathbb{O} \\ \mathbb{O} & \mathbb{O} \end{bmatrix}.$$

The computation is reduced to the inversion of a $j \times j$ submatrix of $\mathbb{I} - \mathbb{V}\mathbb{H}^{-1}(i-1)\mathbb{V}$ and to two rectangular matrix multiplications, of size $m \times j$ by $j \times j$ and $m \times j$ by $j \times m$, see formula (5). Thus the entire arithmetic cost of one iteration of Ref. [3] is dominated by the arithmetic cost, $M(m,j,m)$, of the $m \times j$ by $j \times m$ matrix multiplication. Respectively, the cost of solving the LPP is $O(LmM(m,j,m))$ arithmetic operations of $O(Lm \log^2 m)$ parallel steps and $M(m,j,m)$ processors, where $j = O(\sqrt{m})$, $M(m,j,m) = O(m^{\beta})$. Surely $\beta \leqslant 2.25$, for $M(m,j,m) = M(j)(m/j)^2 = O(m^{2.25})$, if $j = O(\sqrt{m})$, but in fact β is upper bounded by 2.165 [4, p. 108; cf. 5]. This implies the sequential time bound $O(Lm^{3.165})$ and (see Appendix A of Ref. [2]) the processor bound $M(m,j,m) = O(m^{2.165})$ [with the parallel time $O(Lm \log^2 m)$] on the complexity of the LPP. As we have mentioned, large overhead makes fast matrix multiplication algorithms nonpractical. Note, however, that even with straightforward matrix multiplication $M(m,j,m) = m^2(2j-1)$, which implies decreasing the parallel cost of one iteration of Ref. [3] to $O(\log^2 m)$ parallel steps and $m^{2.5}$ processors (using as many iterations as in Ref. [3], i.e. $O(Lm)$).

Remark 4

The latter asymptotic complexity estimates (but with double overhead) could be deduced relying on the inversion of $\mathbb{A}^T\mathbb{D}(i)\mathbb{A}$.

Acknowledgements—V. Pan gratefully acknowledges the support given by NSF Grants MCS 8203232 and DCR 8507573. J. Reif was supported by the Office of Naval Research, Contract No. N00014-80-C-0647.

REFERENCES

1. G. H. Golub and C. F. van Loan, *Matrix Computations*. Johns Hopkins Univ. Press, Baltimore, Md (1983).
2. V. Pan and J. Reif, Fast and efficient parallel solution of linear systems. Technical Report TR-02-85, Center for Research in Computer Technology, Aiken Computation Lab., Harvard Univ., Cambridge, Mass. (1985). (Short version in *Proc. 17th A. ACM STOC*, Providence, R.I., pp. 143–152.)
3. N. K. Karmarkar, A new polynomial time algorithm for linear programming. *Combinatorica* **4**, 373–395 (1984).
4. P. A. Gartenberg, Fast rectangular matrix multiplication. Ph.D. Thesis, Dept. of Mathematics, Univ. of California, Los Angeles, Calif. (1985).
5. G. Lotti and F. Romani, On the asymptotic complexity of rectangular matrix multiplication. *Theor. Comput. Sci.* **23**, 171–185 (1983).
6. V. Pan, How to multiply matrices faster. *Lecture Notes in Computer Science*, Vol. 179. Springer, Berlin (1984).
7. M. Gondran and M. Minoux, *Graphs and Algorithms*. Wiley-Interscience, New York (1984).
8. K. G. Murty, *Linear and Combinatorial Programming*. Wiley, New York (1976).
9. V. Pan, Fast finite methods for a system of linear inequalities. *Comput. Math. Applic.* **11**, 355–394 (1985).
10. N. Z. Shor, New development trend in nondifferentiable optimization. *Cybernetics* **13**, 881–886 (1977).
11. V. Chvatal, *Linear Programming*. Freeman, San Francisco, Calif. (1983).
12. K. G. Murty, *Linear Programming*. Wiley, New York (1983).
13. V. Pan and J. Reif, Extension of the parallel nested dissection algorithm to the path algebra problems. Technical Report 85-9, Computer Science Dept, SUNY, Albany, N.Y. (June 1985).
14. R. Lipton, D. Rose and R. E. Tarjan, Generalized nested dissection. *SIAM Jl numer. Analysis* **16**, 346–358 (1979).
15. S. Pissanetsky, *Sparse Matrix Technology*. Academic Press, New York (1984).
16. Å Björck, Methods for sparse linear least squares problems. In *Sparse Matrix Computations* (Edited by J. R. Bunch and D. J. Rose), pp. 177–200. Academic Press, New York (1976).
17. R. J. Lipton and R. E. Tarjan, A separator theorem for planar graphs. *SIAM Jl appl. Math.* **36**, 177–189 (1979).
18. J. A. George, Nested dissection of a regular finite element Mesh. *SIAM Jl numer. Analysis* **10**, 345–367 (1973).
19. V. Pan, Fast and efficient algorithms for the exact inversion of integer matrices. In *Proc. 5th Conf. on Foundations of Software Engineering and Theoretical Computer Science*, Indian Institute of Technology/Tata Institute of Fundamental Research, New Delhi, pp. 504–521 (Dec. 1985).
20. V. Pan, Parallel complexity of polylogarithmic time matrix computations. Technical Report 86-14, Computer Science Dept, SUNY, Albany, N.Y. (April 1986).
21. V. Pan, On the complexity of a pivot step of the revised simplex algorithm. *Comput. Math. Applic.* **11**, 1127–1140 (1985).
22. J. Renegar, A polynomial-time algorithm, based on Newton's method, for linear programming. Technical Report 07118-86, MSRI, Berkeley, Calif. (1986).

APPENDIX

Current Computational Cost of Solving the LPP

In Table A1 below we display the estimates for the computational cost of one iteration of the simplex and Karmarkar's algorithms for the LPP having a dense $m \times n$ input matrix $\mathbb{A}$ [cf. 21]. We will restrict our analysis to the cases where $n \leqslant m = O(n)$. As in Ref. [21], we will not use the possible accelerations based on fast matrix multiplication, instead we will apply the results of Refs [19, 2] and the improvement of Karmarkar's algorithm from the end of Section 5.

There is a certain amount of controversy about the current upper estimates for the number of iterations in the two cited algorithms. The worst-case upper bounds, $O(Lm)$ for Ref. [3] and 2^m for the simplex algorithms, greatly exceed the number of iterations required when the same algorithms run in practice or use random input instances. This uncertainty complicates the theoretical comparison of the effectiveness of the two algorithms. However, a preliminary comparison can be based on the partial information already available. In particular, let us assume the empirical upper bound $O(n \log m)$ on the number of iterations (pivot steps) of the simplex algorithms, cited by some authors who refer to decades of practical computation [11, pp. 45–46; 12, p. 434]. The bound implies that a total of $O(m^3 \log m)$ arithmetic operations suffice in the simplex algorithm vs the $O(m^3)$ already used in the first iteration of Ref. [3]. Moreover, there are special methods that efficiently update the triangular factorization of the basis matrices used in the simplex algorithms, which further simplifies each iteration of the simplex algorithms in the case of sparse input matrices [11, Chaps 7, 24; 12, Chap. 7]. On the

Table A1

	Arithmetic operations	Parallel steps	Processors
First iteration of Ref. [3]	$O(m^3)$	$O(\log^2 m)$	$m^3 \log m$
Average over n iterations of Ref. [3]	$O(m^{2.5})$	$O(\log^2 m)$	$m^{2.5}$
Any iteration of revised simplex algorithms	$O(m^2)$	$O(m)$	m

other hand, if appropriate *modifications* of Karmarkar's original algorithm indeed run in a sublinear number of iterations [as he reported on at the *TIMS/ORSA Mtg*, Boston, Mass. (May 1985) and at the *12th Int. Symp. on Mathematical Programming*, Boston, Mass. (Aug. 1985)], this would immediately imply a substantial acceleration of the simplex algorithms at least in the case of (i) parallel computation and dense input matrices (see Table A1) and (ii) both parallel and sequential computations where the graph associated with the matrix $\mathbb{H}$ of system (2) has an $s(m+n)$-separator family with $s(m+n) = O((m+n)^q)$, $q < 1$ (see the estimates of Section 5).

Concluding remark

In June 1986, J. Renegar showed that $O(\sqrt{m+n}\,L)$ iterations (each reduced to solving a LLSP) suffice for solving LPP, see Ref. [22].