THE AT&T *KORBX*® SYSTEM

Yun-Chian Cheng, David J. Houck, Jr., Jun-Min Liu, Marc S. Meketon, Lev Slutsman, Robert J. Vanderbei, and Pyng Wang

This paper describes the AT&T KORBX system, which implements certain variants of the Karmarkar algorithm on a computer that has multiple processors, each capable of performing vector arithmetic. We provide an overview of the KORBX system that covers its optimization algorithms, the hardware architecture, and software implementation techniques. Performance is characterized for a variety of applications found in industry, government, and academia.

Yun-Chian Cheng, David J. Houck, Jun-Min Liu, Marc S. Meketon, Lev Slutsman, *and* ***Pyng Wang*** *are in the Advanced Decision Support Systems Department at AT&T Bell Laboratories in Holmdel, New Jersey.* ***Robert J. Vanderbei*** *is a member of technical staff in the Mathematics of Networks and Systems Department at AT&T Bell Laboratories in Murray Hill, New Jersey. Mr. Cheng joined the company in 1985 and has a Ph.D. in computer science from the University of Wisconsin at Madison. Mr. Houck joined the company in 1979 and has both a B.A. in mathematics and a Ph.D. in operations research from The Johns Hopkins University. Mr. Meketon joined the company in 1982 and has a B.S. in mathematics from Villanova University and both an M.S. and Ph.D. in operations research from Cornell University.* (continued on page 19)

Introduction

This paper describes the AT&T KORBX system, which comprises several variants of the Karmarkar linear programming algorithm implemented on a parallel/vector machine called the KORBX system processor. Within AT&T, the KORBX system is being used to solve difficult planning problems, such as the Pacific Basin network-facilities-planning problem. The KORBX system is also being used by commercial and government customers for many types of applications. For example, it is being used by the U.S. Air Force Military Airlift Command (MAC) to solve critical logistics problems and by commercial airlines to solve scheduling problems, such as crew planning. (Panel 1 defines terms and acronyms used in this paper.)

Before getting into details, we should address the question: What is a linear program? It is an optimization problem in which one wants to minimize or maximize a linear function (of a large number of variables) subject to a large number of linear equality and inequality constraints. For example, linear programming may be used in manufacturing-production planning to develop a production schedule that meets demand and workstation capacity constraints, while minimizing production costs.

Since 1947 when the conventional *simplex method* was proposed by George Dantzig,[1] linear programming has been effectively employed to solve decision-making problems in nearly all sectors of industry. Linear programming is also commonly used in engineering and scientific analyses. The KORBX system implements new technol-

Panel 1. Acronyms and Terms

CE	computational element
IEEE	Institute of Electrical and Electronics Engineers
IP	interactive processor
KLP	the linear programming solver (a set of software modules)
`kmps`	routine that converts the input data
`krep`	routine that generates reports
MAC	U.S. Air Force Military Airlift Command
MCNF	multicommodity network flow
MFLOPS	million floating-point operations per second
MINOS 5.1	Fortran implementation of the simplex method from Stanford University's Systems Optimization Laboratory
MPS	Mathematical Progamming System
MPS III	Ketron Management Science Inc. implementation of the simplex method
MPSX	IBM Corporation implementation of the simplex method
NETLIB	collection of problems distributed electronically
`opt`	routine that solves the preprocessed problem
`postp`	routine that undoes the preprocessor's transformations
`prep`	routine that preprocesses the linear program

ogy and is being used today to solve problems that either were too large or ran too slowly to be solved by earlier linear programming methods.

The paper is organized as follows. In the next two sections, we describe the linear programming problem and the linear programming algorithms that are available in the system. Next, the KORBX system is described. Finally, we try to characterize the performance of the system by giving results on a variety of actual and randomly generated problems from academia, industry, and government.

Linear Programming

Mathematically, there are many ways to represent a linear program. But for formulating models the following form is convenient:

$$\begin{aligned} \text{Optimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{b} - \mathbf{r} \le A\mathbf{x} \le \mathbf{b} \\ & \mathbf{l} \le \mathbf{x} \le \mathbf{u} \end{aligned} \tag{1}$$

The matrix A is called the *constraint matrix*, the vector $\mathbf{c}$ is the *objective vector* (a column vector), $\mathbf{c}^T$ is the transpose of $\mathbf{c}$ (a row vector), $\mathbf{b}$ is the *right-hand side*, $\mathbf{l}$ is the vector of *lower bounds*, $\mathbf{u}$ is the vector of *upper bounds*, and $\mathbf{r}$ is the *range* of the constraints. The term "optimize" stands for either "minimize" or "maximize." The function $\mathbf{c}^T\mathbf{x}$ is called the *objective function*. Commonly, m denotes the number of rows of A (i.e., constraints) and n the number of columns (i.e., variables). Sometimes, elements in the lower- or upper-bound vectors are negative or positive infinity, respectively.

While system (1) is convenient for modeling, it is mathematically preferable to formulate the problem as follows:

$$\begin{aligned} \text{Minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{0} \le \mathbf{x} \le \mathbf{u} \end{aligned} \tag{2}$$

Problems formulated as in system (1) are easily converted into form (2). We call system (2) the *primal linear program*. The constraint matrix, objective vector, right-hand-side vector, and upper-bounds vector—when written in this form—are generally different from the original ones. To simplify notation, let us assume that all the upper bounds are infinite. Note, however, that the

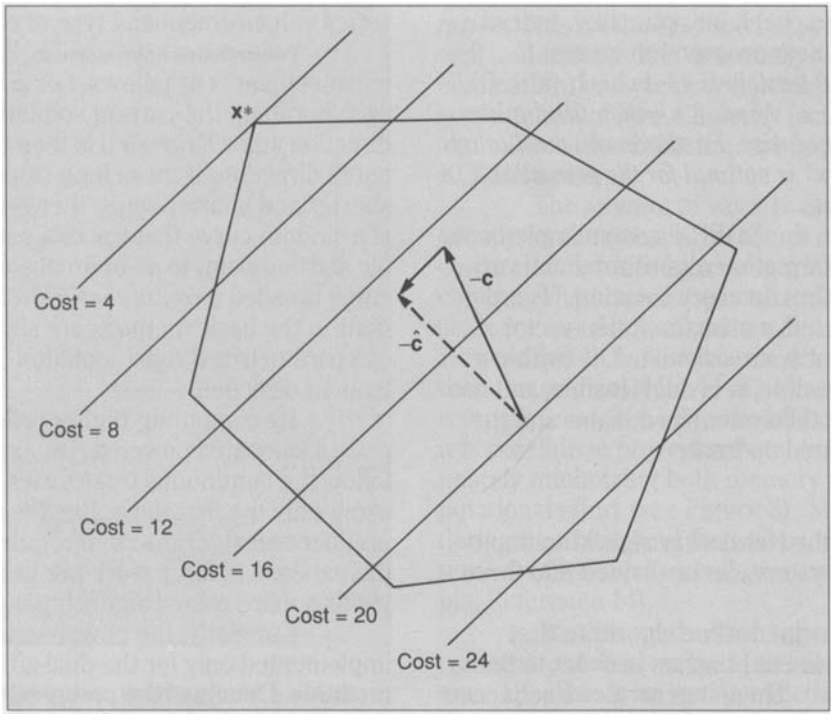


Figure 1. Polyhedron. The walls of the polyhedron are formed by the intersection of the plane $\{x \mid Ax = b\}$ and the nonnegativity constraints $x_i \geq 0$. The optimal point is x^* and the negative of the projected cost vector is $-\hat{c}$. The lines perpendicular to the projected cost vector are the level sets; all points on a level set have the same cost.

KORBX system does indeed handle problems with finite upper bounds.

The constraints $A\mathbf{x} = \mathbf{b}$ form an $n - m$ dimensional hyperplane in an n dimensional space. Each nonnegativity constraint $x_i \geq 0$ cuts off a half space from the hyperplane, resulting in a multidimensional polyhedron (see Figure 1) that is called the *feasible region*. Associated with each point in the feasible region is a cost $\mathbf{c}^T\mathbf{x}$. The objective is to find a feasible point that minimizes this cost. This is called an *optimal solution*. If the problem has a single optimal solution, it must be at a vertex of the polyhedron. However, most real-world problems have multiple optima. The set of multiple optima always consists of some face of the polyhedron. In either case, there are optimal solutions that are at vertices.

The simplex method starts at a vertex of the polyhedron and, by jumping to an adjacent vertex, reduces the objective function value in each iteration. In contrast, the Karmarkar algorithm[2] and its variants start at an interior point and then jump in a direction of descent from one interior point to another, eventually converging to an optimal solution.

Corresponding to any linear program is another linear program, called its *dual*. The dual linear program that corresponds to program (2), assuming the upper bounds are infinite, is:

$$\begin{aligned} &\text{Maximize} && \mathbf{b}^T\mathbf{w} \\ &\text{subject to} && A^T\mathbf{w} \leq \mathbf{c} \end{aligned} \qquad (3)$$

The dual linear program is important for several reasons.

For example, it is used to check for optimality. Indeed, the duality theorem of linear programming says:
THEOREM: *If* $\mathbf{x}^*$ *is primal feasible [i.e.,* $A\mathbf{x}^* = \mathbf{b}$ $(\mathbf{x}^* \geq \mathbf{0})$*] and* $\mathbf{w}^*$ *is dual feasible (i.e.,* $A^T\mathbf{w}^* \leq \mathbf{c}$*), then the duality gap* $\mathbf{c}^T\mathbf{x}^* - \mathbf{b}^T\mathbf{w}^*$ *is nonnegative. Furthermore, the duality gap is zero if and only if* $\mathbf{x}^*$ *is optimal for the primal and* $\mathbf{w}^*$ *is optimal for the dual.*

The optimizer in the KORBX system implements several variants of the Karmarkar algorithm. Each variant is an iterative algorithm. In every iteration, regardless of the method selected, a primal solution vector $\mathbf{x}$ and a dual solution vector $\mathbf{w}$ are calculated. If (within a tolerance) $\mathbf{x}$ is primal feasible, $\mathbf{w}$ is dual feasible, and the duality gap is zero, then the system terminates and the current solution is declared optimal.

Algorithms

The variants of the Karmarkar algorithm implemented in the KORBX system can be divided into three basic categories:

- *Primal-affine.*[3–6] The primal-affine algorithm first solves an augmented linear program in order to find a primal feasible solution. Then, it generates a sequence of primal feasible solutions with decreasing objective value, until eventually the corresponding dual solution is feasible and the duality gap is zero.
- *Dual-affine.*[7,8] The dual-affine algorithm works the other way around. It starts by finding a dual feasible solution and iterates, while maintaining dual feasibility, until eventually the corresponding primal solution is feasible and the duality gap is zero.
- *Primal-dual.*[9,10] The primal-dual algorithm first achieves and then maintains both primal and dual feasibility, while all along reducing the duality gap until it is zero.

The affine algorithms (both primal and dual) are often called *affine-scaling* algorithms.

There are several algorithmic options that may be used with the three basic methods outlined above. We mention here the two most important examples: power-series enhancement and type of equation solver.

Power-Series Enhancement. The idea in power-series enhancement is as follows. For all three methods, at each iteration, the current solution is used to compute a direction and a large step is then taken along this computed direction. If these long steps are replaced by shorter and shorter steps, then—in the limit—we arrive at a smooth curve that traces a path from any given feasible starting point to an optimal solution. This smooth curve is called a *continuous trajectory.* The step directions in the basic methods are simply first derivatives of a parametrized representation of the curve (i.e., the tangent direction).

By computing higher-order derivatives to generate a truncated power-series expansion, it is possible to follow the continuous trajectories more closely than by using only the first derivative (see Figure 2). Doing this produces an algorithm that requires fewer iterations at the expense of more work per iteration. Furthermore, it yields a more robust algorithm.

Currently, the power-series enhancement is implemented only for the dual-affine and primal-dual methods. Details of the power-series enhancement for these two methods are given in Reference 11.

Equation Solver. The second important option regards the method for solving systems of equations of the form:

$$AD^2A^T\mathbf{y} = \mathbf{d} \qquad (4)$$

where $\mathbf{d}$ is a known vector and D is a known diagonal matrix (i.e., a square matrix with zeros off the diagonal). This is the most computationally intensive part of any variant of the Karmarkar algorithm. By default, all the methods use *Cholesky factorization* (see, for example, Reference 12). The idea here is to first factor AD^2A^T into the product of a lower triangular matrix L (called the *Cholesky factor*) and its transpose L^T:

$$AD^2A^T = LL^T$$

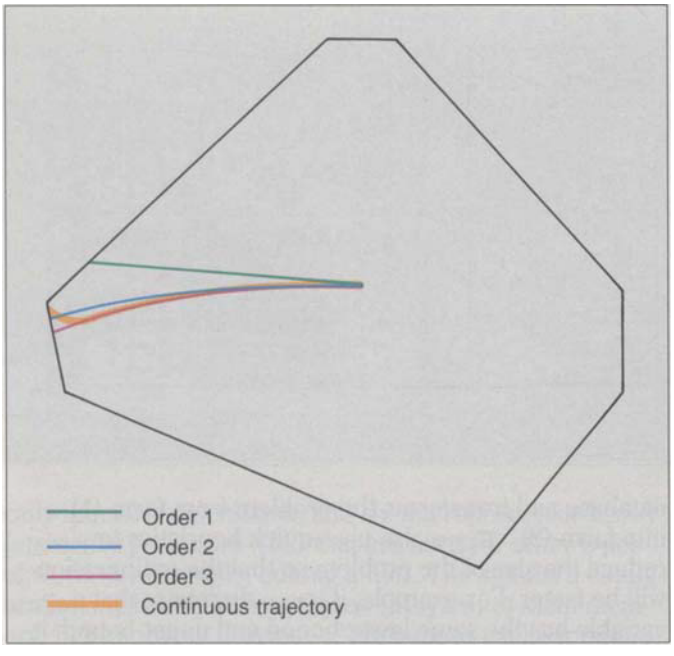


Figure 2. Power series trajectories. Starting from a solution near the center of the polytope, the green line is the order-1 trajectory, the blue line is the order-2 trajectory, the red line is the order-3 trajectory and the gold line is the continuous trajectory. The optimal point is labeled x^*. Note that the higher-order trajectories track the continuous trajectory better than the lower-order, power-series trajectories.

To solve system (4), one exploits the fact that L is lower triangular to first solve for $\mathbf{z}$:

$$L\mathbf{z} = \mathbf{d}$$

by a simple process called *forward substitution*. Then, $\mathbf{y}$ is the solution to:

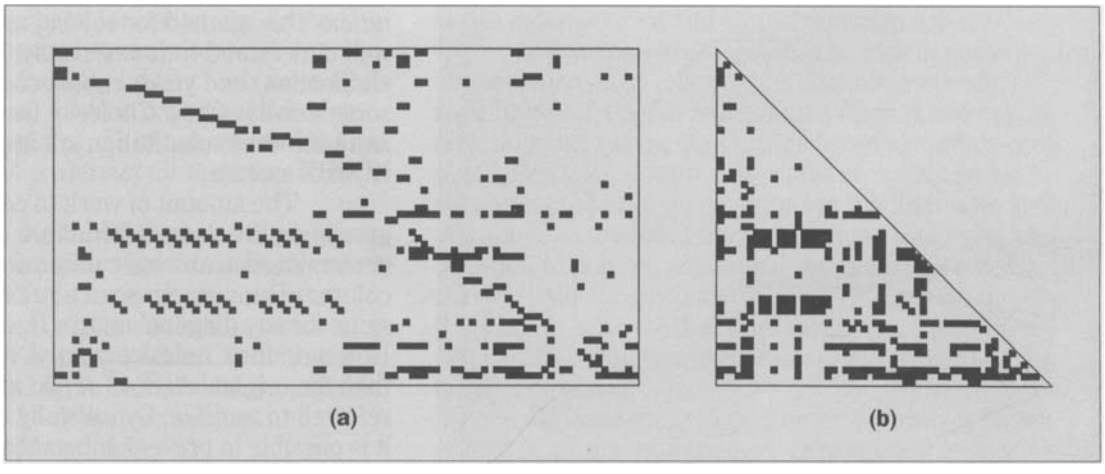
$$L^T\mathbf{y} = \mathbf{z}$$

obtained by a similar process called *backward substitution*. This method for solving systems of equations is well understood (it is essentially the same as Gaussian elimination) and yields robust codes. Reference 13 gives some details of how Cholesky factorization and forward and backward substitution are implemented in the KORBX system.

The amount of work to compute L depends greatly on the nonzero structure of AD^2A^T. Typically, constraint matrices have only a few nonzeros in each column. The nonzero structure of AD^2A^T, which is the same for any diagonal matrix D, usually is also sparse. However, the Cholesky factor L turns out to be denser than the original AD^2A^T. The extra nonzeros in L are referred to as *fill-in*. By carefully ordering the rows of A, it is possible to prevent substantial fill-in, thereby approximately minimizing both memory requirements and computational effort (see Figure 3). The most common reordering heuristic (and the one used in the KORBX system) is called *minimum-degree ordering* (see, for example, Reference 14).

Another method for solving system (4) is the *conjugate-gradient method* (see, for example, Reference 12). This is an iterative method that, at each iteration, generates an improved estimate of $\mathbf{y}$. If AD^2A^T is approximately the identity matrix, then this method converges quickly. The *preconditioned conjugate-gradient method* uses an approximate Cholesky factorization to transform AD^2A^T into such an approximate identity matrix.

Because there are many implementation details (including the choice of preconditioner), developing a robust and quick algorithm is difficult. Some details of our implementation, including new data structures, have been described in Reference 15. Currently, the preconditioned conjugate-gradient option is available only with the dual-affine method and precludes using the power-series enhancement. The advantage of using the preconditioned conjugate-gradient option is that, when properly tuned, it uses less memory than Cholesky factorization. Hence, extremely large problems can be solved.



The KORBX System

The KORBX system includes the linear programming solver (a set of software modules collectively referred to as KLP) and the KORBX system processor (a parallel/vector computer on which the solver runs).

Linear Programming Solver. KLP consists of several modules:

- `kmps`—converts the input data.
- `prep`—preprocesses the linear program.
- `opt`—solves the preprocessed problem.
- `postp`—undoes the transformations of the preprocessor.
- `krep`—generates reports.

Linear programs are usually formulated in the industry-standard Mathematical Progamming System (MPS) format. Mathematically, this format describes linear programs as presented in equation (1). A utility called `kmps` reads an MPS-format file and creates files that store this information in a sparse, binary format. This collection of data files is called a *KLP problem database*. Generating the KLP problem database directly typically saves time in problem generation because `kmps` is bypassed.

A preprocessor, `prep`, reads a KLP problem database and transforms the problem from form (1) into form (2). `Prep` also uses quick heuristics to reduce the size of the problem so that the optimization will be faster. For example, if `prep` discovers that a variable has the same lower bound and upper bound, it will eliminate that variable from the problem until `postp` reinserts it. It is possible that `prep` may determine the problem is infeasible or unbounded and may even find an optimal solution.

The optimizer, `opt`, reads in the problem that `prep` produced, rearranges the rows of A to reduce fill-in, and solves the linear program using one of the methods described in the previous section. Although any method may be selected, the default is dual power-series order 5, where order 5 refers to the number of terms used in the power-series expansion.

Following `opt`, there is a postprocessor, `postp`, that undoes the transformations of `prep`. The final primal and dual solutions are stored in sparse, binary files.

Finally, KLP has utilities for generating reports based on these solution files.

KORBX System Processor. The KORBX system hardware consists of eight parallel processors, called *computational elements* (CEs) that are used for numeri-

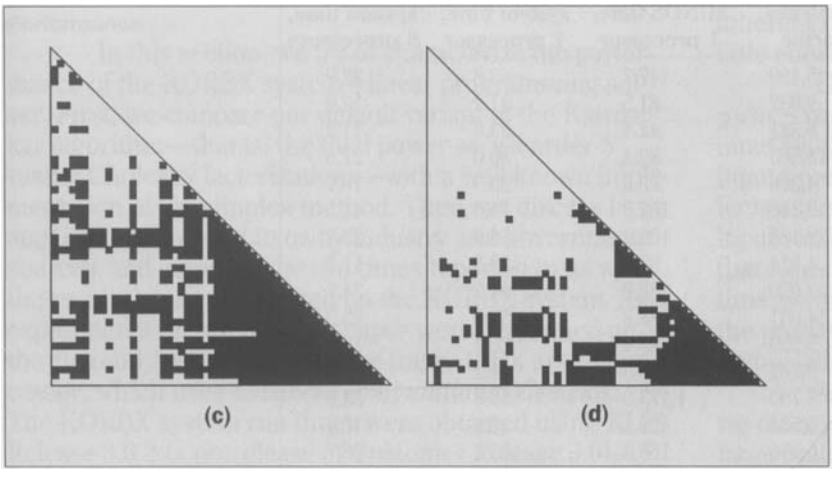


Figure 3. Reordering the matrix for less fill-in. (a) *A* matrix; shows the nonzero structure of the *A* matrix. (b) AD^2A^T matrix; shows the nonzero structure below the diagonal of the AD^2A^T matrix. (c) Cholesky factor *L* with original ordering; shows the nonzero structure below the diagonal of the Cholesky factor. After the matrix has been reordered using the minimum-degree heuristic, there are fewer nonzeros in the Cholesky factor. (d) Cholesky factor after row reordering; shows the nonzeros below the diagonal, reordered by the minimum-degree heuristic.

cally intensive calculation, and six microprocessor-based *interactive processors* (IPs) that are used for other types of work (for example, editing a file). The standard configuration includes 256 megabytes (Mbytes) of main memory, 512 kilobytes (kbytes) of high-speed cache memory, four 1.1-gigabyte (Gbyte) disk drives, a tape drive, printer, and communications hardware. Each CE can process instructions separately and has full access to memory. Also, there are hardware-level memory and process locks. Efficient use of the eight parallel processors can lead to an eightfold speedup over single processor times. Each CE is a vector processor. In fact, each CE includes eight *vector registers*, and each register can handle 32 double-precision numbers at one time. Efficient use of the vector hardware can lead to a fourfold speedup over scalar processing. However, on some sparse matrix computations, the vectorizing benefits are generally modest.

In scalar mode, each processor operates at about 1.0 million floating-point operations per second (MFLOPS) for double-precision arithmetic written in Fortran. (The floating-point hardware follows the IEEE standard.) If a Fortran routine reaps full benefit from vectorization and parallelization, then one should expect to see about 32 MFLOPS. By carefully managing main memory and cache memory, it is possible to do even better. For example, we have a dense matrix-multiplication routine—which is written in assembly language and is optimized to the greatest extent possible—that achieves 53 MFLOPS. For linear programs, a reasonable rule of thumb is to expect about 37 MFLOPS on dense problems and about 2 MFLOPS on sparse ones.

KORBX System Software. The KORBX operating system, an extension of the UNIX® operating system, has been specially designed to accommodate parallel processing. It supports virtual memory and, hence, handles processes that require up to about 1 Gbyte of memory. Included with the system are Fortran and C language compilers. The Fortran compiler has a loop analyzer that tries to vectorize and parallelize a user's program automatically. In addition, there are compiler directives—written as comment statements—that allow the programmer to control the loop analyzer. But for complicated subroutines such as some of those in `opt`, assembly language programming may be needed to achieve the greatest possible parallelization and vectorization. For C programmers, a library provides many routines for generating parallel algorithms.

Table I. Test problems and comparisons between MINOS and KLP

Problem name	Rows	Columns	Nonzeros in constraint matrix	Nonzeros in Cholesky factor	MINOS time, 1 processor	KORBX system time, 1 processor	KORBX system time, 8 processors
pilot4	410	1,000	5,141	15,190	147.7	63.8	22.9
scfxm2	660	914	5,183	9,932	81.5	31.2	12.9
grow15	300	645	5,620	6,321	42.4	23.6	11.2
fffff800	524	854	6,227	18,503	82.4	80.0	27.6
ship04l	402	2,118	6,332	4,296	25.9	29.9	16.7
sctap2	1,090	1,880	6,714	12,348	102.3	71.0	44.5
ganges	1,309	1,681	6,912	29,188	109.7	56.9	22.3
ship08s	778	2,387	7,114	4,514	37.8	22.5	11.9
sierra	1,227	2,036	7,302	13,030	223.0	64.7	34.7
scfxm3	990	1,371	7,777	14,631	188.8	58.0	23.3
ship12s	1,151	2,763	8,178	5,529	89.4	26.7	13.3
grow22	440	946	8,252	9,261	99.0	34.2	15.5
ken7‡	2,426	3,602	8,404	8,790	1,173.0	59.8	33.1
scsd8	397	2,750	8,584	5,588	429.1	22.1	10.1
sctap3	1,480	2,480	8,874	16,956	165.3	102.1	68.7
pilot.we	722	2,789	9,126	18,303	1,029.6	109.3	45.2
doa‡	2,617	4,640	9,427	8,546	808.8	75.8	52.2
25fv47	821	1,571	10,400	35,456	1,545.4	106.7	34.1
czprob	929	3,523	10,669	6,779	327.0	95.3	56.4
ship08l	778	4,283	12,802	6,484	103.4	51.3	28.3
pilotnov	975	2,172	13,057	59,999	455.3	133.4	47.3
nesm	662	2,923	13,288	28,524	478.6	174.4	59.9
pilot.ja	940	1,988	14,698	52,793	1,310.3	212.6	69.7
ship12l	1,151	5,427	16,170	9,433	286.5	64.5	33.5
m3000‡	3,001	6,000	17,998	12,008	1,820.5	59.7	32.0
80bau3b	2,262	9,799	21,002	44,165	6,460.5	577.9	373.5
ken9‡	6,601	11,137	25,474	51,951	33,147.8	349.2	170.1
greenbea	2,392	5,405	30,877	52,727	14,758.5	444.3*	138.1*
greenbeb	2,392	5,405	30,877	52,421	8,044.5	311.9	119.8
pilots	1,441	3,652	43,167	226,172	15,046.5	1,099.5	261.9
jim100‡	2,007	19,020	61,976	34,958	27,890.8	205.3	80.2
mac10‡	15,246	47,333	101,849	1,838,050	†	12,633.5	2,736.6
t01p08‡	6,532	10,315	293,750	3,306,674	§	34,506.6#	5,286.9#

NOTE: All times are in seconds. This test set includes all the NETLIB problems that have more than 5,000 nonzeros plus other problems from various sources. MINOS was run on one processor; KLP was run on one processor and on the standard configuration. Both systems were run using default parameters (for KLP, this means dual power series, order 5). The KLP run times include `prep` plus `opt` plus `postp` (excluding `kmps`). The MINOS times also exclude data conversion.

* Using the primal-dual power series, order-3 method and setting the parameter `phlmult` to 3.5.

† Failed to reach the optimal solution after 1,000 degenerate pivots (and after 6,725.6 seconds).

§ Problem is too large.

‡ Non-NETLIB problem.

Using the preprocessor option (`linrow`).

Performance

In this section, we try to characterize the performance of the KORBX system's linear programming solver. First, we compare our default variant of the Karmarkar algorithm—that is, the dual power-series order 5 (using Cholesky factorization)—with a well-known implementation of the simplex method. Then, we discuss large applications supplied to us by industry and government sources, and compare the run times reported to us with the results we have obtained on the KORBX system. All experiments reported in this paper were performed on the currently released version of the KORBX system processor, which uses *advanced computational elements*. The KORBX system run times were obtained using KLP Release 3.0.2 (a prerelease of Customer Release 3.0).

Comparison with the Simplex Method. For comparisons of our system with the simplex method, we ran MINOS 5.1, which is a Fortran implementation of the simplex method from the Systems Optimization Laboratory at Stanford University.[16] Both optimizers were run with their default parameters. We chose MINOS because it is widely available, is often used for benchmarks, and is easily ported to the KORBX system processor.

Table I shows our results. The collection of problems that were used include all problems from the NETLIB suite[17] that have at least 5,000 nonzeros in their constraint matrix. We have also included seven other problems; some are very sparse, and some are very dense. The table is sorted according to the number of nonzeros in the constraint matrix.

Because MINOS is designed for a serial machine, we ran MINOS on one CE and ran KLP on both one and eight CEs. (It is believed that interior point methods can be made parallel more easily than can the simplex method.) Note that some of the critical subroutines in our system are written in assembly language and are designed to exploit efficient cache and main memory management.[13] Also, vectorization can result in a speedup of up to a factor of 4. Although we compiled MINOS with the loop analyzer turned on so it could reap any inherent benefit from vectorization, this seemed to have little effect.

The KLP times include the `prep`, `opt`, and `postp` run times. We have omitted the time for the data input routine, `kmps`. The MINOS times also exclude data input time. We should note that, for many small problems and some large sparse problems, the time for data input can be significant. Studies we have made indicate that `kmps` generally takes about the same amount of time as the analogous portion of MINOS. Note that, as the problems get larger, the relative speedup of the optimization increases.

Performance on Large-Scale Problems. In this section, we discuss the performance of the KORBX system on large-scale problems obtained from industry, government, and academia. For several of these problems, we compare results obtained on the KORBX system to the run times reported to us by customers during their product evaluation phase. Table II summarizes these problems. These reported results generally reflect IBM's simplex implementation, MPSX, that runs on an IBM 3090 mainframe computer. Depending on the model, IBM 3090 computers achieve between 7 and 14 MFLOPS. In comparison, recall that KLP runs between 2 and 37 MFLOPS, depending on the problem.

Pacific Basin facility planning. An important internal AT&T application is the Pacific Basin facility-planning problem. The problem's objective is to determine where undersea cables and satellite circuits should be installed, when they will be needed, number of circuits needed, technology to be used for the cables, and routes for calls. This problem has been modeled as a linear program.[18,19]

The KORBX system solves the 10-year planning-horizon problem (called `pbfp10` in Table II) in about 21 minutes. Using a special version of the dual-affine method developed for Pacific Basin facility planning and implemented on the KORBX system processor, this problem was solved in about 4 minutes. This implementation was customized for these problems to ensure maximum use of the parallel processors. It took about

Table II. Summary of large-scale problems

Problem name	Rows	Columns	Nonzeros in constraint matrix	Comparative environment	Comparative time	KORBX system time
mopp2	3,685	12,701	83,253	IBM 3090/MPS III	11	5
airline1	4,420	6,711	101,377	IBM 3090/MPSX	500	23
financial	846	44,974	136,970	IBM 3090/MPSX	*	12
pbfp10	15,425	34,553	137,866	IBM 3090/MPSX	76	18
mopp3	5,316	13,236	174,231	IBM 3090/MPS III	47	15
airline2	6,642	10,707	203,597	IBM 3090/MPSX	†	78
pbfp19	27,971	76,815	249,455	IBM 3090/MPSX	N	53
mac30	46,863	152,058	326,252	Cyber 273 and IBM 3081/MCNF	‡	234§
mac40	63,027	209,739	449,209	Cyber 273 and IBM 3081/MCNF	‡	360§
mac50	79,019	267,357	571,792	Cyber 273 and IBM 3081/MCNF	‡	612§
mopp1	24,902	43,018	583,732	IBM 3090/MPS III	‡	450
mlp	20,420	519,660	2,017,380	Honeywell DPS-6 and IBM 3081/Benders decomposition	#	74

NOTE: Run times are in minutes. The comparative environment and run times were reported to us by the client, except for financial, pbfp10, and pbfp19, which were run at AT&T.

* Did not solve after 120 minutes and 100,000 iterations.

† Did not solve after 1440 minutes.

§ Run times using a prerelease version of KLP Release 3.1

‡ Problem believed unsolvable by these systems.

Method failed.

76 minutes to solve the same problem on an IBM 3090 computer using MPSX.

The ability to solve these problems quickly gives planners the opportunity to develop a more efficient network design. Currently, this algorithm is being used to solve the 19-year planning problem (pbfp19), which MPSX cannot handle because of size limitations.

Military-officer personnel planning. Another linear program, called mopp1 in Table II, plans U.S. Army officer promotions (to Lieutenant, Captain, Major, Lieutenant Colonel, and Colonel), accessions (new hires), and training requirements by skill category. The objective is to develop a plan that meets the Army's force structure requirements as closely as possible.

This linear program (which has 21,000 constraints and 43,000 variables) was solved on the KORBX system in 7.5 hours (using the primal-dual, power-series, order-3 method). This problem has not been solved using commercial simplex packages because of their size and speed limitations. In an attempt to obtain a reasonable solution using these simplex packages, the linear program was decomposed into five smaller models according to rank. Interestingly, solving the larger model produced a plan that was closer to the force structure requirements by 32 percent when compared to the aggregate of the solutions to the five smaller models.

The performance of the KORBX system on the smaller models is also interesting. The KORBX system solved the Lieutenant model (problem mopp2) the fastest—i.e., in 5 minutes—while, according to the Army, MPS III (a commercial implementation of the simplex method) on an IBM 3090 Model 200 computer took

11 minutes. The Lieutenant Colonel model (problem `mopp3`) took 15 minutes on the KORBX system, compared to 47 minutes on the IBM 3090 computer. The performance advantage was similar on the other problems.

Military logistics planning. The U.S. Air Force Military Airlift Command has a patient evacuation problem that has been modeled as a *multicommodity network-flow* problem. The MAC uses this patient distribution system to determine the flow of patients moved via an airlift from an area of conflict to bases and hospitals in the continental United States. The objective is to minimize the time that patients are in the air-transport system. The constraints are:

- Demand must be met.
- Size and composition of the hospitals, staging areas, and air fleet are conserved.

The MAC has generated a series of problems based on the number of time periods (days). Using a specialized simplex code, the MAC was able to solve as large as a 5-day problem. Both the specialized simplex code and MINOS spent thousands of iterations without changing the objective value, and were unable to solve the 10-day problem (called `mac10` in Table I). The KORBX system solved this problem in less than 2 hours. The dual conjugate-gradient method can solve the 50-day problem (79,000 constraints and 572,000 variables) in 10.2 hours. Table II summarizes the results of the 30-, 40-, and 50-day problems (called `mac30`, `mac40`, and `mac50`).

The Department of Defense Joint Chiefs of Staff has a logistics planning problem that models the feasibility of supporting military operations during a crisis. The problem is to determine if different materials (called movement requirements) can be transported overseas within strict time windows. The linear program models capacities at the embarkation and debarkation ports, capacities of the various airplanes and ships that carry the movement requirements, and penalties for missing delivery dates.

Using simulated data, we generated a problem (called `mlp`) that has 15 time periods, 12 ports of embarkation, 7 ports of debarkation and 9 different vehicles for carrying 20,000 movement requirements. This problem, which has about 21,000 constraints and 500,000 variables, took about 75 minutes to run. The military has generated similar problems that have proved intractable, even after applying techniques such as Bender's decomposition and network flow approaches (with and without side constraints).

Airline problems. The airline industry has supplied us with several, relatively large, dense linear programs (average of 50 nonzeros per column). According to one airline, the KORBX system solved its sample problems about 20 times faster than MPSX on an IBM 3090 computer (problems `airline1` and `airline2` in Table II). However, this airline has developed specialized heuristics to generate initial solutions. If they start MPSX with these customized initial solutions, they can cut MPSX's run time by at least a factor of 6 from those reported in Table II. Even from a "cold start," the KORBX system solved these problems faster than MPSX from a "warm start."

Financial industry problems. We experimented with a class of *bond arbitrage* problems. These problems (we have three of them) have about 850 constraints and 45,000 variables. After 100,000 iterations (and 120 CPU-minutes), MPSX—running on an IBM 3090 computer—was still performing degenerate pivots at the original vertex.

The KORBX system solves each of these problems in about 12 minutes. Table II shows the results for one of these problems (`financial`).

Multicommodity network-flow problems. An important class of linear programming applications are the multicommodity network-flow problems. These problems are used to find a minimum cost routing for each of several commodities through a network whose arcs have finite capacity. For these problems, there are specialized simplex-based codes that are much faster than general-purpose simplex codes.

Table III. Randomly generated multicommodity flow problems

Problem name	Rows	Columns	Nonzeros in constraint matrix	MCNF	KORBX system time, 1 processor	KORBX system time, 8 processors
`jlk001`	1,152	1,890	5,300	267.1	58.7	19.1
`jlk002`	5,422	5,530	15,280	2,074.3	322.6	102.4
`jlk003`	10,836	10,530	29,420	2,093.3	429.2	180.1
`jlk004`	22,069	26,120	72,930	*	12,441.6	2,551.5

NOTE: The MCNF column is the run time of Kennington's specialized primal-simplex multicommodity flow solver. KLP run times (seconds) include `prep` plus `opt` plus `postp`.
* Method reported floating-point exception after 101,927 seconds.

We compared the KORBX system against a specialized, simplex-based, multicommodity network-flow (MCNF) code developed at Southern Methodist University by Kennington.[20] The problems we used were created with a generator developed by Ali and Kennington.[21] We ran the KORBX system on one CE and then on eight CEs. Kennington's solver was compiled by the Fortran compiler with the loop analyzer turned on. However, his code was not designed for a parallel machine and, hence, benefited little from vectorization and parallelization. Table III shows the results.

Results for two other randomly generated multicommodity flow problems, `ken7` and `ken9`, are part of the benchmarks presented earlier in the simplex-method comparison.

Summary

Karmarkar-based optimization algorithms, parallel/vector processing hardware, and sophisticated software implementation techniques have been combined, yielding the AT&T KORBX system. The system is now tackling linear-programming problems that were heretofore unsolvable by other methods.

Acknowledgments

The AT&T KORBX system is the result of long hours of hard work by many individuals. We especially owe a great deal to our colleagues on AT&T's Advanced Decision Support Systems team. In particular, we would like to acknowledge some of the people whose contributions have had significant impact on the final product: Efthymios Housos and Chih Chung Huang for work on parallelization; Bob Murray, Adrian Kester, and Srinivas Sataluri for developing and enhancing the preprocessor and postprocessor; and Karen Medhi for improvements to the dual conjugate-gradient method. We would also like to thank Narendra Karmarkar and Ram Ramakrishnan for developing prototype code for both the primal-affine and dual conjugate-gradient methods and for other help they have given us.

References

1. G. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," *Activity Analysis of Production and Allocation*, Tj. C. Koopmans (ed.), John Wiley & Sons, New York, 1951, pp. 339-347.
2. N. K. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica*, Vol. 4, No. 4, 1984, pp. 373-395.
3. I. I. Dikin, "Iterative solution of problems of Linear and Quadratic Programming," *Soviet Mathematics Doklady* Vol. 8, No. 3, 1967, pp. 674-675.
4. I. I. Dikin, "On the speed of an iterative process," *Upravlyaemye Sistemi*, Vol. 12, No. 1, 1974, pp. 54-60.
5. E. Barnes, "A variation on Karmarkar's algorithm for solving linear programming problems," *Mathematical Programming*, Vol. 36, 1986, pp. 174-182.
6. R. J. Vanderbei, M. S. Meketon, and B. A. Freedman, "A modi-

fication of Karmarkar's algorithm," *Algorithmica*, Vol. 1, No. 4, 1986, pp. 395-407.
7. R. J. Vanderbei, "The affine-scaling algorithm for linear programs with free variables," *Mathematical Programming*, Vol. 43, 1989, pp. 31-44.
8. I. Adler, N. K. Karmarkar, M. G. C. Resende, and G. Veiga, "An implementation of Karmarkar's algorithm for linear programming," Technical Report ORC 86-8, Operations Research Center, University of California, Berkeley, California, 1986.
9. M. Kojima, S. Mizuno, and A. Yoshise, "A primal-dual interior point algorithm for linear programming," Report No. B-188, Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan, 1987.
10. R. Monteiro, and I. Adler, "Interior path following primal-dual algorithms, Part I: Linear programming," *Mathematical Programming*, to be published, 1988.
11. N. K. Karmarkar, J. C. Lagarias, L. Slutsman, and P. Wang, "Power Series Variants of Karmarkar-Type Algorithms," *The AT&T Technical Journal*, Vol. 68, No. 3, May/June 1989, pp. 20-36.
12. G. H. Golub, and C. Van Loan, *Matrix Computations*, The John Hopkins University Press, Baltimore, Maryland, 1983.
13. E. Housos, C. C. Huang and J. M. Liu, "Parallel Algorithms for the AT&T KORBX® System," *AT&T Technical Journal*, Vol. 68, No. 3, May/June 1989, pp. 37-47.
14. A. George, and J. W-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
15. N. K. Karmarkar, and K. G. Ramakrishnan, "Implementation and Computational Results of the Karmarkar Algorithm for Linear Programming, Using an Iterative Method for Computing Projections," presented at the Thirteenth International Symposium on Mathematical Programming, Tokyo, Japan, August 1988.
16. B. A. Murtagh, and M. A. Saunders, "MINOS 5.1 User's Guide," Technical Report 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, December 1983, revised January 1987.
17. D. Gay, "Electronic mail distribution of linear programming test problems," *COAL newsletter* (Committee on Algorithms), No. 13, 1985, pp. 10-12.
18. N. F. Dinn, N. K. Karmarkar, and L. P. Sinha, "Karmarkar algorithm enables interactive planning of networks," *AT&T Bell Laboratories RECORD*, March 1986, pp. 11-13.
19. L. P. Sinha, B. A. Freedman, N. K. Karmarkar, A. Putcha, and K. G. Ramakrishnan, "Overseas network planning—application of Karmarkar's algorithm," *Proceedings of Network '86*, Tarpon Springs, Florida, 1986.
20. J. L. Kennington, "MCNF85 (Primal simplex, multi-commodity network flow solver)," Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, Texas, 1985.
21. A. I. Ali, and J. L. Kennington, "MNETGN program documentation," Technical Report No. IEOR 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, Texas, 1977.

Biographies (continued)

Mr. Liu joined the company in 1982 and has a Ph.D. in operations research and system engineering from The University of North Carolina at Chapel Hill. Mr. Slutsman joined the company in 1981 and has an M.S. in mathematics from the University of Leningrad, U.S.S.R., and a Ph.D. in applied mathematics from Leningrad Mining Institute, U.S.S.R. Mr. Vanderbei joined the company in 1984 and has both a B.S. in chemistry and an M.S. in operations research and statistics from Rensselaer Polytechnic Institute, and both an M.S. and Ph.D. in applied mathematics from Cornell University. Mr. Wang joined the company in 1980 and has a B.S. in mathematics from the National Taiwan Normal University, Taiwan; an M.S. in operations research and systems analysis from the University of North Carolina at Chapel Hill; and a Ph.D. in mathematics from the University of Illinois at Champaign-Urbana.

(Manuscript received April 11, 1989)