Progress in Linear Programming

Robert E. Bixby

September, 1993
(revised October 1993)

TR93-40

Progress in Linear Programming

Robert E. Bixby
Department of Computational and Applied Mathematics
Rice University, Houston, Texas 77251
bixby@rice.edu

There is little doubt that barrier methods are now indispensable tools in the solution of large-scale linear programming problems. However, it is our opinion that the results of Lustig, Marsten, and Shanno (hereafter LMS) somewhat overstate the performance of these methods relative to the simplex method. We will present a slightly different view of progress in linear programming, one in which barrier methods do not dominate in the solution of large-scale problems.

1. A Basis for Comparison

The primary test set of models considered in LMS is listed in Table 1.

Table 1: Sample Models

Name	Rows	Columns	Nonzeros
schedule	23259	29342	75520
energy1	16223	28568	88340
initial	27441	15128	95971
energy2	8335	21200	161160
car	43387	107164	189864
continent	10377	57253	198214
fuel	18401	33905	205789
energy3	27145	31053	268153
TOTALS	174568	323613	1283011

These models are accurately called large-scale, though certainly not among the largest models that are now solved. They are also relatively sparse, which probably makes them better candidates for barrier methods. As LMS also note, there is likely some bias in the set, having originated in part from users who were unhappy with the performance of their current LP solvers.

As a basis for examining progress in the simplex method, the first released version of the CPLEX linear optimizer, CPLEX 1.0, will be used. (CPLEX is a trademark of CPLEX Optimization, Inc.) The simplex routines in CPLEX 1.0, and in subsequent versions of the software, were written by the author.

When CPLEX 1.0 was first distributed in late 1988, we believe it was among the better portable implementations of the simplex method. In tests we carried out

on the NETLIB set of linear programs [8], it was shown to be about four times as fast as the XMP [14] simplex code and the then current version of MINOS [17]. Subsequent versions of MINOS represent significant improvements over that earlier code.

We ran CPLEX 1.0 on the models listed in Table 1 (the two large Delta models from LMS were not made available) with an iteration limit of 500,000. A SPARCstation 10/Model 41 was used in this and subsequent tests. The results are listed in Table 2.

Table 2: CPLEX 1.0 Running Times[a]

Name	Iterations	Time[d]
schedule	[b]500000	174463.5
energy1	46937	8621.8
initial	40734	21890.8
energy2	477273	80248.2
car	44061	9986.3
continent	43937	2507.1
fuel	210628	39451.1
energy3	[c]500000	110788.2
TOTAL		447957.0

[a] SPARCstation 10/model 41, iteration limit 500,000

[b] Not optimal

[c] Not feasible

[d] CPU seconds

Note that two of the models did not solve within the allowed number of iterations. One of those two, *energy3*, did not reach feasibility. The *schedule* model was having significant difficulties due to degeneracy: CPLEX 1.0 contains no feature for directly dealing with degeneracy. It does contain a very effective implementation of what will be referred to as *multiple-partial pricing*, an approach that may be considered a hybrid of the classical multiple- and partial-pricing ideas [16]. Multiple–partial pricing remains part of the default scheme employed in the current primal simplex implementation. CPLEX 1.0 contains none of the more advanced full-pricing ideas discussed later in this commentary.

Other features of CPLEX 1.0 that are unchanged in subsequent versions include:

- The procedure for constructing initial bases, as described in [2].
- An implementation of the ratio test, as suggested in [11]. This approach exploits feasibility tolerances to expand the set of possible entering variables, and hence improve numerical stability.

- An implementation of the *extended composite simplex method* [19] in phase 1. This method views the phase 1 objective, minimizing the total bound infeasibility, as piecewise linear.

- A Markowitz [13] based LU-factorization routine, as described in [18].

2. Preprocessing

One of the important features missing in CPLEX 1.0 is problem preprocessing. Preprocessing methods are not new [4], but aggressive implementations were not available in many codes until more recently. One reason was that smaller model instances, such as those represented by the NETLIB set, did not seem to benefit markedly from preprocessing. The emergence of barrier methods has also motivated further developments. They are very sensitive to extraneous model components that may, for example, make the interior of the feasible region empty. They are also sensitive to practices such as splitting free variables, which necessarily make the set of optimal solutions unbounded (if nonempty).

The current version of CPLEX includes an essentially full implementation of the reductions present in OB1. It differs from the implementation in OB1 in the order in which the reductions are applied. In addition, our implementation does not remove linearly dependent equations.

Model sizes after preprocessing are reported in Table 3. The biggest reductions occurred in the number of rows, the most important of the three quantities reported. Indeed, the reduction in number of rows was approximately 70% for the two models that were not solved by CPLEX 1.0 within the allowed 500,000 iterations. Thus, the improvements in solution times reported in Table 4 (which is measuring the improvement due to preprocessing) are somewhat exaggerated.

Table 3: Preprocessed Sizes

Name	Rows	Columns	Nonzeros
schedule	5743	12416	40269
energy1	11145	22399	74883
initial	21050	12863	84823
energy2	6845	18307	128948
car	41763	102208	183284
continent	6935	45784	158404
fuel	10163	24854	174703
energy3	9521	28649	188393
TOTALS	113165	267480	1033707
% Remaining	65%	83%	81%

Table 4: CPLEX 1.0 with Preprocessing

Name	Iterations	Time
schedule	[a]500000	41268.7
energy1	47096	6970.0
initial	34677	4007.3
energy2	88600	11533.6
car	53962	11927.3
continent	39363	1636.5
fuel	163977	19556.8
energy3	68927	5948.2
TOTAL		102848.4

[a] Reached iteration limit

3. Improvement in the Primal Simplex Method

Let us denote the current, research version of CPLEX by CPLEX 2.2. Its primal implementation includes several important improvements over CPLEX 1.0. The most important improvements are the following:

- A bound perturbation method for dealing with *stalling* (long sequences of iterations with no progress in the objective function).
- Implementation of an LU–update, in the spirit of [7]. CPLEX 1.0 uses a product-form update. See [5] for a nice description of factorization updates, including those mentioned here.
- The Harris [11] version of the ratio test does improve stability and the handling of degeneracy, but can lead to basic variables drifting infeasible. This phenomenon is combated by a kind of bound shifting [9]. Bound shifting and bound perturbation are closely related.
- Improved pricing.

3.1 Pricing in the Primal

The most important improvements in primal simplex implementations over the last several years have come in the pricing step. As will become clear, much of the credit for these improvements is due Don Goldfarb. Consider a linear programming problem of the form

$$\text{(LP)} \qquad \begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

where $c \in Q^n$ (Q denotes the set of rational numbers), $A \in Q^{m \times n}$ and $b \in Q^m$ are the given data, m and n positive integers, and $x \in Q^n$ is a vector of variables. A *feasible basis* B of (LP) is an $m \times m$ nonsingular submatrix of A such that the basic variables have nonnegative values: $x_B = \mathbf{B}^{-1}b \geq 0$. The ordered set B, often referred to as the *basis header*, denotes the index-set of the variables in the basis, and its bold-face counterpart $\mathbf{B}$ denotes the corresponding submatrix of A. The *nonbasic variables* x_N are set to zero.

The simplex method, introduced by Dantzig in 1947, is a simple iterative process that will be familiar to all readers of this article. At the beginning of a generic iteration, not the first, let us denote the basis by $\bar{B}$, the basis at the beginning of the previous iteration by B, and the *shadow prices* for B by π, so that

$$\mathbf{B}^T\pi = c_B.$$

Let i^* denote the index of the column that is different in B and $\bar{B}$—assuming there is one—and let A_{j^*} denote the column of A that replaced column i^* in B. Let y be the solution of $By = A_{j^*}$. Thus, x_{j^*} is the *entering variable* from the previous iteration, and y the representation of the corresponding column in terms of the basis. The vector y is computed at each iteration independent of the pricing paradigm.

The *pricing* part of the current iteration may then be described as follows:

(P1) Solve

$$\mathbf{B}^T z = e_{i^*},$$

where $e_{i^*} \in Q^m$ is the unit vector with a 1 in position i^*.

(P2) Compute the shadow prices for $\bar{B}$,

$$\bar{\pi} = \pi + \alpha z,$$

where $\alpha = d_{j^*}/y_{i^*}$ and d_{j^*} is the *reduced cost* (see (P3)) of x_{j^*} at the start of the previous iteration.

(P3) Compute all or part of the vector of reduced costs

$$\bar{d}_{\bar{N}} = c_{\bar{N}} - A_{\bar{N}}^T\bar{\pi}.$$

If $\bar{d}_{\bar{N}} \geq 0$, $\bar{B}$ is optimal and the algorithm terminates; otherwise, some nonbasic variable x_j with $\bar{d}_j < 0$ is selected as the entering variable.

For many years *full pricing*, the computation of all of d_N at each iteration, was considered computationally too expensive; however, essentially all of the recent advances in pricing use full pricing in some form.

Let us consider the question of efficiently computing all of d_N, at each iteration. Of course, there is no single most efficient approach. Computer architectures and individual problem structures differ too significantly. The approach that we have

taken is the following. First, given that one is computing all of d_N, it is clearly advantageous to introduce some sort of update. The standard one is

$$\bar{d}_{\bar{N}} = d_{\bar{N}} + \alpha A_{\bar{N}}^T z,$$

where z and α are as defined in (P1)–(P3). The advantage of this computation is that z is usually much sparser than π. To implement it one must compute $A_{\bar{N}}^T z$. In a straightforward approach, it is not obvious how to effectively exploit both the sparsity of z and A. We have therefore chosen to use the fact that computer memory is now relatively cheap, and store the constraint matrix A twice, once by column, in the usual input format, and once by row for the nonbasic columns. In the form stored by row, if some entry of z is zero, then with a single test for that zero, one avoids all the multiplications and additions for the corresponding row of A_N. The speedup is substantial.

3.2 Normalized pricing

It is not proposed to use d_N to implement the classical *most-negative rule*: select as the entering variable x_j the one that minimizes d_j. The performance of that rule, both in time and, perhaps surprisingly, in the number of iterations is inferior to sensible implementations of partial pricing. In a test we performed on the NETLIB set it was found that, in total, the most-negative rule used about 20% more iterations than multiple-partial pricing.

While direct application of the most-negative rule is not a good idea, various kinds of *normalized pricing*, in which the reduced costs are scaled before selecting the entering variable, are effective. The normalizations we have in mind replace d_j by

$$\text{(SE)} \qquad \frac{d_j}{(\bar{A}_j^T \bar{A}_j + 1)^{\frac{1}{2}}}$$

where $\bar{A}_j = \mathbf{B}^{-1} A_j$. The geometric interpretation is straightforward. Reduced cost measures the change in the objective per unit change along the corresponding coordinate axis. The normalized version measures the change in the objective per unit change along the edge of $\{x \geq 0 : Ax = b\}$ that joins the points determined by the current basis and the proposed new basis (due to degeneracy, this edge may have zero length). It is not hard to imagine that the latter criterion is more directly related to the true geometric nature of the problem, and not its orientation relative to the axes.

Using the exact form (SE) is called *steepest-edge pricing*. We will also consider a particular alternative to (SE) called devex [11], which can be viewed as an approximation to (SE). The fundamental paper on steepest-edge algorithms is [10]. The main point of [10] is that a direct implementation of (SE) is impractical, since the

computation of the denominators is too expensive. In [10] it is shown how these norms can be updated. Assuming that the full vector d_N is computed, the extra work for this update per iteration may be summarized as follows:

(SE1) As in (P1)-(P3), let x_{j^*} be the entering variable at the last iteration and let B the basis at the start of the last iteration. Where $\mathbf{B}y = A_{j^*}$, solve

$$\mathbf{B}^T w = y.$$

(SE2) Compute the inner products $\{w^T A_j : z^T A_j \neq 0\}$.

The extra work implied by (SE1) and (SE2) can be substantial, particularly if w has a large fraction of nonzeros, as it frequently does. Our tests indicate that while steepest-edge is an invaluable tool, it is too expensive as a general-purpose pricing rule. As an alternative, we consider the devex rule [11]. It uses approximations to the denominators in (SE). Let us denote these approximations by T_j for each nonbasic variable x_j. As noted below, the T_j values are periodically re-initialized to 1. Suppose that R was the set of indices of the nonbasic variables at the last re-initialization of the T_j values. Let

$$\rho = (\delta(j^*) + \sum_{B_i \in R} y_i^2)^{\frac{1}{2}},$$

where

$$\delta(j^*) = \begin{cases} 1 & \text{if } j^* \in R \\ 0 & \text{otherwise.} \end{cases}$$

Let

$$r_N = \frac{\rho}{y_{i^*}} A_N^T z,$$

where z is as in (P1) and $By = A_{j^*}$. The value T_j for each nonbasic variable is then replaced by

$$\max\{T_j, r_j\},$$

where $T_{B_{i^*}}$ is taken to be 1 in this formula. Since the T_j are obviously monotone nondecreasing, it is necessary to monitor their values relative to the true norms, and occasionally reset them to 1. See [10] for a further discussion. The work in implementing devex, in addition to computing the full vector d_N, is: Computation of $A_N^T z$ and the "norm" ρ, and the comparisons of the T_j and r_j values.

Devex involves relatively little work in addition to full pricing. The default primal pricing in current versions of CPLEX is a hybrid of multiple–partial pricing and devex, in which the code begins with the cheaper multiple–partial pricing and, based on progress in the objective and the relative cost of the linear algebra per iteration, conditionally switches to devex. Hybrid approaches like this have been adopted in several optimizers.

The results of running CPLEX 2.2 with default pricing are reported in Table 5.

Table 5: CPLEX 2.2 Default Primal Simplex

Name	Iterations	Time
schedule	100990	8521.5
energy1	22190	1831.6
initial	9719	713.7
energy2	46728	5586.2
car	26323	4348.8
continent	31380	705.0
fuel	86858	7342.3
energy3	85923	3091.1
TOTAL		32140.2

It should be noted that in [6] a number of alternative steepest-edge algorithms are studied. Among the primal algorithms, we have implemented only the one from the original paper [10], though several others seem promising. For the dual simplex method, our work has been heavily influenced by [6].

4. The Dual Simplex Method

The dual simplex method was introduced in 1954 [12]. The idea is to solve the dual problem using the primal representation. The actual algorithm may be viewed as a *working basis* method in which all the linear algebra is carried out on a naturally associated primal basis embedded within the dual basis. Thus, if the primal problem is as given in (LP), then the corresponding dual problem is

$$\text{(DLP)} \qquad \begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y + s = c \\ & y \text{ free}, s \geq 0. \end{array}$$

Assuming that the rows of A are linearly independent, it is easy to prove that all y variables may be assumed basic in any optimal solution. Thus, it may be assumed that any basis for (DLP) has the form

$$\begin{pmatrix} \mathbf{B}^T & 0 \\ A_N^T & I \end{pmatrix}$$

where I is an $(n-m) \times (n-m)$ identity matrix. B is then the associated "primal" basis. The details of the dual simplex method may be worked out by assuming a basis has the above form and decomposing the solutions of all required linear systems to ones using $\mathbf{B}$ alone. In particular, assuming that a dual feasible basis is

available, and denoting the shadow prices for the dual by $x \in Q^n$, it is easy to see that the dual reduced costs are given by x_B, with s_B the corresponding vector of dual nonbasic variables. We deduce the standard dual pricing rule of selecting the most infeasible "basic" variable x_{B_i} to leave B, equivalent to selecting s_{B_i} to enter the dual basis.

That the dual simplex method is not new is apparent. However, in the past, its use was restricted largely to integer programming. What is new is that now the dual method is considered as one of the basic alternatives for solving any linear program. That development has led to full-featured implementations of the dual simplex method. Applying our dual simplex implementation to the models from Table 1, using the pricing rule that selects the most infeasible "basic" variable, yields the results in Table 6.

Table 6: Dual Pricing–Biggest Infeasibility

Name	Iterations	Time
schedule	644015	53811.9
energy1	79827	5018.5
initial	31892	2162.6
energy2	110419	14911.3
car	50979	19163.4
continent	26795	1447.2
fuel	295345	54315.7
energy3	65601	7832.2
TOTAL		158662.8

These results do not look like progress; however, the difficulty is not the dual method itself, but the pricing rule. It is equivalent to most-negative reduced-cost pricing in the primal, a rule for which we did not even bother to present detailed results.

A much better, steepest-edge alternative to "biggest infeasibility" is presented in [6]. The idea may be described as follows. The steepest-edge paradigm looks at the rate of change of the objective as the current solution moves along an edge of the polyhedron of feasible solutions. In general, taking different representations of a given problem gives different polyhedra and different edges. If the edges are different, then the update formulas for the corresponding norms can be expected to be different, and some may well be simpler than others. For the primal, there seems to be one most-natural representation. For the dual, the situation is different. In particular, taking the simplest form of the dual, without the dual slacks, it is easy to show that the edge directions for a particular basic solution are given by the rows of the corresponding $\mathbf{B}^{-1}$ matrix. The update formulas for the 2-norms of these rows are easily derived. The extra work in implementing them is just the solution of a single linear system: $\mathbf{B}w = z$, where $\mathbf{B}^T z = e_{i^*}$. The default dual

pricing in CPLEX 2.2 is the version of dual steepest-edge with this update, using initial norms of 1. This method is called *unit steepest-edge* in [6]. The results of running the default dual on the models from Table 1 are reported in Table 7. Table 8 summarizes the best of the primal and dual results using defaults.

Table 7: Default Dual Pricing–Unit Steepest Edge

Name	Iterations	Time
schedule	21736	1773.6
energy1	23153	2321.3
initial	7843	746.6
energy2	28062	4702.0
car	15740	5120.4
continent	14952	1517.3
fuel	61956	16452.4
energy3	19463	1943.8
TOTAL		34577.4

Table 8: Best of Default–Primal/Dual Simplex

Name	Iterations	Time	Method
schedule	21736	1773.6	Dual
energy1	22190	1831.6	Primal
initial	9719	713.7	Primal
energy2	28062	4702.0	Dual
car	26323	4348.8	Primal
continent	31380	705.0	Primal
fuel	86858	7342.3	Primal
energy3	19463	1943.8	Dual
TOTAL		23360.8	

We close this section by noting that dual steepest-edge algorithms have been particularly useful in theoretical work on integer programming. Most of the leading work that is now being carried out on branch-and-cut approaches to combinatorial optimization problems would likely not be possible without them.

5. Barrier and Crossover

CPLEX 2.2 includes a C-implementation of the primal/dual barrier method with predictor-corrector, written by Irv Lustig and Roy Marsten. In addition, "basis crossover" has been implemented and is fully integrated with the barrier code.

The *basis crossover* problem is the following: Given feasible, complementary primal and dual solutions x and (y, s) for (LP) and (DLP), respectively, find a basis B that is both primal and dual feasible. Thus, B must satisfy $x_B = \mathbf{B}^{-1}b \geq 0$ and $d = c - A^T\pi \geq 0$, where $\mathbf{B}^T\pi = c_B$. Clearly these conditions imply that B is optimal for (LP). Of course, in practice one can only assume that x and (y, s) are "nearly feasible," and "nearly complementary," in the same way that the simplex method must allow for small nonnegativity violations in x_B and d.

We will not embark here upon a discussion of why one would want to crossover from an optimal barrier to an optimal basic solution. Suffice it to say that there are some situations where it is necessary and others in which it is not only unnecessary, but undesirable [1]. Fortunately, the cost of crossing over to a basis does not change the qualitative nature of our results.

Our crossover is an implementation of the proof of the main theorem in [15]. This implementation is joint work of the author and Irv Lustig. Details will be presented in a separate paper [3]. Table 9 summarizes our barrier results for the problems from Table 1. They do not differ significantly from those presented in LMS except that the OB1 preprocessor is much more effective on the problem *initial*, and this difference has a marked effect on the barrier solution time. However, since this problem is best solved by a simplex method, there is no real change in the overall results. (The simplex method applied to *initial* is not particularly sensitive to these reductions.)

Table 9: Primal/Dual Barrier with Crossover

Name	Iterations	Cholesky	Barrier Time	Crossover Time	Total Time
schedule	33	890901	2107.3	40.3	2147.6
energy1	43	137404	129.8	129.4	259.2
initial	54	7139477	39669.3	261.7	39931.0
energy2	47	559214	852.3	29.4	881.7
car	55	225442	539.2	609.5	1148.7
continent	67	318876	723.3	51.1	774.4
fuel	58	1482722	4831.3	224.6	5055.9
energy3	82	350980	796.0	62.0	858.0
TOTALS			49648.5	1408.0	51056.5

6. Summary

In Tables 10 and 11, we summarize the results of the previous sections. For a test set of reasonably large problems, we obtain approximately a 40–fold improvement in solution time relative to a code that did not solve two of the problems (in the allowed number of iterations), a code that was itself an improvement over several standard

codes. Remarkably, this improvement exceeds the improvements in workstation speed over that period. This result was obtained by comparing a Sun 3/150 to a SPARCstation 10/model 41, for our application. Moreover, it is our opinion that the results would not be substantially different if the tests had been carried out on other similar sets of larger test problems. However, the results would be different using smaller problems, such as those in the NETLIB set, where the effect of the algorithmic improvements has not exceeded those in the computing machinery.

Table 10: Progress Summary

Implementation	Time
CPLEX 1.0	[a]447957.0
CPLEX 1.0 + Preprocessing	[b]102848.4
Default Primal Simplex[c]	32140.2
Best of Default Primal/Dual Simplex[c]	23360.8
Best of Default Barrier/Simplex[c]	11395.8

[a] Two models not solved

[b] One model not solved

[c] CPLEX 2.2

Table 11: Running-Time Summary[a]

Name	Simplex		Barrier +	Best	Method
	Primal	Dual	Crossover		
schedule	8521.5	1773.6	2147.6	1773.6	Dual
energy1	1831.6	2321.3	259.2	259.2	Barrier
initial	713.7	746.6	39931.0	713.7	Primal
energy2	5586.2	4702.0	881.7	881.7	Barrier
car	4348.8	5120.4	1148.7	1148.7	Barrier
continent	705.0	1517.3	774.4	705.0	Primal
fuel	7342.3	16452.4	5055.9	5055.9	Barrier
energy3	3091.1	1943.8	858.0	858.0	Barrier
TOTALS	32140.2	34577.4	51056.5	11395.8	

[a] SPARCstation 10/model 41 using CPLEX 2.2 with default settings

How do our results on the models from Table 1 compare to those in LMS? They are qualitatively somewhat different. In LMS, barrier was clearly faster on 7 of the 8 problems. Our results show barrier much faster on 4 of the 8 problems, somewhat faster on 1 (primal steepest-edge solves *fuel* in 5500 seconds), slightly slower on 2, and (as in LMS) much slower on the model *initial*. One reason for the difference, and the much more even performance of the simplex method on the models for which it did not win, was the use of dual steepest edge. This option is presumably

available in OSL, in view of [6], but was not used in LMS since it was apparently not the default. It should also be noted that the runs in LMS were made on IBM RS6000 Powerstations, an architecture more favorable to barrier methods.

Because of our feeling that the models in Table 1 may not provide the most accurate picture of the "large-model" landscape, we conclude by presenting results for a different set of large models. Over the past several years, the author has collected an assortment of real models both from academia and industry. For the present test only models with at least 100,000 nonzeros in the constraint matrix were used. We also considered at most one instance from a given model type, when it was clear two models were closely related. For example, if there were 1-, 2-, 4- and 8-period instances of a given model, we took only the 8-period instance. The sizes of the models we selected are listed in Table 12 and their sizes after preprocessing in Table 13. Table 14 summarizes the results of solving these models using primal simplex, dual simplex, and barrier plus crossover.

In practice, when solving real models, especially larger ones, some tuning almost always occurs, exploiting previous experience and the availability of smaller instances from the same class. However, tuning was largely ignored in the construction of Table 14. The times reported for the simplex method are those with defaults, if default runs were ever made. For the barrier method, when non-default runs were made, they were preceded by an attempted run with defaults. The boldface entries designate the winning method. See the remarks following Table 14 for further information. The results in Table 14 indicate to us that there is no single dominate method for the solution of large-scale linear programming problems.

Table 12: Large Models

Name	Rows	Columns	Nonzeros
binpacking	42	415953	3526288
distribution	24377	46592	2139096
forestry	1973	60859	2111658
electronics	41366	78750	2110518
maintenance	17619	79790	1681645
finance	26618	38904	1067713
unknown1	43687	164831	722066
finland	56794	139121	658616
crew	4089	121871	602491
gams	8413	56435	435839
roadnet	463	42183	394187
multiperiod	22513	99785	337746
unknown2	44211	37199	321663
appliance	25302	41831	306928
food	27349	97710	288421
airfleet	21971	68136	273539
energy3	27145	31053	268153
unknown5	19168	63488	255504
military	33874	105728	230200
fuel	18800	38540	219880
continent	10377	57253	198214
energy4	25541	19027	194797
car	43387	107164	189864
unknown3	2410	43584	166492
stochastic	28240	55200	161640
energy2	8258	21200	145329
4color	18262	23211	136324
unknown4	1088	2393	110095

Table 13: Large Models after Preprocessing

Name	Rows	Columns	Nonzeros
binpacking	22	140483	1223824
distribution	22890	22560	1775310
forestry	1888	60754	2109833
electronics	16781	30291	854901
maintenance	17619	79790	1681554
finance	26618	38855	1065664
unknown1	17438	118829	566536
finland	5778	75786	319851
crew	1587	121624	602245
gams	8152	56174	425710
roadnet	462	42182	394186
multiperiod	21022	94541	319798
unknown2	11115	33779	222189
appliance	8144	23638	179402
food	10938	70499	219458
airfleet	18841	65055	225175
energy3	9521	28649	188393
unknown5	14177	57392	322864
military	28221	99915	213210
fuel	10613	24854	174703
continent	6935	45784	158404
energy4	5305	17568	88364
car	41763	102208	183284
unknown3	2410	43584	166492
stochastic	16920	26088	85800
energy2	6845	18307	128948
4color	11831	16835	154905
unknown4	1088	2393	110095

Table 14: Running Times of Large Models

Name	Simplex		Barrier +
	Primal	Dual	Crossover
binpacking	**29.5**	62.8	560.6
distribution	**18568.0**	*Not run*	$\|AA^T\| = 12495446$
forestry	**1354.2**	1911.4	2348.0
electronics	**312.2**	4950.0	$\|AA^T\| = 4595724$
maintenance	[a]57916.3	[a]89890.9	**3240.8**
finance	9037.4	**2563.8**	$\|AA^T\|$ > memory
unknown1	[a]59079.1	*Not run*	[a]**29177.6**
finland	4594.1	7948.6	[a]**4586.6**
crew	7182.6	16172.1	**1264.2**
gams	104278.0	86445.8	**32083.3**
roadnet	380.3	1190.8	**318.0**
multiperiod	26771.5	**5023.7**	18362.7
unknown2	2004.6	**961.2**	1131.0
appliance	**861.0**	1599.2	[a]2161.6
food	4789.0	1967.6	**974.3**
airfleet	[a]71292.5	[a]108015.0	**37627.3**
energy3	3091.1	1943.8	**858.0**
unknown5	427.4	516.6	**412.0**
military	25184.0	**5606.3**	[b]$\|L\| = 6637271$
fuel	7342.3	16452.4	**5055.9**
continent	**705.0**	1517.3	774.4
energy4	**68.7**	83.8	138.2
car	4348.8	5120.4	**1148.7**
unknown3	**46.1**	10212.4	3530.3
stochastic	2670.1	**930.0**	[a]1700.7
energy2	5586.2	4702.0	**881.7**
4color	[a]**45870.2**	*Not run*	$\|L\| = 44899242$
unknown4	543.4	**381.0**	2153.4

[a] Not run with defaults; see remarks in Section 6.1

[b] $|L|$ denotes the size of the Cholesky factors

6.1 Remarks

1. The quantity $|A^TA|/2$ is a lower bound on the size of the Cholesky factors, when they are computed symbolically. In the cases where $|A^TA|$ or $|L|$ is specified in Table 14, it was clear from the size of that number that the barrier method could not compete with the simplex times, and no run was made. In none of these cases was this phenomenon due to a manageable number of "dense columns."

2. The models *maintenance*, *unknown1*, and *airfleet* were all suspected in advance to be best solved using steepest-edge. That was the only pricing algorithm used. For the dual, that meant exact initial norms were computed, rather than taking initial norms of 1.

3. The models *unknown1* and *finland* exceeded the barrier iteration limit of 100, and had to be restarted with a larger iteration limit.

4. The aggregation part of the preprocessor was turned off in solving *stochastic* with the barrier method. That prevented the re-joining of "split columns" and improved the solution time significantly.

5. A total of 28 dense columns were removed from *appliance*. Without doing so, $|L|$ exceeded $3,000,000$.

6. In the cases of *distribution* and *unknown1* dual simplex was run until its running time exceeded the best known solution time, and then terminated.

7. *4color* originated from work on a possible new proof of the 4-color theorem. The zero solution was known to be feasible, and the authors of the model conjectured that this solution was optimal. It was therefore natural to perturb the problem before solving. As it turned out, zero was not optimal.

8. Preprocessing time for the *binpacking* model dominated the solution times and was not included, in order to more clearly show the differences in the algorithms. It is also worth noting that the solve times without preprocessing were not significantly worse than those with preprocessing.

9. One striking example of the possible effects of tuning is provided by the model *multiperiod*. This is a multi-commodity flow problem. Using the network simplex method to get a crash starting basis, followed by the application of the dual simplex method, yields a total solution time of 880.0 seconds.

References

[1] K. D. Anderson and E. Christiansen, 1993. Limit Analysis With the Dual Affine Scaling Algorithm, Institut for Matematik og Datalogi, Odense Universitet (Denmark) Technical Report No. 0903–3920 (19 pp.).

[2] R. E. Bixby, 1992. Implementing the Simplex Method: The Initial Basis. *ORSA Journal on Computing 4:3*, 267–284.

[3] R. E. Bixby and I. J. Lustig, 1993. An Implementation of a Strongly-Polynomial Algorithm for Basis Recovery. *In preparation.*

[4] A. L. Brearly, G. Mitra and H. P. Williams, 1975. Analysis of Mathematical Programming Problems Prior to Applying the Simplex Method. *Mathematical Programming 8*, 54–83.

[5] V. Chvátal, 1983. *Linear Programming*, W. H. Freeman, New York.

[6] J. J. Forrest and D. Goldfarb, 1992. Steepest-Edge Simplex Algorithms for Linear Programming. *Mathematical Programming 57*, 341–374.

[7] J. J. H. Forrest and J. A. Tomlin, 1972. Updating Triangular Factors of the Basis to Maintain Sparsity in the Product-Form Simplex Method. *Mathematical Programming 2*, 263–278.

[8] D. M. Gay, 1985. Electronic Mail Distribution of Linear Programming Test Problems. *COAL Newsletter 13*, 10–12.

[9] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, 1989. A Practical Anti-Cycling Procedure for Linearly Constrained Optimization, *Mathematical Programming 45*, 437–474.

[10] D. Goldfarb and J. Reid, 1977. A Practicable Steepest-Edge Simplex Algorithm. *Mathematical Programming 12*, 361–371.

[11] P. M. J. Harris, 1973. Pivot Selection Methods of the Devex LP Code. *Mathematical Programming 5*, 1–28.

[12] C. E. Lemke, 1954. The Dual Method of Solving the Linear Programming Problem. *Naval Research Logistics Quarterly 1*, 36–47.

[13] H. M. Markowitz, 1957. The Elimination Form of the Inverse and Its Application to Linear Programming. *Management Science 3*, 255–269.

[14] R. E. Marsten, 1981. The Design of the XMP Linear Programming Library. *ACM Trans. Math. Software 7*, 481–497.

[15] N. Megiddo, 1991. On Finding Primal- and Dual-Optimal Bases. *ORSA Journal on Computing 3:1*, 63–65.

[16] B. Murtagh, 1981. *Advanced Linear Programming*, McGraw Hill.

[17] B. A. Murtagh and M. Saunders, 1987. Minos 5.1 User's Guide, Technical Report SOL 83-20R, Systems Optimization Lab, Stanford University, Stanford, California.

[18] U. H. Suhl and L. M. Suhl, 1990. Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases. *ORSA Journal on Computing 2*, 325–335.

[19] P. Wolfe, 1965. The Composite Simplex Algorithm. *SIAM Review 7:1*, 42–54.