Recovering an Optimal LP Basis from an Interior Point Solution

Robert Bixby
Matthew J. Saltzman

CRPC-TR91184
October, 1991

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Revised March, 1992.

Recovering an Optimal LP Basis from an Interior Point Solution

Robert E. Bixby
Department of Mathematical Sciences
Rice University
Houston TX
713/527-4805 bixby@rice.edu

Matthew J. Saltzman
Department of Mathematical Sciences
Clemson University
Clemson SC
803/656-3434 mjs@clemson.edu

Keywords: Programming (linear programming, interior point algorithms, simplex method, basis recovery); Analysis of algorithms (data structures, algorithms for basis recovery).

Abstract

An important issue in the implementation of interior point algorithms for linear programming is the recovery of an optimal basic solution from an optimal interior point solution. In this paper we describe a method for recovering such a solution. Our implementation links a high-performance interior point code (OB1) with a high-performance simplex code (CPLEX). Results of our computational tests indicate that basis recovery can be done quickly and efficiently.

In recent years, interior point methods for linear programming have generated much excitement. High-performance interior point codes such as OB1 (OB1 is a trademark of XMP, Inc.) compete with advanced simplex codes such as CPLEX (CPLEX is a trademark of CPLEX, Inc.) to solve problems on a scale considered completely impractical only a few years ago. Currently, each class of methods outperforms the other on some types of problems, and neither has proved to be superior overall.

One important challenge in the implementation of interior point algorithms is the development of methods to construct an optimal basis. Whereas the simplex method naturally terminates with an optimal basic pair of primal and dual solutions, interior point algorithms provide only a "nearly" (strictly) complementary primal-dual pair. In fact, if multiple primal or dual optima exist, the interior point solution is likely be a point in the "middle" of the optimal face [5, 11]. This is a significant drawback in some applications, since the current state of the art allows post-optimality analysis to proceed only from a basic primal-dual optimal pair. (See [1] for an investigation into the potential for post-optimality analysis in the context of interior point methods.)

Of particular interest is the reoptimization of a modified problem, given the optimal solution to the unmodified problem. This situation arises frequently in integer programming, where branching or adding cutting planes causes the last known solution to become infeasible. It is also an issue in nonlinear programming methods that solve sequences of LPs, and in many planning applications where problem structure remains the same, but coefficient values change from one period to the next.

An additional difficulty with current implementations of interior point methods is their numerical stability near the optimal solution. The linear system solved at each iteration of the Newton barrier algorithm implemented in OB1 is singular at a strictly complementary point, and as the solution is approached, the system becomes progressively more ill-conditioned. This may lead to problems such as difficulty meeting convergence criteria, or unacceptably large residuals. Although future developments in interior point techniques may eventually overcome these difficulties, recovery of a basic optimum is currently the best method available to ensure accuracy when the interior point methods encounter difficulties.

One early attempt to recover basic optima from interior optima by "crossing over" to the simplex method is described in [10]. This experiment applied an extension to Marsten's XMP simplex code to handle "superbasic" variables and combat stalling to the solution generated by a dual affine interior point algorithm. The dual affine algorithm produces a dual interior solution and a nearby complementary primal solution, which may not be primal feasible. The (then-current) extended XMP accepted a superbasic solution (see below), and set up an all-logical basis. Variables near their bounds were locked at their bounds, and the simplex method was started. After an optimal solution was found, the locked variables were released, and the problem was reoptimized. Finally, any remaining superbasic variables were moved to their bounds or pivoted into the basis. This experiment met with decidedly mixed success. Overall, the basis recovery phase took about 45% of the interior point time and about 30% of the time required to solve the problems from scratch using XMP's simplex algorithm. On some problems, basis recovery took longer than the interior-point phase.

The approach described in this paper is philosophically similar to that of [10], but it starts with the output of a primal-dual interior algorithm, generates an advanced starting basis, and uses different rules for choosing and manipulating superbasic variables. We demonstrate that the combination of simple crossover rules, switching from a primal-dual interior algorithm and a robust simplex implementation yields a very effective basis recovery method. The critical factors in designing a successful crossover are the identification of the positive variable in a primal-dual complementary pair, and the construction of a stable starting basis.

We begin in Section 1 with a review of essential concepts. In Section 2, we describe the rules for selecting variables to appear in a proposed starting basis. In Section 3, we describe the techniques we apply to repair the initial basis in case the proposed basis is invalid. Section 4 describes the pivoting strategy to move to a basic optimum. Recent results by Megiddo [12] have shown that, given a complementary primal-dual optimal pair, there is a strongly polynomial algorithm for basis recovery. In Section 5, we describe an interpretation of Megiddo's result and compare this method with our proposal. Finally, Section 6 describes a comprehensive set of tests on the Netlib test set [3], and Section 7 presents final remarks.

1 Basic Concepts

We consider the bounded-variable linear program (LP):

$$\begin{array}{rl} \text{Minimize} & c^T x \\ \text{subject to} & Ax = b \\ & l \leq x \leq u \end{array} \qquad (1)$$

where A is an $m \times n$ matrix. Arbitrary bounds, including $l_j = -\infty$ and $u_j = \infty$, are allowed.

When presented with a problem in this form, OB1 transforms it into a problem in which all variables satisfy $0 \leq x_j \leq u_j$, with $u_j = \infty$ allowed. The problem is transformed back to its original form before it is handed off to the crossover routine.

A *normal basic solution* to (1) is described by a partition of the index set $\{1, \ldots, n\}$ into subsets $\mathcal{B}, \mathcal{L}, \mathcal{U}$ and $\mathcal{F}$, such that the following properties hold (for convenience, we define $\mathcal{N} = \mathcal{L} \cup \mathcal{U} \cup \mathcal{F}$):

- $|\mathcal{B}| = m$ and the submatrix B of A, consisting of the columns $\{A_{\cdot j} \mid j \in \mathcal{B}\}$ is nonsingular.
- $\mathcal{L} \subseteq \{j \in \mathcal{N} : l_j > -\infty\}$;
- $\mathcal{U} \subseteq \{j \in \mathcal{N} : u_j < \infty\}$;
- $\mathcal{F} = \{j \in \mathcal{N} : l_j = -\infty \text{ and } u_j = \infty\}$.

  The variables are assigned the values

$$\begin{aligned} x_j &= \begin{cases} l_j & j \in \mathcal{L} \\ u_j & j \in \mathcal{U} \\ 0 & j \in \mathcal{F} \end{cases} \quad \text{for all } j \in \mathcal{N} \\ x_B &= B^{-1}(b - Lx_L - Ux_U - Fx_F) = B^{-1}(b - Nx_N) \end{aligned}$$

  (where L, U, F and N are defined similarly to B).

A *superbasic solution* to (1) is described by a similar partition, with the following modification:

- $j \in \mathcal{F}$ is permitted for any j such that $l_j < u_j$, and for $j \in \mathcal{F}$, any value for x_j is permitted, as long as $l_j < x_j < u_j$.
- the solution is specified by the sets $\mathcal{B}$, $\mathcal{L}$, $\mathcal{U}$ and $\mathcal{F}$, and the value of x_F. The value of x_B is computed as above.

Note that any feasible solution to (1) can be described as a superbasic solution involving any basis $\mathcal{B}$ (with B nonsingular), and suitably-chosen $\mathcal{L}$, $\mathcal{U}$, $\mathcal{F}$ and x_F.

1.1 The Interior Point Algorithm

The interior point code used in this experiment is OB1, which implements a primal-dual barrier method with a predictor-corrector enhancement. The details of the algorithm are described in [9], and the details of the Cholesky factorization are described in [7]. Reordering of the rows of A to control fill-in in the factorization of AA^T is accomplished using the Multiple Minimum Degree heuristic [8]. The only modification to OB1 itself was the substitution of our own crossover routine for the routine which crosses over from the interior-point phase to the XMP simplex code.

1.2 The Simplex Algorithm

The simplex code we use is CPLEX, modified to handle superbasic primal variables. These are treated as free variables, except that, if one is selected to enter the basis, the appropriate bound participates in the distance computation for selecting a leaving variable, and it may move to that bound without requiring a pivot.

At the end of a simplex run, any remaining superbasic variables are forced to move to a bound or enter the basis, and free variables are forced to enter the basis or move to zero. CPLEX terminates with a normal basic feasible solution.

Code to accept a starting superbasic solution and generate a usable starting basis is included, and this is described in detail below. The superbasic code uses partial/multiple reduced gradient pricing or a hybrid strategy mixing *Devex* [6] and partial/multiple pricing, and handles structural or slack variables between their bounds. Variables that violate their bounds are permitted only if they are in the basis.

2 Selecting a Candidate Basis

The first step in our crossover process is to identify which primal variables are to be considered away from their bounds. This is accomplished as follows:

- For each x_j such that $l_j < u_j$, we compute $s_j = \min\{x_j - l_j, u_j - x_j\}$. The variables are sorted in decreasing order of s_j. If x_j is free, then $s_j = |x_j|$. Fixed variables are excluded from consideration.
- Variables are examined in order of decreasing s_j. A variable x_j is considered to be between its bounds as long as $s_j > \epsilon_1$, where

$\epsilon_1 = 10^{-5}$ was used in all our computational tests.

- The first m such variables are marked as candidates for the initial basis. Any remaining variables with $s_j > \epsilon_2$ are marked as superbasic, where $\epsilon_2 = 10^{-4}$ in our tests. The case where fewer than m candidates are available is treated below. All variables with $s_j \leq \epsilon_2$ are marked as being at the corresponding upper or lower bound, whichever is closer to the current value.

The status lists thus constructed are used as a starting point in the construction of the initial basis. Once an acceptable starting basis has been generated, the remaining superbasic variables are checked again, and set at the nearest bound if $s_j \leq \epsilon_2$.

3 Forming the Starting Basis

As we have pointed out, any superbasic solution can be described in terms of any basis. Our task is to choose a basis in which to describe the interior solution. This basis should contain a maximal set of variables strictly away from their bounds. The difficulty is that when we compute x_B as the solution to $Bx_B = b' = b - Nx_N$, the resulting residual $b' - Bx_B$ may be acceptably small (since Gaussian elimination effectively minimizes the residual), but the computed x_B may not accurately approximate the x_B that is passed from the interior algorithm. Even if the residual is within acceptable limits, x_B may be highly sensitive to perturbations caused by moving nonbasic variables to bounds. (See [4], for example, for a discussion of these issues.) An inaccurately-computed x_B may appear to have large infeasibilities (with respect to bounds), and may cause other numerical difficulties for the simplex algorithm. Thus it is extremely important to construct a very well-conditioned basis, even if this means rejecting variables that appear desirable, and that would be acceptable in other circumstances. We construct such a basis using the method described in this section.

The interior solution and the status list constructed above are used to generate a candidate basis (partial basis, in case fewer than m candidates are found). CPLEX attempts to factor this matrix, as described in [2] (see also [14]).

One aspect of the factorization that is of interest here is the *singularity tolerance*, τ. The LP is scaled by CPLEX as part of its initialization, so that $\max_{i,j} |a_{ij}| \leq 1$. The singularity tolerance gives the minimum absolute magnitude of an acceptable pivot entry. For this initial factorization attempt. we select $\tau = 10^{-3}$ (compared to the default of $\tau = 10^{-8}$ for normal operation). Columns for which no acceptable pivot can be found are rejected. Each of the remaining columns is said to *cover* the row in which its pivot entry appears.

If the basis is incomplete, either because of an insufficient number of candidates or because of rejected columns, it is extended as follows: any non-basic slack variable that covers an uncovered row is included; other candidates are taken from the list of variables initially marked superbasic (not including rejected variables), if any, then from the variables at bounds, according to criteria described in [2]. The factorization is attempted again. Any uncovered rows after this pass are covered by artificial variables. The factorization and addition of artificial columns may be repeated up to ten times, if required to produce a complete, successfully factored basis.

Once a basis has been successfully constructed. the value of x_B is computed. and the resulting solution becomes the initial candidate starting superbasic solution. If the total scaled infeasibility of this solution is greater than 1.0. this is taken as an indication that the chosen basis is too ill-conditioned to be acceptable. In this case. the entire process is repeated. starting with the current candidate basis. with $\tau = 10^{-1}$.

The test for infeasibility after the first successful factorization (with $\tau = 10^{-3}$) essentially approximates a test for the accuracy of the computed x_B. Although this test does not distinguish between infeasibilities due to inaccuracy and infeasibilities due to superbasic variables being pushed to bounds. it is quite effective in practice. If the infeasibility is small enough. we accept the current basis. If not. we demand a better-conditioned basis. by applying the selection process again with $\tau = 10^{-1}$. If the new solution is still not satisfactory. then the infeasibility is probably not due to ill-conditioning, and we proceed using the solution in any case. An improvement to this method would be to use large infeasibility as an indicator to adjust both τ and ϵ_2, *i.e.*. to attempt to control infeasibility due to ill-conditioning as well as movement of nonbasic variables. These issues remain a topic for future investigation.

4 Progress Toward a Normal Basic Optimum

After an acceptable basis has been constructed and variables moved to bounds as described above. the remaining nonbasic variables are made superbasic at their interior-solution values. Then x_B and the reduced costs corresponding to the current basis are computed. Because of the adjustments to the values of x_L and x_U, the computed values of x_B may be slightly different from their interior-solution values, and may even no longer be feasible. Also, the reduced costs associated with the chosen basis may not satisfy the optimality condition. Nevertheless, the superbasic solution we have constructed can be used to start phase I of the simplex method.

We apply CPLEX's implementation of Wolfe's piecewise-linear phase-I algorithm [15] (if necessary), beginning with the constructed superbasic solution. When a primal feasible solution is reached, we apply CPLEX's phase-II algorithm using partial/multiple pricing. A superbasic candidate to enter the basis is treated the same way as a free variable, except that it may reach a bound before any basic variable, in which case it is moved into $\mathcal{L}$ or $\mathcal{U}$, as appropriate, rather than entering the basis.

When all feasibility and optimality conditions are satisfied, there may remain superbasic variables with reduced cost close enough to zero to be considered optimal. The last phase of the algorithm moves each such variable in turn to a bound: if the variable has only one bound, it is moved there; if it has upper and lower bounds, it is moved to the nearest bound (or to the opposite bound if the pivot is rejected). Free variables are moved to zero. During each step of this process, if a basic variable reaches its bound, it is exchanged in the basis with the superbasic variable being moved. This step is performed last, because superbasic variables often move during the reoptimization phase anyway. Every such movement results in the removal of one superbasic variable from the list. Thus, there may be significantly fewer superbasic variables at the end of the reoptimization process than at the beginning.

5 Complexity and Megiddo's Method

In [12], Megiddo describes a strongly polynomial algorithm for recovering an optimal basic solution from any complementary primal and dual feasible pair of solutions. He also shows that, if a strongly polynomial algorithm existed for recovering a basic optimal solution from a primal or dual optimal solution alone, then there would be a strongly polynomial algorithm for the general LP problem.

Given an exact complementary pair of primal and dual optimal solutions, Megiddo proposes inserting va[illegible]bles into the basis one at a time, as follows (for simplicity, we assume in this section that $0 \leq x_j \leq \infty$):

1. Insert positive primal variables until a maximal linearly independent set of columns has been included. Each time a column corresponding to a positive variable is found to be linearly dependent on columns already in the basis, move the variable to zero or pivot it into the partial basis if a basic variable goes to zero first. After this step all positive variables have been inserted in the basis or set to zero.

2. If any zero-valued variables have duals equal to zero, include as many as possible with columns linearly independent of those in the partial basis.

3. Attempt to include each of the remaining variables until a complete basis has been constructed. For each zero-valued variable included with positive dual, move the dual to zero. If a nonbasic dual (with respect to the primal basis) goes to zero, pivot the corresponding variable into the partial basis.

When these steps are completed, a complete basis has been constructed, all basic dual variables are zero and all nonbasic primal variables are zero. Since (under exact arithmetic), every step increases the size of the basis or reduces the number of positive nonbasic variables, it follows that the number of iterations is bounded by n.

Megiddo's method differs from the method presented here in two ways. First, rather than build the basis one column at a time, we construct a full basis at the start and use the usual simplex pivoting mechanism to recover our solution. Megiddo's algorithm can be easily modified to accommodate this strategy, as described below. Second, our method does not make use of the dual solution provided. We discuss the implications of each of these differences next.

Megiddo's method can be modified to start by constructing a full basis, and then use normal simplex pivots. Starting with a complementary primal-dual pair of solutions, perform the following steps:

1. Find a maximal linearly independent set of columns corresponding to variables with positive primal values (*i.e.*, dual values of zero). Any remaining positive primal variables will be superbasic.

2. Fill out the basis with any linearly independent columns from among the remaining variables. First include variables with primal and dual values both zero, then those with nonzero dual values.

3. For each of the remaining superbasic variables, move the variable to a bound, or until a variable in the basis reaches a bound. If the latter occurs, exchange the variables.

4. For each basic variable with positive dual value, move the dual value to zero, or until a nonbasic dual variable reaches zero (as in the dual simplex method). If the latter occurs, exchange the variables.

At the completion of Step 2, a basis matrix has been constructed, with respect to which the original solutions are primal- and dual-superbasic, *i.e.*, some nonbasic primal variables may be positive, and some basic dual variables may be positive. After each iteration in Step 3, there will be one fewer nonbasic positive primal variable. If the exchange occurs, the incoming variable must replace a variable in the basis with a positive value, since the basis contains a maximal linearly independent set of these. At the conclusion of Step 3, the primal solution is basic and feasible, but the corresponding reduced costs may violate the optimality conditions. At this point the original dual solution may be viewed as a superbasic starting solution for the dual simplex method. Each iteration of step 4 reduces the number of nonzero basic dual variables by one, without sacrificing dual feasibility. At the conclusion of this step, the algorithm will produce a normal basic optimal solution to the LP. (Note: if there are nonbasic free variables, they can be moved to zero or pivoted into the basis in Step 3.) The algorithm is guaranteed to run in at most n iterations in Steps 3 and 4.

The other difference of our method from Megiddo's method is that we do not make use of the dual solution provided. Megiddo shows that a basis-recovery algorithm using only primal or dual information could not have strongly polynomial worst-case running time unless a strongly polynomial algorithm for the general LP problem exists. Nevertheless, we will show that our algorithm performs comparatively well in practice. Although we have not implemented the superbasic dual simplex method, we can still use Megiddo's algorithm as a suggestive standard for comparison. This is because once a starting basis has been constructed, the number of pivots required by the algorithm (under the assumption of ideal arithmetic) can be easily estimated. What cannot be estimated is the fraction of primal and dual pivots that will require an exchange of variables in the basis. This can have a significant impact on the total running time of the algorithm, since pivots requiring an exchange are more costly. On the other hand, there is no *a priori* reason to expect this fraction to be lower than for our method.

It is unlikely that Megiddo's method could be implemented in a way that meets the guarantees of the analysis, for two reasons. The theoretical analysis begins with an exact optimal primal-dual pair, and assumes that the set of primal variables not at bounds can be identified precisely. Any practical implementation will start with an interior solution that is only approximately complementary (and perhaps only approximately feasible). Adjusting values of primal and dual variables to achieve complementarity may adversely affect primal or dual feasibility. The theoretical analysis also assumes that linear independence is a strictly combinatorial notion. Again, this cannot be true for linear systems represented as floating point numbers on digital computers. The condition of the basis is a critical problem, and nonsingular matrices can appear to be numerically singular or *vice versa.* If incorrect decisions are made with regard to inclusion of variables in the basis, a maximal set of linearly independent positive variables may not be included. In this eventuality, reduction in the number of primal or dual superbasic variables can no longer be guaranteed. If the basis is ill-conditioned, the computed solution may not be accurate (or feasible) and ratio tests may not be performed accurately.

For each of the problems we solved, we estimated the number of Megiddo pivots to be expected under ideal conditions. The estimate was computed, after a satisfactory initial basis had been constructed, as the number of superbasic primal variables plus the number of superbasic dual variables. This number is reported together with the times and number of pivots required by our algorithm, in Table II. We observe that on only 14 problems does the number of actual pivots performed by our method exceed the number predicted. Overall, the number of pivots predicted exceeds the number actually performed by about 37%. If we assume that the average cost of an iteration is the same for both methods on each problem, the projected total solution time from Megiddo's method is higher by 47% than the actual time. As the predictions are for ideal conditions, we can expect that the actual cost for an implementation of Megiddo's method using the same starting basis would be even higher.

It is apparent that, in most cases, our algorithm requires significantly fewer pivots than the Megiddo method. We might take this to be another instance of a familiar syndrome in linear programming: that algorithms with worst-case behavior that is polynomial often exhibit their worst-case behavior, whereas more daring algorithms may perform better on most practical problems, even if their worst-case behavior is not guaranteed to be good. On the other hand, there are likely to be certain problems for which Megiddo's method will outperform our method by a significant margin (in our test set, *wood1p* is a potential example). It is not apparent how to make the distinction *a priori*.

6 Computational Tests

We tested the proposed basis recovery algorithm on the problems in the Netlib test set [3]. Tests were run on a Sun 4/490 with 64 megabytes of RAM, running SunOS 4.1.1. OB1 was compiled using the Sun `f77` compiler, version 1.3.1. and the command line performance options `-O4 -cg89 -dalign -libmil`. CPLEX was compiled with the Sun `cc` compiler supplied with the operating system. using the same command line options.

Two complete sets of tests were run. using different termination criteria for the interior point phase. In both tests. the default settings for OB1's parameters were used. except:

- The problem *fit2p* could not be solved using a reasonable amount of memory without resorting to a technique for handling dense columns. We used the OB1 parameter `dense 10`, which effectively removes all 25 columns in this model with more than one nonzero element from the Cholesky factorization, and handles them separately using a Schur complement technique [9].

- In the first set of tests, OB1 was terminated when the relative duality gap reached the default tolerance of 10^{-8}. To test the effect of a more accurate interior solution, the second set of tests was run until the duality gap reached 10^{-12}. OB1 occasionally terminated the high-accuracy run due to numerical difficulties encountered before the desired accuracy was achieved. In problems *perold*, *fffff800* and *nesm*. the objective function values differed from the standard-accuracy run by 10^{-8}.

In all cases, the CPLEX parameters were left at their defaults, except for the choice of pricing strategy. The results of comparisons among these choices are discussed below.

Table I lists the vital problem statistics and the results of the OB1 runs. The original number of rows, columns and nonzeros in the constraint matrix is given. as well as the number of rows, columns and constraint nonzeros after preprocessing by OB1's `crush` procedure. The total CPU time for preprocessing includes the `crush` phase plus the time for reordering rows of the constraint matrix to control fill-in in the Cholesky step. The number of iterations, CPU seconds, objective function value, and a measure of how strict the complementarity of the solution is, is given for the standard accuracy run. Individual problem statistics are not given for the high-accuracy run. The results are summarized below. The complementarity measure is computed by calculating the largest p such that the floating point value $a + 10^{-p}b = a$, where a and b are the larger and smaller of s_j and z_j (the reduced cost), respectively. The ratio of the minimum 10^{-p} to *machine epsilon* is given in the table. Values close to one indicate difficulty determining whether s_j or z_j should be considered nonzero.

Table II lists the results of the CPLEX phase. Times are divided between "setup time," which includes loading and scaling the problem and constructing the starting basis, and "solve time," which includes phase I and II pivots, plus the pivots to eliminate remaining superbasic variables. The pivot counts are given, as well as the number of pivots which involved an exchange of variables in the basis and the number which involved moving a nonbasic variable to a bound, for the simplex and superbasic cleanup phases. The estimated number of Megiddo primal and dual pivots is also listed. For the setup phase, the number of factorizations is listed. Where two numbers are listed in this column, the first is the number of factorizations with $\tau = 10^{-3}$, and the second is the number with $\tau = 10^{-1}$.

The total time for OB1 for the entire test set at default accuracy (including setup and solution time) is 6063 seconds, and the total CPLEX time (including setup and solution time) is 584 seconds, or 9.6% of the total interior point time. For the high-accuracy interior point run, the total OB1 time is 7426 seconds and the total CPLEX time is 432 seconds or 5.8% of the interior point time. The overall times are 6647 seconds for standard accuracy and 7848 seconds for high accuracy. Thus, the more accurate interior point solution is helpful for the basis recovery phase, but the additional cost of obtaining the high-accuracy solution to begin with more than offsets the savings in the basis-recovery phase. Note that almost half the CPLEX time is due to one problem, *stocfor3*. If this problem is dropped from the test set, basis recovery time drops to only 5% of interior point time. Our experience with *stocfor3* is described below.

These results can be contrasted with the results of [10] on a subset of 26 of these problems. The overall cost of basis recovery in that paper was 46% of the total interior point time. For the same subset of problems, the method presented here takes 18% of the interior point time. Furthermore, the dual affine interior algorithm and quotient minimum degree reordering heuristic of [10] are significantly slower than the implementation in OB1. In fact, it is interesting to see that the total dual affine time for the tests in [10] was 8075 seconds, and the total recovery time was 3731 seconds (on a VAX 8600 with no floating point accelerator). The same test on the Sun 4/490 using OB1's primal-dual predictor-corrector ran in 385 seconds and the CPLEX basis recovery time was 57 seconds. Although the effect of the change of machine is not clear, the total test now runs almost 27 times faster than the earlier version. The interior point portion of the test is 21 times faster, and the recovery portion is over 65 times faster.

6.1 Pricing

CPLEX's superbasic support is integrated with both partial/multiple pricing and *Devex* pricing. We tested both partial/multiple pricing and a hybrid pricing strategy using *Devex* and partial/multiple pricing. Applying the hybrid strategy did not result in significant reductions in the number of iterations. In particular, the reduction in iteration counts did not compensate for the increased cost per iteration. We conjecture that the improved direction-selection criteria of the hybrid algorithm produces little or no benefit since the starting solution is already on the optimal face. Also, the number of iterations required in the basis-recovery phase is small enough that the setup cost for *Devex* pricing will not be amortized over the pivots.

6.2 Perturbation

Of the total 584 seconds reported in the previous section for basis recovery on the Netlib set, 288 seconds are attributable to two problems. *Degen3* took 934 iterations and 27 seconds and *stocfor3* took 2240 iterations and 262 seconds. With these two problems removed, the total basis-recovery time would have been 322 seconds, and the total OB1 time would have dropped only slightly to 5701 seconds.

The behavior of *stocfor3* can be explained as follows (*degen3* is similar). The crushed version of this model has a basis of size 16643. Taking the result of our accurate run, the OB1 solution has 13551 primal variables (including slacks) with values greater than 10^{-3}. All other variables have values less than 10^{-10}, and only 1322 have dual values less than 10^{-2}. Thus, ignoring artificial variables and using any reasonable tolerances for primal and dual zeroes, there will be at least $16643 - (13351 + 1322) = 1770$ basic variables with positive dual values. Using Megiddo's procedure, these would take a minimum of 1770 iterations to correct, worse than the 1565 iterations that our algorithm actually took using the accurate solution as input (there were no artificials left in the basis at optimality). For the standard run, the selection of the initial basis can be expected to be even more difficult.

One approach for dealing with this problem is to introduce a perturbation, as follows. For each structural variable x_j, we replace the bounds on x_j by $l_j - L_j\epsilon$ and $u_j + U_j\epsilon$, respectively, where L_j and U_j are independent uniform $[0,1]$-random variables, and ϵ is some specified tolerance (say 10^{-4}). The resulting perturbed problem, $\widehat{LP}$, is then given to OB1, solved, and the solution passed to CPLEX. Basis recovery is carried out for $\widehat{LP}$ using this solution. The recovered basis, $\mathcal{B}$ is used as the starting point for a simplex solution of the original problem. The expectation is that this $\mathcal{B}$ will be already be optimal for the original problem, and that no additional iterations will be required. The idea of the perturbation (which is essentially that already used in CPLEX for dealing with stalling in the normal simplex method) is not only to make the optimal solution a vertex, but to put the vertices in one-to-one correspondence with the basic feasible solutions. This approach differs from that proposed by Mehrotra [13], who perturbs only the objective, thus producing a unique vertex solution, but not a unique basic optimal solution.

For *stocfor3* this perturbation approach reduces the basis recovery time to 64 seconds (and 324 iterations). For *degen3* it reduces the time to 5.7 seconds (and 104 iterations). However, for several other problems that were tested, perturbation led to significant numerical difficulties for OB1, which could not then solve these problems to acceptable accuracy. This latter behavior has yet to be explained.

7 Concluding Remarks

We have described an effective method for recovering an optimal basis from an optimal primal/dual interior point solution. In tests on the Netlib problem set, this method was able to recover an optimal basis in 5% of the interior point solution for most problems. Although highly primal degenerate problems such as *stocfor3* appear to be intrinsically difficult, we have also indicated that perturbing the problem may effectively eliminate this difficulty. More experimentation is warranted, both in this area and in the selection of a robust starting basis. Nevertheless, it is clear that, in general, basis recovery can be done in reasonable time. For problems where the interior method is indicated. the requirement for an optimal basic solution need not be considered a significant drawback.

Acknowledgements

This research was supported in part by the Center for Research on Parallel Computation through NSF Cooperative Agreement No. CCR-8809615.

References

[1] I. Adler and R. D. C. Montiero. A geometric view of parametric linear programming. Unnumbered technical report, Dept. of Industrial Engineering and Operations Research. University of California. Berkeley CA. 1989.

[2] R. E. Bixby. Implementing the simplex method: The initial basis. Technical Report TR90-32. Dept. of Mathematical Sciences. Rice University. Houston TX. 1990.

[3] D. M. Gay. Electronic mail distribution of linear programming test problems. *COAL Newsletter*, 13:10–13, December 1985.

[4] P. E. Gill. W. Murray, and M. H. Wright. *Numerical Linear Algebra and Optimization*. volume 1. Addison Wesley, Redwood City CA, 1991.

[5] O. Güler and Y. Ye. Convergence behavior of some interior-point algorithms. Working Paper Series 91-4, Dept. of Management Sciences, The University of Iowa, Iowa City IA. 1991.

[6] P. M. J. Harris. Pivot selection methods in the Devex LP code. *Mathematical Programming*, 5:1–28, 1973.

[7] H.-W. Jung, R. E. Marsten, and M. J. Saltzman. Numerical factorization methods for interior point algorithms. *ORSA Journal on Computing*, 4, 1992. forthcoming.

[8] J. W.-H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*. 11(2):141–153. June 1985.

[9] I. J. Lustig. R. E. Marsten. and D. F. Shanno. On implementing Mehrotra's predictor-corrector interior point method for linear programming. Technical Report SOR90-03, Dept. of Civil Engineering and Operations Research. Princeton University, Princeton NJ, 1990.

[10] R. E. Marsten. M. J. Saltzman, D. F. Shanno, G. S. Pierce, and J. F. Ballintijn. Implementation of a dual affine interior point algorithm for linear programming. *ORSA Journal on Computing*, 1(4):287–297, 1989.

[11] N. Megiddo. Pathways to the optimal set in linear programming. In N. Megiddo. editor. *Progress in Mathematical Programming—Interior Point and Related Methods*. pages 131–158. Springer-Verlag, New York NY, 1989.

[12] N. Megiddo. On finding primal- and dual-optimal bases. *ORSA Journal on Computing*. 3(1):63–65. 1991.

[13] S. Mehrotra. On finding a vertex solution using interior point methods. Technical Report 89-22, Dept. of Industrial Engineering and Management Sciences. Northwestern University, Evanston IL. 1990.

[14] U. H. Suhl and L. M. Suhl. Computing sparse *LU* factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, 2(4):325–335. 1990.

[15] P. Wolfe. The composite simplex algorithm. *SIAM Review*, 7(1):42–54, 1965.

Table I: Test problem dimensions and interior point performance.

Name	Before crush			After crush			Prep	Solve			
	m	n	nz	m	n	nz	secs	secs	iter	objval	compl
afiro	27	32	83	25	32	81	0.03	0.07	9	−4.64753140e+02	1e+03
sc50b	50	48	118	48	48	118	0.03	0.14	8	−7.00000000e+01	1e+05
sc50a	50	48	130	49	48	130	0.02	0.14	10	−6.45750770e+01	1e+05
sc105	105	103	280	104	103	280	0.03	0.30	10	−5.22020610e+01	1e+03
kb2	43	41	286	43	41	286	0.03	0.27	15	−1.74990010e+03	1e+02
adlittle	56	97	383	53	97	380	0.05	0.26	12	2.25494960e+05	1e+03
scagr7	129	140	420	95	140	385	0.06	0.40	12	−2.33138980e+06	1e+03
stocfor1	117	111	447	106	111	433	0.06	0.65	19	−4.11319760e+04	1e+04
blend	74	83	491	72	83	489	0.04	0.46	14	−3.08121500e+01	1e+06
sc205	205	203	551	203	202	550	0.10	0.63	11	−5.22020610e+01	1e+02
recipe	91	180	663	75	170	603	0.08	0.36	10	−2.66616000e+02	1e+04
share2b	96	79	694	93	79	691	0.06	0.51	12	−4.15732240e+02	1e+00
vtpbase	198	203	908	53	194	444	0.07	0.36	13	1.29831460e+05	1e+04
lotfi	153	308	1078	134	308	1045	0.11	1.15	16	−2.52647060e+01	1e+00
share1b	117	225	1151	110	225	1142	0.10	1.13	20	−7.65893190e+04	1e+00
boeing2	166	143	1196	125	143	801	0.10	0.96	14	−3.15018730e+02	1e+02
scorpion	388	358	1426	292	358	1312	0.17	1.34	14	1.87812480e+03	1e+01
bore3d	233	315	1429	140	304	1165	0.17	1.14	18	1.37308040e+03	1e+02
scagr25	471	500	1554	347	500	1429	0.21	2.01	16	−1.47534330e+07	1e+01
sctap1	300	480	1692	284	480	1638	0.17	1.75	15	1.41225000e+03	1e+03
capri	271	353	1767	259	367	1816	0.23	3.07	18	2.69001290e+03	1e+02
brandy	220	249	2148	126	247	2084	0.22	2.04	19	1.51850990e+03	1e+02
israel	174	142	2269	163	142	2258	0.42	9.46	23	−8.96644820e+05	1e+00
finnis	497	614	2310	386	607	2082	0.38	4.37	24	1.72791070e+05	1e+00
giffpin	616	1092	2377	590	1092	2341	0.29	3.43	18	6.90223600e+06	1e+03
scsd1	77	760	2388	77	760	2388	0.15	1.15	11	8.66666670e+00	1e+05
etamacro	400	688	2409	334	669	2189	0.34	10.13	29	−7.55715230e+02	1e−01
agg	488	163	2410	327	163	1890	0.29	6.08	23	−3.59917670e+07	1e−01
bandm	305	472	2494	254	467	2397	0.24	2.51	17	−1.58628020e+02	1e+01
e226	223	282	2578	161	281	2472	0.23	2.70	22	−1.87519290e+01	1e+04
scfxm1	330	457	2589	282	457	2516	0.22	2.93	17	1.84167590e+04	1e+03
grow7	140	301	2612	140	301	2612	0.18	1.89	14	−4.77878120e+07	1e+02
standata	360	1183	3139	310	1064	2915	0.28	3.08	15	1.25769950e+03	1e+07
scrs8	490	1169	3182	450	1165	3086	0.33	6.69	27	9.04296950e+02	1e+02
beaconfd	173	262	3375	113	262	3286	0.27	0.98	10	3.35924860e+04	1e+01
shell	536	1775	3556	496	1775	3476	0.33	5.50	21	1.20882530e+09	1e+04
boeing1	351	384	3485	308	384	2822	0.47	5.07	24	−3.35213570e+02	1e−01
standmps	467	1075	3679	409	1064	3563	0.38	5.77	24	1.40601750e+03	1e+07
stair	356	467	3856	356	473	3874	0.55	7.00	14	−2.51266950e+02	1e+02
degen2	444	534	3978	444	534	3978	1.74	8.05	14	−1.43517800e+03	1e+03
agg2	516	302	4284	458	302	4110	0.55	10.40	18	−2.02392520e+07	1e−01
agg3	516	301	4300	458	302	4126	0.54	9.99	17	1.03121160e+07	1e+00
scsd6	147	1350	4316	147	1350	4316	0.22	2.43	12	5.05000000e+01	1e+02
ship04s	402	1458	4352	241	1458	4157	0.41	3.09	15	1.79871470e+06	1e+03
seba	515	1028	4352	449	994	4205	2.06	78.32	19	1.57116000e+04	1e+03
tuff	333	587	4520	284	583	4406	0.54	5.46	19	2.92147770e−01	1e+00
forplan	161	421	4563	133	421	4558	0.28	3.78	21	−6.64218960e+02	1e+00
bnl1	643	1175	5121	564	1172	4976	0.70	13.00	27	1.97762960e+03	1e+01
pilot4	410	1000	5141	397	1088	7190	1.07	21.26	35	−2.58113930e+03	1e−01
scfxm2	660	914	5183	564	914	5037	0.42	6.57	19	3.66602620e+04	1e+00

Table I: Test problem dimensions and interior point performance (cont.).

Name	Before crush			After crush			Prep	Solve			
	m	n	nz	m	n	nz	secs	secs	iter	objval	compl
grow15	300	645	5620	300	645	5620	0.41	4.71	16	−1.06870940e+08	1e+03
perold	625	1376	6018	600	1453	7101	0.92	32.51	36	9.38075520e+03	1e−01
fffff800	524	854	6227	484	854	6170	0.95	15.81	28	5.55679570e+05	1e−01
ship04l	402	2118	6332	317	2118	6101	0.69	4.50	15	1.79332450e+06	1e−01
sctap2	1090	1880	6714	1033	1880	6489	0.71	10.03	20	1.72480710e+03	1e+05
ganges	1309	1681	6912	1137	1681	6740	0.84	16.60	16	−1.09585740e+05	1e+00
ship08s	778	2387	7114	326	2387	6305	0.64	4.01	14	1.92009820e+06	1e+04
sierra	1227	2036	7302	1212	2036	7272	0.65	12.02	18	1.53943620e+07	1e+00
scfxm3	990	1371	7777	846	1371	7558	0.70	10.66	20	5.49012550e+04	1e+02
ship12s	1151	2763	8178	417	2763	7357	0.73	6.04	18	1.48923610e+06	1e+03
grow22	440	946	8252	440	946	8252	0.59	7.05	16	−1.60834340e+08	1e+03
stocfor2	2157	2031	8343	2141	2031	8319	1.14	20.73	22	−3.90244090e+04	1e+04
scsd8	397	2750	8584	397	2750	8584	0.47	4.30	10	9.05000000e+02	1e+05
sctap3	1480	2480	8874	1408	2480	8595	0.91	12.23	17	1.42400000e+03	1e+02
pilotwe	722	2789	9126	705	2863	9601	0.96	32.00	44	−2.72010750e+06	1e−01
maros	846	1443	9614	680	1412	8653	1.05	17.07	31	−5.80637440e+04	1e+00
fit1p	627	1677	9868	627	1677	9868	4.02	446.95	16	9.14637810e+03	1e+03
25fv47	821	1571	10400	780	1571	10360	1.15	33.86	25	5.50184590e+03	1e+00
czprob	929	3523	10669	689	3521	10126	2.31	16.36	35	2.18519670e+06	1e+03
ship08l	778	4283	12802	520	4283	11614	1.29	8.56	16	1.90905520e+06	1e+03
pilotnov	975	2172	13057	870	2115	12514	1.76	51.85	20	−4.49727620e+03	1e+00
nesm	662	2923	13288	646	2914	13256	0.91	31.09	30	1.40760370e+07	1e−01
fit1d	24	1026	13404	24	1026	13404	0.53	6.25	18	−9.14637810e+03	1e+03
bnl2	2324	3489	13999	2113	3483	13690	2.76	161.39	32	1.81123650e+03	1e+00
pilotja	940	1988	14698	831	2017	15354	2.13	119.81	43	−6.11313650e+03	1e+00
ship12l	1164	5427	16170	687	5427	14913	1.56	12.73	18	1.47018790e+06	1e+00
cycle	1903	2857	20720	1528	2799	17293	2.68	68.08	30	−5.22639300e+00	1e−01
80bau3b	2262	9799	21002	2020	9768	20646	4.27	89.32	35	9.87224190e+05	1e−01
degen3	1503	1818	24646	1503	1818	24646	48.91	138.76	18	−9.87294000e+02	1e+01
greenbea	2392	5405	30877	1951	5296	28075	6.83	81.17	41	−7.25552480e+07	1e+00
greenbeb	2392	5405	30877	1947	5293	27746	6.77	65.11	34	−4.30226030e+06	1e−01
d2q06c	2171	5167	32417	2098	5167	32344	4.06	415.54	31	1.22784210e+05	1e−01
woodw	1098	8405	37474	711	8285	32695	2.63	36.72	20	1.30447630e+00	1e+00
pilot	1441	3652	43167	1391	3631	42974	11.90	568.91	29	−5.57489730e+02	1e−01
fit2p	3000	13525	50284	3000	13525	50284	2.22	76.27	18	6.84642930e+04	1e+01
stocfor3	16675	15695	64875	16643	15695	64819	9.53	267.09	35	−3.99767840e+04	1e−01
wood1p	244	2594	70215	171	2594	69422	3.11	25.72	14	−1.44290240e+00	1e+00
pilot87	2030	4883	73152	1991	4859	72967	34.97	2596.14	41	3.01710350e+02	1e−01
fit2d	25	10500	129018	25	10500	129018	5.38	84.85	24	−6.84642930e+04	1e+02
total							190.06	5873.03			

Table II: Basis recovery performance.

	Setup		Solve				Pivot type				Megiddo	
				pivots			simplex		cleanup		est. pivots	
Name	secs	fact	secs	ph. 1	ph. 2	cleanup	exch	push	exch	push	primal	dual
afiro	0.00	2	0.00	0	0	2	0	0	1	1	2	7
sc50b	0.02	1	0.02	0	0	0	0	0	0	0	0	0
sc50a	0.02	2	0.05	0	1	0	1	0	0	0	0	3
sc105	0.05	2	0.02	0	1	0	1	0	0	0	0	11
kb2	0.04	1	0.00	0	0	0	0	0	0	0	0	0
adlittle	0.05	2	0.02	0	9	16	9	0	9	7	25	8
scagr7	0.02	1	0.00	0	2	0	0	2	0	0	2	0
stocfor1	0.04	2	0.03	0	8	0	8	0	0	0	0	7
blend	0.07	2	0.02	0	1	2	1	0	2	0	2	6
sc205	0.08	2	0.05	0	2	0	2	0	0	0	0	11
recipe	0.04	2	0.03	0	3	58	3	0	17	41	59	3
share2b	0.02	2	0.05	0	6	6	6	0	5	1	6	9
vtpbase	0.02	2	0.00	0	0	0	0	0	0	0	0	1
lotfi	0.06	2	0.07	0	7	28	7	0	10	18	30	8
share1b	0.03	1	0.02	0	1	0	0	1	0	0	0	0
boeing2	0.06	2	0.07	0	15	13	15	0	8	5	13	30
scorpion	0.11	2	0.13	0	30	0	30	0	0	0	0	22
bore3d	0.05	2	0.05	0	6	0	6	0	0	0	2	10
scagr25	0.11	2	0.18	0	32	0	30	2	0	0	32	72
sctap1	0.09	2	0.25	0	26	101	26	0	43	58	104	53
capri	0.09	2	0.15	0	12	35	12	0	11	24	41	7
brandy	0.11	2	0.13	1	2	5	3	0	2	3	7	9
israel	0.10	2	0.13	1	10	19	11	0	8	11	30	11
finnis	0.13	2	0.37	0	42	65	39	3	39	26	69	91
giffpin	0.21	2	1.05	0	193	0	193	0	0	0	77	233
scsd1	0.05	2	0.20	0	61	12	61	0	3	9	12	46
etamacro	0.10	2	0.23	0	33	21	32	1	17	4	24	46
agg	0.08	2	0.10	0	21	5	2	19	0	5	31	5
bandm	0.14	2	0.00	0	0	0	0	0	0	0	0	2
e226	0.11	2	0.08	0	5	4	5	0	2	2	4	8
scfxm1	0.12	2	0.15	0	9	12	9	0	4	8	12	25
grow7	0.22	2	0.83	6	48	97	16	38	67	30	151	6
standata	0.13	2	0.08	0	3	12	3	0	0	12	12	212
scrs8	0.22	2	0.18	0	5	16	5	0	16	0	16	111
beaconfd	0.05	2	0.05	0	2	4	2	0	2	2	6	1
shell	0.19	2	0.42	0	55	1	55	0	0	1	8	146
boeing1	0.13	2	0.27	0	22	15	22	0	10	5	18	32
standmps	0.15	2	0.22	0	19	14	19	0	6	8	14	209
stair	0.78	2	0.28	0	0	1	0	0	1	0	1	1
degen2	0.12	2	1.92	0	250	0	250	0	0	0	0	171
agg2	0.13	2	0.50	4	94	33	17	81	12	21	131	14
agg3	0.13	2	0.47	9	65	34	13	61	15	20	107	12
scsd6	0.14	2	0.32	0	44	78	44	0	12	66	80	40
ship04s	0.20	2	0.15	0	11	1	11	0	0	1	1	27
seba	0.17	2	0.08	0	1	0	1	0	0	0	0	9
tuff	0.15	2	0.27	0	24	51	24	0	16	35	51	113
forplan	0.10	2	0.10	0	15	0	15	0	0	0	2	26
bnl1	0.27	2	1.53	41	37	227	77	1	76	151	292	76
pilot4	0.77	3	1.22	37	36	34	33	40	4	30	42	19

Table II: Basis recovery performance (cont.).

	Setup		Solve				Pivot type				Megiddo	
				pivots			simplex		cleanup		est. pivots	
Name	secs	fact	secs	ph. 1	ph. 2	cleanup	exch	push	exch	push	primal	dual
scfxm2	0.25	2	0.35	0	23	23	23	0	8	15	31	41
grow15	0.77	3	2.57	17	5	233	22	0	161	72	254	8
perold	0.99	4	3.33	19	142	37	67	94	19	18	57	29
fffff800	0.17	2	0.47	2	33	51	25	10	19	32	77	89
ship04l	0.26	2	0.40	0	29	1	29	0	1	0	1	45
sctap2	0.31	2	1.45	0	47	230	47	0	34	196	230	249
ganges	0.55	3	2.93	136	3	147	139	0	87	60	286	0
ship08s	0.28	2	0.38	0	30	0	30	0	0	0	0	68
sierra	0.38	2	0.90	0	64	12	62	2	1	11	26	234
scfxm3	0.36	2	0.62	0	21	34	21	0	12	22	34	46
ship12s	0.33	2	0.43	0	37	0	37	0	0	0	0	76
grow22	1.15	3	4.82	20	0	409	20	0	291	118	429	0
stocfor2	0.58	2	2.70	0	176	0	176	0	0	0	0	195
scsd8	0.33	2	1.45	0	47	177	47	0	83	94	205	39
sctap3	0.42	2	2.87	0	92	277	92	0	42	235	277	425
pilotwe	0.87	3	3.68	29	106	0	116	19	0	0	47	90
maros	0.33	2	1.23	3	105	14	108	0	2	12	18	241
fit1p	0.27	1	0.03	0	0	0	0	0	0	0	0	1
25fv47	0.59	2	1.35	3	43	18	46	0	6	12	26	67
czprob	0.45	2	0.78	0	33	60	33	0	17	43	93	34
ship08l	0.52	2	0.77	0	46	0	46	0	0	0	0	91
pilotnov	0.53	3/1	10.35	103	0	1190	103	0	442	953	1293	5
nesm	0.57	4	9.72	87	1503	229	249	1341	96	133	1498	98
fit1d	0.17	1	0.03	0	0	0	0	0	0	0	0	0
bnl2	0.83	2	7.25	0	44	504	43	1	167	337	519	297
pilotja	1.20	4	4.78	55	67	99	115	7	42	57	133	48
ship12l	0.68	2	1.37	0	65	1	65	0	0	1	1	223
cycle	0.55	2	5.85	0	117	410	117	0	157	253	410	570
80bau3b	1.61	2	8.83	10	380	94	386	4	48	46	161	650
degen3	0.68	2	26.13	0	932	2	932	0	1	1	3	628
greenbea	2.17	2/4	13.48	69	375	235	440	4	61	174	289	992
greenbeb	2.59	3/4	12.98	58	301	220	348	11	69	151	324	966
d2q06c	2.00	2	8.03	2	142	31	141	3	13	18	44	253
woodw	1.06	2	3.97	121	120	320	196	45	0	320	401	129
pilot	13.51	4	14.42	3	57	54	48	12	19	35	87	40
fit2p	2.83	3	3.58	4	5	0	9	0	0	0	4	9
stocfor3	5.55	2	255.82	195	2045	0	2240	0	0	0	364	3191
wood1p	0.90	2	3.70	48	183	0	231	0	0	0	0	54
pilot87	45.59	3	44.38	11	57	28	54	8	7	21	41	21
fit2d	2.21	2	1.58	1	6	3	7	0	1	2	6	2
total	101.66		482.30									