Adaptive Use of Iterative Methods in Predictor-Corrector Interior Point Methods for Linear Programming

Weichung Wang [a]

[a] *Department of Mathematics Education, National Tainan Teachers College, Tainan 700, Taiwan*
E-mail: wwang@ipx.ntntc.edu.tw

Dianne P. O'Leary [b,*]

[b] *Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, U.S.A.*
E-mail: oleary@cs.umd.edu

Dedicated to Richard Varga, for making the use of iterative methods a science, without diminishing the artistry.

In this work we devise efficient algorithms for finding the search directions for interior point methods applied to linear programming problems. There are two innovations. The first is the use of updating of preconditioners computed for previous barrier parameters. The second is an adaptive automated procedure for determining whether to use a direct or iterative solver, whether to reinitialize or update the preconditioner, and how many updates to apply. These decisions are based on predictions of the cost of using the different solvers to determine the next search direction, given costs in determining earlier directions. We summarize earlier results using a modified version of the OB1-R code of Lustig, Marsten, and Shanno, and we present results from a predictor-corrector code PCx modified to use adaptive iteration. If a direct method is appropriate for the problem, then our procedure chooses it, but when an iterative procedure is helpful, substantial gains in efficiency can be obtained.

Keywords: Interior point methods, linear programming, iterative methods for linear systems, adaptive algorithms, self-timing algorithms

AMS Subject classification: Primary 65K05, 65F10, 90C05

* This work was supported in part by the National Science Foundation under Grants CCR 95-03126 and CCR 97-32022.

1. Introduction

Interior point algorithms are now widely used to solve linear programming problems

$$\begin{aligned} &minimize\; c^T x \\ &subject\ to\; Ax = b, \\ &\qquad\qquad x \geq 0, \end{aligned} \tag{1}$$

where c, x are real n-vectors, b is a real m-vector, and A is a real $m \times n$ matrix of rank m, with $m \leq n$. These methods typically solve a sequence of logarithmic barrier subproblems with the barrier parameter decreasing to zero. Newton's method is applied to solve the first order optimality conditions corresponding to each of the logarithmic barrier subproblems. The bulk of the work in such algorithms is the determination of a search direction for each step.

Gonzaga [19] and Wright [38] surveyed interior point methods, and many computational issues are addressed by Lustig, Marsten, and Shanno [26] and Andersen, Gondzio, Mészáros, and Xu [1]. Therefore, in this section we focus only on the linear systems arising in interior point methods. For definiteness, we consider the primal-dual formulation of interior point methods, but the linear algebra of primal formulations and dual formulations is similar.

The search direction is usually determined by solving either the reduced KKT (Karush-Kuhn-Tucker) system,

$$\begin{pmatrix} -X^{-1}Z & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} r_d + Ze - \mu X^{-1}e \\ r_p \end{pmatrix}, \tag{2}$$

or the normal equations, formed by eliminating Δx from this system. Defining $r_p = b - Ax$, $r_d = c - A^T y - z$, and $D^2 = Z^{-1}X$, we obtain

$$(AD^2A^T)\Delta y = AD^2(r_d + Ze - \mu x^{-1}e) + r_p. \tag{3}$$

Here z is the vector of dual slack variables, μ is the barrier parameter, and X and Z are diagonal matrices containing x and z (respectively) on their main diagonals. Once Δy is determined from the normal equations, Δx may be easily computed from

$$-(X^{-1}Z)\Delta x + A^T\Delta y = r_d + Ze - \mu X^{-1}e. \tag{4}$$

Comparing the normal equations (3) and the KKT system (2), we observe that the matrix for the normal equations is positive definite and symmetric, has smaller size ($m \times m$), and may be more dense. In contrast, the KKT matrix is symmetric indefinite and usually more sparse.

One nice feature of these problems is that only D and the right hand side of the system change from step to step. Thus, the sparsity structure of the problem remains the same, in contrast to the linear systems arising in the simplex algorithm which differ by exchanges of columns of A. Some interior point algorithms (e.g., OB1-R) solve one linear system with each matrix, while others (e.g., PCx) solve multiple systems.

The roots of interior point algorithms date back to the algorithms of Fiacco and McCormick [10], but ever since interior point algorithms first gained prominence in 1984 [20], researchers have given attention to speeding up the step time through efficient solution of the linear system. Direct methods that rely on sparse matrix factorizations have been the most popular approaches (e.g., [25], [33]), although iterative methods for solving linear systems have also received a fair amount of attention.

Karmarkar and Ramakrishnan reported computational results of Karmarkar's dual projection algorithm using a preconditioned conjugate gradient solver [21]. An incomplete Cholesky factorization of the matrix AD^2A^T was computed for one interior point step and then used as a preconditioner over several subsequent steps. In their experiments, Cholesky factorization was performed on average every 2 to 3 steps. Mehrotra used preconditioned conjugate gradients to solve the normal equations in a dual affine scaling interior point algorithm [27]. He addressed issues such as the stopping criterion and the stability of the implementation. At each interior point step, an incomplete Cholesky factor was computed and used as the preconditioner. Carpenter and Shanno used a diagonal preconditioner for a conjugate gradient solver for the normal equations in an interior point method for quadratic and linear programs [3]. They also considered recomputing the preconditioner every other step. Portugal, Resende, Veiga, and Júdice introduced a truncated primal-infeasible dual-feasible interior point method, focusing on network flow problems [32]. The preconditioned conjugate gradient algorithm was used to solve the normal equations. They initially used the diagonal of the matrix AD^2A^T as a preconditioner and replaced it by spanning tree preconditioners in later steps. Mehrotra and Wang [28] used an incomplete Cholesky factor of AD^2A^T as a preconditioner for conjugate gradients in a dual interior point method for network flow problems. Gill, Murray, Saunders, Tomlin, and Wright established the equivalence between Karmarkar's projected method and their projected Newton barrier method [15]. They used LSQR [31], preconditioned by an approximation to AD^2A^T, to find the search directions. Goldfarb and Mehrotra developed a relaxed version of Karmarkar's method that allows inexact projection [17]. They applied CGLS [31] to determine the search direction. Nash and Sofer investigated the choice of a preconditioner in the positive definite system Z^TGZ where Z is rectangular and G is general symmetric [29].

Chin and Vannelli [5] solved a reduced KKT system using the preconditioned conjugate gradient algorithm and Bi-CGSTAB with incomplete factorization. In a different paper [4] they used an incomplete factorization as a preconditioner for the normal equations (3). Freund and Jarre [11] employed a symmetric variant of the quasi-minimal residual (QMR) method to solve the KKT systems. They suggested using indefinite SSOR preconditioners to accelerate the convergence.

Despite all of this work, the use of iterative methods has so far produced limited success. The obstacles to the use of these methods are considerable.

- Over the course of the interior point steps, the requirements on accuracy change greatly; approximate solutions can be allowed early in the steps but can cause the algorithm to fail later when the iterates are near the boundary.
- The matrix D changes quite rapidly and becomes highly ill-conditioned in the final steps.

For these reasons, it is difficult to find a preconditioning strategy that produces good performance of iterative methods over the entire course of the interior point computation.

In this paper we develop an adaptive algorithm that changes strategy over the course of the interior point algorithm. It determines dynamically whether the preconditioner should be held constant, updated, or recomputed, and it switches to a direct method when it predicts that an iterative method will be too expensive. In our experiments, we use a preconditioned conjugate gradient iteration on the linear system involving the matrix ADA^T, but our ideas could be extended to iterations involving the KKT formulation as well.

This idea of choosing among various numeric algorithms depending on the timing performance or timing prediction of algorithmic components for a particular problem on a particular machine architecture was summarized in a 1995 report by O'Leary and Wang [36] and elaborated by Wang in his 1996 thesis [35]. This idea has proved quite useful in other numeric algorithms, such as a 1997 algorithm of Frigo and Johnson for computing Fourier transforms [12] and a 1998 proposal by Whaley and Dongarra for linear algebra computations [37].

In the next section, we discuss the characteristics of direct and iterative methods and present our preconditioner. Section 3 focuses on our algorithm for the adaptive choice of direct vs. iterative methods and the adaptive choice of a preconditioner. Numerical results are presented in § 4. Final comments are made in § 5.

2. The linear system solvers

The most expensive part of an interior point algorithm is determining the search direction by solving one or more linear systems. Either direct or iterative methods may be used for these systems. In this section, we focus on the solution of the normal equations (3). This discussion sets the goals to be accomplished in designing an efficient algorithm.

We will assume that the columns of A have been permuted using standard techniques in order to improve sparsity in the Cholesky factor of AD^2A^T (e.g., [9], [24]).

2.1. Direct solvers: Cholesky factorization

Most existing linear programming interior point methods solve the normal equations by direct methods. The implementations OB1-R of Lustig, Marsten, and Shanno [25] and PCx of Czyzyk, Mehrotra, and Wright [7] are representative of these methods, and the iterative methods will be compared with these implementations.

To solve equation (3), the OB1-R implementation computes a sparse Cholesky factorization of the matrix $K = AD^2A^T$ as LPL^T, where L is a unit lower triangular matrix and P is a diagonal matrix. Forward and backward substitution are then applied to compute the search direction Δy_k. The OB1-R algorithm then checks whether $A\,\Delta x_k$ is close enough to the artificial variables $(b - Ax_k)$. If not, iterative refinement using the factored matrix LPL^T is employed repeatedly until the one-norm of the difference is sufficiently small. To deal with the dense columns in A, the OB1-R algorithm adopts the method suggested by Choi, Monma, and Shanno [6].

The PCx implementation uses a similar strategy for the solution of linear systems, using the Ng-Peyton [30] sparse matrix code, with modification by replacing small pivots by a very large number, and again dealing with dense columns separately. The algorithm also performs iterative refinement using the conjugate gradient algorithm with the factorization as a preconditioner.

There are three main disadvantages to direct methods. First, the iterative refinement used in the OB1-R code may fail if the matrix $K = AD^2A^T$ is very ill-conditioned because the factorization may not be accurate enough to produce an iteration matrix with spectral radius less than one. Such a situation can occur when the primal and dual variables are near to the optimal solution, since then the matrix D is quite ill-conditioned. The iteration can also be affected by ill-conditioning in A.

Another potential problem of direct methods is fill-in. Although the dense columns of A can be treated separately, the remaining Cholesky factor may still be rather dense. This might be caused by difficulty in detecting "dense" columns or by the nature of the problem. For example, network problems solved by linear programming may lead to a Cholesky factor that is much more dense than AD^2A^T even though A has no dense columns.

Lastly, the direct algorithms must form and factor the matrix $K = AD^2A^T$ each time μ is changed. This procedure may be expensive in time, especially when the problem size is large. If $m \ll n$, the resulting matrix K may be small and easy to factor, but forming it can still be costly.

2.2. Iterative solvers: preconditioned conjugate gradients

A variety of iterative methods can be used to solve the normal equations or the KKT system. For definiteness, we focus on the preconditioned conjugate gradient method for solving equation (3). In this method, we compute a sequence of approximate solutions that converge to the true solution. The work during each iteration involves one product of K with a vector, one solution of a linear system involving the preconditioner, and several vector operations. More details about the method can be found in [18].

The storage requirement for the preconditioned conjugate gradient method is quite low, amounting to a few vectors of length m. Although a matrix-vector multiplication $Kv = (AD^2A^T)v$ is required at each iteration, we may compute Kv as $(A(D^2(A^Tv)))$ and thus need only to store the nonzeros of A and the diagonal of D rather than the matrix AD^2A^T, which can be quite dense. The preconditioner should also be chosen to conserve storage.

Since accuracy requirements for the search direction in the beginning phase of the interior point algorithm are quite low, only a few conjugate gradient iterations are required. As the primal and dual variables approach the optimal solution, the convergence tolerance must be tightened and more iterations are needed.

The crucial issue in the preconditioned conjugate gradient algorithm is to find a preconditioner for each step of the interior point method. A good preconditioner may dramatically accelerate the convergence rate and gain great computational savings. We consider some strategies for choosing the preconditioners in the next subsection.

2.3. The preconditioner

Convergence of the conjugate gradient iteration will be rapid if the preconditioned matrix has either a small condition number or great clustering of eigenvalues [18, Chap. 10]. We discuss our strategy for preconditioning.

The basic preconditioner is the Cholesky factorization of one of the matrices that has been generated in the course of the interior point method. PCx always uses the sparse piece of the current matrix, but this requires frequent factorizations.

An alternative to computing a new Cholesky factorization on every interior point step is to reuse the preconditioner that was computed for one value of the barrier parameter μ in order to solve systems for several successive values of μ [3] [21]. This reduces the computational work in forming the factorization.

An incomplete Cholesky factorization, originally proposed by Varga [34], could be used in place of the Cholesky if density of the matrix factors is too great, but we do not pursue that idea in our implementations.

Rather than keeping the preconditioner fixed when μ changes, though, we can update it by a small-rank change, since the normal equations matrix is a continuous function of μ. Let $\hat{D}$ be the current diagonal matrix and D be the one for which we have a factorization $AD^2A^T = LPL^T$. Define $\Delta D = \hat{D}^2 - D^2$

and let a_i be the i-th column of matrix A. Since

$$A\hat{D}^2A^T = AD^2A^T + A\,\Delta DA^T \;=\; LPL^T + \sum_{i=1}^{n} \Delta d_{ii} a_i a_i^T, \tag{5}$$

we may obtain an improved preconditioner $\hat{L}\hat{P}\hat{L}^T$ by applying a rank-α update to LPL^T, where $\alpha \leq n$. This update may be computed as in [2] and [8]. We choose α large enough to include most of the large magnitude terms in the summation. Then we have factored a matrix that differs from $A\hat{D}^2A^T$ by a matrix of rank $n-\alpha$. This difference matrix can be expressed as a matrix of small norm plus one of small rank, and we can hope for rapid convergence of the conjugate gradient iteration.

We now turn our attention to criteria for deciding when to keep or update or reinitialize the current preconditioner and how many iterations to perform.

3. The adaptive algorithm

Our interior point algorithm chooses the initial variables, the step lengths, the barrier parameter μ, and convergence criteria following standard strategies [7,25]. Each step requires the solution of one or more linear systems, and that is where the bulk of the computational work lies. The difference between our algorithm and standard ones is that for each step of the interior point method (each distinct value of μ), we choose an efficient linear equation solver adaptively.

We need to specify when to use an iterative method, when to refactor the matrix, how many updates to use in the preconditioner, and how to terminate the iteration.

3.1. When to use the iterative method

In the first step of the algorithm, the normal equations (3) are solved directly by factoring $K = AD^2A^T = LPL^T$. Starting from the second step, the algorithm uses preconditioned conjugate gradients. The preconditioner for each step is determined by factoring the current matrix K or by updating the current preconditioner. This *"factor-update cycle"* will be continued up to the *"end-game,"* entered when the relative duality gap is small enough. In the end-game, the iterates are close to the optimal solution and accuracy requirements are high. The elements in matrix D vary significantly and make the matrix $K = AD^2A^T$ very ill-conditioned. The Cholesky factorization of K may not generate a good preconditioner, even if stable methods such as [14] are used. For all of these reasons, a direct method is used to determine the final search directions.

We also switch to a direct method when OB1-R computes a Cholesky factorization with a zero on the diagonal. This contingency could be avoided by using a modified Cholesky factor; see, for example, [16, Chap. 4].

For interior point methods like PCx that use the predictor-corrector strategy [?,?], the "factor-update cycle" is modified when the number of different μ values between refactorization drops to 3 or fewer. At that point (the middle stage), we begin to force a refactorization at least every 3 steps. This continues until the relative gap drops below a user-defined tolerance (1.0×10^{-8} in the current implementation) at which time a refactorization is performed at least every other step (the late stage), and then the algorithm proceeds with the end-game as above.

While the adaptive algorithm monitors the cost of the iterative method, it separates out problems that are not well suited to iterative methods. If twice in a row the updated preconditioner is inefficient in the step after the preconditioner is reinitialized, then the algorithm will use only the direct method from then on. An example of such a situation is illustrated in Figure 5.

In summary, our algorithm uses direct methods for linear systems in the first step, in the final (end-game) steps, periodically in the middle and late stages of predictor-corrector methods, and at other times in which the iterative method is estimated to be more expensive than the direct method.

3.2. Deciding whether to refactor or to update the preconditioner

We make decisions regarding refactorization or update of the preconditioner based on the actual cost incurred in determining previous search directions, as measured in seconds by a system timing program:

`drct_cost`	=	the cost of factoring and solving the system directly;
`updt_cost`	=	the cost of each rank-one update;
`pcgi_cost`	=	the cost of each conjugate gradient iteration.

(For simplicity, we neglect the fact that updates and downdates have slightly different costs.) We initialize each of these estimates to zero, but after the first few steps of the interior point method, we have accurate estimates of each. In order to reduce the effects of variability from the timer output, though, we suggest that these estimates continue to be updated over many steps.

The decision to update the current preconditioner or refactor the matrix AD^2A^T to obtain a new preconditioner is based on the approximate cost of the preceding iteration, including the cost of any updates that were made to the preconditioner. This cost is

$$\texttt{prev\_cost} = (\texttt{updt\_cost} \times \texttt{updt\_nmbr}) + (\texttt{pcgi\_cost} \times \texttt{pcgi\_nmbr}) + (overhead),$$

where `updt_nmbr` is the number of updates that were performed and `pcgi_nmbr` is the number of pcg iterations. The overhead includes operations such as initializing the solution to zeros, computing the norm of the right-hand side, deciding on the number of rank-one updates, etc.

- If the cost of determining the previous search direction was high, we reinitialize the preconditioner by factoring the current matrix $K = AD^2A^T$. We take this action when the cost of the previous iteration exceeds 80% of the cost of direct solution:

$$\texttt{prev\_cost} > .8 \times \texttt{drct\_cost}.$$

- If the cost of the previous iteration was not that high, then we base our decision on a *prediction* of the cost of the current iteration, refactoring if the predicted cost is greater than the cost of the direct method.
  Our prediction method is simple and requires only a few arithmetic operations. We fit a straight line to the number of iterations required to determine two preceding search directions. We choose the previous number, and the latest other one that gives a line with positive slope, and use this line to predict the number of iterations, `predi_nmbr`, required to determine the current search direction. If the solver refactored on the previous step, or if we cannot obtain a positive slope with data since the last refactorization, then our predicted number of iterations is one more than the number taken last time, `predi_nmbr` $=$ `pcgi_nmbr`$+1$.
  Given this predicted number of iterations, our predicted cost for computing the search direction, neglecting overhead, is

$$\texttt{pred\_cost} = (\texttt{updt\_cost} \times \texttt{updt\_nmbr}) + (\texttt{pcgi\_cost} \times \texttt{predi\_nmbr}).$$

  If this cost is less than `drct_cost`, then the preconditioner is obtained by updating the previous one. Otherwise it is obtained by factoring $K = AD^2A^T$.

3.3. The adaptive updating strategy

We adopt the strategy discussed in § 2.3: we update the Cholesky factors using the `updt_nbmr`$= \alpha$ "largest" outer product matrices as determined by $|\Delta d_{ii}|$. (We could have used $|\Delta d_{ii}| \|a_i\|^2$ instead.)

We change the number of Cholesky updates adaptively over the course of the algorithm in order to improve efficiency. The number is increased if the previous search direction took many iterations, and decreased if it took a very small number.

Two parameters `sml` $<$ `lrg` are initially set to 20 and 30 respectively. The parameter `sml` denotes a number of conjugate gradient iterations that takes time much less than `drct_cost`, while `lrg` denotes a number that requires a more substantial fraction of `drct_cost`. After timing data is available, we set

$$\texttt{lrg} = 0.15 \times \frac{\texttt{drct\_cost}}{\texttt{pcgi\_cost}}; \qquad \texttt{sml} = 0.12 \times \frac{\texttt{drct\_cost}}{\texttt{pcgi\_cost}}.$$

To decide the number of rank-one updates, `updt_nmbr`, to be performed, let `pcgi_slope` be the slope of the line connecting last two `pcgi_nmbr`s.

$$\text{The } \texttt{updt\_nmbr} \text{ is } \begin{cases} \text{increased}, & \text{if } \texttt{lrg} \leq \texttt{pcgi\_nmbr} \text{ and } \texttt{pcgi\_slope} > 0, \\ \text{decreased}, & \text{if } \texttt{pcgi\_nmbr} \leq \texttt{sml} \text{ and } \texttt{pcgi\_slope} < 0, \\ \text{unchanged}, & \text{otherwise.} \end{cases}$$

Increases or decreases in `updt_nmbr` are proportional to the `pcgi_slope`:

$$\text{(to increase)} \quad \texttt{updt\_nmbr} = \texttt{updt\_nmbr} \times \max(1.2,\ \frac{\texttt{pcgi\_slope}}{8.0}),$$

$$\text{(to decrease)} \quad \texttt{updt\_nmbr} = (\texttt{updt\_nmbr} \times 0.9) + 1\,.$$

Note that the sparsity of the Cholesky factors remains the same, no matter how many updates are used.

3.4. Terminating the PCG iteration

After computing the preconditioner, we solve the normal equations using the preconditioned conjugate gradient method. We start from an initial guess of zero, and iterate until the computed residual norm is less than a parameter ε_{pcg} times the norm of the right-hand side. We choose the parameter ε_{pcg} adaptively: for OB1-R,

$$\varepsilon_{pcg} = \begin{cases} 10^{-8}, & \text{if } \texttt{relgap} > 10^{-2}; \\ 10^{-8} \times (\texttt{relgap})^{\frac{1}{2}}, & \text{otherwise,} \end{cases}$$

where `relgap` is the relative duality gap for the previous value of μ. This is similar to the stopping criterion in [28]. For PCx, we use

$$\varepsilon_{pcg} = \begin{cases} 5.0 \times 10^{-3}, & \text{for the beginning stage,} \\ \min(\texttt{relgap}\times 10^{3},\ 1.0 \times 10^{-3}), & \text{for the middle stage,} \\ \min(\texttt{relgap}\times 10^{4},\ 1.0 \times 10^{-4}), & \text{for the late stage,} \end{cases}$$

If the preconditioned conjugate gradient iteration number exceeds the maximum number of iterations allowed, then the current preconditioner is abandoned and a new preconditioner is determined by Cholesky factorization. If this happens twice, the iterative method is not suitable and we switch to a direct method. Unfortunately, the preconditioned conjugate gradient iteration might be stopped just before convergence, thereby making the refactoring wasteful, but we consider such a safeguard bounding the number of iterations to be important.

The maximum number of iterations is set to the number that produces a cost of 1.2 times the cost of a direct method:

$$\texttt{max\_pcg\_itn} = 1.2 \times \frac{\texttt{drct\_cost}}{\texttt{pcgi\_cost}}.$$

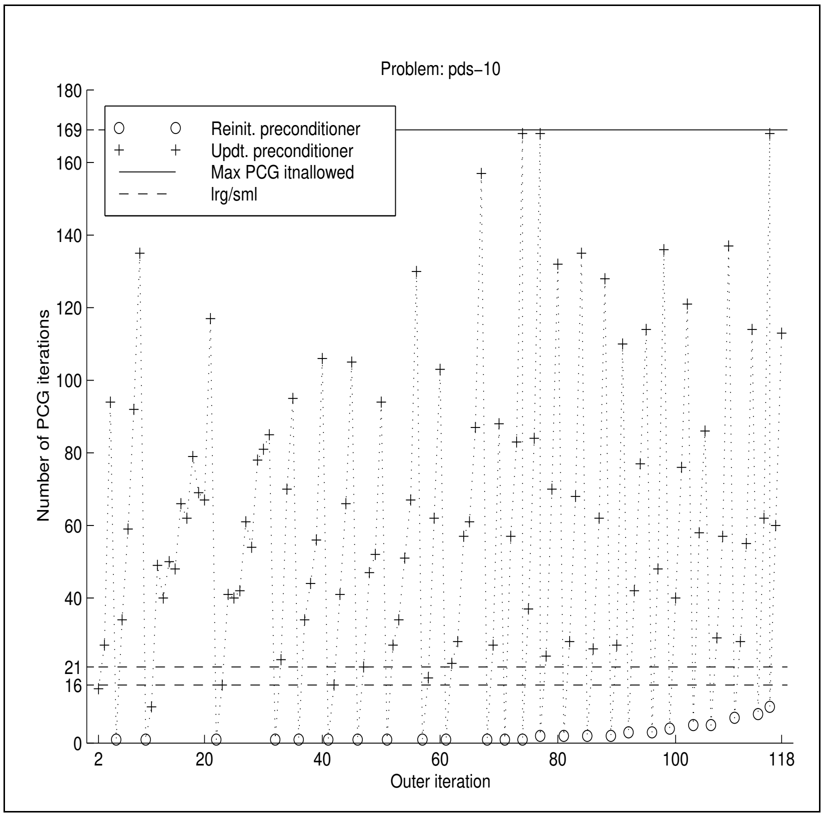

Figure 1. Number of PCG iterations for the adaptive algorithm

To summarize, our algorithm solves the normal equations directly to determine the first search direction, uses a preconditioned conjugate gradient method starting from the second search direction, and switches back to the direct method for the final search directions. The preconditioned conjugate gradient solver solves the normal equations by first choosing and computing a preconditioner using an adaptive strategy to decide whether to refactor the matrix and the rank of the update performed. The algorithm automatically sets all parameters expected to influence performance, based on actual time performance of the components of the algorithm.

4. Numerical results

4.1. OB1-R

We modified the code OB1-R to adaptively choose the linear system solver, and we performed numerical experiments comparing the results of this modified version of OB1-R to the standard OB1-R code, dated December 1989.

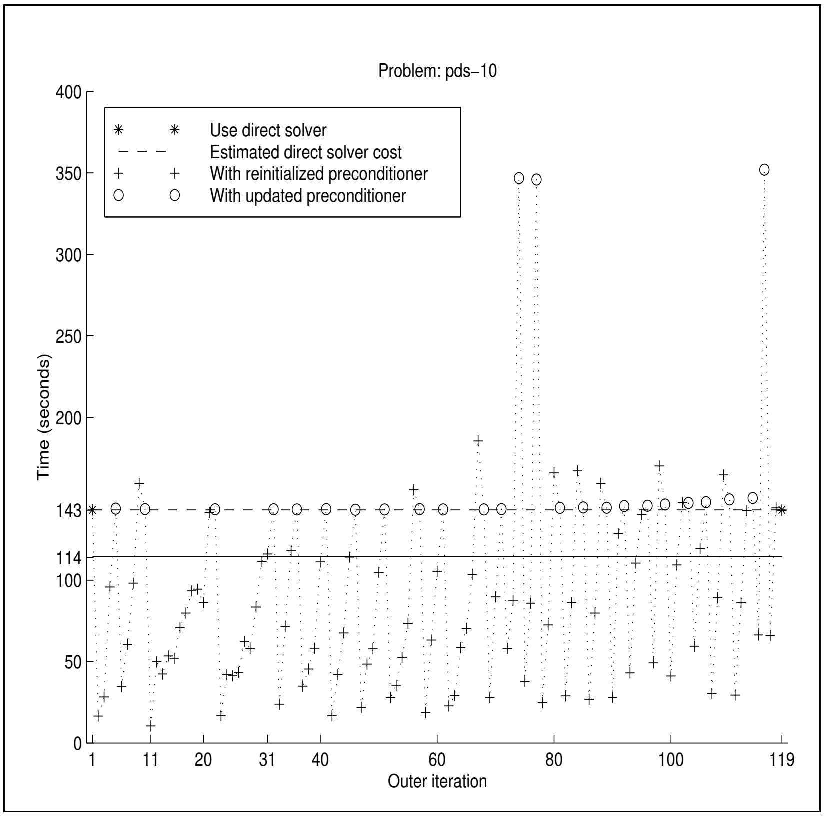

Figure 2. Timing performance for the adaptive algorithm

Both OB1-R and the adaptive algorithm are coded in FORTRAN and use double precision arithmetic. Our experiments were performed on a SUN SPARC-station 20 with 64 megabytes of main memory, running SunOS Release 4.1.3. The FORTRAN optimization level was set to -O3. We report CPU time in seconds, omitting the time taken by the preprocessor HPREP since it is the same for both codes.

Before comparing the two codes, we illustrate the behavior of the adaptive algorithm on a large problem, `pds-10` (with artificial variables) whose problem characteristics are given in Table 1. Figure 1 shows the number of iterations needed by the preconditioned conjugate gradient method for the μ values chosen by OB1-R. Conjugate gradients are used for μ_2 through μ_{118}, and then the algorithm chooses to switch to direct solution because it detects a zero on the diagonal of the preconditioner. The horizontal line at 169 marks the maximum number of conjugate gradient iterations allowed (i.e. `max_pcg_itn`). The two dashed lines at 21 and 16 indicate `lrg` and `sml`, respectively. The Cholesky factorization is recomputed 25 times, marked by circles in the figure. This is a savings of 92 factorizations compared to the OB1-R algorithm. In between refactorizations, the number of conjugate gradient iterations generally grows, more quickly for later values of μ than for earlier ones.

Figure 2 displays the time taken by each of these linear system solves. The dashed line is `drct_cost`, the estimated direct solver cost based on its performance for the first value of μ. The solid line marks 0.8 times `drct_cost`.

We highlight the following observations from the figures.

- The adaptive algorithm produces significant savings in the beginning stage, especially from the 11th to the 31st value of μ.
- The frequency of reinitializing the preconditioner grows as μ is decreased.
- The preconditioners obtained from refactoring the matrix AD^2A^T are unsuitable in the late stage.
- The adaptive algorithm succeeds in keeping the cost near or better than the direct cost on all iterations but three. On those, the predicted number of iterations is too low.

We now summarize computational results on various types of linear programming problems. If the total time for solution is small (i.e., 5 minutes or less), then the performance of the two algorithms is similar. On more costly problems, the adaptive method is faster: e.g., 9% faster on `pilot87`, 16% faster on `dfl001`, and 28% faster on `maros-r7` from the NETLIB collection.

More complete results can be found in [36].

4.2. PCx

We modified the code PCx to adaptively choose the linear system solver, and we performed numerical experiments comparing the results of this modified version of PCx to PCx version 1.1, dated November 1997.

Both PCx and the adaptive algorithm are coded by FORTRAN and C language and use double precision arithmetic. Our experiments were performed on a HP C1100/9000 workstation with 128 megabytes of main memory, running HP-UX B.10.20 operating system. The FORTRAN and C optimization level were set to `-O`. We report CPU time in seconds, including the time taken by the preprocessor which is the same for both codes.

We report computational results on various types of linear programming problems chosen from NETLIB, NETLIB's Kennington problems, and some network problems. We omit data for problems taking fewer than 10 seconds, since direct methods are quite suitable for these.

Table 1 summarizes the problem characteristics. The numbers of rows and columns indicated in the table refer to the output from the PCx preprocessor and may be different from the data in [13]. The tabulated number of nonzero

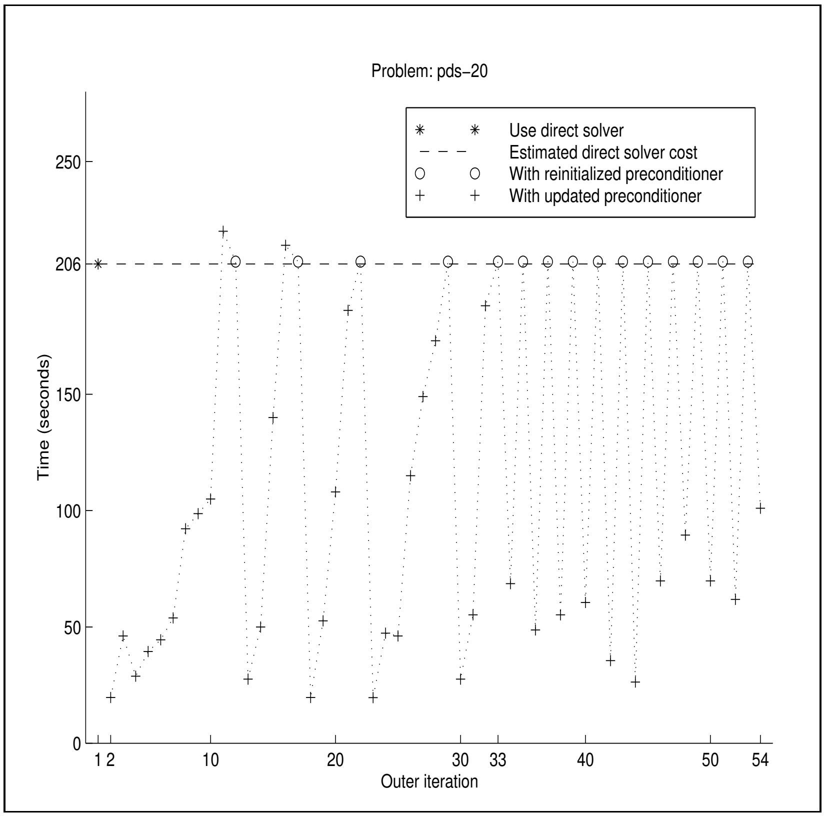

Figure 3. Timing performance for the adaptive algorithm on problem `pds-20`.

elements of the Cholesky factor L include the diagonal part of L. The density of L is computed as

$$\frac{2 \times (\text{Number of nonzeroes of } L) - \widehat{m}}{\widehat{m} \times \widehat{m}},$$

where $\widehat{m}$ is the number of rows of A after presolving.

Minimum cost flow network problems may be solved using linear programming algorithms (although it is generally more efficient to use a network algorithm like [22]). We test our algorithm on this class of problems because the matrix AA^T and its resulting Cholesky factor tend to be much more dense than the original coefficient matrix A, even if there is no dense column in A. Forming and factoring the matrix AD^2A^T is thus quite expensive. We generated minimum cost flow network problems using NETGEN, developed by Klingman, Napier, and Stutz [23].

Before comparing the two codes, we illustrate the typical behavior of the adaptive algorithm using three examples.

	LP size		Cholesky factor L	
Problem	Rows	Columns	Nonzeros	Density
NETLIB:				
d2q06c	2132	5728	137349	0.060
degen3	1503	2604	120906	0.106
dfl001	5984	12143	1638085	0.091
fit2d	25	10524	324	0.997
greenbea	1933	4164	49055	0.026
maros-r7	2152	7440	534188	0.230
pilot	1368	4543	200812	0.214
pilot87	1971	6373	425654	0.219
stoch3	15362	22228	177936	0.001
Kennington:				
cre-b	5336	36382	248629	0.017
cre-d	4102	28601	212094	0.025
ken-11	10085	16740	102906	0.002
ken-13	22534	36561	298417	0.001
ken-18	78862	128434	1928863	0.002
osa-07	1081	25030	28276	0.047
osa-14	2300	54760	60795	0.023
osa-30	4313	104337	115081	0.012
osa-60	10243	243209	265909	0.005
pds-06	91556	28472	589339	0.014
pds-10	15648	48780	1687660	0.014
pds-20	32287	106180	7089645	0.014
Network:				
net0108	1000	8000	207560	0.414
net0116	1000	16000	280678	0.560
net0408	4000	8000	556366	0.069
net0416	4000	16000	1766394	0.221
net0816	8000	16000	2201390	0.069
net0832	8000	32000	7000874	0.219
net0864	8000	64000	12247346	0.383
net1632	16000	32000	8653616	0.068

Table 1
Statistics of the test problems.

Figure 3 shows the time taken by linear system solves on problem `pds-20`. The time for computing predictor and corrector are summed. The dashed line is `drct_cost`, the estimated direct solver cost based on its performance for the first value of μ. Preconditioned conjugate gradients are used for μ_2 through μ_{54}. The algorithm switches to the late stage at μ_{33} and then to the ending stage at μ_{55} because it detects a relative duality gap that is smaller than the parameter `ending_tol`. The Cholesky factorization is recomputed 15 times, marked by cir-

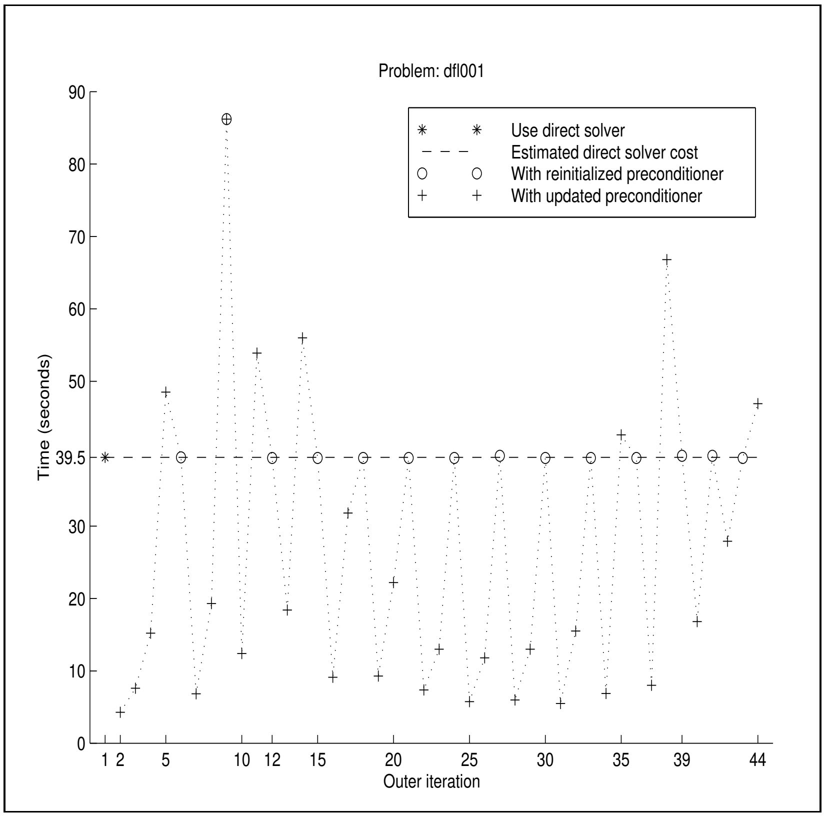

Figure 4. Timing performance for the adaptive algorithm on problem `dfl001`.

cles in the figure. This is a savings of 38 factorizations compared to the PCx algorithm. In between refactorizations, the number of conjugate gradient iterations generally grows, more quickly for late values of μ than for earlier ones. We highlight the following observations from the figure, similar to the observations for the OB1-R code.

- The adaptive algorithm produces significant savings in the beginning stage, especially from the 2nd to the 10th value of μ.
- The frequency of reinitialization of the preconditioner grows as μ is decreased.
- The adaptive algorithm succeeds in keeping the cost at or better than the direct cost on all iterations but two. On those, the timings are close to `drct_cost`.

Figure 4 shows that Problem `dfl001` has a long middle stage. While computing the predictor at μ_9, the adaptive algorithm does not converge to the predefined tolerance within the maximum number of iterations allowed. The algorithm thus decides to carry out refactorization, resulting in high cost. The algorithm switches to the middle stage at μ_{12} and remains there through μ_{38}. The adaptive algorithm keeps the cost close to or better than the direct cost on

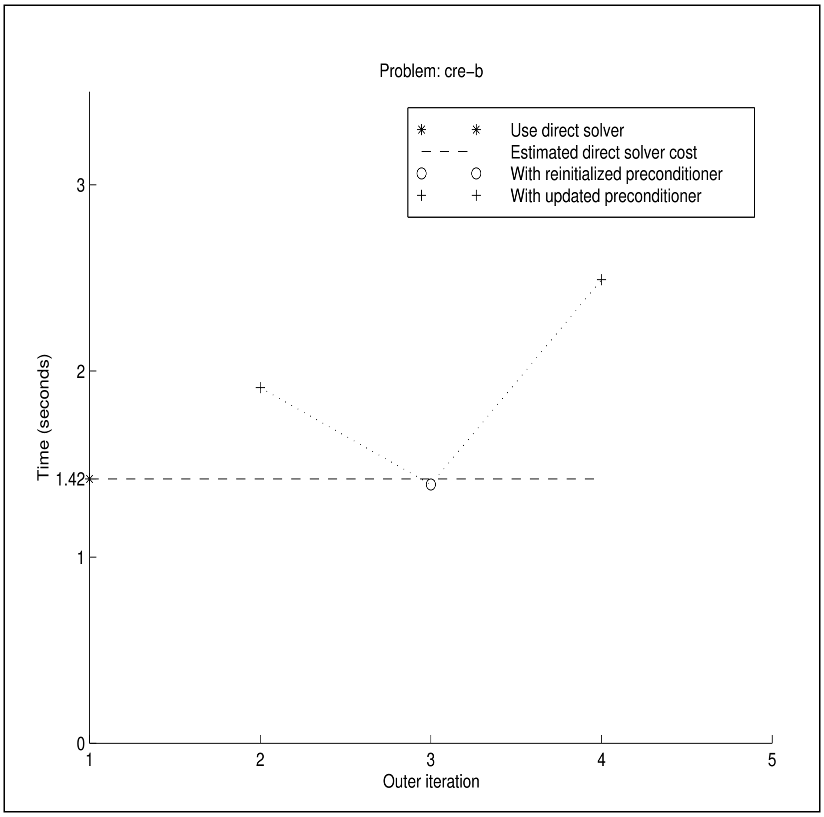

Figure 5. Timing performance for the adaptive algorithm on problem `cre-b`.

all iterations in the middle stage except for μ_{14} and μ_{38}. The adaptive algorithm switches to late and ending stages at μ_{39} and μ_{45} respectively.

Problem `cre-b` is not suitable for the iterative method. The adaptive algorithm discovers this at μ_5 and switches to the direct method (Figure 5). This happens because twice in a row the updated preconditioner is inefficient right after the preconditioner is reinitialized. Such behavior occurs in problems like `cre-d`, `ken-11`, `ken-13`, and `osa-07`. It is vitally important that the algorithm can make this decision automatically.

Table 2 shows the computational results on the three problem sets, comparing the number of μ values needed by the interior point method, the relative duality gap in the final answer, and the CPU time required by PCx and the adaptive algorithm. The last column is the difference between the PCx and the adaptive times. A positive difference means the adaptive algorithm is faster.

We summarize the following observations from the results in table 2.

- Both PCx and the adaptive algorithm converge to solutions satisfying the optimality criteria defined in PCx except on the problem `greenbea`, which is well-known to be difficult for interior point methods [33]. The optimal criteria

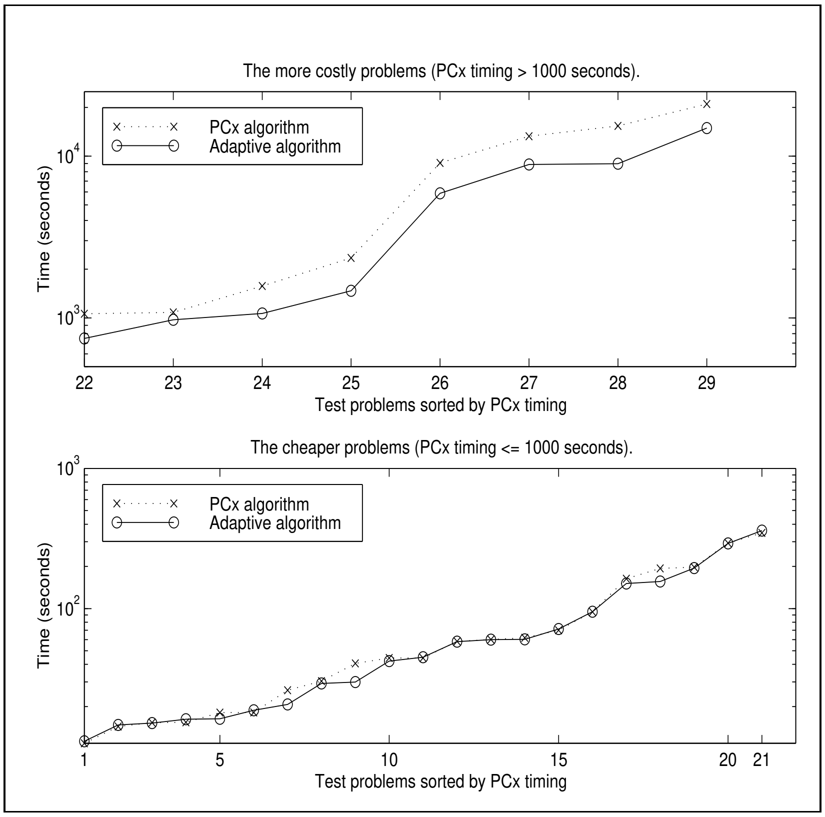

Figure 6. Timing performance comparison for all test problems, sorted by the time taken by the PCx algorithm.

include small duality gap, primal feasibility, and dual feasibility.

- The algorithms take a similar number of μ values and achieve similar duality gaps in most of the test problems. On some problems like `pds-10`, `pds-20`, `degen3`, and `maros-r7`, however, the adaptive algorithm takes 1 or 3 additional steps and achieves duality gaps several orders of magnitude smaller.
- In `dfl001`, the adaptive algorithm achieves the optimal criteria in 10 fewer steps, obtains a duality gap 1 order of magnitude smaller, and is faster than PCx by 880 seconds.
- If the total time for solution is small (i.e., 7 minutes or less), then the performance of the two algorithms is similar. On more costly problems such as `dfl001`, `pilot87`, `net0832`, and `pds-20`, the adaptive method is faster. Figure 6 compares the timing of the problems.

Problem	IPM ite. PCx	IPM ite. Adap	Rel. dul. gap PCx	Rel. dul. gap Adap	Time PCx	Time Adap	Time Diff.
NETLIB:							
d2q06c	29	29	2.53e-07	2.55e-07	18.14	18.79	-0.65
degen3	16	19	1.18e-08	8.49e-12	18.11	20.70	-2.59
dfl001	58	48	5.63e-08	2.58e-09	2350.12	1470.08	880.04
fit2d	23	23	4.94e-07	3.67e-07	15.36	14.77	0.59
greenbea	48	50	1.90e-10	2.13e-09	10.93	11.31	-0.38
maros-r7	18	19	1.31e-08	2.97e-14	60.23	60.08	0.15
pilot	36	36	2.88e-07	2.91e-07	44.40	45.00	-0.60
pilot87	34	35	1.96e-07	2.31e-08	163.25	155.89	7.36
stoch3	31	31	6.13e-08	8.20e-08	26.16	29.90	-3.74
Kennington:							
cre-b	40	40	9.69e-07	8.40e-07	70.04	71.49	-1.45
cre-d	40	40	1.21e-06	1.46e-06	58.43	60.31	-1.88
ken-11	21	21	6.44e-08	6.45e-08	14.42	15.25	-0.83
ken-13	26	26	5.42e-07	5.20e-07	44.34	16.34	28.00
ken-18	30	29	1.84e-06	5.75e-06	293.36	291.66	1.70
osa-07	25	25	2.55e-07	2.11e-07	15.44	16.24	-0.80
osa-14	27	27	1.75e-08	1.68e-08	40.62	42.24	-1.62
osa-30	27	26	2.22e-08	5.72e-08	94.70	94.91	-0.21
osa-60	30	31	3.60e-07	1.95e-07	346.73	359.73	-13.00
pds-06	37	36	8.51e-07	1.75e-06	197.41	194.37	3.04
pds-10	41	44	4.59e-06	8.34e-08	1081.83	973.82	108.01
pds-20	55	58	7.19e-06	9.82e-08	13268.11	8969.40	4298.71
Network:							
net0108	16	16	7.68e-08	7.68e-08	30.37	29.23	1.14
net0116	18	19	1.58e-07	2.23e-09	61.90	58.04	3.86
net0408	20	21	4.38e-08	1.31e-08	193.73	150.97	42.76
net0416	19	20	5.72e-10	1.10e-13	1062.08	746.55	315.53
net0816	20	21	3.95e-07	2.81e-08	1575.71	1064.75	510.96
net0832	20	21	3.54e-07	3.87e-09	9067.49	5886.13	3181.36
net0864	20	21	1.03e-07	9.87e-09	20954.51	14905.89	6048.62
net1632	22	23	1.34e-07	2.46e-08	15344.07	8867.81	6476.26

Table 2
Computational results for the test problems. A positive value in the last column means that the adaptive algorithm is faster.

5. Conclusion

For interior point methods, with or without predictor-corrector strategies, we have presented an adaptive automated procedure for determining whether to use a direct or iterative solver, whether to reinitialize or update the preconditioner, and how many updates to apply, and demonstrated that it can enhance performance of interior point algorithms on large sparse problems.

Our preconditioning strategy is based on recomputing or updating the previous preconditioner.

Our numerical tests were performed using two specific codes, but it is possible to implement this idea in other codes by adding three pieces:

1. a mechanism to determine whether a direct or iterative solver should be used;
2. a routine that performs updating and downdating of an existing Cholesky factorization; and
3. an iterative solver, such as a preconditioned conjugate gradient method.

Further improvements could be made in the algorithm. Deeper understanding of effective termination criteria for the iterative method may further enhance the efficiency of the algorithm. A block implementation of the matrix updating and downdating would reduce overhead. Finally, parameters such as `max_pcg_itn`, `lrg`, `sml`, and `updt_nmbr` might be tuned to particular problem classes.

Acknowledgements

We are grateful to Roy Marsten for giving us access to OB1-R, and to him and Irvin Lustig for helpful comments. Dianne O'Leary is grateful for the hospitality provided by Professor Walter Gander and the Departement Informatik, ETH Zürich, Switzerland.

References

[1] Erling D. Andersen, Jacek Gondzio, Csaba Mészáros, and Xiaojie Xu. Implementation of interior point methods for large scale linear programming. Technical Report Technical Report 1996.3, Logilab, HEC Geneva, Section of Management Studies, University of Geneva, 102 Bd Carl Vogt, CH-1211 Geneva 4, Switzerland, January 1997.

[2] Richard Bartels and Linda Kaufman. Cholesky factor updating techniques for rank 2 matrix modifications. *SIAM Journal on Matrix Analysis and Applications*, 10(4):557–592, October 1989.

[3] Tamra J. Carpenter and David F. Shanno. An interior point method for quadratic programs based on conjugate projected gradients. *Computational Optimization and Applications*, 2:5–28, 1993.

[4] P. Chin and A. Vannelli. Computational methods for an LP model of the placement problem. Technical Report UWE&CE-94-02, Department of Electrical and Computer Engineering, University of Waterloo, November 1994.

[5] P. Chin and A. Vannelli. Iterative methods for the augmented equations in large-scale linear programming. Technical Report UWE&CE-94-01, Department of Electrical and Computer Engineering, University of Waterloo, October 1994.

[6] In Chan Choi, Clyde L. Monma, and David F. Shanno. Further development of a primal-dual interior point method. *ORSA Journal on Computing*, 2(4):304–311, 1990.

[7] Joseph Czyzyk, Sanjay Mehrotra, and Stephen J. Wright. PCx user guide. Technical Report ANL/MCS-TM-217, Argonne National Laboratory, Argonne, Illinois, 1997.

[8] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, 1979.

[9] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.

[10] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming : Sequential Unconstrained Minimization Techniques*. John Wiley & Sons, New York, 1968. Reprint : Volume 4 of *SIAM Classics in Applied Mathematics*, SIAM Publications, Philadelphia, PA 19104–2688, USA, 1990.

[11] Roland W. Freund and Florian Jarre. A QMR-based interior-point algorithm for solving linear programs. Technical report, AT&T Bell Laboratories and Institut für Angewandte Mathematik und Statistik, 1995.

[12] Matteo Frigo and Steven G. Johnson. The fastest fourier transform in the west. Technical Report MIT-LCS-TR-728, Massachusetts Institute of Technology, September 1997.

[13] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Soc. COAL Newsletter*, 1985.

[14] Philip E. Gill and Walter Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Mathematical Programming*, 7:311–350, 1974.

[15] Philip E. Gill, Walter Murray, Michael A. Saunders, J. A. Tomlin, and Margaret H. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36:183–209, 1986.

[16] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1981.

[17] D. Goldfarb and S. Mehrotra. A relaxed version of Karmarkar's method. *Mathematical Programming*, 40(3):289–315, 1988.

[18] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, second edition, 1989.

[19] Clovis C. Gonzaga. Path-following methods for linear programming. *SIAM Review*, 34(2):167–224, June 1992.

[20] N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[21] N. K. Karmarkar and K. G. Ramakrishnan. Computational results of an interior point algorithm for large scale linear programming. *Mathematical Programming*, 52:555–586, 1991.

[22] J. L. Kennington and R. V. Helgason. *Algorithms for network programming*. John Wiley and Sons, New York, NY, 1980.

[23] D. Klingman, A. Napier, and J. Stutz. NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20(5):814–821, January 1974.

[24] J. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11:141–153, 1985.

[25] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra and Its Application*, 152:191–222, 1991.

[26] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. Interior point methods for linear programming: Computational state of the art. *ORSA Journal on Computing*, 6(1):1–14, Winter 1994.

[27] Sanjay Mehrotra. Implementation of affine scaling methods: Approximate solutions of systems of linear equations using preconditioned conjugate gradient methods. *ORSA Journal on Computing*, 4(2):103–118, 1992.

[28] Sanjay Mehrotra and Jen-Shan Wang. Conjugate gradient based implementation of inte-

rior point methods for network flow problems. Technical Report 95-70.1, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208-3119, U.S.A., October 1995.

[29] Stephen G. Nash and Ariela Sofer. Preconditioning of reduced matrices. Technical Report Report 93-01, Department of Operations Research and Engineering, George Mason University, Fairfax, VA 22030, February 1993.

[30] E. Ng and B. W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. on Scientific Computing*, 14:1034–1056, 1993.

[31] C. C. Paige and M. A. Saunders. LSQR : An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8:43–71, 1982.

[32] L. F. Portugal, M. G. C. Resende, G. Veiga, and J. J. Júdice. A truncated primal-infeasible dual-feasible network interior point method. November 1994.

[33] Robert J. Vanderbei. LOQO : An interior point code for quadratic programming. Program in Statistics and Operations Research, Princeton University. `rvdb@princeton.edu`, 1995.

[34] Richard S. Varga. Factorization and normalized iterative methods. In Rudolph E. Langer, editor, *Boundary Problems in Differential Equations*, pages 121–142. University of Wisconsin Press, Madison, 1960.

[35] Weichung Wang. *Iterative Methods in Interior Point Methods for Linear Programming*. PhD thesis, Applied Mathematics Program, University of Maryland, 1996.

[36] Weichung Wang and Dianne P. O'Leary. Adaptive use of iterative methods in interior point methods for linear programming. Technical Report CS-TR-3560, Computer Science Department, University of Maryland, November 1995. http://www.cs.umd.edu/Dienst/UI/2.0/Describe/ncstrl.umcp/CS-TR-3560.

[37] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *SC 1998 Proceedings (Electronic Publication) http://www.netlib.org/utk/people/JackDongarra/papers.html*. IEEE Press, 1998.

[38] M. H. Wright. Interior methods for constrained optimization. In A. Iserles, editor, *Acta Numerica 1992*, pages 341–407. Cambridge University Press, New York, USA, 1992.