Multiple Centrality Corrections in a Primal–Dual Method for Linear Programming*†

Jacek Gondzio‡
Logilab, HEC Geneva, Section of Management Studies, University of Geneva,
102 Bd Carl Vogt, CH-1211 Geneva 4, Switzerland
e-mail: gondzio@divsun.unige.ch

Technical Report 1994.20
November 11, 1994, revised May 6, 1995

Abstract

A modification of the (infeasible) primal–dual interior point method is developed. The method uses multiple corrections to improve the centrality of the current iterate. The maximum number of corrections the algorithm is encouraged to make depends on the ratio of the efforts to solve and to factorize the KKT systems. For any LP problem, this ratio is determined right after preprocessing the KKT system and prior to the optimization process. The harder the factorization, the more advantageous the higher-order corrections might prove to be.

The computational performance of the method is studied on more difficult Netlib problems as well as on tougher and larger real–life LP models arising from applications. The use of multiple centrality corrections gives on the average a 25% to 40% reduction in the number of iterations compared with the widely used second-order predictor–corrector method. This translates into 20% to 30% savings in CPU time.

Key words. Centrality correctors, higher-order method, linear programming, interior point methods.

---

*Supported by the Fonds National de la Recherche Scientifique Suisse, grant #12-34002.92.

†to appear in *Computational Optimization and Applications.*

‡On leave from the Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland.

1 Introduction

It is now widely accepted [1] that an infeasible primal–dual interior point method is a powerful tool for solving very large linear programs. Most issues concerning its efficient implementation seem now to be deeply understood (see, e.g., [2] for a survey).

All modern codes based on the primal–dual method proceed in a more or less similar way: at every iteration they apply some direct method to factorize the Karush–Kuhn–Tucker (KKT) equations and later solve them twice for predictor and corrector terms in the Newton step [3]. The predictor term is responsible for *optimization* (reducing the primal and dual infeasibilities and the duality gap). The corrector term keeps the current iterate away from the boundary of the feasible region (and ideally keeps it close to the central path [4]) to improve the chances for a long step to be made in the next iteration.

The factorization of the KKT equations almost always involves significantly more computational effort than the solutions for the predictor and corrector direction terms that follow factorization. The ratio of a hundred between these two efforts is quite usual. It is thus natural to seek maximum utilization of the information available from such expensive factorizations in order to reduce their number to a minimum. This desire gave rise to the introduction of higher-order terms when computing directions.

The first such approach was proposed by Karmarkar, Lagarias, Slutsman, and Wang [5] who constructed a parametrized representation of the (feasible) trajectory motivated by the use of differential equations.

One successful technique that incorporates higher-order information in the primal–dual algorithm comes from Mehrotra [3, 6]. This method builds a higher-order Taylor approximation of the (infeasible) central trajectory and pushes an iterate towards an optimum along such an approximation. Mehrotra [6] gives a rigorous presentation of this approach as well as rich computational results.

Another approach, from Domich, Boggs, Rogers, and Witzgall [7] used three independent directions and solved an auxilliary linear program in a three-dimensional subspace to find a search direction.

The method of Sonnevend, Stoer, and Zhao [8] used subspaces spanned by directions generated by higher-order derivatives of the central path, or earlier computed points of it, as a predictor step. This step was later followed by one (or more) centering steps to drive the next iterate sufficiently close to the central path. This approach required feasibility and was tested only in the context of a MATLAB implementation on small and dense problems.

In addition, Hung and Ye [9] studied theoretically higher-order predictor–corrector techniques incorporated in a homogenous self–dual algorithm, and Gonzaga [10] analyzed their use in a primal–dual method.

The most widely used higher-order method is the second-order variant of Mehrotra's predictor–corrector. The computational results of [6] confirmed the advantages of using a second-order method over the use of a pure—first-order—primal–dual algorithm, but at the same time showed that introducing higher-order terms (higher than two) had hardly predictable consequences for the overall efficiency measured with CPU time. The discouraging results concerning the use of the higher-order predictor–corrector method were later confirmed in [11].

We suppose that there are at least two reasons for these findings. The first is a consequence of the sometimes complicated shape of the central trajectory and of the large errors in its approximation based on local information available at a current point. The second is a need to solve $o(n)$ higher-order polynomial inequalities to find the maximum stepsizes in Mehrotra's original algorithm. These computations contribute significantly to the solution time, especially when the order of the approximation grows.

In the method proposed in this article, we will not move along the higher-order approximation of the central trajectory (such an approximation has little chance, in general, to be reliable). Instead, we shall look for a direction–corrector term that improves the centrality of the next iterate and increases stepsizes in the primal and dual spaces. To achieve this, we shall enlarge the stepsizes in both spaces and make a hypothetical further move along the predictor direction to the so–called *trial point*. Note that we shall cross, in general, the boundary of the feasible region. Next, we shall define a corrector direction that drives from this trial point towards some better centered *target* [12]. An important consideration in this approach is that the target is not a (usually unachievable) analytic center but rather is some point in a large neighborhood of the central path that, we hope, may be easier to reach.

We expect advantages to this approach because a trial point is explicitly known and its distance to the central trajectory can be inexpensively measured. The outliers—that is, the very small or very large complementary pairs—are easily identified and remain subject to correction by a modified centering term. Furthermore, the correction process can be repeated the required number of times, and the cost of the $k+1$st correction is exactly the same as that of the kth correction, since the correction resolves to one solution of the KKT system and one ratio test (there is no need to solve $o(n)$ polynomial inequalities of order $k+1$).

A heuristic is given to choose the maximum number of corrections allowed for a given LP problem. It exploits the information available after preprocessing the KKT system for factorization. The smaller the ratio of the computational effort of one solution of the KKT system to that of its factorization, the larger the number of corrections allowed. Naturally, no correction is allowed if the problem has a very inexpensive factorization. Additionally, to save on useless corrections, the process is prematurely terminated if it does not sufficiently improve the quality of the new trial point.

The multiple centrality correction technique has been incorporated into our experimental HOPDM 2.0 code (Higher-Order Primal–Dual Method, version 2.0) [13, 14]. Use of the technique results in a significant reduction in the number of iterations compared with the widely used predictor–corrector methods. This translates into 20% to 30% savings in CPU time on the most difficult problems of our test set.

The article is organized as follows. In section 2 we give a brief description of the primal–dual method; in particular, in section 2.2 we recall the (second-order) predictor–corrector technique in the form in which it is usually implemented. In section 3 we discuss multiple centrality corrections. In section 4 we present computational results, and finally, in section 5 we give our conclusions.

2 Primal–dual algorithm

In this section we very briefly discuss an (infeasible) primal–dual method. The first theoretical results for the primal–dual algorithm come from Megiddo [15] and Kojima, Mizuno, and Yoshise [16]. Descriptions of its efficient implementations can be found in [1, 3, 6, 17] and in the survey [2].

2.1 Fundamentals of the primal–dual method

Let us consider a primal linear programming problem:

$$\begin{aligned} \text{minimize} \quad & c^T x, \\ \text{subject to} \quad & Ax = b, \\ & x + s = u, \\ & x, s \geq 0, \end{aligned} \tag{1}$$

where $c, x, s, u \in \mathcal{R}^n, b \in \mathcal{R}^m, A \in \mathcal{R}^{m*n}$. Let us also consider its dual:

$$\begin{aligned} \text{maximize} \quad & b^T y - u^T w, \\ \text{subject to} \quad & A^T y + z - w = c, \\ & z, w \geq 0, \end{aligned} \tag{2}$$

where $y \in \mathcal{R}^m$ and $z, w \in \mathcal{R}^n$.

Next, let us replace the nonnegativity of constraints in the primal formulation with logarithmic barrier penalty terms in the objective function. It gives the following logarithmic barrier function:

$$L(x, s, \mu) = c^T x - \mu \sum_{j=1}^{n} \ln x_j - \mu \sum_{j=1}^{n} \ln s_j. \tag{3}$$

The first-order optimality conditions for equation (3) are

$$\begin{aligned} Ax &= b, \\ x + s &= u, \\ A^T y + \mu X^{-1} e - w &= c, \\ \mu S^{-1} e - w &= 0. \end{aligned} \tag{4}$$

Substituting

$$z = \mu X^{-1} e,$$

these conditions give

$$\begin{aligned} Ax &= b, \\ x + s &= u, \\ A^T y + z - w &= c, \\ XZe &= \mu e, \\ SWe &= \mu e, \end{aligned} \tag{5}$$

where X, S, Z, and W are diagonal matrices with the elements x_j, s_j, z_j, and w_j, respectively, e is the n-vector of all ones, and μ is a barrier parameter.

A single iteration of the basic primal–dual algorithm makes one step of Newton's method applied to the first-order optimality conditions (5) with a given μ, and then μ is updated (usually decreased). The algorithm terminates when infeasibility and the complementarity gap are reduced below a predetermined tolerance.

Having an $x, s, z, w \in \mathcal{R}^n_+, y \in \mathcal{R}^m$, Newton's direction is obtained by solving the following system of linear equations:

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & 0 & I & 0 & 0 \\ 0 & A^T & 0 & I & -I \\ Z & 0 & 0 & X & 0 \\ 0 & 0 & W & 0 & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \\ \Delta z \\ \Delta w \end{bmatrix} = \begin{bmatrix} \xi_b \\ \xi_u \\ \xi_c \\ \mu e - XZe \\ \mu e - SWe \end{bmatrix}, \tag{6}$$

where

$$\begin{aligned} \xi_b &= b - Ax, \\ \xi_u &= u - x - s, \\ \text{and} \qquad \xi_c &= c - A^T y - z + w, \end{aligned}$$

denote the violations of the primal constraints, primal bounds and the dual constraints, respectively. Primal–dual infeasible interior point methods do not require the feasibility of the solutions (ξ_b, ξ_u, and ξ_c might be nonzero) during the optimization process. Feasibility is attained during the process as optimality is reached. It is easy to verify that if a step of length one is made in Newton's direction (6), then feasibility is reached at once. This is seldom the case since a smaller stepsize usually has to be chosen (a damped Newton iteration is taken) to preserve the nonnegativity of x, s, z, and w. If this latter situation is the case and a stepsize $\alpha < 1$ is applied, then infeasibilities ξ_b, ξ_u, and ξ_c are reduced $1 - \alpha$ times.

The set of linear equations (6) reduces to the *augmented system*

$$\begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix}, \tag{7}$$

or further to the *normal equations system*

$$(AD^2A^T)\Delta y = AD^2 r + h, \tag{8}$$

where

$$\begin{aligned} D^2 &= (X^{-1}Z + S^{-1}W)^{-1}, \\ r &= \xi_c - X^{-1}(\mu e - XZe) + S^{-1}(\mu e - SWe) - S^{-1}W\xi_u, \\ h &= \xi_b. \end{aligned} \tag{9}$$

Computing $(\Delta x, \Delta y)$ from equation (7) or Δy from equation (8) is usually (when a direct approach [18] is applied) divided into two phases: *factorization* of the matrix to some easily invertible form and the following *solution* that exploits this factorization.

Once direction $(\Delta x, \Delta y, \Delta s, \Delta z, \Delta w)$ has been computed, the maximum stepsizes α_P and α_D that maintain nonnegativity of variables in the primal and dual spaces are found. Next, a new iterate is computed using a step-reduction factor $\alpha_0 = 0.99995$:

$$\begin{aligned} x^{k+1} &= x^k + \alpha_0\alpha_P\Delta x, \\ s^{k+1} &= s^k + \alpha_0\alpha_P\Delta s, \\ y^{k+1} &= y^k + \alpha_0\alpha_D\Delta y, \\ z^{k+1} &= z^k + \alpha_0\alpha_D\Delta z, \\ w^{k+1} &= w^k + \alpha_0\alpha_D\Delta w. \end{aligned} \tag{10}$$

After making the step, the barrier parameter μ is updated and the process is repeated.

2.2 Predictor–corrector technique

A *factorization* of the matrix (7) or (8) is usually at least an order of magnitude more expensive than the following *solution* for the direction. The factorizations of KKT systems often take 60% to 90% of the total CPU time needed to solve a problem. It is thus natural to seek a reduction of their number to the necessary minimum, even at the expense of some increase in the cost of a single iteration.

Mehrotra's predictor–corrector technique [3, 6] incorporates higher-order information when approximating the central trajectory and computing the direction step. Formally, this method looks for a parametric representation of the trajectory that guides from a given (infeasible) point in primal and dual spaces to a solution of (1)–(2). It then builds an approximation of this trajectory using the Taylor polynomial expansion derived for the above-mentioned representation.

Computational results [6] showed that the use of the second-order method (with one correction term) gives evident savings over the basic first-order variant of the algorithm. The use of higher-order terms (higher than two) in the given direction further reduces the number of iterations; however, such use does not, in general, translate into computational time savings.

The second-order method thus became the computational state of the art [1, 17] for the following couple of years. This variant of the predictor–corrector method used to be implemented in a simplified way that abused rigorous mathematics, but it proved particularly efficient in computations. Our presentation of it will follow the implementational practice.

The predictor–corrector technique decomposes the direction step $(\Delta x, \Delta s, \Delta y, \Delta z, \Delta w)$ (denoted with Δ for short) from two parts,

$$\Delta = \Delta_a + \Delta_c; \tag{11}$$

i.e., the technique combines affine-scaling (Δ_a) and centering (Δ_c) components. The term Δ_a is obtained by solving system (6) for $\mu = 0$, and Δ_c is the solution of equations like (6) for the right-hand side

$$(0, 0, 0, \mu e - XZe, \mu e - SWe)^T,$$

where $\mu > 0$ is some centering parameter ($\mu = \frac{x^T z + s^T w}{2n}$, for example, refers to centering that does not change the current complementarity gap). The term Δ_a is responsible for *optimization* while Δ_c keeps the current iterate away from the boundary.

Let us observe that the affine-scaling (predictor) direction Δ_a solves the linear system (6) for the right-hand side equal to the current violation of the first-order optimality conditions for (1)–(2), i.e., with $\mu = 0$. This direction is usually "too optimistic" — if a full step of length one could be made in this direction, the LP problem would be solved in one step. The predictor–corrector makes a hypothetical step in this direction. The maximum stepsizes in the primal (α_{Pa}) and in the dual (α_{Da}) spaces preserving the nonnegativity of (x, s) and (z, w), respectively, are determined, and the predicted complementarity gap,

$$g_a = (x + \alpha_{Pa}\Delta x)^T (z + \alpha_{Da}\Delta z) + (s + \alpha_{Pa}\Delta s)^T (w + \alpha_{Da}\Delta w),$$

is computed. It is then used to determine the barrier parameter

$$\mu = \left(\frac{g_a}{g}\right)^2 \frac{g_a}{n}, \tag{12}$$

where $g = x^T z + s^T w$ denotes the current complementarity gap. Let us observe that the term g_a/g measures the achievable progress in the affine-scaling direction. If only a small step in the affine-scaling direction can be made, then g_a/g is close to one and $\mu = g/n$, which means that we only want to improve centrality. If the affine-scaling direction offers considerable progress in the reduction of the complementarity gap, then a more optimistic target (closer to the optimum) is chosen.

For such a μ, the corrector direction Δ_c is computed:

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & 0 & I & 0 & 0 \\ 0 & A^T & 0 & I & -I \\ Z & 0 & 0 & X & 0 \\ 0 & 0 & W & 0 & S \end{bmatrix} \begin{bmatrix} \Delta_c x \\ \Delta_c y \\ \Delta_c s \\ \Delta_c z \\ \Delta_c w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mu e - \Delta X_a \Delta Z_a e \\ \mu e - \Delta S_a \Delta W_a e \end{bmatrix}. \tag{13}$$

Finally, the direction Δ of equation (11) is determined.

A single iteration of the (second-order) predictor–corrector primal–dual method thus needs two solutions of the same large, sparse linear system for two different right-hand sides: first to solve system (6) with $\mu = 0$ and then to solve system (13).

3 Multiple centrality corrections

There are several reasons for the success of the second-order predictor–corrector technique, and understanding these reasons is a key to the use of higher-order corrections.

- The second-order terms $\Delta X_a \Delta Z_a$ and $\Delta S_a \Delta W_a$ correspond to the violation of the complementarity conditions if a full step in the affine-scaling direction were taken. In a computational practice, we optimistically assume that this would be the case. Hence, the corrector direction Δ_c of system (13) is supposed to drive from a hypothetical (and usually unachievable) point $(x + \Delta_a x,\ s + \Delta_a s,\ y + \Delta_a y,\ z + \Delta_a z,\ w + \Delta_a w)$ to some point on the central trajectory.
- The barrier parameter (12) that defines a target on the central path is chosen by a simple but very clever heuristic based on an achievable reduction of the complementarity gap g_a/g when moving along the affine-scaling direction.

Let us observe an obvious inconsistency in the practical use of the predictor–corrector technique. We assume that a *full* step is taken when we compute the second-order term, and we take into account *real* stepsizes when defining a target.

In the method presented in this article, special care is taken to restore the centrality of the next iterate and, at the same time, to increase stepsizes in the primal and dual spaces. The effort of multiple corrections does not concentrate on reducing the complementarity gap (that, we believe, will be sufficiently reduced if a long step along a primal–dual affine-scaling direction is made). Driving the primal–dual point as close to the central path as possible is thus an investment that is expected to pay off in the ability to make a larger step in the next iterate.

3.1 Motivation

Multiple correction terms can be computed in several different ways. Constructing the Taylor or parametric higher-order (higher than two) approximation of the central trajectory seems to be a natural choice. However, extensive computational experience [6] showed that the influence of this approximation on the efficiency of the code was questionable: the method of order two (one corrector term) was shown to be the fastest on the average.

Unfortunately, we belive that it will be impossible in general to build a reliable high-order approximation of the central trajectory whenever this curve has sharp turns. Consequently, in the method presented in this article, we will only use the (very successful) second-order predictor–corrector technique to produce our *predictor* direction. The following *corrector* terms will be concerned with improving the centrality of the subsequent iterate and with increasing the stepsizes in primal and dual spaces.

A natural way to measure centrality is to compute the norm of the residual of the last two equations in (5). The perfectly centered points have the advantageous property that a long step in Newton's direction can be made from them. In practice, however, long steps are observed also for points that belong to a large neighborhood of the central path. What really reduces the efficiency of a primal–dual algorithm is a large discrepancy between the complementarity products $x_j z_j$ and $s_j w_j$. Complementarity products that are either too small or too large (compared with their average $\mu_a = (x^T z + s^T w)/2n$) are undesirable. Our experience indicates that the former might be more disastrous. The following measure of centrality,

$$\delta(x,s,y,z,w,\mu) = \|Xz - \mu e\| + \|X^{-1}Z^{-1}e - \frac{1}{\mu}e\| + \|Sw - \mu e\| + \|S^{-1}W^{-1}e - \frac{1}{\mu}e\|, \quad (14)$$

seems appropriate to evaluate the quality of a given point.

Let us look more closely at the mechanism of the undesireable behavior mentioned above. The step Δ of system (6) aims at drawing all complementarity products to the same value μ. To reduce the complementarity gap, one needs $\mu < \mu_a$. Thus, if the current iterate is badly centered, i.e., if the complementarity products differ in orders of magnitude, then the right-hand side of system (6) is very badly scaled. Newton's direction concentrates on reducing large products, but due to the presence of smaller ones, only small steps are allowed. A theoretical justification exists for the idea that a large discrepancy between complementarity products causes an iterate to be far from the region where Newton's method for centering converges fast. This, in practice, manifests in the small steps allowed in the primal and dual spaces.

Jansen, Roos, Terlaky, and Vial [12] suggest a remedy to this problem: they define a sequence of traceable *targets* (weighted analytic centers). Such a sequence goes from an arbitrary interior point to a point close to the central path. The algorithm follows these targets and continuously improves the centrality of subsequent iterates. Morover, these authors show that the most natural way to define the sequence of targets is to do so in the space of complementarity products.

The method presented in this article translates this idea into a computational practice. The multiple centrality correctors we propose have two complementary goals:

1. achieving larger stepsizes in primal and dual spaces; and

2. improving the centrality (reducing δ) of the current iterate.

The motivation for the first goal is to obtain a faster reduction of the primal and dual infeasibilities, and hence an acceleration of the convergence. The argument for the second one is to increase the chances for a long step to be taken in the following iteration. The next section gives a formal presentation of the method proposed.

3.2 Modified centering directions

Assume that (x, s) and (y, z, w) are primal and dual solutions at a given iteration of the primal–dual algorithm (x, s, z, and w are strictly positive). Next, assume that a *predictor* direction Δ_p at this point has been determined and that the maximum stepsizes in primal (α_P) and dual (α_D) spaces have been computed that preserve nonnegativity of primal and dual variables, respectively.

We look for a *corrector* direction Δ_m such that larger stepsizes in primal and dual spaces are allowed for a composite direction

$$\Delta = \Delta_p + \Delta_m. \tag{15}$$

Assume that we would like to enlarge the stepsizes in the primal and dual spaces from α_P and α_D to $\tilde{\alpha}_P = \min(\alpha_P + \delta_\alpha, 1)$ and $\tilde{\alpha}_D = \min(\alpha_D + \delta_\alpha, 1)$, respectively. To make this possible, a corrector term Δ_m has to compensate for the negative components in the primal and dual variables:

$$\begin{aligned} (\tilde{x}, \tilde{s}) &= (x, s) + \tilde{\alpha}_P(\Delta_p x, \Delta_p s), \\ (\tilde{y}, \tilde{z}, \tilde{w}) &= (y, z, w) + \tilde{\alpha}_D(\Delta_p y, \Delta_p z, \Delta_p w). \end{aligned} \tag{16}$$

Since we expect advantages from improving the centrality of the next iterate $(\hat{x}, \hat{s}, \hat{y}, \hat{z}, \hat{w})$, an additional requirement is imposed on the Δ_m term, namely, to drive the trial point (16) back to the vicinity of the central path. It is an empirical observation that, in general, these two goals—increasing stepsizes and improving centrality—might be contradictory. However, at least for small δ_α, they do not have to be so.

Theoretically, much freedom exists in the choice of a target close to the central path. The most natural guess would be to drive the trial point (16) to the analytic center, i.e., to define the following target in the space of complementarity products:

$$v = (\mu e, \mu e) \in \mathcal{R}^{2n}, \tag{17}$$

where μ is the barrier parameter chosen by the heuristic of Mehrotra (equation (12)). However, extensive computational tests showed this point to be a much too optimistic (hence, unreachable) target. Instead, we compute the complementarity products for the trial point

$$\tilde{v} = (\tilde{X}\tilde{z}, \tilde{S}\tilde{w}) \in \mathcal{R}^{2n} \tag{18}$$

and project them componentwise on a hypercube $H = [\beta_{\min}\mu, \beta_{\max}\mu]^{2n}$ to define the target

$$v_t = \pi(\tilde{v}|H) \in \mathcal{R}^{2n}. \tag{19}$$

We believe that it is much better to concentrate the effort on correcting only outlier complementarity products (the ones that do not belong to the interval $(\beta_{\min}\mu, \beta_{\max}\mu)$).

A corrector term of the direction solves the following system of linear equations:

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & 0 & I & 0 & 0 \\ 0 & A^T & 0 & I & -I \\ Z & 0 & 0 & X & 0 \\ 0 & 0 & W & 0 & S \end{bmatrix} \begin{bmatrix} \Delta_m x \\ \Delta_m y \\ \Delta_m s \\ \Delta_m z \\ \Delta_m w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ v_t - \tilde{v} \end{bmatrix}. \tag{20}$$

Note that the right-hand side in this system is full of zeros, since $v_t - \tilde{v}$ is nonzero only for components that refer to the complementarity products that do not belong to $(\beta_{\min}\mu, \beta_{\max}\mu)$.

Such a defined right-hand-side vector can still remain badly scaled if there are very large complementarity products in $\tilde{v}$. To prevent the undesirable effect of this bad scaling, all components of $v_t - \tilde{v}$ smaller than $-\beta_{\max}\mu$ are, in our implementation, replaced with this value, which corresponds to limiting the expected decrease of large complementarity products.

The modified centering direction, Δ_m, is now used to correct the search direction:

$$\Delta = \Delta_p + \Delta_m, \tag{21}$$

The new stepsizes in the primal and dual spaces $\hat{\alpha}_P$ and $\hat{\alpha}_D$ are determined, and the primal–dual algorithm applies equations (10) to move to the next iterate.

The correcting process can easily be repeated a desirable number of times. Direction Δ of equation (21) becomes in such a case a new predictor Δ_p for which a new trial point is computed from equations (16). The point (18) in the complementarity products space is then used to define the new target (19). Next, a new modified centering direction Δ_m that solves system (20) is computed and added to the predictor term, as in equation (21).

It is important to measure the improvement resulting from the use of the modified centering direction, especially if multiple corrections are allowed. In our implementation, correcting terminates when the stepsizes in the primal and dual spaces $\hat{\alpha}_P$ and $\hat{\alpha}_D$ determined for a composite direction (equation (21)) do not increase sufficiently compared with the stepsizes α_P and α_D found earlier for a predictor direction. Namely, we stop correcting if

$$\hat{\alpha}_P < \alpha_P + \gamma\delta_\alpha \qquad \text{or} \qquad \hat{\alpha}_D < \alpha_D + \gamma\delta_\alpha, \tag{22}$$

where γ is some prescribed tolerance.

The approach presented in this article has several advantages. The first is that it uses real objects—trial points—for which the quality of centrality can inexpensively be measured: if the latter is found unsatisfactory, then the point becomes subject to a reliable correction. We want to stress here that the primary aim of corrections is improving centrality, not obtaining a reduction in the complementarity gap, as suggested in section 6 of [1] and in [11]. Of course, if there are large outliers in the complementarity products (larger than $\beta_{\max}\mu$), then the corrector term is supposed to reduce them; consequently, the complementarity gap may also decrease. Although this decrease is beneficial, we consider it to be a side effect and not the primary goal of our approach.

Another advantage of the method presented in this article is that the computation of every single corrector term requires the same effort. This effort is dominated by the solution of the system of equations (20): the remaining operations—computing the trial point in complementarity products space, projecting it on H, computing the new Δ, and performing a ratio test for it—do not contribute too much. This advantage is in contrast to the approach of [6], in which (apart from the above-mentioned operations) computing the kth-order corrector requires additionally the solution of $o(n)$ polynomial inequalities of order k.

We end this section with a formal presentation of a prototype algorithm of multiple centrality corrections.

Multiple Centrality Corrections

Input

(x, s) and (y, z, w): current pair of primal and dual solutions, respectively;

Parameters

$\beta_{\min}$, $\beta_{\max}$: relative threshold values for outlier complementarity products;
δ_α: the required increase of stepsizes;
γ: the minimum acceptable increase of stepsizes;
K: the maximum number of corrections allowed;

Initialize

Δ_p: predictor direction (Mehrotra's predictor–corrector direction (11));
α_P, α_D: stepsizes along Δ_p in the primal and dual spaces, respectively;
$k = 0$: corrections counter;

Compute correctors

```
begin
    while   k < K   do
        compute trial point (16);
        define the target (19);
        compute the corrector Δ_m from (20);
        perform the ratio test for a composite direction (21);
        Test for improvement:
            if  (α̂_P ≥ α_P + γδ_α  and  α̂_D ≥ α_D + γδ_α)  then
                k = k + 1;
                Δ_p = Δ;
            else
                Δ = Δ_p;
                terminate corrections;
            endif
    end
end
```

In our implementation $\beta_{\min} = 0.1$, $\beta_{\max} = 10$, $\delta_\alpha = 0.1$, and $\gamma = 0.1$. The choice of the maximum number of correctors allowed, K is the subject of the following section.

3.3 How many corrections?

Fortunately, the use of multiple centrality corrections has a predictable influence on the number of iterations needed to solve the problem. We ran the method on a collection of about 200 LP problems (including the *Netlib* set of 90 problems) for different numbers of corrections, and we compared the iteration numbers with those obtained by the predictor–corrector version 2.0 of the HOPDM code [14].

The results have shown that an addition of one centrality corrector term to a predictor-corrector direction reduces the number of iterations from 10% to 30%. More important, for more than 80% of problems tested the use of this term brought at least a 15% reduction in the iterations number; only in a case of 12 problems (known to be rather unstable, like GREENBEA or PILOT.WE from the Netlib set) did it produce an increase in the iteration count.

A second centrality corrector gave again an average 5% to 15% reduction of the iteration number. For more than 75% of the problems, the use of two corrections gave at least a 20% reduction of the iteration number (compared with the predictor–corrector algorithm). Applying more than two centrality correctors produced an additional but less important reduction of the iteration number (0% to 10% on the average).

Summing up, one can expect an almost guaranted 15% to 25% iteration count reduction (over the standard predictor–corrector) from the use of one centrality corrector or a 20% to 35% reduction if two correctors are allowed. These reductions seem encouraging, although they clearly do not always translate into CPU time savings, since their influence on the overall computational effort depends on the ratio of the costs of factorization and solution of the KKT system. Fortunately, this ratio can be measured after preprocessing the KKT system (reordering for sparsity) and before the optimization starts. A simple heuristic can determine the maximum number of centrality corrections K allowed when solving a given problem.

The KKT systems are in the HOPDM code reduced to the normal equations (8). Hence, after neglecting the operations of multiplying the LP constraint matrix, or its transposition, with a vector and that of building AD^2A^T, we use the following measures [2] for factorization and solution efforts:

$$E_f = \sum_{i=1}^{m} l_i^2 \qquad \text{and} \qquad E_s = 2 \times \sum_{i=1}^{m} l_i + 12 \times n, \tag{23}$$

where l_i is the number of off-diagonal nonzero entries in column i of the Cholesky matrix and m, n are the LP problem sizes. The term $12 \times n$ in E_s is expected to account for the necessary ratio tests and vector initializations.

We can now give the estimates of the computational effort of a single iteration in Mehrotra's predictor–corrector method $E_{PC} = E_f + 2E_s$ and in the k-corrections method presented in this article, i.e., $E_{MCC}^{(k)} = E_f + (2+k)E_s$. The average cost of a single iteration in the k-corrections method is then

$$q_k = \frac{E_{MCC}^{(k)}}{E_{PC}} = 1 + \frac{k}{E_f/E_s + 2} \tag{24}$$

times larger than the average cost of the predictor–corrector iteration. If we want to get savings in the overall CPU time to solve the problem, then this increase of the single iteration effort has to be compensated by the more important decrease of the iterations count $p_k = \frac{\text{iters}_{MCC}^{(k)}}{\text{iters}_{PC}}$ (we require $p_k q_k < 1$).

Our estimates of q_k are known in advance and are quite reliable. Unfortunately, we cannot say the same about the estimates of p_k, which have to be derived from the computational experience. We thus proceed in the following way.

We compute the ratio $r_{f/s} = E_f/E_s$ and allow one centrality corrector if $r_{f/s}$ exceeds 10. In such a case, an additional solution adds less than 10% to a single primal–dual iteration, but we expect at least a 15% reduction of the iterations count, and hence a saving in the overall CPU time. If $r_{f/s} \leq 10$, then no centrality corrector is added, so the method reduces to the classical Mehrotra'a predictor–corrector.

Due to the smaller expected savings, the use of the second centrality corrector is allowed only if $r_{f/s}$ exceeds 30. Third correction is allowed if $r_{f/s}$ exceeds 50. More than three correctors are allowed only for problems with very expensive factorizations—namely, if $r_{f/s} > 50 \times p$, then $K = p + 2$ correctors can be added. We do not allow K to exceed 10, but we also observed that problems where more than four correctors are used appear very rarely.

We are aware of certain disadvantages of estimates (23). In general, such estimates should take into account all intermediate steps in the computations (building AD^2A^T, multiplications Ax and A^Ty, etc.). Additionally, they should distinguish between flops done on indirectly accessed data of sparse matrices and directly accessed data of dense window and supernodes in the computation of Cholesky factorization. Furthermore, such estimates should take into account the particularities of different computer architectures. Consequently, the same LP code could choose a different "optimal" number of corrections when run on different platforms. Although this issue may seem important for the developers of commercial software, it is beyond the scope of this article. The estimates (23) correspond to the choice made in our implementation.

4 Computational experience

The method of multiple centrality corrections has been incorporated into our experimental primal–dual code HOPDM (its version 2.0 [14] used Mehrotra's predictor–corrector technique, while the newest version 2.1 applies multiple correctors). HOPDM is written in ANSI FORTRAN 77; hence, it is easily portable to any platform. The results reported here were obtained by running the code on an IBM Power PC workstation (type 7011, model 25T). The program was compiled with options –O –qarch=ppc. All problems are solved to an eight-digit accurate optimum (see e.g., [2], section 3.6). All solution times are in seconds. They include the whole processing time except problem input and solution output.

4.1 Test problems

As stated before, we ran the multiple centrality correction approach on a large set of about 200 LP models. Unfortunately, some of these are proprietary. To avoid an unacceptable practice of publishing (unverifiable) results obtained on problems that are not available to the whole LP community, we limited ourselves to reporting results only for problems from the Netlib collection [19] and some large-scale industry models that we are allowed to distribute.

4.2 Numerical results

We start with a demonstration of the advantages of using multiple centrality correctors on two large Netlib problems, namely, (dense) PILOT87 and (sparse) STOCFOR3 and on two energy-planning models developed at IIASA [20, 21, 22], namely, MOD2 and WORLD. Their original sizes (the number of rows, columns, and nonzero entries in the LP constraint matrix A) are $(m, n, nz) = (2030, 4883, 73804)$, $(m, n, nz) = (16676, 15695, 74004)$, $(m, n, nz) =$

$(35664, 31728, 220116)$, and $(m, n, nz) = (35510, 32734, 220748)$, respectively. The Cholesky factors for these problems contain 435676, 205791, 976878, and 1032555 subdiagonal entries, respectively. The ratios of factorization to solution computational efforts are $r_{f/s} = 194.0$ for PILOT87, $r_{f/s} = 6.3$ for STOCFOR3, $r_{f/s} = 36.1$ for MOD2, and $r_{f/s} = 44.4$ for WORLD. Hence, the maximum numbers of centrality corrections allowed would be set to 5 for PILOT87, 0 for STOCFOR3, 2 for MOD2, and 2 for WORLD. However, to see how the multiple corrections technique works, we let the maximum number of correctors go from 0 through 10, and we marked in bold those runs that correspond to the default "optimal" number of corrections chosen by the heuristic of section 3.3.

Table 1. Efficiency of multiple centrality corrections on some difficult problems.

Method	PILOT87		STOCFOR3		MOD2		WORLD	
	iters	time	iters	time	iters	time	iters	time
Cplex 3.0 - SM	10167	722.2	12712	496.4	117360	18199.9	134270	22469.5
Cplex 3.0 - PD	41	602.8	33	97.7	57	947.0	62	1145.6
HOPDM - 0 corr.	43	824.8	**32**	74.3	61	1247.8	69	1643.2
HOPDM - 1 corr.	31	606.3	29	86.4	50	1050.1	57	1437.0
HOPDM - 2 corr.	27	540.4	25	87.7	**47**	1069.1	**51**	1345.2
HOPDM - 3 corr.	25	508.3	24	97.6	45	1085.4	50	1400.1
HOPDM - 4 corr.	24	498.5	25	103.5	43	1127.3	46	1381.4
HOPDM - 5 corr.	**24**	501.4	24	107.6	41	1113.1	42	1329.0
HOPDM - 6 corr.	24	505.3	24	113.2	38	1116.8	41	1358.1
HOPDM - 7 corr.	25	522.0	24	122.6	39	1149.9	41	1374.7
HOPDM - 8 corr.	25	524.9	24	122.0	37	1108.9	40	1377.8
HOPDM - 9 corr.	25	521.6	24	121.3	40	1259.3	40	1407.5
HOPDM - 10 corr.	25	522.0	24	123.2	35	1104.8	40	1468.9

Table 1 reports the numbers of iterations and the Power PC CPU times to solve these problems to the standard eight-digit accurate optimum. To give the reader some idea of the efficiency of our HOPDM 2.1 primal–dual implementation, table 1 includes the results of running CPLEX 3.0 simplex (SM) and predictor–corrector primal–dual barrier (PD) codes [23]. Let us observe that, in terms of iterations needed to reach optimality, our predictor–corrector code compares favorably with the CPLEX barrier. Where CPU efficiency is concerned, our code turns out to be about 23% faster on the sparse STOCFOR3 problem but loses about 40% on the dense PILOT87 and energy-planning tests. We suppose that the reason for this comes from the simplicity of our home implementation of the Cholesky factorization, which switches to a dense window near the end of the decomposition but does not use more sophisticated enhancements (such as higher-level BLAS routines, supernodes, loop unrolling, etc.).

The analysis of the results collected in table 1 clearly shows that the use of multiple centrality correctors leads to the reduction of the number of iterations performed by the primal–dual algorithm. The first centrality corrector cuts off 28%, 10%, 18%, and 17% of iterations compared with the predictor–corrector algorithm for PILOT87, STOCFOR3, MOD2, and WORLD problems, respectively. Two centrality correctors save 37%, 22%, 23%, and 26% iterations, respectively. These reductions translate into CPU time savings that are important for PILOT87 (up to 40% with order-five corrections) and still significant for MOD2 and WORLD problems (14% and 18%, respectively, for order-two corrections) and into the loss of efficiency for STOC-

FOR3, which is consistent with the ratios $r_{f/s}$ for these problems. Note that the heuristic of section 3.3 proved very successful in the choice of the "optimal" order of corrections appropriate for a given problem.

In the following experiment, we ran HOPDM 2.1 on problems from the Netlib collection [19]. Centrality correctors were allowed only in the case of 31 problems (of the whole 90 LP tests). Table 2 summarizes our experience for 39 larger Netlib problems for which the ratio $r_{f/s}$ exceeds 5.0. It reports the number of off-diagonal nonzero entries in the Cholesky factor ($nz(L)$), the ratio $r_{f/s}$, and the automatically chosen maximum number of centrality correctors (K). Then it reports the solution statistics of the multiple-corrections version of the code and a benchmark, classical predictor–corrector variant of the algorithm (version 2.0 of the HOPDM code). Runs corresponding to the automatically chosen default number of corrections are marked in bold. For the FIT1P and PILOT87 problems, five corrections are allowed in a default run of the HOPDM 2.1, so the last two columns report results for such runs.

The analysis of table 2 results indicates clear advantages for the use of multiple centrality correctors. A wise choice of the maximum number of corrections allowed for a given problem makes savings in CPU time almost certain. These savings vary for different problems, but they reach, on the average, 10% to 30% and show a tendency to increase for more difficult problems.

In the last experiment, we used more difficult, large-scale industry problems. Table 3 gives their statistics both in the original form and after presolve reductions [14], the ratios of factorization and solve efforts ($r_{f/s}$), and the maximum number of correctors chosen automatically by the heuristic of section 3.3 (K). Problems BL through UK are MARKAL economical models for different countries [24, 25, 26, 27]. The remaining problems come from Kennington's collection, available via Netlib. Table 4 reports results of solving these problems with the HOPDM 2.1 code.

Again, the maximum number of corrections allowed is set by hand to 1, 2, 3, or 4, and the automatically chosen order is marked in bold. Due to very expensive factorizations, HOPDM 2.1 allows even seven corrections for the PDS-10 problem. The CPU time savings over the predictor–corrector primal–dual method resulting from the application of multiple centrality corrections are evident and on these problems vary between 10% and 40%.

5 Conclusions

We have presented a computationally attractive technique that introduces multiple centrality corrections to the primal–dual method for linear programming.

The approach has several advantages:

- it leads on the average to important reductions in the number of iterations quite independently of the problems;
- it is able to accurately predict an increase in the cost of a single iteration (the cost of multiple corrections grows linearly with their number);
- it is able to choose a preferable maximum number of corrections for a given problem that is expected to offer maximum CPU time savings; and

- it is easy to implement within the logic of a primal–dual algorithm (or, more generally, any path–following algorithm).

The technique of multiple centrality corrections seems thus a highly valuable option that gives significant CPU time savings on nontrivial problems. More important, these savings have a tendency to increase when the problems get more difficult.

Acknowledgments

The author is grateful to Tamas Terlaky and Jean-Philippe Vial for drawing attention both to a particular role of centrality and to the importance of choosing traceable targets in the primal–dual method.

The author is grateful to Marek Makowski and Sabine Messner from IIASA for submitting the energy-planning test problems MOD2 and WORLD, to Olivier Bahn for submitting MARKAL models, and to the different authors of these models for their granting permision to use them.

Notes

The source code of the version 2.1 of the HOPDM program (including an implementation of the multiple centrality correctors technique) has been made public domain for research and teaching purposes through the Netlib.

Both the large–scale LP test problems MOD2 and WORLD, and the MARKAL problems that were used to test the approach presented in this article are public domain. To obtain them, send a request to the author at gondzio@divsun.unige.ch.

References

[1] I.J. Lustig, R.E. Marsten, and D.F. Shanno. "Interior Point Methods for Linear Programming: Computational State of the Art", ORSA Journal on Computing, vol. 6, pp. 1–14, 1994.

[2] J. Gondzio and T. Terlaky. "A Computational View of Interior Point Methods for Linear Programming", in Advances in Linear and Integer Programming, J. Beasley (ed.), Oxford University Press, Oxford, 1995 (to appear).

[3] S. Mehrotra. "On the Implementation of a Primal–Dual Interior Point Method", SIAM Journal on Optimization, vol. 2, pp. 575–601, 1992.

[4] C.C. Gonzaga. "Path–Following Methods for Linear Programming", SIAM Review, vol. 34, pp. 167–224, 1992.

[5] N. K. Karmarkar, J. C. Lagarias, L. Slutsman, and P. Wang. "Power Series Variants of Karmarkar–Type Algorithms", AT&T Technical Journal, vol. 68, pp. 20–36, 1989.

[6] S. Mehrotra. "Higher Order Methods and their Performance", Technical Report 90–16R1, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA, 1991.

[7] P.D. Domich, P.T. Boggs, J.E. Rogers, and Ch. Witzgall. "Optimizing Over Three-Dimensional Subspaces in an Interior-Point Method for Linear Programming", Linear Algebra and its Applications, vol. 152, pp. 315–342, 1991.

[8] G. Sonnevend, J. Stoer, and G. Zhao. "Subspace Methods for Solving Linear Programming Problems", Technical Report, Institut fur Angewandte Mathematik und Statistik, Universitat Wurzburg, Wurzburg, Germany, January 1994.

[9] P.-F. Hung and Y. Ye. "An Asymptotical $O(\sqrt{n}L)$-Iteration Path–Following Linear Programming Algorithm That Uses Long Steps", Technical Report, Department of Mathematics, University of Iowa, Iowa City, USA, March 1994, SIAM Journal on Optimization (to appear).

[10] C.C. Gonzaga. Private communication, August, 1994.

[11] T.J. Carpenter, I.J. Lustig, J.M. Mulvey, and D.F. Shanno. "Higher Order Predictor–Corrector Interior Point Methods with Application to Quadratic Programming", SIAM Journal on Optimization, vol. 3, pp. 696–725, 1993.

[12] B. Jansen, C. Roos, T. Terlaky, and J.-P. Vial. "Primal–Dual Target Following Algorithms for Linear Programming", Technical Report 93–107, Faculty of Technical Mathematics and Informatics, Technical University Delft, Delft, The Netherlands, 1993, to appear in a special issue of Annals of Operation Research, K. Anstreicher and R. Freund (eds.).

[13] A. Altman and J. Gondzio. "An Efficient Implementation of a Higher Order Primal–Dual Interior Point Method for Large Sparse Linear Programs", Archives of Control Sciences, vol. 2 no. 1/2, pp. 23-40, 1993.

[14] J. Gondzio. "Presolve Analysis of Linear Programs Prior to Applying an Interior Point Method", Technical Report 1994.3, Department of Management Studies, University of Geneva, Switzerland, February 1994, revised in December 1994, ORSA Journal on Computing (to appear).

[15] N. Megiddo. "Pathways to the Optimal Set in Linear Programming", in Progress in Mathematical Programming: Interior Point and Related Methods, N. Megiddo (ed.), Springer–Verlag, New York, 1989, pp. 131–158.

[16] M. Kojima, S. Mizuno, and A. Yoshise. "A Primal–Dual Interior Point Algorithm for Linear Programming", in Progress in Mathematical Programming: Interior Point and Related Methods, N. Megiddo (ed.), Springer–Verlag, New York, 1989, pp. 29–48.

[17] I.J. Lustig, R.E. Marsten, and D.F. Shanno. "On Implementing Mehrotra's Predictor–Corrector Interior Point Method for Linear Programming", SIAM Journal on Optimization, vol. 2, pp. 435-449, 1992.

[18] I.S. Duff, A.M. Erisman, and J.K. Reid. Direct Methods for Sparse Matrices, Oxford University Press, New York, 1989.

[19] D.M. Gay. "Electronic Mail Distribution of Linear Programming Test Problems", Mathematical Programming Society COAL Newsletter, vol. 13, pp. 10-12, 1985.

[20] S. Messner. "User's Guide for the Matrix Generator of MESSAGE II", IIASA WP-84-71a, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1984.

[21] M. Strubegger. "User's Guide for the Post-Processor of MESSAGE II", IIASA WP-84-71b, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1984.

[22] N. Nakicenovic, A. Gruebler, A. Inaba, S. Messner, S. Nilsson, Y. Nishimura, H-H. Rogner, A. Schaefer, L. Schrattenholzer, M. Strubegger, J. Swisher, D. Victor, and D. Wilson. "Long-Term Strategies for Mitigating Global Warming", Energy—The International Journal (special issue), vol. 18, no. 5, pp. 409-601, 1993.

[23] R.E. Bixby. "Progress in Linear Programming", ORSA Journal on Computing, vol. 6, pp. 15–22, 1994.

[24] P. Schaumann et al. "Integrierte Gesamtstrategien der Minderung energiebedingter Treibhausgasemissionen (2005/2020)", Studie im Auftrag der Enquete-Kommission "Schutz der Erdatmosphaere" des 12, Deutschen Budnestages, Stuttgart, September 1994.

[25] J.-P. Waaub and R. Loulou. "CO_2 Control with Cooperation in Quebec and Ontario: A MARKAL Perspective", Energy Studies Review, vol. 4, no. 3, pp. 278-296, 1992.

[26] R. Loulou. "Cost Minimization of Quebec's and Ontario's Energy System in the Long Term", Technical Report, GERAD, McGill University, Montreal, Canada, 1994.

[27] P.A. Okken et al. "Energy Systems and CO_2 Constraints", Technical Report ECN-C-93-014, Energieonderzoek Centrum Nederland, Petten, March 1994.

Table 2. Efficiency of multiple centrality correctors on some Netlib problems.

Problem	Prob. statistics			Pred-Corr		1 corrector		2 correctors		3 correctors		4 correctors	
	$nz(L)$	$r_{f/s}$	K	its	time	its	time	its	time	its	time	its	time
25fv47	32238	26.6	1	28	13.9	**24**	12.4	22	12.5	21	12.4	19	12.8
80bau3b	36588	10.2	1	43	42.6	**31**	35.0	27	35.7	28	42.3	25	43.0
agg	9215	12.0	1	19	2.0	**17**	1.9	14	1.9	13	2.0	13	2.0
agg2	19322	21.0	1	21	4.3	**16**	3.9	15	4.0	15	4.2	15	4.3
agg3	19322	21.0	1	21	4.3	**16**	3.8	15	4.0	15	4.1	14	4.1
bnl1	10317	9.0	0	35	6.0	**21**	4.4	19	4.7	19	5.1	18	5.1
bnl2	81245	59.4	3	30	46.8	24	43.8	22	43.0	**21**	42.8	19	40.9
capri	3486	6.7	0	**17**	1.1	15	1.2	14	1.3	13	1.3	13	1.4
cycle	54809	26.2	1	41	31.4	**30**	26.0	28	27.0	28	28.3	27	29.5
d2q06c	127569	60.4	3	41	142.2	34	102.5	30	95.0	**28**	94.9	27	95.9
d6cube	54470	56.8	3	23	43.3	18	34.4	17	34.6	**17**	36.6	15	34.3
degen2	15365	21.2	1	11	2.7	**11**	2.7	9	2.5	9	2.6	10	2.8
degen3	119202	57.0	3	17	43.6	16	42.3	15	41.3	**15**	42.3	15	43.3
e226	2992	6.1	0	**23**	1.4	18	1.3	16	1.3	17	1.5	16	1.5
etamacro	9895	16.9	1	24	3.1	**20**	2.9	19	3.1	18	3.2	17	3.2
fffff800	8251	10.1	1	32	3.6	**26**	3.6	22	3.5	23	3.8	23	4.2
fit1p	195389	191.	5	19	139.6	20	150.7	17	130.7	14	110.2	(5) **14**	110.8
forplan	2659	8.9	0	**21**	1.4	20	1.6	17	1.6	17	1.7	16	1.7
ganges	11384	5.1	0	**20**	4.0	17	3.9	17	4.5	14	4.1	14	4.4
greenbea	46894	10.4	1	48	36.9	**52**	42.9	50	46.3	48	46.6	50	52.9
greenbeb	45167	10.1	1	47	32.0	**33**	23.9	32	27.0	29	28.0	28	28.3
israel	8560	25.7	1	22	2.3	**17**	2.0	16	2.0	14	1.9	14	2.0
maros	11935	7.0	0	**26**	4.9	21	4.7	19	5.0	17	5.1	17	5.5
nesm	20407	12.8	1	31	11.6	**26**	10.9	21	10.7	22	12.4	20	12.5
perold	22410	23.6	1	37	12.5	**38**	12.9	34	13.2	35	14.0	32	13.5
pilot	191454	108.	4	44	247.8	29	166.4	26	154.6	25	153.1	**23**	145.1
pilot4	14409	22.9	1	34	7.7	**37**	9.2	37	10.0	34	9.8	34	9.9
pilot87	435676	194.	5	43	824.8	31	606.3	27	540.4	25	508.3	(5) **24**	501.4
pilot-ja	39993	37.3	2	38	28.7	40	30.1	**33**	27.5	33	28.2	30	26.7
pilot-we	14714	10.3	1	38	11.3	**43**	13.5	33	12.3	38	15.8	33	15.3
pilotnov	40055	35.2	2	19	14.3	14	12.1	**14**	12.8	14	12.9	14	13.2
sierra	12555	7.0	0	**18**	4.8	15	5.4	14	5.9	14	6.6	13	6.9
stair	14417	22.4	1	15	3.0	**16**	3.1	17	3.4	13	3.0	11	2.7
stocfor2	27418	6.7	0	**19**	5.8	16	6.6	14	7.0	14	7.6	14	8.0
stocfor3	205791	6.3	0	**32**	74.3	29	86.4	25	87.7	24	97.6	25	103.5
truss	58092	18.6	1	18	18.0	**15**	17.0	13	16.8	13	18.2	12	18.4
tuff	6219	11.8	1	15	1.8	**17**	2.1	17	2.3	15	2.2	15	2.5
wood1p	11473	17.4	1	28	24.3	**22**	22.3	22	23.6	23	24.9	22	26.4
woodw	29997	10.4	1	33	24.0	**26**	22.4	28	27.4	23	26.1	21	27.5

Table 3. Statistics of large problems.

Problem	Original size			Size after presolve			$nz(L)$	$r_{f/s}$	K
	m	n	nonz	m	n	nonz			
BL	6325	8018	58607	5468	7369	28030	183363	38.1	2
BL2	6325	8018	58607	5480	7382	28156	191203	42.7	2
CH	3852	5062	42910	3353	4695	17431	91896	30.6	2
CO5	5878	7993	92788	4471	6811	45521	147980	23.2	1
CO9	10965	14851	176346	8510	12630	85804	415051	47.4	2
CQ5	5149	7530	83564	3800	6333	40764	135757	23.5	1
CQ9	9451	13778	157598	7073	11643	76639	361104	44.9	2
GE	10339	11098	53763	8361	9695	34766	263443	39.0	2
NL	7195	9718	102570	6478	9246	39290	277587	54.0	3
UK	10131	14212	128341	6829	9717	40897	175047	23.5	1
cre-a	3516	4067	19054	2994	3938	13798	28470	4.1	0
cre-b	9648	72447	328542	5336	31818	107073	236851	21.4	1
cre-c	3068	3678	16922	2375	3249	11183	25072	4.1	0
cre-d	8926	69980	312626	4102	25188	82940	206747	21.5	1
pds-02	2953	7535	21252	2603	7172	15521	38218	11.0	1
pds-06	9881	28655	82269	9119	27852	60093	561080	129.0	4
pds-10	16558	48763	140063	15587	47729	103048	1603379	263.0	7

Table 4. Efficiency of multiple centrality correctors on large problems.

Problem	Pred-Corr		1 corrector		2 correctors		3 correctors		4 correctors	
	iters	time	iters	time	iters	time	iters	time	iters	time
BL	35	119.2	26	99.4	**24**	100.1	23	102.8	22	102.4
BL2	35	129.8	26	107.2	**26**	114.8	24	115.6	23	115.6
CH	67	96.4	26	48.9	**23**	47.8	23	51.6	21	51.5
CO5	55	133.0	**44**	122.3	39	122.1	38	127.4	37	131.8
CO9	57	491.2	48	443.2	**43**	430.7	40	423.3	36	404.9
CQ5	56	125.2	**33**	86.1	30	86.8	30	93.9	32	105.7
CQ9	54	392.1	43	343.8	**38**	325.1	41	361.4	33	319.8
GE	52	251.4	38	201.7	**35**	200.4	34	214.3	32	205.0
NL	38	228.6	29	188.7	26	182.5	**23**	171.2	23	178.8
UK	34	121.4	**28**	115.3	26	118.5	24	121.4	25	129.0
cre-a	**32**	17.8	25	18.4	24	20.4	20	19.6	22	23.1
cre-b	46	302.4	**36**	266.8	33	275.2	33	289.7	35	328.2
cre-c	**29**	14.2	22	13.2	22	15.5	19	15.3	21	17.4
cre-d	45	260.9	**35**	233.3	34	251.4	30	238.4	28	253.1
pds-02	24	19.1	**19**	19.0	17	20.2	18	23.9	17	23.5
pds-06	42	843.4	28	629.4	26	627.7	26	656.7	**24**	593.8
pds-10	49	4967.8	41	4299.7	32	3560.8	34	4016.4	(7) **29**	3393.7