# Implementation of Interior Point Methods for Large Scale Linear Programming[*][†]

Erling D. Andersen [‡], Jacek Gondzio[§], Csaba Mészáros[¶], Xiaojie Xu[‖]

Technical Report 1996.3
Logilab, HEC Geneva, Section of Management Studies,
University of Geneva, 102 Bd Carl Vogt, CH-1211 Geneva 4, Switzerland
January 24, 1996

**Abstract**

In the past 10 years the interior point methods (IPM) for linear programming have gained extraordinary interest as an alternative to the sparse simplex based methods. This has initiated a fruitful competition between the two types of algorithms which has lead to very efficient implementations on both sides. The significant difference between interior point and simplex based methods is reflected not only in the theoretical background but also in the practical implementation. In this paper we give an overview of the most important characteristics of advanced implementations of interior point methods. First, we present the infeasible-primal-dual algorithm which is widely considered the most efficient general purpose IPM. Our discussion includes various algorithmic enhancements of the basic algorithm. The only shortcoming of the "traditional" infeasible-primal-dual algorithm is to detect a possible primal or dual infeasibility of the linear program. We discuss how this problem can be solved with the homogeneous and self-dual model.

The IPMs practical efficiency is highly dependent on the linear algebra used. Hence, we discuss this subject in great detail. Finally we cover the related topics of preprocessing and obtaining an optimal basic solution from the interior-point solution.

[*]The research of the second author has been supported by the Fonds National de la Recherche Scientifique Suisse, grant #12-42503.94. The research of the third author has been supported by the Hungarian Research Found OTKA N° T-016413.

[†]To appear as a chapter 6 in *Interior Point Methods in Mathematical Programming*, T. Terlaky (ed.), Kluwer Academic Publishers, 1996.

[‡]Department of Management, Odense University, Campusvej 55, DK-5230 Odense M, Denmark, e-mail: eda@busieco.ou.dk.

[§]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 (on leave from the Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland).

[¶]Department of Operations Research and Decision Support Systems, Computer and Automation Research Institute Hungarian Academy of Sciences, Lágymányosi u. 11, Budapest, Hungary, e-mail: meszaros@lutra.sztaki.hu

[‖]Institute of Systems Science, Academia Sinica, Beijing 100080, China.

# 1 Introduction

As early as in the late 1940's, almost at the same time when Dantzig presented the famous simplex method, several researchers, including von Neumann (1947) [68], Hoffman et al.(1953) [41] and Frisch (1955) [27], proposed interior-point algorithms which traverse across the interior of the feasible region in an attempt to avoid the combinatorial complexities of vertex-following algorithms.

However, the expensive computational steps they require, the possibility of numerical instability in the calculations, and some discouraging experimental results led to a consensus view that such algorithms would not be competitive with the simplex method in practice.

In fact, it would have been very difficult to find serious discussion of any approach other than the simplex method before 1984 when Karmarkar [46] presented a novel interior point method, which, as he claimed, was able to solve large-scale linear programs up to 50 times faster than the simplex method. Karmarkar's announcement led to an explosion of interest in interior point methods (IPMs) among researchers and practitioners.

Soon after Karmarkar's publication, Gill et al. [31] showed a formal relationship between the new interior point method and the classical logarithmic barrier method. The barrier method is usually attributed to Frisch (1955) [27] and is formally studied in Fiacco and McCormick [23] in the context of nonlinear optimization. Much research has concentrated on the common theoretical foundations of linear and nonlinear programming. A fundamental theme is the creation of continuously parametrized families of approximate solutions that asymptotically converge to the exact solution. A basic iteration of such a *path-following* algorithm consists of moving from one point in a certain neighborhood of a path to another one called a *target* that preserves the property of lying in the neighborhood of the path and is "near" to the exact solution.

In the past ten years several efficient implementations of interior point methods have been developed. Lustig, Marsten and Shanno [54] have made particularly important contribution in this area with their code OB1. Although the implementations of the simplex method, has improved a lot in the recent years [78, 9, 24], extensive numerical tests (cf. [54]) have indicated conclusively that an efficient and robust implementation of an interior point method can solve many large scale LP problems substantially faster than the state-of-the-art simplex code.

The most efficient interior point method today is the *infeasible-primal-dual algorithm.* Therefore in this chapter we discuss techniques used in an efficient and robust implementation of the primal-dual method. Although the chapter focuses on implementation techniques, some closely related theoretical issues are addressed as well.

Most relevant issues of interior point method implementations are illustrated by computational results. Small set of test problems (from the public domain collections of LPs) is in such a case chosen to illustrate *typical* behavior of the presented implementational techniques. The reader intersested to see excessive numerical results that demonstrate how the given technique works in practice should consult the appropriate references.

The presentation starts in Section 2 with a description of the infeasible-primal-dual method. Most issues of the theory and implementation of this method are now well understood. However, two of them still remain open, namely detecting infeasibility of the problem and the choice of a well-centered starting point. A solution to these problems that is both mathematically elegant and implementable in practice, comes with the use of a *homogeneous and self-dual linear feasibility* model. We will address this model in Section 3.

The practical success of any IPM implementation depends on the efficiency and the reliability of the linear algebra kernel in it. We focus on these issues in Section 4.

The major work in a single iteration of any IPM consists of solving a set of linear equations, the so-called Newton equation system. This system reduces in all IPMs to the problem that is equivalent to an orthogonal projection of a vector on the null space of the scaled linear operator. The diagonal scaling matrix depends on the variant of the method used and it changes considerably in subsequent IPM iterations. All general purpose IPM codes use a *direct* approach [19] to solve the Newton equation system. The alternative, *iterative* methods has not been used as much due to difficulties in choosing a preconditioner. There are two competitive direct approaches for solving the Newton equations: the *augmented system* approach [6, 7] and the *normal equations* approach. The former requires factorization of a symmetric indefinite matrix, the latter works with a smaller positive definite matrix.

In Section 4, we discuss both these approaches in detail, analyse their advantages and point out some difficulties that arise in their implementation. Moreover, we present a unified framework which covers all previously presented techniques. We also briefly discuss hardware dependencies of the implementations.

Other issues related to an efficient implementation of IPMs are addressed in Section 5. We discuss the important role of *preprocessing* the linear program and recall some related problems, such as the impact of the presence of free variables and dense columns in the LP problem.

As mentioned before a direct approach is used to solve a system of Newton equations in every IPM iteration. Therefore, in each iteration a matrix factorization is computed that requires a nontrivial amount of work. In contrast, the following backsolve step is usually significantly cheaper. An obvious idea, known from different applications of the Newton method, is to reuse the factorization in several iterations or, equivalently, to repeat several backsolves to guess a better next iterate. We call such an approach a *higher order method*. The first higher order method was incorporated into a dual affine-scaling method of AT&T's Korbx system [47]. An efficient high-order method was proposed by Mehrotra; his second-order predictor-corrector strategy [62] has been incorporated in all primal-dual type implementations. As shown in Mehrotra [61], the improvement from using orders higher than 2 is very limited. Recently, Gondzio [36] proposed a new way to exploit high order information in a primal-dual algorithm and showed considerable improvements in solving large scale problems. We shall address the use of higher order methods in Section 6.

An important issue is when to terminate an IPM. Contrary to the simplex algorithm an IPM never generates the exact optimal solution; instead it generates an infinite sequence converging towards an optimal solution. Hence, it is necessary to be able to terminate an IPM after a finite number of iterations and report the exact optimal solution. This problem is solved with Ye's finite termination scheme, see [89]. A closely related problem is to generate an optimal basic solution from an optimal interior point solution. In general, if an LP problem has multiple optimal solutions an IPM does not produce an optimal solution which is also a basic solution. Megiddo [59] has shown that if an exact primal and dual optimal solution is known, then an optimal basic solution can be produced in strongly polynomial time using a simplified simplex algorithm. In Section 7, we discuss a method which combines Ye's finite termination scheme and Megiddo's method to produce an optimal basic solution.

Interior point methods are now very reliable optimization tools. Sometimes only for the reason of inertia, the operations research community keeps using the simplex method in applications that could undoubtedly benefit from the new – interior point technology. This is particularly important in those applications which require the solution of very large linear programs (with tens or hundreds of thousand constraints and variables). We thus end the chapter with a brief guide to the interior point software available nowadays. We shall list in Section 8 both commercial and experimental (research) LP codes based on interior point methods. Among the latter, there exist very efficient programs that are public domain in a form of source code and are competitive (in terms of speed) with the best commercial products.

Although the past ten years brought an enormous development of both the theory and the implementations of IPMs, several issues still remain open. We shall address them in Section 9 before giving our conclusions in Section 10.

## 2 The primal–dual algorithm

The computationally most attractive IPM is an infeasible-primal-dual algorithm. Indeed it has been implemented in all commercial software packages. Hence, we start the paper presenting this algorithm.

The algorithm generates iterates which are positive (i.e. are interior with respect to the inequality constraints) but do not necessarily satisfy the equality constraints. Hence, the name *infeasible-interior-point primal-dual method.* For the sake of brevity, we call it the primal-dual algorithm.

The first theoretical results for this method are due to Megiddo [58] who proposed to apply a logarithmic barrier method to the primal and the dual problems at the same time. Independently, Kojima, Mizuno and Yoshise [49] developed the theoretical background of this method and gave the first complexity results.

The first implementations [57, 16] showed great promise and encouraged further research in this field. These implementations have been continuously improved and have led to the development of several highly efficient LP codes. Today's computational practice of the primal-dual implementation follows [51, 53, 54, 62, 36].

The practical implementations of the primal-dual algorithm still differ a lot from the theoretical algorithms with polynomial complexity since the latter give too much importance to the worst-case analysis. This gap between theory and practice has been closed recently by Kojima, Megiddo and Mizuno [48] who show that the primal-dual algorithm with some safe-guards has good theoretical properties.

### 2.1 Fundamentals

Let us consider a primal linear programming problem

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

where $c, x, s, u \in \mathcal{R}^n, b \in \mathcal{R}^m, A \in \mathcal{R}^{m \times n}$ and its dual

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

where $y \in \mathcal{R}^m$ and $z, w \in \mathcal{R}^n$. An LP problem is said to be *feasible* if and only if its constraints are consistent; it is called *unbounded* if there is a sequence of feasible points whose objective value goes to infinity. An LP problem is said to *have a solution* if and only if it is feasible and bounded.

With some abuse of mathematics, to derive the primal-dual algorithm one should:

- replace nonnegativity constraints on the variables with logarithmic barrier penalty terms;
- move equality constraints to the objective with the Lagrange transformation to obtain an unconstrained optimization problem and write first order optimality conditions for it; and
- apply Newton's method to solve these first order optimality conditions (i.e. to solve a system of nonlinear equations).

Let us do this exercise.

Replacing nonnegativity constraints with the logarithmic penalty terms gives the following logarithmic barrier function

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

Next, we write the first order optimality conditions for it

$$\begin{aligned} Ax &= b, \\ x + s &= u, \\ A^T y + z - w &= c, \\ XZe &= \mu e, \\ SWe &= \mu e, \end{aligned} \tag{4}$$

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, $\mu$ is a barrier parameter and $z = \mu X^{-1} e$.

Let us observe that the first three of the above equations are linear and force primal and dual feasibility of the solution. The last two equations are nonlinear and depend on the barrier parameter $\mu$. They become the complementarity conditions for $\mu = 0$, which together with the feasibility constraints provides optimality of the solutions.

It can be seen that (4) is identical to the Karush-Kuhn-Tucker (KKT) system for the LP problem, in which the complementarity conditions are perturbed by $\mu$. Hence, (4) is called the perturbed KKT conditions.

A nonnegative solution of (4) is called an *analytic center*. It clearly depends on the value of the barrier parameter $\mu$. The set of such solutions $(x(\mu), s(\mu))$ and $(y(\mu), z(\mu), w(\mu))$ defines a trajectory of centers for the primal and dual problem, respectively and is called the *central path*. The quantity

$$g = x^T z + s^T w,$$

measures the error in the complementarity and is called a *complementarity gap*. Note that for a feasible point, this value reduces to the usual duality gap. For a $\mu$-center, for example,

$$g = 2\mu e^T e = 2n\mu, \tag{5}$$

and it vanishes at an optimal solution.

One iteration of the primal-dual algorithm makes one step of Newton's method applied to the first order optimality conditions (4) with a given $\mu$ and then $\mu$ is updated (usually decreased). The algorithm terminates when the infeasibility and the complementarity gap are reduced below predetermined tolerances.

Given an $x, s, z, w \in \mathcal{R}^n_+$, $\quad y \in \mathcal{R}^m$, 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 and the dual constraints, respectively. We call the linear system (6) the Newton equations system.

Note that the primal-dual method does not require feasibility of the solutions ($\xi_b, \xi_u$ and $\xi_c$ might be nonzero) during the optimization process. Feasibility is attained during the process as optimality is approached. It is easy to verify that if a step of length one is made in the Newton's direction (6), then feasibility is reached immediately. This is seldom the case as a smaller stepsize usually has to be chosen (a damped Newton iteration is taken) to preserve positivity of $x, s, z$ and $w$. If this is the case and a stepsize $\alpha < 1$ is applied, then infeasibilities $\xi_b, \xi_u$ and $\xi_c$ are reduced by a factor $(1-\alpha)$.

Let us take a closer look at the Newton equation system. After elimination of

$$\begin{aligned} \Delta z &= X^{-1}(\mu e - XZe - Z\Delta x), \\ \Delta s &= \xi_u - \Delta x, \\ \Delta w &= S^{-1}(\mu e - SWe - W\Delta s) = S^{-1}(\mu e - SWe - W\xi_u + W\Delta x), \end{aligned} \tag{7}$$

it reduces to

$$\begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix}, \tag{8}$$

where

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

The solution of the reduced Newton equations system (8) is the computationally most involved step of any interior point method. We shall discuss it in detail in Section 4.

Once the system (8) has been solved, $\Delta x$ and $\Delta y$ are used to compute $\Delta s, \Delta z$ and $\Delta w$ by (7). Next the maximum stepsizes in primal space ($\alpha_P$) and dual space ($\alpha_D$) are computed such that the nonnegativity of variables is preserved. These stepsizes are slightly reduced with a factor $\alpha_0 < 1$ to prevent hitting the boundary. Finally a new iterate is computed as follows

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

After making the step, the barrier parameter $\mu$ is updated and the process is repeated.

## 2.2 From theory to computational practice

In the previous section we have outlined the primal–dual algorithm. Now, we shall address some practical issues of its implementation.

From theory it is known that if the barrier parameter is only reduced slightly in each iteration it is possible to take long steps in the Newton direction. It implies fast convergence of Newton's method and all iterates are close to the central path. In practice it is not efficient to reduce the barrier parameter slightly in every iteration and stay very close to central path. (Recall we want to find a solution where the barrier parameter is zero). On the other hand it is not efficient to move too far away from the central path and close to the boundary, because in that case the algorithm might get stuck taking small step in the Newton direction. Hence, convergence will be painfully slow.

**Starting point**

The first difficulty arising in implementing the primal–dual method is the choice of an initial solution. (Note that this problem is solved in an elegant way when a homogeneous model is used, cf. Section 3.) One would like this point to be well centered and to be as close to primal and dual feasibility as possible. Surprisingly, points that are relatively close to the optimal solution (but are not well centered) often lead to bad performance and/or numerical difficulties.

Mehrotra [62] has proposed to solve a certain quadratic programming problem to obtain the initial solution. We will now present a variant of his idea. As starting solution we use the optimal solution to the following QP problem

$$\begin{array}{ll} \text{minimize} & c^T x + \frac{\varrho}{2}(x^T x + s^T s) \\ \text{subject to} & Ax = b, \\ & x + s = u, \end{array} \tag{11}$$

where $\varrho$ is a predetermined weight parameter. A solution of (11) can be given by an explicit formula and can be computed at a cost comparable to a single interior point iteration. It is supposed to minimize the norm of the primal solution $(x, s)$ and it promotes points that are better in the sense of the LP objective. As the solution of (11) may have negative components in $x$ and $s$, those negative components are pushed towards positive values sufficiently bounded away from zero (all elements smaller than $\delta$ are replaced by $\delta$, say, $\delta = 1$). Independently, an initial dual solution $(y, z, w)$ is chosen similarly to satisfy $y = 0$ and the dual constraint (2). Again, all elements of $z$ and $w$ smaller than $\delta$ are replaced by $\delta$.

### Stepsize

The simplest way to ensure that all iterates remain close to the central path is to decrease the barrier parameter slowly in subsequent IPM iterations. This gave rise to so called *short step methods* that are known to have nice theoretical properties but they are also known to demonstrate hopelessly slow convergence in practice.

In *long step methods* the barrier parameter is reduced much faster than what the theory suggests. To preserve good convergence properties of this strategy the theory requires that several Newton steps are computed within each primal-dual iteration such that the new point is in a close neighborhood of the central path. In practice this is ignored and only one Newton step is made before the barrier parameter is reduced. A negative consequence of it is that the iterates cannot be kept close to the central path. However, the computational practice shows that even if they remain in a relatively large vicinity of the central path, the algorithm still converges fast.

The barrier parameter is chosen as some fraction of the average complementarity product at the current point (cf. equation (5))

$$\mu_{new} = \gamma \mu_{average} = \gamma (g/2n), \tag{12}$$

where $\gamma \in [0,1]$. The choice of $\gamma = 1$ corresponds to a pure recentering step while the choice of $\gamma < 1$ is expected to reduce the complementarity gap in the next iterate. Indeed if the iterates are feasible the complementarity gap is guaranteed to be reduced by a factor $(1 - \alpha(1 - \gamma))$.

The choice of $\gamma$ or, more generally, the choice of a point (so-called target) to which the next iterate will hopefully be taken is a crucial issue for the efficiency of the primal–dual method. We shall discuss it in detail in Section 6.

Let us observe that current implementations use different stepsizes in the primal and dual spaces. This implies that the infeasibility is reduced faster than if the same stepsize was used. All implementations use a variant of the following strategy. First the maximum possible stepsizes are computed by the formulae

$$\begin{array}{rrcl} & \alpha_P & := & \max \alpha > 0 : \ (x,s) + \alpha(\Delta x, \Delta s) \geq 0, \\ \text{and} & \alpha_D & := & \max \alpha > 0 : \ (z,w) + \alpha(\Delta z, \Delta w) \geq 0, \end{array} \tag{13}$$

and these stepsizes are slightly reduced with a factor $\alpha_0 = 0.99995$ to ensure that the new point is strictly positive. Some codes use smaller $\alpha_0$ in those iterations in which 0.99995 might be too aggressive. However, in most cases this aggressive choice of $\alpha_0$ seems to be the best.

In general, the algorithm cannot be guaranteed to be globally convergent with the choice $\alpha_0 = 0.99995$. However, Kojima, Megiddo and Mizuno [48] has proved global convergence of a variant of the primal–dual method that allows the aggressive choice of $\alpha_0$ in most iterations. To ensure global convergence, the stepsizes must be chosen such that the infeasiblities converge faster to zero than the complementarity gap and the iterates are not allowed to move too far away from the central path. For most LP problems with the default starting point (described previously) the additional safe-guards are not constraining for the stepsize.

### Stopping criteria

Interior point algorithms terminate when the first order optimality conditions (4) are satisfied with some predetermined tolerance. In the case of the primal–dual method, this translates to the following conditions imposed on the relative primal and dual feasibility and the relative duality gap

$$\frac{||Ax - b||}{1 + ||b||} \leq 10^{-p} \quad \text{and} \quad \frac{||x + s - u||}{1 + ||u||} \leq 10^{-p}, \tag{14}$$

$$\frac{||A^T y + z - w - c||}{1 + ||c||} \leq 10^{-p}, \tag{15}$$

$$\frac{|c^T x - (b^T y - u^T w)|}{1 + |b^T y - u^T w|} \leq 10^{-p} \tag{16}$$

where $p$ is the number of digits accurate in the solution. An 8–digits exact solution ($p = 8$) is typically required in the literature.

Let us observe that conditions (14-16) depend strongly on the scaling of the problem. In particular, the denominators of their left hand sides usually decrease after scaling of the problem.

In practice, it is rare that condition (16) is satisfied and at the same time one of the conditions (14) or (15) does not hold. The explanation of this phenomena comes from the analysis of the first order optimality conditions (4). Observe that the first three equations, that impose primal and dual feasibility, are linear. They are thus "easier" to satisfy for Newton's method than the last two equations that are nonlinear and, additionally, change in subsequent interior point iterations. Consequently, the most important and perhaps the only condition that really has to be checked is (16).

**Complexity**

At least at one point the theory is still far from the the computational practice; it is in the estimates of the worst–case complexity. Theoretical bound of $\mathcal{O}(\sqrt{n}\ \log \frac{1}{\epsilon})$ iterations to obtain an $\epsilon$–exact solution to an LP is still extremely pessimistic as, in practice, the number of iterations is something like $\mathcal{O}(\log n)$ or $\mathcal{O}(n^{1/4})$. It is rare that the current implementation of the primal–dual method uses more than 50 iterations to reach $10^{-8}$–optimality.

# 3 Self-dual embedding

Two important elements in the primal-dual algorithm has not been solved satisfactorily from a practical point of view.

The first element is the choice of a initial solution. Even though the heuristic presented in the previous section works well in practice, it is scaling dependent and there is no guarantee that the method is producing a well-centered point.

The second element is the lack of a reliable technique to detect infeasibility or unboundedness of the LP problem. The infeasibility or unboundedness of one of the problems (1) and (2) usually manifests in a rapid growth of the primal or dual objective function and immediately leads to numerical problems. This is really a critical point in any implementation of the primal-dual algorithm.

The algorithm presented in this section removes both these drawbacks. It is based on a skew-symmetric and self-dual artificial LP model first considered by Ye et al. [90]. Somewhat later Jansen et al. [45] presented the skew-symmetric self-dual model for a primal-dual pair in a symmetric form. Xu et al. [86, 87] considered a *homogeneous and self-dual linear feasibility* (HLF) model that was in fact studied already in the 60s by Goldman and Tucker [33, 80]. Xu [84, 85] developed a large step path following LP algorithm based on the HLF model and implemented it.

The main advantage of the algorithm is it solves the LP problem without any regularity assumption concerning the existence of optimal, feasible, or interior feasible solutions. If the problem is infeasible or unbounded, the algorithm correctly detects the infeasibility for at least one of the primal and dual problems. Moreover, the algorithm may start from any positive primal-dual pair, feasible or infeasible, near the central ray of the positive orthant. Finally, even if the algorithm takes large steps it achieves $O(\sqrt{n}L)$-iteration complexity.

Compared to the primal-dual method from the previous section this algorithm has only one disadvantage: it requires one additional solve with the factorization of the Newton equation matrix in each iteration.

### 3.1 HLF model

Let us now present the HLF model. For the sake of simplicity, we will work throughout this section with a simplified primal LP formulation (in which all primal variables are nonnegative and without upper bound) that is

$$\text{minimize} \quad c^T x \quad \text{subject to} \quad Ax = b, \quad x \geq 0, \tag{17}$$

where $c, x \in \mathcal{R}^n, b \in \mathcal{R}^m, A \in \mathcal{R}^{m \times n}$ and its dual

$$\text{maximize } b^T y \quad \text{subject to} \quad A^T y \leq c, \tag{18}$$

where $y \in \mathcal{R}^m$. Introducing a homogeneous variable $\tau$ and coupling the primal and dual problem together gives the homogeneous and self-dual linear feasibility model

$$\begin{array}{rccl} & Ax & -b\tau & = 0, \\ -A^T y & & +c\tau & \geq 0, \\ b^T y & -c^T x & & \geq 0, \\ y \text{ free}, & x \geq 0, & \tau \geq 0. & \end{array} \tag{19}$$

This linear feasibility system is homogeneous and has zero as its trivial solution. The zero solution is of course not of interest, but LP theory tells us that a strictly complementary solution exists to any linear program. Now the HLF model (19) is an LP problem with zero objective function and a zero right hand side. Furthermore, it is self-dual. Denote by $z$ the slack vector for the second (inequality) constraint and by $\kappa$ the slack scalar for the third (inequality) constraint. By the skew-symmetric and self-dual property, the complementary pairs are $(x, z)$ and $(\tau, \kappa)$. A strictly complementary solution for the HLF model satisfies (19) and

$$\begin{array}{rl} Xz = 0, & \tau\kappa = 0, \\ \text{and} \quad x + z > 0, & \tau + \kappa > 0 \end{array} \tag{20}$$

where $X = diag(x)$. Let $(y^*, x^*, \tau^*, z^*, \kappa^*)$ be a strictly complementary solution of the HLF model. We can prove the following:

- If $\tau^* > 0$, then $(y^*/\tau^*, x^*/\tau^*, z^*/\tau^*)$ is an optimal strictly complementary solution to (17) and (18).
- If $\tau^* = 0$, then $\kappa^* > 0$, which implies that $c^T x^* - b^T y^* < 0$, i.e. at least one of $c^T x^*$ and $-b^T y^*$ is strictly less than zero. If $c^T x^* < 0$ then (18) is infeasible; if $-b^T y^* < 0$ then (17) is infeasible; and if both $c^T x^* < 0$ and $-b^T y^* < 0$, then both (17) and (18) are infeasible.

### 3.2 A path following algorithm

Due to the third constraint of (19), the HLF model does not have a feasible interior point. Therefore, a definition of a central path similar to (4) makes no sense since it is restricted to the interior feasible region. In this subsection we will define a central path, which connects any given initial positive pair $(x, z)$ and $(\tau, \kappa)$ and a strictly complementary solution of the HLF model. Afterwards an algorithm is developed based on following such an "infeasible" central path to a strictly complementary solution.

For any $(y, x > 0, \tau > 0, z > 0, \kappa > 0)$, the feasibility residuals and the average complementarity residual are defined as

$$\begin{aligned} r_P &= b\tau - Ax, \\ r_D &= c\tau - A^T y - z, \\ r_G &= c^T x - b^T y + \kappa, \\ \text{and} \quad \mu &= (x^T z + \tau\kappa)/(n+1), \end{aligned} \tag{21}$$

respectively. Given $(y^0, x^0 > 0, \tau^0 > 0, z^0 > 0, \kappa^0 > 0)$, the following barrier problem with a parameter $\lambda$ defines a central path:

$$\begin{array}{lllllllll} \text{minimize} & z^T x & +\kappa\tau & -\lambda\mu^0 \sum_i (\ln x_i + \ln z_i) - \lambda\mu^0(\ln\tau + \ln\kappa) & & & & & \\ \text{subject to} & & A\,x & -b\,\tau & & & = & -\lambda\, r_P^0, \\ & -A^T\,y & & +c\,\tau & -z & & = & \lambda\, r_D^0, \\ & b^T\,y & -c^T\,x & & & -\kappa & = & -\lambda\, r_G^0. \end{array} \tag{22}$$

where $(r_P^0, r_D^0, r_G^0)$ and $\mu^0$ are initial residuals at $(y^0, x^0 > 0, \tau^0 > 0, z^0 > 0, \kappa^0 > 0)$. As shown in Xu [87], it is essential to introduce feasibility residual terms in the right hand sides of (22). Along the central path, the feasibility and complementarity residuals are reduced at the same rate and eventually converge to zero. The same rate of reduction in the feasibility and complementarity residuals guarantee that the limit point is a strictly complementary solution of the HLF model (19).

By using the skew-symmetric property, the first order optimality conditions for the barrier problem (22) are

$$\begin{array}{rrrrrcl} & A\,x & -b\,\tau & & & = & -\lambda\, r_P^0, \\ -A^T\,y & & +c\,\tau & -z & & = & \lambda\, r_D^0, \\ b^T\,y & -c^T\,x & & & -\kappa & = & -\lambda\, r_G^0, \\ & & & Xz & & = & \lambda\,\mu^0 e, \\ & & & & \tau\kappa & = & \lambda\,\mu^0, \\ & x, & \tau, & z, & \kappa & > & 0. \end{array} \tag{23}$$

for $\lambda \in (0, 1]$.

It is worth to compare this system with the analogous first order optimality conditions (4) used in the primal-dual algorithm presented in the previous section. Note, for example, that conditions (23) define the central path even though the model (19) has not an interior point. This is important when highly degenerate problems are solved. Indeed for this reason it might be helpful, to add feasibility residuals into (4).

Analogously to the primal-dual algorithm, the search direction for the "infeasible" path following algorithm is generated by applying Newton's method to (23). Actually, in each iteration the algorithm solves the following linear equation system for the direction $(\Delta y, \Delta x, \Delta\tau, \Delta z, \Delta\kappa)$:

$$\left[\begin{array}{ccc|cc} & A & -b & & \\ -A^T & & c & -I & \\ b^T & -c^T & & & -1 \\ \hline & Z & & X & \\ & & \kappa & & \tau \end{array}\right] \left[\begin{array}{c} \Delta y \\ \Delta x \\ \Delta\tau \\ \Delta z \\ \Delta\kappa \end{array}\right] = \left[\begin{array}{c} (1-\gamma)\, r_P^k \\ -(1-\gamma)\, r_D^k \\ (1-\gamma)\, r_G^k \\ \gamma\mu^k e - X^k z^k \\ \gamma\mu^k - \tau^k\kappa^k \end{array}\right] \tag{24}$$

where $(r_P^k, r_D^k, r_G^k)$ and $\mu^k$ are residuals at the current point $(y^k, x^k > 0, \tau^k > 0, z^k > 0, \kappa^k > 0)$ and $\gamma \in [0,1]$ is a chosen reduction rate of the barrier (or path) parameter. Setting $\gamma = 0$ yields an affine direction, and setting $\gamma = 1$ yields a pure centering direction. After the Newton direction has been computed, a stepsize is chosen, using the same method as in the primal-dual algorithm, such that the new point is stricly positive.

The algorithm continues until one of the following stopping criteria is satisfied.

- The LP problem is infeasible (or near infeasible) if

$$\mu/\mu^0 < 10^{-14}, \quad \frac{\tau}{\kappa}/\frac{\tau^0}{\kappa^0} < 10^{-8}.$$

- Optimal (approximate) solution is obtained if

$$\frac{|c^T x - b^T y|}{\tau + |b^T y|} < 10^{-8}, \quad \frac{||r_P||}{\tau + ||x||} < 10^{-8}, \quad \frac{||r_D||}{\tau + ||z||} < 10^{-8}.$$

If the step length is chosen such that the updated solution is still in a certain neighborhood of the central path, then a worst case polynomial complexity result can be established. Xu [84] restricted all iterates to stay within an intersection of an $\infty$-norm neighborhood and a large 2-norm neighborhood of the central path. In this case, the implementation achieves $O(\sqrt{n}L)$-iteration complexity in the worst case.

Clearly the dimension of the Newton equation system solved by the homogeneous algorithm is slightly larger than the corresponding system solved in the primal-dual method. In fact the dimension is increased by exactly one. The primal-dual method can implemented such that the same factorization as in primal-dual method is computed in each iteration. However, the factorization must be used in one more solve to compute the solution of the Newton equation system, see [86] for details.

# 4 Solving the Newton equations

In Section 2 we noted that the solution of the Newton equations system is the computationally most involved task in the primal-dual method. This system reduces, in practice, to the following set of equations

$$\begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix}. \tag{25}$$

It should be noted that all IPMs solve an identical system of linear equations. The only difference is in the value of the diagonal matrix $D^2$ and the right-hand side. This is the reason why the comparison of different variants of interior point methods is often simplified to a comparison of the number of iterations (Newton steps).

The linear system (25) can be solved using either *direct* or *iterative* methods. Iterative methods, e.g., conjugate gradient algorithms are not competitive in general case due to the difficulties in choosing a good and computationally cheap preconditioner. Some success with iterative methods for special LP problems has been obtained, see [71, 70]

Consequently, all state of the art implementations of the general purpose IPMs use a *direct* approach [19] to solve the Newton equations. We can be even more specific to say that they all use some variant of the symmetric triangular $L\Lambda L^T$ decomposition, where $L$ is a lower triangular matrix and $\Lambda$ is a block diagonal matrix with blocks of dimension 1 or 2. To complete the discussion, let us mention an alternative direct approach – the QR decomposition of $A$. Although this approach uses orthogonal transformations and guarantees high accuracy of solutions, it cannot be used in practice since it is prohibitively expensive.

Summing up, the only practicable approach to solve the Newton equations in general purpose IPM codes is the $L\Lambda L^T$ decomposition. There exist numerous variants of its implementations. They differ essentially in restrictions imposed on the choice of the pivot order and, from some perspective, they can all be viewed within the same unifying framework that we shall present later in this section. We will be able to do it after we will have described the two major alternative approaches. The first one reduces (25) to the *normal equations*

$$(AD^2A^T)\Delta y = g = AD^2r + h \tag{26}$$

by pivoting down the diagonal elements of $-D^{-2}$ in (25). The other approach solves the *augmented system* (25) directly without necessarily pivoting in the $-D^2$ part first.

Next we shall address some technical aspects of the implementation and its dependency on the computer hardware. Due to the rapid changes in the computing technology, a detailed discussion of the effect of computer hardware goes beyond the scope of this book. We shall display, however, several important points where different computer architectures influence the efficiency the most. Finally, we shall discuss some issues of accuracy control within IPM implementations.

## 4.1 The normal equations approach

An advantage of the normal equations approach is that it works with a positive definite matrix $AD^2A^T$ (we assume that the LP constraint matrix has full row rank; $D^2$ is positive definite by definition). Thus the Cholesky decomposition of this matrix exists for any $D^2$ and numerical pivoting is not necessary to maintain stability. Moreover, the sparsity pattern in the decomposition is independent of the value of $D^2$ and hence is constant in all IPM iterations. Consequently, a good sparsity preserving pivot order can be chosen with much care (even if it involves considerable computational effort) since it will be used extensively throughout the whole solution process. This argument has been used to justify the application of the normal equations approach in the first professional IPM implementations [1, 47, 57].

The success of the implementation of the Cholesky factorization depends on the quality of its *analysis* phase [19, 29], i.e. reordering for sparsity. Its goal is to find a permutation matrix $P$ such that the Cholesky factor of $PAD^2A^TP^T$ is the sparsest possible. In practice, heuristics are used to solve this problem since finding an optimal permutation (that is, by the way, an NP-complete problem [88]) would be unacceptably expensive. Two such heuristics, namely the *minimum degree* and the *minimum local fill-in* orderings [19, 29, 30] are particularly useful in the context of IPM implementations. They are both local, i.e. they rely on the pivot choice limited to a small subset of the most attractive pivot candidates. Let us briefly discuss these two heuristics.

### Minimum degree ordering

Assume that in the $k$th step of the Gaussian Elimination, the $i$th column of the Schur complement contains $c_i$ nonzero entries and its diagonal element becomes a pivot. The $k$th step of elimination requires thus

$$f_i = \frac{1}{2}(c_i - 1)^2, \tag{27}$$

floating point operations *flops* to be executed. We exploit the fact that the decomposed matrix $AD^2A^T$ is positive definite so the pivot choice can be limited to the diagonal elements. In fact, this choice has to be limited to diagonal elements to preserve symmetry. Function $f_i$ evaluates the computational effort and gives an overestimate of the fill-in that can result from the elimination if the $i$th diagonal element becomes a pivot ($f_i$ is the Markowitz merit function [55] applied to a symmetric matrix [79]).

The "best" pivot at step $k$, in the sense of the number of flops required to perform the elimination step, is the one that minimizes $f_i$. Interpreting this process in terms of the elimination graph [29], one can see that it is equivalent to the choice of the node in the graph which has the minimum degree (this gave the name to this heuristic). The minimum degree ordering algorithm can be implemented efficiently both in terms of speed and storage requirements. For details, the reader is referred to the excellent summary in [30].

### Minimum local fill–in ordering

Let us observe that, in general, function (27) considerably overestimates the expected number of fill–ins in a given iteration of the Gaussian Elimination because it does not take into account the fact that in many positions of the predicted fill–in, nonzero entries already exist. It is possible that another pivot candidate, although more expensive in terms of (27), would produce less fill–in as the elimination step would mainly update already existing nonzero entries of the Schur complement. The minimum local fill–in ordering chooses such a pivot. Generally, the minimum local fill–in algorithm produces a sparser factorization but at higher initial cost to obtain the ordering [54], because the analysis that exactly predicts fill–in and chooses the pivot producing its minimum number is very expensive.

Another efficient technique to determine the pivot order has been proposed in [65]. The method first selects a set of attractive pivot candidates and, in the next step, from this smaller set chooses the pivot that generates the minimal predicted fill–in. Computational experience shows considerable improvement in speed without the loss in the quality of the ordering.

### Numerical examples

To give the reader some rough idea about the advantages of the two competitive ordering schemes, we shall compare their performance on a subset of medium scale linear problems from the Netlib collection [28]. Table 1 collects the results of this comparison. Abbreviations MDO and MFO in it denote the minimum degree ordering and the minimum local fill-in ordering, respectively.

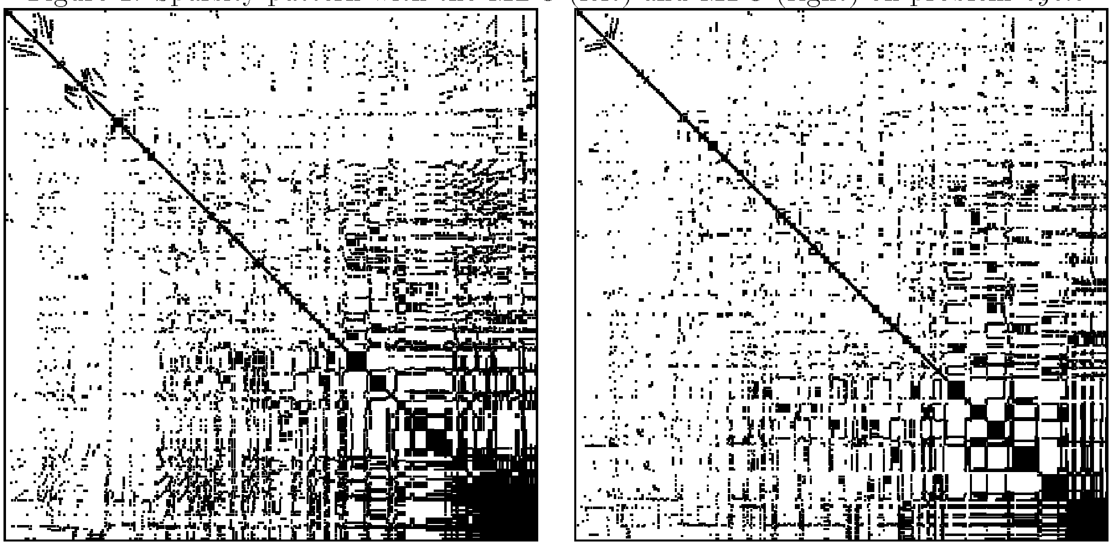
The first three columns of Table 1 contain the problem names and the times (in seconds) of the analysis phase for the two orderings considered. The analysis time includes the setup for the ordering (i.e. building a representation of $AA^T$), the ordering time, and the time for building the nonzero patterns of the Cholesky factors. For both algorithms, the ordering time is the dominating factor. The following columns contain the number of nonzeros in the Cholesky factors produced by the two orderings, the number of flops (in thousand) needed to compute the factorization including flops required by the computation of $AA^T$. The last two columns contain the average time (in seconds) to execute one factorization on a SUN Sparc-10 workstation.

The results presented in Table 1 indicate that MDO is usually faster than MFO (`degen3` is one exception) but it usually produces denser Cholesky factors. Without going into details, we note that on problems where the nonzeros of $AA^T$ are concentrated in a tight band near the diagonal (e.g.: `grow22, maros-r7`), MFO does not offer any advantage over MDO. In contrast,

Table 1: Comparison of minimum degree (MDO) and minimum local fill–in (MFO) orderings

| Name | Analysis time | | Nonzeros in L | | Flops in thousand | | Factorization time | |
|---|---|---|---|---|---|---|---|---|
| | MDO | MFO | MDO | MFO | MDO | MFO | MDO | MFO |
| 25fv47 | 0.50 | 1.38 | 32984 | 27219 | 1282 | 811 | 0.345 | 0.244 |
| 80bau3b | 1.22 | 2.12 | 37730 | 34006 | 1171 | 893 | 0.424 | 0.361 |
| bnl2 | 0.91 | 2.82 | 59437 | 56705 | 3860 | 3420 | 0.957 | 0.889 |
| cycle | 0.93 | 1.80 | 54682 | 39073 | 2004 | 920 | 0.565 | 0.305 |
| d2q06c | 1.89 | 5.74 | 135960 | 91614 | 11327 | 4752 | 2.693 | 1.308 |
| degen3 | 20.77 | 13.33 | 119403 | 115681 | 7958 | 7403 | 2.312 | 2.198 |
| dfl001 | 37.40 | 552.44 | 1632101 | 1445468 | 711739 | 547005 | 160.471 | 129.905 |
| greenbea | 2.21 | 2.11 | 47014 | 45507 | 907 | 842 | 0.379 | 0.341 |
| grow22 | 0.21 | 0.51 | 8618 | 8590 | 157 | 156 | 0.064 | 0.055 |
| maros-r7 | 6.70 | 47.49 | 510148 | 511930 | 70445 | 72568 | 15.730 | 15.945 |
| pilot | 5.67 | 25.18 | 191704 | 172264 | 24416 | 18956 | 5.704 | 4.731 |
| pilot87 | 19.27 | 110.71 | 423656 | 389787 | 88504 | 75791 | 20.725 | 18.138 |
| pilot-we | 0.29 | 0.58 | 14904 | 13887 | 350 | 292 | 0.124 | 0.100 |

Figure 1: Sparsity pattern with the MDO (left) and MFO (right) on problem *cycle*



on problems with "hard" structures (e.g.: `cycle, dfl001`) MFO may be more efficient. Figure 1 shows the sparsity patterns of the Cholesky factors obtained by the minimum degree and minimum local fill–in orderings for the problem `cycle`, on which the largest difference between the two heuristics has been observed.

We have to be careful when giving final conclusions. An additional difficulty comes with the fact that the numerical factorization depends very much on the hardware and, in particular, on the ratio of the performance of integer and floating point operations on a given machine. We shall address this problem in more detail in Section 4.4. Here we only conclude that the minimum degree ordering performs sufficiently well to be a default option in any IPM implementation. In some cases, however, when very difficult problems are solved or a sequence of problems with the same sparsity patterns is solved, the more involved analysis of the minimum local fill-in ordering may pay off.

### Disadvantages of the normal equations approach

The normal equations approach shows a uniformly good performance when applied to the solution of the majority of all linear programs. Unfortunately, it suffers from two drawbacks.

Normal equations behave badly whenever the primal linear program contains free variables. To transform the problem to the standard form (1), any free variable has to be replaced with a difference of two nonnegative variables: $x_F = x^+ - x^-$. The presence of logarithmic terms in the objective function causes very fast growth of both split brothers. Although their difference may be kept relatively close to the optimal value of $x_F$, both $x^+$ and $x^-$ tend to infinity. This results in a serious loss of accuracy in (26). A remedy used in many IPM implementations is to prevent excessive growth of $x^+$ and $x^-$.

A more serious drawback of the normal equations approach is that it suffers dramatically from the presence of dense columns in $A$. The reason is that a dense column in $A$ with $p$ nonzero elements creates a dense window of size $p \times p$ in the $AD^2A^T$ matrix (subject to its symmetric row and column permutation). Assume that

$$A = [A_1 \; A_2], \tag{28}$$

where $A_1 \in \mathcal{R}^{m \times n-k}$ and $A_2 \in \mathcal{R}^{m \times k}$ are matrices built of sparse and dense columns, respectively. Several techniques have been proposed to treat the $A_2$ part separately.

The simplest one, due to Birge et al. [8] makes a choice between the factorizations of $AA^T$ and $A^TA$ matrices. The latter factorization easily accommodates dense columns of $A$ (dense rows of $A^T$). The approach clearly fails when $A$ contains both dense columns and dense rows.

Another possibility is the *column splitting* technique [35, 82]. It cuts a long column into shorter pieces, introducing additional linking constraints. Unfortunately, it works satisfactorily only for a small number of dense columns [37].

The most popular way of treating dense columns within the normal equations approach employs the *Schur complement mechanism.* It is based on (28) and an explicit decomposition of the matrix

$$AD^2A^T = A_1D_1^2A_1^T \; + \; A_2D_2^2A_2^T, \tag{29}$$

into a presumably sparse part $A_1D_1^2A_1^T$ and a significantly denser symmetric rank-$k$ update of it. A Cholesky decomposition is then computed for the "sparse" part and the dense rank-$k$ update is handled via the Sherman-Morrison-Woodbury formula. This method is not guaranteed to work correctly because the sparse part may be rank deficient (clearly, a full row rank assumption on $A$ does not guarantee that $A_1$ has full row rank). Whenever this happens, the Cholesky decomposition of $A_1D_1^2A_1^T$ does not exist and the Sherman-Morrison-Woodbury update is not well defined. Therefore in a practical implementation a small diagonal regularization term is added to $A_1D_1^2A_1^T$ such that the decomposition exists. The method usually works satisfactorily for a small number of dense columns.

Recently, Andersen [5] proposed a remedy to the rank deficiency arising in the Schur complement mechanism. His approach employs an old technique due to Stewart [74]. The technique corrects all unacceptably small pivots during the Cholesky factorization by adding a regularizing diagonal term to them. Consequently, instead of computing the decomposition of $A_1D_1^2A_1^T$, it computes a decomposition of another matrix $A_1D_1^2A_1^T + \sigma EE^T$, where $\sigma$ is a regularizing term and $E$ is a matrix built of unit columns with nonzeros appearing in rows corresponding to corrected pivots. Once such a stable decomposition is obtained

$$A_1D_1^2A_1^T + \sigma EE^T = L\Lambda L^T, \tag{30}$$

it is used as a stable "working basis" in the Sherman-Morrison-Woodbury update to compute

$$(A_1 D_1^2 A_1^T)^{-1} = (L \Lambda L^T \ - \ \sigma E E^T)^{-1}.$$

Stewart's technique is attractive, of course, only for a small rank deficiency of $A_1 D_1^2 A_1^T$.

Andersen [5] observed that the rank deficiency of $A_1 D_1^2 A_1^T$ cannot exceed $k$, the number of columns handled separately. His method consists of correcting too small pivots in the factorization of $A_1 D_1^2 A_1^T$ by computing the following (stable) Cholesky decomposition

$$A_1 D_1^2 A_1^T + E E^T = L_1 \Lambda_1 L_1^T.$$

Next, this factorization is employed in the Schur complement mechanism to compute

$$(A D^2 A^T)^{-1} = (L_1 \Lambda_1 L_1^T \ + \ (A_2 D_2^2 A_2^T - E E^T))^{-1}.$$

Summing up, it is possible to overcome the most important drawback of the normal equations approach, i.e. to handle dense columns in it. However, there still remains a question about the heuristic to choose the columns that should be treated separately. A trivial selection rule based on the number of nonzero elements in a column does not identify all "hard" columns; we shall discuss this issue in the next section.

Recall that the Schur complement mechanism is efficient if the number of dense columns in the constraint matrix is not excessive. This motivated several researchers to pay special attention to the augmented system form of the Newton equations which allows more freedom in the pivot choice.

## 4.2 The augmented system approach

The augmented system approach is an old and well understood technique to solve a least squares problem [6, 7, 11, 19]. It consists in the application of the Bunch-Parlett [13] factorization to the symmetric indefinite matrix

$$\begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix} = L \Lambda L^T, \tag{31}$$

where $\Lambda$ is an indefinite block diagonal matrix with $1 \times 1$ and $2 \times 2$ blocks.

In contrast to the normal equations approach in which the analysis and factorization phases are separated, the factorization (31) is computed dynamically. This means that the choice of pivot is concerned with both the sparsity and stability of the triangular factor. It is obvious that, due to the careful choice of stable pivots, this factorization must be at least as stable as the one of the normal equations. On the other hand, due to the greater freedom in the choice of the pivot order, the augmented system factorization may produce a significantly sparser factor than that of the normal equations. Indeed the latter is a special case of (31) in which the first $n$ pivots are chosen from the $D^2$ part regardless their stability properties and without any concern about the fill-in they produce.

Advantageous stability properties of the augmented system approach motivated several researchers to incorporate it into their IPM codes [20, 26, 56, 81, 83]. Soon afterwards, other advantages of this approach, namely, an ease of handling free LP variables and dense columns in $A$ and an ability of its easy extension to handling quadratic programming problems were recognized [83, 60, 14].

The success of the augmented system factorization depends highly on the efficiency of the pivot selection rule. Additionally, to save on the expensive analysis phase, the pivot order is reused in subsequent IPM iterations and only occasionally updated when the numerical properties of the Newton equation matrix has changed considerably. Mehrotra's implementation [26, 60], for example, is based on the Bunch-Parlett factorization [13] and on the use of the generalized Markowitz [55] count of type (27) for $2 \times 2$ pivots.

On the other hand, it has been shown in [66] that the $1 \times 1$ pivot scheme is always valid when computing the symmetric factorization of the augmented matrix, and if a valid pivot order is computed for a certain $D^2$, it will in theory be valid for arbitrary $D^2$ matrices occurring during the interior point iterations. However, this ordering might be numerically unstable.

A popular way of the pivot selection rule is detecting "dense" columns and pivoting first in the diagonal positions of $D^{-2}$ in the augmented matrix falling outside of them. A difficulty arises, however, with the choice of a threshold density used to group columns of $A$ into the sparse and the dense parts in (28). A fixed threshold value approach works well only in a case when dense columns are easily identifiable, i.e. when the number of nonzeros in each of them exceeds significantly the average number of entries in sparse columns [83]. Whenever more complicated sparsity structure appears in $A$, a more sophisticated heuristic is needed. Maros and Mészáros [56] give a detailed analysis of this issue that we shall present below.

Instead of (28), he considers the following partition of the LP constraint matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{32}$$

where $A_{11}$ is supposed to be very sparse and additionally it is assumed to create a sparse adjacency structure $A_{11}A_{11}^T$, $A_{12}$ is a presumably small set of "difficult" columns, e.g., dense columns or columns referring to free variables, and $[A_{21}\ A_{22}]$ is a set of "difficult" rows. An efficient heuristic to find such a partition is given in [56].

Once the partition (32) is determined, (25) becomes

$$\begin{bmatrix} -D_1^{-2} & & A_{11}^T & A_{21}^T \\ & -D_2^{-2} & A_{12}^T & A_{22}^T \\ A_{11} & A_{12} & & \\ A_{21} & A_{22} & & \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta y_1 \\ \Delta y_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ h_1 \\ h_2 \end{bmatrix}.$$

The analysis of this system shows immediately which block can be inexpensively pivoted out and which one should be delayed as much as possible. Elimination of $D_1^{-2}$ causes very limited fill-in and reduces the matrix to

$$\begin{bmatrix} -D_2^{-2} & A_{12}^T & A_{22}^T \\ A_{12} & A_{11}D_1^2A_{11}^T & A_{11}D_1^2A_{21}^T \\ A_{22} & A_{21}D_1^2A_{11}^T & A_{21}D_1^2A_{21}^T \end{bmatrix}. \tag{33}$$

The elimination of the $D_2^{-2}$ block should be delayed after all attractive pivot candidates from $A_{11}D_1^2A_{11}^T$ and $A_{21}D_1^2A_{21}^T$ blocks are exploited. The normal equations approach makes no such a distinction and pivots out both $D_1^{-2}$ and $D_2^{-2}$ blocks.

It is worth to note a close relationship of the approach of [56] and the Schur complement mechanism applied to handle the block of "difficult" columns in $A$. Observe that the normal equations

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} D_1^2 & \\ & D_2^2 \end{bmatrix} \begin{bmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{bmatrix} \begin{bmatrix} \Delta y_1 \\ \Delta y_2 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix},$$

Table 2: Comparison of normal equations (NE) and augmented system (AS) factorizations

| Name | Densest column | Analysis time NE | Analysis time AS | Nonzeros NE | Nonzeros AS | Flops in thousand NE | Flops in thousand AS | Fact. time NE | Fact. time AS |
|---|---|---|---|---|---|---|---|---|---|
| aircraft | 751 | 115.2 | 0.97 | 1437398 | 20317 | 361174 | 37 | 79.19 | 0.122 |
| fit1p | 627 | 14.22 | 0.33 | 206097 | 10120 | 42920 | 63 | 9.281 | 0.058 |
| fit2p | 3000 | - | 1.73 | - | 50583 | - | 266 | - | 0.328 |
| ganges | 13 | 0.58 | 0.98 | 35076 | 23555 | 770 | 316 | 0.252 | 0.122 |
| pilot4 | 27 | 0.64 | 0.58 | 18851 | 14153 | 488 | 265 | 0.146 | 0.082 |
| stair | 34 | 0.44 | 0.48 | 17990 | 11693 | 461 | 188 | 0.129 | 0.062 |
| 25fv47 | 21 | 0.93 | 2.77 | 43202 | 43569 | 1282 | 1297 | 0.363 | 0.412 |
| 80bau3b | 11 | 2.02 | 3.38 | 57202 | 57683 | 1171 | 1181 | 0.476 | 0.487 |
| d2q06c | 34 | 5.34 | 22.30 | 167318 | 178763 | 11328 | 14480 | 2.85 | 3.604 |

can be replaced with the following system

$$\begin{bmatrix} A_{11}D_1^2A_{11}^T & A_{11}D_1^2A_{21}^T & A_{12}D_2 \\ A_{21}D_1^2A_{11}^T & A_{21}D_1^2A_{21}^T & A_{22}D_2 \\ D_2A_{12}^T & D_2A_{22}^T & -I \end{bmatrix} \begin{bmatrix} \Delta y_1 \\ \Delta y_2 \\ \Delta v \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ 0 \end{bmatrix}, \qquad (34)$$

in which all "difficult" columns are handled as a symmetric rank-$k$ update of an "easy" part (cf. (29))

$$\begin{bmatrix} A_{11} \\ A_{12} \end{bmatrix} D_1^2 \begin{bmatrix} A_{11}^T & A_{12}^T \end{bmatrix} + \begin{bmatrix} A_{12}D_2 \\ A_{22}D_2 \end{bmatrix} \begin{bmatrix} D_2A_{12}^T & D_2A_{22}^T \end{bmatrix}.$$

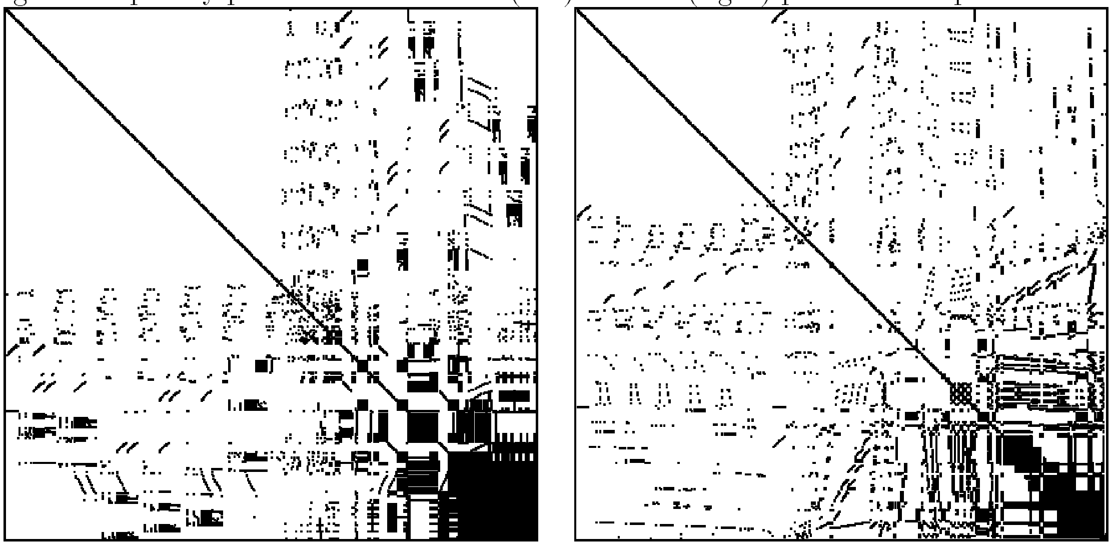
It is easy to verify that the matrix involved in the system (34) has exactly the same sparsity pattern (subject to symmetric row and column permutations) as that in (33).

### Normal equations versus the augmented system

Table 2 compares the efficiency of the normal equations (NE) and the augmented system (AS) approaches. We cluster our test problems into three groups. The first group contains problems with dense columns (`aircraft, fit1p, fit2p`). In the second group we collect some problems without dense columns, but with a "prejudicial" nonzero pattern for the normal equations (`ganges, pilot4, stair`). The last group contains problems without any advantageous structure for the augmented system. The first two columns of Table 2 contain the name of the problem and the number of nonzeros in the densest column. The following two columns show the setup time (in seconds) for the two competing approaches. Note that the setup time includes not only the generation of the pivot order and the sparsity pattern analysis but also the time of one numerical factorization. Columns 5 and 6 contain the number of nonzeros in the factorization (in a case of the NE, this corresponds to the sum of nonzeros in the Cholesky factor of (26) and nonzeros in $A$). Columns 7 and 8 contain the number of flops (in thousands) required by one factorization for the two approaches compared. The last two columns show the average times (in seconds) to compute one factorization during the algorithm. All results are obtained on a SUN Sparc-10 workstation.

The results of Table 2 obtained for problems with dense columns show an unquestionable advantage of the augmented system over a trivial implementation of the normal equations in which dense columns are not handled separately. Our 64 Mbyte workstation was unable to store the lower triangular part of a $3000 \times 3000$ totally dense matrix that resulted from the normal

Figure 2: Sparsity patterns with the NE (left) and AS (right) pivot rule on problem *stair*



equations approach applied to the problem `fit2p`. In contrast, the augmented system produced a very sparse factorization in this case. For our second group of problems, the performance of the augmented system is also much better. Finally, for our third group of problems, the much lower setup cost of the normal equations made the augmented system approach disadvantageous.

Figure 2 gives a bit of insight into the sparsity patterns generated for the problem `stair`. It displays the factored augmented matrices for the two competitive approaches.

Based on the previous examples, we find that both methods are important for a computational practice. It would be advantageous to have both of them implemented as well as to have an analyzer that is able to determine which of them should be used [56].

### 4.3 The numerical factorization

In this section we shall demonstrate several issues of the implementation of the numerical factorization step. We use the normal equations approach because of notational convenience, but the methods used here can be applied in a similar way to the general symmetric decomposition of the augmented system.

Let $M = AD^2A^T$ and consider its Cholesky factorization $M = L\Lambda L^T$, where $L$ is a lower triangular matrix and $\Lambda$ is a diagonal matrix. We note that the solution of a sparse symmetric system of linear equations is a very important problem in scientific computing. Therefore, it is a well developed area both in theory and in the computational practice. The basic formulae for computing the column $j$ of $L$ (denoted by $L_j$) and the pivot $\Lambda_{jj}$ are:

$$\Lambda_{jj} = M_{jj} - \sum_{k=1}^{j-1} l_{jk}^2, \tag{35}$$

$$L_j = \frac{1}{\Lambda_{jj}} \left( M_j - \sum_{k=1}^{j-1} (\Lambda_{kk} l_{jk}) L_k \right). \tag{36}$$

Several approaches have been developed to compute the factorization. They exploit sparsity in an efficient way and use different techniques of storage management in the computations.

George and Liu [29] demonstrate how these calculations can be organized either by rows or by columns.

During the *row*–Cholesky factorization the rows of the Cholesky factor $L$ are computed one by one. This approach is called the *bordering* method. Several enhancements of it can be found in [29, 50].

An alternative approach is the *column*–Cholesky factorization in which the columns of $L$ are computed one by one. This is the most commonly used form; its efficient implementations can be found, for example, in the Yale Sparse Matrix Package [21] and Waterloo SPARSPAK [29]. This method is also called *left looking* factorization, because for computing the column $L_j$ the information from the left part of the factor (i.e. the columns prior to $L_j$) is used in the computations. Its implementation uses dynamic linked lists to identify the 'left' columns when updating the pivot column, and a double precision work array to accumulate the column modifications and to resolve the nonzero matching between different columns.

The third approach is the *submatrix*–Cholesky factorization, also referred to as the *right looking* factorization. In this approach, once a column $L_j$ has been computed, it immediately generates all contributions to subsequent columns, i.e. to columns to the right of it in the matrix. The matching of nonzeros during the transformations with this approach is not a trivial problem; several solutions have been found for its efficient implementation [19, 72]. The interest in this approach has increased in the past few years because of its ability to better exploit high performance architectures and the memory hierarchy.

We shall present a few of the most important techniques that increase the efficiency of the numerical factorization step in interior point methods. These techniques come from parallel and vector computations and the common trick is the use of the matrix–vector operations in 'dense' mode to reduce the overhead of the sparse computations.

### Dense window

The most straightforward improvement of the factorization is exploitation of the *dense window*. In practice, the triangular factors become completely dense in the last steps of the Cholesky factorization (see e.g. Figures 1 and 2). The last partition of columns can be handled as a dense matrix when those columns are factored. In this way the overhead of doing sparse computations is avoided. It might also be advantageous to include some almost dense columns into the dense window (see, e.g., [19]).

### Supernodes

The dense window technique can be generalized using the following observation. Due to the way the Cholesky decomposition works, some blocks of columns in $L$ tend to have the same sparsity pattern below the diagonal. Such a block of columns is called a *supernode* and it can be treated as a dense submatrix. The supernode terminology comes from the elimination graph representation of the Cholesky decomposition [29]. There exist two different types of supernodes:

they are presented in the figures below.

$$
\begin{array}{cc}
\text{Type 1 supernode} & \text{Type 2 supernode} \\
\begin{bmatrix}
* & & \\
* & * & \\
* & * & * \\
* & * & * \\
 & & \\
* & * & * \\
* & * & *
\end{bmatrix}
&
\begin{bmatrix}
* & & \\
 & * & \\
 & & * \\
* & * & * \\
 & & \\
* & * & * \\
* & * & *
\end{bmatrix}
\end{array}
$$

Both types of supernodes are exploited in a similar manner within the numerical factorization step. Analogously to the dense window technique, the use of supernodes increases the portion of flops that use dense matrix–vector transformations to save on indirect addressing and memory references. The following operations take advantage of the presence of supernodes:

**(i)** whenever column $j$ is a member of a supernode, the operation of building $L_j$ is simplified (operations on other members of the supernode are done in dense mode);

**(ii)** whenever column $j$ is not a member of a supernode (but it depends on a set of columns that belong to a supernode), then a temporary work array is used to accumulate the contribution of the whole supernode before this contribution is added to $L_j$.

It is advisable to impose a lower bound on the size of supernodes since the extra work in step *(ii)* does not pay off in the case of too small supernodes. Another suggestion is the use of an upper bound on the number of nonzeros in each supernode to better exploit the cache memory on several computer architectures [52]. The effect of the supernodal methods is highly hardware–dependent and several results can be found in the literature: the efficiency of the supernodal decomposition on the shared–memory multiprocessors is discussed by Esmond and Peyton [69], the exploitation of the cache memory on high–performance workstations is studied by Rothberg and Gupta [72] in the framework of the right looking factorization while the case of the left looking factorization was investigated by Mészáros [64].

**Block Cholesky factorization**

Another possibility to use dense computations is the partitioning of $L$ into smaller, presumably dense blocks. We try to divide $L$ into block diagonal 'supernodal' submatrices. This technique is very effective in some cases, because the typical Cholesky factor contains many such blocks (the largest of them is usually the dense window located at the bottom of the matrix). Consider the following matrix:

$$AD^2A^T = M = \begin{bmatrix} B_{11} & B_{21}^T \\ B_{21} & B_{22} \end{bmatrix},$$

with an additional simplifying assumption that the blocks $L_{11}$ and $L_{22}$ of the Cholesky factor define supernodes. The Cholesky factorization of this matrix can be computed in the following steps:

1. Factorize $L_{11}\Lambda_{11}L_{11}^T = B_{11}$.
2. Update $L_{21} = B_{21}(L_{11}^{-1})^T$.

3. Update $\hat{B}_{22} = B_{22} - L_{21}\Lambda_{11}L_{21}^T$.

4. Factorize $L_{22}\Lambda_{22}L_{22}^T = \hat{B}_{22}$.

The advantage is that steps 1, 2, and 4 can be performed in dense mode, resulting in a very efficient implementation on high performance computers.

### Loop unrolling

Dense computations can be further specialized to exploit a loop unrolling technique. The typical inner loop of the factorization adds a multiple of a column to another one. Let $a$ be the target column, $b$ the source column and $c$ the multiplier. If we assume that $c$ is kept in a single register, then the steps performed by the computer to execute one transformation $a \leftarrow a + cb$ can be written as follows:

1. read $a(i)$ from the memory,
2. read $b(i)$ from the memory,
3. compute $a(i) + cb(i)$,
4. store the result in the memory.

Consequently, three memory references (steps 1, 2, and 4) are associated with only one arithmetical operation (step 3). During the factorization, multiple column modifications are performed on a single column, which opens the possibility to unroll the loop over the column transformations. Let $a$ be the target column, $b, c, d, e, f$ and $g$ be the source columns, and $h(1), \ldots, h(6)$ their scalar multipliers, respectively. A 6–step loop unrolling technique consists of the following transformation:

$$a \leftarrow a + h(1)b + h(2)c + h(3)d + h(4)e + h(5)f + h(6)g.$$

An execution of this transformation needs only eight memory references and six arithmetical operations (multiplications). Hence, ten memory references have been saved compared with an execution of six elementary flops that do not exploit loop unrolling. This technique brings considerable time savings on all computer architectures although the savings may vary significantly on different computers.

### Numerical examples

To give the reader some idea about the efficiency of all techniques discussed by now (i.e. dense window, supernodes and loop unrolling), we show the computational results of their application on a small set of test problems for one widely used computer architecture, namely, a SUN Sparc-10 workstation.

In Table 3 we compare the times (in seconds) of computing one decomposition with a standard left looking factorization, *T(ll)*, the one using dense window technique, *T(dw)*, supernodal factorization without loop unrolling, *T(sn1)*, and, finally, supernodal factorizations with 2-, 4-, and 6-step loop unrolling, *T(sn2)*, *T(sn4)* and *T(sn6)*, respectively.

To cover a possibly wide set of LP problems, we have chosen 6 test examples with very different characteristics. Problems `aircraft` and `fit2p` have extremely sparse factorization with

Table 3: Comparison of different techniques of the factorization

| Name | T(ll) | T(dw) | T(sn1) | T(sn2) | T(sn4) | T(sn6) |
|---|---|---|---|---|---|---|
| aircraft | 0.131 | 0.143 | 0.142 | 0.135 | 0.133 | 0.129 |
| fit2p | 0.344 | 0.324 | 0.343 | 0.347 | 0.334 | 0.328 |
| 80bau3b | 0.504 | 0.464 | 0.528 | 0.480 | 0.428 | 0.424 |
| 25fv47 | 0.471 | 0.437 | 0.463 | 0.404 | 0.349 | 0.346 |
| dfl001 | 434.964 | 349.539 | 219.333 | 191.068 | 175.221 | 160.594 |
| maros-r7 | 28.901 | 26.285 | 22.308 | 18.422 | 16.416 | 15.734 |

the augmented system (cf. Table 2). Problems `80bau3b` and `25fv47` are "usual" sparse problems, and `maros-r7` and `dfl001` are examples of very dense ones. In the case of sparse problems, these techniques have very little influence on the factorization times. On usual "sparse" problems, the dense window method is unequivocally superior to the standard left looking method but the savings resulting from the use of supernodes and loop unrolling are not evident. The 4-step loop unrolling gives a better execution time, but the effect of the 6-step loop unrolling is negligible. Finally, for dense problems, the superiority of the simple supernodal method over the use of dense window is unquestionably; moreover the computation times monotonically decrease with the degree of the loop unrolling.

### 4.4 Hardware–dependent aspects

As noted previously, modern computer architectures have many different characteristics that influence the choice of the algorithm to be applied. The most important of them are:

- cache memory,
- pipelining,
- vectorization,
- superscalar capabilities.

The simple choice of the "best" algorithm is usually impossible without extensive computational tests. The reader can find a good discussion of these issues (as well as many numerical examples) in [52]. Let us collect some general suggestions.

There is a choice between two ordering methods: the minimum degree and the minimum local fill–in one. The hint is the ratio of the cost of integer (or logical) and floating point operations. If the latter are executed fast compared with the former, then there is little chance that the savings in numerical factorization can compensate excessive effort during the analysis phase. In this case, the faster minimum degree ordering seems more appropriate. In other cases, e.g., for standard "low–cost" workstations, the minimum local fill–in ordering may become an attractive alternative.

In the numerical factorization phase the two most commonly used methods are the right looking and left looking algorithms. The right looking factorization exploits the cache memory better, because the supernodes enter into the cache only once during the factorization, while in the case of the left looking factorization a supernode enters cache memory many times. However, the right looking factorization requires an additional indirect addressing. As it can be presumed, the criteria for choosing the numerical factorization algorithm must be based on the investigation of the cache memory (its size, the time of bringing information into it, etc.).

### 4.5 Handling of rank deficiency and instability

The standard assumption in the theory of interior point methods is that the LP constraint matrix $A$ has full row rank. In practice, this property may not be satisfied. It is possible to determine the maximum set of independent rows using Gaussian Elimination [2]. The computational cost of such an operation is relatively low in most of the cases. On the other hand, IPMs that use a starting point similar to (11) can benefit from the additional Cholesky factorization of $AD^2A^T$ with a well conditioned $D^2$ to detect linearly dependent rows. Whenever a pivot in the factorization falls below a predetermined tolerance, i.e. $\Lambda_{ii} < \epsilon$, row $i$ can be dropped from the LP model. Although the latter approach is, in general, less reliable than the specialized Gaussian Elimination of $A$, its application does not need any additional computational effort as it only exploits the factorization that anyway has to be computed. The practice shows that this approach solves the problem of dependent rows as well.

Even if we manage to satisfy the full row rank property at the beginning of the optimization proces, then during the solution process matrix $AD$ may become "numerically" rank deficient. This is often the case due to the presence of primal degeneracy. Consequently, IPM implementations have to be able to deal with rank deficient matrices $AD^2A^T$.

For feasible and bounded LPs, the negative influence of the ill-conditioning of $AD^2A^T$ on the accuracy of the solution of the normal equations is surprisingly small. Stewart [75] gives a nice explanation to this common experience, derived from an analysis of the properties of the right hand side vector in (26). Stewart's result does not apply to the case when the LP problem is infeasible. In practice, this case usually manifests in a serious loss of accuracy when solving the Newton equations.

As mentioned before, the numerical difficulties usually appear close to the optimal solution, especially in the presence of primal degeneracy. There exist several ways to overcome them. These techniques are not always mathematically elegant and, additionally, they are often treated as the most precious know-how that is not revealed by IPM specialists. Below we present some suggestions how instability problems can be overcome and we end this section with a detailed presentation of the technique to control accuracy applied in a public domain IPM code (available trough Netlib) [37, 36, 38].

One possibility to handle nearly rank deficient factorizations is to remove presumably dependent rows only "virtually" by setting the nonzeros of $L_i$ to zero and $\Lambda_{ii}$ to one. Note that it is much easier if an inverse of the diagonal matrix $\Lambda$ is stored instead of $\Lambda$ itself (in this case it suffices to set $\Lambda_{ii}^{-1} = 0$).

Another way is to add a small regularizing term $\epsilon I$ to the matrix $AD^2A^T$ before or during the factorization. This helps to complete the factorization step but needs special safeguards in the following solution steps.

The approach used in the public domain LP code [38] consists of several safeguard techniques. First of all, it uses a dynamically adjusted diagonal regularizing matrix $R \in \mathcal{R}^m$. Its elements $R_{ii}$ vary from nearly zero value added to acceptably stable pivots $\Lambda_{ii}$ to quite large regularizations $R_{ii} = 1$ added to unstable pivots $\Lambda_{ii}$. Consequently, instead of a decomposition of $AD^2A^T$, a (stable) factorization of a different regularized matrix $AD^2A^T + R$ is computed. It is very rare that $R$ contains more than a few large regularizing terms; they refer to those rows of $AD$ which are nearly linearly dependent.

An important issue is to take $R$ into account in all the following computations (i.e. in solves for direction). Observe that the form of the decomposed matrix could have been obtained from the following perturbed augmented system

$$\begin{bmatrix} -D^{-2} & A^T \\ A & R \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_R \\ h_R \end{bmatrix}. \tag{37}$$

Note that it is the Newton equations system corresponding to the following quadratic programming problem (closely related to the dual LP problem (2))

$$\begin{array}{ll} \text{maximize} & b^T y - u^T w + (y - y_0)^T R(y - y_0) \\ \text{subject to} & A^T y - w + z = c, \\ & z, w \geq 0, \end{array} \tag{38}$$

in which $y_0$ is some reference point, e.g., current iterate $y$. The right hand side vectors $r_R$ and $h_R$ in (37) are derived from the first order optimality conditions for the barrier problem associated with (38)

$$\begin{aligned} r_R &= \xi_c - X^{-1}(\mu e - XZe) + S^{-1}(\mu e - SWe) - S^{-1}W\xi_u, \\ h_R &= \xi_b + \Delta(y - y_0). \end{aligned}$$

Note that they become identical to (9) if we take a particular reference point $y_0 = y$. The regularization technique can be interpreted as the use of quadratic penalty for the changes of those dual variables $y_i$ for which presumably unstable pivots were computed. Computational experience shows that it well prevents the propagation of round-off errors.

Apart from the quadratic regularization technique mentioned above, the LP code of [38] uses extensively iterative refinement process to improve the accuracy of the Newton direction. Iterative refinement technique is always applied to the augmented system formulation of the Newton equations system although the direction is computed via their reduced, normal equations form (see, e.g., [6]).

# 5 Presolve

The previous section was concerned with the efficiency of solving the Newton equation system using advanced numerical linear algebra. Another way to improve the efficiency of solving the Newton equation system is to reduce its size and make the system sparser. This aim can be achieved by analyzing the LP problem and remove redundancies. In practice, almost all large-scale LP problems contain redundancies. There are several reasons to this. First of all, the model formulators, tend to chose a formulation that makes it easy to understand and to modify the model. This often leads to an introduction of superfluous variables and redundant constraints.

Unfortunately, it is impossible to remove all redundancies in a large-scale LP problem manually. Therefore a presolve analysis aims at improving problem formulation. More precisely, its goals might be defined as follows:

- reduce the size of the LP problem as much as possible;
- reformulate the model to the most suitable form for a solver.

The use of a presolve phase is an old idea, see, e.g., Brearley et al. [12]; its role was acknowledged already in simplex type optimizers. The simplex method for LP works with sparse submatrices of $A$ (bases) [78] while any IPM needs an inversion of a considerably denser $AA^T$

matrix. Consequently, the potential savings resulting from an initial problem reduction may be larger in IPM implementations. This is the reason why the presolve analysis has recently enjoyed great attention [1, 51, 3, 37, 2, 9, 54, 77]. An additional important motivation for its use is that large LP problems are solved routinely nowadays and the amount of redundancy is increasing with the size of the problem.

## 5.1 The reduction methods

In general, finding all redundancies in an LP problem is computationally too expensive. Therefore all presolve procedures use an arsenal of simple inspection techniques to detect obvious forms of redundancies. These techniques are applied repeatedly until the problem cannot be reduced any further. Below, we briefly present the most common reduction procedures. Further details about the presolve phase are presented in [3, 37]. The following reduction techniques are used:

1. Empty rows and columns are removed.
2. A fixed variable ($0 = u_j$) can be substituted out of the problem.
3. A row singleton defines a simple variable bound; after an appropriate bound modification the row can be removed.
4. Lower and upper limits for every constraint $i$ are determined
$$\underline{b_i} = \sum_{\{j:a_{ij}<0\}} a_{ij}u_j, \quad \text{and} \quad \overline{b_i} = \sum_{\{j:a_{ij}>0\}} a_{ij}u_j, \tag{39}$$
that clearly satisfy
$$\underline{b_i} \leq \sum_j a_{ij}x_j \leq \overline{b_i}. \tag{40}$$
Observe that due to the nonnegativity of $x$, the limits $\underline{b_i}$ and $\overline{b_i}$ are nonpositive and nonnegative, respectively. If the inequalities (40) are at least as tight as the original (inequality type) LP constraint, then the constraint $i$ is *redundant*. If one of them contradicts the LP constraint, then the problem is infeasible. Finally, in some special cases (e.g.: "less than or equal to" row with $\underline{b_i} = b_i$, "greater than or equal to" row with $\overline{b_i} = b_i$, or equality type row for which $b_i$ equals to one of the limits $\underline{b_i}$ or $\overline{b_i}$), the LP constraint becomes a *forcing* one. This means that the only way to satisfy the constraint is to fix all variables that appear in it on their appropriate bounds.
5. Constraint limits (39) are used to generate implied variable bounds. (Note, that LP variables were transformed to the standard form $0 \leq x \leq u$, before). This technique makes use of the original form of an LP constraint (i.e. its form before a slack variable has been added to it to transform it to the "standard" equality row of (1)). Assume, for example, that a nonredundant "less than or equal to" type constraint is given, i.e.
$$\underline{b_i} < \sum_j a_{ij}x_j \leq b_i.$$
Then
$$\begin{aligned} \forall k: \quad & a_{ik} > 0 \qquad \underline{b_i} + a_{ik}x_k \leq \sum_j a_{ij}x_j \leq b_i, \\ \text{and} \quad \forall k: \quad & a_{ik} < 0 \qquad \underline{b_i} + a_{ik}(x_k - u_k) \leq \sum_j a_{ij}x_j \leq b_i, \end{aligned}$$

and new implied bounds are given for all variables in row $i$ by

$$\begin{aligned} x_k \leq u_k^{'} &= (b_i - \underline{b_i})/a_{ik} \quad \text{for all} \quad k: \quad a_{ik} > 0, \\ x_k \geq l_k^{'} &= u_k + (b_i - \underline{b_i})/a_{ik} \quad \text{for all} \quad k: \quad a_{ik} < 0. \end{aligned}$$

If these bounds are tighter than the original ones, then variable bounds are improved. Note, that this technique is particularly useful when it imposes finite bounds on free variables. Free variables do not, in such a case, have to be split and represented as the difference of two nonnegative variables.

6. Variable $j$ is a free column singleton, if

$$\exists k : a_{jk} \neq 0 \wedge (a_{ij} = 0, \forall i \neq k) \wedge l_j = -\infty \wedge u_j = +\infty.$$

In this case variable $x_j$ can be substituted out of the problem: variable $x_j$ disappears and the $k$th constraint is eliminated. The same technique can be applied to eliminate a singleton *implied free variable*, i.e. a variable for which implied bounds (generated by the technique of point 5) are at least as tight as the original bounds.

7. Nonnegative unbounded variables ($0 \leq x \leq +\infty$) referring to singleton columns are used to generate bounds on dual variables $y$. Namely, if the variable $j$ refers to a singleton column with an entry $a_{ij}$ and $u_j = +\infty$ (i.e. $w_j = 0$), then the dual constraint (2) becomes an inequality

$$a_{ij} y_i \leq c_j.$$

This inequality can be solved and, depending on the sign of $a_{ij}$, produces a lower or upper bound on $y_i$.

These bounds on the dual variables are used to generate lower and upper limits for all dual constraints (a technique similar to that of point 4 is used). The limits are then used to determine the variables reduced costs. Whenever the reduced cost is proved to be strictly positive or strictly negative, the corresponding variable is fixed on an appropriate bound and eliminated from the problem.

8. Dual constraint limits (obtained with a technique of point 7) are used to generate new implied bounds on the dual variables. A technique similar to that of point 5 is applied. Implied bounds tighter than the original ones replace old bounds and open the possibility to eliminate more variables with the technique of point 7.

## 5.2 Detecting redundancy and improving sparsity in $A$

The presolve techniques described in the previous section involve a considerable amount of arithmetical operations. The techniques discussed in this section are based mainly on the sparsity pattern analysis. We list them below.

1. Removing duplicate constraints. Two constraints are said to be duplicate if they are identical up to a scalar multiplier. One of the duplicate constraints is removed from the problem.

2. Removing linearly dependent constraints. The presence of more than a few linearly dependent rows in $A$ may lead to serious numerical problems in an interior-point methods, since it implies a rank deficiency in the Newton equation system. Subramanian et al. [76] and Andersen [2] report that in some cases the computational savings from removing the linearly dependent constraints are significant.

3. Removing duplicate columns. Two columns are said to be duplicate if they are identical up to a scalar multiplier. An example of duplicate columns are two non-negative split brothers used to replace a free variable.

   When discussing the disadvantages of the normal equations approach in Section 4.1, we have mentioned the negative consequences of the presence of split free variables. Sometimes it is possible to generate a finite implied bound on a free variable [37] and avoid the need of splitting it. Whenever possible, general duplicate variables are replaced with an aggregate variable (a linear combination of duplicates).

4. Improving sparsity of $A$. We look for a nonsingular matrix $M \in \mathcal{R}^{m \times m}$ such that the matrix $MA$ is as sparse as possible. Primal feasibility constraints can in such case be replaced with an equivalent formulation

$$MAx = Mb, \tag{41}$$

   much more suitable for a direct application of the interior point solver. Exact solution of this *Sparsity Problem* [15] is an NP–complete problem but efficient heuristics [1, 15, 37] usually produce satisfactory nonzero reductions in $A$. The algorithm of [37], for example, looks for such a row of $A$ that has a sparsity pattern being the subset of the sparsity pattern of other rows and uses it to pivot out nonzero elements from other rows.

## 5.3 Other types of reductions

The common feature of the previous presolve techniques is that they cannot increase the number of nonzeroes in the LP problem. It may in some cases be advantageous to allow a limited fill-in that results from the elimination of certain variables and constraints. We list these elimination techniques below.

1. Free and implied free variables can be eliminated not necessarily only in a case when they correspond to singleton columns (cf. Section 5.1, point 6) but also in a case when they correspond to denser columns. It should be noted, however, that this elimination technique has to be used carefully as it may introduce large amount of fill-in and, in particular, create dense columns. Hence, it requires additional sparsity structure analysis to be implemented properly [76].

2. Doubleton rows corresponding to equality type constraints can be used to pivot out one of the variables. This operation is clearly the opposite to splitting dense columns: it causes a concatenation of two shorter columns into a longer one but it may be advantageous if the length of the new column is not excessive [25].

The application of all presolve techniques described by now often results in impressive reductions of the initial LP formulation. Hopefully, the reduced problem obtained after the presolve analysis can be solved faster. Once its solution is found, the solution is used to recover the complete primal and dual solutions to the original problem. This phase is called a *postsolve analysis*; it has been discussed extensively in [3].

Table 4: Advantages of the presolve analysis.

| Name | ROWS | COLS | NZA | RROWS | RCOLS | RNZA | LROWS | LCOLS | LNZA |
|---|---|---|---|---|---|---|---|---|---|
| 80BAU3B | 2263 | 9799 | 29063 | 1960 | 8679 | 18969 | 1960 | 8679 | 18969 |
| aa3 | 826 | 8627 | 79433 | 763 | 8572 | 67571 | 690 | 8572 | 58604 |
| CRE-B | 9649 | 72447 | 328542 | 5324 | 31818 | 107603 | 5316 | 31818 | 107551 |
| KEN-13 | 28633 | 42659 | 139834 | 22525 | 36552 | 81168 | 22356 | 36552 | 79478 |
| NUG12 | 3193 | 8856 | 44244 | 3192 | 8856 | 38304 | 2794 | 8856 | 33528 |
| OSA-30 | 4351 | 100024 | 700160 | 4279 | 96119 | 262872 | 4279 | 96119 | 262872 |
| PDS-10 | 16559 | 48763 | 140063 | 15609 | 47729 | 103290 | 15598 | 47729 | 103169 |
| PILOT87 | 2031 | 4883 | 73804 | 1966 | 4592 | 70375 | 1966 | 4592 | 70375 |
| WOOD1P | 245 | 2594 | 70216 | 170 | 1717 | 44573 | 169 | 1717 | 44306 |
| Sum | 67750 | 298652 | 1605359 | 55788 | 244634 | 794725 | 55128 | 244634 | 778852 |

Table 5: Cholesky factors after presolve.

| Name | NZL | RNZL | LNZL |
|---|---|---|---|
| 80BAU3B | 40521 | 38308 | 38308 |
| aa3 | 204468 | 185961 | 152501 |
| CRE-B | 957052 | 246616 | 245903 |
| KEN-13 | 340070 | 269227 | 256213 |
| NUG12 | 2732346 | 2732346 | 1993043 |
| OSA-30 | 222863 | 14125 | 14125 |
| PDS-10 | 1729070 | 1615648 | 1615516 |
| PILOT87 | 423242 | 422195 | 422195 |
| WOOD1P | 18347 | 11658 | 11488 |

### 5.4 Numerical examples

In Table 4 we present some computational results reproduced from [2]. The columns ROWS, COLS and NZA show the number of rows, columns and nonzero elements in $A$, respectively. The following columns: RROWS, RCOLS and RNZA show the same numbers, but after presolve. Finally, LROWS, LCOLS and LNZA present the LP matrix statistics after presolve and after elimination of all linearly dependent rows. The results collected in Table 4 clearly advocate for the use of an involved presolve analysis, although they also show that there exist (rare in practice) almost irreducible problems.

The advantages of the presolve analysis become clearer if one compares the sparsity of the Cholesky factors obtained for the original and the reduced LP formulations. Table 5 reports the number of nonzeros in the Cholesky factor, NZL for all problems listed in Table 4. These numbers are given for the original problem formulation, the reduced one, and the final reduced form, in which linearly dependent rows have been eliminated.

## 6 Higher order extensions

The computationally most expensive step in any implementation of an IPM is the solution of the Newton equation system. This system has to be solved in each iteration and, as we have discussed in Section 4, it requires computing a symmetric factorization of one of the matrices (25) or (26) followed with a solve employing this factorization. Both in theory and in practice the factorization phase is computationally much more expensive than the solve phase. Therefore, we can allow to do several solves in each iteration if these solves help to reduce the total number of interior point iterations and therefore also the number of factorizations.

This is the main idea for using *high-order methods* which we shall discuss below. Their common feature is that they reuse the factorization of the Newton equations system in several solves with the objective to compute a "better" search direction. There exist several approaches of this type: they apply different schemes to compute the search direction. We shall review them briefly.

The first such approach was proposed by Karmarkar et al. [47] who constructed a parameterized representation of the (feasible) trajectory motivated from the use of differential equations.

Mehrotra's method [62, 61] builds a higher order Taylor approximation of the (infeasible) primal-dual central trajectory and pushes an iterate towards an optimum along such an approximation. The second order variant of this method proved very successful.

Another approach, due to Domich et al. [18] uses three independent directions and solves an auxiliary linear program in a three dimensional subspace to find a search direction.

The method of Sonnevend et al. [73] uses subspaces spanned by directions generated by higher order derivatives of the feasible central path, or earlier computed points of it as a predictor step. This is later followed by one (or more) centering steps to take the next iterate sufficiently close to the central path.

Hung and Ye [42] studied theoretically higher order predictor–corrector techniques incorporated in a homogeneous self–dual algorithm.

The approach of Gondzio [36] defines a sequence of targets in a vast neighborhood of the central path. These targets are usually easier to reach than analytic centers. The correctors are supposed to take the iterates to these points. Consequently, the iterates remain relatively well centered (a large discrepancy of the complementarity products is avoided) and larger steps can be taken in the primal and dual spaces.

In the following part of this section we shall concentrate on two approaches that proved to be the most attractive in computations: a second order predictor–corrector technique [62] and a multiple centrality correction technique [36].

## 6.1 Predictor–corrector technique

Mehrotra's predictor-corrector strategy [62, 61] has two components which are an adaptive choice of the barrier parameter and the computation of a high-order approximation to the central path.

The first step of the predictor-corrector strategy is to compute the affine scaling (predictor) direction. The affine scaling direction solves the Newton equation system (6) for $\mu = 0$ and is denoted with $\Delta_a$. It is easy to show that if a step of size $\alpha$ is taken in the affine scaling direction, then the infeasibility is reduced by the factor $(1-\alpha)$. Moreover, if the current point is feasible, then the complementarity gap is also reduced by the same factor. Therefore, if a large step can be made in the affine scaling direction, then a desirable progress in the optimization is achieved. On the other hand, if the feasible stepsize in the affine-scaling direction is small, then the current point is probably too close to the boundary. In this case the barrier parameter should not be reduced too much.

Mehrotra suggested to use the predicted reduction in the complementarity gap along the affine scaling direction to estimate the new barrier parameter. After the affine scaling direction has been computed, the maximum stepsizes along this direction in the primal ($\alpha_{Pa}$) and in the dual ($\alpha_{Da}$) spaces are determined preserving nonnegativity of $(x, s)$ and $(z, w)$. Next 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 and the barrier parameter is chosen using the heuristic

$$\mu = \left(\frac{g_a}{g}\right)^2 \frac{g_a}{n}. \tag{42}$$

Next, the high-order component of the predictor-corrector direction is computed. Note that we ideally want to compute a direction such that the next iterate is perfectly centered, i.e.

$$(X + \Delta X)(z + \Delta z) = \mu e. \tag{43}$$

(We have an equivalent relation for variables $s$ and $w$ associated to the upper bounds). The above system can be rewritten as

$$Z\Delta x + X\Delta z = -Xz + \mu e - \Delta X\Delta z. \tag{44}$$

Let us observe that in the computations of the Newton direction in equation (6), the second order term $\Delta X \Delta z$ is neglected. Instead of setting the second order term equal to zero, Mehrotra proposes to estimate $\Delta X \Delta z$ using the affine-scaling direction $\Delta X_a \Delta z_a$. His predictor-corrector direction is obtained by solving the Newton equations system with (44) as the linearized complementarity conditions and the barrier parameter $\mu$ chosen through (42).

We should note here that the above presentation of the predictor–corrector technique follows the computational practice. It abuses mathematics in the sense that stepsizes $\alpha_P$ and $\alpha_D$ are not taken into account when building the higher order Taylor approximation of the central trajectory. The reader interested to see a detailed rigorous presentation of this approach can consult [61].

Let us observe that a single iteration of the (second order) predictor–corrector primal–dual method needs two solves of the same large, sparse linear system for two different right hand sides. The benefit of the method is, we obtain a good estimate for the barrier parameter $\mu$ and a high order approximation to the central path. Indeed computational practice shows that the additional computational cost of the predictor-corrector strategy is more than offset by a reduction in the number of iterations (factorizations).

The predictor–corrector mechanism can be applied repeatedly leading thus to methods of order higher than two. However, the computational results presented in [61] show that the number of iterations does not decrease sufficiently to justify the additional computations. Consequently, the second order predictor–corrector technique became for a couple of years the computational state of the art [54, 62].

The disappointing results for the use of higher (than two) order predictor–corrector technique used to be explained with a difficulty of building an accurate higher order approximation of the central trajectory. On the other hand, many large scale linear programs exist for which the factorizations are extremely expensive. For those problems the need to save on the number of factorizations becomes more important.

The method presented in the next section responds to this need.

## 6.2 Modified centering directions

Let us observe that the step $(\Delta x, \Delta y, \Delta s, \Delta z, \Delta w)$ of (6) aims at drawing all complementarity products to the same value $\mu$. Moreover, to ensure the progress of optimization, the barrier parameter $\mu$ has to be smaller than the average complementarity product $\mu_{average} = (x^T z + s^T w)/2n$. Such perfectly centered points usually cannot be reached. Although the theory requires that subsequent iterates are in the neighborhood of the central path, in the computational practice, they may stay quite far away from it without negative consequences for the ability of taking large steps (and the fast convergence).

The approach proposed by Gondzio [36] applies multiple centrality corrections and combines their use with a choice of reasonable, well centered targets that are supposed to be easier to reach than perfectly centered (but usually unreachable) analytic centers. The idea to use targets that are not analytic centers comes from Jansen, Roos, Terlaky and Vial [44]. They define a sequence of traceable weighted analytic centers, *targets* that goes from an arbitrary interior point to a point close to the central path. The algorithm follows these targets and continuously (although very slowly) improves the centrality of subsequent iterates. The targets are defined in the space of the complementarity products.

The method of [36] translates this approach into a computational practice combining the choice of attractive targets with the use of multiple correctors. It abuses the theory of [44] in the sense that it does not limit the improvement of centrality (measured with the discrepancy between the largest and the smallest complementarity product). Below, we briefly present this approach.

Assume $(x, s)$ 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 $\Delta_p$ at this point is determined and the maximum stepsizes in primal, $\alpha_P$ and dual, $\alpha_D$ spaces are computed that preserve nonnegativity of the primal and dual variables, respectively.

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

$$\Delta = \Delta_p + \Delta_m. \tag{45}$$

To enlarge these stepsizes from $\alpha_P$ and $\alpha_D$ to $\tilde{\alpha}_P = \min(\alpha_P + \delta_\alpha, 1)$ and $\tilde{\alpha}_D = \min(\alpha_D + \delta_\alpha, 1)$, respectively, a corrector term $\Delta_m$ has to compensate for the negative components in the primal and dual variables

$$\begin{aligned} (\tilde{x}, \tilde{s}) &= (x, s) + \tilde{\alpha}_P(\Delta_p x, \Delta_p s), \\ (\tilde{y}, \tilde{z}, \tilde{w}) &= (y, z, w) + \tilde{\alpha}_D(\Delta_p y, \Delta_p z, \Delta_p w). \end{aligned} \tag{46}$$

We try to reach the goal by adding the corrector term $\Delta_m$ that drives from this exterior trial point to the next iterate $(\hat{x}, \hat{s}, \hat{y}, \hat{z}, \hat{w})$ lying in the vicinity of the central path. However, we are aware that there is little chance to reach the analytic center in one step that is to reach $v = (\mu e, \mu e) \in \mathcal{R}^{2n}$ in the space of the complementarity products. Hence, we compute the complementarity products of the trial point $\tilde{v} = (\tilde{X}\tilde{z}, \tilde{S}\tilde{w}) \in \mathcal{R}^{2n}$, and concentrate the effort on correcting only their outliers. We thus project the point $\tilde{v}$ componentwise on a hypercube $H = [\beta_{min}\mu, \beta_{max}\mu]^{2n}$ to get the following target

$$v_t = \pi(\tilde{v}|H) \in \mathcal{R}^{2n}. \tag{47}$$

The corrector direction $\Delta_m$ solves the linear system similar to (6) for the following right hand side

$$(0, 0, 0, v_t - \tilde{v}) \in \mathcal{R}^{4n+m}, \tag{48}$$

with nonzero elements only in a subset of positions of $v_t - \tilde{v}$ that refer to the complementarity products which do not belong to $(\beta_{min}\mu, \beta_{max}\mu)$.

Once the corrector term $\Delta_m$ is computed, the new stepsizes $\hat{\alpha}_P$ and $\hat{\alpha}_D$ are determined for the composite direction

$$\Delta = \Delta_p + \Delta_m, \tag{49}$$

and the primal–dual algorithm can move to the next iterate.

The correcting process can be repeated a desirable number of times. In such a case, the direction $\Delta$ of (49) becomes a new predictor $\Delta_p$ and is used to compute the new trial point (46). An advantage of this approach is that computing every single corrector term needs exactly the same effort (it is dominated by the solution of the system like (6) with the right hand side (48)).

The questions arise about the choice of the "optimal" number of corrections for a given problem and the criteria to stop correcting if it brings no improvement. They were answered in [36]. Naturally, the more expensive the factorizations of (25) or (26) compared with the following backsolves, the more correctors should be tried.

The computational experience of [36] proved that, when applied to the solution of nontrivial problems, this method gives significant CPU time savings over the second order predictor–corrector technique of Mehrotra.

# 7 Optimal basis identification

In many practical applications of linear programming, a sequence of closely related problems has to be solved. This is, for example, the case in the branch and bound algorithm for integer programming or in column generation (cutting planes) methods. Obviously when two closely related problems are solved the previous optimal solution should and can be used to solve the new problem faster. In the context of the simplex algorithm this aim is achieved by starting from the previous optimal basic solution. In the context of an interior-point method, warm start procedure still does not exist and it is not obvious that this problem will ever be solved satisfactorily (cf. Section 9). Some hope comes from the particular IPM application in which approximate analytic centers are looked for [32, 63], but in the general case, interior-point warm-start is inefficient. Consequently, the approach adopted nowadays is to solve the first problem of a sequence of closely related problems using an IPM and then cross-over to the simplex method. In this case the advantages of both methods are exploited.

In this section, we shall address the problem of recovering an optimal basis from an almost optimal primal-dual interior-point solution. Before, we would like to note that there exist LP applications in which an optimal interior-point solution is preferable, see, e.g., Christiansen and Kortanek [17] and Greenberg [39].

The primal-dual algorithm discussed in the previous sections produces an optimal basic solution only if the optimal solution is unique (which is very rare in practice). In fact, in the case of either multiple primal or multiple dual optimal solutions, the primal-dual method will generate an optimal solution in the analytic center of the optimal face, see Güler and Ye [40]. Therefore an algorithm is needed that generates an optimal basis from an optimal interior-point solution.

## 7.1 Notation

In this section we will work with the problem in a simplified standard form (in which primal variables have no upper bounds)

$$\text{minimize} \quad c^T x \quad \text{subject to} \quad Ax = b, \quad x \geq 0. \tag{50}$$

The dual to (50) is

$$\text{maximize} \quad b^T y \quad \text{subject to} \quad A^T y + z = c, \quad z \geq 0. \tag{51}$$

It is well-known that any optimal solution $(x^*, y^*, z^*)$ must satisfy the complementarity slackness conditions $x_j^* z_j^* = 0$. Moreover, it is known that there exists a strictly complementary solution that satisfies $x_j^* + z_j^* > 0$, see Goldman and Tucker [34]. Let $(x^*, y^*, z^*)$ be such a strictly complementary solution and define $\mathcal{P}^* = \{j : x_j^* > 0\}$. It can be shown that $\mathcal{P}^*$ is invariant with respect to all strictly complementary solutions. Hence $\mathcal{P}^*$ is unique. The pair $(\mathcal{P}^*, \bar{\mathcal{P}}^*)$, where $\bar{\mathcal{P}} = \{1, \ldots, n\} \setminus \mathcal{P}$ for any set $\mathcal{P}$, determines an optimal partition.

Furthermore we use the notation $x_{\mathcal{P}} = x_{j \in \mathcal{P}}$ for any vector $x$ and any set $\mathcal{P}$. $|\mathcal{P}|$ means the number of elements in $\mathcal{P}$. For any set $\mathcal{P}$, $P$ denotes a matrix built of columns corresponding to variables that belong to $\mathcal{P}$, namely, $P = a_{(.,j \in \mathcal{P})}$, where $a_{(.,j)}$ is the $j$th column of $A$.

Let $(\mathcal{B}, \mathcal{N})$ denote a partition of the variables into basic and non-basic variables. $(\mathcal{B}, \mathcal{N})$ is an optimal basis, if $B$ is non-singular,

$$x_{\mathcal{B}} = B^{-1} b \geq 0, \; x_{\mathcal{N}} = 0 \tag{52}$$

and

$$y = B^{-T} c_{\mathcal{B}}, \; s_{\mathcal{N}} = c_{\mathcal{N}} - N^T y \geq 0, \; s_{\mathcal{B}} = 0. \tag{53}$$

A basic solution is said to be primal (dual) degenerate if at least one component in $x_{\mathcal{B}}$ ($z_{\mathcal{N}}$) is zero.

## 7.2 The pivoting algorithm

The best algorithm to generate an optimal basis has been proposed by Megiddo [59]. It constructs an optimal basis in less than $n$ iterations starting from any complementary solution and it is strongly polynomial. Megiddo has proved an even stronger result: he has shown that an optimal basis cannot be constructed from a primal or dual optimal solution in strongly polynomial time unless there exists a strongly polynomial algorithm for LP.

Below we shall discuss Megiddo's algorithm and its implementation. For convenience we assume that a set of artificial variables has been added to the problem (50). Let $\mathcal{V} = \{1, \ldots, m\}$ denote the set of artificial variables; naturally, we must have $x_{\mathcal{V}} = 0$ in any optimal solution. Furthermore, we assume that a strictly complementary solution is known. Hence, we assume that:

a. We know the optimal partition $(\mathcal{P}^*, \bar{\mathcal{P}}^*)$ and $\mathcal{V} \subseteq \bar{\mathcal{P}}^*$.

b. We know an optimal primal solution $\tilde{x}$ such that $A\tilde{x} = b$, $\quad \tilde{x}_{\bar{\mathcal{P}}^*} = 0$ and $\tilde{x}_{\mathcal{P}^*} > 0$.

c. We know an optimal dual solution $(\tilde{y}, \tilde{z})$ such that $A^T \tilde{y} + \tilde{z} = c$, $\tilde{z}_{\bar{\mathcal{P}}^* \setminus \mathcal{V}} > 0$ and $\tilde{z}^*_{\mathcal{P}^*} = 0$.

In fact, the algorithm presented below works for any complementary solution, i.e. when the conditions $\tilde{x}_{\mathcal{P}^*} > 0$ and $\tilde{z}_{\bar{\mathcal{P}}^* \setminus \mathcal{V}} > 0$ in assumption b and c are relaxed to $\tilde{x}_{\mathcal{P}^*} \geq 0$ and $\tilde{z}_{\bar{\mathcal{P}}^* \setminus \mathcal{V}} \geq 0$.

Megiddo's algorithm consists of a primal and a dual phase. Let us start with a description of the primal phase. Let $(\mathcal{B}, \mathcal{N})$ be any partition of the variables of the problem (50) into basic and non-basic parts. Then

$$x_{\mathcal{B}} := B^{-1}(b - N\tilde{x}_{\mathcal{N}}) = \tilde{x}_{\mathcal{B}} \geq 0 \tag{54}$$

because $B$ is non-singular. The solution $(\tilde{x}_{\mathcal{B}}, \tilde{x}_{\mathcal{N}})$ is called a *super-basic* solution since some of the non-basic variables are not at their lower bound zero (the non-basic variables that are not identical to zero are called super-basic). The idea of the primal phase is to move all super-basic variables to zero or to pivot them into the basis using simplex like iterations. The resulting basis is primal optimal, because it is feasible and it is complementary with respect to the dual optimal solution $(\tilde{y}, \tilde{z})$. Each move or pivot step reduces the number of super-basic variables by one: since the number of super-basic variables cannot exceed $|\mathcal{P}^*|$, the algorithm terminates after at most $|\mathcal{P}^*|$ iterations.

Now we will state the algorithm

**Algorithm 7.1**

1. *Choose a basis $\mathcal{B}$ and let $x = \tilde{x}$.*
2. *while($\exists\ j \in \mathcal{P}^* \setminus \mathcal{B} : x_j \neq 0$)*
3. *Use the primal ratio test to move variable $x_j$ to zero if possible or pivot it into the basis.*
4. *Update $(\mathcal{B}, \mathcal{N})$ and $x$.*
5. *end while*
6. *$\mathcal{B}$ is a primal optimal basis.*

It can be observed that in step 1, it is always possible to choose a basis. One possible choice is $\mathcal{B} = \mathcal{V}$. Algorithm 7.1 is a simplified version of the primal simplex algorithm, because there is no pricing step (the incoming variables are predetermined).

The dual phase of Megiddo's algorithm is similar to the primal phase because, in this case, a super-basic dual solution is known. This means that some of the reduced costs corresponding to the basic variables might not be zero. Similarly to the primal phase, those reduced costs can either be moved to zero or the corresponding primal variable has to be pivoted out of the basis. The dual algorithm can be stated as follows

**Algorithm 7.2**

1. *Choose a basis $\mathcal{B}$ and let $y = \tilde{y}$, $z = c - A^T y$.*
2. *while($\exists\ j \in \bar{\mathcal{P}}^* \cap \mathcal{B} : z_j \neq 0$)*
3. *Use the dual ratio test to move variable $z_j$ to zero if possible or take it out of the basis.*
4. *Update $(\mathcal{B}, \mathcal{N})$, $y$ and $z$.*
5. *end while*
6. *$B$ is a dual optimal basis.*

If the initial basis is primal feasible, then it remains feasible throughout all steps of Algorithm 7.2 because all pivots are primal degenerate. Once Algorithm 7.2 terminates, the final basis is both primal and dual feasible and hence optimal. Furthermore, the number of iterations in the dual phase cannot exceed $|\bar{\mathcal{P}}^*|$.

Summing up, Algorithms 7.1 and 7.2 generate an optimal basis after at most $n$ iterations. In practice, the number of iterations is dependent on the level of primal and dual degeneracy.

### 7.3 Implementational issues of the pivoting algorithm

Megiddo's algorithm presented in the previous section assumes that an exact optimal solution is known. This assumption is never met in practice, because the primal-dual algorithm only generates a sequence converging towards the optimal solution. Furthermore, due to the finite precision of computations, the solution returned by the primal-dual algorithm is neither exactly feasible nor complementary.

Bixby and Lustig solve this problem using a Big-M version of Megiddo's algorithm that is their cross-over procedure drives both complementarity and feasibility to zero. This algorithm adds, in the worst case, several simplex pivots to obtain an optimal basis. Their approach works well but, unfortunately, it complicates the implementation of the cross-over procedure.

Andersen and Ye [4] propose an alternative solution to this problem. Let $(x^k, y^k, z^k)$ be the iterate generated by the primal-dual algorithm in iteration $k$ and $(\mathcal{P}^k, \bar{\mathcal{P}}^k)$ be a guess of the optimal partition generated in iteration $k$. Now define the following perturbed problem

$$\text{minimize} \quad (c^k)^T x \quad \text{subject to} \quad Ax = b^k, \quad x \geq 0, \tag{55}$$

where

$$b^k = P^k x^k_{\mathcal{P}^k}, \quad c^k_{\mathcal{P}^k} = (P^k)^T y^k \quad \text{and} \quad c^k_{\bar{\mathcal{P}}^k} = (\bar{P}^k)^T y^k + z^k_{\bar{\mathcal{P}}^k}.$$

Assume the variables in (55) are reordered such that $x = (x_{\mathcal{P}^k}, x_{\bar{\mathcal{P}}^k})$ then the vector $(x, y, s) = ((x^k_{\mathcal{P}^k}, 0), y^k, (0, z^k_{\bar{\mathcal{P}}^k}))$ is a strictly complementary solution to (55). Moreover, if $x^k$ converges towards an optimal primal solution and $\mathcal{P}^k$ converges towards $\mathcal{P}^*$, then $b^k$ converges towards $b$ and, similarly, $c^k$ converges towards $c$. Therefore the two problems (50) and (55) will eventually share optimal bases. This advocates for an application of Megiddo's algorithm to the perturbed problem (55) instead of to (50). Note, that an optimal complementary solution to the problem (55) is known.

An important practical issue is a choice of an indicator $\mathcal{P}^k$ for the optimal partition $\mathcal{P}^*$. A trivial one is

$$\mathcal{P}^k = \{j : x^k_j \geq z^k_j\}. \tag{56}$$

Unfortunately, this indicator is not invariant with respect to the column scaling. Hence it is less attractive. Another indicator is

$$\mathcal{P}^k = \{j : |\Delta^a_{x_j}|/x^k_j \leq |\Delta^a_{z_j}|/z^k_j\}, \tag{57}$$

where $(\Delta^a_x, \Delta^a_z)$ is the primal-dual affine scaling search direction and this indicator is scaling invariant. It uses the variable changes to guess the optimal partition. This indicator is justified by the theory in [22] and is also reliable in practice.

Another question is the choice of the right iteration to terminate the interior point algorithm and to start the cross-over. The optimal basis generation can only be expected to produce the correct optimal basis if the interior point solution is almost optimal and $\mathcal{P}^k$ is a good guess for $\mathcal{P}^*$. A good practical criteria when to make a switch is when fast (quadratic) convergence of primal-dual algorithm sets in.

Finally, for a discussion of linear algebra issues related to implementing the pivoting algorithm and computational results we refer the reader to the papers [10, 4].

## 8 Interior point software

Now more than ten years after Karmarkar's publication, interior point methods is a well understood area both in theory and practice. The current implementations are sophisticated optimization tools capable to solve very large linear programs. Moreover, the interior-point methods have proved to be significantly more efficient than the best available simplex implementations for many LP problems [54].

Several efficient LP codes based on interior-point methods have been developed the most recent years. Almost all codes are based on the primal-dual algorithm presented above although they differ in many implementational details. There exist several commercial vendors, e.g.: AT&T (KORBX), CPLEX(CPLEX/BARRIER, `http://www.cplex.com`), DASH(XPRESS-MP, `http://www.dash.com`) and IBM (OSL, `http://www.research.ibm.com/osl/`) as well as numerous research codes, some of them public domain in an executable or even in a source code form. The reader may find it surprising that these research codes compare favorably with the best commercial products. Three public domain research codes draw particular attention.

Vanderbei's LOQO is an implementation of the predictor-corrector primal-dual method for linear and quadratic programming. The code is written in C and is available in an executable form of a callable library. LOQO is available from `http://www.sor.princeton.edu/~rvdb/`. Note that LOQO is only free if it is used for academic purposes.

Zhang's LIPSOL, written in MATLAB and FORTRAN also implements the predictor-corrector primal-dual method. Its undoubtful advantage is an ease of comprehension resulting from the use of MATLAB's programming language. LIPSOL is available from
`http://pc5.math.umbc.edu/~yzhang/`.

Gondzio's HOPDM is an implementation of a higher order primal-dual method (cf. Section 6.2). This code is public domain in a form of FORTRAN source files from
`http://ecolu-info.unige.ch/~logilab/software/`.

Mészáros' BPMPD, is another implementation of a higher order primal-dual method. The code is available in a form of FORTRAN source files for academic purposes from
`ftp://ftp.sztaki.hu/pub/oplab/SOFTWARE/BPMPD`.

The reader interested in more information about these LP codes (both commercial and research ones) should consult the LP FAQ (LP Frequently Asked Questions). The World Wide Web address of the LP FAQ is

- `http://www.skypoint.com/subscribers/ashbury/linear-programming-faq`
- `ftp://rtfm.mit.edu/pub/usenet/sci.answers/linear-programming-faq`

To give the reader some idea about the efficiency of available commercial and research LP codes, we run them on a few public domain test problems. Table 6 gives their sizes, i.e. the number of rows, columns and nonzero elements, m, n, and nonz, respectively. Problems `pilot87`, `dfl001` and `pds-10` come from Netlib; problems `mod2`, `world` and `NL` belong to the collection maintained at the University of Iowa. Table 7 reports statistics on their solution (iterations and CPU time in seconds to reach 8-digit optimality) on an IBM Power PC workstation (model 601: 66 MHz, 64 MB RAM). In the case of 8-digits optimality could not be reached, we give in parenthesis the number of exact digits in the suboptimal solution. The following solvers are compared: CPLEX version 3.0 SM (simplex method), CPLEX version 3.0 BARRIER, LIPSOL version 0.3, LOQO version 2.21, HOPDM version 2.12 and BPMPD version 2.1. CPLEX represents the current state-of-the-art commercial LP optimizer. The remaining solvers are the earlier mentioned research codes.

Table 6: Test problem statistics.

| Name | n | m | nonz |
|---|---|---|---|
| pilot87 | 2030 | 4883 | 73804 |
| dfl001 | 6071 | 12230 | 41873 |
| pds-10 | 16558 | 48763 | 140063 |
| mod2 | 35664 | 31728 | 220116 |
| world | 35510 | 32734 | 220748 |
| NL | 7195 | 9718 | 102570 |

Table 7: Commercial vs. public domain solvers.

| Problem | Cplex SM | | Cplex/BAR | | LIPSOL | | LOQO | | HOPDM | | BPMPD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | its | time | its | time | its | time | its | time | its | time | its | time |
| pilot87 | 10167 | 722.2 | 41 | 602.8 | 38 | 817.4 | 47 | 1220.8 | 24 | 501.4 | 30 | 391.9 |
| dfl001 | 63389 | 3766.9 | 47 | 3829.3 | 85 | 9906.8 | 53 | 7378.6 | (7)33 | 5294.3 | 34 | 2921.4 |
| pds-10 | 38327 | 1222.7 | 60 | 3462.1 | 51 | 5824.3 | 51 | 5616.9 | 29 | 3393.7 | 30 | 2715.4 |
| mod2 | 117360 | 18199.9 | 57 | 947.0 | 72 | 3737.3 | 73 | 1591.7 | 47 | 1069.1 | 48 | 875.7 |
| world | 134270 | 22469.5 | 62 | 1145.6 | 60 | 3115.2 | 74 | 1753.6 | 51 | 1345.2 | 57 | 1109.4 |
| NL | 32273 | 1124.8 | 31 | 147.2 | 35 | 319.3 | 29 | 210.0 | 23 | 171.2 | 30 | 148.5 |

Before analyzing the results collected in Table 7 we would like to warn the reader that the computational results are dependent on many different factors. For example the choice of test problems, the choice of computer and the choice of algorithmic parameters all influence the relative performance of the codes. The results reported in Table 7 have been obtained when all compared codes were run with their default options.

The analysis of results collected in Table 7 indicates that there is only insignificant difference in the efficiency of commercial and public domain research codes. The latter are available free of charge.

Although there are many different LP codes available nowadays, the reader may be interested in preparing his own implementation of an IPM. We have to warn him that it might not be a trivial task. A lot of different issues have to be dealt with, e.g., the system design, the choice of the programming language, etc.

In general, implementing the primal-dual algorithm in C or FORTRAN is a time-consuming job. Whereas any IPM can be implemented fast using the MATLAB environment. MATLAB has a sparse matrix capability which means that relatively large LP problems can be solved efficiently with it.

If performance is the ultimate goal, then the code should be implemented in C or FORTRAN. Which programming language to use is a matter of taste. Even the commercial codes does not use the same language.

When a programming language has been chosen the next step is to choose a system design. It is advisable to build the code based on well structured modules. For instance the Cholesky factorization should be implemented in a separate module. Another recommendation is to build the optimizer such that it can be called as a stand alone procedure.

Regarding the form of input data, the standard MPS format surely has to be accepted although more efficient binary formats might be advantageous. We refer the reader to the book [67] for a good discussion of the MPS format.

A good reason to be able to read the MPS format is that the majority (if not all) test problems are available in it. One such collection is the so-called Netlib suite available via anonymous ftp to `netlib.att.com` (`cd netlib/lp`). Another source of larger and more difficult problems is an LP test collection gathered at the University of Iowa. It is also available via anonymous ftp to `col.biz.uiowa.edu` (`cd pub/testprob/lp`).

# 9 Is all the work already done?

From reading the previous section, the reader should have an impression that the area of interior point methods for linear programming is deeply explored. Indeed current IPM implementations are extremely powerful, robust and often significantly faster than the simplex codes. A natural question arises about the relevant problems that still remain open about the methods implementation. From our point of view there are at least a two important questions:

- implementation of postoptimal analysis in a correct way, and
- warm start.

When implementing LP algorithms one has to consider methods to produce shadow prices and ranges. OR practitioners are used to the simplex based postoptimal analysis that assume the knowledge of an optimal basis and are not always aware of potential mathematical errors and sometimes misleading economical consequences of it [39]. This is especially the case if the LP problem is degenerate which is almost always the case in practice. An interior-point based postoptimal analysis will in this case give more accurate answers, see Jansen et al.[43]. However, an interior-point based postoptimal analysis is potentially computationally much more expensive than the simplex based method.

The general warm start procedures in IPMs still work unsatisfactorily slow and are not competitive to the simplex based reoptimizations. As mentioned in Section 7, the only promising results to date have been obtained in a particular case when an IPM is used to find an approximate analytic center of a polytope (not to optimize an LP). It seems that the best approach currently is to solve difficult problems with an IPM, identify its optimal basis and later employ the simplex method if reoptimization is required.

Apart from the two practical problems mentioned above further implementational improvements can be expected. Although we have concluded that current IPM implementations work efficiently, we are aware that there exist LP problems that are very sparse but produce surprisingly dense symmetric factorizations, e.g.: `dfl001` or `pds-` problems from the Netlib collection. It is possible that the right way to solve these problems is to apply iterative approaches to the Newton equations system.

Finally an increasing accessibility of parallel computers in the near future will make IPM methods that exploit this architecture more important. Indeed such algorithm will be able to solve LP problems much larger than currently possible. This will have important consequences for the areas of integer programming (improved cutting-plane methods) and the area of stochastic optimization.

## 10 Conclusions

In the previous sections we have addressed the most important issues of an efficient implementation of interior-point methods.

Our discussion has been concentrated on the most important algorithmic issues such as the role of centering (or, equivalently, following the central path) and the way of treating infeasibility in a standard primal-dual algorithm (we have presented the HLF model which solves the problem of detecting infeasibility efficiently). Furthermore, we have discussed in detail the computationally most expensive part of the IPM methods – the solution of the Newton equations system.

The progress in the IPM methods for LP during the past decade is impressive. Indeed a complete theory of interior point methods have been developed. Moreover, based on this theory many efficient implementations of IPMs have been constructed. In fact due to this algorithmic development and the improvements in the computer hardware much larger LP problems can be solved routinely today than a decade ago. Even though the methods are not going to improve so dramatically over the next decade we nevertheless predict significant improvements in the current implementations.

Finally, we hope and believe that these developments are useful to the OR practitioners.

## References


[1] I. Adler, N. Karmarkar, M. G. C. Resende, and G. Veiga. Data structures and programming techniques for the implementation of Karmarkar's algorithm. *ORSA J. on Comput.*, 1(2):84–106, 1989.

[2] E. D. Andersen. Finding all linearly dependent rows in large-scale linear programming. *Optimization Methods and Software*, 6:219–227, 1995.

[3] E. D. Andersen and K. D. Andersen. Presolving in Linear Programming. Preprint 35, Dept. of Math. and Computer Sci., Odense University, 1993. To appear in Math. Programming.

[4] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms for linear programming. Technical report, Department of Management Sciences, The University of Iowa, 1994. Available from ftp://col.biz.uiowa.edu/pub/papers/cross.ps.Z, to appear in Management Science.

[5] K. D. Andersen. A modified Schur complement method for handling dense columns in interior point methods for linear programming. Technical report, Dept. of Math. and Computer Sci., Odense University, 1994. Submitted to ACM Transaction on Mathematical Software.

[6] M. Arioli, J. W. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Mat. Anal. Appl.*, 10(2):165–190, 1989.

[7] M. Arioli, I. S. Duff, and P. P. M. de Rijk. On the augmented system approach to sparse least-squares problems. *Numer. Math.*, 55:667–684, 1989.

[8] J. R. Birge, R. M. Freund, and R. Vanderbei. Prior reduced fill–in solving equations in interior point algorithms. *Oper. Res. Lett.*, 11:195–198, 1992.

[9] R. E. Bixby. Progress in linear programming. *ORSA J. on Comput.*, 6(1):15–22, 1994.

[10] R. E. Bixby and M. J. Saltzman. Recovering an optimal basis from an interior point solution. *Oper. Res. Lett.*, 15(4):169–178, 1993.

[11] A. Björk. Methods for sparse linear least squares problems. In J. R. Bunch and D. J. Rose, editors, *Sparse Matrix Computation*, pages 177–201. Academic Press INC., 1976.

[12] A. L. Brearley, G. Mitra, and H. P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Math. Programming*, 15:54–83, 1975.

[13] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinit systems of linear equations. *SIAM J. Numer. Anal.*, 8:639–655, 1971.

[14] T. J. Carpenter, I. J. Lustig, J. M. Mulvey, and D. F. Shanno. Separable quadratic programming via a primal-dual interior point method and its use in a sequential procedure. *ORSA J. on Comput.*, 5:182–191, 1993.

[15] S. F. Chang and S. T. McCormick. A hierachical algorithm for making sparse matrices sparse. *Math. Programming*, 56:1–30, 1992.

[16] I. C. Choi, C. L. Monma, and D. F. Shanno. Further Development of a Primal-Dual Interior Point Method. *ORSA J. on Comput.*, 2(4):304–311, 1990.

[17] E. Christiansen and K. O. Kortanek. Computation of the collapse state in limit analysis using the LP primal affine scaling algorithm. *J. Comput. Appl. Math.*, 34:47–63, 1991.

[18] P. D. Domich, P. T. Boggs, J. E. Rogers, and C. Witzgall. Optimizing over three dimensional subspaces in an interior-point method for linear programming. *Linear Algebra Appl.*, 152:315–342, 1991.

[19] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices.* Oxford University Press, New York, 1989.

[20] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. The factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal.*, 11:181–204, 1991.

[21] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. The Yale sparse matrix package, I. the symmetric code. *Internat. J. Numer. Methods Engrg.*, 18:1145–1151, 1982.

[22] A. S. El-Bakry, R. A. Tapia, and Y. Zhang. A study of indicators for identifying zero variables in interior-point methods. *SIAM Rev.*, 36(1):45–72, 1994.

[23] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley and Sons, New York, 1968.

[24] J. J. H. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Math. Programming*, 57:341–374, 1992.

[25] J. J. H. Forrest and J. A. Tomlin. Implementing the simplex method for optimization subroutine library. *IBM Systems J.*, 31(1):11–25, 1992.

[26] R. Fourer and S. Mehrotra. Solving symmetric indefinite systems in an interior point method for linear programming. *Math. Programming*, 62:15–40, 1993.

[27] K. R. Frisch. The logarithmic potential method of convex programming. Technical report, University Institute of Economics, Oslo, Norway, 1955.

[28] D. M. Gay. Electronic mail distribution of linear programming test problems. *COAL Newsletter*, 13:10–12, 1985.

[29] A. George and J. W. -H. Liu. *Computing Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[30] A. George and J. W. -H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31:1–19, 1989.

[31] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. On the projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Math. Programming*, 36:183–209, 1986.

[32] J. L. Goffin and J. P. Vial. Cutting planes and column generation techniques with the projective algorithm. *J. Optim. Theory Appl.*, 65:409–429, 1990.

[33] A. J. Goldman and A. W. Tucker. Polyhedral convex cones. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and related Systems*, pages 19–40, Princeton, New Jersey, 1956. Princeton University Press.

[34] A. J. Goldman and A. W. Tucker. Theory of linear programming. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and related Systems*, pages 53–97, Princeton, New Jersey, 1956. Princeton University Press.

[35] J. Gondzio. Splitting dense columns of constraint matrix in interior point methods for large scale linear programming. *Optimization*, 24:285–297, 1992.

[36] J. Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. Technical Report 1994.20, Logilab, HEC Geneva, Section of Management Studies, University of Geneva, November 1994. Revised May 1995, to appear in Computational Optimization and Applications.

[37] J. Gondzio. Presolve analysis of linear programs prior to applying the interior point method. Technical Report 1994.3, Logilab, HEC Geneva, Section of Management Studies, University of Geneva, 1994. Revised Dec. 1994, to appear in ORSA J. on Comp.

[38] J. Gondzio. HOPDM (version 2.12) - A fast LP solver based on a primal-dual interior point method. *European J. Oper. Res.*, 85:221–225, 1995.

[39] H. J. Greenberg. The use of the optimal partition in a linear programming solution for postoptimal analysis. *Oper. Res. Lett.*, 15(4):179–186, 1994.

[40] O. Güler and Y. Ye. Convergence behaviour of interior-point algorithms. *Math. Programming*, 60(2):215–228, 1993.

[41] A. J. Hoffman, M. Mannos, D. Sokolowsky, and N. Wiegman. Computational experience in solving linear programs. *Journal of the Society for Industrial and Applied Mathematics*, 1:17–33, 1953.

[42] P. -F. Hung and Y. Ye. An asymptotical $O(\sqrt{n}L)$-iteration path-following linear programming algorithm that uses wide neighborhoods. Technical report, Department of Mathematics, The University of Iowa, March 1994. To appear in SIAM J. on Optimization.

[43] B. Jansen, C. Roos, and T. Terlaky. An interior point approach to postoptimal and parametric analysis in linear programming. In *Interior point methods*. Eövös Loránd University, Department of Operations Research, H-1088 Budapest, Múzeum krt. 6-8., Hungary, 1992.

[44] 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.

[45] B. Jansen, T. Terlaky, and C. Roos. The theory of linear programming: Skew symmetric self-dual problems and the central path. *Optimization*, 29:225–233, 1994.

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

[47] N. K. Karmarkar, J. C. Lagarias, L. Slutsman, and P. Wang. Power series variants of Karmarkar–type algorithms. *AT&T Tech. J.*, 68:20–36, 1989.

[48] M. Kojima, N. Megiddo, and S. Mizuno. A primal-dual infeasible-interior-point algorithm for linear programming. *Math. Programming*, 61:263–280, 1993.

[49] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior point algorithm for linear proramming. In N. Megiddo, editor, *Progress in Mathematical Programming: Interior-Point Algorithms and Related Methods*, pages 29–47. Springer Verlag, Berlin, 1989.

[50] J. W. -H. Liu. A generalized envelope method for sparse factorization by rows. *ACM Trans. Math. Software*, 17(1):112–129, 1991.

[51] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra Appl.*, 20:191–222, 1991.

[52] I. J. Lustig, R. E. Marsten, and D. F. Shanno. The interaction of algorithms and architectures for interior point methods. In P. M. Pardalos, editor, *Advances in optimization and parallel computing*, pages 190–205. Elsevier Sciences Publishers B.V., 1992.

[53] I. J. Lustig, R. E. Marsten, and D. F. Shanno. On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM J. on Optim.*, 2(3):435–449, 1992.

[54] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Interior point methods for linear programming: Computational state of the art. *ORSA J. on Comput.*, 6(1):1–15, 1994.

[55] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Sci.*, 3:255–269, 1957.

[56] I. Maros and Cs. Mészáros. The role of the augmented system in interior point methods. Technical Report TR/06/95, Brunel University, Department of Mathematics and Statistics, London, 1995.

[57] K. A. McShane, C. L. Monma, and D. F. Shanno. An implementation of a primal-dual method for linear programming. *ORSA J. on Comput.*, 1(2):70–83, 1989.

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

[59] N. Megiddo. On finding primal- and dual- optimal bases. *ORSA J. on Comput.*, 3(1):63–65, 1991.

[60] S. Mehrotra. Handling free variables in interior methods. Technical Report 91-06, Department of Industrial Engineering and Managment Sciences, Northwestern University, Evanston, USA., March 1991.

[61] S. Mehrotra. High order methods and their performance. Technical Report 90-16R1, Department of Industrial Engineering and Managment Sciences, Northwestern University, Evanston, USA., 1991.

[62] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. on Optim.*, 2(4):575–601, 1992.

[63] O. du Merle, J. L. Goffin, and J. P. Vial. A short note on the comparative behaviour of Kelley's cutting plane method and the analytic center cutting plane method. Technical Report 1996.4, Logilab, HEC Geneva, Section of Management Studies, University of Geneva, January 1996.

[64] Cs. Mészáros. Fast Cholesky factorization for interior point methods of linear programming. Technical report, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1994. To appear in Computers & Mathematics with Applications.

[65] Cs. Mészáros. The "inexact" minimum local fill–in ordering algorithm. Working paper WP 95–7, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1995.

[66] Cs. Mészáros. The augmented system variant of IPMs in two–stage stochastic linear programming computation. Working paper WP 95–1, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1995.

[67] J.L. Nazareth. *Computer Solution of Linear Programs.* Oxford University Press, New York, 1987.

[68] J. von Neumann. On a maximization problem. Technical report, Institute for Advanced Study (Princeton, NJ, USA), 1947.

[69] E. Ng and B. W. Peyton. A supernodal Cholesky factorization algorithm for shared–memory multiprocessors. *SIAM J. Sci. Statist. Comput.*, 14(4):761–769, 1993.

[70] L. Portugal, F. Bastos, J. Judice, J. Paixõ, and T. Terlaky. An investigation of interior point algorithms for the linear transportation problems. Technical Report 93–100, Faculteit der Technische Wiskunde en Informatica, Technische Universiteit Delft, Nederlands, 1993.

[71] M. G. C. Resende and G. Veiga. An efficient implementation of a network interior point method. Technical report, AT&T Bell Laboratores, Murray Hill, NJ, USA, February 1992.

[72] E. Rothberg and A. Gupta. Efficient Sparse Matrix Factorization on High-Performance Workstations-Exploiting the Memory Hierarchy. *ACM Trans. Math. Software*, 17(3):313–334, 1991.

[73] G. Sonnevend, J. Stoer, and G. Zhao. Subspace methods for solving linear programming problems. Technical report, Institut fur Angewandte Mathematik und Statistic, Universitat Wurzburg, Wurzburg, Germany, January 1994.

[74] G. W. Stewart. Modifying pivot elements in Gaussian elimination. *Math. Comp.*, 28:537–542, 1974.

[75] G. W. Stewart. On scaled projections and pseudoinverses. *Linear Algebra Appl.*, 112:189–193, 1989.

[76] R. Subramanian, R. P. S. Scheff Jr., J. D. Qillinan, D. S. Wiper, and R. E. Marsten. Coldstart: Fleet assigment at Delta Air Lines . *Interfaces*, 24(1), 1994.

[77] U. H. Suhl. MPOS - Mathematical optimization system. *European J. Oper. Res.*, 72(2):312–322, 1994.

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

[79] W. F. Tinney and J. W. Walker. Direct solution of sparse network equations by optimally ordered triangular factorization. In *Proceedings of IEEE*, volume 55, pages 1801–1809. 1967.

[80] A. W. Tucker. Dual systems of homogeneous linear relations. In *Linear inequalities and related systems*, pages 3–18. Princeton University Press, Princeton, NJ, 1956.

[81] K. Turner. Computing projections for Karmarkar algorithm. *Linear Algebra Appl.*, 152:141–154, 1991.

[82] R. J. Vanderbei. Splitting dense columns in sparse linear systems. *Linear Algebra Appl.*, 152:107–117, 1991.

[83] R. J. Vanderbei and T. J. Carpenter. Symmetric indefinite systems for interior point methods. *Math. Programming*, 58:1–32, 1993.

[84] X. Xu. An $\mathcal{O}(\sqrt{n}L)$-iteration large-step infeasible path-following algorithm for linear programming. Technical report, College of Business Administration, The University of Iowa, Iowa City, IA 52242, August 1994.

[85] X. Xu. On the implementation of a homogeneous and self-dual linear programming algorithm. Technical report, 1994. Manuscript.

[86] X. Xu, P.-F. Hung, and Y. Ye. A simplified homogeneous and self-dual linear programming algorithm and its implementation. Technical report, Department of Management Sciences, The University of Iowa, 1993.

[87] X. Xu and Y. Ye. A generalized homogeneous and self-dual algorithm for linear programming. *Oper. Res. Lett.*, 17:181–190, 1995.

[88] M. Yannakakis. Computing the minimum fill–in is NP–complete. *SIAM J. Algebraic Discrete Methods*, pages 77–79, 1981.

[89] Y. Ye. On the finite convergence of interior-point algorithms for linear programming. *Math. Programming*, 57:325–335, 1992.

[90] Y. Ye, M. J. Todd, and S. Mizuno. An $O(\sqrt{n}L)$ - iteration homogeneous and self-dual linear programming algorithm. *Math. Oper. Res.*, 19:53–67, 1994.