The Efficient Implementation of Interior Point Methods for Linear Programming and their Applications

by

Csaba Mészáros

Supervisor: Dr. István Maros

Ph.D. Thesis

Eötvös Loránd University of Sciences,
Ph.D. School of Operations Research, Applied Mathematics and Statistics

Budapest, 1996.

Contents

1 Introduction **3**

2 The infeasible-interior-point primal-dual algorithm **5**

3 The augmented system variant of interior point methods **10**

4 The theoretical background of the $L\Lambda L^T$ decomposition **14**

5 An adaptive procedure for structure recognition **16**

6 A new symbolical ordering method **20**

7 The supernodal and loop unrolling techniques in the numerical computation **31**

8 Applications **35**

8.1 Special network problems 37

8.2 Two–stage stochastic linear programming problems 40

8.3 Approximation in L_1 and L_∞ norms under nonnegativity restrictions . . . 45

9 Summary **46**

1 Introduction

I started research on practical mathematical programming in 1992. As continuation of my Master Thesis, my first activity was the investigation of the numerical behavior of the simplex implementations of linear programming [57, 48]. Simultaneously, as result of the influences during my short staying period at the Operations Research Department, Eötvös Loránd University, my interest turned to the interior point methods, as of many researchers and practitioners of linear programming.

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) [64], Hoffman et al. (1953) [34] and Frisch (1955) [25], 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 [38] 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. [29] 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) [25] and is formally studied in Fiacco and McCormick [21] 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 [46] 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 [68, 10, 23], extensive numerical tests (cf. [46]) 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 computationally most attractive IPM is an infeasible-primal-dual algorithm. During my Ph.D. research, I developed an own implementation (called BPMPD) of this method which was used in my research to perform all necessary computational tests. To implement such an algorithm efficiently is a highly complex task and a time consuming job: my code is a result of three years development and consists of approximately 12 thousand Fortran source lines. Of course, an implementation contains many parts of different theoretical and implementational complexity. To demonstrate this, I divided my implementation to different parts and counted their dividend relative to the whole 12 thousand source lines as:

Implementation fragment	dividend
Memory management, others	6 %
Input/output	15 %
Presolve	18 %
Symbolical investigations	23 %
Numerical procedures	20 %
Basic algorithm	12 %
Higher order extensions	6 %

The numeric computations are the most involved steps of any implementation. To make them efficient is of extraordinary importance. Even the symbolical investigations are closely related to the numerical procedures, because they are used to reduce the number of necessary arithmetical operations as possible. In my research I derived techniques which make the "numerical algebra kernel" of interior point implementations robust and fast. This investigation contains theoretical likewise implementational discussions. Because the topics of my research are closely related to the infeasible-primal-dual method, in section 2 I present this algorithm. In section 3 I demonstrate the importance of the "augmented system" approach of interior point methods and I propose a new factorization scheme, based on 1×1 pivoting. In section 4 I prove a statement which is the theoretical basis of my approach. One important corollary of the statement is that the new factorization scheme can be used efficiently in interior point implementations. Section 5 describes my adaptive procedure for structure recognition. This procedure is based on the evaluation of the sparsity pattern of the constraint matrix, and it is able to make algorithmic decisions. Additionally, the informations about structure are used in the pivot choice principle. This is an important and valuable part of my work and a new technique in the implementation methodology of IPMs. In section 6 I present a parametric set of symbolic orderings for the normal equations approach. This set of ordering methods connects the "minimum degree" and "minimum local fill–in" approaches, and I show that in the "neighborhood" of the minimum degree ordering a practically efficient method exist. The further increase of the efficiency of the implementation was made by exploiting the supernodal structute of the factorizations during computations. In section 7 I present an algorithm for using the supernodal loop unrolling technique in the framework of the left looking factorization and I demonstrate its efficiency. The application of developed tools is an important issue by any science. In case of operations research this means modelling and solving practical problems. In section 8 I describe three different applications, on which my techniques provide much better performance than the existing ones. I consider these applications as a particularly valuable part of my Ph.D. research.

2 The infeasible-interior-point primal-dual 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 [53] who proposed to apply a logarithmic barrier method to the primal and the dual problems at the same time. Independently, Kojima, Mizuno and Yoshise [41] developed the theoretical background of this method and gave the first complexity results. But, at this point the theory is still far from the computational practice. The theoretical bound of $\mathcal{O}(\sqrt{n}\ \log\frac{1}{\epsilon})$ iterations to obtain an ϵ–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.

The first implementations [52, 15] 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 [43, 45, 46, 56, 31].

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 [40] who show that the primal-dual algorithm with some safe-guards has good theoretical properties.

Let us consider a primal linear programming problem

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

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

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

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 not unbounded.

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 problem:

$$\begin{array}{ll} \text{minimize} & c^T x - \mu \sum_{j=1}^{n} \ln x_j - \mu \sum_{j=1}^{n} \ln s_j \\ \text{subject to} & \begin{array}{rl} Ax & = b, \\ x + s & = u, \end{array} \end{array} \tag{3}$$

where μ is the barrier parameter. The Lagrangian for (3) is

$$L(x, s, y, w, \mu) = c^T x - \mu \sum_{j=1}^{n} \ln x_j - \mu \sum_{j=1}^{n} \ln s_j - y^T (Ax - b) - w^T (u - x - s)$$

and the first order optimality conditions for (3) are:

$$\begin{array}{rcl} \nabla_x L = & c - \mu X^{-1} e - A^T y + w & = 0, \\ \nabla_s L = & -\mu S^{-1} e + w & = 0, \\ \nabla_y L = & b - Ax & = 0, \\ \nabla_w L = & u - x - s & = 0, \end{array}$$

where X, S and W are diagonal matrices with the elements x_j, s_j and w_j, respectively and e is the n-vector of all ones. By introducing $Z = \mu X^{-1}$ and $z = Ze$ we can write the first order optimality conditions of (3) as

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

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 μ. 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 μ. 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 μ. 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 μ-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 μ and then μ 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 (ξ_b, ξ_u and ξ_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 ξ_b, ξ_u and ξ_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

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

In context of interior point methods (8) is usually referred as *augmented system*. Once the system (8) has been solved, Δx and Δy are used to compute $\Delta s, \Delta z$ and Δw by (7). Next the maximum stepsizes in primal space (α_P) and dual space (α_D) are computed such that the nonnegativity of variables is preserved:

$$\begin{aligned} \alpha_P &= \frac{1}{\max\limits_{k=1\ldots n}\left\{1, -\frac{\Delta x_k}{x_k}, -\frac{\Delta s_k}{s_k}\right\}}, \\ \alpha_D &= \frac{1}{\max\limits_{k=1\ldots n}\left\{1, -\frac{\Delta z_k}{z_k}, -\frac{\Delta w_k}{w_k}\right\}}. \end{aligned}$$

These stepsizes are slightly reduced with a factor $\alpha_0 < 1$ to prevent hitting the boundary. Finally a new iterate is computed as follows

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

After making the step, the barrier parameter μ is decreased by a given factor and the process is repeated.

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

$$\frac{||Ax-b||}{1+||b||} \leq 10^{-p} \quad \text{and} \quad \frac{||x+s-u||}{1+||u||} \leq 10^{-p}, \tag{11}$$

$$\frac{||A^Ty+z-w-c||}{1+||c||} \leq 10^{-p}, \tag{12}$$

$$\frac{|c^Tx-(b^Ty-u^Tw)|}{1+|b^Ty-u^Tw|} \leq 10^{-p}, \tag{13}$$

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 (11-13) 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 (13) is satisfied and at the same time one of the conditions (11) or (12) 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 (13).

The computationally most expensive step in any implementation of an IPM is the solution of the augmented system (8). This system has to be solved in each iteration which requires computing a symmetric factorization of its matrix followed with a backsolve 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 *higher-order techniques*, which are the most important extensions of the basic primal-dual algorithm. Their common feature is that they reuse the factorization of the augmented 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. Because of their importance in the practice, I shall review them briefly.

The first such approach was proposed by Karmarkar et al. [39] who constructed a parameterized representation of the (feasible) trajectory motivated from the use of differential equations.

Mehrotra's method [56, 55] 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. [17] 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. [66] 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.

The approach of Gondzio [31] 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.

3 The augmented system variant of interior point methods

As noted previously, a most difficult step of any interior point method is to solve the augmented system (8). In the following we are going to denote the matrix on the left side of (8) with M. Several methods can be used to solve (8), both direct and iterative ones. In practice, the symmetric decomposition of M is the most important and commonly used approach. By this, the

$$\begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix} = L\Lambda L^T \tag{14}$$

factorization is computed where L is lower triangular and Λ is diagonal matrix.

Depending on the pivot agenda, this approach opens two possibilities. One of them is the so-called *normal equations approach*, where Δx is eliminated first from (8) which is equivalent to pivoting down the diagonal elements of $-D^{-2}$. Then the remaining system will be just the $AD^2A^T\Delta y = AD^2r + h$ system of normal equations, which gave the name of the approach. Since AD^2A^T is positive definite matrix, the Cholesky factorization and in the preparation phase one of the well–known ordering methods can be used. As pointed out in many papers, the positive definite property makes this approach very attractive, but catastrophic fill–in arise when A is of special structure. Several modifications of the normal equations approach were developed to overcome its bad fill–in behavior, but so far, no solution was found which is able to handle all possible cases.

The first approach to keep fill-in under control was introduced by Todd [70] and elaborated by Hurd and Murphy [36]. The authors consider the case, when the constraint matrix has a special primal block angular or doubly bordered block diagonal structure.

The presented method works well in such cases, but cannot easily be generalized for handling general sparse matrices. The authors demonstrated the efficiency of this approach only a special set of the applications, such as network and stochastic problems or for handling generalized and variable upper bounds.

Another technique due to Birge, Freund and Vanderbei [9] uses the choice between the AA^T and A^TA considerations. The systems of equations to be solved are $m \times m$ or $n \times n$ depending on the choice of the algorithm. Experiments and comparisons show that the approach works well only in the case if $m \gg n$ and A does not contain dense row(s).

Another possibility is the well know *column splitting* technique which splits dense columns into shorter pieces. This method requires an additional variable and a constraint per columns split. A common experience is that this technique is not competitive with other approaches [49, 2].

A popular way of treating dense columns within the normal equations approach employs the *Schur complement mechanism.* It is based on partitioning of the columns of A to sparse (A_1) and dense (A_2) parts, and an explicit decomposition of the matrix

$$AD^2A^T = A_1D_1^2A_1^T \; + \; A_2D_2^2A_2^T, \tag{15}$$

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.

As an alternative to the normal equations, the *augmented system approach* can be used, which is the direct factorization of (14) without prescribing the pivoting in $-D^{-2}$ at the beginning. The augmented system approach is well understood technique to solve a least squares problems [4, 5, 11, 19]. It consists in the application of the Bunch-Parlett factorization [13] to the symmetric indefinite matrix M.

Advantageous fill–in and stability properties of the augmented system approach motivated several researchers to incorporate it into their IPM codes [19, 24, 48, 71, 73]. Soon afterwards, other advantages of this approach, namely, an ease of handling free LP variables and an ability of its easy extension to handling quadratic programming problems were recognized [73, 54, 14]. Additionally, the treatment of the IPM methodology is much easier by the augmented system approach. As pointed out in [58], the modifications on the basic IPM algorithms for handling special cases can be derived using simple pivoting techniques on the augmented system.

Next, I will show the main advantage of the augmented system approach, namely that as a result of the increased freedom in the pivot agenda much better behavior in fill–in can

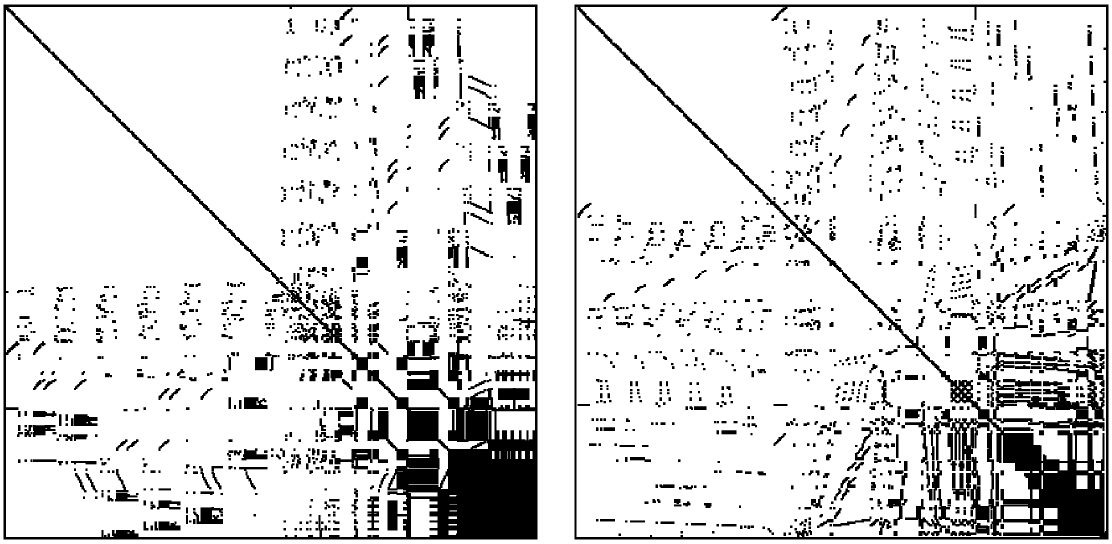
Table 1: Comparison of normal equations (NE) and augmented system (AS) factorizations

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

be achieved in certain cases. Table 1 compares the efficiency of the normal equations (NE) and the augmented system (AS) approaches on a demonstrative set of test problems. By both approaches the minimum degree (that is the minimum Markowitz count) principle was used for pivot selection. I cluster the test problems into three groups. The first group contains problems with dense columns (`aircraft, fit1p, fit2p`). In the second group I 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 1 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 AD^2A^T and nonzeros in A). Columns 7 and 8 contain the number of arithmetic operations (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 1 obtained for problems with few 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 3000×3000 totally dense matrix that resulted from the normal 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. It is to be noted that up to date, no extension of the normal equations approach is know, which can match comparable performance with the augmented system on this type of problems.

Figure 1: Sparsity patterns with the NE (left) and AS (right) pivot rule on problem *stair*



Finally, for our third group of problems, the lower setup cost of the normal equations made the augmented system approach disadvantageous.

Figure 1 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.

Until recently, the augmented system scheme was implemented in two different ways: Mehrotra used the Bunch-Parlett factorization [13] which allows 2×2 blocks in Λ as well. But, the common experience with this approach is that it tend to be computationally burdensome. As an alternative to this, in a partial case, if M can be transformed to a quasidefinite system, the factorization with 1×1 pivots was suggested [73]. This latter approach works only if the linear programming problem is formulated in inequality form, which is not the usual case.

I propose a factorization scheme on (14) with 1×1 pivots without requiring the quasidefinite property. In contrast to the normal equations approach in which the analysis and factorization phases are separated, I propose to compute the factorization (14) 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 (14) 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. In the next section I give the basic theoretical foundations of my pivot scheme. I show, additionally, that to save on the expensive analysis phase, the reuse of the pivot order in subsequent IPM iterations is possible.

Based on large number of numerical experiments, I found that both methods are important for a computational practice and complement each other. My aim was to exploit the possibilities of the augmented system as much as possible but preserve the benefits of normal equations when appropriate. This results in an implementation that includes both versions. To choose one of them should be made by the implementation itself, which requires the detection of special structures. In section 5 I describe the special structure searched for, and an analyzer, based on the investigation of the sparsity pattern of the problems which is able to perform the strategic decision in the solution process. This analyzer provides the fundamental informations for the pivot selection rule as well.

4 The theoretical background of the $L\Lambda L^T$ decomposition

The augmented system approach requires a symmetric decomposition of a symmetric matrix which is neither definite nor quasidefinite. On the one hand, it is possible to use the traditional Bunch-Parlett factorization [13] which allows 2×2 pivot blocks during the decomposition. But, until very recently, no general result was given to supports the 1×1 pivot scheme.

For presenting my ideas I need the general framework of the numerical factorization process on the symmetric system $M = \begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix}$. On the one hand a similar subject was extensively studied both theoretically and computationally in numerical analysis in context of least squares problems (see e.g. [5, 18] and the references therein). On the other hand the use of the augmented factorization in the framework of interior point methods raises some special questions. In this section I present important properties which make the augmented system approach efficient in interior point methods of linear programming.

First we examine the case when A is of full row rank. Thus, because M is of order $m+n$, the elimination process includes $m+n-1$ steps. Let I^s denote the set of row/column indices that were not pivots until the stage s and

$$M^s = \begin{bmatrix} D^s & (A^s)^T \\ A^s & P^s \end{bmatrix}$$

is the active submatrix, i.e., the part of the matrix that has not yet been factorized. With this convention, let denote $M^0 = M$. Thus $A^0 = A$, $D^0 = -D^{-2}$ and P^0 is an $m \times m$ null matrix. It is to be noted that $D^0_{ii} < 0$ for $i = 1, ..., n$ always holds as can be seen from (9). At each step we choose the pivot position from the diagonal of the current active submatrix, which preserves the symmetric structure. At the sth step of the elimination we choose a pivot index $p \in I^s$ for which $m^s_{pp} \neq 0$. Then, $I^{s+1} = I^s - \{p\}$ and the elements of the active submatrix at stage $s+1$ can be computed as

$$m_{ij}^{s+1} = \begin{cases} m_{ij}^s & \text{if } m_{pj}^s = 0 \\ m_{ij}^s - \dfrac{m_{pi}^s m_{pj}^s}{m_{pp}^s} & \text{if } m_{jp}^s \neq 0 \end{cases} \quad \text{for } (i,j) \in I^{s+1} \times I^{s+1}, \tag{16}$$

The sequence of the pivot indices $\{p\}_1^m$ define the ordering permutation and the factorization (14) is obtained by

$$\begin{aligned} \Lambda_{ii} &= m_{p_i p_i}^{p_i} \quad \text{for } i = 1, ..., m+n, \\ L_{ij} &= \frac{m_{p_i p_j}^{p_i}}{m_{p_i p_i}^{p_i}} \quad \text{for } i, j = 1, ..., m+n,\ p_i \leq p_j. \end{aligned}$$

Due to the fact that M is an indefinite matrix and not quasidefinite, it does not stand to reason that such elimination is always successful. For example, the symmetric indefinite matrix

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 0 \\ 2 & 0 & 4 \end{bmatrix}$$

can be factorized by 1×1 pivots following the pivot sequence $(2,3,1)$. The decomposition, however fails when we choose $(1,1)$ for the first pivot position. The following statement guarantees that on our system M, the 1×1 pivot scheme is always valid. Having been fixed for a certain D, it is valid for arbitrary diagonal matrix D with positive diagonal values.

Statement: At each elimination step $0 \leq s < m+n$, system D^s is negative definite and system P^s is positive semidefinite.

Proof: (By induction on s.) The proposition is evidently true for $s = 0$. For $s > 0$, let D^{s+1} and P^{s+1} be the matrices obtained from D^s and P^s by elimination (16). If the pivot was chosen from D^s, i.e., if $p \leq n$ then, the negative definiteness of D^{s+1} follows directly from the negative definiteness of D^s. Furthermore, P^{s+1} is computed as

$$P^{s+1} = P^s - \frac{a_p^s (a_p^s)^T}{m_{pp}^s}, \tag{17}$$

where a_p^s denote the pth column of A^s. Since D^s is negative definite, therefore, $m_{pp}^s < 0$ and the transformation (17) does not change the positive semidefiniteness of P^s. The other case, when $p > n$ may be proven in the similar manner.

It follows from the statement that if A is of full row rank, M can always be factorized by 1×1 pivots. Among them, exactly n pivots are negative and m are positive and for all permitted values of index i, $\Lambda_{ii} > 0 \iff p_i > n$. Let us observe the next consequence of the above statement.

Corollary: $P^s_{ii} = 0$ if and only if the submatrix of A consisting of the columns and rows defined by the pivot positions until the sth elimination step and the ith row of A is not of full row rank.

Actually, in my interior point implementation, I perform a numerical factorization by $D = I$ at the very beginning. As result of this factorization, the pivot agenda and the corresponding sparsity pattern is obtained. The numerical result of the decomposition is used for computation of the starting point. The last corollary is true for any diagonal matrix D with positive diagonal values and so, the pivot order is theoretically valid during all interior point iterations. Practically, indefiniteness may cause problems if some of the variables go very close to zero at early stages of the iterations, still far away from optimality. It happens very rarely if we apply a variant of the primal dual method that efficiently follows the central path. If we reach a very small pivot value during the factorization process and there are still some unprocessed columns of the original constraint matrix in the pivot agenda that could modify this diagonal value, then there are two alternatives:

(i) The very small values may simply be changed (see [67, 3]) and be corrected later during the back-solve operation. Unfortunately, this method carries some undesirable computational overhead per back-solve.

(ii) The pivot agenda is modified by moving the small pivot to the end of the pivot sequence. In this way, this pivot will be positive unless A is rank deficient. This process is very simple and fast but it may create some more fill-in in the factorization. According to my experiences, it has hardly any influence on the total solution efficiency.

5 An adaptive procedure for structure recognition

The basic idea in determining the pivot agenda for the augmented system is that we try to partition the augmented matrix into two parts. The first one is considered "sparse" and the second part is "dense". (This does not necessarily mean that different storage schemes are used for the two parts.) In the first stage of the factorization we make pivot steps in the sparse part as long as possible. This stage may terminate because all columns have been pivotal, or because it was not possible to select a proper pivot element in the remaining columns. In the second stage we determine the pivot elements for the rest of the problem (part two and the remaining columns of part one). This selection is primarily based on the column count and a pivot tolerance. During this stage we may select pivots from the second but also from the first part if it was not fully pivoted.

More formally, we look for a permutation that results in partitioning of A in such a way that

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}, \tag{18}$$

where A_1 is sparse with no dense columns or rows, A_2 and A_3 are considered dense borders.

We consider this partitioning successful if the borders are relatively small, that is, if the number of columns in A_2 and, similarly, the number of rows in A_3 are much smaller than the number of columns/rows in A_1.

Regarding system (8), first we pivot along the diagonal elements in D falling outside A_2. This will result in a very little fill-in. We go in this way as long as possible. When this part is exhausted we use a minimum degree rule for the rest of the augmented matrix.

It is clear that partitioning (18) plays a very important role in the pivot selection. If an algorithm based on a fixed threshold value is used to group columns into dense and sparse parts then we can observe better behavior only in very few cases [73], namely in those for which the threshold value is favorable. For different problems it can be different and manual tuning would be necessary to exploit the scheme. But a further question arises: On what information can such a manual tuning be based? Of course, no general rule can be determined for a decision to use the augmented system without examining the nonzero structure of the constraint matrix. At the same time, this examination itself and the algorithm based on it must be fast. This calls for some kind of an adaptive heuristics that is efficient and cheap.

In an attempt to fulfill these requirements as much as possible, jointly with I. Maros, I developed an algorithm that is described in the sequel [49].

The purpose of this heuristic algorithm is to find such a partitioning where the density of A_1 and that of A_2 differ considerably. Further purposes are that A_1 should be as sparse as possible but should not be too small in size. The main steps of the algorithm can be demonstrated by the following verbal description.

First, the procedure creates groups of columns having the same number of nonzero counts by group (groups of "equidense" columns). Each group will be characterized by the common nonzero count referred to as size.

At the beginning all columns are marked to belong to A_1. The groups are processed in decreasing order of size. The current next group becomes a candidate to join A_2. At each step we examine whether extending A_2 by this new group is acceptable or not. We use two criteria to decide.

Dimension condition (DIMCON): Will the dimension of A_1 remain greater than a predefined **relative measure** ?

Density condition (DENCON): Is the size of the last group in A_2 significantly different from that of the current next group (which is the largest group by size in A_1)?

If DIMCON is true we check DENCON and proceed with the next group. If DIMCON is false and DENCON was not true yet then we try to enlarge A_3 (if it is still not too large) by the row with largest row count and restart the procedure with the reduced matrix (and recomputed counts). If A_3 cannot be enlarged or DENCON was true at least once before the procedure terminates with the last accepted partitioning.

If DENCON is true A_2 is enlarged by all the candidate groups identified so far. There are many subtleties of the procedure that can be identified in the more formal description that follows.

Let c_j denote the column count of column j and r_i the row count of the row i. We introduce the active set of the column counts denoted by K, that is $k \in K \iff \exists j, c_j = k$. Let f_k denote the number of columns with column count value k.

1. let $M = m$, set all rows 'enabled', mark all columns 'sparse'
2. let $P = M + 1$
3. $N = n$
4. $p = \min\{k \in K \text{ and } k < P\}$
5. if $p < minlength$ GOTO 11.
6. $N = N - f_p$
7. if $(M + N) < (m + n)(1 - \beta)$ GOTO 11
8. $P = \min\{k \in K \text{ and } k < p\}$
9. if $p^2/P^2 > dgap$ then
   mark all columns with $c_j > P$ as 'dense' and
   all others with $c_j < p$ as 'sparse'
10. GOTO 5.
11. if $(M + n) < (m + n)(1 - \beta)$ or Step 9 was at least once true GOTO 17.
12. select row j from the 'enabled' with maximal row count
13. set row j disabled
14. $M = M - 1$
15. update counters c_* and f_* and the index set K
16. GOTO 2.
17. if any columns marked as 'dense' ...

If at the termination of the process we found any "dense" columns, then we perform the augmented system factorization by the pivot agenda described earlier. In the other case, the normal equations approach is used, which can be implemented efficiently using symbolical ordering methods [60, 27].

We can summarize the algorithm as follows:

- We accept the partitioning only if A_1 is relatively big (test in the Step 7).
- We do not search the partitioning, when the maximum column count value is small.
- The difference between the sparsity of A_1 and A_2 is tested in the Step 9, where p^2 is the upper bound of the fill-in when pivoting on a column with column count p. The partitioning is accepted only if the difference between the 'sparse' and 'dense' parts is significant.
- We use the 'last' accepted partitioning (if there are more possibilities) where A_1 is the sparsest.

Table 2: Comparison of Cplex and BPMPD on problems when the heuristics selected the augmented system

Problem	Nonzeros		Iterations		Sol. time(sec)	
name	Cplex	BPMPD	Cplex	BPMPD	Cplex	BPMPD
bore3d	3190	2741	17	13	0.88	0.66
fit1p	197351	274	15	16	98.54	3.35
fit2p	$*^{(2)}$	299	$*^{(2)}$	17	$*^{(2)}$	27.68
ganges	32741	17146	16	15	6.15	5.44
israel	11540	1962	23	19	3.51	0.94
pilot4	14631	9285	38	31	7.91	5.55
seba	54445	1134	18	14	25.16	1.65
stair	15545	7842	15	13	3.24	1.98
stocfor3	233793	165858	32	28	115.18	106.56
vtp_base	2454	2104	36	12	1.31	0.44
aircraft	1420885	50	10	12	3049.35	9.40
car4	171560	36700	18	11	141.76	50.54
cep1	315867	5190	17	15	199.82	4.94
complex	473531	73955	$*^{(3)}$	9	$*^{(3)}$	35.53
cr42	289362	5203	19	15	305.05	2.69
pgp2	2214003	15558	23	23	4986.36	20.87
pf2177	15196778	283780	$*^{(1)}$	4	$*^{(1)}$	151.10
route	3436416	256520	$*^{(1)}$	21	$*^{(1)}$	230.53
zed	4230	196	16	10	0.88	0.16

$*^{(1)}$ Cplex was able to make the symbolic factorization but ran out of memory (32 Mbyte) when started iterations
$*^{(2)}$ No data because Cplex ran out of memory (32 Mbyte) during the symbolic phase
$*^{(3)}$ Cplex was unable to solve due to numerical troubles

The parameter values used in our experiments are: $minlength = 10$, $\beta = 0.17$, $dgap = 2.5$. To demonstrate the need of detecting and handling the special structure in the linear programming problems, I prepared Table 2 containing the comparison of the performance of my implementation, called BPMPD, and the state-of-the-art commercial package CPLEX version 2.1. I collected test problems from the NETLIB [26] collection extended with other real–life problems. On these problems the heuristic described in this section detected the structure (18). Note that CPLEX uses the normal equations approach which does not give the flexibility to handle specail structures. Figures given in Table 2 include number of fill–in in the factor, number of iterations to optimality and the solution time on an IPM PC Pentium-90 machine.

The detailed computational results obtained by our method are given in [49]. They show that the heuristic works well in the practice and reliable detects the structure of A.

6 A new symbolical ordering method

The conclusion of numerical experiments showed that it is advantageous to implement both the normal equations and augmented system approaches, because both of them have advantageous and disadvantageous properties [49]. The success of the implementation of the normal equations approach depends on the quality of its *analysis* phase [18, 27], 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 [74]) would be unacceptably expensive. Two such heuristics, namely the *minimum degree* and the *minimum local fill-in* orderings [18, 27, 28] 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 me briefly discuss these two heuristics.

Minimum degree ordering

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

$$f_i = \frac{1}{2}(c_i - 1)^2, \tag{19}$$

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 ith diagonal element becomes a pivot (f_i is the Markowitz merit function [47] applied to a symmetric matrix [69]).

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 [27], 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. The details of this method can be found in the excellent summary of George and Liu [28].

Minimum local fill–in ordering

Let us observe that, in general, function (19) 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 (19), 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 [46], because the analysis that exactly predicts fill–in and chooses the pivot producing its minimum number is very expensive.

Fundamental terms

For better understanding the new ordering heuristic, I need to introduce the fundamental terms of the symbolical orderings. Basically, all symbolic ordering methods work on a symbolic representation of the matrix AA^T. The common way to represent the sparsity structure of AA^T is the use of the following boolean $m \times m$ matrix $\hat{M}$:

$$\hat{m}_{ij} = \begin{cases} true & \text{if position } (ij) \text{ is filled in } AA^T \\ false & \text{if position } (ij) \text{ is not filled in } AA^T \end{cases} . \tag{20}$$

Of course, in the practical sparsity-exploiting implementation we store $\hat{M}$ in a compact form. In contrast to most of the implementation of the minimum degree ordering, my method is based on the explicit representation of the elimination. It means that starting from $I^0 = \{1, ..., m\}$ and $\hat{M}^0 = \hat{M}$, after choosing a pivot index p at the sth step of the elimination we update the set of possible pivot indices as $I^{s+1} = I^s - \{p\}$, and the matrix of the elimination (called as *active submatrix*) $\hat{M}^{s+1}$ as

$$\hat{m}_{ij}^{s+1} = \begin{cases} \hat{m}_{ij}^s & \text{if } \hat{m}_{jp}^s = false \\ \hat{m}_{ij}^s \text{ OR } \hat{m}_{ip}^s & \text{if } \hat{m}_{jp}^s = true \end{cases} \quad \text{for} \quad (i,j) \in I^{s+1} \times I^{s+1}, \tag{21}$$

where OR is the conventional "logical or" function. The sequence of the pivot indices $\{p\}_1^m$ define the ordering permutation. To follow the graph theory conventions up to the end we denote the possible pivot indices $i \in I^s$ as *nodes*, and we call the nodes i and j *adjacent*, if $\hat{m}_{ij}^s = true$. The matrix $\hat{M}^s$ is also called an *adjacency structure*.

Both in the pivot searching and elimination processes such nodes play an important role the columns of which share the same nonzero pattern. They are referred to as *indistinguishable* nodes. In other words, we denote the nodes $i, j \in I^s$ indistinguishable if

$$\hat{m}_{ik}^s = \hat{m}_{jk}^s \text{ for } k \in I^s. \tag{22}$$

Clearly, the indistinguishable nodes i and j behave in the same way, i.e. if one of them is transformed by some other column, the other is also transformed in a similar a way. Therefore, their indistinguishable property remains unchanged. The advantages of the indistinguishable nodes can be exploited with the so–called *supernodal* technique, i.e., we store, update and check by pivot searching only one representative of the indistinguishable nodes. This representative is called a supernode [27]. The administration of the other indistinguishable nodes is possible in the following way: We set up a linked list, which holds the indices of the indistinguishable nodes. Each list is linked to their supernode. When choosing the supernode as pivot, the nodes in the corresponding linked list are inserted into the pivot sequence as well. This technique is usually referred as *mass elimination*.

Two important counts were defined in context of the symbolic orderings. The *true degree* is the synonym of the row and column counts of the unsymmetric case, i.e., the true degree of a node is the number of nodes adjacent to it. By the *external degree* of a node, we mean the number of nodes adjacent to it which are not indistinguishable from the given node itself. In [42], Liu suggests the use of the external degree in the minimum degree algorithm instead of the true degree. The motivation comes from the fact that the external degree contains better expectation about the fill–in than the true degree.

The "inexact" minimum local fill–in heuristic

I proposed an another efficient technique to determine the pivot order [60]. 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.

The motivation to this technique is coming from the unsymmetric Gaussian elimination. Here the Markowitz criterion is the general principle to the pivot selection. By this, however, the number of pivot candidates is considerably large: all nonzero entries in the active submatrix (satisfying some stability criteria in the case of the numerical elimination) are candidates for pivoting. Therefore, choosing one with the smallest Markowitz count is very expensive. To allow improvements in the speed of the pivot searching, another pivot choice principle was used: from among the elements which satisfy the numerical criteria, we want to choose a pivot one with *small* Markowitz count [68]. A number of various techniques were developed to exploit this scheme. Zlatev [75] proposed the searching in the first some rows in increasing order of row counts, Suhl [68] suggested the algorithm which searches among some rows and columns with a small row or column count. My method can be regarded the application of the above two ideas to the symmetric symbolical factorization. My procedure computes the caused fill–in of the first p nodes in the increasing order of the "true" or "external" degree, and we chooses one of them which introduces the least amount of fill–in.

I present the pivot searching algorithm in a Fortran–like pseudo–code. Similarly to the algorithm described in [68], I use linked lists to access to the nodes with the same true or external degree. I discussed two cases. In the first case I used the true degree to set up the list, and to process the searching. In the second case the nodes were linked based on their external degree plus one. Thus the key, used for listing the nodes, is between 1 and m in both cases. To store and manage the linked lists we need two integer working arrays of size m, **degpnt** and **deglst**. In $degpnt(j)$ we hold the entry point into the sublist of nodes with true degree j (or with external degree $j-1$, respectively), and the array **deglst** is used for holding the sublists, represented by the successors.

As pointed out in [18], it is not necessary to recompute the number of fill–in for all pivot candidates. Therefore I introduce the use of the integer working array **fillin** of size m to control this and to make an access to the previously computed values as follows:

$$\text{for } j \in I^s \; fillin(j) = \begin{cases} \text{number of fill–in produced by node } j \\ -1, \text{ if the number of fill–in is to be recompute} \end{cases} \tag{23}$$

Let p denote the search parameter. The input parameters of the procedure are m, p, **degpnt**, **deglst** and **fillin**. The output parameter is $pivot$, which holds the selected pivot position, or 0 if no other pivots are available. The function Compute_Fill predict the number of fill–in of a node, its efficient implementation can be found in [60].

1. $fill = m^2 + 1$;
2. $k = 1$;
3. $deg = 1$;
4. $pp = 0$;
5. $pivot = 0$;
6. $j = degpnt(deg)$;
7. if $j > 0$ then
8.   if $fillin(j) < 0$ let $fillin(j) =$Compute_Fill$(j, ...)$;
9.   if $fillin(j) \geq fill$ GOTO 13.
10. $fill = fillin(j)$;
11. $pivot = j$;
13.   if $fill = 0$ GOTO 18.
14. $j = deglst(j)$;
15. $pp = pp + 1$;
16.   if $pp \leq p$ GOTO 7.
17. endif
18. $deg = deg + 1$;
19. if $deg \leq m$ GOTO 6.
20. RETURN ($pivot$)

The procedure finishes the search if all active nodes were checked (step 19), or the number of nodes examined exceeds the value of the parameter p (step 16), or a node is found which produces no fill–in (step 13). As a result, the node is obtained which produces the minimal amount of fill–in from the set of the first $p+1$ nodes with a small true or external degree. If more than one node exist with minimal fill–in, the procedure chooses one with the smallest true or external degree.

Figure 2: Nonzero pattern of problem 25fv47

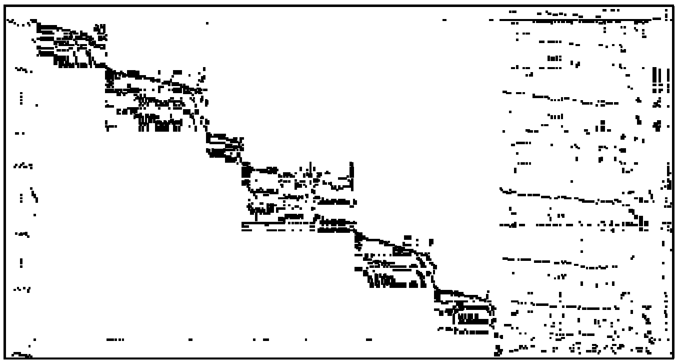
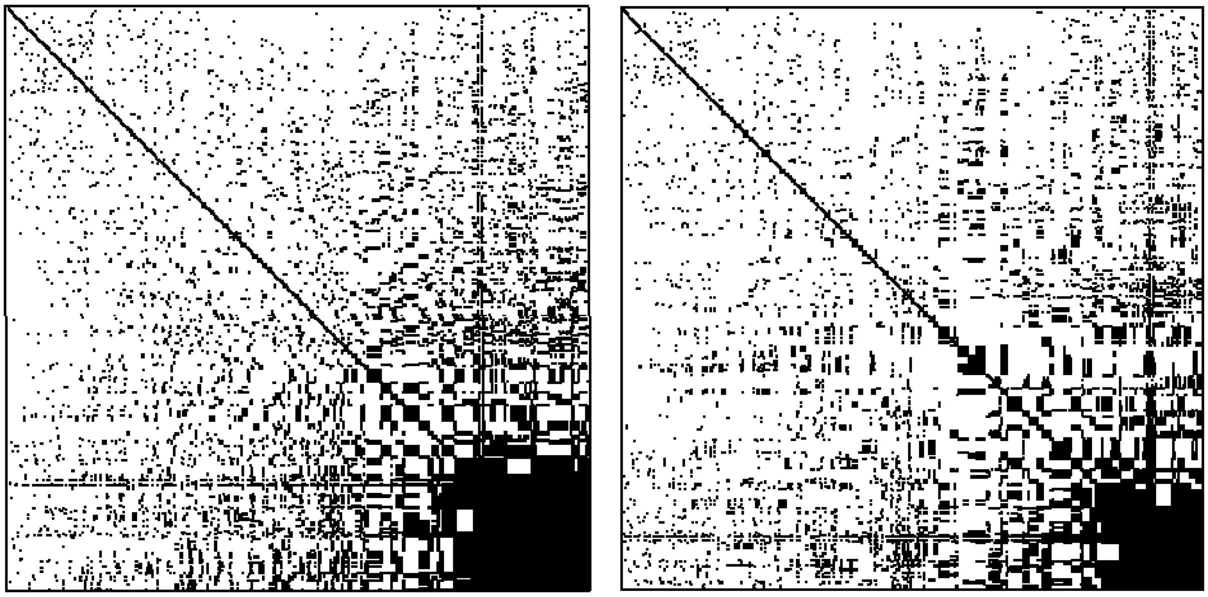
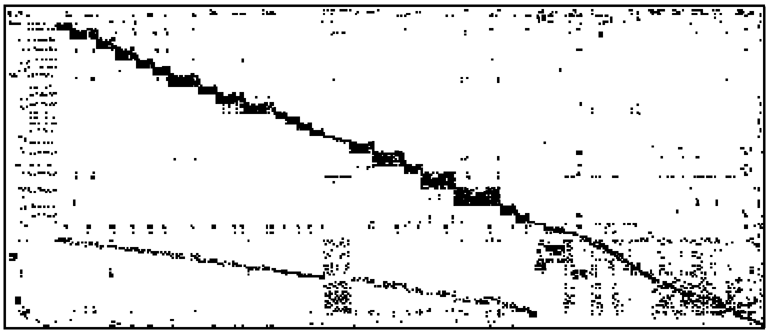


Figure 3: Sparsity pattern of the factorization via MDO and MFO on problem 25fv47



The most important parameter of my algorithm is p, which balances between the cost and the efficiency of the pivot searching. When increasing p, we search longer, and probably we gain a better pivot. Let me note that the setting $p = 0$ gives the minimum degree principle, and $p = m$ the exact minimum local fill–in rule. That is the algorithm describes a parametrized set of ordering methods which connects the two most important symbolical approaches. Naturally, no unique value exists for p which gives the

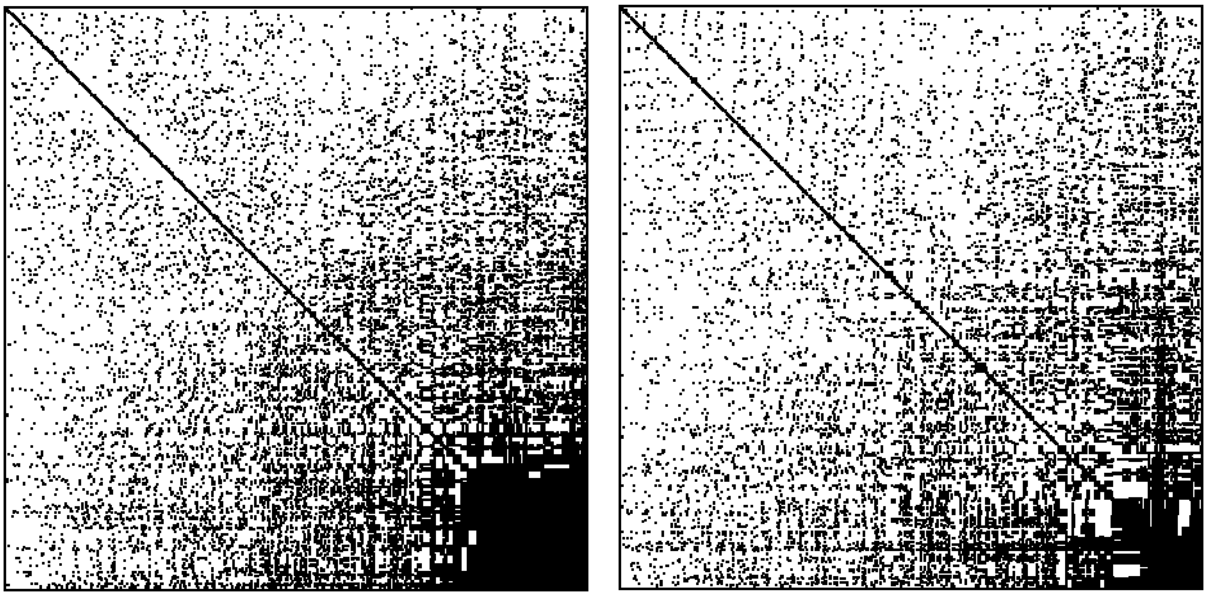
Figure 4: Nonzero pattern of problem `d2q06c`



best performance on all problems. Finding a *good* value is subject of extensive testing.

I choose two test problems, `25fv47` and `d2q06c` from the NETLIB test set [26] to demonstrate the effect of the search parameter p. Problem `25fv47` behaves in an average from our point of view, that is, the minimum local fill–in heuristic produces 15% sparser factorization than minimum degree. Figure 2 shows the sparsity pattern of this problem. The patterns of the factorization of AA^T with the minimum degree and minimum local fill–in ordering can be found in the left– and right–hand side of Figure 3, respectively.

Figure 5: Sparsity pattern of the factorization via MDO and MFO on problem `d2q06c`



Our other test problem is `d2q06c`. Figure 4 shows its sparsity pattern, and Figure 5 the pattern of the factorization with the two ordering methods. I choose this problem because by it the minimum local fill–in outperforms significantly the minimum degree in the sense of the fills.

Figure 6: No. of nonzeros in the factorization (in thousand) on problems 25fv47 (left) and d2q06c (right) as a function of p

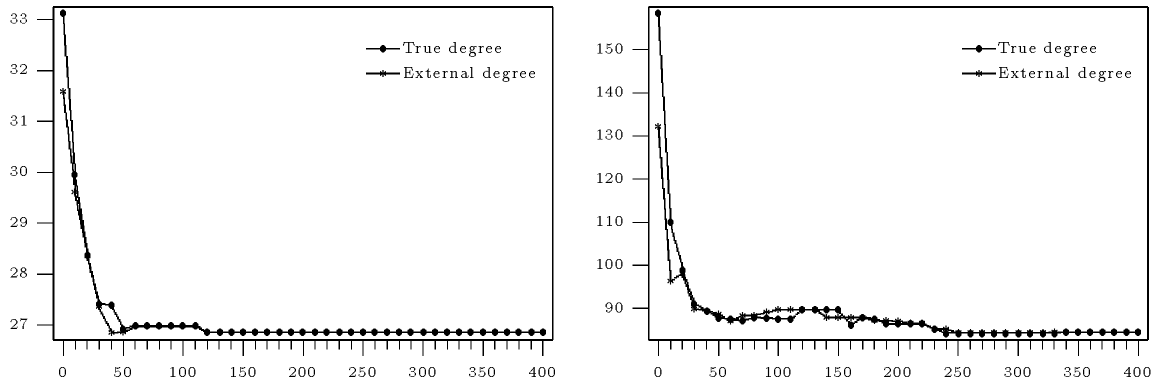
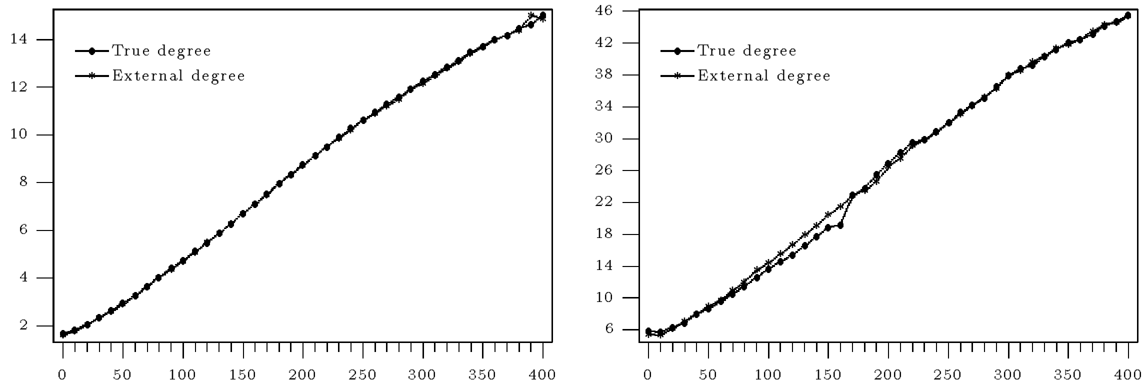


Figure 7: Ordering times (in seconds) on problems 25fv47 (left) and d2q06c (right) as a function of p



In the first experiment I step p between 0 and 400 with 10. I executed both the true degree and external degree variant of the algorithm. Figure 6 shows the number of nonzeros in the factorization in thousands in the case of problems `25fv47` and `d2q06c` respectively, it is plotted against the value of p. From 0 to 30 the results show significant improvement in the number of fill–in, which slows down between 30 and 50. As it can be seen, if $p > 50$, the longer search for a better pivot does not make marked improvement. In both problems the time spent on obtaining the ordering (reported on Figure 7), grows nearly linearly with the value of p. These results suggest concentrating on the values $p \in [0, 50]$.

Figure 8: No. of nonzeros in the factorization (in thousand) on problems 25fv47 (left) and d2q06c (right) as a function of p

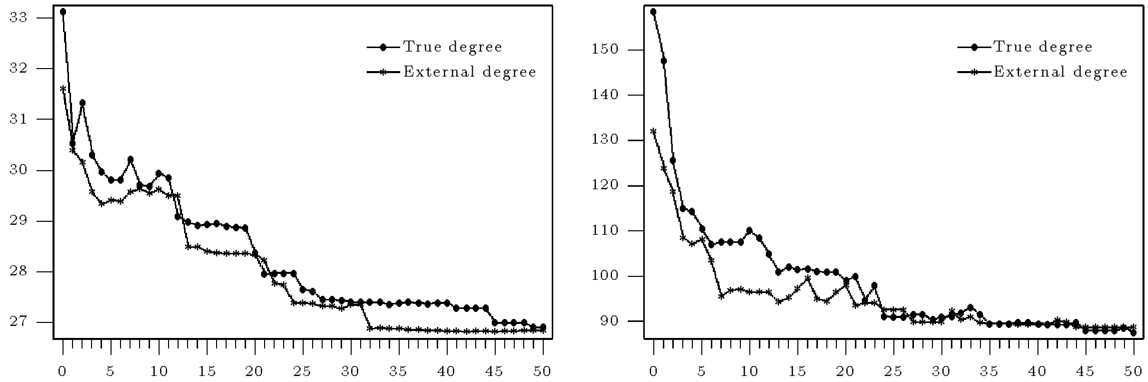
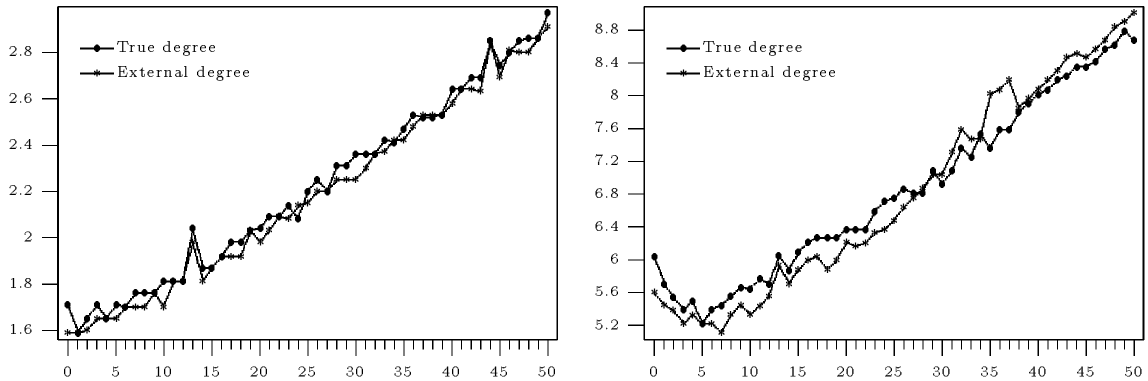


Figure 9: Ordering times (in seconds) on problems 25fv47 (left) and d2q06c (right) as a function of p



In the second experiment I step p between 0 and 50 with 1. In Figure 8 we examine the behavior of fill–in depending on the value of p by problems `25fv47` and `d2q06c`. In both problems the external degree version tends to produce better factorization, but in the later stages (after about $p = 45$ and $p = 25$ by the problems `25fv47` and `d2q06c`, respectively) the difference becomes insignificant. The ordering time reported on the left side of Figure 9 is in keeping with our previous experiences. A more dramatic effect is displayed in the case of problem `d2q06c`. By this, the ordering time is decreased in the first stage as a result of the large decrease in the number of fill–in.

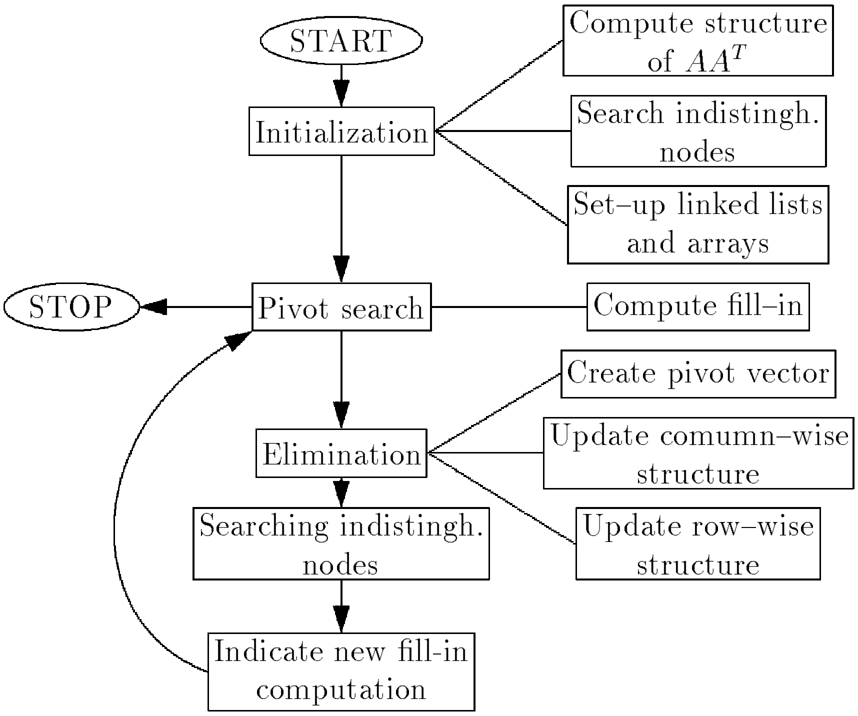
The heuristical assumption, namely the node which produces few fill–in must have small degree, was confirmed by the experiments. The other advantageous feature of my pivot searching algorithm is that it computes the predicted fill–in on nodes with small degree only. It is very important because, as pointed out in [60], the complexity of this process depends on the degree of the nodes.

My experiments show that in general the increase of p result in a nearly uniform increase in the ordering time, but the quality of the ordering remains unchanged after a certain threshold. Therefore, there is no reason for using a value larger than 35. Due to the better performance, I suggest using the external degree version of my algorithm. Based on solving many problems with different settings, my final findings is that $p = 25$ is a good value in the practice [60].

Implementation

The first step of the implementation performs the initialization. It includes the computation of the sparsity structure of the initial matrix $\hat{M} = AA^T$, the search on it for indistinguishable nodes, furthermore the set–up of linked lists and arrays used in the elimination loop. The elimination loop begins with the pivot searching. Then, the update (21) is performed on the active submatrix. This process is very similar to the unsymmetric case. This part of my implementation is based on the ideas described in [68] with modifications to exploit the specialities of the symmetric and symbolic factorization. Before returning to the pivot search, the implementation searches indistinguishable nodes caused by the last transformation, and indicates the requirement of the recomputation of fill–in for certain nodes. The design of my implementation is displayed in Figure 10.

Figure 10: The outline of the implementation



Since M furthermore the active submatrix at each elimination step is symmetric, only the storage of the lower triangular part is necessary. The off–diagonal nonzeros of the lower triangular part of the active submatrix are stored column–wise as a sequence of sparse vectors. A pointer array is used for pointing to the starting position of columns in the column file which contains the row indices. An other array is used for holding the number of entries of each column. By reason of the efficient access, I maintain the row–wise representation also. I use the similar data structure, a pointer array points to the column file and the an array holds the number of entries of each column. Due to the symmetric property of the active submatrix, the row–wise storage scheme can be interpreted as the columns of the upper triangular part of the active submatrix.

It is to be noted that the memory requirement of my implementation is significantly larger than that of the minimum degree algorithm. But, in context of the interior point methods, it is not a decisive issue. The matrix ordering is only a preparation phase of the interior point methods, therefore all working arrays used by the main algorithm are free at this stage. Since the higher order techniques and the efficient numerical factorization methods require considerably large amount of memory, usually no other memory array are needed to be reserved for the ordering.

To demonstrate the advantages of my ordering scheme to the minimum degree ordering described in [27], I shall compare their performance on a subset of large scale linear problems from the NETLIB collection [26]. Table 3 collects the results of this comparison. Abbreviations MDO and MFO in it denote the minimum degree ordering [27] and the "inexact" minimum local fill-in ordering, respectively.

The first three columns of Table 3 contain the problem names and the times (in seconds on a SUN Sparc-2000 workstation) 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-2000 workstation.

Taking on the whole, the numerical comparison with SPARSPAK's GENQMD procedure shoved 17 % reduction in the symbolical time, 20 % reduction in the number of necessary operations to compute factorizations and 32 % reduction in the total solution time of the interior point implementation.

Table 3: 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	1.35	1.09	33572	27379	1337	831	0.380	0.280
80bau3b	3.14	1.83	39821	33977	1341	890	0.521	0.386
bnl2	2.47	1.67	64476	55759	4655	3376	1.020	0.809
cycle	2.59	2.58	51160	39796	1674	911	0.550	0.363
d2q06c	5.84	3.54	154397	92492	16249	4923	3.654	1.481
degen3	72.43	9.89	119241	115749	7930	7441	2.604	2.956
dfl001	194.23	338.80	1619564	1396096	683586	506614	197.125	149.829
greenbea	7.93	1.75	47669	45110	941	826	0.486	0.441
nesm	0.64	1.16	20594	19720	591	535	0.225	0.220
perold	1.21	0.85	23192	19229	885	596	0.230	0.176
pilot87	83.07	74.20	464167	389356	106260	75749	29.214	20.934
scfxm3	0.39	0.28	12723	10257	181	127	0.082	0.070
sctap3	0.83	0.58	14769	14432	224	208	0.120	0.104
shell	0.13	0.16	3505	3301	33	28	0.020	0.020
ship12l	0.78	0.23	4920	4896	39	38	0.055	0.051
truss	1.57	1.79	64575	51593	2784	1689	0.942	0.634
tuff	0.91	0.17	6806	6238	160	129	0.050	0.050
wood1p	2.36	0.86	11473	11473	1087	1087	0.480	0.482
osa-30	1957.81	31.11	110768	110768	2533	2533	6.211	4.580
osa-60	11692.44	109.82	255666	255666	5635	5635	12.979	11.284

7 The supernodal and loop unrolling techniques in the numerical computation

In this section I will demonstrate several practical issues of the numerical factorization step. By this, the decomposition (14) is computed numerically by a given pivot agenda. As noted previously, this is the most time consuming step of any interior point methods, therefore the efficient implementation of this process is of extraordinary importance. For presenting ideas, we suppose that the augmented matrix is permuted in a valid pivot order. As consequence of the results in section 4 the numerical factorization of the augmented system with 1×1 pivots does not differ from the "traditional" Cholesky factorization of positive definite matrices.

One popular method to compute symmetric decomposition is the *column*–Cholesky factorization in which the columns of L are computed one by one. This method is also called *left looking* factorization, because for computing column j of L the information from the left part of the factor (i.e. the columns prior to 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.

We needed to store only the nonzero entries of matrix L, so our formulae for computing the column j of L (denoted by L_j) and Λ_{jj} are:

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

$$L_j = \frac{1}{\Lambda_{jj}} \left(M_j - \sum_{k=1, l_{jk} \neq 0}^{j-1} (\Lambda_{kk} l_{jk}) L_k \right).$$

In the sparsity exploiting implementation we store the nonzeros in the packed form. Let $c(j)$ denote the number of the nonzeros in the column j of L, then we store the nonzero values and their row indices in the two parallel working arrays $nz(1,j), nz(2,j), ..., nz(c(j),j)$ and $indx(1,j), indx(2,j), ..., indx(c(j),j)$. Furthermore the diagonal values of the matrix Λ are stored in a double precision vector $\lambda(.)$. Because M is symmetric, and the sparsity pattern of the off lower diagonal part of M is a subset of the sparsity pattern of L, at the begin of the factorization we can store the matrix M in the place of L in the $nz(.,.)$ and $indx(.,.)$ arrays. For computing a column of L, we use a double precision work vector $wr(.)$ to accumulate the transformations of the previous columns, and the integer work vector $p(.)$ to hold the actual row pointers of the completed columns of L. The pseudo code of this algorithm can be described as follows:

1. do $j = 1, m+n$
2. $\quad p(j) = 1$
3. $\quad \lambda(j) = M(j,j)$
4. $\quad wr(indx(i,j)) = nz(i,j)$ for $i = 1, ..., c(j)$
5. $\quad$ do $k = 1, j-1$
6. $\quad\quad$ if $(c(k) \geq p(k))$ and $(indx(p(k),k) = j)$ then
7. $\quad\quad\quad \lambda(j) = \lambda(j) - \lambda(k) * nz(p(k),k) * nz(p(k),k)$
8. $\quad\quad\quad$ do $i = p(k)+1, c(k)$
9. $\quad\quad\quad\quad wr(indx(i,k)) = wr(indx(i,k)) - \lambda(k) * nz(i,k) * nz(p(k),k)$
10. $\quad\quad\quad$ enddo
11. $\quad\quad\quad p(k) = p(k) + 1$
12. $\quad\quad$ endif
13. $\quad$ enddo
14. $\quad nz(i,j) = wr(indx(i,j))$ for $i = 1, ..., c(j)$
15. enddo

Steps 2-14 are the inner loop for computing the columns L_j. In step 4. we extract the target column in the working array, step 6 checks the condition $L_{jk} \neq 0$. Steps 7-9 are the transformations with the previous columns which fulfil the condition. In step 14. we compress the working vector to the final position of L_j. An alternative way for computing sparse symmetric factorization is the so–called right looking algorithm, which was extensively and very successfully studied by Rothberg and Gupta [65] from our point of view.

The first improvement of the left looking method is the exploitation of the dense window. In the practice the last columns of L are often totally dense. This last partition of the columns of L (the so-called dense window) can be handled separately, dense matrix operations can be used, i.e. no indirect addressing is needed during the transformations. The details of this idea can be found in the book of Duff, Erisman and Reid [18].

The supernodal algorithm

Originally, the idea of the supernodal factorization is coming from parallel and vector computation areas [6]. Their efficiency on the shared-memory multiprocessors is discussed by Esmond and Peyton [20], the exploitation of the cache memory on high-performance workstations is studied by Rothberg and Gupta by right looking factorization [65]. In [61] I examined the efficiency of the supernodal elimination technique by the left looking factorization, which is a superior method to the right looking one on commonly used "low cost" workstations (see [44]). I introduced the tools of its efficient implementation as well.

We can consider the supernodes as the generalization of the dense window : The supernode is a partition of the continuous columns, which share the same nonzero structure. In other words, apart from the empty rows a supernode consists of any, totally dense columns. In my experiments I use another type of the supernode too. By this supernode, the nonzero structure of the columns are identical under the diagonal block only.

$$
\begin{array}{cc}
\text{Type 1 supernode} & \text{Type 2 supernode} \\[6pt]
\begin{bmatrix} * & & \\ * & * & \\ * & * & * \\ * & * & * \\ & & \\ * & * & * \\ * & * & * \end{bmatrix}
&
\begin{bmatrix} * & & \\ & * & \\ & & * \\ * & * & * \\ & & \\ * & * & * \\ * & * & * \end{bmatrix}
\end{array}
$$

By the transformations with the supernodes likewise to the dense window method we can use dense matrix-vector transformations and can save indirect addressing and thereby memory references. For the computation I use one new integer working vector. The vector *snhead*(.) represents the supernode structure in the following manner: each position of *snhead*(.) corresponds to one column and it holds the last column of the corresponding supernode partition, or 0 if the column itself is not a member in any supernode. The new pseudo code is as follows:

```
do j = 1, m + n
   p(j) = 1
   λ(j) = M(j, j)
   wr(indx(i, j)) = nz(i, j) for i = 1, ..., c(j)
   k = 1
   if (c(k) ≥ p(k)) and (indx(p(k), k) = j) then
      if(snhead(k) ≠ 0) then
         l = min(snhead(k), j − 1)
         nz(i − p(k), j) = wr(ind(i, k) for i = p(k) + 1, .., c(k)
          call dense(k, l, j, p, c, nz, λ)
          wr(ind(i, k)) = nz(i − p(k), j) for i = p(k) + 1, .., c(k)
          p(i) = p(i) + 1 for i = k, .., l
          k = snhead(k)
        else
          λ(j) = λ(j) − λ(k) ∗ nz(p(k), k) ∗ nz(p(k), k)
          do i = p(k) + 1, c(k)
             nz(indx(i, k)) = nz(indx(i, k)) − λ(k) ∗ nz(i, k) ∗ nz(p(k), k)
          enddo
          p(k) = p(k) + 1
        endif
     endif
```

22. $k = k + 1$
23. if $(k \leq j - 1)$ GOTO 6.
24. $nz(i, j) = wr(indx(i, j)$ for $i = 1, ..., c(j)$
25. enddo

The new steps 7-13 are the supernodal transformations. I use the subroutine *dense* to compute dense matrix-vector transformations, the columns $k, ..., l$ are applied to column j. The dense computation can be further exploited by loop unrolling technique. In the typical inner loop of the factorization we add a multiple of a column into another (e.g. step 17). Let a be the target vector, b the source vector and c the multiplicator. If we assume that c is kept in a register, then the steps of the computer by one transformation of the $a \leftarrow a + cb$ can be written as follows:

1. reading $a(i)$ from the memory
2. reading $b(i)$ from the memory
3. computing $a(i) + cb(i)$
4. storing the result in the memory

In this case we do one arithmetical transformation (one addition and a multiplication) in step 3 and three memory references (steps 1,2 and 4). During factorization we do multiple column modifications on a single column, and can unroll the loop over the column transformations. Let a be the target vector, b, c, d, e, f and g the source vectors, and $h(1), ..., h(6)$ scalar multiplicators. By the 6-way loop unrolling we compute the following transformation:

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

To do this transformation eight memory references and six arithmetical transformations were needed in the inner loop of the transformation, which is by ten-memory-reference less than the equivalent transformations in the original way.

To demonstrate the importance of these techniques I show the computational results of their application on a set of test problems for one widely used computer architecture, namely, a SUN Sparc-10 workstation.

In Table 4 I 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, I have chosen 6 test examples with very different characteristics. Problems `aircraft` and `fit2p` have extremely sparse factorization with the augmented system. Problems `80bau3b` and `25fv47` are "usual" sparse

Table 4: Speed comparison of different factorization techniques

Name	T(ll)	T(dw)	T(sn1)	T(sn2)	T(sn4)	T(sn6)
aircraft	0.13	0.14	0.14	0.13	0.13	0.12
fit2p	0.36	0.36	0.37	0.37	0.36	0.35
80bau3b	0.63	0.58	0.66	0.60	0.54	0.53
25fv47	0.56	0.52	0.55	0.48	0.42	0.41
rat7a	266.0	252.8	213.6	178.2	164.0	151.1
dfl001	461.2	370.7	232.0	202.3	185.6	171.8

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.

The efficiency of the 6-way loop unrolling supernodal method and the standard left-looking method is compared on a larger set of the test problems. The results obtained on a SUN Sperc-10 workstation are reported in table 5. The first four columns contain the name of the problems, the numbers of the rows, columns and nonzero elements in the original constraint matrix. Column 4. holds the number of the different supernode partitions, column 6. the number of the supernodal columns. The last two columns contain the time required by the standard left looking Cholesky factorization and by supernodal loop unrolling technique in seconds. The first part of table 5 contains the results on NETLIB problems, the second part on other ones.

8 Applications

An important findings of my research is that in context of IPMs, the role of the augmented system is the viable alternative for cases when the normal equations approach suffers heavy fill–in. It is clear that the partitioning (18) plays very important role in the detection of such cases. But a further question arises: on what kind of applications the structure (18) can be recognized? In this section I would like to demonstrate the necessity of the use the augmented system approach and sophisticated investigation techniques on the structure of the linear programming problems. I introduce three different real–life models on which the augmented system approach gives remarkably better performance.

Table 5: Comparison of the standard and the 6-way loop unrolling techniques

Name	Rows	Cols.	Nonz.	S.nodes	S.cols.	T(ll)	T(sn6)
25fv47	821	1571	10400	92	493	0.56	0.41
80bau3b	2262	9301	20413	105	457	0.63	0.53
bnl2	2324	3489	13999	136	732	2.56	1.63
cycle	1903	2857	20720	214	1190	1.26	0.98
d2q06c	2171	5167	32417	152	1098	6.96	4.27
degen3	1503	1818	24646	144	727	3.21	2.63
dfl001	6071	12230	35632	69	2046	461.2	171.8
fit2d	25	10500	129018	1	23	0.59	0.65
fit2p	3000	13525	50284	1	24	0.36	0.35
ganges	1309	1681	6912	66	401	0.36	0.14
greenbea	2392	5302	30715	149	803	1.33	1.04
greenbeb	2392	5290	30676	150	805	1.26	0.99
maros	846	1408	9576	85	481	0.30	0.25
nesm	662	2748	13078	51	323	0.29	0.22
pilot	1441	3449	41092	76	882	9.63	6.03
pilot87	2030	4663	70682	101	1217	33.94	21.22
pilot-ja	940	1677	11821	64	482	1.47	0.99
pilot-we	722	2711	8862	36	250	0.19	0.15
pilotnov	975	1968	12186	70	490	1.31	0.90
scfxm3	990	1371	7777	120	530	0.09	0.08
sctap2	1090	1880	6714	100	323	0.15	0.11
sctap3	1480	2480	8874	159	493	0.17	0.14
ship08l	778	4283	12802	28	144	0.08	0.09
ship08s	778	2387	7114	44	208	0.04	0.05
ship12l	1151	5427	16170	58	202	0.10	0.12
ship12s	1151	2763	8178	61	298	0.05	0.06
stocfor3	16675	15695	64875	2105	6649	1.39	1.44
truss	1000	8806	27836	69	684	0.76	0.61
woodw	1098	8405	37474	115	632	0.81	0.66
aircraft	3754	7517	20267	1	5	0.13	0.12
complex	1023	1398	46453	11	396	4.62	2.90
cr42	905	1513	6614	5	18	0.06	0.06
l30	2701	15380	51169	149	1926	5.02	3.06
rat1	3136	9408	88267	70	2785	115.6	67.00
rat5	3136	9408	137413	93	2973	103.7	59.89
rat7a	3136	9408	268908	19	2973	266.0	151.1
slptsk	2861	3347	72465	19	274	9.35	8.62
south31	18425	35223	93673	26	64	0.57	0.58

8.1 Special network problems

I observed that the partitioning (18) exists on a class of optimization problems on network structures, where the decision variables can be classified as *primary* and *secondary* ones in the following manner:

- For each arc in the network there are more different primary variables given, they represent different possible actions.
- For each node one secondary variable is given which accumulates the effect of the primary variables on a given node.

Then the partitioning (18) is as follows: The block of the constraint matrix consisting of the primary variables and the constraints describing the connections between variables and the network belongs to $A1$. The columns corresponding to the secondary variables defines $A2$ since the constraints describing the technological relationships between the primary variables define $A3$.

By the problem of allocating limited resources to multiple projects [1] and by a vehicle routing problem this structure of the mathematical model was recognized. Here, I would like to demonstrate the advantages of the augmented system on this latter, because it was particularly brought to our attention in respect of an industrial project at the Hungarian Academy of Sciences, Department of Operation Research and Decision Support Systems. We developed a software package to solve the problem of gas cylinder delivery for the international company Primagáz-Hungária. One part of this task was to plain daily routes for trucks under certain technological restrictions. This kind of problems are usually referred as *vehicle routing* problems [12, 30]. In our real application the following data were given:

- deterministic demands for gas cylinders to be transported a day from a certain filling station, the sales points to be supplied from there,
- the data related to possibly different trucks available during a day, i.e., their transporting capacities (number of gas cylinders), the transportation cost of gas cylinders (HUF/km), the speed and their possible maximum length of working time a day.

The daily transportation problems are formulated by considering the next requirements:

- demands must be satisfied to almost the highest extent, and simultaneously, priorities should be given among the sales points,
- the delivery capacity of trucks should not be exceeded,
- trucks may have at most 2 routes a day and

- restrictions regarding the total working time of daily routes for each truck are taken into account.

Let NV, NR and N denote the number of the trucks, the maximum number of routes for a truck in a day (at most 2 in our case) and the sales points including the given filling station as the first one, respectively.

The $0-1$ variables of the model are

- $x^l_{(ij)k}$ - the lth route of the kth truck between the sales points (ij) (primary decision variables),
- ν_i - the service of the ith sales point (secondary decision variables),

and the model parameters:

- c_k - the cost of the kth truck (HUF/km),
- cp_k - the capacity of the kth truck,
- t_k - the working hours of the kth truck (hour),
- v_k - the speed of the kth truck (km/hour),
- p_i - the priority of the ith sales point in a scale $1-100$,
- d_i - the demand of the ith sales point,
- b_i - the service time in the ith sales point,
- $|(i,j)|$ - the distance of the ith and jth sales points (km),

where $i,j = 1,\ldots,N$, $k = 1,\ldots,NV$ and $l = 1,\ldots,NR$.

In the case of the filling station, $b_1 = 0$, $d_1 = 0$, $p_1 = 0$, $|(1,1)| = 0$. Our multiple vehicle routing problem can be formulated as:

$$\max K\sum_{i=1}^{N} p_i\nu_i - \sum_{i,j=1}^{N}\sum_{k=1}^{NV}\sum_{l=1}^{NR}|(i,j)|c_k x^l_{(i,j)k} \tag{24}$$

$$\sum_{i=1}^{N}\sum_{k=1}^{NV}\sum_{l=1}^{NR} x^l_{(i,j)k} - \nu_j = 0, \quad j = 2,\ldots,N, \tag{25}$$

$$\sum_{i,j=1}^{N} x^l_{(i,j)k} d_j \le cp_k, \quad k = 1,\ldots,NV, \quad l = 1,\ldots,NR, \tag{26}$$

$$\sum_{i,j=1}^{N}\sum_{l=1}^{NR}(|(i,j)|/v_k + b_j)x^l_{(i,j)k} \le t_k, \quad k = 1,\dots,NV, \tag{27}$$

$$\sum_{i=1}^{N} x^l_{(i,j)k} = \sum_{i=1}^{N} x^l_{(j,i)k}, \quad j = 2,\dots,N, \quad k = 1,\dots,NV, \quad l = 1,\dots,NR, \tag{28}$$

$$\sum_{j=2}^{N} x^l_{(1,j)k} \le 1, \quad k = 1,\dots,NV, \quad l = 1,\dots,NR, \tag{29}$$

$$x^l_{(i,j)k} \in S, \quad i,j = 1,\dots,N, \quad k = 1,\dots,NV, \quad l = 1,\dots,NR, \tag{30}$$

where K is a suitably large constant and S represents the subtour breaking constraints ([30], pp. 84)

The aim of optimization is to maximize a utility function based on customer priority and to minimize the total routing transportation costs of all the trucks, where an item related to a truck is the distance covered a day (km) multiplied by its transportations cost/km. Constraints (25) give a relation between the routes and services, (26) capacity bounds for the trucks, (27) working time bounds, (28) a law of nodes, (29) vehicle availability conditions and (30) conditions for avoiding subcycles. Such problems depend considerably on the constraints, therefore, the algorithms are mainly heuristic, based on the special structural properties.

Let me note, that in a typical problem in our case ($NV \approx 10,\ NR = 2,\ N \approx 140$) the model (after using reduction techniques) resulted in a mixed integer linear programming problem with 23 thousand variables and 20 thousand constraints. Of course, it is hopeless to solve such problems to optimality due to complexity issues. Nevertheless, to solve the continuous relaxation of the problem is very important, due to the fact, that it provides a lower bound on the exact optimum value, which can be used in heuristic methods [12, 30].

To demonstrate my observations, I show the nonzero pattern of a typical problem (called `route`) in Figure 11. The dimensions of the problem are $m = 20894, \quad n = 23923$ and the number of nonzeros is $nz = 187686$. The longest column has 44 nonzeros which reflects a very low density (0.21%). It is noteworthy that the partitioning heuristics described in the section 5 classifies this problem to be solved by the augmented system. Next, in Figure 12 I show the nonzero pattern of the factorization of the augmented system with the normal equations approach and with the heuristics discussed previously. On the left hand side of Figure 12 the result with the normal equations (using the minimum degree ordering) can be seen. The procedure generated 3281093 nonzeros in the lower triangular part of the factored form of (14). The right hand side of Figure 12 shows the nonzero pattern of the factorization of the augmented system using the heuristics of section 5. In this case the number of nonzeros is 444206. By using this factorization the solver BPMPD was able to solve the problem in 230 seconds on a powerful IBM PC [49]. In our software package [22] we use linear programming solver and vehicle routing heuristic algorithm to solve the optimization problems, database management to help in completing the organization and geographical tools for the visualization of the results.

Figure 11: Nonzero pattern of problem `route`

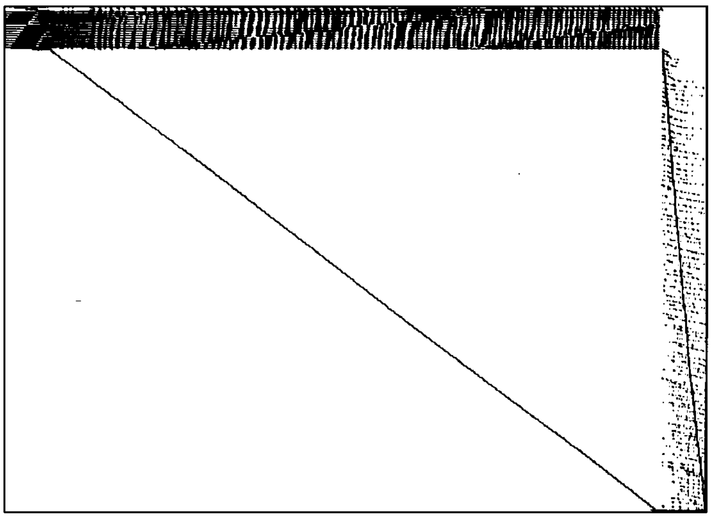
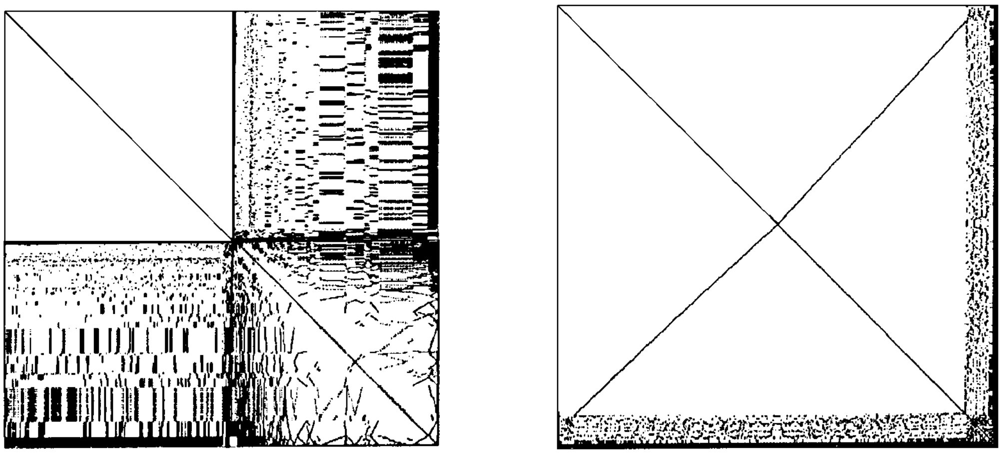


Figure 12: Nonzero pattern of the factorization via NE and AS pivot agenda



8.2 Two–stage stochastic linear programming problems

Stochastic programming handles mathematical programming problems where some of the given parameters are random variables. The importance of stochastic programming is that a more realistic mathematical model can be formulated when the uncertainty is presented. As a result of the wide variety of applications, a number of different stochastic optimization models appeared in the literature. An important one is as follows

$$
\begin{array}{ll}
\min & c^T x \\
\text{subject to} & \hat{A}x = b, \\
& \text{“}T(\omega)x = h(\omega)\text{”}, \\
& x \geq 0,
\end{array}
\tag{31}
$$

where ω is from the probability space Ω, $\hat{A} \in R^{m_1 \times n_1}$ is a matrix, $T(\omega) \in R^{m_2 \times n_1}$ is a random matrix, $c, x \in R^{n_1}$ and $b \in R^{m_1}$ are vectors and $h(\omega) \in R^{m_2}$ is a random vector. Let me note that (31) is meaningless from decision making point of view. The different decision principles, according to the various applications, make (31) right problem. Many real applications, from manufacturing through transportation and financial optimization can be formulated as (31) with the "two–stage" decision principle. In this decision principle the decisions are made in two stages and the observation of the random parameters takes place in between.

In practice, the two–stage stochastic linear programming model can be solved as a linear programming (LP) problem if the probability space Ω has finite number of elements. However, the size of the deterministic problem grows linearly with the number of realizations. Consequently, the effort to handling more uncertainty increases both the model size and complexity. Such problems have always been a challenge for the developers. The birth of the standard input format of stochastic problems [8] and the existence of a public accessible test problem library [35] gave also an impetus for researchers to implement and compare algorithms. As a consequence of the size explosion, numerous methods have been developed to solve the problem by exploiting their specialities. The reader interested can find a detailed and profound summary about the computational methods of stochastic programming in [7].

By the two–stage principle, we make some intermediate decision in the first stage of problem (31) without the knowledge of the random parameters. After observing uncertainty (presented by the realization of the random parameters), we correct the decision of stage 1. The aim in the first stage is to find x which can be corrected for all possible realizations, furthermore the sum of the first stage objective and the expected value of the correction cost is minimal:

$$
\begin{array}{ll}
\min & c^T x + E(\eta(\omega, x)) \\
\text{subject to} & \hat{A}x = b, \\
& x \geq 0.
\end{array}
\tag{32}
$$

The correction cost is also called as "recourse function" and it can be obtained as

$$
\begin{array}{ll}
\eta(\omega, x) = & \min\ d(\omega)^T y \\
\text{subject to} & W(\omega)y + T(\omega)x = h(\omega), \\
& y \geq 0.
\end{array}
\tag{33}
$$

The expression on the right hand of (33) for fixed ω and x is a linear programming problem, where $y \in R^{n_2}$ denotes the correction vector, $d(\omega)$ is a n_2 vector and $W(\omega)$ is a $m_2 \times n_2$ matrix. Usually, x and y are referred to as first and second stage decision vectors, and $W(\omega)$ is called the recourse matrix. We consider the case if the probability space is of finite number of elements. Let denote d_i, W_i, T_i and h_i for $i = 1, ..., k$ the possible realizations of $d(\omega)$, $W(\omega)$, $T(\omega)$ and $h(\omega)$ with $p_1, p_2, \ldots, p_k$ probabilities, then the solution of the large–scale linear programming problem

$$\begin{array}{llllll} \min & (c^T x + p_1 d_1^T y_1 + \ldots + p_k d_k^T y_k) & & & & \\ \text{Subject to} & \hat{A}x & & & & = b, \\ & T_1 x & +W_1 y_1 & & & = h_1, \\ & \vdots & & \ddots & & \vdots \\ & T_k x & & & +W_k y_k & = h_k, \\ & & & x \geq 0, \ y_1 \geq 0, \ldots, y_k \geq 0 & & \end{array} \tag{34}$$

gives the optimal solution of the first stage and all possible second stage problems in one step.

We note that the matrix in (34) has $m_1 + km_2$ rows and $n_1 + kn_2$ columns. This results in very large but sparse linear programming problems where the attention to the structure can be very important. Some different important classes of the general problem (34) are identified in the literature. If W is not random, i.e., $W_1 = W_2 = \cdots = W_k$, then the problem has "fixed recourse". If the second stage of a fixed recourse problem is feasible for arbitrary right–hand side vectors, then we say that the problem has a "complete recourse".

It is a special case when $W = [I, -I]$ is called a "simple recourse" problem. By this, the columns of the second stage decisions indicate upper and lower bounds on the dual problem of (34). Let $\hat{x}$ and $\hat{y}_1, \ldots, \hat{y}_k$ denote the dual variables of (34), then the reduced dual problem can be written as

$$\begin{array}{ll} \max & b^T \hat{x} + \sum_{i=1}^{k} h_i \hat{y}_i \\ \text{Subject to} & \hat{A}^T \hat{x} + \sum_{i=1}^{k} T_i^T \hat{y}_i \leq c, \\ & p_i d \geq \hat{y}_i \geq -p_i d \quad \text{for } i = 1, \ldots k. \end{array} \tag{35}$$

Thus, the remaining problem has only n_1 rows and $m_1 + km_2$ columns, and can be solved more efficiently than (34). Let us observe that in the particular case when $T_1 = T_2 = \cdots = T_k$, problem (35) can be regarded the equivalent of a piecewise linear programming problem with the variable replacement method. Due to the fact that an equivalent piecewise linear programming problem has only n_1 rows and $m_1 + m_2$ columns, a specialized simplex method to piecewise linear programs seems to be the most powerful tool to solve the simple recourse problem in this particular case.

However, if W is a general matrix, no such successful reduction is possible. Therefore handling and exploiting the special structure of the problem is important when applying either interior point or simplex techniques to solve the deterministic equivalent problem. It is the general observation that among the interior point approaches, the augmented system variant gives the best performance on two–stage stochastic linear programming problems [16].

In context of interior point methods, the important partitioning (18) exists, namely the columns of the second–stage decision variables play A_1, the columns of the first–stage decision variables play A_2, while A_3 is usually empty. But, in some of the cases it is imaginable that as result of the structure, the difference between the sparsity of A_1 and A_2 is not significant. This means that in certain cases the two–stage stochastic linear programming problem can be solved efficiently with the normal equation system. It is the case especially in the multi–stage stochastic programming problems where the structure of the constraint matrix may more often vary depending on the number of stages and realizations. Therefore the analyzer which examines the structure of the matrix and decides about the use of either the augmented factorization or the standard Cholesky factorization plays an important role.

In [59] I compared the efficiency of my solver (called BPMPD) and the implementation of Mehrotra (called as FO1AUG) [24, 16]. This latter uses the traditional Bunch-Parlett factorization with 2×2 pivots. This solver is generally considered as the state–of–the–art presentative of the augmented system based implementation of interior point methods. For the comparison of the efficiency of the two implementations, I prepared Table 6 containing the larger problems from the collection of D. Holmes [35] and the stochastic generalized network problems provided by Mulvey and Vladimirou [63]. As a common, platform I used a Sun Sparc-2000 workstation in the experiments. Both of the solvers were compiled with the Sun f77 Fortran compiler using the -fast -O3 compiler options. The Fortran source codes were configured to use at most 128 Mbyte of memory.

Table 6 shows the comparison of the overall performance of the two implementations. Figures given include the number of iterations to optimality, the total solution time and the primal objective function values. All timing results are given in seconds, and reflect on the pure algorithmic time, without the input of the MPS file and the time spent on produce an output.

In all cases, except from `scfxm1-2r-256` on which FO1AUG ran out of memory after 15 iterations, the final objective values obtained in the two implementations match in at least 8 significant digits. I believe that it demonstrates the numerically robustness of the two solvers, and confirms the correct optimal values of the test problems. The difference in the number of iterations seems insignificant as well, but the solution times are much different. Its reason is that BPMPD more successfully avoid the numerical difficulties than FO1AUG. These results confirm the superiority of using the 1×1 pivot scheme to the traditional Bunch-Parlett factorization.

Table 6: Comparison of BPMPD and FO1AUG performance

Problem	Iterations		Solution time		Primal objective	
Name	BPMPD	FO1AUG	BPMPD	FO1AUG	BPMPD	FO1AUG
sc205-2r-200	12	13	8.39	11.52	-10.0704935	-10.0704935
sc205-2r-400	13	14	18.57	26.63	-10.0704935	-10.0704935
sc205-2r-800	16	15	48.94	60.74	-10.0704935	-10.0704935
scagr7-2b-4	13	13	0.21	0.29	-832941.326	-832941.327
scagr7-2b-16	16	16	1.12	1.40	-832902.149	-832902.150
scagr7-2b-64	27	25	51.31	43.39	-832901.547	-832901.547
scfxm1-2r-96	40	34	192.48	749.85	2891.70611	2891.70611
scfxm1-2r-128	39	33	267.17	1622.39	2891.70612	2891.70611
scfxm1-2r-256	45	*15	736.86	*1833.83	2891.70612	*3976.95358
scfxm1-2b-64	41	37	270.23	1076.39	2877.56386	2877.56386
scrs8-2r-128	12	17	4.38	9.05	1177.34504	1177.34504
scrs8-2r-256	14	16	10.64	18.11	1144.16062	1144.16062
scrs8-2r-512	14	16	24.18	38.80	1074.09934	1074.09934
scrs8-2c-64	12	15	1.51	3.73	111.881190	111.881190
scrs8-2b-64	11	14	1.62	3.55	1121.45123	1121.45123
scsd8-2r-108	14	14	33.53	29.86	23.6999943	23.6999944
scsd8-2r-216	15	14	79.03	60.11	23.3999839	23.3999839
scsd8-2r-432	15	16	175.75	158.50	25.7999695	25.7999695
scsd8-2c-64	9	6	67.66	38.84	14.9992320	14.9992320
scsd8-2b-64	9	6	68.62	38.53	15.9986680	15.9986680
sctap1-2r-108	16	19	25.87	36.35	248.000000	248.000000
sctap1-2r-216	17	20	58.92	70.91	249.000000	249.000000
sctap1-2r-480	19	23	152.80	188.64	248.500000	248.500000
sctap1-2c-64	16	17	11.78	14.21	200.390625	200.390626
sctap1-2b-64	19	22	77.99	100.48	484.560000	484.560000
fxm2-16	24	26	27.39	58.05	18416.7590	18416.7590
pltexpA2-16	17	16	7.36	7.71	-9.66330824	-9.66330825
stormG2-8	39	50	94.33	149.13	15535236.6	15535236.6
stormG2-27	59	65	471.28	505.48	15508983.2	15508983.2
deter1	23	26	48.09	60.34	-2.55756435	-2.55756435
deter3	20	25	60.33	82.11	-2.10779072	-2.10779072
deter5	21	25	42.05	53.06	-2.26122357	-2.26122358
deter7	22	25	58.23	67.18	-2.22856797	-2.22856797

* On problem `scfxm1-2r-256` FO1AUG ran out of memory after 15 iterations

8.3 Approximation in L_1 and L_∞ norms under nonnegativity restrictions

In many real–life applications appears the problem to determine the "best" nonnegative solution vector x that satisfies a set of overdetermined system of linear equations as close as possible. The nonnegativity requirement can best be satisfied if the problem is reformulated as an optimization problem. The deviation $||Ax - b||$ can be measured by different vector norms leading to different optimization problems. They turned out to be very difficult problems for the general purpose optimizers. This is reflected in bad numerical behavior and very large solution times. The source of the difficulties lies in the special structure of these problems.

In general, we can write this mathematical programming problem as follows:

$$\begin{aligned} \min \quad & \|\hat{A}x - b\|_* \\ & x \geq 0, \end{aligned} \tag{36}$$

where $\hat{A}$ is an $m \times n$ real matrix, $b \in R^m$ vector and $\|.\|_*$ represents any vector norm. We investigate the possibilities in cases of the L_1 and L_∞ norms.

The use of these norms leads especially to troubles due to the fact that the objective of problem (36) is not differentiable. For both cases problem (36) can be written as a linear programming problem. In the L_1 case, we introduce new non–negative variables d_i^+ to balance the positive violation of constraint i and d_i^- to balance the negative violation of constraint i, respectively. After this, the LP model can be written as follows:

$$\begin{array}{rcl} \min & & e^T d^+ + e^T d^- \\ [\,\hat{A} \quad -I \quad I\,] \begin{bmatrix} x \\ d^+ \\ d^- \end{bmatrix} & = & [\,b\,], \\ x & \geq & 0, \\ d^+ & \geq & 0, \\ d^- & \geq & 0, \end{array} \tag{37}$$

where $e^T = (1, ..., 1)$. It is to be noted that with this formulation we increase the size of problem (36) by $2m$ variables and $2m$ nonzeros. By many applications the approximation problem is strongly overdetermined. This means that the number of rows in $\hat{A}$ is much larger than the number of columns. By this class of problems, the partition (18) of problem (37) is naturally exists, namely $[-I, I]$ plays A_1, $\hat{A}$ plays A_2, while A_3 and A_4 are empty.

The L_∞ case can be formulated as an LP model, as well. We need one additional variable (denoted by d) to the existing ones. It will hold the value of the maximal violation. The LP model of the L_∞ minimization can be written as follows:

$$
\begin{array}{rcl}
& \min & d \\
[\, \hat{A} \quad -I \quad I \,] \begin{bmatrix} x \\ d^+ \\ d^- \end{bmatrix} & = & [\, b \,], \qquad (38) \\
\begin{bmatrix} I & & -1 \\ & I & -1 \end{bmatrix} \begin{bmatrix} d^+ \\ d^- \\ d \end{bmatrix} & \leq & \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad (39) \\
x & \geq & 0, \\
d^+ & \geq & 0, \\
d^- & \geq & 0.
\end{array}
$$

It is to be noted that with this formulation we increase the size of problem (36) by $2m + 1$ new variables, $2m$ slack variables and by $2m$ new constraints. The increase in the number of the nonzeros is $5m$. Additionally, there is an other important structural difference from the previously formulated LP model, namely the structure of a column corresponding to the new variable d. It is a dense column, with $2m$ nonzero entries. Therefore in the partition (18) A_1 includes the columns from (38) corresponding to $\hat{A}$ and the last (dense) column from (39) corresponding the variable d. All the other columns from (38) and (39) belong to A_2, and thus, A_3 and A_4 are empty.

In [50] we discussed the computational implementation of these approaches to solve image reconstruction problems and reported on numerical experiments.

The advantage of the augmented system on these types of approximation problems was particularly brought to our attention in respect of a few industrial models which were under investigation by our group at Brunel University. The research system FortMP [51] developed by the group has sparse simplex and predictor-corrector IPM capabilities. This latter uses the normal equations approach, incorporates supernode elimination strategy to take advantage of vector processing [62] and is also adapted to run on SIMD parallel computers [33]. Yet, in case of these industrial models FortMP/IPM on parallel machines was unable to match computational results achieved by the augmented system approach of my solver BPMPD using a powerful IBM PC [49].

9 Summary

My Ph.D. research concerned the algorithmic issues of the interior point methods. My basic idea at the very beginning was to break with the "traditional" normal equations approach and elaborate the increased freedom of the augmented system. An important point in my research was to realize that these approaches complement each other. To exploit the advantages of both the augmented system and normal equations approaches, a new heuristics was developed which is able to recognize the structure of the linear programming problems and to make algorithmical decisions. Both in qualitative sense and computationally, my method shows its advantage in comparison with any other method applied so far.

The efficient use of the augmented system approach raised special questions. Actually, the interior point methods of linear programming with upper/lower bounds on variables and ranges on constraints can be handled using quasidefinite matrices [72]. Unfortunately, linear programs, as formulated usually do not fit into this form. The other method used in context of solving indefinite system of equations is the Bunch–Parlett factorization, but it tend to be more computationally burdensome. I developed a new approach for computing a symmetric decomposition of an indefinite system. It does not require the quasidefinite property (and the bounded variable and range constraint formulation of the linear programming problem), but it uses 1×1 pivots during the decomposition. As such, my approach can be regarded as the common generalization of the Bunch-Parlett decomposition and the quasidefinite theory, where the good properties of both approaches are preserved. I view this as main contribution of my Ph.D. research.

I investigated all important parts of the efficient implementation of the "linear algebra kernel" of interior point methods. A new technique was proposed for symbolical ordering. It gives a connection between the minimum degree and minimum local fill–in heuristics through a parametrized set of algorithms. The computational results supported that the evolved technique works well in the practice and improves both of the original ordering methods.

The importance of stucture–exploitation during numerical computations was recognized. The method of the supernodal elimination was applied in the framework of the left looking Cholesky factorization. I reported on remarkable decrease in the factorization time by using this technique.

An important part of my research is the applications of the developed techniques on practical problems. By special network problems the advantageous structure for the augmented system was recognized. I made a systematic computational and theoretical investigation of interior point methods for solving two–stage stochastic linear programming problems. The efficiency of the new techniques was demonstrated on an important large scale goal programming problems which are hard for traditional approaches.

Further research

Todays the area of interior point methods for linear programming is deeply explored. The 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 this 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 [32]. 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.[37]. 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. 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 the common conclusion is that current IPM implementations work efficiently, there exist LP problems that are very sparse but produce surprisingly dense symmetric factorizations, e.g.: `dfl001` 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.

Acknowledgments

I would like to express my sincere thanks to my supervisor, István Maros without whose help this work could never have been done. I wish to thank my colleagues at the Hungarian Academy of Sciences, Department of Operation Research and Decision Support Systems for giving attention on my Ph.D. study. I'm thankful to those people with whom I jointly worked, namely to E. Andersen at the University Odense, to J. Gondzio at the University of Geneva, to G. Mitra at the Brunel University, to T. Szántai at the University of Technology Budapest, to Gy. Sonnevend at the Eötvös Loránd University Budapest, to T. Terlaky at the University of Technology Delft and to X. Xiaoije at the Academia Sinica for their valuable comments and instructions during our discussions and joint works. My Ph.D. research was partly supported by Hungarian Research Found OTKA 2587.

References

[1] M. Al-ftise, Cs. Mészáros, and T. Koltay. Linear programming as a tool to solve the problem of project tardiness. *Periodica Polytechnica*, 1995. to appear.

[2] E.D. Andersen, J. Gondzio, C. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. To appear in: Interior point methods in mathematical programming, T. Terlaky (ed.), Kluwer Academic Publisher, 1996.

[3] 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.

[4] 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.

[5] 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.

[6] C.C. Ashcraft, R.G. Grimes, J.G. Lewis, B.W. Peyton, and H.D. Simon. Progress in sparse matrix methods for large scale linear systems on vector supercomputers. *The Internat. J. of Supercomputer Appl.*, 1(4):10–30, 1987.

[7] J.R. Birge. Current trends in stochastic programming computation and application. Technical report, Department of Industrial and Operations Engineering, University of Michigan, 1995.

[8] J.R. Birge, M.A.H. Dempster, H.I. Gassman, E.A. Gunn, A.J. King, and S.W. Wallace. A standard input format for multiperiod stochastic linear programs. *COAL newsletter*, 17:1–20, 1987.

[9] 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.

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

[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., New York, 1976.

[12] L. Bodin, B. Golden, A. Assad, and M. Ball. Routing and scheduling of vehicles and crews, the state of the art. *Computers & Operations Research*, 10:63–211, 1983.

[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] 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.

[16] J. Czyzyk, R. Fourer, and S. Mehrotra. A study of the augmented system and column–splitting approaches for solving two–stage stochastic linear programs by interior point methods. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Illinois, 1994.

[17] 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.

[18] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, 1986.

[19] 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.

[20] N.G. Esmond and B.W. Peyton. A supernodal Cholesky factorization algorithm for shared-memory multiprocessors. *SIAM Journal on Scientific Computing*, 2(14):761–769, 1993.

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

[22] F. Fölsz, Cs. Mészáros, and T. Rapcsák. Distribution of gas cylinders. *European J. Oper. Res.*, 87:613–623, 1995.

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

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

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

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

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

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

[29] 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.

[30] B. Golden and A. Assad. *Vehicle routing: methods and studies.* North-Holland, Amsterdam, The Netherland, 1988.

[31] 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.

[32] 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.

[33] H. Hafsteinsson, R. Levkovitz, and G. Mitra. Solving large scale linear programming problems using an interior point method on a massively parallel simd computer. *Journal of Parallel Algorithms and Applications*, 4(3&4):301–316, 1994.

[34] 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.

[35] D. Holmes. A collection of stochastic programming problems. Technical report, Deptartment of Industrial and Operations Engineering, The University of Michigan, 1994.

[36] J.K. Hurd and F.M. Murphy. Exploiting special structure in primal dual interior point methods. *ORSA Journal on Computing*, 4(1):38–44, 1992.

[37] 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.

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

[39] 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.

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

[41] 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.

[42] J.W.-H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11(2):141–153, 1985.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] I. Maros and Cs. Mészáros. A numerically exact implementation of the simplex method. *Annals of Operations Research*, 58:3–17, 1995.

[49] 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.

[50] I. Maros and Cs. Mészáros. Vector norm minimization approaches to image reconstruction problems. Technical report TR/11/95, Brunel University, Department of Mathematics and Statistics, London, 1995.

[51] I. Maros and G. Mitra. Simplex algorithms. In J. Beasley, editor, *Recent Advances in Linear and Integer Programming*. Oxford University Press., 1996.

[52] 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.

[53] 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.

[54] 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.

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

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

[57] Cs. Mészáros. Basis inverse of large-scale and sparse linear programming problems. *Alkalmazott Matematikai Lapok*, 16:279–310, 1992. in Hungarian.

[58] Cs. Mészáros. On the modifications of the affine scaling algorithm for linear programming. *Alkalmazott Matematikai Lapok*, 17:185–194, 1993. in Hungarian.

[59] 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.

[60] 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.

[61] Cs. Mészáros. Fast Cholesky factorization for interior point methods of linear programming. *Computers & Mathematics with Applications*, 31(4/5):49–54, 1996.

[62] G. Mitra and R. Levkovitz. Solution of large linear programs: A review of hardware, software and algorithmic issues. In T. Ciriani and R. Leachman, editors, *Optimization in Industrial Environments*, pages 139–171. John Wiley, 1993.

[63] J.M. Mulvey and H. Vladimirou. Stochastic network optimization models for investment planing. *Annals of Operations Research*, 20:187–217, 1989.

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

[65] 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.

[66] 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.

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

[68] 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.

[69] 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.

[70] M.J. Todd. Exploiting special structure in Karmarkar's linear programming algorithm. *Mathematical Programming*, 41:81–103, 1988.

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

[72] R.J. Vanderbei. Symmetric quasi-definite matrices. *SIAM J. on Optim.*, 5(1), February 1995.

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

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

[75] Z. Zlatev. On some pivotal strategies in Gaussian elimimination by sparse techniques. *SIAM Journal on Numerical Analysis*, 17(1):18–30, 1980.