Presolve Analysis of Linear Programs Prior to Applying an Interior Point Method*

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

Technical Report 1994.3
February 7, 1994, revised December 20, 1994

Abstract

Several issues concerning an analysis of large and sparse linear programming problems prior to solving them with an interior point based optimizer are addressed in this paper. Three types of presolve procedures are distinguished. Routines from the first class repeatedly analyze an LP problem formulation: eliminate empty or singleton rows and columns, look for primal and dual forcing or dominated constraints, tighten bounds for variables and shadow prices or just the opposite, relax them to find implied free variables. The second type of analysis aims at reducing a fill-in of the Cholesky factor of the normal equations matrix used to compute orthogonal projections and includes a heuristic for increasing the sparsity of the LP constraint matrix and a technique of splitting dense columns in it. Finally, routines from the third class detect, and remove, different linear dependecies of rows and columns in a constraint matrix.

Computational results on problems from the *Netlib* collection, including some recently added infeasible ones, are given.

Key words. Presolving, linear programs, interior point methods.

1 Introduction

This paper deals with several techniques of presolve analysis for large scale linear programs. We assume that an interior point method is applied to solve them and that a normal equations approach is used to compute orthogonal projections. We shall refer to a primal linear programming (LP) problem

$$\text{minimize } c^T x, \tag{1}$$
$$\text{subject to } Ax = b, \tag{2}$$
$$l \leq x \leq u, \tag{3}$$

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

$$\text{maximize } b^T y + l^T z - u^T w, \tag{4}$$
$$\text{subject to } A^T y + z - w = c, \tag{5}$$
$$z, w \geq 0, \tag{6}$$

*Supported by the Fonds National de la Recherche Scientifique Suisse, grant #12-34002.92.
†on leave from the Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland

where $y \in \mathcal{R}^m$ and $z, w \in \mathcal{R}^n$. In these formulations we shall sometimes call equations (2) and (5) the primal and the dual constraints, respectively.

Many linear programs are formulated in a way that is not necessarily the most suitable for a direct application of an LP solver. It is known, almost since the first applications of linear programming, that a presolve analysis of the problem before passing it to a solver often reduces solution time considerably. There are exceptions to this rule in the case of a simplex type optimizer: they usually result from the presence of degeneracy that may sometimes induce longer path to the optimum in the reduced problem. In the case of an interior point based optimizer, the situation seems simpler. The computational effort in interior point methods is dominated by the orthogonal projections. These computations take usually 60% to 90% of the overall CPU time to solve the problem.

In general, the smaller (sparser) constraint matrix A, the cheaper are the orthogonal projections. (An exception to this rule of thumb is an "unsplit" technique that eliminates doubleton equality row and causes a concatenation of two short columns into a longer one.) Additionally, as interior point methods show a "more deterministic behavior" than the simplex, we do not expect an increase of the iteration number for a reduced problem. Unless the problem is irreducible, time spent in the presolve analysis will almost surely payoff in terms of overall CPU time.

The issue of presolve analysis has been addressed by many authors [2, 4, 6, 20, 29]. In this paper we discuss several known presolve techniques, we extend some of them, simplify the others and, finally, show how they all work in practice. We distinguish three groups among them.

Techniques from the first group repeatedly analyze the LP problem: they eliminate empty or singleton rows and columns, look for primal or dual forcing or dominated constraints, tighten bounds for variables and shadow prices or, just to the opposite, relax these bounds to find implied free variables. Most of these techniques are already known, at least since the paper of Brearley et al. [6]. We survey them and discuss several extensions in Section 2. We also show that some of them, although they lead to a problem size reduction, may result in an LP formulation that is less suitable for an interior point solver.

The second type of analysis aims at reducing the effort to compute the Cholesky factor of the matrix of normal equations. In its full generality, the problem of finding the sequence of Gaussian elimination operators that reduce a matrix A to its sparsest form, is known to be an NP complete problem [2], [22]. The same problem applied to AA^T, as it is required in interior point methods, is even more difficult. We thus propose in Section 3 a cheap and easy to implement heuristic that finds a (pivot) row of A whose sparsity pattern is a subset of those of the other rows. The pivot row is then used to eliminate nonzero entries from all rows with the superset sparsity pattern. Such an approach can never produce fill-in. We also recall in Section 3 the technique of splitting dense columns [15], [31], and later show limits of its application.

Finally, in Section 4 we address the problem of identifying linear dependencies in the matrix A. We show that our heuristic for making A sparser naturally eliminates linearly dependent rows. We also discuss the problem of identifying and, when possible, aggregating linearly dependent columns. Duplicate columns that mimic split free variables are known to cause considerable difficulties to interior point algorithms if they are not handled with a special care.

Since the interior point algorithm is applied to a reduced form of the problem, it is important to reconstruct a full primal–dual pair of solutions to the original problem. Although the most of operations that perform this task are straightforward, we find it useful to recall them. We discuss this problem in Section 5.

Presolve analysis techniques discussed in this paper have all been incorporated into our higher order primal-dual interior point code HOPDM [3]; this results in a considerable improvement of the efficiency of its new version 2.0. From now on, we shall always refer to version 2.0 of HOPDM. In Section 6, we briefly describe our implementation of the primal-dual method and in Section 7 we show that when run on the Netlib [13] suite of 91 problems, our code looses only 74 iterations to OB1.60 (a state-of-the-art implementation of Lustig et al. [20], version of September 1993 of a predictor-corrector primal-dual method of Mehrotra [25]). This means that we loose in the average one iteration per problem, a loss of efficiency less than 5%. We also demonstrate (on infeasible problems that have recently been added to the Netlib set) the HOPDM's ability to early detect problem's infeasibility.

2 Logical analysis of linear programs

In this section, we shall concentrate on these aspects of presolve analysis that aim to reduce the problem dimension by deleting rows and columns and tightenning bounds on variables and shadow prices. For the ease of presentation we shall assume that an LP problem is both primal and dual feasible and that an optimal solution to it exists. Detecting unboundedness or infeasibility naturally follows, in most cases, from the analysis. If, for example, a new implied bound on a variable x_j (or a shadow price y_i) contradicts the already existing ones, then the problem is primal (or dual) infeasible. Let us mention that some of the techniques discussed here already appeared in the paper of Brearley, Mitra and Williams [6]. We survey them all and propose useful extensions, e.g.: improving bounds on shadow prices, eliminating weakly dominated columns, detecting implied free variables (see e.g., [4]) and generating finite bounds for free variables (a particularly useful technique for an interior point optimizer).

2.1 Empty rows and columns

Depending on the type of the constraint and of the sign of a right hand side component, an empty row must either be redundant or infeasible.

Depending on the presence of finite bounds and the sign of the objective coefficient, a variable refering to an empty column is either fixed to one of its bounds, or the problem is unbounded (or dual infeasible).

Observe that empty rows/columns did not necessarily have to be so in an original formulation of the problem. They usually appear in the intermediate stages of the presolve analysis.

2.2 Singleton rows

An equality type singleton row uniquely determines the value of a variable that appears in it. If this value is beyond the variable's bounds, then the problem is infeasible. An inequality type singleton row introduces a new bound on an appropriate variable. Such a bound can be redundant, tighter than already existing one or infeasible.

Our rule is to eliminate all singleton rows. If the row is of equality type, then an appropriate variable is fixed and eliminated from the problem. This elimination often creates new singleton rows and the process is repeated. To implement it efficiently, we need a comfortable access to all singleton rows. Therefore we build a linked list of singleton rows and we update it after every column elimination.

2.3 Forcing and redundant constraints

Assume all LP variables have explicit (possibly infinite) bounds (3). These bounds can be used to determine the following lower and upper limits for a constraint i

$$\underline{b_i} = \sum_{j \in P_{i*}} a_{ij} l_j + \sum_{j \in N_{i*}} a_{ij} u_j, \tag{7}$$

and

$$\overline{b_i} = \sum_{j \in P_{i*}} a_{ij} u_j + \sum_{j \in N_{i*}} a_{ij} l_j, \tag{8}$$

where $P_{i*} = \{j : a_{ij} > 0\}$ and $N_{i*} = \{j : a_{ij} < 0\}$.

For any x satisfying (3) we then have

$$\underline{b_i} \le \sum_j a_{ij} x_j \le \overline{b_i}. \tag{9}$$

We shall now concentrate on one particular type "less than or equal to" constraint. A similar analysis applies to a "greater than or equal to" row. Let us also observe that any equality type constraint can be replaced (for the purpose of this analysis only) with two inequalities, like an inequality row with a finite range (see e.g., [28]). We then assume that row i of (2) originally had the form

$$\sum_j a_{ij} x_j \le b_i. \tag{10}$$

The following four possibilities may occur:

- $b_i < \underline{b_i}$. Constraint (10) cannot be satisfied; the problem is infeasible.
- $\underline{b_i} = b_i$. Constraint (10) is *forcing*. The only way to satisfy it is to fix all variables refering to positive entries ($j \in P_{i*}$) to their lower bounds and to fix all variables refering to negative entries ($j \in N_{i*}$) to their upper bounds. Constraint (10) and the abovementioned variables are eliminated.
- $\overline{b_i} \le b_i$. Constraint (10) is *redundant*; row i can be eliminated.
- $\underline{b_i} < b_i < \overline{b_i}$. Constraint (10) cannot be eliminated. This case is nonconstructive.

2.4 Tightening variable bounds

The nonconstructive case

$$\underline{b_i} \le b_i \le \overline{b_i} \tag{11}$$

of the previous section can often be used to improve the bounds on variables involved by (10). (For the sake of simplicity, we continue to restrict our analysis to a "less than or equal to" type constraint.)

Equations (7) and (10) imply

$$\forall k \in P_{i*}, \qquad \underline{b_i} + a_{ik}(x_k - l_k) \le \sum_j a_{ij} x_j \le b_i, \tag{12}$$

and

$$\forall k \in N_{i*}, \qquad \underline{b_i} + a_{ik}(x_k - u_k) \le \sum_j a_{ij} x_j \le b_i. \tag{13}$$

This gives new implied bounds on variables involved by row i

$$x_k \le u_k^{'} = l_k + (b_i - \underline{b_i})/a_{ik} \qquad \forall k \in P_{i*}, \tag{14}$$

and

$$x_k \ge l_k^{'} = u_k + (b_i - \underline{b_i})/a_{ik} \qquad \forall k \in N_{i*}. \tag{15}$$

These new bounds can be *constructive* (in which case we tighten appropriate bound), *redundant* or *contradictory* to the existing bound (in which case the problem is infeasible).

The analysis presented above is correct under the assumption that $\underline{b_i}$ and l_k or u_k for $k \in P_{i*}$ or $k \in N_{i*}$, respectively, are finite. To our knowlegde, this is the way in which it has been implemented in [4], [6] and [20].

We go a step further and show that in some cases finite implied bounds can be derived from (7) and (10) even when $\underline{b_i}$ is not finite.

If there exists only one infinite bound in (7), say $l_k = -\infty$ for some $k \in P_{i*}$ or $u_k = +\infty$ for some $k \in N_{i*}$, then still an implied bound on x_k (upper one, if $k \in P_{i*}$ or lower one, if $k \in N_{i*}$) can be derived. In such a case (12) and (13) are replaced by the following inequalities

$$a_{ik}x_k + \sum_{j \in P_{i*}-\{k\}} a_{ij}l_j + \sum_{j \in N_{i*}} a_{ij}u_j \le \sum_j a_{ij}x_j \le b_i, \qquad \text{if} \quad k \in P_{i*}, \tag{16}$$

or

$$a_{ik}x_k + \sum_{j \in P_{i*}} a_{ij}l_j + \sum_{j \in N_{i*}-\{k\}} a_{ij}u_j \le \sum_j a_{ij}x_j \le b_i, \qquad \text{if} \quad k \in N_{i*}, \tag{17}$$

giving

$$x_k \le u_k^{'} = \Big(b_i - \sum_{j \in P_{i*}-\{k\}} a_{ij}l_j - \sum_{j \in N_{i*}} a_{ij}u_j\Big)/a_{ik}, \qquad \text{if} \quad k \in P_{i*}, \tag{18}$$

and

$$x_k \ge l_k^{'} = \Big(b_i - \sum_{j \in P_{i*}} a_{ij}l_j - \sum_{j \in N_{i*}-\{k\}} a_{ij}u_j\Big)/a_{ik}, \qquad \text{if} \quad k \in N_{i*}, \tag{19}$$

respectively.

The use of this technique seems particularly important if the problem contains free variables. We recall that the classical approach applying (12) and (13) in unable to generate finite limits $\underline{b_i}$ and $\overline{b_i}$ for all rows in which an entry in a free column appears. It is thus unable to generate an implied bound for a free variable. With our approach we do generate such bounds; we eliminate free variables from all problems in the Netlib collection, except three models: PEROLD, PILOT-JA and PILOT4. (The presolver of [4] leaves free variables in twelve problems.)

2.5 Implied free variables

The implied bounds determined by the technique described in the previous section, can be used in a different manner. If a given LP constraint introduces bounds on a variable k that are at least as tight as the original bounds l_k and u_k, then the original bounds can be removed and the variable becomes free. If a simplex based optimizer is used, this effect is beneficial since free variables enter the basis at the beginning and stay in it until optimum is reached [5]. It would undoubtedly be highly undesirable for an interior point solver (at least the one using normal equations approach to compute orthogonal projections).

However, if an implied free variable refers to a singleton column, it may be removed from the problem formulation since its elimination does not affect any other constraint. An LP constraint that implies bounds on a variable can also be eliminated in such a case: it becomes a free row. However, this constraint cannot be forgotten as it conveys information about the linear dependency of the eliminated variable with respect to the variables that remain in the reduced problem. This information will be used to recover the optimal value of an implied free variable from the solution of the reduced problem. The last issue in the elimination of a row, say row i, with an implied free variable x_j, is to make sure that the objective function of the reduced problem matches the original objective function. The analysis of a dual constraint (5) indicates that whenever we get a singleton implied free variable x_j, it determines a unique (feasible) value for the associated dual variable

$$y_i = c_j / a_{ij}. \tag{20}$$

The easiest way to take the contribution of x_j into account is to modify the objective function

$$c' = c - a_{i*}^T y_i, \tag{21}$$

where a_{i*} denotes row i of A.

Although this presolve technique seems rather complicated, it can be implemented in an efficient routine. Fast access to singleton columns, with potentially implied free variables, is ensured by a linked list. All eliminated rows are reordered to the last positions in A before the number of rows of a reduced problem is set up. These bottom rows are stored in the same LP data structure but they are inactive in the optimization process. They are used in a postsolve analysis to compute the value of a free variable and to restore the original objective function.

2.6 Bounds on shadow prices

All presolve techniques discussed so far have been concerned with the analysis of a primal formulation (1)-(3) of the linear program. Some of them can be applied in a similar way to the dual (4)-(6).

Let us start from an observation that the optimality conditions impose a sign constraint on all reduced costs $z - w$ associated with variables that have at least one infinite bound

$$w_j = 0 \qquad \text{if} \quad u_j = +\infty, \tag{22}$$

$$z_j = 0 \qquad \text{if} \quad l_j = -\infty. \tag{23}$$

In such a case the associated constraint in (5) becomes an inequality, either

$$a_{*j}^T y \le c_j, \tag{24}$$

or

$$a_{*j}^T y \ge c_j, \tag{25}$$

respectively. (a_{*j} denotes column j of A.) If a_{*j} is a singleton column with an entry a_{ij}, then each such inequality can be used to produce a bound on the shadow price y_i.

It is usual to apply this technique at least to all primal slack variables added to the inequality constraints without finite ranges, i.e., to all unbounded slacks. However, we also apply it to those structural singleton columns without a finite bound that could not have been made implied free earlier by the technique of Section 2.5.

2.7 Dominated variables

Assume all dual variables y have explicit (possibly infinite) bounds

$$p_i \le y_i \le q_i, \qquad i = 1, 2, \cdots, m. \tag{26}$$

We can use these bounds to define the following lower and upper limits for the dual constraints

$$\underline{c_j} = \sum_{i \in P_{*j}} a_{ij} p_i + \sum_{i \in N_{*j}} a_{ij} q_j, \tag{27}$$

and

$$\overline{c_j} = \sum_{i \in P_{*j}} a_{ij} q_i + \sum_{i \in N_{*j}} a_{ij} p_j, \tag{28}$$

where $P_{*j} = \{i : a_{ij} > 0\}$ and $N_{*j} = \{i : a_{ij} < 0\}$.

Any feasible y clearly satisfies

$$\underline{c_j} \le \sum_i a_{ij} y_i \le \overline{c_j}. \tag{29}$$

In a subsequent part of this section and also in the next section, we shall assume that the variable x_j has no finite upper bound, i.e., the corresponding dual constraint has the form (24). The following possibilities may thus occur:

- $c_j < \underline{c_j}$. Constraint (24) cannot be satisfied: the problem is dual infeasible.
- $\overline{c_j} < c_j$. The induced cost z_j is strictly positive so the variable x_j is *dominated*: it can be fixed at its lower bound $x_j = l_j$. If, however, this lower bound is not finite, then the probem is unbounded.
- $\overline{c_j} = c_j$. Constraint (24) is always sasfied with $z_j \ge 0$.
- $\underline{c_j} \le c_j < \overline{c_j}$. The variable x_j cannot be eliminated.

If the third case occurs, the variable x_j a called *weakly dominated.* It is claimed in [4] that the variable can be eliminated like a strongly dominated variable ($\overline{c_j} < c_j$). Below, we prove a more general condition that qualifies a weekly dominated variable for elimination. Our result holds also in the case when bounds on the shadow prices (26) have been tightened by the technique described in the next section (and not necessarily derived directly from (24) or (25) for singleton columns).

Proposition 1. *Assume $l_j > -\infty$, $u_j = +\infty$ and $\overline{c_j} = c_j$. If for all k in P_{*j} the LP constraint k is of the 'less than or equal to' type, and for all k in N_{*j} the LP constraint k is of the 'greater than or equal to' type, then there exists an optimal solution to (1)-(6) such that*

$$x_j^* = l_j \tag{30}$$

or the linear program has no optimal solution.

Proof. The weak dominance condition $\overline{c_j} = c_j$ and equations (5), (22), (28) and (29) yield

$$\begin{aligned} z_j^* &= c_j - \textstyle\sum_i a_{ij} y_i^* \\ &\ge c_j - \overline{c_j} = 0. \end{aligned}$$

We may distinguish two possibilities: $z_j^* > 0$ and $z_j^* = 0$. In the first case, we have by complementarity equation (30).

Assume the second case holds and an optimal solution to (1)-(3) exists, such that $x_j^* > l_j$. We shall show that whenever such an optimal solution x^* exists, it may be pushed to another optimal solution $\hat{x}$ satisfying (30).

Let us partition the primal variables x into x_j, the slack part x_S associated with the slack variables added to all constraints that involve x_j, and the rest of x called x_R. Next, let us partition an LP constraint matrix accordingly

$$A = \begin{bmatrix} a_{*j} & A_R & A_S \end{bmatrix}.$$

An optimal solution $x^* = (x_j^*, x_R^*, x_S^*)$ with $(x_j^* > l_j)$ clearly satisfies (2). Define

$$\hat{x} = \left(\; l_j, \quad x_R^*, \quad x_S^* + A_S^{-1} a_{*j}(x_j^* - l_j) \; \right). \tag{31}$$

The assumptions of the proposition imply that all elements of $A_S^{-1} a_{*j}$ are positive so

$$\hat{x}_S > x_S^* \geq 0.$$

Point $\hat{x}$ of (31) thus satisfies (2) and (3) so it is also optimal, which completes the proof.

Let us observe that the asumptions on the type of all LP constraints that involve x_j play a crucial role in the proof as they allow pushing x^* to $\hat{x}$ satisfying (30). In our implementation the assumptions are checked while computing limits (27) and (28). On the other hand, we do not bother where the bounds on the shadow prices come from. They do not necessarily have to be generated by singleton columns, as it was required in [4]. The column j is thus weakly dominated by some, in general unknown, subset of columns.

An analogous condition can be given for the second type of weak domination ($\underline{c}_j = c_j$).

Proposition 2. *Assume $l_j = -\infty$, $u_j < +\infty$ and $\underline{c}_j = c_j$ If for all k in P_{*j} the LP constraint k is of the 'greater than or equal to' type, and for all k in N_{*j} the LP constraint k is of the 'less than or equal to' type, then there exists an optimal solution to (1)-(6) such that*

$$x_j^* = u_j$$

or the linear program has no optimal solution.

The proof is similar to that of Proposition 1.

2.8 Improving bounds on shadow prices

Assume that $l_j > -\infty$ and $u_j = +\infty$. We stated in the previous section that if $\underline{c}_j \leq c_j < \overline{c}_j$, then nothing could be said about the optimal value of x_j. However, we can still apply the same technique as in Section 2.4 to derive new, possibly better, bounds on the dual variables. From (24), (27) and (29) we get

$$\forall k \in P_{*j} \qquad \underline{c}_j + a_{kj}(y_k - p_k) \leq \sum_i a_{ij} y_i \leq c_j, \tag{32}$$

and

$$\forall k \in N_{*j} \qquad \underline{c}_j + a_{kj}(y_k - q_k) \leq \sum_i a_{ij} y_i \leq c_j. \tag{33}$$

These inequalities imply the new bounds on y_k

$$y_k \leq q_k^{'} = p_k + (c_j - \underline{c}_j)/a_{kj} \qquad \forall k \in P_{*j}, \tag{34}$$

and

$$y_k \geq p_k^{'} = q_k + (c_j - \underline{c_j})/a_{kj} \qquad \forall k \in N_{*j}, \tag{35}$$

respectively. If the new bound improves the older one, we call it *constructive*. If it is *redundant*, we omit it. Finally, if it is *contradictory*, the problem is dual infeasible.

Let us observe that we have implicitly assumed that all p_k for $k \in P_{*j}$ and q_k for $k \in N_{*j}$ and consequently $\underline{c_j}$ of (27) are finite. However, it is still possible to obtain a constructive bound on y_k applying the technique similar to that of Section 2.4, even in the case when one bound p_k for some $k \in P_{*j}$ or q_k for some $k \in N_{*j}$ is infinite.

In our implementation we count the number of infinite bounds on y_k while computing $\underline{c_j}$ and $\overline{c_j}$. The computational overhead to catch a case of a single infinite bound is thus negligible. Applying this technique usually produces bounds on free dual variables associated with equality constraints. Once such bounds have been determined, they allow obtaining better limits $\underline{c_j}$ and $\overline{c_j}$; consequently, they open the possibility of eliminating more dominated (or weakly dominated) columns.

3 Making normal equations sparser

In this section we shall be concerned with different presolve techniques that aim at reducing the computational effort of a single interior point iteration. We shall look for an equivalent, but not necessarily reduced, LP problem formulation that is more suitable for an application of an interior point optimizer.

We assume that a normal equations approach is used to find the orthogonal projections. We also assume that Cholesky decomposition

$$LL^T = A\Theta A^T,$$

where Θ is some diagonal scaling matrix, has to be computed at every iteration of the algorithm. A competitive — augmented system approach [10], [12], [30], [32] — has some advantages over the normal equations one: it naturally handles free variables and dense columns of A and easily extends to nonseparable quadratic programming. However, it is considerably less efficient, in the average, [12].

To our knowledge, Adler et al. [2] were the first to improve the sparsity of L by increasing the sparsity of A. Independently, [7] and [22] addressed a more general problem of finding a nonsingular matrix M such that MA is the sparsest possible. Both [2] and [7] propose rather expensive algorithms to reduce the number of nonzeros of A.

We propose a considerably simpler, easy to implement heuristic that makes A sparser after relatively little analysis effort.

The normal equations approach used to compute orthogonal projections considerably suffers from the presence of dense columns in A. Such columns create (subject to a symmetric row and column permutation) completely dense blocks *dense windows* in $A\Theta A^T$ matrices and in their Cholesky factors. Dense columns may be handled by the Schur complement mechanism [8], iterative method [1] or a combination of the two [19].

In our implementation we split [15], [31] dense columns, i.e., we cut them into shorter pieces.

3.1 Making A sparser

The basic idea of our heuristic for improving the sparsity of A is to analyze every equality type row, say, a_{i*} and to look for all LP constraints with the sparsity pattern being the superset of that of a_{i*}. If a_{k*} is such an LP constraint, then, for any real α,

$$a_{k*}^{'} = a_{k*} + \alpha a_{i*}, \tag{36}$$

has the same sparsity pattern as a_{k*}. The linear program in which a_{k*} is replaced by $a_{k*}^{'}$ and b_k is replaced by $b_k^{'} = b_k + \alpha b_i$, is clearly equivalent to the original one.

An appropriate choice of α can cause elimination of at least one nonzero entry from a_{k*}, or more, if additional, unexpected cancellations occur, thus producing sparser $a_{k*}^{'}$. No extra storage is required to implement this technique as it can never produce fill-in, which clearly simplifies its implementation. The main difficulty is a potentially time consuming search for constraints with superset sparsity pattern to a given one a_{i*}. To reduce the computational effort we choose the shortest column a_{*j} of all that have an entry in row i and examine only rows that have a nonzero element in this column. Observe that if row k is not a 'superset' of row i, then it is usually rejected after analyzing its first two or three elements or even before, if the number of its nonzero entries is less than that of a_{i*}.

Summing up, after a relatively little analysis effort, we find all superset rows to a given one and eliminate at least one nonzero entry from every one of them.

In our implementation of this technique we build a linked list of all row pivot candidates, i.e., all equality type rows. This list is ordered with an increasing number of nonzero entries in the rows. Short rows are more probable to be subsets of other ones. Each row pivot candidate is removed from the list after it has been analyzed and any equality type row in which a cancellation occured reenters the list.

This technique to make A sparser is not optimal. Its choice of the pivot row is based on the analysis of the sparsity pattern of A only. Consequently, our heuristic is unable to find two rows with different sparsity patterns (none of them being subset of the other), such that the linear combination (36) produces limited fill-in and, at the same time, cancells more nonzero entries [2, 7]. However, our computational experience indicates that, although we do not exploit such a possibility (its detection would involve floating point operations and would require considerably more effort), a simple heuristic proposed in this paper produces comparably good results as more involved algorithms.

3.2 Splitting dense columns

The idea of splitting dense columns has its origin in stochastic programming [21], [27]. When applied to general linear programs [15], [31] it consists of cutting dense columns into shorter pieces and replacing one large dense window of $A\Theta A^T$ with several smaller ones that can be easier accommodated by the Cholesky decomposition.

A dense column d of A with n_d nonzero elements is replaced with a set of columns $d_1, d_2, \ldots, d_k$ such that

$$d = \sum_{i=1}^{k} d_i. \tag{37}$$

Consistency is obtained by adding the linking constraints

$$x_d^i - x_d^{i+1} = 0, \qquad i = 1, 2, \ldots, k-1, \tag{38}$$

to force the variables $x_d^1, x_d^2, \ldots, x_d^k$ associated with all the pieces d_i to be the same. We have no longer large dense block of size $n_d \times n_d$ created by the dense column d (subject to a symmetric row and column permutation of $A\Theta A^T$). Instead, the new adjacency structure $\tilde{A}\tilde{\Theta}\tilde{A}^T$ of a transformed problem contains k smaller dense windows resulting from pieces d_i, for which we pay with a need of dealing with $k-1$ additional constraints (38).

Splitting has a nice property as it does not introduce rank deficiency into $\tilde{A}$, a feature that neither the Schur complement mechanism [8] nor the preconditioned conjugate gradient approach [1] share. On the other hand, our experience indicates that its applicability is restricted to a case when the number of dense columns is small.

The crucial issue of its implementation is the appropriate distribution of nonzeros of d in d_i of (37). Arbitrary splitting into columns that are no longer than a given threshold has been proposed in [31]. Further reduction of the number of nonzero entries in the adjacency structure $\tilde{A}\tilde{\Theta}\tilde{A}^T$ and in the Cholesky factor can be obtained after careful analysis of the sparsity pattern of a sparse part of matrix A [15]. A heuristic implemented in our code is based on such an analysis. It looks for a compromise between the following two goals. First, we choose the short pieces d_i in such a way that the small and dense windows they create add the minimum number of the new nonzero entries to the adjacency structure. Secondly, we look at the number of $\tilde{A}\tilde{\Theta}\tilde{A}^T$ nonzero entries that do not belong to the new dense blocks and keep this number as small as possible. The reader interested in more detail in the implementation of this technique is referred to [15].

Let us observe that any equality type doubleton row can be transformed to the form (38). Some LP codes, e.g., OSL of IBM [11], offer as an option, a presolve technique to eliminate doubleton rows. This operation is clearly opposite to splitting: it causes a concatenation of two shorter columns into a longer one but it may be advantageous if the length of the new column is not excessive.

4 Linearly dependent rows and columns

The presence of linearly dependent rows in A causes rank deficiency of $A\Theta A^T$ matrices. Commercial LP codes are robust enough to handle dependent equations, especially if their number is small. However, if their number is excessive, the Cholesky factorization becomes very sensitive to the ordering: it sometimes results in numerical difficulties for the implemented interior point algorithm. It is thus clearly desirable to detect and eliminate dependent rows before the solution starts.

Linearly dependent columns are in general less dangerous unless they refer to a difference of two nonnegative and unbounded variables

$$x_F = x_{j_1} - x_{j_2}, \tag{39}$$

in which case they model some free variable $x_F \in (-\infty, +\infty)$. Primal-dual method (such as probably any other interior point algorithm) drives x_F close to its optimal value but at the same time increases both x_{j_1} and x_{j_2} to infinity. Both Θ_{j_1} and Θ_{j_2} grow in such a case very quickly and cause loss of accuracy in normal equations that sometimes makes convergence impossible.

4.1 Linearly dependent rows

Identification of all linearly dependent rows in a linear program would require applying some stable variant of Gaussian Elimination to A, so it is not a practicable approach for large and sparse problems. Several authors have thus tried to detect at least the simplest linearly dependent rows, i.e., duplicate rows [4, 29], as a search for them can take full advantage of the sparsity of A. By duplicate rows, we mean, as in [29], two rows a_{i*} and a_{k*} for which a real α exists such that

$$a_{k*} = \alpha a_{i*}. \tag{40}$$

Let us observe that our heuristic of Section 3.1 to make A sparser naturally detects such a situation if at least one of the constraints i and k is of the equality type. Assume constraint i is an equality type row. The algorithm to make A sparser finds the pattern of a_{k*} to be a superset of that of a_{i*}. It then tries to eliminate one nonzero from a_{k*} and, due to unexpected cancellations, finds out that a'_{k*} is empty.

Even more important, however, is the algorithm's ability to naturally detect nontrivial linear dependencies of rows of A that involve more than two rows. Such a possibility exists unless all linearly dependent LP constraints have completely different sparsity patterns, none of them being subset of any other.

We end this section with two simple examples of problems with linearly dependent rows:

Example 1.

$$\begin{array}{rrrcr} x_1 & & +x_3 & = & 2, \\ & x_2 & +x_3 & = & 3, \\ x_1 & -x_2 & & = & -1. \end{array}$$

Example 2.

$$\begin{array}{rrrcr} x_1 & & +x_3 & = & 2, \\ & x_2 & +x_3 & = & 3, \\ x_1 & +x_2 & +2x_3 & = & 5. \end{array}$$

Both problems are feasible: point $(1, 2, 1)$ satisfies all equations. A routine that looks only for duplicate rows (40) is unable to detect the linear dependency in any of them. Our algorithm to make A sparser will identify it at least in Example 2.

4.2 Duplicate columns

We identify all pairs of columns a_{*j} and a_{*k} for which a real α exists such that

$$a_{*k} = \alpha a_{*j}. \tag{41}$$

An algorithm that performs this is similar to that for making A sparser. For a given column a_{*j} we choose the shortest row a_{i*} among all those with an entry in a_{*j}. Next, we analyze all columns a_{*k} that have a nonzero entry in a_{i*} and check if they are duplicates of a_{*j}. Column candidates with different number of nonzero entries than that of a_{*j} are rejected before entering to a relatively expensive check of condition (41). An analysis if (41) is satisfied requires at most τ multiplications, where τ is a number of nonzeros of a_{*j}, but it usually terminates earlier if (41) is not fulfilled. Summing up, we are able to identify all duplicate columns after a moderate analysis effort.

The way of treating duplicate columns depends on a sign of α, the relation of c_j and c_k and the presence of finite bounds on x_j and x_k.

If $c_k = \alpha c_j$, then, in general, we replace variables x_j and x_k with an aggregate one

$$x'_k = x_k + \alpha x_j. \tag{42}$$

Bounds of this new variable depend on the sign of α

$$l'_k = l_k + \alpha l_j \quad \text{and} \quad u'_k = u_k + \alpha u_j \quad \text{if} \quad \alpha > 0,$$

or

$$l_k' = l_k + \alpha u_j \quad \text{and} \quad u_k' = u_k + \alpha l_j \quad \text{if} \quad \alpha < 0.$$

Let us observe that if $\alpha < 0$ and both lower bounds l_j, l_k or both upper bounds u_j, u_k are infinite, then (42) becomes a free variable. It is highly desireable to detect such a situation and to treat x_j and x_k in a special way, i.e., to prevent their growth to infinity, within the interior point algorithm.

If $c_k \neq \alpha c_j$, then, under some additional assumptions, one of the columns dominates the other. Let us focus our attention on

$$c_k \geq \alpha c_j. \tag{43}$$

An analogous analysis applies to the oposite case. From (41), (43) and (5) we get

$$\begin{aligned} z_k^* - w_k^* &= c_k - a_{*k}^T y^* \\ &> \alpha c_j - \alpha a_{*j}^T y^* \\ &= \alpha(z_j^* - w_j^*). \end{aligned} \tag{44}$$

Now, if $u_j = +\infty$ (i.e., $w_j^* = 0$) and $\alpha > 0$, or $l_j = -\infty$ (i.e., $z_j^* = 0$) and $\alpha < 0$, then (45) yields

$$z_k^* - w_k^* > 0.$$

Variable x_k can then be fixed to its lower bound unless $l_k = -\infty$, in which case the problem is dual infeasible.

5 Recovering solutions of the original problem

A user of a general purpose LP solver obviously expects the problem solution to be given in terms of the original constraints and variables and not in terms of some mysteriously reduced problem. It is thus important to be able to recover complete primal and dual optimal solutions in the original problem formulation from those of the equivalent reduced problem. We shall call this operation a postsolve analysis.

To be able to perform it efficiently, one has to store some trace information on all presolve modifications done to the primal and dual variables, respectively. This is acomplished in our implementation by means of a stack-type *presolve history list*. The postsolve analysis scans the presolve history list in the reverse order and "undoes" subsequent primal or dual variable transformations.

Two important remarks apply to the use of the presolve history list. First, it is advantageous to separate it into two independent lists that keep track of the primal and dual variable transformations, respectively. Secondly, all entries of these lists have a very simple form of a triple built of two integers and one real element. Hence storage requirements to handle them remain very modest.

Primal history list

The primal–dual algorithm is applied to the LP formulation

$$\text{minimize } c^T x, \tag{45}$$

$$\text{subject to } Ax = b, \tag{46}$$

$$x + s = u, \tag{47}$$

$$x, s \geq 0, \tag{48}$$

that differs slightly from the original one. All variables of the original formulation are pushed to their zero lower bound and the upper bound of (3) is modified accordingly. The original variable $x \in [l, u]$ is thus replaced with $\tilde{x} = x - l \in [0, \tilde{u}]$, where $\tilde{u} = u - l$. Once an optimal solution to the modified problem $\tilde{x}^*$ is found, a solution of the original problem is retrieved from

$$x^* = l + \tilde{x}^*.$$

The presolve analysis can modify variable bounds. If an upper bound is tightened, then we only have to save its new value in the appropriate data structure. Any modification of the zero lower bound to some value l_j' needs a trace to be left in a primal history list. We save the variable index j and the correction of its lower bound, l_j'. The LP problem is modified accordingly

$$b' = b - a_{*j} l_j'.$$

To keep track of variable aggregations one has to save indices of the duplicate variables, k and j and the real α of (41).

Finally, to save information on the elimination of the implied free variable (Section 2.5) one needs to remember an index of the variable j and the value of the dual variable (20). Recall that this presolve technique causes modification of the objective function (21) that will have to be undone.

We need a simple code that is able to distinguish the above situations. We use integers in a triple (i_1, i_2, κ) to implement it. A lower bound corrector is saved as $(i_1 = j, i_2 = 0, \kappa = l_j')$. A variable aggregation is remembered as $(i_1 = k, i_2 = j, \kappa = \alpha)$. Finally, an implied free variable is stored as $(i_1 = j, i_2 = -i, \kappa = y_i)$. Hence, a sign of the second integer uniquely determines the type of information stored.

After the primal optimal solution of the reduced problem $\tilde{x}$ is found, we scan the primal history list in the reverse order and "undo" subsequent presolve modifications. The following two types of presolve corrections: the lower bound push and the implied free variable elimination, have inverse operations uniquely defined. Note that in a case of a singleton free variable we have saved the value of a dual variable $y_i^* = \kappa$ and we have changed the objective function.

The undo operation for an aggregate variable is not well defined. In fact we have much freedom in the choice of x_j and x_k. Their optimal values have only to satisfy

$$x_k^* + \alpha x_j^* = \tilde{x}_k^*,$$

and stay within the original bounds.

Dual history list

The modifications of the dual variables y have a less complicated character. The only operations that affect them are the computations of linear combinations of two constraints when improving the sparsity of A (Section 3.1). Equation (36) applies a Gaussian elementary operator to rows i and k of matrix A. We use again a triple, two integers and a real value $(i_1 = i, i_2 = k, \kappa = \alpha)$, to store all the information necessary to be able to undo this operation.

The product of Gaussian elementary operators is a nonsingular matrix M that is never explicitly formulated. Recall that we replaced the original constraint matrix A with a sparser one $\tilde{A} = MA$. Consequently, the optimal dual variables of the transformed problem satisfy

$$A^T M^T \tilde{y}^* + z^* - w^* = \tilde{c},$$

and the optimal dual variables of the original problem are

$$y^* = M^T \tilde{y}^*.$$

The reader interested in more detail in the organization of the computations that involve a sequence of Gaussian elementary operations is referred to the excellent book of Duff et al. [9].

To complete the discussion of the postsolve procedure let us observe that some particular presolve modifications need some additional variable transformations to be done.

The dual variables corresponding to empty rows can be set to any value, for example 0. The dual variables for the linking constraints added by the splitting technique are ignored. The dual variables for (eliminated) singleton rows are chosen so as to satisfy $z_j^* - w_j^* = 0$ for a fixed column with an entry in a singleton row.

Once all dual variables y^* have been determined we pass to the calculation of the reduced costs $z - w$. Whenever possible, the reduced cost is calculated directly from (5). Next, depending on the sign of $z - w$, one of the variables z or w is set to zero and the other to the nonnegative value $|z - w|$.

6 Primal-dual method

All presolve techniques presented in this paper have been incorporated into a new version 2.0 of the HOPDM [3] code. In this section we shall very briefly address several issues of a primal-dual logarithmic barrier method - an interior point algorithm implemented in the HOPDM optimizer. The first theoretical results for this method come from [18] and [24]. Early implementations of it can be found in [8, 23]. For a current state-of-the-art implementations of the primal-dual method see [19, 20, 25, 26, 11].

Originally, the version 1.0 of the HOPDM code implemented the higher order primal-dual interior point method of Mehrotra [26] and used second or third order central trajectory approximations. Its new version 2.0, uses a second order predictor-corrector technique implemented in a manner like [19].

To derive the algorithm we replace nonnegativity constraint (48) with the logarithmic barrier penalty term in the objective function, which gives the following Lagrangian

$$L(x, s, y, w, \mu) = c^T x - y^T(Ax - b) + w^T(x + s - u) - \mu \sum_{j=1}^n \ln x_j - \mu \sum_{j=1}^n \ln s_j. \quad (49)$$

The first order optimality conditions for (49) give

$$\begin{array}{rcl} Ax &=& b, \\ x + s &=& u, \\ A^T y + z - w &=& c, \\ XZe &=& \mu e, \\ SWe &=& \mu e, \end{array} \quad (50)$$

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

A single iteration of the basic primal-dual algorithm makes one step of the Newton's method applied to conditions (50) for a given μ

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & 0 & I & 0 & 0 \\ 0 & A^T & 0 & I & -I \\ Z & 0 & 0 & X & 0 \\ 0 & 0 & W & 0 & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \\ \Delta z \\ \Delta w \end{bmatrix} = - \begin{bmatrix} Ax - b \\ x + s - u \\ A^T y + z - w - c \\ XZe - \mu e \\ SWe - \mu e \end{bmatrix}, \tag{51}$$

and updates (usually decreases) μ.

Mehrotra [25] composed the direction step $(\Delta x, \Delta s, \Delta y, \Delta z, \Delta w)$ (denoted with Δ for short) from two parts

$$\Delta = \Delta_a + \Delta_c. \tag{52}$$

The affine scaling predictor direction Δ_a solves a linear system like (51) for a right hand side equal to the current violation of the first order optimality conditions for (45)-(48)

$$r_a = \left(\; b - Ax, \quad u - x - s, \quad c - A^T y - z + w, \quad -XZe, \quad -SWe \; \right)^T. \tag{53}$$

Next, the maximum primal and dual stepsizes, α_{Pa} and α_{Da}, preserving the nonnegativity of (x, s) and (z, w) respectively are determined and the predicted complementarity gap

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

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

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

where $g = x^T z + s^T w$ denotes the current complementarity gap. For such a μ, one computes the corrector direction Δ_c that solves an equation like (51) for a right hand side

$$r_c = \left(\; 0, \quad 0, \quad 0, \quad \mu e - \Delta X_a \Delta Z_a e, \quad \mu e - \Delta S_a \Delta W_a e \; \right)^T, \tag{54}$$

and one determines the direction step Δ of (52). Finally, one evaluates the maximum stepsizes in the primal and dual spaces that maintain nonnegativity of the variables; the stepsizes are then slightly reduced with a factor $\alpha_0 = 0.99995$ and a new iterate is computed. This last operation involves separate stepsizes in the primal and dual spaces.

A single iteration of the second order predictor-corrector primal-dual method needs thus two solves of the same large, sparse linear system (51) for two different right hand sides (53) and (54). In our implementation, (51) is reduced to the normal equations form

$$A \Theta A^T \Delta y = h, \tag{55}$$

where $\Theta = (X^{-1}Z + S^{-1}W)^{-1}$. Next, a sparse Cholesky factorization [16] of $A\Theta A^T$ is computed and we use it twice in solves for the predictor and corrector components of (52).

We still use a starting point that approximately solves an auxilliary QP problem [3]. However, to prevent the negative influence of the presence of large upper bounds in (47) that draw the initial point far away from the origin, we replaced all upper bounds larger than 10^3 with this value when computing $(x^0, s^0, y^0, z^0, w^0)$.

The algorithm terminates when 8-digit accurate solution is obtained, i.e., when

$$\frac{|c^T x - (b^T y - u^T w)|}{1 + |b^T y - u^T w|} \leq 10^{-8}. \tag{56}$$

7 Numerical results

We shall demonstrate in this section practical performance of our presolve procedure applied to solving linear problems from the Netlib collection [13], including the recently added infeasible ones. In fact, we ran HOPDM on a larger suite of tests, some of them of a considerably larger size than the biggest Netlib models and we sometimes obtained spectacular reductions. Unfortunately, some of these problems are proprietary so they could not be made available to other researchers and thus could not be used for comparisons. For this reason we restrict the analysis to the Netlib collection.

In all runs reported in this paper we use the same stopping criterion (56) as [19] and [25], i.e., we terminate after an 8-digit accurate solution is obtained.

There are three major modules to be distinguished in the presolve analysis: CLEAN implements all techniques of the logical analysis discussed in Section 2 and the search for duplicate columns (Section 4.2); MAKESPARSER implements the heuristic of Section 3.1; and SPLIT implements the heuristic recalled in Section 3.2 (see [15] for more detail). These modules are called in the following sequence CLEAN, MAKESPARSER, CLEAN, SPLIT. The second call of CLEAN is clearly skipped if the algorithm to make A sparser could not remove a single nonzero entry from A.

CLEAN consists of five routines: R_RW_SNG that removes row singletons (Section 2.2); EL_CNSTS that improves variable bounds and uses them to eliminate redundant and forcing constraints applying techniques of Sections 2.3 and 2.4; R_CL_SNG that removes column singletons, i.e., implied free variables using the technique of Section 2.5; EL_VRBLS that tightens bounds on shadow prices and eliminates dominated columns using the techniques of Sections 2.6, 2.7 and 2.8; and FD_DUPL that identifies duplicate columns and aggregates them whenever possible (Section 4.2). All routines check for empty rows and eliminate them. EL_VRBLS additionally eliminates empty columns. All routines restore, at termination, the LP data structures in the internal format of HOPDM for a current reduced problem. It clearly implies some loss of efficiency. Restoring the data structures is a desireable feature for an experimental software (like ours). In a production code, one should rather adapt a more efficient approach whereby eliminated rows/columns are only marked out by the appropriate routines, and LP data structures are updated once the whole CLEAN is completed.

We tried several different sequences of calling CLEAN routines. Extensive testing (on about 160 real-life linear programs) showed that the best reductions are obtained by the following one EL_VRBLS, R_RW_SNG, R_CL_SNG, FD_DUPL, EL_CNSTS. This sequence defines one pass of the logical analysis. It is repeated in our CLEAN routine untill no further reduction is obtained.

Our experience indicates that usually the first pass reduces the problem the most. It might thus be advantageous to limit CLEAN to one pass (or at most two). However, we did not exploit this possibility as we wanted to see what are the limits (of the implementation discussed in this paper) of possible problem reductions.

Table 1 demonstrates these reductions for 89 Netlib problems. Its columns 2, 3 and 4 contain problem dimensions before applying presolve routine (but after removing empty rows and fixed columns). Columns 5, 6 and 7 contain problem dimensions after presolve. Column 5 additionally indicates after a comma the number of links added by SPLIT. The last two columns contain the number of nonzero elements removed by the algorithm to make A sparser, MKSP and the SUN SPARC 10 CPU time in seconds spent in the whole presolve analysis.

The results collected in Table 1 vary considerably for different problems. GROW or SCSD problems, for example, are irreducible, AGG, FFFFF800, SCORPION and SHIP models are reduced to about half of their original size, some other problems (BEACONFD, BORE3D, VTP-

BASE) are reduced even more, up to spectacular success on SEBA. The average reductions of the number of rows vary from 10% to 20% and this surely gratifies the presolve effort.

Adler et al. [2] give results of their presolve (CLEAN and SPARSE) for only 35 Netlib problems (see Table I of [2]). They are concerned with inversion of $A^T A$ matrices and apply their presolve procedure to differently defined A. Although this is not stated clearly in their paper, we suppose that they count the nonzeros of the objective as well as the slack entries, while our results purely refer to matrix A of (2). We thus want to warn the reader from drawing too far-fetched conclusions from the comparison that follows. Our presolver obtains better nonzero reductions on 27 problems (for three of them: AGG, BEACONFD and SCORPION the number of nonzeros in a new matrix A is more than two times smaller than that of [2]) and looses on 5 problems. This improvement results mainly from our sophisticated CLEAN routine.

Chang and McCormick [7] report in their Table 7.1 results of HASP (Hierarchical Algorithm for a Sparsity Problem) run after using their less involved CLEAN on 67 Netlib problems. They include objective nonzeros to A, which again makes comparison of their results with ours inconsistent. The presolver described in this paper obtains better reductions for 58 problems (on 34 of them an improvement exceeds 20%, on 8 problems it exceeds 50%) and looses insignificantly for 3 problems. The improvement surely results from our better CLEAN.

Some impressive presolve reductions for the Netlib problems have been obtained by Andersens [4]. A comparison of our results with those of Tables 6 and 7 of [4] reveals that our presolver reaches better nonzeros reductions for 45 of 88 problems and looses on 16 problems. The presolver of [4] is of the CLEAN type, so our improvement mainly results from the presence of the heuristic to make A sparser.

We would like to further comment on the reductions due to the use of the heuristic to make A sparser (MKSP column in Table 1). The heuristic is applied to the problem that has already been reduced by the first pass of CLEAN, hence its reductions are not that impressive, especially when measured with the average number of nonzeros eliminated. However, one has to keep in mind that these eliminations often open the possibility of further reductions achieved by the second pass of CLEAN. Its advantages become much more important on larger problems from beyond the Netlib collection.

The advantages of applying a presolve analysis become clearer if we look closer at the effort to compute the Cholesky factorization of the $A\Theta A^T$ matrix. Table 2 gives such an insight. Its columns contain the number of off-diagonal nonzeros in the adjacency structure AA^T, the number of off-diagonal nonzeros in the Cholesky factor L and the number of *flops* (floating point operations) required to compute L. This data is reported for two cases: when the problem in its original form is solved (empty rows and fixed columns are not counted) and when the problem was first treated with a complete presolve analysis (including the heuristic to make A sparser).

The results collected in Table 2 show that the presolve analysis is able to reduce significantly the effort to compute the Cholesky factorization in a single iteration of the interior point method. This effort was reduced at least twice in the case of 25 problems and it has been reduced more that five times for 8 problems. Let us observe that the splitting of dense columns considerably improved the sparsity of L for ISRAEL; surprisingly enough, it did not help too much in the case of FIT1P. This technique seems attractive only if the number of dense columns is very small.

After the analysis of all computational results presented so far, one can expect considerable savings in the overall time to solve the problems. Surprisingly enough, computational experience does not confirm this.

We have run the HOPDM with and without presolve on the whole Netlib collection. We found that the code looses about a hundred iterations if the presolve is used. Many problems have nevertheless been solved faster as the reduction of the effort to compute Cholesky factors compensated the small increase of the iterations number and the time spent in the presolve analysis. Reduced problems were often numerically easier to solve (recall, for example, that we managed to remove free variables from all but three problems PEROLD, PILOTJA and PILOT4; we also managed to remove a remarkable fraction of the linearly dependent rows from the constraint matrices). Consequently, HOPDM with presolve was able to reach the required 8-digit accuracy for all problems, while HOPDM without presolve could not obtain it for GREENBEA (it terminated with only 3 digits exact). There was, however, a subset of problems (AGG, BNL1, CYCLE, D2Q06C, E226 and SCFXM) that, in a reduced form, required at least five more iterations than in the original formulation.

Quite natural a question in such a case is to identify and disable that presolve technique which slows down the convergence. After extensive testing, we came to the conclusion that tightening variable upper bounds (see Section 2.4) may indirectly be dangerous. In several problems, the presolver generated variable upper bounds for most variables. Some of these bounds were large, often exceeding 10^3. They thus changed the starting point considerably and the primal-dual algorithm needed more iterations to localize the optimum.

Unfortunately, disabling the technique of improving variable upper bounds is not a good solution. The presence of finite bounds u_j helps producing better limits $\underline{b_i}$ and $\overline{b_i}$ (see equations (7) and (8)): it is crucial for eliminating forcing and redundant constraints. Additionally, it opens more possibilities of identifying implied free variables. We thus cannot simply disable it. Consequently, we decided to impose a threshold u_0 on acceptable implied upper bounds and to ignore all bounds larger than it. The value of $u_0 = 10^2$ has been found to be a good compromise and it is now used in a *default* presolve analysis of HOPDM.

Table 3 compares the efficiency of HOPDM and the OB1.60 code of [20], version of September 1993. The results for OB1.60 come from the paper of Xu et al. [33], Table I, as we have no direct access to this implementation. For HOPDM, we report the number of iterations to reach 8-digit optimality and the CPU time on a SUN SPARC 10 workstation in two cases when presolve was disabled (NoP) and when the default presolve was used (WithP). In the latter case, we report the reductions of default presolve and the time spent in the analysis. When reporting CPU times for HOPDM, we have excluded the time for model input and solution output. The time spent in the presolve analysis and in the preprocessing for the Cholesky factorization (with the multiple minimum degree ordering of [14] and with the symmetric factorization [16]) is included in the overall solution time. For OB1.60, we report only the number of iterations to reach the optimum.

These results show that the presolve analysis gives, in the average, reduction of the solution time. The time reductions coincide with the savings in the effort to form the Cholesky factors. For the reasons explained earlier, there exist a set of 44 problems on which the use of our presolver caused an increase of the iterations number; 29 of these problems are solved faster if our presolve is disabled. On the other hand, if there is no increase of the iterations number, then the reduced problems are usually solved faster. The time savings resulting from the presolve phase exceed 10% for 36 problems. For 16 of these problems, the time savings exceed 20% and in a case of 6 problems they exceed 50%.

The most favorable feature of the code resulting from the presence of the presolver is, however, its good overall performance measured in the number of iterations needed to solve the problems. The comparison with the OB1.60 indicates that on the whole suite of 91 test problems we have lost 74 iterations to it, which gives, in the average, 1 iteration per problem.

Another advantage of the presolve analysis is its ability to detect early infeasibility. A set of 29 infeasible problems has recently been added to the Netlib collection. The results of running HOPDM on them are presented in Table 4. We report in it, in particular, the problem sizes before and after the presolve analysis and the number of primal-dual iterations to identify problem's infeasibility.

Infeasibility of 15 problems has been identified during (default) presolve analysis. This is marked in Table 4 with a zero number of iterations. The problem size after presolve refer in such case to the one at the moment infeasibility has been stated. The dimension of the reduced problem is not necessarily explicitly known in such case, which explains the presence of question marks in Table 4.

8 Conclusions

We have addressed in this paper an important practical problem of the analysis of large and sparse linear program before solving it with the interior point optimizer. Several presolve techniques, (for logical problem analysis, for improving sparsity of A and the Cholesky factor of $A\Theta A^T$ and for identifying and eliminating linearly dependent rows and columns from A) have been discussed. They have all been incorporated into our experimental predictor-corrector primal-dual code HOPDM.

Applying our presolve analysis usually reduces the solution time, sometimes in a considerable way. The exceptions come from irreducible problems and those linear programs for which many large upper bounds are generated in which case the number of iterations to reach optimality increases. This, in our opinion, results from the choice of worse starting point. A high sensitivity of the initial point routine and, consequently, of the solution process to the presence of bounds is a different problem that undoubtedly dimmed advantages of presolve analysis.

The presolve analysis removes much of instability from the problems: it eliminates linearly dependent rows, discovers hidden free variables (split by hand in an original formulation) and generates finite bounds on many free variables. It thus produces problems that are easier to be solved to the desired accuracy.

Finally, a much involved presolve proves particularly useful in a detection of problem's infeasibility.

Summing up, we find the presolve analysis to be a highly useful option that should be added to any interior point optimizer.

Acknowledgement
The author is grateful to Professor Joseph Liu for the possibility of using his multiple minimum degree ordering routine, GENMMD. The author is greatly indebted to Professor Jean-Philippe Vial for help in the preparation of the revised version of this paper.

References

[1] Adler I., N. Karmarkar, M.G.C. Resende and G. Veiga. An implementation of Karmarkar's algorithm for linear programming, Mathematical Programming 44 (1989) pp. 297-336.

[2] Adler I., N. Karmarkar, M.G.C. Resende and G. Veiga. Data structures and programming techniques for the implementation of Karmarkar's algorithm, ORSA Journal on Computing 1 (1989) pp. 84-106.

[3] Altman A. and J. Gondzio. An efficient implementation of a higher order primal-dual interior point method for large sparse linear programs, Archives of Control Sciences 2 (1993), No 1/2, pp. 23-40.

[4] Andersen E.D. and K.D. Andersen. Presolving in linear programming, Technical Report, Department of Mathematics and Computer Science, Odense University, Denmark, August 1993.

[5] Bixby R. Implementing the simplex method: the initial basis, ORSA Journal on Computing 4 (1992) pp. 267-284.

[6] Brearley A.L., G. Mitra and H.P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm, Mathematical Programming 15 (1975) pp. 54-83.

[7] Chang S.F. and S.T. McCormick. A hierarchical algorithm for making sparse matrices sparser, Mathematical Programming 56 (1992) pp. 1-30.

[8] Choi I.C., C.L. Monma and D.F. Shanno. Further development of a primal-dual interior point method, ORSA Journal on Computing 2 (1990) pp. 304-311.

[9] Duff I.S., A.M. Erisman and J.K. Reid. Direct methods for sparse matrices, Oxford University Press, New York, 1987.

[10] Duff I.S., N.I.M. Gould, J.K. Reid, J.A. Scott and K. Turner. The factorization of sparse symmetric indefinite matrices, IMA Journal of Numerical Analysis 11 (1991) pp. 181-204.

[11] Forrest J.J.H. and J.A. Tomlin. Implementing interior point linear programming methods in the Optimization Subroutine Library, IBM Systems Journal 31 (1992) pp.26-38.

[12] Fourer R. and S. Mehrotra. Solving symmetric indefinite systems in an interior point method for linear programming, Mathematical Programming 62 (1993) pp. 15-39.

[13] Gay D.M. Electronic mail distribution of linear programming test problems, Mathematical Programming Society COAL Newsletter 13 (1985) pp. 10-12.

[14] George, A. and J.W.H. Liu. The evolution of the minimum degree ordering algorithm, SIAM Review 31 (1989) pp. 1-19.

[15] Gondzio J. Splitting dense columns of constraint matrix in interior point methods for large scale linear programming, Optimization 24 (1992) pp. 285-297.

[16] Gondzio J. Implementing Cholesky factorization for interior point methods of linear programming, Optimization 27 (1993) pp. 121-140.

[17] Karmarkar N. A new polynomial-time algorithm for linear programming, Combinatorica 4 (1984) 373-395.

[18] Kojima M., S. Mizuno and A. Yoshise. A primal-dual interior point algorithm for linear programming, in: N. Megiddo ed., Progress in Mathematical Programming, Springer-Verlag, New York, 1989, pp. 29-48.

[19] Lustig I.J., R.E. Marsten and D.F. Shanno. On implementing Mehrotra's predictor-corrector interior point method for linear programming, SIAM Journal on Optimization 2 (1992) pp. 435-449.

[20] Lustig I.J., R.E. Marsten and D.F. Shanno. Interior point methods for linear programming - computational state of the art, ORSA Journal on Computing 6 (1994) 1–14.

[21] Lustig I.J., J.M. Mulvey and T.J. Carpenter. Formulating two-stage stochastic programs for interior point methods, Operations Research 39 (1991) pp. 757-770.

[22] McCormick S.T. Making sparse matrices sparser: computational results, Mathematical Programming 49 (1990) pp. 91-111.

[23] McShane K.A., C.L. Monma and D.F. Shanno. An implementation of a primal-dual interior point method for linear programming, ORSA Journal on Computing 1 (1989) pp. 70-89.

[24] Megiddo N. Pathways to the optimal set in linear programming, in: N. Megiddo ed., Progress in Mathematical Programming, Springer-Verlag, New York, 1989, pp. 131-158.

[25] Mehrotra S. On the implementation of a primal-dual interior point method, SIAM Journal on Optimization 2 (1992) pp. 575-601.

[26] Mehrotra S. Higher order methods and their performance, Technical Report 90-16R1, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, July 1991.

[27] Mulvey J.M. and A. Ruszczynski. A diagonal quadratic approximation method for large scale linear programs, Operations Research Letters (1992).

[28] Murtagh B. Advanced Linear Programming: Computation and Practice, McGrew Hill, New York, 1981.

[29] Tomlin J. and J.S. Welch. Finding duplicate rows in a linear program, Operations Research Letters 5 (1986) pp. 7-11.

[30] Turner K. Computing projections for the Karmarkar algorithm, Linear Algebra and its Applications 152 (1991) pp. 141-154.

[31] Vanderbei R.J. Splitting dense columns in sparse linear systems, Linear Algebra and its Applications 152 (1991) pp. 107-117.

[32] Vanderbei R. and T.J. Carpenter. Symmetric indefinite systems for interior point methods, Technical Report SOR 91-7, Department of Civil Engineering and Operations Research, Princeton University, Princeton, New Jersey, 1991.

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

Table 1. Problem size reductions after presolve.

Problem	Before presolve			After maximum presolve			MKSP	time
	m	n	nonz	m	n	nonz		
25fv47	820	1571	10400	768	1534	9957	154	2.07
80bau3b	2235	9301	20413	1965	8736	19048	0	3.33
adlittle	56	97	383	53	95	365	8	0.08
afiro	27	32	83	21	29	72	0	0.03
agg	488	163	2410	163	109	836	0	0.30
agg2	516	302	4284	291	286	2461	22	0.45
agg3	516	302	4300	296	286	2503	22	0.50
bandm	305	472	2494	209	364	1576	141	0.54
beaconfd	173	262	3375	25	76	137	103	0.26
blend	74	83	491	66	73	388	30	0.06
bnl1	632	1175	5121	558	1109	4619	191	0.56
bnl2	2280	3489	13999	1848	3007	12458	284	2.65
boeing1	348	384	3485	294	373	2733	2	0.39
boeing2	140	143	1196	125	143	801	0	0.08
bore3d	233	314	1427	64	89	404	44	0.27
brandy	182	249	2148	117	194	1536	138	0.32
capri	271	337	1735	234	296	1285	20	0.24
cycle	1886	2857	20720	1380	2383	14061	439	2.94
czprob	927	3294	9983	661	2688	5376	0	0.78
d2q06c	2171	5167	32417	2010	4962	30259	776	7.28
d6cube	404	6184	37704	403	5443	32523	1710	9.11
degen2	444	534	3978	444	534	3976	2	0.19
degen3	1503	1818	24646	1503	1818	24363	283	1.28
e226	223	282	2578	161	260	2169	137	0.26
etamacro	400	606	2060	331	540	1846	2	0.35
fffff800	524	854	6227	291	642	4175	607	0.74
finnis	497	569	2092	357	441	1473	0	0.35
fit1d	24	1026	13404	24	1024	13358	28	1.59
fit2d	25	10500	129018	25	10363	127760	0	25.88
fit1p	627	1677	9868	627,11	1688	9868	0	1.06
forplan	135	418	4494	102	367	3994	66	0.62
ganges	1309	1681	6912	835	1168	5423	128	1.43
gfrd-pnc	616	1092	2377	590	1066	2325	0	0.31
greenbea	2389	5302	30715	1848	3886	23112	271	5.67
greenbeb	2389	5290	30676	1846	3876	23014	267	5.50
grow15	300	645	5620	300	645	5620	0	0.12
grow22	440	946	8252	440	946	8252	0	0.23
grow7	140	301	2612	140	301	2612	0	0.11
israel	174	142	2269	166,3	144	2262	0	0.13
kb2	43	41	286	43	41	265	21	0.05
lotfi	153	308	1078	117	282	596	0	0.12
maros	844	1408	9576	626	953	5744	258	1.44
maros-r7	3136	9408	144848	2152	6578	80167	0	4.71
modszk1	686	1620	3168	658	1405	2863	1	0.83
nesm	662	2748	13078	646	2676	12926	0	1.40
perold	625	1312	5832	580	1232	5379	81	1.54

Table 1. (continued). Problem size reductions after presolve.

Problem	Before presolve			After maximum presolve			MKSP	time
	m	n	nonz	m	n	nonz		
pilot	1440	3449	41092	1340	3326	40454	21	11.06
pilot4	410	970	5041	389	878	4719	103	0.96
pilot87	2029	4663	70682	1967	4595	70359	20	14.70
pilot-ja	924	1677	11821	793	1546	10925	152	3.42
pilot-we	722	2711	8862	691	2486	8418	51	3.38
pilotnov	951	1968	12186	830	1871	11492	94	3.52
recipe	87	154	628	61	99	371	21	0.12
sc105	104	103	280	104	103	280	0	0.02
sc205	204	203	551	203	202	550	0	0.09
sc50a	49	48	130	49	48	130	0	0.03
sc50b	48	48	118	48	48	118	0	0.02
scagr25	471	500	1554	341	492	1310	100	0.25
scagr7	129	140	420	89	132	338	28	0.13
scfxm1	330	457	2589	257	412	2087	109	0.67
scfxm2	660	914	5183	515	825	4183	217	1.31
scfxm3	990	1371	7777	773	1238	6279	325	1.97
scorpion	388	358	1426	180	212	581	462	0.32
scrs8	490	1169	3182	418	1089	2725	198	0.71
scsd1	77	760	2388	77	760	2388	0	0.11
scsd6	147	1350	4316	147	1350	4316	0	0.23
scsd8	397	2750	8584	397	2750	8584	0	0.57
sctap1	300	480	1692	269	452	1557	0	0.28
sctap2	1090	1880	6714	977	1768	6159	0	0.76
sctap3	1480	2480	8874	1346	2356	8229	0	1.04
seba	515	1028	4352	0	0	0	0	0.10
share1b	117	225	1151	107	217	990	122	0.13
share2b	96	79	694	92	79	632	51	0.12
shell	536	1525	3056	487	1448	2902	0	0.39
ship04l	360	2118	6332	292	1890	4275	0	0.46
ship04s	360	1458	4352	216	1266	2860	0	0.33
ship08l	712	4283	12802	470	3099	7100	0	0.75
ship08s	712	2387	7114	276	1582	3622	0	0.43
ship12l	1042	5427	16170	610	4147	9230	0	0.99
ship12s	1042	2763	8178	340	1919	4273	0	0.54
sierra	1222	2016	7252	1129	2008	6956	120	1.31
stair	356	385	3666	356	384	3664	0	0.31
standata	358	1059	2974	292	441	1026	0	0.26
standgub	359	1168	3082	292	441	1026	0	0.29
standmps	466	1059	3622	388	1011	2356	0	0.48
stocfor1	117	111	447	94	96	359	0	0.07
stocfor2	2157	2031	8343	1968	1854	7064	0	0.72
stocfor3	16675	15695	64875	15336	14382	55088	0	5.39
truss	1000	8806	27836	1000	8806	27836	0	0.93
tuff	294	584	4517	241	515	3659	10	0.58
vtp-base	197	185	801	46	72	195	23	0.13
wood1p	244	2594	70215	170	1717	43657	916	10.16
woodw	1098	8405	37474	703	5347	19727	60	3.08

Table 2. Effort to compute Cholesky factors.

Problem	Before presolve			After maximum presolve		
	nz(AA^T)	nz(L)	flops	nz(AA^T)	nz(L)	flops
25fv47	11074	33516	2.475212D+06	10454	32236	2.310864D+06
80bau3b	9972	42709	2.559623D+06	9176	36588	1.963226D+06
adlittle	328	355	2.749000D+03	308	340	2.668000D+03
afiro	63	80	2.680000D+02	47	61	2.070000D+02
agg	11183	16629	6.772970D+05	1854	2685	5.700500D+04
agg2	12883	21657	1.119601D+06	5090	8234	3.545800D+05
agg3	12893	21657	1.119601D+06	5153	8105	3.300250D+05
bandm	3419	4430	8.519200D+04	1931	3133	6.356700D+04
beaconfd	2669	2754	6.591000D+04	40	40	7.400000D+01
blend	743	989	1.670700D+04	526	726	9.884000D+03
bnl1	4395	11827	4.755470D+05	3973	10317	3.550990D+05
bnl2	13457	82291	1.246523D+07	12242	81245	1.280746D+07
boeing1	5077	7201	1.854170D+05	3710	5298	1.141180D+05
boeing2	1876	2665	6.133100D+04	1151	1864	3.591200D+04
bore3d	2191	2941	6.809100D+04	561	778	1.261200D+04
brandy	2541	3258	9.530200D+04	1679	2223	5.822300D+04
capri	2841	5746	2.000600D+05	1755	3384	8.208200D+04
cycle	27714	76526	6.098104D+06	18224	52363	3.489537D+06
czprob	6616	7004	7.708200D+04	2688	2790	1.500800D+04
d2q06c	26991	155592	3.109822D+07	24846	127555	1.963086D+07
d6cube	13054	54480	1.039490D+07	13036	54470	1.039451D+07
degen2	6868	15344	8.437960D+05	6865	15365	8.446610D+05
degen3	50178	119665	1.540946D+07	49759	119202	1.536064D+07
e226	2600	3508	8.016000D+04	2163	2992	7.051400D+04
etamacro	2371	11018	5.791700D+05	2124	9895	4.985410D+05
fffff800	10091	18358	9.120660D+05	4612	7833	2.757650D+05
finnis	2771	5368	9.718600D+04	1845	3679	5.927700D+04
fit1d	267	272	4.156000D+03	267	272	4.156000D+03
fit2d	296	299	4.853000D+03	296	299	4.853000D+03
fit1p	196251	196251	8.196750D+07	165251	182321	6.554193D+07
forplan	2850	3667	1.329930D+05	1767	2537	8.912700D+04
ganges	7656	29389	1.487873D+06	6537	11294	2.186080D+05
gfrd-pnc	835	1798	6.864000D+03	809	1763	6.889000D+03
greenbea	33791	83599	5.836595D+06	26594	46767	1.591875D+06
greenbeb	33766	81865	5.415211D+06	26488	45054	1.437854D+06
grow15	3130	7090	1.751800D+05	3130	7090	1.751800D+05
grow22	4600	10552	2.619880D+05	4600	10552	2.619880D+05
grow7	1450	3090	7.382000D+04	1450	3090	7.382000D+04
israel	11053	11265	9.896870D+05	7010	8560	5.373340D+05
kb2	402	460	5.830000D+03	357	421	4.873000D+03
lotfi	1043	1776	3.017000D+04	588	977	1.167500D+04
maros	11409	26717	1.569197D+06	6762	11935	3.322430D+05
maros-r7	330472	1137233	4.452802D+08	161040	519612	1.435535D+08
modszk1	5658	13866	4.142700D+05	5199	11502	2.946700D+05
nesm	4081	20495	1.010167D+06	4057	20407	9.999890D+05
perold	6307	25200	1.753190D+06	5457	22408	1.508892D+06

Table 2 (continued). Effort to compute Cholesky factors.

Problem	Before presolve			After maximum presolve		
	nz(AA^T)	nz(L)	flops	nz(AA^T)	nz(L)	flops
pilot	59540	204136	5.221072D+07	58177	190399	4.728008D+07
pilot4	6333	14875	1.022505D+06	5753	14409	9.934030D+05
pilot87	115951	439164	1.888943D+08	115286	416138	1.675774D+08
pilot-ja	13250	54492	6.664482D+06	11179	39984	4.040114D+06
pilot-we	4825	16529	7.692550D+05	4464	14714	6.143680D+05
pilotnov	10857	54110	6.766992D+06	9174	40054	3.847968D+06
recipe	495	584	7.800000D+03	135	144	5.840000D+02
sc105	226	595	4.341000D+03	226	595	4.341000D+03
sc205	451	1287	1.070500D+04	451	1287	1.070500D+04
sc50a	101	196	8.940000D+02	101	196	8.940000D+02
sc50b	93	211	1.171000D+03	93	211	1.171000D+03
scagr25	1922	3380	3.660400D+04	1531	3104	3.541000D+04
scagr7	500	759	6.475000D+03	361	611	5.185000D+03
scfxm1	2903	4450	8.879800D+04	2183	3419	6.363900D+04
scfxm2	5826	9112	1.848440D+05	4394	6876	1.266480D+05
scfxm3	8749	13783	2.814170D+05	6605	10385	1.910550D+05
scorpion	1713	2102	1.577800D+04	583	1167	8.777000D+03
scrs8	1708	6151	1.728950D+05	1202	4047	7.888500D+04
scsd1	1056	1315	2.607100D+04	1056	1315	2.607100D+04
scsd6	1952	2398	4.251200D+04	1952	2398	4.251200D+04
scsd8	3883	7566	1.543480D+05	3883	7566	1.543480D+05
sctap1	1386	2391	2.870900D+04	1248	2149	2.590100D+04
sctap2	5505	14097	5.861550D+05	4927	10911	2.995030D+05
sctap3	7386	17459	5.676810D+05	6694	14922	3.931340D+05
seba	51400	53914	1.058269D+07	0	0	0.0D+00
share1b	884	1361	2.036300D+04	705	1144	1.479200D+04
share2b	775	1225	1.825500D+04	667	815	8.243000D+03
shell	1455	4096	7.607000D+04	1406	4006	7.597200D+04
ship04l	4228	4428	5.887600D+04	2320	2392	2.069600D+04
ship04s	2912	3252	4.274800D+04	1533	1605	1.354500D+04
ship08l	8512	8936	1.203120D+05	3864	4025	3.594100D+04
ship08s	4728	5588	7.543600D+04	1911	2057	1.767500D+04
ship12l	10673	11493	1.556690D+05	4868	5002	4.332400D+04
ship12s	5345	6441	7.979300D+04	2139	2273	1.885900D+04
sierra	4896	11910	2.778580D+05	4418	12555	4.061110D+05
stair	6197	14417	8.138230D+05	6197	14417	8.138230D+05
standata	1399	3072	4.628400D+04	824	2106	2.874400D+04
standgub	1399	3072	4.628400D+04	824	2106	2.874400D+04
standmps	2587	4613	8.303300D+04	1596	2874	3.534800D+04
stocfor1	504	903	9.295000D+03	374	706	6.878000D+03
stocfor2	12738	33207	7.924750D+05	9806	27418	6.075760D+05
stocfor3	103360	253020	5.934746D+06	80640	205791	4.347043D+06
truss	12561	58092	4.122486D+06	12561	58092	4.122486D+06
tuff	5077	8544	3.654680D+05	3706	5886	2.096920D+05
vtp-base	1575	2291	3.468700D+04	222	291	2.303000D+03
wood1p	18046	18082	1.553610D+06	11467	11473	9.413870D+05
woodw	20421	47729	3.086441D+06	12611	29997	1.670215D+06

Table 3. Comparison of HOPDM 2.0 and OB1.60.

Problem	Default presolve				HOPDM-NoP		HOPDM-WithP		OB1.60
	m	n	nonz	time	iters	time	iters	time	iters
25fv47	769	1535	9959	1.32	26	23.69	28	24.33	22
80bau3b	1965	8736	19048	2.99	48	82.60	43	67.22	29
adlittle	53	95	365	0.06	12	0.42	11	0.43	14
afiro	25	31	80	0.02	8	0.19	7	0.24	9
agg	319	109	1543	0.19	17	5.65	19	3.73	23
agg2	455	294	3956	0.30	18	8.27	21	8.72	18
agg3	455	294	3972	0.29	18	7.80	21	8.79	17
bandm	211	366	1654	0.49	18	2.05	20	2.26	19
beaconfd	73	124	537	0.19	11	1.29	11	0.71	10
blend	66	73	388	0.07	10	0.51	10	0.40	12
bnl1	558	1109	4619	0.56	28	8.13	35	9.72	25
bnl2	1848	3007	12458	2.44	36	99.27	30	84.64	32
boeing1	294	373	2733	0.44	24	4.07	21	3.56	24
boeing2	125	143	801	0.09	21	1.53	17	1.16	14
bore3d	64	89	404	0.18	16	1.36	14	0.77	16
brandy	117	194	1536	0.33	15	1.55	19	1.63	17
capri	235	298	1325	0.25	24	3.61	17	1.91	18
cycle	1400	2403	14111	2.57	27	54.13	41	43.81	28
czprob	661	2688	5376	0.73	33	11.11	27	7.90	31
d2q06c	2012	4964	30263	5.06	31	220.51	41	220.19	32
d6cube	403	5443	32523	8.53	21	56.53	22	66.14	21
degen2	444	534	3976	0.19	12	4.75	11	4.61	14
degen3	1503	1818	24363	1.23	19	87.32	17	78.50	19
e226	161	260	2169	0.24	19	2.05	23	2.35	21
etamacro	331	540	1846	0.31	23	5.82	24	5.74	26
fffff800	313	663	4388	0.43	34	15.94	32	6.96	28
finnis	357	441	1473	0.35	27	4.03	20	2.64	20
fit1d	24	1024	13358	1.52	20	4.42	19	5.56	15
fit2d	25	10363	127760	25.48	28	52.80	25	71.10	18
fit1p	638	1688	9868	1.02	16	224.66	19	252.42	16
forplan	104	367	4022	0.34	22	2.84	21	2.61	21
ganges	840	1172	5487	1.17	19	13.26	20	6.92	16
gfrd-pnc	590	1066	2325	0.22	13	1.66	13	1.81	19
greenbea	1872	3911	23161	3.71	43[1]	128.85	44	63.58	41
greenbeb	1865	3895	23052	3.72	45	110.63	48	60.51	33
grow15	300	645	5620	0.14	14	3.01	14	3.18	12
grow22	440	946	8252	0.20	14	4.21	14	4.47	12
grow7	140	301	2612	0.07	13	1.30	12	1.28	10
israel	166	144	2262	0.09	22	6.29	22	4.52	25
kb2	43	41	265	0.02	13	0.45	13	0.48	15
lotfi	117	282	596	0.14	16	1.07	12	0.76	16
maros	626	952	5743	1.02	24	16.75	26	8.94	30
maros-r7	2152	6578	80167	4.72	22	2322.21	15	540.13	-
modszk1	658	1405	2863	0.86	24	8.21	26	8.24	-
nesm	646	2676	12926	1.29	32	20.13	31	20.41	31
perold	580	1233	5381	1.13	31	22.42	37	24.61	36

1. Only 3 digits accurate in solution.

Table 3 (continued). Comparison of HOPDM 2.0 and OB1.60.

Problem	Default presolve				HOPDM-NoP		HOPDM-WithP		OB1.60
	m	n	nonz	time	iters	time	iters	time	iters
pilot	1350	3329	40506	7.98	45	511.79	44	460.41	29
pilot4	389	878	4719	0.80	33	15.32	34	15.98	35
pilot87	1968	4595	70361	9.54	44	1652.50	43	1594.84	41
pilot-ja	795	1546	10931	2.32	32	64.92	38	55.34	43
pilot-we	691	2486	8418	2.21	34	20.09	38	18.56	37
pilotnov	830	1861	11466	2.63	19	32.07	19	25.72	20
recipe	61	99	371	0.15	10	0.60	6	0.48	9
sc105	104	103	280	0.04	9	0.39	9	0.43	10
sc205	203	202	550	0.07	10	0.63	11	0.73	11
sc50a	49	48	130	0.02	7	0.35	7	0.34	9
sc50b	48	48	118	0.04	6	0.36	6	0.32	8
scagr25	344	495	1316	0.25	15	1.51	17	1.83	17
scagr7	92	135	344	0.15	12	0.55	13	0.63	15
scfxm1	268	422	2159	0.42	18	2.28	22	2.69	17
scfxm2	536	844	4323	0.74	20	4.99	25	5.59	19
scfxm3	804	1266	6487	1.09	20	7.50	25	8.33	20
scorpion	180	212	581	0.34	11	1.14	10	0.91	12
scrs8	418	1089	2725	0.68	20	3.36	21	3.71	19
scsd1	77	760	2388	0.13	8	0.87	8	1.02	9
scsd6	147	1350	4316	0.26	10	1.40	10	1.73	10
scsd8	397	2750	8584	0.55	10	2.98	10	3.51	9
sctap1	269	452	1557	0.29	15	1.47	15	1.74	14
sctap2	977	1768	6159	0.76	11	4.60	12	4.94	12
sctap3	1346	2356	8229	1.03	11	5.57	13	6.99	15
seba	0	0	0	0.10	21	25.42	0	0.11	18
share1b	107	217	990	0.14	18	1.17	21	1.33	20
share2b	92	79	632	0.11	11	0.63	11	0.61	11
shell	487	1448	2902	0.34	22	4.24	22	4.18	20
ship04l	292	1890	4275	0.44	9	2.20	12	2.91	12
ship04s	216	1266	2860	0.32	10	1.77	13	2.20	13
ship08l	470	3099	7100	0.77	12	4.99	15	5.55	14
ship08s	276	1582	3622	0.44	11	2.90	13	2.83	14
ship12l	610	4147	9230	0.96	14	7.44	15	7.21	16
ship12s	340	1919	4273	0.49	13	3.91	14	3.35	14
sierra	1129	2008	6956	1.27	18	6.90	18	8.53	15
stair	356	384	3664	0.26	12	4.28	15	5.32	14
standata	292	441	1026	0.25	14	1.95	11	1.35	12
standgub	292	441	1026	0.28	14	1.97	11	1.49	12
standmps	388	1011	2356	0.44	22	3.41	23	3.43	18
stocfor1	94	96	359	0.05	10	0.43	11	0.48	16
stocfor2	1968	1854	7064	0.71	18	11.22	19	11.28	25
stocfor3	15336	14382	55088	5.12	30	149.12	32	135.88	34
truss	1000	8806	27836	0.91	18	32.28	18	33.20	18
tuff	246	523	3707	0.46	17	3.71	15	3.16	19
vtp-base	46	72	195	0.12	21	1.34	9	0.54	13
wood1p	170	1717	43657	9.99	30	45.65	28	37.20	17
woodw	703	5347	19727	3.02	28	44.36	33	38.67	23

Table 4. Efficiency of HOPDM 2.0 on infeasible problems.

Problem	Before presolve			After default presolve			MKSP	iters
	m	n	nonz	m	n	nonz		
bgdbg1	348	407	1440	267	326	?	?	0
bgetam	400	606	2060	334	511	1837	?	0
bgindy	2671	10116	65502	2621	9302	61916	0	9
bgprtr	20	34	64	14	24	46	?	0
box1	231	261	651	231	261	651	0	3
ceria3d	3576	824	17602	3576	824	17602	?	0
chemcom	288	720	1566	288	720	1566	0	9
cplex1	3005	3221	8944	3005	3221	8944	?	0
cplex2	224	221	1058	224	221	1052	6	85
ex72a	197	215	467	197	215	467	0	3
ex73a	193	211	457	193	211	457	0	3
forest6	66	95	210	66	95	210	0	11
galenet	8	8	16	4	8	12	?	0
gosh	3738	10733	97231	3004	8447	70422	?	0
gran	2603	2489	19906	1378	1264	?	?	0
greenbea	2390	5290	30679	2298	4972	?	?	0
itest2	9	4	17	7	4	15	?	0
itest6	11	8	20	6	3	?	?	0
klein1	54	54	696	54	54	696	0	13
klein2	477	54	4585	476	54	4584	0	11
klein3	994	88	12107	993	88	12106	0	3
mondou2	270	477	954	201	277	554	?	0
pang	357	434	2615	302	367	2103	117	16
pilot4i	410	970	5041	401	889	?	?	0
qual	323	459	1633	188	311	1162	143	15
reactor	318	635	2418	306	635	2406	?	0
refinery	323	459	1613	188	311	1140	148	15
vol1	323	459	1633	188	311	1162	143	30
woodinfe	35	89	140	31	85	?	?	0