Mathematical Programming 71 (1995) 221-245

Presolving in linear programming

Erling D. Andersen [a,*], Knud D. Andersen [b,1]

[a] *Department of Management, Odense University, Campusvej 55, DK-5230 Odense M, Denmark*
[b] *Department of Mathematics and Computer Sciences, Odense University, Campusvej 55, DK-5230 Odense M, Denmark*

Received 1 December 1993

Abstract

Most modern linear programming solvers analyze the LP problem before submitting it to optimization. Some examples are the solvers WHIZARD (Tomlin and Welch, 1983), OB1 (Lustig et al., 1994), OSL (Forrest and Tomlin, 1992), Sciconic (1990) and CPLEX (Bixby, 1994). The purpose of the presolve phase is to reduce the problem size and to discover whether the problem is unbounded or infeasible.

In this paper we present a comprehensive survey of presolve methods. Moreover, we discuss the restoration procedure in detail, i.e., the procedure that undoes the presolve.

Computational results on the NETLIB problems (Gay, 1985) are reported to illustrate the efficiency of the presolve methods.

Keywords: Linear programming; Presolving; Interior-point methods

1. Introduction

Even though computers and LP algorithms have been improved a lot during the last ten years, it is still important to present an LP problem to a solver in a properly formulated form. There are some rules a properly formulated LP problem must satisfy. It should contain as few variables, constraints and nonzeros as possible, it must be well-scaled and the constraints must be linearly independent.

These simple rules may not be satisfied in practice. Therefore, it is advantageous to implement a presolve procedure in an LP solver to detect redundancy and to remove it.

Presolving is an old idea and has been treated in [8,9,22-24]. All these authors propose the same basic approach to presolving, namely to use an arsenal of simple and

* Corresponding author. e-mail: eda@busieco.ou.dk.

[1] This author was supported by a Danish SNF Research studentship.

SSDI 0025-5610(95)00016-X

fast inspection type procedures to detect various forms of redundancy and then repeat these procedures until the problem cannot be reduced further. We use the same strategy, but propose a more exhaustive presolve procedure.

Clearly, there is a trade-off between how much redundancy a presolve procedure detects and the time spent in the presolve procedure. The optimal strategy is to detect and remove the types of redundancy which lead to a reduction in the total solution time. However, this is not possible, and therefore the conservative strategy of detecting simple forms of redundancy, but doing it fast, seems to be the best strategy.

Another type of presolve procedure has been proposed by Adler et al. [2] and Chang and McCormick [11]. They propose using general linear transformations on the constraints to reduce the number of nonzeros in the problem. This option is not discussed here.

Papers [8,9,22–24] discuss a presolve procedure in combination with the simplex algorithm. Here we use the presolve procedure in combination with an interior-point algorithm. Papers [1,15] already document that a presolve procedure in combination with an interior-point algorithm is beneficial.

The paper is organized as follows. In Section 2 we present our notation. In Section 3 the presolve procedure is described. In Section 4 we discuss how to recover an optimal primal and dual feasible solution to the original problem. In Section 5 we discuss our implementation of the presolve procedure. In Section 6 computational results are presented and finally in Section 7 our conclusions are presented.

2. Notation

Any LP problem can be formulated in the following form:

$$\begin{aligned} \min \quad & c^{\mathrm{T}}x, \\ \text{s.t.} \quad & Ax = b, \\ & l \leqslant x \leqslant u, \end{aligned} \tag{1}$$

where $x, c, l, u \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. Some of the simple bounds l_j and u_j may be $-\infty$ and $+\infty$ respectively. Define $L = \{j\colon\ l_j > -\infty\}$ and $U = \{j\colon\ u_j < +\infty\}$; then the optimality conditions to (1) can be stated as follows:

$$\begin{cases} Ax^* = b, & \\ A^{\mathrm{T}}y^* + z^* = c, & \\ (x_j^* - l_j)z_j^* = 0, & \forall j \in L, \\ (u_j - x_j^*)z_j^* = 0, & \forall j \in U, \\ z_j^* \geqslant 0, & \forall j \in \{j \in L\colon\ x_j^* = l_j \wedge l_j < u_j\}, \\ z_j^* \leqslant 0, & \forall j \in \{j \in U\colon\ x_j^* = u_j \wedge l_j < u_j\}, \\ z_j^* = 0, & \forall j \notin L \cup U, \\ l \leqslant x^* \leqslant u, & \end{cases} \tag{2}$$

Table 1
Conclusion that can be reached from the optimality conditions

l_j	u_j	z_j^*	x_j^*	Problem status
$> -\infty$	$+\infty$	> 0	l_j	–
$-\infty$	$< +\infty$	< 0	u_j	–
$-\infty$	?	> 0	–	Unbounded
?	$+\infty$	< 0	–	Unbounded
?	$< l_j$	?	–	Infeasible

"?" means any value; "–" means unknown.

where $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$. y and z are the Lagrange multipliers corresponding to the linear constraints and the linear inequalities in (1), respectively. Note that the superscript "$*$" means optimal values. In Table 1 we have listed the conclusions that can be deduced from the optimality conditions (2). For example, if it can be established that $z_j^* > 0$, then x_j is fixed at its lower bound if it is finite, otherwise the problem is unbounded.

3. A presolve procedure

The computational complexity of simplex and interior-point type LP algorithms is dependent on the number of constraints and variables in (1). Even more important in practice is the sparsity of A, because only nonzeros are stored and the work is dependent on the number of nonzeros. Therefore, a presolve procedure should reduce the size of A without creating any new nonzeros in A. This has the advantage that the storage cost cannot be increased by the presolve procedure. The disadvantage of not allowing the creation of new fill-ins is that general linear transformations on A are not allowed, even though it might be beneficial in some circumstances. However, we allow the presolve procedure to modify the objective function c, the right-hand side b and the simple bounds l and u.

In the following, we assume that A is sparse, i.e., contains less than, say, 30% nonzeros. The presolve procedure might not be advantageous if this assumption is not fulfilled. Furthermore, we assume that all reductions on (1) are applied immediately and therefore we always work on the reduced problem. Whenever we write $\forall i$ and $\forall j$, it means for all rows and for all columns in the current reduced problem.

The idea of the presolve procedure is to make several passes through A and in each pass various forms of redundancies are detected and removed. In the following sections we discuss the procedure in detail.

3.1. Simple presolving methods

Here we present some standard (simple) presolve methods. All of these have been treated before, see, for example, [8], but are included here for the sake of completeness.

(i) *An empty row*:

$$\exists i: \quad a_{ij} = 0, \quad \forall j. \tag{3}$$

Either the ith constraint is redundant or infeasible.

(ii) *An empty column*:

$$\exists j: \quad a_{ij} = 0, \quad \forall i. \tag{4}$$

Dependent on the bounds on variable x_j and its objective coefficient c_j, variable x_j is fixed at one of its bounds or the problem is unbounded.

(iii) *An infeasible variable*:

$$\exists j: \quad l_j > u_j. \tag{5}$$

The problem is trivially infeasible.

(iv) *A fixed variable*:

$$\exists j: \quad l_j = u_j. \tag{6}$$

Variable x_j can be substituted out of the problem.

(v) *A singleton row*:

$$\exists (i,k): \quad a_{ij} = 0, \quad \forall j \neq k, \quad a_{ik} \neq 0. \tag{7}$$

The ith constraint fixes variable x_j at level b_i/a_{ik}.

It should be noted that one reduction may lead to further reductions. For example, if A is a permuted lower triangular matrix, repeated use of (7) solves the problem. To exploit this possibility, our presolve procedure first counts the number of nonzeros in each constraint. Then, a list containing all singleton constraints is created. As the constraints in this list are used to fix variables, the number of nonzeros for each constraint is updated and new singleton constraints are appended to the list.

3.2. Column singletons

Now we study how to exploit column singletons where the definition of a column singleton is

$$\exists (j,k): \quad a_{ij} = 0, \quad \forall i \neq k, \quad a_{kj} \neq 0. \tag{8}$$

If one of the bounds on the column singleton is infinite, then the column singleton imposes a bound on one of the optimal Lagrange multipliers. All such bounds are listed in Table 2. Such bounds will be useful later.

(vi) *A free column singleton*:

$$\exists (j,k): \quad (a_{ij} = 0, \forall i \neq k, a_{jk} \neq 0) \wedge l_j = -\infty \wedge u_j = +\infty. \tag{9}$$

The substitution

$$x_j = \frac{b_i - \sum_{p \neq j} a_{ip} x_p}{a_{ik}} \tag{10}$$

can be done.

Table 2
Bounds on the optimal Lagrange multipliers y^* implied by the column singleton a_{kj}

l_j	u_j	a_{kj}	z_j^*	y_k^*
$-\infty$	$+\infty$	$\neq 0$	0	c_j/a_{kj}
$> -\infty$	$+\infty$	> 0	$\geqslant 0$	$\leqslant c_j/a_{kj}$
$> -\infty$	$+\infty$	< 0	$\geqslant 0$	$\geqslant c_j/a_{kj}$
$-\infty$	$< +\infty$	> 0	$\leqslant 0$	$\geqslant c_j/a_{kj}$
$-\infty$	$< +\infty$	< 0	$\leqslant 0$	$\leqslant c_j/a_{kj}$

The free column singleton reduction is very advantageous, because one constraint and one variable is removed from the problem without generating any fill-ins in A, although the objective function is modified. Next we will discuss two methods for generating more free column singletons.

(vii) *A doubleton equation combined with a column singleton*:

$$\exists i, j, k: \quad a_{ij}x_j + a_{ik}x_k = b_i, \quad j \neq k, \quad a_{ij} \neq 0, \quad a_{ik} \neq 0. \tag{11}$$

If variable x_k is a column singleton, then the bounds on the variable x_j are modified so that the feasible region is unchanged even if the bounds on x_k are removed. Afterwards variable x_k can be substituted out of the problem.

It is also possible in some cases to establish that bounds on a singleton are redundant. An example is

$$x_j - \sum_{k \neq j} x_k = 0, \quad x_j, x_k \geqslant 0, \tag{12}$$

where x_j is assumed to be a column singleton. Obviously, the simple bounds on variable x_j can be dropped without changing the feasible region, thereby creating a free column singleton. This example can be generalized as follows.

Whenever our presolve procedure detects a column singleton x_j, it tries to establish that it is implied free using the following technique. For each $a_{ij} \neq 0$ the following bounds on variable x_j can be computed:

$$u'_{ij} = \begin{cases} \left(b_i - \sum_{k \in P_{ij}} a_{ik}l_k - \sum_{k \in M_{ij}} a_{ik}u_k\right)/a_{ij}, & a_{ij} > 0, \\ \left(b_i - \sum_{k \in P_{ij}} a_{ik}u_k - \sum_{k \in M_{ij}} a_{ik}l_k\right)/a_{ij}, & a_{ij} < 0, \end{cases} \tag{13}$$

and

$$l'_{ij} = \begin{cases} \left(b_i - \sum_{k \in P_{ij}} a_{ik}l_k - \sum_{k \in M_{ij}} a_{ik}u_k\right)/a_{ij}, & a_{ij} < 0, \\ \left(b_i - \sum_{k \in P_{ij}} a_{ik}u_k - \sum_{k \in M_{ij}} a_{ik}l_k\right)/a_{ij}, & a_{ij} > 0, \end{cases} \tag{14}$$

where $M_{ij} = \{k: a_{ik} < 0, j \neq k\}$ and $P_{ij} = \{k: a_{ik} > 0, j \neq k\}$. It can be verified that $l'_{ij} \leqslant x_j \leqslant u'_{ij}$ for any feasible solution x to (1). If these new bounds are at least as tight as the original ones, the variable is said to be *implied free*.

(viii) *An implied free column singleton*:

$$\exists j, k: \quad (a_{ij} = 0, \forall i \neq k, a_{kj} \neq 0) \wedge l_j \leqslant l'_{kj} \leqslant u'_{kj} \leqslant u_j. \tag{15}$$

It is assumed that l'_{kj} and u'_{kj} are computed according to (13) and (14). The variable x_j can be treated as a free column singleton (see (9)).

Detection of column singletons has been used in other presolve procedures [8, pp. 149–152], but, at least to our knowledge, a systematic check whether a column singleton is implied free has not previously been proposed. In the case of a doubleton equation, Bradley et al. [8] propose always modifying the simple bounds on one of the variables such that the other one becomes free. Afterwards, the free variable is substituted out of the problem. We have not implemented this more general procedure, because it might causes fill-ins in A.

3.3. *Forcing and dominated constraints*

In this section, the simple bounds on the primal variables are exploited to detect forcing and dominated constraints.

In order to establish whether a constraint is of the forcing type or the dominated type, we compute

$$g_i = \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in M_i} a_{ij} u_j \quad \text{and} \quad h_i = \sum_{j \in M_i} a_{ij} l_j + \sum_{j \in P_i} a_{ij} u_j, \tag{16}$$

where $P_i = \{j:\ a_{ij} > 0\}$ and $M_i = \{j:\ a_{ij} < 0\}$. Clearly we have

$$g_i \leqslant \sum_j a_{ij} x_j \leqslant h_i, \tag{17}$$

for any solution x that satisfies the simple bounds. The following possibilities can occur.

(ix) *An infeasible constraint*:

$$\exists i: \quad h_i < b_i \vee b_i < g_i. \tag{18}$$

In this case the problem is infeasible.

(x) *A forcing constraint*:

$$\exists i: \quad g_i = b_i \vee h_i = b_i. \tag{19}$$

If $g_i = b_i$, then due to linearity the only feasible value of x_j is l_j (u_j) if $a_{ij} > 0$ ($a_{ij} < 0$). Therefore, we can fix all variables in the ith constraint. A similar statement can be made if $h_i = b_i$.

It is highly advantageous to get rid of forcing constraints, because all variables in them are structurally degenerate.

In the previous section, (13) and (14) were developed to compute implied bounds on variable x_j. These bounds in combination with the original bounds can be used to detect more free column singletons. A procedure using this method is called a dominated constraint procedure. (The term "dominated" is proposed in [19].)

In order to limit the computational cost of the dominated constraint procedure, we only compute l'_{ij} and u'_{ij} for each $a_{ij} \neq 0$ if either g_i or h_i are finite. Furthermore, g_i and

h_i are computed during the forcing constraint check and they can be reused to compute the implied bounds cheaply using the following formulas:

$$u'_{ij} = \begin{cases} (b_i - g_i)/a_{ij} + l_j, & a_{ij} > 0, \\ (b_i - h_i)/a_{ij} + l_j, & a_{ij} < 0, \end{cases} \tag{20}$$

and

$$l'_{ij} = \begin{cases} (b_i - h_i)/a_{ij} + u_j, & a_{ij} > 0, \\ (b_i - g_i)/a_{ij} + u_j, & a_{ij} < 0. \end{cases} \tag{21}$$

It should be noted that the implied bounds are only used to detect more free column singletons. Dependent on the context in which the dominated constraint procedure is used, the implied bounds may of course be used to tighten or relax the bounds on the primal variables. For example, the bounds should be tightened if the problem is a mixed integer problem. On the other hand, they could also be used to relax the simple bounds. The last possibility should certainly be exploited to create free variables if the reduced problem is solved by a simplex type algorithm, see [22].

Forcing constraints have been implemented in all presolve procedures, at least to our knowledge. The first reference to a dominated constraint procedure is [9, p. 63] (see also [24]) and our procedure is similar to that, although in [9] the procedure is executed more often than we propose. We limit the number of times the procedure is executed, because we have found it to be computationally expensive for some dense problems.

The only other reference to a dominated constraint procedure, we know of, is [19], i.e., it is implemented in the LP solver OB1. However, OB1's dominated constraint procedure seems not to be as general as the procedure proposed in this section.

3.4. *Forcing and dominated columns*

The dominated constraint procedure discussed in the previous section can, of course, be used on the dual problem. We call such a procedure a *dominated column procedure* and it has been proposed by Williams [24].

The first step of the dominated column procedure is to find all column singletons and use them to compute implied lower and upper bounds on the optimal Lagrange multipliers y^*. How this is done has been treated in Section 3.2. Let $\bar{y}_i$ and $\hat{y}_i$ be the maximum and minimum respectively of all such bounds, i.e., we have

$$\bar{y}_i \leqslant y_i^* \leqslant \hat{y}_i, \quad \forall i. \tag{22}$$

These simple bounds on the Lagrange multipliers can be used to fix some of the primal variables. Define

$$e_j = \sum_{i \in P_j} a_{ij}\bar{y}_i + \sum_{i \in M_j} a_{ij}\hat{y}_i \quad \text{and} \quad d_j = \sum_{i \in P_j} a_{ij}\hat{y}_i + \sum_{i \in M_j} a_{ij}\bar{y}_i, \tag{23}$$

where $P_j = \{i\colon\ a_{ij} > 0\}$ and $M_j = \{i\colon\ a_{ij} < 0\}$. Clearly,

$$e_j \leqslant \sum_i a_{ij} y_i^* \leqslant d_j. \tag{24}$$

From the optimality conditions (2) and (24) it can be deduced that

$$c_j - d_j \leqslant z_j^* = c_j - \sum_i a_{ij} y_i^* \leqslant c_j - e_j. \tag{25}$$

(xi) *A dominated column*:

$$\exists j\colon\quad c_j - d_j > 0 \vee c_j - e_j < 0. \tag{26}$$

If $c_j - d_j > 0$, then by (25) $z_j^* > 0$ and according to Table 1 variable x_j can be fixed at its lower bound. However, if $l_j = -\infty$, the problem is unbounded. An analogue conclusion can be reached if $c_j - e_j < 0$.

(xii) *A weakly dominated column*:

$$\exists j \notin S\colon\quad c_j - d_j = 0 \wedge l_j > -\infty, \tag{27}$$

$$\exists j \notin S\colon\quad c_j - e_j = 0 \wedge u_j < +\infty, \tag{28}$$

where $S = \{j\colon\ a_{ij}$ is a column singleton for some $i\}$. Note that the column singletons are used to generate the bounds d_j and e_j and, therefore, they cannot be dropped with this test. In the following proposition we prove that in the cases (27) and (28) the variable x_j can be fixed at its lower and upper bound, respectively.

Proposition 3.1. *Assume* (1) *has an optimal solution. If* (27) *or* (28) *holds, then there exists an optimal solution such that* $x_j^* = l_j$ *or* $x_j^* = u_j$, *respectively.*

Proof. Assume (27) holds. Furthermore assume without loss of generality that $l = 0$, $u = \infty$ and $j = n$. Then the dual problem of (1) is

$$\begin{aligned} &\max \quad b^{\mathrm{T}} y, \\ &\text{s.t.} \quad A^{\mathrm{T}} y + s = c, \\ &\qquad s \geqslant 0. \end{aligned} \tag{29}$$

Given the assumptions of the proposition, (29) has an optimal solution. Therefore there exists a dual optimal solution $\bar{x} \in \mathbb{R}^{n-1}$ to (29) without the last constraint which implies

$$A(\bar{x}, 0) = b \quad \text{and} \quad \bar{x} \geqslant 0. \tag{30}$$

Clearly $x = (\bar{x}, 0)$ is an optimal solution to (1) proving there exists an optimal solution such that $x_j^* = 0 = l_j$ is the case. The proof for (28) is similar. □

(xiii) *A forcing column*:

$$\exists j \notin S\colon\quad (c_j - e_j = 0 \wedge u_j = \infty) \vee (c_j - d_j = 0 \wedge l_j = -\infty). \tag{31}$$

Assume that $c_j - e_j = 0$ and $u_j = \infty$, which implies $0 \leqslant z_j^* \leqslant c_j - e_j = 0$. By construction we have

$$y_k^* = \begin{cases} \bar{y}_k, & a_{kj} > 0, \\ \hat{y}_k, & a_{kj} < 0. \end{cases} \tag{32}$$

If $-y_k^*$ times the kth constraint is subtracted from the objective function when $a_{kj} \neq 0$, the kth constraint can be removed from the problem. We have not implemented this method because we believe it occurs rarely.

Finally, the bounds e_j and d_j are used to compute new bounds on the optimal Lagrange multipliers y^* in the following manner. Assume that $l_j > \infty$ and $u_j = +\infty$. It follows from the optimality conditions (2) that

$$0 \leqslant z_j^* = c_j - \sum_i a_{ij} y_i^*. \tag{33}$$

If $a_{kj} \neq 0$, we have

$$y_k^* \leqslant \frac{c_j - e_j}{a_{kj}} + \bar{y}_k, \quad a_{kj} > 0, \tag{34}$$

or

$$y_k^* \geqslant \frac{c_j - e_j}{a_{kj}} + \hat{y}_k, \quad a_{kj} < 0. \tag{35}$$

Similarly, if $l_j = -\infty$, we obtain

$$y_k^* \geqslant \frac{c_j - d_j}{a_{kj}} + \hat{y}_k, \quad a_{kj} > 0, \tag{36}$$

and

$$y_k^* \leqslant \frac{c_j - d_j}{a_{kj}} + \bar{y}_k, \quad a_{kj} < 0. \tag{37}$$

In the case of (34) we have $c_j - e_j \geqslant 0$. Otherwise (26) is the case. Observing that $(c_j - e_j)/a_{kj} \geqslant 0$, it follows

$$\bar{y}_k \leqslant y_k^* \leqslant \frac{c_j - e_j}{a_{kj}} + \bar{y}_k, \tag{38}$$

so the new bounds on the Lagrange multiplier cannot be inconsistent. Similarly all the bounds (35), (37) and (36) must be consistent.

The new bounds on the Lagrange multipliers y^* generated by (34)–(37) are used to recompute e_j and d_j which in turn are used to detect more dominated columns.

3.5. *Duplicate rows*

Linearly dependent rows in A are undesirable, because in an interior-point algorithm a certain linear system must be solved in each iteration. This linear system has a singular matrix if A contains linearly dependent rows.

A simple form of linear dependent rows is when two rows are identical except for a scalar multiplier. Two such rows are called *duplicate* by Tomlin and Welch [23]. However, they treat slack variables separately and therefore we use a generalized version of their definition which is

$$\exists i, k: \quad a_{ij} = v a_{kj}, \quad i \neq k, \quad \forall j \notin S, \tag{39}$$

where $S = \{j: a_{ij}$ is a column singleton for some $i\}$ and v is a scalar multiplier.

Assume the ith and the kth row are duplicate; by definition we have

$$\begin{cases} \sum_{j \notin S} a_{ij} x_j + \sum_{j \in S} a_{ij} x_j = b_i, \\ \sum_{j \notin S} a_{kj} x_j + \sum_{j \in S} a_{kj} x_j = b_k, \\ a_{ij} = v a_{kj}, \quad \forall j \notin S. \end{cases} \tag{40}$$

If the ith row is added $-v$ times to the kth, we obtain

$$\begin{cases} \sum_{j \notin S} a_{ij} x_j + \sum_{j \in S} a_{ij} x_j = b_i, \\ \sum_{j \in S} a_{kj} x_j - v\left(\sum_{j \in S} a_{ij} x_j \right) = b_k - v b_i. \end{cases} \tag{41}$$

The following situations may occur.

(xiv) (a) *The kth row is empty*, (b) *the kth row contains an implied free variable* or (c) *the kth row is a doubleton equation containing a singleton column.*

All these cases have already been treated.

(xv) *Constraint k contains only column singletons*:

$$a_{ij} = 0, \quad \forall j \in S. \tag{42}$$

In this case we have two independent problems which can be solved separately. We believe this is a rare occurrence and it is not implemented in our presolve procedure.

These checks should be done twice when the ith row is added to the kth and, then, vice versa.

The main objection against finding all duplicate rows is, of course, that it may be computationally too expensive. However, if the problem is sparse, we have found that not to be the case. We find all the duplicate columns with a method similar to that proposed by Tomlin and Welch [23, p. 8]. The method consists of two phases. First rows with the same sparsity pattern in nonsingleton columns are found, because if two rows are duplicate, their sparsity pattern must be identical. It is completed in $O(nz(A))$ operations and in practice this phase is very fast. In the second phase the duplicate columns are detected in each group of columns with the same sparsity pattern. The second phase is typically more expensive than the first one, because it involves floating-point operations. In [23] the two phases are combined, but we have separated them because on a sparse problem, typically, very few rows have the same sparsity pattern. Thereby the computational cost in the second phase is minimized.

Table 3
Bounds on the Lagrange multipliers when two columns are duplicate

l_k	u_k	z_k^*	v	$c_j - vc_k$	z_j^*
?	$+\infty$	$\geqslant 0$	$\geqslant 0$	> 0	> 0
?	$+\infty$	$\geqslant 0$	$\leqslant 0$	< 0	< 0
$-\infty$	?	$\leqslant 0$	$\geqslant 0$	< 0	< 0
$-\infty$	?	$\leqslant 0$	$\leqslant 0$	> 0	> 0

"?" means any value.

3.6. *Duplicate columns*

The dual of duplicate rows are duplicate columns, i.e., columns that are identical except for a scalar multiplier. Even though the number of duplicate columns are expected to be modest in number, there exist classes of LP problems which contain a large proportion of duplicate columns, see [3].

We define column j and k to be *duplicate* if they satisfy the following definition:

$$\exists j, k: \quad a_{ij} = v a_{ik}, \quad \forall i, j \neq k, \tag{43}$$

where v is a scalar multiplier. We have that

$$z_j^* = c_j - \sum_i y_i^* a_{ij} = c_j - v \sum_i y_i^* a_{ik} = c_j + v(z_k^* - c_k) = c_j - vc_k + vz_k^*. \tag{44}$$

Using (44), it is possible to generate bounds on the optimal Lagrange multipliers. All such bounds are listed in Table 3.

(xvi) *Fixing a duplicate column*:

$$c_j - vc_k \neq 0. \tag{45}$$

Using Table 3, it is possible to establish a bound on z_j^*. This bound is used to fix variable x_j or to detect that the problem is dual infeasible.

(xvii) *Replacing two duplicate columns by one*:

$$c_j - vc_k = 0. \tag{46}$$

Let us study the part of the problem (1) which amounts to the two duplicate columns. We have

$$\begin{aligned} &\cdots c_j x_j + c_k x_k \cdots, \\ &\cdots a_{.j} x_j + a_{.k} x_k \cdots = b, \\ &l_j \leqslant x_j \leqslant u_j, \qquad l_k \leqslant x_k \leqslant u_k. \end{aligned} \tag{47}$$

Define

$$x_k = -v x_j + x_k' \tag{48}$$

and if x_k in (47) is replaced by the right-hand side of (48), we obtain

$$\begin{aligned} &\cdots c_k x_k' \cdots, \\ &\cdots a_{.k} x_k' \cdots = b, \\ &l_j \leqslant x_j \leqslant u_j, \qquad l_k \leqslant x_k' - v x_j \leqslant u_k. \end{aligned} \tag{49}$$

Table 4
Replacement bounds

v	l'_k	u'_k
< 0	$l_k + vu_j$	$u_k + vl_j$
> 0	$l_k + vl_j$	$u_k + vu_j$

Variables x_j and x_k can be replaced by variable x'_k if its bounds are specified as shown in Table 4. In summary, whenever two columns are duplicate and (46) holds, the duplicate column procedure modifies the bounds on variable x_k according to Table 4 and removes variable x_j from the problem.

The duplicate columns are found with the same algorithm as the duplicate rows.

The detection of duplicate columns has been proposed in [23] and has been used in a special case, see [3] and [7], but nowhere are general results of this possibility reported. Note that the duplicate column procedure will detect all split-free variables, i.e., free variables modeled by a difference of nonnegative variables. Such variables mean serious numerical difficulties in a primal–dual interior-point algorithm. The reason is that the set of optimal primal solutions is unbounded and the interior of the dual feasible region is empty. If the split-free variables are combined into free variables, the problem can normally be solved.

4. Postsolve

After problem (1) has been presolved, and if it is not detected infeasible or unbounded, the reduced problem must be solved by an LP algorithm. In general, the optimal solution to the reduced problem is not feasible for the original problem. Therefore, a restoration procedure is needed to "undo" the presolving.

There are two kinds of restoration procedures. One of them solves the original problem by the simplex algorithm, using the optimal solution to the reduced problem as a hot start. The other type of restoration procedure is a reverse presolve procedure. The first method has been used by Williams [24], but can be very inefficient. Observing this, Tomlin and Welch [21] proposed the second procedure, calling it a "postsolve". The first type of restoration procedure is excluded in our case because we use an interior-point based LP algorithm. We will therefore not discuss this approach further.

The approach of the postsolve procedure is first to undo the last reduction made by the presolve procedure in such a manner that if the primal and dual solutions to the reduced problem are optimal and feasible, then the restored solution is also optimal and feasible. This process is then repeated until the optimal solution to the original problem is recovered.

To do the postsolving correctly and efficiently, it is necessary that the presolve procedure store some information on a presolve "stack". In the following we assume that all the necessary information is available from the presolve stack.

It should be noted that whenever we write $\sum_i y_i^* a_{ij}$, it means that the sum is over the rows in the current problem.

An empty row. No primal postsolving is necessary and y_i^* can be set to any value, for example 0.

A singleton row. Assume a_{ij} is the only element in the ith row. No primal postsolving is necessary, because x_j is fixed at its optimal value. The value

$$y_i^* = \frac{c_j - \sum_{k \neq i} y_k^* a_{kj}}{a_{ij}} \quad \text{and} \quad z_j^* = 0 \tag{50}$$

is dual feasible and optimal.

An empty column, or a fixed variable. The optimal Lagrange multiplier $z_j^* = c_j - \sum_i y_i^* a_{ij}$ for the variable must be computed.

A free column singleton or an implied free column singleton. Assume the last reduction made by the presolve procedure is to use the column singleton a_{ij} to eliminate variable x_j and the ith constraint. The optimal value x_j^* can be computed using (10). The dual optimal value is given by $y_i^* = c_j/a_{ij}$ and $z_j^* = 0$.

A doubleton equation combined with a column singleton. In this case the relations

$$a_{ij}x_j + a_{ik}x_k = b_i, \qquad l_j \leqslant x_j \leqslant u_j, \qquad l_k \leqslant x_k \leqslant u_k \tag{51}$$

are replaced by

$$a_{ij}x_j + a_{ik}x_k = b_i \quad \text{and} \quad \tilde{l}_k \leqslant x_k \leqslant \tilde{u}_k, \tag{52}$$

where a_{ij} is a column singleton, $\tilde{l}_k$ and $\tilde{u}_k$ are chosen so that the set of primal feasible solutions to (51) and (52) are identical and variable x_j is substituted out of the problem. The optimal x_j^* is given by

$$x_j^* = \frac{b_i - a_{ik}x_k^*}{a_{ij}}. \tag{53}$$

Let $\tilde{z}_k^*$ be the optimal Lagrange multiplier corresponding to x_k^* obtained from the reduced problem. The optimal Lagrange multiplier y_i^* is not known. However, we have

$$\tilde{z}_k = c_k - \frac{c_j}{a_{ij}}a_{ik} - \sum_{p \neq i} y_p^* a_{pk} = c_k - \sum_p y_p^* a_{pk}, \tag{54}$$

where $y_i^* = c_j/a_{ij}$. Eq. (54) follows as a result of x_j being substituted out of the problem. This value for y_i^* is not necessarily optimal and feasible if $x_k^* = \tilde{l}_k > l_k$ or $x_k^* = \tilde{u}_k < u_k$. In these cases the dual optimal value is

$$y_i^* = \frac{c_k - \sum_{p \neq i} y_p^* a_{pk}}{a_{ik}} \quad \text{and} \quad z_k^* = 0. \tag{55}$$

A forcing constraint. A forcing constraint forces its variables to be at one of their bounds. Assume, for simplicity, that the ith constraint forces all its variables to be at their lower bound. This implies that either $a_{ij} \leqslant 0$ or $a_{ij} \geqslant 0$ for all j. The Lagrange multiplier y_i^* must be chosen such that

$$0 \leqslant z_j^* = c_j - \sum_{p \neq i} y_p^* a_{pj} - y_i^* a_{ij}, \tag{56}$$

for all j, where $a_{ij} \neq 0$. If $a_{ij} > 0$ for all j, then the feasible y_i^* is given by

$$y_i^* \leqslant \min_{j \in \{k:\ a_{ik} > 0\}} \frac{c_j - \sum_{p \neq i} y_p^* a_{pj}}{a_{ij}}. \tag{57}$$

Otherwise, if $a_{ij} < 0$ for all j, then

$$y_i^* \geqslant \max_{j \in \{k:\ a_{ik} < 0\}} \frac{c_j - \sum_{p \neq i} y_p^* a_{pj}}{a_{ij}}. \tag{58}$$

Note that there always exists a feasible value for y_i^*. Indeed, there are an infinite number of feasible values which can be assigned to y_i^*. We have treated a special case, but it is easy to generalize it to arbitrary bounds and constraints. Postsolving of a forcing constraint has been treated in [21].

A dominated constraint. A dominated constraint is a constraint containing one implied free column singleton. How to postsolve this case has already been treated.

A dominated column or a weakly dominated column. If column j is a dominated column, it is sufficient to compute $z_j^* = c_j - \sum_i y_i^* A_{ij}$.

Duplicate rows. If two rows are duplicate, one of the rows is added to the other. If the new row is empty, containing an implied free column singleton, or is a doubleton equation containing a column singleton, it can be eliminated. The method to postsolve these cases has already been treated. However, the linear transformation on A must also be postsolved. Let the nonsingular matrix M represent the transformation. This transformation does not affect the primal solution, but the dual solution is affected, because the new dual constraints are

$$(A^{\mathrm{T}}M)\tilde{y} + z = c \quad \text{and} \quad \tilde{y} = M^{-1}y, \tag{59}$$

where y and $\tilde{y}$ are the Lagrange multipliers in the original and transformed problem, respectively. When the optimal $\tilde{y}^*$ to the reduced problem is known, y^* can be computed by multiplying $\tilde{y}$ with M. Note that M can be represented by two indices and a real, because it is an elementary matrix (one row is added to another).

Fixing a duplicate column. Same as a fixed variable.

Replacement of two duplicate columns by one. Assume that the columns $a_{\cdot j}$ and $a_{\cdot k}$ are duplicate and that they have been replaced by variable x_k' in the reduced problem. From (48) and (49) we have the relations

$$\begin{cases} x_k^* = -v x_j^* + x_k'^*, \\ l_j \leqslant x_j^* \leqslant u_j, \\ l_k \leqslant x_k^* - v x_j^* \leqslant u_k, \end{cases} \tag{60}$$

where $x_k'^*$ is the optimal value as obtained from the reduced problem. Any value for x_j^* and x_k^* which satisfies (60) is primal feasible and optimal, but the values should be chosen such that complementary slackness conditions are satisfied.

We have now outlined how to recover both primal and dual optimal feasible solutions if an optimal solution to the reduced problem is known. It should be noted that the amount of work spent in the postsolve phase in all cases is modest.

5. Implementation

In this section we will discuss our implementation of the presolve procedure.

The presolve procedure assumes that A is stored sparse in a row-wise and column-wise format and that the objective function c, the simple bounds l and u are stored in dense vectors. Furthermore, a Boolean flag is tagged to each constraint and variable. This flag has the value "false" if the constraint or the variable has been eliminated from the problem. Whenever a constraint or a variable is mapped off, it is treated as nonexistent by the presolve procedure. Therefore, during the presolve phase, it is not necessary to change the data structure of A.

The presolve procedure can be outlined as follows.

Procedure Presolve
  Remove all fixed variables.
  repeat
    Check rows
      Remove all row singletons
      Remove all forcing constraints
    Dominated constraints
      Remove all dominated constraints
    Check columns
      Remove all free, implied free column singletons and
      all column singletons in combination with a doubleton equation
    Dominated columns
    Duplicate rows
    Duplicate columns
  until(no reductions in last pass)
  Remove all empty rows and columns
end Presolve.

The interpretation of the above presolve procedure is that each subprocedure makes a pass through A. During the pass the redundancy is discovered and removed. The presolve procedure is terminated when it is not possible to reduce the problem further. Finally, empty rows and columns are removed. Furthermore, the procedure is terminated if it discovers that the problem is either primal or dual infeasible.

During the presolve phase the procedure informs the user of the kind of reductions it makes. (This is optional.) This information may be useful for the user to improve the model formulation.

6. Computational results

Can the proposed presolve procedure really reduce the size of an LP problem and is the presolve phase worthwhile? These are the main questions we will answer with a

test.

Let us first outline the test environment. The presolve procedure is used as a front-end procedure to Mehrotra's predictor–corrector algorithm, see [17,20]. The predictor–corrector algorithm is extended with the global convergence ideas of [14] (see also [18]). The most expensive operation in the primal–dual algorithm is the solution of the least-squares problem in each iteration. This is solved using a supernodal column Cholesky decomposition and the ordering is obtained with the multiple minimum-degree heuristic (see [16] for a discussion of details). The source code is written in ANSI C and all time results are obtained with the C procedure "clock" which measures the time in number of CPU-seconds. The testing is conducted on an IBM RISC 6000/230 workstation and on a Convex 3240 vector computer both at Odense University. On the workstation the code is compiled with the standard AIX compiler using the -O option. On the Convex the Convex C compiler is used with the option -O2, meaning that the code is vectorized. Furthermore, compiler directives are inserted in the source code to obtain a higher degree of vectorization.

It should be noted that the benefit of the presolve procedure is dependent on the efficiency of the LP code, because an inefficient LP code increases the benefit of the presolve procedure. Another warning is that the results are dependent on the computer architecture, because the presolve phase consists of operations which cannot be vectorized. Therefore, the relative benefit of the presolve phase is expected to be greater in a workstation environment than in a vector computer environment.

In the first part of the test, all the small NETLIB problems [13] were solved on the IBM RISC 6000/230 computer. By "small" we mean that they can be solved fast on the workstation.

In Table 5 the optimal primal objective values XFEAS and ZFEAS for each problem are shown where

$$\text{XFEAS} = \max\left(\|Ax^* - b\|_\infty , -\min_{j\in L}(x_j^* - l_j), -\min_{j\in U}(u_j - x_j^*) \right) \tag{61}$$

and

$$\text{ZFEAS} = \max\left(-\min_{j\in L\setminus U}(z_j^*), \max_{j\in U\setminus L}(z_j^*), \max_{j\notin L\cup U}(|z_j^*|), 0.0 \right). \tag{62}$$

XFEAS and ZFEAS measure the maximal violation by primal and dual variables of their simple bounds. In the case of an equality constraint in the primal or dual problem, it is assumed that the constraint is formulated with a fixed slack variable. Moreover, Table 5 shows the number of significant figures in the optimal objective value in the column SIGF. SIGF is computed as follows:

$$\text{SIGF} = \left| \log_{10}\left(\frac{|\text{primal obj. - dual obj.}|}{1.0 + |\text{dual obj.}|} \right) \right|. \tag{63}$$

We want to point out that the results are reported after the postsolve has been done, i.e., for the original problem.