OR Spectrum (2003) 25: 575–595
DOI: 10.1007/s00291-003-0130-x

OR Spectrum

Advanced preprocessing techniques for linear and quadratic programming

Cs. Mészáros[1,⋆] **and U.H. Suhl**[2]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences, 1111 Budapest, Lágymányosi u. 11, Hungary (e-mail: mcsaba@oplab.sztaki.hu)
[2] Institut für Produktion, Wirtschaftsinformatik und OR, Freie Universität Berlin, 14195 Berlin, Garystraße 21, Germany (e-mail: suhl@wiwiss.fu-berlin.de)

Abstract. The paper presents an overview on the preprocessing techniques of linear programming. A new reduction technique is also introduced and the presolve is extended to mixed integer and quadratic programming problems. Numerical results are presented to demonstrate the impact of presolving in interior point and simplex implementations. The demonstrative results are given on large-scale linear, mixed integer and quadratic programming test problems.

Key words: Linear programming – Presolve – Simplex and interior point methods

1 Introduction

It is widely recognized, that LP preprocessing is very important for solving large-scale linear programming (LP) problems efficiently [4,9,11,2]. This is true for both interior point and simplex algorithms. Although LP software and computers have become much faster, LP models have increased in size. Furthermore, LP optimizers are used in interactive applications and in integer programming, where many LPs have to be solved. More efficient algorithms and improved implementation techniques are therefore still very important. All practical LP models are generated by computer programs either directly or within a modeling system. The model generator derives the computer model from the mathematical model structure and the model data. Sometimes several models are combined or submodels have to be solved. Most model generators have very limited capabilities for data analysis. As a consequence, there is usually a significant part of the model that is redundant. The purpose of this paper is to present improved preprocessing techniques which offer significant progress over recent published LP preprocessing techniques. Some of

⋆ Supported in part by Alexander von Humboldt Foundation
Correspondence to: U.H. Suhl

the larger Netlib problems [8] will be used for a comparison. In addition we will present numerical results on other large problems and offer some insight into the impact of LP preprocessing to sparsity aspects of factorizations used in interior point and simplex algorithms. A comparison of the interior point code BPMPD [18,17] and the simplex code MOPS [21,22] with and without the new LP preprocessing will give additional insight into the importance of LP preprocessing for interior point and simplex algorithms. Furthermore, it will be seen that there are still models where efficient simplex codes are significantly faster than interior point codes. A very important aspect which has been frequently overlooked in papers on LP preprocessing is its impact on integer optimization problems (IP). Significant algorithmic progress for solving IPs during the last decade is based on the notion of tight linear programming relaxation [10,23,6]. The goal is to reformulate or transform (IP) into an integer equivalent problem (IP') such that the feasible region of (LP') is as small as possible. As a rough measure one can use the increase of the LP objective function value z(LP') compared to z(LP). A key role play cutting planes (facets or faces of high dimensions) derived from (IP) which are added resulting in an integer equivalent problem (IP') with tighter (LP') than (LP). The result is, in general, a larger LP problem with more constraints. It is even more important to remove redundancies because there is in general no impact on the tight linear programming relaxation but the model reduced by preprocessing will be re–optimized many times during the branch and bound algorithm. This effect will be shown on some mixed integer optimization problems in Section 4. In Section 5 we will summarize how the described presolving techniques can be extended for quadratic programming. We report numerical results which show that presolving is also important for nonlinear optimization.

2 Preprocessing techniques for LP

Using matrix notation we may state an LP problem in the computational standard form (LP):

$$\begin{aligned} \min \quad & c^T x, \\ & \overline{b} \geq Ax \geq \underline{b}, \\ & u \geq x \geq l, \end{aligned} \tag{1}$$

and its dual,

$$\begin{aligned} \max \; & q^T \underline{b} - p^T \overline{b} + v^T l - w^T u, \\ & A^T y + v - w = c, \\ & y + q - p = 0, \\ & v, w, q, p \geq 0 \end{aligned}$$

where l, u,c,x, v and w are n-vectors, $\underline{b}, \overline{b}$,$y, p$ and q are m-vectors, and A is an $m \times n$ matrix. The vectors $\underline{b}, \overline{b}, l$ and u may contain elements that are plus or minus infinity. $I = \{1, ..., m\}$ and $J = \{1, ..., n\}$ are the index sets of constraints and variables which will be updated by the presolve and postsolve process. Note that

(LP) supports all the specialities of practical model formulations such as fixed and free variables and ranged constraints. The main goals of LP preprocessing are:

- eliminate as many redundant constraints as possible;
- fix as many variables as possible;
- transform bounds of single structural variables (either relaxing them during LP preprocessing or tightening bounds during IP preprocessing);
- reduce the number of variables and constraints by eliminations;

After solving the preprocessed problem to optimality, a *postsolve* operation is necessary to

- present a full primal–dual optimal solution of the original problem from the primal–dual optimal solution of the preprocessed problem;
- create an optimal basis solution of the original problem from the optimal basis solution of the preprocessed problem.

The postsolving process is performed in reverse order of the presolving steps and computes the full solution corresponding to the previous presolving stage. Since most of the LP preprocessing techniques are now well documented and understood [4,11,2,9] we will only briefly present the standard techniques and concentrate on those aspects which make the new preprocessing superior to other published preprocessing techniques.

2.1 Tests based on primal and dual feasibility

The first group of presolving steps considered uses feasibility tests on the primal and dual problem to detect redundancy. These steps fix variables at one of their bounds or remove constraints but do not transform the problem algebraically.

2.1.1 Empty rows and columns. If row i has no nonzero entries then it can be deleted from the problem unless $\overline{b}_i < 0$ or $\underline{b}_i > 0$. In the latter cases the problem is primal infeasible. This reduction requires no postsolving operations, and an optimal basis for the original problem can be obtained by including the ith slack variable in the optimal basis of the presolved problem. Dual values are generated as:

$$y_i = 0, \quad p_i = 0, \quad q_i = 0. \tag{2}$$

Column j with no nonzero entries can be fixed at its lower or upper bound depending on the sign of the cost coefficient. Dual infeasibility of the problem may be detected if the appropriate bound is infinite. This step also does not require postsolve operations, dual values v_j and w_j have to be set to zero.

2.1.2 Singleton rows. Singleton rows may be replaced by bounds on primal variables. The resulting new bound may be infeasible or fixes the variable at a bound.

Let row i be a singleton row which has the nonzero entry in column j, then we compute in the postsolve operation

$$y_i = \frac{\left(c_j - \sum_{k \in I, k \neq i} a_{kj} y_k\right)}{a_{ij}}, \; v_i = w_i = 0, \; q_i = \max(0, -y_i), \; p_i = \max(0, y_i), \tag{3}$$

which gives the optimal dual values for the ith row. If column j is basic or it is non–basic at one of its original bounds then the basis has to be extended with the slack variable corresponding to the ith row. If column j is non–basic at a bound introduced by the singleton row, then column j has to be included in the basis and the dual values have to be recomputed as:

$$\begin{aligned} &y_i = 0, \; p_i = 0, \; q_i = 0, \\ &v_j = \max\left(0, \sum_{k \in I} a_{kj} y_k - c_j\right), \quad w_j = \max\left(0, c_j - \sum_{k \in I} a_{kj} y_k\right). \end{aligned} \tag{4}$$

2.1.3 Primal feasibility tests. The key of these techniques are lower resp. upper row sums which are based on current lower and upper bounds of the primal variables. Let us define

$$\overline{b}'_i = \sum_{k \in J, a_{ik} > 0} a_{ik} u_k + \sum_{k \in J, a_{ik} < 0} a_{ik} l_k, \quad \underline{b}'_i = \sum_{k \in J, a_{ik} > 0} a_{ik} l_k + \sum_{k \in J, a_{ik} < 0} a_{ik} u_k \; .$$

Primal infeasibility is detected if $\underline{b}'_i > \overline{b}_i$ or $\overline{b}'_i < \underline{b}_i$. If $\underline{b}'_i = \overline{b}_i$ or $\overline{b}'_i = \underline{b}_i$ then each variable, affected by constraint i, can be fixed at one of its bound. If $\overline{b}_i \geq \overline{b}'_i \geq \underline{b}'_i \geq \underline{b}_i$ then row i is redundant. In the postsolve we have to extend the basis with the ith slack variable. Dual values for removed rows are computed as in (2), for fixed columns as in (4).

2.1.4 Cheap dual tests. In this step we remove column j for which

$$\begin{aligned} &\text{if } c_j \leq 0 \text{ and } \begin{cases} \overline{b}_i = +\infty \text{ if } a_{ij} > 0 \\ \underline{b}_i = -\infty \text{ if } a_{ij} < 0 \end{cases} i \in I, \\ &\text{if } c_j \geq 0 \text{ and } \begin{cases} \overline{b}_i = +\infty \text{ if } a_{ij} < 0 \\ \underline{b}_i = -\infty \text{ if } a_{ij} > 0 \end{cases} i \in I. \end{aligned} \tag{5}$$

In the first case variable x_j is fixed at its upper bound while in the second case at its lower bound. If the appropriate bound is not finite and $c_j \neq 0$ then the problem is dual infeasible. Note that this test includes the removal of empty columns. If the appropriate bound is not finite but $c_j = 0$ then all constraints having nonzero entry in column j can be removed, and the value of variable j is left undefined. In this case the postsolve has to determine the most dominating row and to compute the value of x_j correspondingly. The optimal basis has to be extended with variable j and with the removed rows except for the determined most dominating one. The

dual values v_j and w_j are set to zero, the dual values corresponding to the most dominating row are computed as in (3), for the other rows as in (2). In the other case, if variable j is fixed at its bound, the dual values are computed as in (4).

2.1.5 Dual feasibility tests. Similarly to the primal feasibility tests, we use upper and lower bounds on the dual variables, defined as

$$\underline{y}_i = \begin{cases} \infty \text{ if } \underline{b}_i > -\infty \\ 0 \text{ if } \underline{b}_i = -\infty \end{cases}, \quad \overline{y}_i = \begin{cases} -\infty \text{ if } \overline{b}_i < +\infty \\ 0 \text{ if } \overline{b}_i = +\infty \end{cases}$$

for computing upper and lower limits on dual constraints:

$$\underline{c}_j = \sum_{i \in I, a_{ij} > 0} a_{ij} \underline{y}_i + \sum_{i \in I, a_{ij} < 0} a_{ij} \overline{y}_i, \quad \overline{c}_j = \sum_{i \in I, a_{ij} < 0} a_{ij} \underline{y}_i + \sum_{i \in I, a_{ij} > 0} a_{ij} \overline{y}_i.$$

Dual feasibility tests are then performed in two steps:

1. (weak dominance) If $c_j = \overline{c}_j$ (or $c_j = \underline{c}_j$) then variable j can be removed from the problem. We distinguish the following situations:
   (a) If $c_j = \overline{c}_j$ and $l_j > -\infty$ (or $c_j = \underline{c}_j$ and $u_j < \infty$) then variable j is fixed at its lower (or upper) bound.
   (b) If $0 \neq c_j = \overline{c}_j$ and $l_j = -\infty$ (or $0 \neq c_j = \underline{c}_j$ and $u_j = \infty$) then the problem is dual infeasible.
   (c) If $0 = c_j = \overline{c}_j$ and $l_j = -\infty$ (or $0 = c_j = \underline{c}_j$ and $u_j = \infty$) then all constraints having nonzero entry in column j can be removed together with column j and the value of variable j is left undefined. The postsolve performs the same operations as in Section 2.1.4.
2. (strong dominance) At the beginning the bounds on the dual variables are tightened by using the usual bound tightening technique. The dual tests are performed after the bound tightening with the modified lower and upper limits. Dual infeasibility is detected if $\underline{c}_j > \overline{c}_j$. Variable j can be fixed at its lower (or upper) bound if $c_j > \overline{c}_j$ (or $c_j < \underline{c}_j$), or dual infeasibility can be detected if the bound is infinite. In the postsolve we have to perform the same operations as in Section 2.1.4.

2.1.6 Duplicate rows. If $i \neq k$ and α are such that $a_{ij} = \alpha a_{kj}$ for $j = 1, ..., m$ then rows i and k are identical up to a scalar multiplier. Depending on the bounds of the constraints we can detect either primal infeasibility or we can make row i redundant by tightening the bounds of constraint k. In the postsolve, if row k is basic or it is non–basic at one of its original bounds then we set the dual values as in (2) and include in the basis rows. Otherwise, we set

$$y_i = \frac{\hat{y}_k}{\alpha}, \; q_i = \max(0, -y_i), \; p_i = \max(0, y_i),$$
$$y_k = 0, \; q_k = 0, \; p_k = 0.$$

where $\hat{y}_k$ is the optimal dual value in the presolved problem. If row k was basic in the optimum of the presolved problem then we set row i as basic. Otherwise we set row k basic and row i non–basic.

2.1.7 Duplicate columns. If $j \neq k$ and $\alpha \neq 0$ are such that $a_{ij} = \alpha a_{ik}$ for $i \in I$ then columns j and k are identical up to a scalar multiplier. We consider the following two cases:

1. If c_j=αc_k then column j can be removed from the problem while the bounds of column k may be widened. In the postsolve we have to compute the values of x_k and x_j by solving an LP of two variables and one constraint:

$$\alpha x_j + x_k = \hat{x}_k, \tag{6}$$

$$u_j \geq x_j \geq l_j, \quad x_k \geq x_k \geq l_{k,} \tag{7}$$

where $\hat{x}_k$ is the optimal value of variable k in the preprocessed problem. Dual values are set as

$$v_j = \alpha v_k, \quad w_j = \alpha w_k.$$

If k is non–basic in the presolved problem then we set j as non–basic. Otherwise we set that variable as basic (and the other as non–basic) which is in the optimal basis of problem (6–7).
2. If $c_j \neq \alpha c_k$ then one of the variables may be fixed at one of its bounds, or dual infeasibility may be detected depending on the bounds of the variables and on the relation of c_j and c_k. For the fixed variables we have to compute the dual slack values as in (4).

2.2 Reductions based on eliminations

We consider another type of reduction which is based on an algebraic transformation of the problem. The main point is the elimination of free variables. A key issue is, therefore, to generate as many free variables as possible by relaxing redundant finite bounds on structural variables. This relaxation is based on computing implied bounds of variables as:

$$u'_j = \min_i \begin{cases} \left(\underline{b}_i - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} > 0}} a_{ik} l_k - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} < 0}} a_{ik} u_k \right) / a_{ij},\ a_{ij} > 0 \\ \left(\overline{b}_i - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} < 0}} a_{ik} l_k - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} > 0}} a_{ik} u_k \right) / a_{ij},\ a_{ij} < 0 \end{cases}$$

$$l'_j = \max_i \begin{cases} \left(\underline{b}_i - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} < 0}} a_{ik} l_k - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} > 0}} a_{ik} u_k \right) / a_{ij},\ a_{ij} > 0 \\ \left(\overline{b}_i - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} > 0}} a_{ik} l_k - \sum\limits_{\substack{k \in J \setminus \{j\} \\ a_{ik} < 0}} a_{ik} u_k \right) / a_{ij},\ a_{ij} < 0 \end{cases}$$

If $u'_j < u_j$ (or $l'_j > l_j$) then we tighten the bound as $u_j := u'_j$ (or $l_j := l'_j$) and mark the bound which was tightened. Otherwise if $u_j = u'_j$ (or $l_j = l'_j$) we relax the bound by setting $u_j := +\infty$ (or $l_j := -\infty$). This investigation may be performed successively until no modification is available. During bound tightening primal infeasibility can be detected if $u'_j < l'_j$ or variables can be fixed if $u'_j = u^*_j$ or $l'_j = l^*_j$ where u^*_j and l^*_j are the bounds of variable j before the tightening process. At the end of the procedure we relax all bounds which were marked during the bound tightening. It is easy to see that the above bound relaxation procedure can not produce linearly dependent free variables. Therefore, the optimal primal–dual or basis solution of the problem after bound relaxation will also be optimal primal–dual or basis solution for the original problem. The elimination steps are then performed by choosing pivots in columns corresponding to free variables. During the elimination the constraint matrix, the upper and lower bounds of the constraints, as well as the objective function have to be transformed. After the transformations the row and column corresponding to the pivot position are removed from the matrix. During the postsolve process the basis is extended by the eliminated variables. The value of a primal variable corresponding to an eliminated column as well as the value of the dual variable corresponding to the eliminated row can be determined upon the primal and dual feasibility requirements: if variable j is eliminated by pivoting on a_{ij}, then

$$x_j = \frac{\overline{b}_i - \sum_{k \in J, k \neq j} a_{ik} x_k}{a_{ij}}, \quad v_j = 0, \quad w_j = 0, \tag{8}$$

$$y_i = \frac{\left(c_j - \sum_{k \in I, k \neq i} a_{kj} y_k\right)}{a_{ij}}, \quad q_i = \max(0, -y_i), \quad p_i = \max(0, y_i). \tag{9}$$

where b_i is either $\overline{b}_i$ or $\underline{b}_i$, depending on other tests mentioned later.

2.2.1 Singleton columns. A special case is a free singleton column which does not require transforming the constraint matrix since the pivot column has no other nonzero than the pivot. If the nonzero value of the free singleton column lies in a non–equality row then after the elimination, the value of the remaining slack variable has to be determined by the tests of empty columns. This test determines also the value of b_i for the postsolve operation in (8).

2.2.2 Doubleton rows. The next special elimination is the case of equality rows having two nonzero entries. In such a case the bounds of one variable can be tightened such that the bounds of the other variable becomes redundant while preserving the feasible set of the problem, see [7]. The elimination introduces fill–in in A in one column only. The number of nonzeros after the elimination of the pivot row and column does not increase. Let row i be an equality ($\overline{b}_i = \underline{b}_i$) and a doubleton with nonzero entries in columns j and k, and let column j be eliminated while the bounds of column k may be tightened. In the postsolve x_j is computed as in (8). We consider the following three cases:

1. Variable k has a value equal to one of its original bounds. In this case the postsolve is the same as described at the beginning of section 2.2. Variable k should be non–basic, and the basis can be extended by column j to cover row i.
2. Variable k has a value equal to one of its bounds introduced by the bound tightening. In this case we recompute the dual solution as

$$v_k = 0, \quad w_k = 0,$$
$$y_i = \frac{\left(c_k - \sum_{l \in I, l \neq i} a_{lj} y_l\right)}{a_{ik}}, \quad q_i = \max(0, -y_i), \quad p_i = \max(0, y_i),$$
$$v_j = \max\left(0, \sum_{l \in I} a_{lj} y_l - c_j\right), \quad w_j = \max\left(0, c_j - \sum_{l \in I} a_{lj} y_l\right).$$

   Clearly, variable k is basic in the presolved problem but should be non–basic in the postsolved problem. Furthermore, column j needs to have a value equal to one of its bound. Therefore, we set variable j non–basic and include column k in the basis.
3. Variable k has a value which is different from its original and introduced bounds. Clearly, variable j has also a value which differs from its bounds. Therefore we can set the dual values as in (9) and include column j in the basis.

2.2.3 General elimination of free variables. In this step we eliminate original and generated free variables by pivoting. In each step of the elimination we search for a pivot a_{ij}, where variable j is a free variable and row i is an equality ($\overline{b}_i = \underline{b}_i$) and the pivot fulfills the conditions:

$$|a_{ij}| \geq \alpha \max |a_{kj}| \quad k \in I, \quad \text{and}$$
$$(d_j - 1)(r_i - 1) \leq \beta (d_j + r_i),$$

where d_j is the number of nonzeros in column j and r_i the number of nonzeros in row i, α is a relative pivot tolerance and β is a relative fill–in tolerance. The first condition ensures numerical stability while the second one prevents heavy fill–in during the transformations. After the transformations column j and row i are removed from the problem and the elimination is repeated until no more pivots are available. Default values, which were also used in the numerical experiments, $\alpha = 10^{-2}$ and $\beta = 4.0$.

2.3 Other reductions

2.3.1 Finding linear dependency. One trivial reduction is identifying linearly dependent equalities (and linearly dependent free variables) in the problem. Such test can be based on the Gaussian elimination and can be implemented efficiently [1]. This test can detect primal (or dual) infeasibility and can remove linearly dependent equalities (and free variables) from the model. In the postsolve the optimal basis has to be extended by the slack variables corresponding to the removed rows, dual values are set as in (2).

2.3.2 Improving sparsity in A. For the first time, Chang et al. proposed a hierarchical method to make the constraint matrix sparser [5]. The idea was later used by Andersen et al. [3] and by Gondzio [9] in the context of interior point methods. The main motivation for this type of reduction is that later on the algorithm works on a matrix with less nonzero values. Therefore, the iteration computational effort is supposed to be lower. The sophisticated method of Chang et al. requires significantly more computational effort than the simplified procedure proposed by Gondzio which uses equality rows to eliminate at least one nonzero from other rows with the same or a larger nonzero pattern. As postsolve, the dual variables have to be modified by the inverse row transformations in reverse order.

2.4 LP preprocessing overview

The various preprocessing techniques are combined and executed in the following order:

repeat
  repeat
    repeat
      Singleton row tests
      Singleton column tests
      Primal feasibility tests
      Cheap dual tests
    until no more reduction is possible
    Dual feasibility tests
    Duplicate column tests
    Duplicate row tests
  until no more reduction is possible
  Doubleton row elimination
  Bound relaxation
  Scale the model using geometric mean scaling
  Finding linearly dependent rows
  Elimination of free variables
  Improving sparsity in A
until maximum number of passes reached or no more reduction is possible

Note that we collected tests which do not require substantial computational effort to the innermost loop of the process. We observed that a significant part of the total reduction is done in this loop, hence the more costly operations of the outer loops are performed on a reduced model. The middle loop executes procedures of moderate cost: for the dual feasibility tests we execute successive bound reduction on the dual variables and shadow prizes, while duplicate row (and column) tests require the numerical comparison of selected rows (and columns). In the latter case the number of row (and column) comparisons is reduced by using hash buckets. In the outermost loop we execute steps which require the algebraic transformations of A, these are the most costly operations in our preprocessing scheme. In contrast

Table 1. Problem size reduction by presolve

Problem	Before presolve			After presolve			Presolve
name	m	n	nz	m	n	nz	time
bn12	2324	3489	13999	1059	2233	10727	0.14
Cycle	1903	2857	20720	1023	2016	13561	0.06
80bau3b	2262	9799	21002	1965	8920	19461	0.08
degen3	1503	1818	24646	1404	1721	24077	0.03
greenbea	2392	5405	30877	1168	3213	23331	0.08
greenbeb	2392	5405	30877	1171	3208	23154	0.08
d2q06c	2171	5167	32417	1752	4768	30751	0.09
d6cube	415	6184	37704	402	5447	33830	0.16
dfl001	6071	12230	35632	3737	9618	32039	0.17
stocfor3	16675	15695	64875	8093	7139	59201	0.25
pilot87	2030	4883	73152	1868	4493	70564	0.11
pds-10	16558	48763	106436	8399	40491	98232	0.48
maros-r7	3136	9408	144848	2152	6578	80167	0.14
cre-b	9648	72447	256095	4975	31477	107592	0.44
ken-18	105127	154699	259826	46887	96783	233288	10.21
tough	217238	172158	997990	8429	34203	64128	8.55
dbic1	43200	226435	1038761	33688	140359	781948	1.94
gams	391609	295801	1270825	291708	197898	1158840	21.55
osa-60	10280	232966	1397793	10243	232965	839112	2.38
xs01	104376	450915	1864215	32143	374744	2316087	15.19

Solution times in seconds on a Pentium IV (2,2 GHz).

to the two inner loops, where the upper and lower row and column sums as well the lower and upper bounds on the dual variables are kept consistent with the reductions, we have to recompute them after the outermost loop due to the algebraic modifications of A. This introduces an overhead when performing multiple passes in the outermost loop. We observed that in the majority of cases, more than one pass in the outermost loop gives no improvement on the overall execution time and may decrease the numerical stability of the problem. Therefore we prefer to execute the outermost loop only once.

3 Preprocessing benchmarks

Netlib models with more than 1500 rows were used to report numerical results with the new preprocessing techniques and compare the results with those reported by Gondzio for the code HOPDM [9]. Some larger real-life models were added just to show that the techniques are even more important on larger models generated by model generators or modeling systems. There are three major differences between the presented presolve and the one described in [9]:

Table 2. Problem reduction by presolve of HOPDM

Problem	m	n	nz
bn12	1848	3007	12458
cycle	1380	2383	14061
80bau3b	1965	8736	19048
degen3	1503	1818	24363
greenbea	1848	3886	23112
greenbeb	1846	3876	23014
d2q06c	2010	4962	30259
stocfor3	15336	14382	55088
pilot87	1967	4595	70359
maros-r7	2152	6578	80167

- Gondzio suggests to use bound tightening on primal variables and to keep the generated tighter bounds while in our presolving scheme the bounds are relaxed after bound tightening (except for integer and non–linear variables). Gondzio's approach may open more possibilities in further primal presolve reductions but limits the forthcoming dual presolve steps. Additionally, in this case obtaining a basis solution of the original problem from a basis solution of the presolved problem may require pivot steps during postsolve, which is undesirable. Our approach may improve on the dual presolve steps, and it makes the derivation of a basic solution possible without further pivoting during postsolve.
- We introduced advanced elimination techniques in connection with bound relaxation, this was not applied in [9].
- Gondzio introduced the "weak dominance" during the dual presolve operation. His supporting theorem, however, is not valid with the conditions given and this step of his presolve may introduce incorrect reductions [19]. In our presolve we use weak dominance in a mathematically correct way, as a separate step without tightening the bounds on the dual variables. Furthermore, our implementation exploits weak dominance to identify and remove redundant constraints from the problem.

The results presented in Tables 1 and 2 show that our presolve does not perform worse than that of [9], and in several cases it leads to significantly larger reductions. The results suggest that keeping the tightened bounds after bound reduction cannot improve significantly on the efficiency of the presolve, and that the elimination techniques are especially important in several cases. There is a small inconsistency between the original number of nonzeros reported by Gondzio and us. Gondzio reports a smaller number of nonzeros for 80bau3b, greenbea, greenbeb and pilot87. This explains why the preprocessed models according to Gondzio also have a slightly smaller number of nonzeros.

Table 3. Impact of preprocessing on some integer optimization problems

Problem		Before preprocessing			After preprocessing			Solution time	
name		m	n	nz	m	n	nz	LP	IP
dsbmip	old	1656	1886	8373	1170	1799	7199	0.72	3.52
	new				1033	1650	7687	0.25	1.79
oil	old	5563	6181	39597	3187	3629	20286	3.01	233.41
	new				1689	2132	15535	1.77	39.63
rentacar	old	6803	9557	41842	2463	4385	26614	3.04	24.11
	new				890	2709	21401	0.62	3.01
mod011	old	4481	10958	29840	4480	10957	22253	0.70	946.17
	new				2163	8640	17227	0.67	388.20
waterx	old	57023	68064	143643	19731	32856	57917	19.23	25.56
	new				13553	23510	40630	4.77	6.62

Solution times in seconds on a Pentium IV (2,2 GHz).

4 LP preprocessing aspects for integer programming

One important aspect of LP preprocessing is its impact on mixed–integer optimization problems. The design of the LP preprocessor has to take integer variables into account if the preprocessed model is to be solved as an integer optimization problem. The following modifications were implemented:

- A variable is flagged if it can only take integer values;
- Integer variables are not eliminated;
- If bounds of integer variables are reduced the new bounds are rounded to the nearest integer in an obvious way; as a result other bounds may possibly be reduced;
- During bound relaxation the bounds of integer variables are tightened and not relaxed.

Several examples are used to show the impact of the improved LP preprocessing to solve mixed-integer optimization problems with MOPS. The old LP preprocessing is used for a comparison. The simplex engines and the mixed–integer module are unchanged.

The purpose of those benchmarks is to show how LP-preprocessing influences the running time if everything else, in particular IP preprocessing, remains unchanged. In an attempt to derive a strong linear programming relaxation IP-preprocessing adds normally valid inequalities violated by the current LP-solution and thus increases the size of the LP model.

Table 3 shows five MIP models with their original dimensions, the preprocessed models with the old and the new LP preprocessing, the running times for solving the initial LP and for solving the MIP model thereafter. The improvements are due to tighter bounds derived for all variables and the reduced size of the IP model.

5 Preprocessing aspects of quadratic programming

Convex quadratic problems (QP) were introduced as early as in the 1950s [12]. They represent an important class of linearly constrained optimization. Beyond the classical portfolio optimization, quadratic problems arise in a wide variety of applications, including least squares problems, regression analysis, data fitting, risk management. In this section we deal with the quadratic programming problem in the form:

$$\begin{aligned} \min\ & c^T x + \tfrac{1}{2} x^T Q x, \\ & \overline{b} \geq Ax \geq \underline{b}, \\ & u \geq x \geq l, \end{aligned} \tag{10}$$

and its dual in the Wolfe-sense:

$$\begin{aligned} \max\ & q^T \underline{b} - p^T \overline{b} + v^T l - w^T u - \tfrac{1}{2} x^T Q x, \\ & A^T y + v - w - Qx = c, \\ & y + q - p = 0, \\ & v, w, q, p \geq 0 \end{aligned}$$

where $Q \in R^{n \times n}$ is symmetric positive definite and the other quantities are defined as in Section 2. Numerical experiments showed that interior point methods are perhaps the most powerful algorithms for solving QP problems [20], therefore we consider the preprocessing techniques of QPs from the viewpoint of interior point methods. From the viewpoint of primal feasibility, there are no differences between (1) and (10). Thus the tests described in sections 2.1.1, 2.1.2, 2.1.3 and 2.1.6 may be applied to QP without any changes. Dual variables corresponding to fixed variables have to be computed in a slightly different way [13,14]. The duplicate column check (Sect. 2.1.7) has to be extended to test that the appropriate columns of Q are also identical with the same scalar multiplier. Note that the successful test requires that Q is rank deficient. For presolving quadratic programming problems we introduce upper and lower bounds on the gradient of the objective function, namely:

$$\underline{c}^* \leq c + Qx \leq \overline{c}^*$$

where $\underline{c}^*$ and $\overline{c}^*$ may be computed as:

$$\underline{c}_j^* = c_j + \sum_{k \in J, Q_{kj} > 0} Q_{kj} l_i + \sum_{k \in J, Q_{kj} < 0} Q_{kj} u_i,$$

$$\overline{c}_j^* = c_j + \sum_{k \in J, Q_{kj} < 0} Q_{kj} l_i + \sum_{k \in J, Q_{kj} > 0} Q_{kj} u_i$$

Note that $\underline{c}^*$ and $\overline{c}^*$ may be tighter if the upper and lower bounds on primal variables are tighter, therefore, it is advantageous to perform bound tightening on variables having quadratic terms in the objective function before computing $\underline{c}^*$ and $\overline{c}^*$.

5.1 Cheap dual tests for QP

In this step we remove column j for which

$$\text{if } \overline{c}_j^* \leq 0 \text{ and } \begin{cases} \overline{b}_i = +\infty \text{ if } a_{ij} > 0 \\ \underline{b}_i = -\infty \text{ if } a_{ij} < 0 \end{cases} i \in I \,, \tag{11}$$

$$\text{if } \underline{c}_j^* \geq 0 \text{ and } \begin{cases} \overline{b}_i = -\infty \text{ if } a_{ij} > 0 \\ \underline{b}_i = +\infty \text{ if } a_{ij} < 0 \end{cases} i \in I \,. \tag{12}$$

Let γ denote $\overline{c}_j^*$ for case (11) and $\underline{c}_j^*$ for case (12). In the first case column j is fixed at its upper bound while in the second case at its lower bound. If the appropriate bound is not finite and $\gamma \neq 0$ then the problem is dual infeasible. Note that this test includes the removal of empty columns with quadratic terms in the objective. If the appropriate bound is not finite $\gamma = 0$ then all constraints having nonzero entry in column j can be removed, and the value of column j is left undefined. In this case the postsolve requires determining the most dominating row and computing the value of x_j correspondingly. The optimal basis has to be extended with variable j and with the removed rows except for the determined most dominating one. The dual values v_j and w_j are set to zero, the dual values corresponding to the most dominating row are computed as

$$y_i = \frac{\left(c_j + (Qx)_j - \sum_{k \in I, k \neq i} a_{kj} y_k\right)}{a_{ik}}, \quad q_i = \max(0, -y_i), \quad p_i = \max(0, y_i)$$

while for the other removed rows as (2). In the other case, if variable j is fixed at its bound, the dual values are computed as:

$$v_j = \max\left(0, \sum_{k \in I} a_{kj} y_k - c_j - (Qx)_j\right), \tag{13}$$

$$w_j = \max\left(0, c_j + (Qx)_j - \sum_{k \in I} a_{kj} y_k\right). \tag{14}$$

5.2 Dual feasibility tests for QP

We define $\underline{y}, \overline{y}, \overline{c}$ and $\underline{c}$ as in Section 2.1.5. A modified bound tightening technique may be applied to the bounds of the dual variables which use either $\underline{c}$ or $\overline{c}$ instead of c in an obvious way. The dual tests are performed after tightening the bounds with the modified lower and upper limits. Dual infeasibility is detected if $\underline{c}_j > \overline{c}_j$. Variable j can be fixed at its lower (or upper) bound if $\underline{c}_j^* \geq \overline{c}_j$ (or $\overline{c}_j^* \leq \underline{c}_j$), or dual infeasibility can be detected if the bound is infinite. In the postsolve we have to perform similar operations as in the previous section.

Table 4. Preprocessing results on quadratic programming problems

Problem	Original				Preprocessed			
Name	M	N	Nz(A)	Nz(Q)	M	N	Nz(A)	Nz(Q)
dualc2	229	7	1603	21	9	7	51	21
dualc5	278	8	2224	28	1	8	8	28
dualc8	503	8	4024	28	15	8	105	28
q25fv47	820	1571	10400	59053	687	1456	10489	57674
qbandm	305	472	2494	16	162	317	1421	5
qbore3d	233	315	1429	50	28	53	241	0
qbrandy	220	249	2148	49	93	170	1582	27
qcapri	271	353	1767	838	148	214	1448	824
qe226	223	282	2578	897	149	250	2161	737
qfffff80	524	854	6227	1638	276	617	4649	1628
qgfrdxpn	616	1092	2377	108	432	908	2005	108
qpilotno	975	2172	13057	391	731	1754	11833	163
qscagr25	471	500	1554	100	229	380	1296	89
qscfxm1	330	457	2589	677	238	390	2195	674
qscfxm2	660	914	5183	1057	473	777	4433	1049
qscfxm3	990	1371	7777	1132	712	1168	6559	1109
qscrs8	490	1169	3182	88	221	905	2542	60
qscsd8	397	2750	8584	2370	397	2750	8584	2370
qsctap1	300	480	1692	117	284	480	1638	117
qsctap2	1090	1880	6714	636	1033	1880	6489	636
qsctap3	1480	2480	8874	861	1408	2480	8595	861
qseba	515	1028	4352	550	55	110	343	433
qshell	536	1775	3556	34385	293	1280	2517	2010
qship12l	1151	5427	16170	60205	638	4175	10513	43156
qship12s	1151	2763	8178	16361	323	1902	4777	8715
qstair	356	467	3856	952	254	282	5087	703
stcqp1	2052	4097	13338	22506	0	3158	0	1525
stcqp2	2052	4097	13338	22506	0	2045	0	5898

5.3 Other techniques

One can perform algebraic elimination of free variables with quadratic terms but, unfortunately, this requires the transformation of Q which may introduce fill–in in Q, even if the transformations introduce no fill–in in A. As it was shown in [15], the density of Q has a very strong influence on the fill–in during the solution of QPs with interior point methods, therefore fill–in in Q is very undesirable. Therefore we do not recommend to perform the elimination steps of Sections 2.2.1, 2.2.2 and 2.2.3 on variables having quadratic terms. Removing linearly dependent rows and improving the sparsity in A can be performed as in the linear programming case.

6 Preprocessing results on QPs

Table 4 presents the effect of the default preprocessing of BPMPD on quadratic programming problems. Figures given include the number of rows, columns and nonzeros in the constraint matrix, furthermore the number of nonzeros in the lower off–diagonal part of Q before and after the preprocessing of BPMPD. The results indicate that in several cases the original problem size of quadratic problems can be reduced significantly. For example, problems stcqp1 and stcqp2 were reduced to simple bound constrained optimization problems while the presolve reductions completely removed the nonlinearity of problem qbore3d.

7 Sparsity considerations of LU and Cholesky factorization

One other important consideration of LP preprocessing techniques is to consider its impact on numerical stability and sparsity of factorizations of sparse matrices which occur during the LP optimization of the models. From a theoretical point of view no prediction is possible concerning what impact preprocessing can have on sparsity and stability. For those preprocessing techniques without elimination of constraints one can expect that the sparsity and stability are not effected. This is different, however, if the structure and the size of elements will change. Therefore, we present only numerical results for the last (optimal) basis of the simplex algorithm and for the last Cholesky factorization of the interior point code. We report the number of nonzeros and the fill–in with and without the preprocessing techniques. One fundamental problem is that the execution paths with or without full preprocessing can be different. This implies in general that the final matrices to be factorized can be different. Nevertheless it can be helpful to make this comparison to obtain an understanding of how preprocessing effects sparsity of factorizations. Note that this effect is implicitly represented in the overall running times of the models. For problem stocfor3 we observed that the elimination steps of the presolve changed the structure of the problem in such a way that the standard Cholesky factorization of the interior point method suffered heavy fill–in, resulting in a larger number of nonzeros in the factorization than without presolve. Whereas our presented safeguard techniques prevent heavy fill–in of the whole constraint matrix during the eliminations, it may allow fill–in in few columns, which may result in undesirable behavior of the Cholesky factorization. As was discussed in [13] this case can be reliably identified and handled efficiently by using the augmented factorization scheme. Figures given in Table 5 for problem stocfor3 refer to the number of nonzeros by the latter approach rather than for the standard Cholesky factorization.

8 LP benchmarks – simplex versus interior point

One additional purpose of the paper is to show a comparison between two state-of-the-art implementations: an interior point code BPMPD [18] and the simplex code MOPS [21]. The new LP preprocessing is part of BPMPD and has also been

Table 5. Nonzeros in the factorizations with and without preprocessing

Problem	With full preprocessing		Without preprocessing	
name	nz(LU fact.)	nz(Cholesky)	nz(LU fact.)	nz(Cholesky)
80bau3b	5163	35793	6164	38904
Degen3	16386	124811	17065	129724
dfl001	17860	984030	26081	1206564
Stocfor3	36047	99071	51649	242489
pilot87	89803	410520	89360	425993
Pds-10	20426	748042	36477	1060980
maros-r7	66481	606529	82009	1426204
tough	14948	31532	247119	1870551
xs01	199171	145550599	338905	4829181

incorporated into MOPS. Furthermore, MOPS has also been extended by the interior point code of BPMPD. Both codes use, therefore, the same LP preprocessing kernels, the same scaling and postsolve algorithms. Furthermore, both codes were compiled with the same compiler and the same optimization options. Both codes were run with the default tolerances and strategies. The primal simplex optimizer of MOPS was used for the comparison although in some cases the dual optimizer would have been significantly faster than the primal. The interior point code used the supernodal pull Cholesky factorization [14] and the minimum local fill ordering [16]. The column "IPM+OBI" shows the total time needed to solve a model with the interior point code BPMPD, perform the primal and dual basis identification (BI) and establish formal optimality (OBI) with the primal simplex engine of MOPS using the default tolerances of MOPS. Finally the last two columns report numerical results with the primal simplex engine. These results compare also very favorably with the ones published in [2,9] even though the machines used for the computations are different. The following table shows the optimization times with and without preprocessing using the interior point code BPMPD and the primal simplex engine of MOPS. Since it is well established knowledge that LP preprocessing is generally advantageous before LP optimization we selected some models which show several possible cases: models where preprocessing has a different impact on interior point resp. simplex algorithms, models where the discussed preprocessing is very advantageous and those where preprocessing is disadvantageous. The term preprocessing means full preprocessing with all techniques discussed enabled i.e. also with the elimination of constraints and possible fill–in. The meaning of the columns is identical to those in the table above. Each of the selected models is solved with and without preprocessing.

9 Conclusions

It is a widely accepted assumption that preprocessing is more important for interior point methods, since IPMs work with the whole matrix in each iteration while

Table 6. Comparison of IPM and PSX on a Pentium IV (2,2 GHz)

Problem	IPM		BI	IPM+OBI	PSX	
name	Iter.	Sec	Sec	Sec	Iter.	Sec
bn12	22	0.64	0.03	0.81	4537	1.31
cycle	16	0.41	0.06	0.53	943	0.30
80bau3b	33	1.06	0.03	1.19	7389	1.34
degen3	12	0.89	0.08	1.03	4073	2.06
greenbea	26	0.62	0.12	0.92	4494	2.33
greenbeb	34	0.89	0.06	1.06	3923	2.03
d2q06c	22	0.97	0.09	1.25	9354	6.41
d6cube	16	0.75	0.11	0.99	11719	6.62
dfl001	29	27.28	0.42	27.92	27199	54.86
stocfor3	25	2.03	0.20	2.55	4934	7.70
pilot87	24	6.99	1.14	8.59	9153	33.00
pds-10	33	17.19	0.74	18.52	25414	61.08
maros-r7	11	4.16	0.45	4.84	3143	3.23
cre-b	23	6.89	0.31	7.75	14754	50.26
ken-18	20	43.90	18.75	73.56	148144	2235.03
tough	14	0.28	4.67	13.75	22401	42.41
dbic1	51	312.08	99.42	413.84	28675	301.44
gams	29	7.61	55.16	103.67	18321	430.50
osa-60	26	0.69	19.09	22.67	6721	28.34
xs01	102	383.05	38.66	444.83	126023	4469.91

BI: basis identification.
OBI: optimal basis identification after BI using primal simplex engine.

the simplex algorithm accesses only a small part of it. Surprisingly, our numerical experiments do not confirm the above statement. Of course, the following conclusions can only be based on the implemented interior point and simplex algorithms (BPMPD and MOPS) and the selected LP problems listed in the table:

- Concerning solution speed, LP preprocessing seems to be more important for simplex algorithms. This conclusion is based on the ratio of the total times without preprocessing and with preprocessing.
- Concerning numerical stability, preprocessing is more important for interior point algorithms. Our interior point implementation failed on four problems without preprocessing, but it had no numerical problems when preprocessing was turned on (see Table 7). Our numerical results show that the simplex method is numerically less sensitive to the redundancy in the problems: the simplex engine was able to solve all selected problems without LP preprocessing (in the case of problem tough it took a very long time).
- LP preprocessing is similarly important for mixed–integer optimization. The speed–up of solving the initial LP with preprocessing carries over to the branch-

Table 7. Impact of preprocessing on IPM and PSX

Problem	Preproc.	IPM with OBI		PSX	
name		iter.	secs	iter.	secs
bnl2	y	22	0.81	4537	1.31
	n	24	0.97	4339	1.42
cycle	y	16	0.53	943	0.30
	n	20	0.56	1497	0.28
80bau3b	y	33	1.19	7389	1.34
	n	39	1.23	8947	2.17
degen3	y	12	1.03	4073	2.06
	n	12	1.03	5007	2.03
greenbea	y	26	0.92	4494	2.33
	n	Iteration limit	Not finished	7719	5.28
greenbeb	y	34	1.06	3923	2.03
	n	Iteration limit	Not finished	4687	2.86
d2q06c	y	22	1.25	9354	6.41
	n	23	1.05	11428	8.25
d6cube	y	16	0.99	16725	12.94
	n	Iteration limit	Not finished	11719	6.62
dfl001	y	29	27.92	27188	54.86
	n	30	35.62	38046	96.59
stocfor3	y	25	2.55	4934	7.70
	n	27	3.42	8080	20.28
pilot87	y	24	8.59	9153	33.00
	n	26	9.94	9251	32.33
pds-10	y	33	18.52	25414	61.08
	n	31	23.16	7554	28.36
maros-r7	y	11	4.84	3143	3.23
	n	10	12.39	3382	4.12
cre-b	y	23	7.75	14754	50.26
	n	22	14.33	21280	80.05
ken-18	y	22	73.56	148144	2235.03
	n	27	82.06	184978	5995.92
tough	y	14	13.75	22401	42.41
	n	Iteration limit	Not finished	168800	4417.08
dbic1	y	51	413.84	28675	301.44
	n	42	535.39	32618	483.58
gams	y	29	103.67	18321	430.50
	n	28	1616.38	20545	860.33
osa-60	y	26	22.67	6721	28.34
	n	33	43.52	6868	64.27
xs01	y	102	444.83	126023	4469.91
	n	101	538.33	160044	7015.17

Solution times in seconds on a Pentium IV (2,2 GHz).

and-bound-phase where the simplex engine is used. Moreover, every time a new node is selected in a non-LIFO-node selection choice a new LU factorization has to be computed. Sparsity and speed of the LU factorization are improved by using advanced LP preprocessing techniques as can also be seen from Table 5.

– Table 4 indicates that preprocessing is also advantageous for quadratic programming problems. The results show that preprocessing may significantly decrease the number of nonlinear variables which is particularly advantageous.

Experience shows that model sizes increase due to the use of modeling systems and less willingness by users to generate redundant free models. In such a situation LP preprocessing plays an even more important role to reduce the size of LP models for better performance. The benchmark results of Table 6 indicate, that a state–of–the–art interior point code with crossover is on most LP models faster than a state–of–the–art simplex code. In those cases where the simplex code is faster (maros-r7, dbic1), the speed improvement of the simplex code is not as big as for the most suitable models for the interior point code (xs01, ken-18).

Much thoughts went into design and implementation of the optimal basis identification. Nevertheless we believe that the results indicate room for improvements (gams, ken-18, osa-60). There are other numerical difficult problems for the simplex method not shown here, where the crossover time is a multiple of the time to solve the LP with the interior point code.

References

1. Andersen ED (1995) Finding all linearly dependent rows in large-scale linear programming. Optimization Methods and Software 6: 219–227
2. Andersen ED, Andersen KD (1995) Presolving in linear programming. Math Programming 71(1): 221–245
3. Andersen ED, Gondzio J, Mészáros Cs, Xu X (1996) Implementation of interior point methods for large scale linear programs. In: Terlaky T (ed) Interior point methods of mathematical programming, pp 189–252. Kluwer, Amsterdam
4. Brearley AL, Mitra G, Williams HP (1975) Analysis of mathematical programming problems prior to applying the simplex algorithm. Mathematical Programming 15: 54–83
5. Chang SF, McCormick ST (1992) A hierarchical algorithm for making sparse matrices sparse. Mathematical Programming 56: 1–30
6. Dietrich BL, Escudero LF, Chance F (1993) Efficient reformulation for 0–1 programs – methods and computational results. Discrete Applied Mathematics 42: 147–175
7. Forrest JJH, Tomlin JA (1992) Implementing the simplex method for optimization subroutine library. IBM Systems Journal 31(1): 11–25
8. Gay DM (1985) Electronic mail distribution of linear programming test problems. COAL Newsletter 13: 10–12
9. Gondzio J (1997) Presolve analysis of linear programs prior to applying the interior point method. INFORMS Journal on Computing 9(1): 73–91
10. Hoffman KL, Padberg M (1991) Improving LP-representations of zero-one linear programs for branch-and-cut. ORSA Journal on Computing 3: 121–134

11. Karwan MH, Lotfi V, Telgen J, Zionts S (1983) Redundancy in Mathematical Programming. Springe, Berlin Heidelberg New York
12. Markowitz HM (1956) The optimization of a quadratic function subject to linear constraints. Naval Research Logistic Quarterly 3: 111–133
13. Maros I, Mészáros Cs (1998) The role of the augmented system in interior point methods. European Journal on Operational Research 107(3): 720–736
14. Mészáros Cs (1996) Fast Cholesky factorization for interior point methods of linear programming. Computers & Mathematics with Applications 31(4/5): 49–54
15. Mészáros Cs (1998) On the sparsity issues of interior point methods for quadratic programming. Working paper WP 98–4, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest
16. Mészáros Cs (1998) Ordering heuristics in interior point LP methods. In: Gianessi F, Komlósi S, Rapcsák T (eds) New trends in mathematical programming, pp 203–221. Kluwer, Amsterdam
17. Mészáros Cs (1999) BPMPD Home Page. World Wide Web, http:// www.sztaki.hu/ ˜meszaros/bpmpd
18. Mészáros Cs (1999) The BPMPD interior-point solver for convex quadratic problems. Optimization Methods and Software 11&12: 431–449
19. Mészáros Cs, Gondzio J (2001) Addendum to „Presolve analysis of linear programs prior to applying an interior point method". Informs Journal on Computing 13(2): 169–170
20. Mittelmann HD (1999) Benchmarking interior point LP/QP solvers. Optimization Methods and Software (12): 655–670
21. Suhl UH (1994) MOPS – Mathematical OPtimization System. European Journal of Operational Research 72: 312–322
22. SuhlUH (1999)MOPS Home Page. World Wide Web,http://www.mops-optimizer.com/
23. Suhl UH, Szymanski R (1991) Supernode processing of mixed–integer models. Computational Optimization and Applications 3: 317–331