Working Paper

A Modular Presolve Procedure for Large Scale Linear Programming

Artur Świętanowski

WP-95-113
October 1995

A Modular Presolve Procedure for Large Scale Linear Programming

Artur Świętanowski

WP-95-113
October 1995

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 807 □ Fax: +43 2236 71313 □ E-Mail: info@iiasa.ac.at

Foreword

It is possible to solve larger and larger linear programming models because of the growing capacity of computers and the refinement of algorithms. However, the linear programming models to be solved grow even harder. Therefore, it becomes even more important to present the model in such a way to the algorithm that it can be solved most efficiently. This point is further stressed by the fact that large models are usually generated in an automated or semi-automated way, which is primarily based on systematic description of the model and its precise formulation. In this process the quality of the formulation with respect to the solution is difficult to incorporate. Therefore, the present paper is devoted to methods which aim at reformulating the original model is such a way that it is ready for the Simplex Method and also in methods to translate the computatuional results back to the original formulation. The presented work is largely inspired by experiences with IIASA with formulation and solving large linear programming models.

Abstract

In this paper we present a survey of methods used for analysis and simplification of a general single-objective linear program prior to solving it with a simplex type optimizer. We consider the methods known since the early work of Brearley *at al.* as well as less known or appreciated numerical elimination methods. We then proceed to analyze in detail the usefulness of some of the presolve methods. We attempt to explain what impact each of these methods may have on the activity of a simplex type optimizer.

These theoretical speculations are validated by experiments involving the discussed methods and an advanced implementation of the simplex algorithm: a set of very large linear problems analysed with different subsets of available presolve techniques are solved using the simplex optimizer.

The paper is accompanied by a modular linear optimization package consisting of a stand alone presolver and postsolver as well as a new release of our advanced simplex optimizer with embedded presolve capabilities.

Key words: simplex method, presolve analysis

Contents

A Modular Presolve Procedure for Large Scale Linear Programming [*]

Artur Świętanowski[**]

1 Introduction: The Presolve Analysis Rationale

Despite advances in computer technologies, which resulted in a great increase of affordable computing power and equally important developments in the field of linear optimization, there is still demand for more efficient methods for solution of large scale linear programs (LP's). One of possible approaches to this problem is presolve analysis. It is based on the observation that most LP's are formulated inefficiently from the point of view of an optimizer (although this may be the result of a perfectly valid modeling process). Presolve analysis attempts to identify and remove as many redundancies as possible. The analysed problem is then optimized and the optimal values of primal and dual variables and reduced costs of the original problem are recovered.

Presolve analysis aims at reducing the problem solution time and perhaps, making it possible to solve some problems that are too difficult in their original formulation. The goals of a presolve procedure are:

1. reduction of problem dimension (i.e. the number of constraint matrix rows, columns and non-zeros),
2. improving problem's numerical properties and computational characteristics (e.g., by removing linearly dependent rows),
3. early detection of infeasibility or unboundedness,
4. revealing of some properties of the problem that may not have been obvious during model generation (e.g., the fact that some variables may be fixed, some constraints are redundant, etc.).

Typically used analysis methods, known since the work of Brearley *et al.* [4] are heuristic (see also [1]). They are designed to eliminate simple redundancies relatively cheaply, but they fail to discover more complicated relations that might be used to reduce the problem's size. On the other hand there were attempts to develop optimal methods for elimination of certain kinds of redundancies. Among those, an idea of McCormic [14] provided (directly and indirectly) some interesting results and prompted development of new techniques, like the one proposed by Gondzio [10] and followed in this research.

The focus of this paper is on the presentation of the impact of presolve analysis on performance of the revised simplex method. The selection of presolve techniques that we made is in our opinion perfectly suitable for application as a front end to a simplex optimizer. If an interior point method were to be used, then yet another preprocessing stage might be useful, or indeed,

[*]This research was partially sponsored by the Committe for Scientific Research of Poland grant no. 3P40301806.

[**]Institute of Automatic Control & Computation Engineering, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw

necessary to make some problems solvable (for an in-depth discussion of presolve analysis as applied to a primal-dual interior point method the reader is referred to Gondzio [10].)

On the other hand, *all* linear optimization methods might benefit from some or all of the presolve techniques presented here as, surely, redundancies in LP formulation will always remain redundant. Therefore all effort has been made to make our presolver implementation as flexible as possible: the user is allowed to choose virtually any subset of presolve methods that he or she wishes to use for a particular purpose, e.g., as a front-end for a different type of a linear optimizer.

In section 2 the presolve methods are divided into categories, each associated with certain parts of Kuhn–Tucker optimality conditions. In section 3 each of the analysis methods is discussed in detail. All optimal solution recovery procedure is given for every method in section 4. Section 5 covers the results of numerical experiments conducted with a linear program presolve procedure embedded in a simplex optimizer. Finally, the conclusions from this research are given in section 6.

The contents of the linear optimization package developed in cooperation with IIASA project *Methodology of Decision Analysis* are listed in appendix A. In appendix B we give all the information necessary for obtaining the whole optimization package, which is intended for use not only at IIASA but also in other research institutions. The applications are presented in more detail in appendix C. Their calling syntax is defined and a number of typical examples of their use is given.

2 The Kuhn–Tucker optimality conditions

We are concerned with a linear optimization problem of minimizing the objective function

$$f + \mathbf{c}^T\mathbf{x} \tag{1}$$

subject to constraints

$$\begin{array}{ccccc} \underline{\mathbf{b}} & \leq & \mathbf{Ax} & \leq & \overline{\mathbf{b}} \\ \underline{\mathbf{x}} & \leq & \mathbf{x} & \leq & \overline{\mathbf{x}} \end{array} \tag{2}$$

where $\mathbf{A} \in \Re^{m\times n}$, $\mathbf{x}, \mathbf{c} \in \Re^n$, $f \in \Re$, $\underline{\mathbf{x}} \in (\Re \cup \{-\infty\})^n$, $\overline{\mathbf{x}} \in (\Re \cup \{+\infty\})^n$, $\underline{\mathbf{b}} \in (\Re \cup \{-\infty\})^m$ and $\overline{\mathbf{b}} \in (\Re \cup \{+\infty\})^m$. The so-called "fixed adjustment" f usually is not included in the problem formulation, however in this paper it is convenient to introduce it right now.

Let $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ denote the primal and dual variables and the reduced costs, respectively. Their values represent an optimal solution to the problem (1)–(2) if and only if the Kuhn–Tucker optimality conditions are satisfied:

1. Primal feasibility:

$$\begin{array}{ccccc} \underline{\mathbf{b}} & \leq & \mathbf{Ax} & \leq & \overline{\mathbf{b}} \\ \underline{\mathbf{x}} & \leq & \mathbf{x} & \leq & \overline{\mathbf{x}} \end{array} \tag{3}$$

2. Dual feasibility:

$$\mathbf{A}^T\mathbf{y} + \mathbf{z} = \mathbf{c}$$

$$\forall i \in \{1,\ldots,m\} \left\{ \begin{array}{lcl} \underline{b}_i = -\infty & \Rightarrow & y_i \leq 0 \\ \overline{b}_i = +\infty & \Rightarrow & y_i \geq 0 \end{array} \right. \tag{4}$$

$$\forall j \in \{1,\ldots,n\} \left\{ \begin{array}{lcl} \underline{x}_j = -\infty & \Rightarrow & z_j \leq 0 \\ \overline{x}_j = +\infty & \Rightarrow & z_j \geq 0 \end{array} \right.$$

3. Complementarity:

$$\forall j \in \{1,\ldots,n\} \begin{cases} \underline{x}_j > -\infty & \Rightarrow \quad (x_j - \underline{x}_j)z_j = 0 \\ & \quad\text{or} \\ \overline{x}_j < +\infty & \Rightarrow \quad (\overline{x}_j - x_j)z_j = 0 \\ \text{otherwise} & \qquad x_j z_j = 0 \end{cases} \tag{5}$$

By analogy to the grouping of the optimality conditions, the presolve analysis methods may be divided into three categories. We must note, however, that such grouping is only a matter of presentation convenience. In most cases it is possible to derive each method from the analysis of both the primal and the dual problem. We add the fourth group – numerical eliminations performed on the constrained matrix:

1. methods derived from analysis of the primal feasibility conditions:
   - empty constraint removal,
   - singleton row removal,
   - fixed variables removal,
   - row constraint analysis,

     It is applied to one row at a time. Attempts to detect a limited class of redundant constraints (so called forcing and dominated constraints).
   - elimination of slack variables explicitly represented in the LP,
   - free singleton variable removal.

     By a free singleton we mean a variable which has infinite simple bounds and a non-zero coefficient in only one constraint.
2. methods derived from the dual feasibility conditions:
   - removal of empty columns,
   - determining of finite bounds on dual variables and reduced costs.
3. methods derived from the complementarity conditions:
   - fixing of variables for which a positive lower bound or a negative upper bound on reduced cost has been computed.
4. general linear transformations performed on a set of equalities which aim at reducing the density of the constraint matrix.

This paper shall not be directly concerned with detection of split free variables or duplicate constraint matrix rows or columns. They pose a serious problem for an interior point optimizer (see e.g., Gondzio [10]), but the simplex algorithm can handle them easily.

3 Presolve methods

In addition to the notation introduced in the previous section, from this moment on we shall use the following symbols:

- $\mathbf{a}_{i*}$ to denote i-th row of the constraint matrix $\mathbf{A}$, or all of this row except one element singled out in the context,
- $\mathbf{a}_{*j}$ to denote j-th constraint matrix column, or all of this column except one element singled out in the context.

Furthermore, we shall use terms "variable" and "(constraint matrix) column" as well as "(constraint matrix) row" and "constraint" interchangeably.

3.1 Simple presolve methods

3.1.1 Empty constraint

If the i-th row is empty, i.e. $\mathbf{a}_{i*} = \mathbf{0}$, then obviously scalar product $\mathbf{a}_{i*}^T\, \mathbf{x}$ is equal to zero. If $\underline{b}_i \leq 0 \leq \overline{b}_i$ holds true, then the constraint is always fulfilled, and thus redundant. Otherwise the problem is structurally infeasible.

3.1.2 Empty column

Given an empty column $\mathbf{a}_{*j} = \mathbf{0}$ from dual feasibility (4) we have

$$\mathbf{a}_{*j}^T \mathbf{y} + z_j = z_j = c_j. \tag{6}$$

When we compare the value of z_j with its bounds (4) we may either fix variable x_j or declare the dual problem infeasible. The possible cases are presented in the table 1.

Table 1: Variable fixing following dual problem analysis

z_j	$\underline{x}_j$	$\overline{x}_j$	x_j	Note
$= 0$	any	any	$x_j \in < \underline{x}_j, \overline{x}_j >$	May be fixed on any feasible value
< 0	any	$+\infty$	—	Problem unbounded
< 0	any	$< +\infty$	$x_j = \overline{x}_j$	
> 0	$-\infty$	any	—	Problem unbounded
> 0	$> -\infty$	any	$x_j = \underline{x}_j$	

3.1.3 Infeasible simple bounds

If there should exist a variable x_j such that $\underline{x}_j > \overline{x}_j$ then we declare the LP structurally infeasible.

3.1.4 Fixed variable removal

Whenever we fix a variable x_F (i.e. we determine that $\underline{x}_F = x_F = \overline{x}_F$) we eliminate it (and remove the column $\mathbf{a}_{*F}$) from the problem. We also update the fixed adjustment f and modify vectors $\underline{\mathbf{b}}$ and $\overline{\mathbf{b}}$. If the constraints before the reduction were

$$\begin{array}{l} \underline{\mathbf{b}} \leq \mathbf{A}\mathbf{x} + \mathbf{a}_{*F} x_F \leq \overline{\mathbf{b}} \\ \underline{x}_F = x_F = \overline{x}_F \end{array}$$

then after the reduction we "shift" the constraint activity bounds $\underline{\mathbf{b}}$ and $\overline{\mathbf{b}}$

$$\begin{array}{lcl} \underline{\mathbf{b}} & \leftarrow & \underline{\mathbf{b}} - \mathbf{a}_{*F} x_F \\ \overline{\mathbf{b}} & \leftarrow & \overline{\mathbf{b}} - \mathbf{a}_{*F} x_F \end{array} \tag{7}$$

and update the fixed adjustment

$$f \leftarrow f + x_F c_F. \tag{8}$$

3.1.5 Singleton row conversion to variable bounds

A singleton row of the form

$$\underline{b}_i \leq a_{ij} x_j \leq \overline{b}_i \tag{9}$$

may be converted to simple bounds on variable x_j and then removed from the problem. If $a_{ij} > 0$ the resulting bounds are

$$\begin{array}{lcl} \underline{x}_j & \leftarrow & \max(\underline{x}_j, \underline{b}_i / a_{ij}) \\ \overline{x}_j & \leftarrow & \min(\overline{x}_j, \overline{b}_i / a_{ij}). \end{array} \tag{10}$$

If $a_{ij} < 0$ the direction of inequality (9) changes and the implied bounds change appropriately. The problem is found infeasible or the singleton row is removed.

3.1.6 Computing bounds on dual variables.

As dual feasibility conditions (4) state, each infinite simple bound on a primal variable x_j is equivalent to a bound on the corresponding reduced cost z_j. Whenever we establish such bounds on z_j, a dual constraint becomes an inequality or non-binding. Notably, singleton columns of the primal problem may also be singleton rows of the dual one. The analogy to singleton row reduction is obvious.

3.2 Singleton columns

Our approach is to view all column singletons as possible slack variables. A variable x_S, corresponding to a column singleton with zero cost coefficient c_S shall be called a **slack variable**. The procedure removes a slack variable and converts it to wider bounds on row activity and, possibly, an update of the fixed adjustment.

Let us note that if the variable has its only non-zero in an equality row, it is possible to convert the non-zero cost coefficient to zero. Suppose the i-th row has the form

$$\mathbf{a}_{i*}^T \mathbf{x} + a_{iS} x_S = b_i \tag{11}$$

where a_{iS} is the singleton's only non-zero, and the objective function is

$$f + \mathbf{c}^T \mathbf{x} + c_S x_S. \tag{12}$$

The following equivalence

$$x_S = \frac{1}{a_{iS}} \left(b_i - \mathbf{a}_{i*}^T \mathbf{x} \right) \tag{13}$$

allows to change the objective to

$$\left(f + \frac{c_S}{a_{iS}} b_i \right) + \left(\mathbf{c} - \frac{c_S}{a_{iS}} \mathbf{a}_{i*} \right)^T \mathbf{x} \tag{14}$$

with an updated fixed adjustment and singleton's cost coefficient of zero. Thus the variable x_S may also be considered a slack variable.

Note that:

- there may be more than one slack variable in one row, in which case they all correspond to a single "logical slack" variable,
- a singleton column may belong to a free variable, which means that the row is non-binding,
- a set of explicit slacks and slack variables implied by an inequality row may add up to create a "logical free singleton", which implies that the row is non-binding.

The second case (known as a "free singleton column" reduction) will be treated separately in section 3.2.2.

3.2.1 Removal of slack variables

Sometimes LP's are formulated using only equality constraints (non-equality rows have slack variables explicitly added). This hides the real nature of the variable from the linear optimizer. Some efficient crashing algorithms used in the simplex method (see e.g., [2]) base their success on special treatment of slack variables. There are also some other new linear optimization methods that could benefit from detection of explicitly given slacks (see e.g., Gondzio [9] and Wierzbicki [20]).

A slack variable may be removed by converting its bounds to wider bounds on row activity. Given constraints

$$\begin{aligned} \underline{b}_i \leq \mathbf{a}_{i*}^T\mathbf{x} + a_{iS}x_S \leq \overline{b}_i \\ \underline{x}_S \leq x_S \leq \overline{x}_S \end{aligned} \tag{15}$$

we update its activity limits

$$\begin{aligned} \underline{b}_i &\leftarrow \underline{b}_i - \sup_{\underline{x}_S \leq x_S \leq \overline{x}_S} a_S x_S \\ \overline{b}_i &\leftarrow \overline{b}_i - \inf_{\underline{x}_S \leq x_S \leq \overline{x}_S} a_S x_S \end{aligned} \tag{16}$$

and obtain

$$\underline{b}_i \leq \mathbf{a}_{i*}^T\mathbf{x} \leq \overline{b}_i. \tag{17}$$

The variable x_S is removed from the problem. It is also possible that the above conversion will make the row non-binding and thus redundant.

3.2.2 Free singleton columns

A constraint

$$\begin{aligned} \underline{b} \leq \mathbf{a}_{i*}^T\mathbf{x} + a_{iF}x_F \leq \overline{b} \\ -\infty = \underline{x}_F \leq x_F \leq \overline{x}_F = +\infty \end{aligned} \tag{18}$$

in which a free singleton column (with a_{iF} as the only non-zero) appears is non-binding. It is removed from the problem as it does not influence the primal feasible region.

Analogously, an equality row with a free column singleton may be removed. Let us consider a constraint

$$\mathbf{a}_{i*}^T\mathbf{x} + a_{iF}x_F = b_i (= \underline{b}_i = \overline{b}_i). \tag{19}$$

If the objective coefficient c_F is equal to zero, the row is removed without taking any other actions. Naturally, variable x_F is removed as well.

If however $c_F \neq 0$, we modify the objective in order to bring c_F to zero. Since

$$x_F = \frac{b_i - \mathbf{a}_{i*}^T\mathbf{x}}{a_F} \tag{20}$$

the objective

$$f + \mathbf{c}^T\mathbf{x} + c_F x_F \tag{21}$$

is transformed to

$$\left(f + \frac{c_F}{a_F}b_i\right) + \left(\mathbf{c} \leftarrow \mathbf{c} - \frac{c_F}{a_F}\mathbf{a}\right)^T \mathbf{x}. \tag{22}$$

The i-th row is now removed.

In case of non-equality rows there is one more step we have to take before the row is eliminated. Let us consider a "less than" row

$$\mathbf{a}_{i*}^T\mathbf{x} + a_{iF}x_F \leq \overline{b}_i. \tag{23}$$

By adding a slack variable s we transform it into an equality

$$\begin{aligned} \mathbf{a}_{i*}^T\mathbf{x} + a_{iF}x_F + s = \overline{b}_i \\ 0 \leq s \end{aligned} \tag{24}$$

and modify the objective:

$$\left(f + \frac{c_F}{a_F}\overline{b}_i\right) + \mathbf{c}^T\mathbf{x} - \frac{c_F}{a_F}\left(s + \mathbf{a}_{i*}^T\mathbf{x}\right). \tag{25}$$

The free singleton's cost equals zero and the row is removed.

The difference between this and the previous example is that an empty column of the freshly introduced slack variable s is still left. We treat it as we would treat any other empty column (see section 3.1.2):

- if (c_F/a_F) is positive, the problem is declared unbounded,
- otherwise s is fixed at its lower bound, i.e. at zero.

Similar reasoning leads us to conclusions about a “greater than” row. Note that slack s acts only as a conceptual help and is never actually introduced into the LP.

A general constraint

$$\underline{b}_i \le \mathbf{a}_{i*}^T\mathbf{x} + a_{iF}x_F \le \overline{b}_i \tag{26}$$

is transformed to

$$\begin{array}{c} \mathbf{a}_{i*}^T\mathbf{x} + a_{iF}x_F + s = \overline{b}_i \\ 0 \le s \le \overline{b}_i - \underline{b}_i. \end{array} \tag{27}$$

We fix s at $(\overline{b}_i - \underline{b}_i)$ (and update fixed adjustment) if c_F/a_{iF} is positive, at zero otherwise.

Sometimes a column singleton in doubleton equality row is treated as a special case (see e.g., Andersen and Andersen [1]). If a doubleton row has one entry in a singleton column then bounds on the other variable may be modified so as to make singleton's bounds redundant. The singleton becomes an implied free variable and is treated in a manner described previously. It is easy to see that the methods described so far will eliminate such a doubleton row in two phases:

1. first the singleton variable will be removed (converted into wider bounds on row activity) and the row will have only one non-zero left,
2. then the singleton row will be converted into simple bounds on the other variable.

As it has been shown above the row in which a free singleton column has its non-zero may always be removed. In some cases a slack variable is added, but its only lasting effect is a possible change to the fixed adjustment f. We compute the adjustment update based on the type of row activity bounds and sign of (c_F/a_{iF}) (see table 2).

Table 2: Objective adjustment update after free singleton column removal

Row type:	Value of c_F/a_{iF}		
	$=0$	>0	<0
$\dots \le \overline{b}_i$	0	unbounded solution	$(c_F/a_{iF})\overline{b}_i$
$\underline{b}_i \le \dots$	0	$(c_F/a_{iF})\underline{b}_i$	unbounded solution
$\underline{b}_i \le \dots \le \overline{b}_i$	0	$(c_F/a_{iF})\underline{b}_i$	$(c_F/a_{iF})\overline{b}_i$
$\dots = b_i\ (= \overline{b}_i = \underline{b}_i)$	0	$(c_F/a_{iF})b_i$	

3.2.3 Implied free column singleton

A variable is called “implied free” when its simple bounds may be dropped, because row constraints guarantee that the variable stays within limits as least as tight as those imposed by the simple bounds. If the implied free variable is a column singleton, we may perform a very advantageous free column singleton reduction.

The constraint in which the singleton column x_I, $\underline{x}_I \le x_I \le \overline{x}_I$ has its non-zero

$$\begin{array}{c} \underline{b}_i \le \mathbf{a}_{i*}^T\mathbf{x} + a_{iI}x_I \le \overline{b}_i \\ \underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}} \\ \underline{x}_I \le x_I \le \overline{x}_I \end{array} \tag{28}$$

implies bounds $\underline{x}'_I$ and $\overline{x}'_I$ on variable x_I

$$\begin{aligned}
\underline{x}'_I &= \begin{cases} a_{iI} > 0 & \Rightarrow \quad (\underline{b}_i - \sup\limits_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x})/a_{iI} \\ a_{iI} < 0 & \Rightarrow \quad (\overline{b}_i - \inf\limits_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x})/a_{iI} \end{cases} \\
\overline{x}'_I &= \begin{cases} a_{iI} > 0 & \Rightarrow \quad (\overline{b}_i - \inf\limits_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x})/a_{iI} \\ a_{iI} < 0 & \Rightarrow \quad (\underline{b}_i - \sup\limits_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x})/a_{iI} \end{cases}
\end{aligned} \tag{29}$$

If $\langle \underline{x}'_I, \overline{x}'_I \rangle \subseteq \langle \underline{x}_I, \overline{x}_I \rangle$ then simple bounds on the variable are redundant (we have found an implied free variable). If $\langle \underline{x}'_I, \overline{x}'_I \rangle \cap \langle \underline{x}_I, \overline{x}_I \rangle = \emptyset$, then the problem is declared infeasible. Otherwise the simple bounds on variable are tightened:

$$\begin{aligned} \underline{x}_I &\leftarrow \max(\underline{x}_I, \underline{x}'_I) \\ \overline{x}_I &\leftarrow \min(\overline{x}_I, \overline{x}'_I). \end{aligned} \tag{30}$$

3.3 Individual constraint analysis

Analysis of an individual constraint and comparison with box constraints on the variables involved may reveal that some rows are redundant, some variables may be fixed or new variable bounds may be imposed.

3.3.1 Implied bounds on row activity

A constraint

$$\underline{b}_i \le \mathbf{a}_{i*}^T \mathbf{x} \le \overline{b}_i \tag{31}$$

confronted with variables' bounds

$$\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}} \tag{32}$$

reveals implied limits on row activity:

$$\begin{aligned}
\underline{b}'_i &= \inf_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x} &= \sum_{a_{ij} > 0} a_{ij} \underline{x}_j + \sum_{a_{ij} < 0} a_{ji} \overline{x}_j \\
\overline{b}'_i &= \sup_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x} &= \sum_{a_{ij} > 0} a_{ij} \overline{x}_j + \sum_{a_{ij} < 0} a_{ji} \underline{x}_j .
\end{aligned} \tag{33}$$

If $\left\langle \underline{b}'_i, \overline{b}'_i \right\rangle \cap \left\langle \underline{b}_i, \overline{b}_i \right\rangle = \emptyset$ the problem is declared infeasible. If $\underline{b}'_i = \overline{b}_i$ or $\underline{b}_i = \overline{b}'_i$, we call the row "forcing" as it forces all variables involved to their bounds. The row is removed and the variables are fixed on appropriate bounds. Finally, if $\left\langle \underline{b}'_i, \overline{b}'_i \right\rangle \supseteq \left\langle \underline{b}_i, \overline{b}_i \right\rangle$ the i-th constraint is redundant ("dominated") and removed.

3.3.2 Tightening variable bounds

By reversing the procedure presented above we compute variable bounds implied by the row constraints. This process helps to provide more finite simple bounds for dominated and forcing constraint detection.

An example of implied variable bounds computation was provided in section 3.2.3. Identical procedure is employed here:

$$\begin{gathered} \forall i \in \{1, \ldots, m\} \quad \forall j \in \{1, \ldots, n\} \\ \left(\underline{b}_i \le \mathbf{a}_{i*}^T \mathbf{x} + a_{ij} x_j \le \overline{b}_i \right) \Rightarrow \left(\underline{b}_i - \sup_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x} \le a_{ij} x_j \le \overline{b}_i - \inf_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x} \right) \end{gathered} \tag{34}$$

but the purpose is different: we attempt to tighten variable bounds as much as possible.

Note that a procedure for global bounds cross-checking according to the above formula will involve as many bound computations, as there are non-zeros in the constraint matrix. Additionally, if two (or more) variables are active in two (or more) constraints, it is possible that a change of a bound on one variable will necessitate new computation of bound on the other (or others). [1]

This procedure is too expensive and unreliable, as it can cause infinite loops. A much more efficient approach to has been proposed by Gondzio [10] who has observed, that bounds on row activity (which are calculated in forcing and dominated row detection routines) may be used to cheaply compute the implied variable bounds. If both i-th row activity and j-th variable have at least one finite bound each, a simple calculation may provide finite implied bounds.

We can compute $\inf_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x}$ and $\sup_{\underline{\mathbf{x}} \le \mathbf{x} \le \overline{\mathbf{x}}} \mathbf{a}_{i*}^T \mathbf{x}$ cheaply and then calculate implied variable bounds efficiently. We know the following bounds on row activity:

$$\begin{array}{rclcl} \underline{a}_i &=& \inf_{\mathbf{x}, x_j} \left(\mathbf{a}_{i*}^T \mathbf{x} + a_{ij} x_j \right) &=& \inf_{\mathbf{x}} \mathbf{a}_{i*}^T \mathbf{x} + \inf_{x_j} a_{ij} x_j \\ \overline{a}_i &=& \sup_{\mathbf{x}, x_j} \left(\mathbf{a}_{i*}^T \mathbf{x} + a_{ij} x_j \right) &=& \sup_{\mathbf{x}} \mathbf{a}_{i*}^T \mathbf{x} + \sup_{x_j} a_{ij} x_j \end{array} \tag{36}$$

from which it follows

$$\begin{array}{rcl} \inf_{\mathbf{x}} \mathbf{a}_{i*}^T \mathbf{x} &=& \underline{a}_i - \inf_{x_j} a_{ij} x_j \\ \sup_{\mathbf{x}} \mathbf{a}_{i*}^T \mathbf{x} &=& \overline{a}_i - \sup_{x_j} a_{ij} x_j . \end{array} \tag{37}$$

Finally for $a_{ij} > 0$ we have

$$\frac{\underline{b}_i - \overline{a}_i}{a_{ij}} + \overline{x}_j \le x_j \le \frac{\overline{b}_i - \underline{a}_i}{a_{ij}} + \underline{x}_j \tag{38}$$

and for $a_{ij} < 0$

$$\frac{\overline{b}_i - \underline{a}_i}{a_{ij}} + \overline{x}_j \le x_j \le \frac{\underline{b}_i - \overline{a}_i}{a_{ij}} + \underline{x}_j . \tag{39}$$

Needless to say, the above implied bounds may still be infinite.

A complete constraint matrix scan consists of the following computations:

1. for each $i \in \{1, \ldots, m\}$ a pair of implied row activity bounds $\underline{a}_i$ and $\overline{a}_i$ is computed,

2. for each row i with at least one finite activity bound and for each $j \in \{1, \ldots, n\}$ such that $a_{ij} \ne 0$ and $\underline{x}_j > -\infty$ or $\overline{x}_j < +\infty$ we compute implied variable bounds.

3.3.3 Dominated and weakly dominated variables

It is possible to apply some of the analysis methods presented in section 3.3.1 to the dual problem. This will allow us to detect and eliminate forcing and dominated variables and fix some variables on their finite bounds.

Each dual constraint $\mathbf{a}_{*j}$ implies bounds on associated reduced cost z_j. Whenever we are able to determine that the sign of the reduced cost z_j is strictly positive of strictly negative, we can fix variable x_j at one of its bounds. From

$$\underline{z}_j' = c_j - \sup \mathbf{a}_{*j}^T \mathbf{y} \le z_j \le c_j - \inf \mathbf{a}_{*j}^T \mathbf{y} = \overline{z}_j' \tag{40}$$

and the dual feasibility conditions (4) it follows that

[1]The reader may wish to analyze a small LP example:

$$\begin{array}{rcrcl} x_1 &+& x_2 &\le& 1 \\ 2x_1 &+& x_2 &=& 2 \\ && \multicolumn{3}{l}{x_1,\ x_2 \ge 0} \end{array} \tag{35}$$

in which the only feasible point is $x_1 = 1$, $x_2 = 0$. A possibly infinite cycle of bound tightening would occur. The feasible intervals would slowly converge to the solution.

- if the bounds above are inconsistent with those previously known, then the dual problem is infeasible,
- if $\underline{z}'_j > 0$ ($\overline{z}'_j < 0$), we say that variable x_j is dominated and may be fixed at its finite lower (upper) bound, respectively; if the appropriate primal variable bound is infinite, the the problem is declared unbounded,
- if $\underline{z}'_j = 0$ or $\overline{z}'_j = 0$, then x_j may be a so called "weakly dominated" variable.

Andersens [1] give a definition which enables them to treat weakly dominated variables as dominated ones. They require that the bounds on the dual variables $\mathbf{y}$ are derived from singleton columns and that those singletons are not removed from the problem.

Conversely, Gondzio [10] proposes a more general approach in which he allows to use dual variables' bounds of any origin and does not rely on the singletons' existence. Instead he imposes some requirements concerning row types of the matrix rows concerned (see Gondzio [10] for the theorem as well as the proof):

1. If $\underline{z}_j \geq 0$, $\underline{x}_j > -\infty$, and
$$\forall i = 1,\ldots,m \begin{cases} a_{ij} > 0 & \Rightarrow \quad \underline{b}_i = -\infty \\ a_{ij} < 0 & \Rightarrow \quad \overline{b}_i = +\infty \end{cases} \tag{41}$$
then the variable x_j is weakly dominated and it may be fixed at its lower bound.
2. If $\overline{z}_j \leq 0$, $\overline{x}_j < +\infty$, and
$$\forall i = 1,\ldots,m \begin{cases} a_{ij} > 0 & \Rightarrow \quad \overline{b}_i = +\infty \\ a_{ij} < 0 & \Rightarrow \quad \underline{b}_i = -\infty \end{cases} \tag{42}$$
then the variable x_j is weakly dominated and it may be fixed at its upper bound.

3.4 Linear transformations

Chang and McCormic [5] and earlier McCormic [14] have presented an algorithm for solving of a so-called "sparsity problem". Given a sparse matrix $\mathbf{A}$, $\mathbf{A} \in \Re^{m \times n}$, a non-singular matrix $\mathbf{M}$, $\mathbf{M} \in \Re^{m \times m}$ is to be found such that $\mathbf{MA}$ is sparsest possible. The problem arose from consideration of possible ways of reducing time needed to solve a linear optimization problem (1)–(2). It was assumed that an equivalent problem

$$\begin{gathered} \min \mathbf{c}^T \mathbf{x} \\ \mathbf{M}\underline{\mathbf{b}} \leq (\mathbf{MA})\mathbf{x} = \mathbf{M}\overline{\mathbf{b}} \\ \underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}} \end{gathered} \tag{43}$$

will in general be solved faster by a simplex optimizer than the original one. The results reported by McCormic [14] were not encouraging: despite savings in optimization time, the time spent finding $\mathbf{M}$ in all cases exceeded the savings. Only more recent results of Chang and McCormic [5] documeneted an overall gain in the range of 10%.

For this reason we have decided to implement a much less time consuming heuristic algorithm that would perform numerical eliminations. It has been originally described by Gondzio [10].

3.4.1 Numerical elimination heuristic

In a linear problem (1)–(2) transformed to equality form we find such row pairs, in which one row has a non-zero pattern, which is a superset of the non-zero pattern of the other. The shorter row is then used as a pivot row in elimination of at least one non-zero of the longer one. Of course further non-zero cancellations may occur. Among the main advantages of this procedure are its simplicity and effectiveness, ability to reduce problem density and potential to eliminate some duplicate rows.

The main computational effort goes into non-zero pattern comparison. The data structures should facilitate efficient access to both rows and columns of the constraint matrix. We found duplicate storage of the constraint matrix (row-wise and column-wise) very helpful. See Świętanowski [19] for a list of reasons why a simplex optimizer benefits from such double storage.

4 Optimal solution recovery

In this paper we are only concerned with recovery of the values of the primal variables $\mathbf{x}$. Therefore we will proceed to present postsolve methods only for those presolve techniques, which affect their values. Primal variables are removed from the problem when they are fixed, found to be free singletons or found to be slack.

Reader interested in recovery of dual variables and reduced costs is referred to papers by Andersens [1], Brearley et al. [4] and Gondzio [10]. Only [10] presents methods for recovery of dual variables after linear transformations.

4.1 Fixed variable's value recovery

This is a trivial task. The value of the variable is known at the moment of fixing, therefore it may simply be stored and later retrieved during the postsolve phase.

4.2 Free singleton's value recovery

As it was shown in section 3.2.2, the free variable's value may be calculated as

$$x_F = \frac{b_i - \mathbf{a}_{i*}^T \mathbf{x} - s}{a_{iF}} \tag{44}$$

where

$$b_i = \begin{cases} \overline{b}_i & \text{if } \overline{b}_i < +\infty \\ \underline{b}_i & \text{if } \overline{b}_i = +\infty \\ \underline{b}_i = \overline{b}_i & \text{if } \underline{b}_i = \overline{b}_i \end{cases} \tag{45}$$

and

$$s = \begin{cases} \overline{b}_i - \underline{b}_i & \text{if } \underline{b}_i > -\infty \text{ and } \overline{b}_i < +\infty \\ 0 & \text{otherwise.} \end{cases} \tag{46}$$

4.3 Explicit slack's recovery

Recovering the values of explicitly defined slack variables is a more complex problem. However, the difficulty may only arise when more than one variable in the same constraint matrix row is detected to be a slack. We believe it to be a relatively rare case. In our implementation this possibility has been excluded: only one explicit slack reduction per row is allowed.

In case of a general constraint, the variable x_S may take any value which satisfies one of the two sets of inequalities given below:

$$a_{iS} > 0 \Rightarrow \begin{cases} \frac{1}{a_{iS}} \left(\underline{b}_i - \mathbf{a}_{i*}^T \mathbf{x} \right) \le x_S \le \frac{1}{a_{iS}} \left(\overline{b}_i - \mathbf{a}_{i*}^T \mathbf{x} \right) \\ \underline{x}_S \le x_S \le \overline{x}_s \end{cases} \tag{47}$$

or

$$a_{iS} < 0 \Rightarrow \begin{cases} \frac{1}{a_{iS}} \left(\overline{b}_i - \mathbf{a}_{i*}^T \mathbf{x} \right) \le x_S \le \frac{1}{a_{iS}} \left(\underline{b}_i - \mathbf{a}_{i*}^T \mathbf{x} \right) \\ \underline{x}_S \le x_S \le \overline{x}_s \end{cases} . \tag{48}$$

Naturally, the recovery of explicit slack variables is not strictly deterministic.

Unless there is more than one explicit slack per row, any solution to the corresponding system of inequalities is acceptable. Otherwise, for each row with more than one explicit slack eliminated, we would need to find a feasible solution of a combined system of *all* inequalities resulting from separate consideration of *all* slacks removed from that row.

5 Experimental results

In this section we present the results of some numerical experiments conduced with the presolve procedure. Each subset of the presolve methods is analysed with respect to possible gains in computation time when the presolved problem is solved with a simplex type optimizer. The theoretical speculations are supported by tables with computation times and iteration counts of a revised simplex method implementation (see Świętanowski [19]).

Thus far not all of the presolve methods described in this paper have been actually implemented. Dominated and weakly dominated variable reduction methods together with dual variable bound tightening procedures are still missing. Therefore, in lack of numerical evidence, some of the conclusions are of preliminary nature.

The test problems chosen are the largest ones of over a hundred LP's from the extended NETLIB test collection initiated by Gay ([7]). The short characteristics of those problems are given in table 3.

The numerical tests were performed on a CRAY SuperServer 6400 shared memory multiprocessor. However, since the code is entirely sequential (and portable to other platforms, e.g., SUN SparcStation, IBM PC 386, IBM RS6000) all the computation times are given in seconds of sequential CPU work. The times quoted are those measured by a Solaris operating system function `times()` and always refer to entire computation time: presolving (if performed), conversion to standard form, scaling and solution. The reader should take into account the inaccuracy of time measurement (in the range of a few percentage points) resulting from different system loads.

All comparative tables in this paper contain ratios expressed in per cent, e.g., percent of eliminated constraint matrix rows, or percent of computation time saved when a certain presolve technique is used. Time savings are computed according to formulas like

$$\text{saving} = \frac{\text{time\_without\_presolve} - \text{time\_with\_presolve}}{\text{time\_without\_presolve}} \cdot 100\%.$$

All those tables also list average and average deviation for each table column. For data x_j, $j \in \{1, \ldots, n\}$ the average $\overline{x}$ is

$$\overline{x} = \frac{1}{n} \sum_{j=1}^{n} x_j$$

and the average deviation $\tilde{x}$ is

$$\tilde{x} = \frac{1}{n} \sum_{j=1}^{n} |x_j - \overline{x}|.$$

Two of the test problems were not solved in their original form due to numerical difficulties, therefore it is impossible to calculate computation time ratios for those problems. Appropriate positions in the tables are marked with "???". Whenever these problems were not solved successfully with a certain presolve technique, the tables contain "num. diff." instead of the ratios.

5.1 Usefulness of the simple presolve methods

By simple presolve methods we mean those that require little or no searches, comparisons and floating point operations. They include removal of empty rows and columns, conversion of singleton rows into variable bounds and removal of fixed variables. They are singled out because

Table 3: Test problems – the summary

Name	Rows	Columns	Non-zeros	Density [%]	Scaling $\log_{10} \frac{\max_{a_{ij}\neq 0} \lvert a_{ij}\rvert}{\min_{a_{ij}\neq 0} \lvert a_{ij}\rvert}$
25FV47	822	1571	11127	0.86	6.08
80BAU3B	2263	9799	29063	0.13	5.68
BNL2	2325	3489	16124	0.20	5.11
CRE-A	3517	4067	19054	0.13	2.07
CRE-C	3069	3678	16922	0.15	2.15
CYCLE	1904	2857	21322	0.39	7.96
CZPROB	930	3523	14173	0.43	4.94
D2Q06C	2172	5167	35674	0.32	7.07
D6CUBE	416	6184	43888	1.71	2.56
DEGEN3	1504	1818	26230	0.96	0.00
FIT1P	628	1677	10894	1.03	0.00
FIT2P	3001	13525	60784	0.15	0.00
GREENBEA	2393	5405	31499	0.24	6.22
GREENBEB	2393	5405	31499	0.24	6.22
KEN-07	2427	3602	11981	0.14	0.00
KEN-11	14695	21349	70354	0.02	0.00
KEN-13	28633	42659	139834	0.01	0.00
MAROS-R7	3137	9408	151120	0.51	2.78
MAROS	847	1443	10006	0.82	8.23
NESM	663	2923	13988	0.72	4.52
OSA-07	1119	23949	167643	0.63	1.65
OSA-14	2338	52460	367220	0.30	1.65
OSA-30	4351	100024	700160	0.16	1.92
PDS-02	2954	7535	21252	0.10	0.00
PILOT	1442	3652	43220	0.82	8.16
PILOT87	2031	4883	73804	0.74	9.00
PILOT.JA	941	1988	14706	0.79	12.50
PILOTNOV	976	2172	13129	0.62	12.50
SCSD8	398	2750	11334	1.04	0.62
SCTAP3	1481	2480	10734	0.29	1.90
SHIP08L	779	4283	17085	0.51	2.65
SHIP12L	1152	5427	21597	0.35	2.41
SHIP12S	1152	2763	10941	0.34	2.41
STOCFOR3	16676	15695	74004	0.03	3.73
TRUSS	1001	8806	36642	0.42	0.35
WOODW	1099	8405	37478	0.41	5.00

it is possible to implement them easily and efficiently and they may be embedded directly in a linear optimizer at almost no cost.

Empty rows (or columns) are a rather rare occurrence, however most presolve methods may cause elimination of all non-zeros in some rows (columns). The same is true for inconsistent variable simple bounds: they may result form other analysis methods. On the other hand fixed variables and singleton rows are rather common (at least in the NETLIB set). It is worth noting that singleton rows may result from a conversion of general linear constraints (2) to a standard form, in which all variables are subject to non-negativity bounds $\mathbf{0} \leq \mathbf{x}$ in which case all finite lower bounds are shifted to zero and upper bounds are transformed into singleton rows.

We will show that all those redundancies should not affect significantly the performance of the revised simplex method.

Let us assume that there is an empty column $\mathbf{a}_{*j} = \mathbf{0}$ in the constraint matrix. It is clear that it cannot be introduced into the initial basis. Its reduced cost is equal to its objective function coefficient $z_j = c_j - \mathbf{c}_{\mathbf{B}}^T \mathbf{B}^{-1} \mathbf{a}_{*j} = c_j$. If $c_j = 0$ the variable will never become a candidate to enter the basis. Otherwise, the x_j will move between its bounds (if they are both finite) or the problem will be declared unbounded.

If the constraint matrix contains an empty row $\mathbf{a}_{i*}$, an artificial variable (possibly fixed at zero) will be added to i-th row in order to construct a non-singular basis. The variable will never leave the basis and although it may be structurally degenerate, it will never cause degenerate iterations. Furthermore, most modern factorization routines (based on Bartels–Golub or Forrest–Tomlin algorithms — see, e.g., Reid [16]) will locate a singleton row of the basis matrix and will permute it rather than use it in eliminations. Clearly, most of the analysis given above applies directly to singleton rows.

Finally, the fixed variable elimination has to be examined. Typically, fixed variables never enter the basis, except for the initial basis creation in some crashing schemes (see, e.g., Świetanowski [17] or [18]). Their presence in the basis is likely to cause degeneration, but they are the first candidates to leave the basis. We may conclude that their impact on solution times should not be too great.

It has previously been stated that the problem's difficulty is proportional not only to the dimension of the constraint matrix, but also to its number of non-zeros. It is clear that removal of empty or singleton rows and columns must result in increased matrix density.

All in all we do not expect the simple reductions to be very advantageous by themselves. Any substantial changes in computation time should rather be contributed to a change of optimization path followed by the simplex algorithm. These claims are substantiated by data collected in table 4 where solution times and iteration counts with and without the simple presolve methods are compared.

The table lists the percentage of rows, columns and non-zeros removed from the constraint matrices as well as iteration and solution time savings. Ten of the problems were not reduced, but presolving caused only a negligible loss of time. On the average the number of iterations needed to solve the problems has decreased by 1.62% and the average time was cut by 8.33%. Both these average reductions are rather small, which supports our speculations. The simple methods caused a meaningful loss of time in case of only one problem: 25FV47. On the other hand, sometimes they provide savings much exceeding the ratios of problem size reduction. The latter phenomenon is probably only an example of simplex algorithm's sensitivity to initial basis choice.

It may also be noted that whenever a significant portion (10% or more) of the problem's rows, columns or non-zeros is removed, a time saving is always seen. To conclude: large reductions are always helpful, but negligible ones only change optimization path and thus distort iteration counts and times. Simple presolve methods are apparently too simple to reduce many problems.

Table 4: Efficiency of simple presolve methods

Name	Eliminated [%]			Improvement [%]	
	Rows	Columns	Non-zeros	Iter.	Time
25FV47	5.35	1.65	1.38	−11.94	−8.02
80BAU3B	9.72	5.70	3.06	−3.23	4.81
BNL2	7.66	0.97	1.27	23.40	33.63
CRE-A	12.48	0.15	1.96	−2.64	8.86
CRE-C	14.40	0.73	2.55	5.37	19.14
CYCLE	7.72	3.75	2.86	15.32	19.17
CZPROB	20.65	11.89	8.83	9.64	6.11
D2Q06C	3.36	0.19	0.27	8.64	17.21
DEGEN3	0.00	0.00	0.00	0.00	−0.04
FIT1P	0.00	0.00	0.00	0.00	0.00
FIT2P	0.00	0.00	0.00	0.00	−0.02
GREENBEA	3.18	3.26	2.03	−3.91	2.73
GREENBEB	3.26	3.52	2.25	2.05	11.59
KEN-07	40.75	27.46	20.12	0.58	24.26
KEN-11	31.36	21.59	14.98	−0.06	9.20
KEN-13	21.30	14.29	10.40	−0.72	15.35
MAROS-R7	0.00	0.00	0.00	0.00	−0.07
MAROS	4.13	3.60	1.61	3.11	2.91
NESM	2.41	6.26	1.67	5.57	13.26
OSA-07	0.00	0.00	0.00	0.00	−0.59
OSA-14	0.00	0.00	0.00	0.00	−0.32
OSA-30	0.00	0.00	0.00	0.00	−0.24
PDS-02	10.12	3.94	2.85	−5.14	5.45
PILOT	0.55	5.61	4.83	num. diff.	
PILOT87	0.74	4.61	3.38	???	
PILOT.JA	2.23	15.85	19.65	5.87	13.05
PILOTNOV	5.53	10.73	7.08	−1.48	5.02
SCSD8	0.00	0.00	0.00	0.00	0.00
SCTAP3	0.00	0.00	0.00	0.00	0.00
SHIP08L	11.55	0.56	0.42	−1.82	7.32
SHIP12L	27.17	3.76	2.83	−4.10	10.31
SHIP12S	59.46	20.85	15.79	−1.73	34.25
STOCFOR3	0.35	0.20	0.42	3.12	5.68
TRUSS	0.00	0.00	0.00	0.00	−0.02
WOODW	0.09	0.00	0.00	7.67	14.98
Average	8.73	4.89	3.79	1.62	8.33
Avg. dev.	9.31	5.22	4.08	4.36	7.73

5.2 The advantages of column singleton reductions

It has been shown in section 5.1 that the removal of a singleton (or empty) row does not bring about significant decrease in problem solution time. Similar reasoning leads us to believe that removal of a slack variable which is not followed by further reductions is also of very little value. We ought to remember that a simplex optimizer will introduce a slack variable for every non-equality row prior to solving the problem. It follows that only when we manage to remove a slack from an inequality row, the optimizer will actually solve a reduced problem. However, a free singleton variable removal ought to be advantageous, because a whole constraint matrix row is removed.

Table 5 summarizes the reductions obtained by the singleton column analysis. Empty rows and columns are also removed.

The average reductions (1.51% in terms of iteration number and 4.12% in terms of time) are indeed negligible. Both these reductions are highly problem-dependent (which is highlighted by significant average deviation factors). Eleven problems were solved more than 5% faster and 5 LP's lost more than 5% of computation time. Time gains are typically noted when the number of free (and implied free) column singletons removed is over 0.5%.

Surprisingly, we note that removal of explicit slack variables (listed in the table under the heading "relaxed constraints") almost always coincides with iteration and time losses. We are unable to present any satisfying explanation for this phenomenon.

5.3 Usefulness of row constraint analysis

The simplex method should not be affected greatly by presence of a number of dominated constraints. They are always fulfilled and so they never cause degeneration. Their only impact is on the size and density of the subsequent simplex bases, but their presence in the optimal basis will be limited to slack variables (they are inactive at the optimum). Elimination of a dominated row will probably give way to further reductions (e.g., by producing new column singletons).

The forcing constraints — if left undetected — are structurally degenerate and thus much more damaging to simplex method's efficiency. A forcing constraint is eliminated together with all its variables, which is yet another benefit.

These elimination methods rely on presence of the simple presolve techniques as well as on bound tightening, which makes some of the reductions possible. Table 6 summarizes the results gathered when using those techniques.

The overall gains are quite impressive: 7.10% of iterations and 18.88% of computation time. Only three problems lost more than 2% of time. It must be noted, that whenever any forcing constraints are eliminated from the problem, both iteration counts and times are improved (often by as much as 20, 40 or even 60%). On the other hand, dominated row elimination may still lead to computation time loss (see e.g., problems OSA-30 and SCTAP3).

5.4 Advantages of numerical eliminations

We expect that decreased sparsity of the constraint matrix will be reflected by reduced average simplex basis density. This in turn should allow faster factorizations and linear system solution (with the right hand side vectors also sparser). We thus predict an overall better efficiency when solving reduced problems.

Table 7 presents the results obtained when numerical elimination was applied, supported by empty row and column removal and singleton row elimination, which were included because numerical eliminations are likely to create empty rows and columns as well as singleton rows.

The table shows small average improvements (1.22% loss in iterations and 6.64% time gain) with large average deviation. Nine problems were not reduced at all, 19 cut the time by more than 5% and 3 lost more than 5%. Again, it seems that large reductions (10% or more) guarantee

Table 5: Efficiency of singleton column analysis

Name	Eliminated [%]					Improvement [%]	
	Rows	Columns	Non-zeros	Free singl.	Relaxed constr.	Iter.	Time
25FV47	0.61	1.72	0.44	0.32	2.68	−11.59	−10.09
80BAU3B	1.68	1.80	0.83	0.32	0.31	1.11	5.95
BNL2	16.17	12.55	6.89	9.29	2.41	21.15	33.47
CRE-A	0.00	0.00	0.00	0.00	0.00	0.00	−0.05
CRE-C	0.00	0.00	0.00	0.00	0.00	0.00	−0.06
CYCLE	7.04	9.28	4.77	4.69	5.93	20.41	17.52
CZPROB	2.15	0.57	25.84	0.57	0.00	19.21	21.11
D2Q06C	1.38	15.79	4.75	0.56	36.19	6.06	3.98
DEGEN3	0.00	0.55	0.04	0.00	0.66	5.37	7.69
FIT1P	0.00	37.39	5.76	0.00	99.84	−9.92	−1.69
FIT2P	0.00	22.18	4.94	0.00	99.97	−6.88	−1.93
GREENBEA	0.96	0.70	0.59	0.43	0.59	−13.51	−13.06
GREENBEB	0.92	0.68	1.43	0.41	0.59	4.63	14.35
KEN-07	0.00	0.69	0.21	0.00	1.03	2.46	4.26
KEN-11	0.00	0.25	0.08	0.00	0.36	−0.76	−10.19
KEN-13	0.00	0.17	0.05	0.00	0.25	0.82	−2.16
MAROS-R7	31.37	43.79	30.78	10.46	68.60	−3.73	8.28
MAROS	4.72	7.42	5.93	2.77	6.85	−1.71	2.43
NESM	0.00	10.95	2.29	0.00	48.27	−2.44	−2.49
OSA-07	0.00	0.00	0.00	0.00	0.00	0.00	−0.59
OSA-14	0.00	0.00	0.00	0.00	0.00	0.00	−0.68
OSA-30	0.00	0.00	0.00	0.00	0.00	0.00	−0.18
PDS-02	0.00	0.00	0.00	0.00	0.00	0.00	−0.11
PILOT	0.62	0.49	0.08	0.25	0.62	???	
PILOT87	1.33	0.70	0.42	0.55	0.34	???	
PILOT.JA	4.25	7.04	3.27	2.01	10.63	−3.83	−4.61
PILOTNOV	7.38	8.79	4.96	3.31	10.45	−0.23	0.00
SCSD8	0.00	0.22	0.05	0.00	1.51	−11.59	−3.02
SCTAP3	0.00	25.00	5.78	0.00	41.86	−8.38	−8.14
SHIP08L	6.42	1.17	18.62	1.17	0.00	23.55	27.64
SHIP12L	6.68	1.42	19.55	1.42	0.00	10.05	14.43
SHIP12S	6.68	2.79	19.83	2.79	0.00	17.66	23.29
STOCFOR3	7.68	8.16	12.81	8.16	0.00	5.44	19.92
TRUSS	0.00	0.05	0.01	0.00	0.40	−5.85	−6.15
WOODW	0.36	2.00	16.20	0.05	13.83	−7.64	−3.33
Average	3.10	6.41	5.63	1.41	12.98	1.51	4.12
Avg. dev.	3.85	7.51	6.06	1.84	18.16	7.23	9.00

Table 6: Efficiency of row analysis

Name	Eliminated [%]					Improvement [%]	
	Rows	Columns	Non-zeros	Forcing constr.	Dominated constr.	Iter.	Time
25FV47	5.35	1.65	1.38	0.00	0.00	−11.94	−9.48
80BAU3B	10.83	6.16	3.52	0.18	0.71	6.73	17.92
BNL2	9.29	1.63	2.67	0.26	0.99	15.92	27.89
CRE-A	15.21	3.17	4.16	0.37	0.00	3.08	15.05
CRE-C	24.89	11.66	10.71	2.31	0.00	14.94	35.34
CYCLE	23.48	11.45	24.83	3.78	7.93	16.29	33.38
CZPROB	30.97	21.37	16.45	5.16	0.00	20.22	37.22
D2Q06C	3.36	0.19	0.27	0.00	0.00	9.00	14.67
DEGEN3	0.00	0.00	0.00	0.00	0.00	0.00	−0.04
FIT1P	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FIT2P	0.00	0.00	0.00	0.00	0.00	0.00	−0.02
GREENBEA	28.83	23.52	21.66	10.41	0.42	17.07	31.93
GREENBEB	28.79	23.72	21.85	10.28	0.42	30.15	46.21
KEN-07	40.75	27.46	20.12	0.00	0.00	2.78	39.34
KEN-11	31.36	21.59	14.98	0.00	0.00	−0.55	31.43
KEN-13	21.30	14.29	10.40	0.00	0.00	3.39	24.39
MAROS-R7	0.00	0.00	0.00	0.00	0.00	0.00	−0.11
MAROS	26.56	26.40	28.44	6.97	1.18	15.03	28.16
NESM	2.41	6.26	1.67	0.00	0.00	5.57	16.57
OSA-07	3.31	0.00	33.08	0.00	3.31	−1.90	10.98
OSA-14	1.58	0.00	32.77	0.00	1.58	2.83	7.98
OSA-30	0.85	0.00	32.43	0.00	0.85	−5.74	−3.62
PDS-02	12.90	4.82	3.99	1.12	0.14	1.14	11.01
PILOT	5.41	7.37	5.53	1.73	0.55	???	
PILOT87	2.76	5.35	3.57	0.89	0.25	num. diff.	
PILOT.JA	14.98	19.97	22.77	3.40	4.04	6.75	15.35
PILOTNOV	12.40	12.20	9.12	1.64	3.59	7.07	16.81
SCSD8	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SCTAP3	4.86	0.00	2.60	0.00	4.86	−37.60	−37.21
SHIP08L	53.66	26.48	20.23	20.54	1.03	28.48	47.15
SHIP12L	52.95	22.17	16.96	12.67	0.43	21.70	40.72
SHIP12S	67.53	27.76	21.52	3.82	0.43	14.08	49.32
STOCFOR3	0.35	0.20	0.42	0.00	0.00	4.46	8.40
TRUSS	0.00	0.00	0.00	0.00	0.00	0.00	−0.02
WOODW	70.34	36.29	38.24	35.12	0.00	45.43	66.37
Average	17.35	10.38	12.18	3.45	0.93	7.10	18.88
Avg. dev.	15.76	9.75	10.49	4.44	1.15	9.88	16.64

Table 7: Efficiency of numerical eliminations

Name	Eliminated [%]			Improvement [%]	
	Rows	Columns	Non-zeros	Iter.	Time
25FV47	5.35	1.65	2.73	−3.67	5.71
80BAU3B	9.72	5.70	3.06	−3.23	5.09
BNL2	7.66	0.97	3.00	16.88	26.45
CRE-A	12.48	0.15	3.91	−0.38	10.43
CRE-C	14.40	0.73	4.21	−6.94	5.65
CYCLE	7.83	3.82	5.78	8.82	15.86
CZPROB	20.65	11.89	8.83	9.64	22.78
D2Q06C	3.41	0.21	1.55	−32.17	−34.01
DEGEN3	0.00	0.00	1.08	1.83	5.29
FIT1P	0.00	0.00	0.00	0.00	0.00
FIT2P	0.00	0.00	0.00	0.00	−0.40
GREENBEA	3.18	3.26	3.25	−19.41	−16.99
GREENBEB	3.26	3.52	3.48	−8.61	2.28
KEN-07	40.75	27.46	20.12	0.58	35.74
KEN-11	31.36	21.59	14.98	−0.06	30.82
KEN-13	21.30	14.29	10.40	−0.72	19.22
MAROS-R7	0.00	0.00	0.00	0.00	−0.76
MAROS	4.13	3.60	3.33	−12.12	−15.05
NESM	2.41	6.26	1.67	5.57	17.68
OSA-07	0.00	0.00	0.00	0.00	−0.59
OSA-14	0.00	0.00	0.00	0.00	−0.32
OSA-30	0.00	0.00	0.00	0.00	−0.14
PDS-02	10.12	3.94	2.85	−5.14	6.11
PILOT	0.55	5.61	4.88	???	
PILOT87	0.74	4.61	3.40	???	
PILOT.JA	2.23	15.85	20.72	8.38	12.94
PILOTNOV	5.53	10.73	7.98	−0.69	−0.22
SCSD8	0.00	0.00	0.00	0.00	−0.38
SCTAP3	0.00	0.00	0.00	0.00	0.00
SHIP08L	11.55	0.56	0.42	−1.82	5.69
SHIP12L	27.17	3.76	2.83	−4.10	7.73
SHIP12S	59.46	20.85	15.79	−1.73	34.25
STOCFOR3	0.35	0.20	0.42	3.12	7.24
TRUSS	0.00	0.00	0.00	0.00	0.34
WOODW	0.09	0.00	0.16	5.56	10.65
Average	8.73	4.89	4.31	−1.22	6.64
Avg. dev.	9.31	5.22	4.04	5.18	10.03

time savings, while small ones may only change the route the simplex algorithm takes on its way to optimum. Insignificance of small reductions may partially be explained by the fact that the LU factorization scheme would perform most of them.

6 Conclusions

The ultimate results obtained by our presolve procedure are presented in table 8. The total gain measured by average decrease in computation time is smaller than might be expected after the partial results from the previous sections, especially after the row analysis methods.

Apparently, the result of all presolve techniques working together is not much better than the row analysis methods supported only by simple presolve techniques. Both the average time gain and its deviation are almost 20%, which points out again that each linear problem reacts differently to presolve analysis. This time all problems were reduced, even if the smallest reduction was in the range of 0.01% of non-zeros (problem TRUSS) and caused a 9% time loss. The encouraging result is that 24 problems benefited from analysis (and 23 of them by at least 10%) while only 5 lost more than 5% of time. The results already quoted in section 5 could be used as a guideline as to which presolve procedures are worth implementing by themselves and which may only prove advantageous when used in conjunction with a whole set of other analysis methods. Again, as in section 5.2, we notice that constraint relaxation usually coincides with iteration and time losses.

Other general conclusions that may be drawn from the results are:

- the presolve analysis methods may significantly reduce solution time of linear problems and
- the implementation described in this paper gives very encouraging results, even though it is still incomplete,
- addition of dominated variable detection procedure may still allow us to provide even more reliability (measured by the ratio of problems that benefit to those that do not).

7 Acknowledgements

The author wishes to thank Dr. Jacek Gondzio for many helpful comments during the preparation of this paper. We are also grateful to Dr. Marek Makowski for providing the binary input-output library LP-DIT for our applications as well as many interesting real-life linear problems.

Table 8: Global efficiency of presolve analysis

Name	Eliminated [%]							Improvement [%]	
	Rows	Columns	Non-zeros	Free singl.	Relaxed constr.	Forcing constr.	Dom. constr.	Iter.	Time
25FV47	5.96	4.01	3.27	0.25	4.01	0.00	0.00	−17.55	−11.66
80BAU3B	12.28	6.50	4.25	0.30	0.22	0.18	0.66	6.42	15.21
BNL2	25.63	14.42	11.36	9.43	2.54	0.26	0.95	20.74	41.12
CRE-A	15.21	3.17	6.20	0.00	0.00	0.37	0.00	3.30	16.10
CRE-C	24.89	11.66	12.20	0.00	0.00	2.31	0.00	8.77	29.21
CYCLE	29.15	19.18	32.43	3.89	5.88	2.52	9.09	11.64	33.79
CZPROB	32.90	21.88	36.44	0.51	0.00	5.16	0.00	25.35	43.89
D2Q06C	6.91	16.12	5.85	0.50	36.60	0.00	0.00	−22.78	−33.13
DEGEN3	0.07	0.55	1.12	0.00	0.66	0.00	0.00	4.61	10.92
FIT1P	0.00	37.39	5.76	0.00	99.84	0.00	0.00	−9.92	−1.69
FIT2P	0.00	22.18	4.94	0.00	99.97	0.00	0.00	−6.88	−0.31
GREENBEA	28.79	24.74	22.84	0.70	2.05	9.40	0.42	15.79	30.84
GREENBEB	28.83	25.00	23.17	0.70	2.13	9.32	0.42	30.13	45.85
KEN-07	41.16	27.90	24.34	0.00	0.66	0.00	0.41	4.93	41.97
KEN-11	31.53	21.79	19.17	0.00	0.29	0.00	0.16	0.27	34.98
KEN-13	21.35	14.43	12.26	0.00	0.20	0.00	0.05	1.74	24.62
MAROS-R7	31.37	43.79	30.78	10.46	68.60	0.00	0.00	−3.73	10.43
MAROS	31.17	32.99	35.84	2.56	6.85	6.85	1.42	12.07	33.50
NESM	2.41	20.80	4.71	0.00	64.10	0.00	0.00	−25.16	−19.61
OSA-07	3.31	0.00	33.08	0.00	0.00	0.00	3.31	−1.90	12.15
OSA-14	1.58	0.00	32.77	0.00	0.00	0.00	1.58	2.83	7.77
OSA-30	0.85	0.00	32.43	0.00	0.00	0.00	0.85	−5.74	−4.30
PDS-02	12.90	4.82	3.99	0.00	0.00	1.12	0.14	1.14	13.41
PILOT	7.35	8.35	5.89	0.71	0.76	1.66	0.76	???	
PILOT87	3.84	5.98	3.79	0.49	0.44	0.79	0.25	num. diff.	
PILOT.JA	18.81	26.86	26.01	1.46	11.90	3.08	4.14	13.59	23.14
PILOTNOV	15.47	18.55	12.68	1.43	11.07	0.72	3.59	9.75	22.27
SCSD8	0.00	0.22	0.05	0.00	1.51	0.00	0.00	−11.59	−2.26
SCTAP3	4.86	25.00	8.38	0.00	41.86	0.00	4.86	−36.17	−36.05
SHIP08L	60.08	27.64	33.37	1.17	0.00	20.54	1.03	39.72	57.72
SHIP12L	59.64	23.59	32.13	1.42	0.00	12.67	0.43	32.96	52.06
SHIP12S	74.22	30.55	35.69	2.79	0.00	3.82	0.43	36.52	65.75
STOCFOR3	8.03	8.37	13.22	8.16	0.00	0.00	0.00	8.27	23.34
TRUSS	0.00	0.05	0.01	0.00	0.40	0.00	0.00	−5.85	−9.05
WOODW	70.70	38.14	47.70	0.05	13.83	35.12	0.00	45.53	66.81
Average:	20.32	16.76	17.66	1.34	13.61	3.31	1.00	5.72	19.36
Avg. dev.:	16.38	10.64	12.32	1.69	18.83	4.37	1.20	14.03	20.72

References

[1] Erling D. Andersen and Knud D. Andersen. Presolving in linear programming. Technical report, Odense University, August 1993.

[2] Robert E. Bixby. Implementing the Simplex method: The initial basis. *ORSA Journal on Computing*, 4(3):267–284, 1992.

[3] Robert E. Bixby. Progress in linear programming. *ORSA Journal on Computing*, 6(1):15–22, 1994.

[4] A. I. Brearley, G. Mitra, and H. P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, 8:54–83, 1975.

[5] Thomas S. Chang, Frank S. McCormic. A hierarchical algorithm for making sparse matrices sparser. *Mathematical Programming*, 56:1–30, 1992.

[6] CPLEX Optimization, Incline Village. *Using the CPLEX Callable Library and CPLEX Mixed Integer Library*, 1993.

[7] David M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 1985.

[8] Donald E. Goldfarb and John K Reid. A practicable steepest-edge simplex algorithm. *Mathematical Programming*, 12:361–371, 1977.

[9] Jacek Gondzio. Another simplex-type method for large scale linear programming. Technical report, Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland, 1994.

[10] Jacek Gondzio. Presolve analysis of linear programs prior to applying an interior point method. Technical Report 1994.3, Departament of Management Studies, University of Geneva, 102, Bd. Carl-Vogt, 1211 Geneva, Switzerland, 1994. (to appear in ORSA Journal on Computing).

[11] Jacek Gondzio and Marek Makowski. HOPDM, modular solver for LP problems; User's guide to version 2.12. Working Paper WP-95-50, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1995.

[12] IBM, New York. *Mathematical programming system: extended (MPSX) and generalized upper bounding (GUB) program description*, 1972.

[13] M Makowski. LP-DIT data interchange tool for linear programming problems (version 1.20). Working Paper WP-94-36, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[14] Thomas S. McCormic. Making sparse matrices sparser: Computational results. *Mathematical Programming*, 49:91–111, 1990.

[15] W. Ogryczak and K. Zorychta. Modular optimizer for mixed integer programing MOMIP version 2.1. Working Paper WP-94-35, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[16] John K. Reid. A sparsity-exploiting variant of the Bartels–Golub decomposition for linear programming bases. *Mathematical Programming*, 24:55–69, 1982.

[17] A. Świętanowski. A modern implementation of the revised simplex method for large scale linear programming. Master's thesis, Institute of Automatic Control, Warsaw University of Technology, Warsaw, 1993. (in Polish).

[18] A. Świętanowski. SIMPLEX v. 2.17: an implementation of the simplex algorithm for large scale linear problems. User's guide. Working Paper WP-94-37, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[19] A. Świętanowski. A penalty based simplex method for linear programming. Working Paper WP-95-005, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1995.

[20] Andrzej P. Wierzbicki. Augmented Simplex: a modified and parallel version of Simplex method based on multiple objective and subdifferential optimization approach. Working Paper WP-93-059, International Institute for Applied Systems Analysis, 1993.

A The software

It is now quite common for the suppliers of professional optimization packages (e.g. Cplex 3.0, see Bixby [3] or [6]) as well as advanced research codes (e.g., Gondzio [11]) to provide the customer with an option to perform presolve analysis prior to applying an optimization method. On the other hand it may be convenient for the user to be able to perform such analysis and then decide on the optimization software to use or to reformulate a problem. Therefore we have decided to provide both an embedded implementation and a stand alone presolve/postsolve package.

Thus the presolve procedure has become a new feature of our `simplex` optimizer (see Świętanowski [19] for description of the unique features of this implementation of the revised simplex method). At the same time two programs: `presolve` and `postsolv`[2] are provided, which enable the user to perform presolve analysis of a linear problem, solve it with any linear optimizer (provided it can produce the solution in one of the formats understood by the postsolver) and recover the solution of the original problem using the `postsolv`.

B Software availability

The optimization software package consisting of `simplex`, `presolve` and `postsolv` is made freely available for scientific non-commercial use in teaching and research institutions only. Researchers who want to obtain a binary executable version of some or all files of the package should contact Dr. Marek Makowski, *Methodology of Decision Analysis* project, IIASA, A-2361 Laxenburg, Austria. Then a license agreement form under the title "Request for Software" will be mailed back. When the completed form is received by Dr. Makowski, the software will be transferred by `ftp` or by other means together with documentation in form of IIASA working papers. You may also e-mail Dr. Makowski. The address is: `marek@iiasa.ac.at`.

The binary versions of the program are available for the following system platforms:

1. SparcStation running Solaris operating system; code compiled with GNU C++ ver. 2.6.3 or later, or SunPro C++ compiler ver. 3.0.1,

2. CRAY Superserver 6400 running Solaris operating system; code compiled with GNU C++ ver. 2.7.0,

3. IBM PC AT compatible with 386, 486 or Pentium processor; code compiled with GNU C++ ver. 2.6.0 or Watcom C++ compiler ver. 10.0; a 32-bit DOS executable with extender allowing to use the entire physical memory and, in case of the GNU compiler, also virtual memory.

If this should be necessary, the code may be recompiled for any other platform under a separate agreement with the code's author. Other platforms that we know our code to be compatible with are IBM RS 6000 with AIX and DEC workstations running Ultrix operating system.

This report and other IIASA working papers (including those referenced in this paper) are available via WWW at `http://www.iiasa.ac.at/docs/IIASA_Publications.html`.

In case of any problems with the software or questions regarding it's applicability in non-standard applications feel free to contact the author using the address given on the title page or e-mail address: `swietano@ia.pw.edu.pl`.

C User's guide to using the presolve analysis

In this section we shall present the calling syntax of all three applications. It was our intention to make their user interfaces as similar, consistent and easy to use as possible. Therefore we

[2]This is not a typing error. For compatibility with MS-DOS the names of the programs have been limited to eight characters, thus the 'e' at the end of the word 'postsolve' had to be omitted.

shall first give a general description, which applies to all the programs, then we shall proceed to define the options available in each of them. Finally, a number of typical usage examples will be given for each of the programs in the package.

C.1 Batch processing

All three applications (`simplex`, `presolve` and `postsolv`) work non-interactively, taking a number of input files and when required producing some output files. At runtime the programs may output a report to the standard output (normally the screen). When called without any arguments they print out a short reminder of the calling syntax. All the directives concerning the activity of a program have to be given as command line arguments.

C.2 The common characteristics

The common, most general calling syntax is:

```
<program_name> <options>
```

where `<program_name>` (one of `simplex`, `presolve` or `postsolv`) is followed by any number of options. Each option consists of a keyword (beginning with a dash '`-`') and an argument, e.g., `-mps_in afiro.mps`. The options may be given in any order, but may not be repeated (except when repetition is specifically allowed).

Whenever the option's argument refers to a file name, no extensions are assumed or added. In particular input and output file formats are not recognized by file name extensions.

Any error detected during argument parsing causes a runtime error message to appear on standard error device (usually the screen) and the program terminates. In particular, each of the programs requires that one or two input files always be given.

The programs input and output data in the following formats:

1. Fixed or free MPS text file for linear problem input and output.

   MPS format with mixed integer programming extensions is described in IBM's MPSX linear algebra package manuals [12].

   Warning: Mixed integer extensions are understood in input files, but are not present in the output files.

2. A text format for optimal basis output.

   It is defined in the user's guide to the previous release of the `simplex` optimizer (see Świętanowski [18]) and is accepted by the MOMIP mixed integer optimizer of Ogryczak and Zorychta [15].

3. A solution in text format identical to the format provided by the previous release of `simplex`.

   Since the postsolve procedure does not recover the values of the dual variables or reduced costs of the original problem, those values are not available in the solution whenever presolving is performed. Similarly, after presolving the row activities are unavailable.

4. A text file containing the log of all presolve actions.

   This file is only to be used for data interchange between `presolve` and `postsolv`. The format is not published and subject to change without notification.

5. A binary LP-DIT format for linear problem input and output (see Makowski [13]).

6. A binary LP-DIT format for problem solution input and output.

C.3 The options of the `presolver`

Program `presolve` accepts the following options:

- `-mps_in <file name>` or `-dit_in <file name>`

  This is the only mandatory option. It defines the name of the input file in appropriate format. Exactly one input file must be given. In case of an MPS file the keyword `-mps_in` may be omitted.

- `-mps_out <file name>` and `-dit_out <file name>`

  The reduced LP may be output in either MPS format, or LP-DIT format or both.

  There is no default output name.

- `-action <file_name>`

  Defines the name of a file to which the presolve actions will be written.

- `-mode x{+y{+z...}}`

  By default (i.e., when this option does not appear in the command line) the presolver will be invoked with all presolve techniques active.[3] The argument following the option must consist of one or more abbreviations of specific presolve methods (flags) separated only by plus ("`+`") characters (and not by whitespace). For example, to specify all presolve methods *including* the explicit slack conversion you need to write: `-mode all+es`. See Table 9 for a full list of available flags.

  This option may be repeated.

- `-v [none|low|high]`

  The amount and detail of the report that the optimizer produces when it solves a problem is decided by this option. The verbosity level `none` causes the program to operate silently; only the possible error messages are output to standard error.

  The default setting is `low`.

C.4 The options of the `postsolver`

The `postsolv` program reads the following options:

- `-dit_in <file name>`

  This (mandatory) option informs the program of the name of the linear problem solution in LP-DIT format.

- `-action <file_name>`

  The presolve actions are read from the given file. This option is also mandatory.

- `-txt_sol <file name>` or `-dit_sol <file name>`

  Solution output in either text or LP-DIT format or both may be requested. If the problem is not solved (e.g., when it is found to be infeasible) the solution will not be produced.

  There is no default solution file.

[3]Since removal of explicit slack variables is suspected to worsen simplex optimizer's performance, this presolve technique has to be invoked explicitly.

Table 9: Presolve flags

Flag	Meaning
`on`, `all`	all presolve techniques, except explicit slack removal
`off`, `none`	no presolving (has no effect when preceded or followed by other flags)
`sr`	singleton row removal
`fsc`	free and implied free singleton column elimination
`fdr`	forcing and dominated row elimination
`dc`	dominated and weakly dominated constraint detection
`ne`	numerical eliminations
`es`	explicit slack removal
`min`	empty row and column and fixed variable elimination (added by default to all other options except `off/none`)
`simple`	same as `min+sr`
`primal`	same as `min+fdr+fsc`
`dual`	same as `min+dc`

- `-mps_lp <file name>` or `-dit_lp <file name>`

  After presolving, the solution file (either in LP-DIT or MPS format) contains only the values of the primal variables. If, however, the user should wish to obtain the row activities as well, he or she must provide the *original* linear problem to the `postsolv`er using either one of the above options.

  The program reads in the original constraint matrix and multiplies it by the solution vector. The dimensions of the vector and the matrix must match. Additionally, `postsolv` checks whether the primal solution is feasible. Any infeasibility over $1.0E-8$ is reported but no other actions are taken (the row activities are calculated regardless of possible infeasibility).

- `-v [none|low|high]`

  Meaning identical to defined in appendix C.3.

C.5 Using the `presolver/postsolver` pair: examples

Presolver and postsolver are to be used together. Therefore the example here would present the whole processing cycle: from presolving through external optimizer to postsolving. In the example `simplex` will be used, but of course the reader would use an optimizer of his or her choice. For a description of the command line arguments of the simplex optimizer, see sections: C.6 and C.7.

```
presolve 1.mps -action act -dit_out 1.dit

simplex -presolve off -dit_in 1.dit -dit_sol sol

postsolv sol -action act -txt_sol 1.sol -mps_lp 1.mps
```

This example starts with presolving an MPS file `1.mps` to an LP-DIT file `1.dit` with presolve actions written to file `act`. Then the simplex optimizer comes in. It is explicitly told not do any presolving (`-presolve off`). It reads the presolved LP and writes an LP-DIT solution to `sol`. Finally, the postsolver is told to read in the presolved problem's solution `sol`, the presolve actions `act` and the original linear problem `1.mps`. From this it produces the solution to the original problem containing the primal variables and the row activities: a text file `1.sol`. If the option `-mps_lp 1.mps` was absent, the solution would only contain the primal variables.

C.6 The options of the `simplex` optimizer

The `simplex` optimizer accepts the following command line options:

- `-mps_in <file name>` or `-dit_in <file name>`

  Meaning identical to defined in appendix C.3. In particular, exactly one input file is required.

- `-txt_sol <file name>` and `-dit_sol <file name>`

  Meaning identical to defined in appendix C.4.

- `-basis <file name>`

  The optimal basis will be output to a file when this option is present. If the optimal solution is not found, the basis file contents will inform the MOMIP optimizer that the problem was infeasible. When basis output is requested, presolve analysis is disabled.

  By default the optimal basis is not stored.

- `-presolve x{+y{+z...}}`

  Meaning identical to `presolver`'s option `mode` (see appendix C.3).

- `-pric [rc|se|ase]`

  It is possible to specify the simplex pricing scheme. Available schemes are:

  - `rc` for minimum reduced cost selection (Dantzig's criterion),
  - `se` steepest edge (see Goldfarb and Reid [8] or Świętanowski [19]) and
  - `ase` our own efficient steepest edge approximation (see Świętanowski [19]), which is the default mode.

- `-scale [on|off]`

  Allows to turn on or off the linear problem scaling. By default: `on`.

- `-v [none|low|high]`

  Meaning identical to defined in appendix C.3.

C.7 Using the `simplex` optimizer: examples

We shall now provide a number of basic examples of using `simplex`. In those examples we shall assume that we have two linear problems: `1.mps` in MPS format and `2.dit` in LP-DIT format.

1. `simplex 1.mps -txt_sol 1.sol`

   This directs `simplex` to read `1.mps` (assuming it's in MPS format), solve it using all the default options (including default presolving) and store the solution in a text file `1.sol`.

2. `simplex -mps_in 1.mps -txt_sol 1.sol -v none`

   The difference between this and the previous example is such, that there will be no on-screen report. The keyword `-mps_in` does not change anything here.

3. `simplex -dit_in 2.dit -presolve min+ne -dit_sol 2.sol`

   This time the simplex optimizer will read an LP-DIT file (keyword `-dit_in` must be given), perform presolving consisting of numerical eliminations (`ne`), empty row and column reductions and fixed variable removal (`min`). The `min` presolve flag could be omitted. An LP-DIT solution will be written to file `2.sol`.

4. `simplex 1.mps -basis 1.inv`

   The `simplex` optimizer is used here to produce the optimal basis for MOMIP mixed integer optimizer. The problem is not presolved (because the `-basis` option was used).

C.8 Using the `smip` companion application

This application works on a slightly different principle than the other ones. While from user's perspective it appears to be a stand alone mixed integer optimizer, it is actually no more than a client application to a number of other stand alone programs. It calls `simplex` and MOMIP and possibly, some other external applications providing them with necessary input files and command line arguments. Each of the applications takes its input, provides some output data (e.g., for further processing), reports its activity and perhaps produces some error messages.

To make the process of running two or more applications at a time manageable and easy to understand, `smip` takes the burden of streamlining the applications off the user's back. Instead, it operates silently and writes a common activity log file and possibly also an error log. *No screen output is produced.* Because of the long time that integer optimization usually takes we decided that background operation will quite likely be the most useful one. On exit the error log (by default `smip.err` is removed if it is empty. The log file (by default `smip.log`) always remains.

At runtime `smip` creates in the current working directory a number of temporary files used for communication between the applications and for some housekeeping chores. Those temporary files are removed when the computations are finished.

The `smip` integrator accepts the following command line options:

- `-mps_in <file name>` or `-dit_in <file name>`

  Meaning identical to defined in appendix C.3. In particular, exactly one input file is required.

- `-txt_sol <file name>` and `-dit_sol <file name>`

  Meaning identical to defined in appendix C.4.

- `-log <file name>`

  By default `smip` produces a log file called `smip.log`. This name can be changed to any other file name by using the above option.

- `-log <file name>`

  By default `smip` produces an error log file `smip.err`. A different name may be given here. If the error log is empty upon completion of all computations it is not stored at all.

- `-v [none|low|high]`

  Similar to the meaning defined in appendix C.3, but this time (as `smip` writes it's reports to a file rather than to a stream) the verbosity level refers the file output.

C.9 Using the `smip` integrator: examples

The following two examples demonstrate the uses of `smip`:

1. `smip 1.mps -txt_sol 1.sol`

   This command line will cause `smip` to:

   a) convert the MPS input file `1.mps` to LP-DIT format,
   b) solve the LP-DIT problem using the `simplex` optimizer and produce an optimal basis for MOMIP,
   c) invoke MOMIP which will read in the problem in LP-DIT format, solve it starting from the optimal basis produced by `simplex` and write the text solution to file `1.sol`.

2. `smip -dit_in 1.dit -log 1.log -txt_sol 1.sol -dit_sol 1.dsl`

   This time `smip` takes an LP-DIT input file (thus no conversion is necessary), writes the log file in `1.log` and stores two solution files: one in text and one in LP-DIT format.