Preprocessing and Postprocessing in Linear Optimization

Preprocessing and Postprocessing in Linear Optimization

By

Xin Huang

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree
Master of Science

McMaster University

MASTER OF SCIENCE (2004) McMaster University

COMPUTING & SOFTWARE Hamilton, Ontario

TITLE: Preprocessing and Postprocessing in Linear Optimization

AUTHOR: Xin Huang, B.Sc. (Beijing Institute of Technology)

SUPERVISOR: Professor Tamás Terlaky

NUMBER OF PAGES: xvi, 146

Abstract

This thesis gives an overall survey of preprocessing and postprocessing techniques in linear optimization (LO) and its implementations in the software package McMaster Interior Point Method (McIPM).

We first review the basic concepts and theorems in LO. Then we present all the techniques used in preprocessing and the corresponding operations in postprocessing. Further, we discuss the implementation issues in our software development. Finally we test a series of problems from the Netlib test set and compare our results with state of the art software, such as LIPSOL and CPLEX.

Acknowledgements

I gratefully give my thanks to my supervisor, Dr. Tamás Terlaky, an expert in optimization area. It is him who lead me to the wonderful optimization world. I benefit a lot from his invaluable instruction. Without his guidance, discussion and encouragement, I would not finish this thesis. I also thank the members of the defense committee: Dr. Antoine Deza and Dr. Jiming Peng for their careful review and insightful comments on my work.

I appreciate all the kind help from the members of the advanced optimization laboratory, especially thank Xiaohang Zhu who provided great help in the process of my thesis writing and implementation.

Finally my special thanks goes to my husband Changhai Jia, my father Xiaorong Huang and my mother Xiuyun Zhao, for their endless love, encouragement and support.

Contents

List of Figures

List of Tables

Preface

Linear Optimization (LO) has been developed fast since the second world war and it is widely used in practice. When solving an LO problem with a software package, preprocessing and postprocessing plays an important role. In preprocessing, an LO problem is transformed into a standard form, the redundancies are removed and its size is reduced. The more important is that matrix A is made to have full rank so that the LO problem can be solved by Interior Point Methods (IPMs). Preprocessing is an indispensable part in implementations.

In this thesis, we present the fundamental theorems of LO and the techniques in preprocessing and postprocessing. Further, detailed implementation issues are described and the testing results and the comparisons are shown.

In Chapter 1, we first present the LO in standard form and review the basic concepts and duality theorems of LO. They are the basis for preprocessing techniques. The MPS format is also described in this chapter and to make the description more clear, an MPS file example is given as well.

In Chapter 2, we give an overall survey of preprocessing techniques. In Chapter 3, the corresponding operations of postprocessing are described as well. Preprocessing changes the LO problem somehow, thus we need postprocessing to recover the changes made to the LO problem.

Chapter 4 is devoted to implementation issues. In our implementation, we developed three subroutines: MPS reader, preprocessing and postprocessing. Detailed information about the implementation environment, the procedures in preprocessing and postprocessing, the design of data structure, data storage and data link between MATLAB and C are discussed.

Further, we present our testing results based on the Netlib testing set in Chapter 5. The comparison with LIPSOL and CPLEX are also presented in this chapter.

Finally in Chapter 6, we conclude the thesis and some suggestions for future work are given.

Chapter 1

Preliminaries

In this chapter we discuss various representations of linear optimization problems. Basic concepts and fundamental theorems of linear optimization are discussed as well.

1.1 Linear Optimization Problems

Linear Optimization (LO) is a discipline that develops methodologies to find the optimal (minimal or maximal) value of a linear objective function, subject to linear constraints on the variables.

1.1.1 The LO Forms

There are a variety of ways to represent LO problems. In the LO literature, to simplify the description of the theory and the algorithms, in the "standard form", there are only equality constraints and all the variables are nonnegative without upper bounds.

Formally, an LO problem in the standard form [17] is given as:

$$\begin{aligned} \min \quad & c_1x_1 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{i1}x_1 + \ldots + a_{in}x_n = b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \qquad\qquad j = 1, \ldots, n. \end{aligned} \tag{1.1.1}$$

There are n decision variables $x_1, \ldots, x_n$ subject to m constraints where a_{ij} is the coefficient of variable j in constraint i; b_i is the right-hand side (RHS) coefficient of constraint i; c_j is the cost coefficient of variable j in the objective function $c_1x_1 + \ldots + c_nx_n$. All the variables x_j, $j = 1, \ldots, n$, are restricted to be nonnegative.

Using matrix notation, system (1.1.1) can be written as:

$$\begin{aligned} \min \quad & c^Tx \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{1.1.2}$$

where $x = [x_1, x_2, \ldots, x_n]^T \in \mathbf{R}^n$, $c = [c_1, c_2, \ldots, c_n]^T \in \mathbf{R}^n$, $b = [b_1, b_2, \ldots, b_m]^T \in \mathbf{R}^m$, $A = (a_{ij}) \in \mathbf{R}^{m \times n}$ with $\text{rank}(A) = m$, $m \leq n$.

In most practical LO problems, the variables may have lower and upper bounds. If we put all variables' bounds into the constraints, the problem size will be doubled. Therefore, to be efficient, the standard form of LO problem usually includes lower and upper bounds for variables in LO implementation [3, 13]. Thus our standard LO problem with bounds on the variables can be presented in the following form:

$$\begin{aligned} \min \quad & c^Tx \\ \text{s.t.} \quad & Ax = b, \\ & l \leq x \leq u, \end{aligned} \tag{1.1.3}$$

where $l = [l_1, \ldots, l_n]^T \in \mathbf{R}^n$ and $u = [u_1, \ldots, u_n]^T \in \mathbf{R}^n$ are lower and upper bounds for the variables x, respectively. The value of l_j and u_j, $j = 1, \ldots, n$, can be $-\infty$ and $+\infty$, respectively.

Additionally, the general constraints may also have lower and upper bounds. The objective function may include an additive constant c_0 as well. The LO problem can be presented in a more general form [17]. For instance,

$$\begin{aligned} \min \quad & c_0 + c^T x \\ \text{s.t.} \quad & \underline{b} \leq Ax \leq \overline{b}, \\ & l \leq x \leq u, \end{aligned} \tag{1.1.4}$$

where $\underline{b} = [\underline{b}_1, \ldots, \underline{b}_m]^T \in \mathbf{R}^m$ is the lower bound vector for the constraints and $\overline{b} = [\overline{b}_1, \ldots, \overline{b}_m]^T \in \mathbf{R}^m$ is the upper bound vector for the constraints. The value of $\underline{b}_i$ and $\overline{b}_i$, $i = 1, \ldots, m$, can be $-\infty$ and $+\infty$, respectively.

An LO problem can be converted from the general form (1.1.4) into the standard form (1.1.2) (or vise versa) by carrying out appropriate transformations [23].

1.1.2 The MPS Format

When an LO model of a practical problem is built, for instance, in the form of (1.1.2) or (1.1.4), it needs to be presented in a standard readable form to computers. Such a representation form is called external representation.

The most widely used external representation is the Mathematical Programming System (MPS) format [17], an ASCII file with rigid format. The MPS format was first invented by IBM for use in its mathematical programming packages in the 1950s. Due to technology limitations at that time, the data was written on punch cards.

Therefore, the record length and the positions of the data on the card are strictly defined.

The rigid MPS format is still widely used today. It is considered to be the default industry standard, originally created by a single vendor with market dominance. This standard input form makes LO problem data portable from one LO software package to another. This standard representation of problem definition can be read by all commercial LO softwares packages. After read the data by some customizied MPS reader code, the LO problem is converted to a certain internal representation, that will be submitted to the solvers after a preprocessing procedure. Internal representations are not standard. It varies with different software packages and also depends on what algorithms are implemented in the solver.

The MPS format aims to express linear problems and mixed integer linear problems. Recently the MPS format was extended to allow the representation of convex quadratic problems too. To be able to cover most LO problems in general form, an MPS file stores LO problems in the form of (1.1.4). To reduce the amount of data, principally only the nonzero elements are given in an MPS file. However, in practice some zero elements may be included due to some modelling needs, e.g., when different scenarios are considered and data need to be changed in subsequent solutions.

The MPS File

The MPS data file contains 80 character long records and it is composed of seven sections: NAME section, COLUMNS section, RHS section, RANGES section, BOUNDS section and ENDATA section. These sections must display in this order. The RANGES section and the BOUNDS section are optional. The other sections appearance is mandatory.

In each section, there are two types of records: indicator records and data records.

Each section begins with an indicator record that is used to identify the section. The indicator record starts at column 1. It will be one of the following character strings:

```
NAME
ROWS
COLUMNS
RHS
RANGES
BOUNDS
ENDATA
```

Data records give the information about the data A, $\underline{b}$, $\overline{b}$, c, l and u as presented in (1.1.4). There are six fields with predefined positions in a data record. The following diagram shows the character positions in a data record.

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
2-3	5-12	15-22	25-36	40-47	50-61

The content of data records varies with the different sections. We discuss the seven sections one by one.

NAME section: The NAME section is the beginning of the MPS file that states the name of an LO problem. Any information before it can be ignored. There is no data record in this section. This section contains a single identifier record with the string NAME in columns 1-4 and the problem name (optional) in columns 15-22.

Columns 1-4	Columns 15-22
NAME	problem name

Note that in practice, it is allowed that the problem name can be any long up to the end of the record.

ROWS section The ROWS section describes all the rows information. It consists of an indicator record and several data records.

The Indicator Record:

It contains the section indicator ROWS in columns 1-4.

Columns 1-4
ROWS

The Data Records:

Data records in the ROWS section include information about the rows (constraints) of the LO problem: the row type and the row name. They are specified in Field 1 and Field 2, respectively.

Field 1	Field 2
row type	row name

The row name is used to identify a row by a maximum 8 character long name, instead of a numerical index "i". The row type, i.e., equal, less than or equal, greater than or equal, or not restricted, is expressed by a row type code as described in the following table:

Row type	Meaning	Row Type Code
$=$	Equality constraint	E
$\leq$	Less than or equal constraint	L
$\geq$	Greater than or equal constraint	G
Objective	Objective function	N
Free	No restriction	N

More than one N type rows may appear in the ROW section. The reason is that the MPS file was designed to be a flexible data file that enables the user to

express information about different model variants in one file. Objective functions vary in different model variants, thus they may become a constraint or irrelevant in another and they are subject to the same constraints, therefore all the objective functions are displayed in one file. This also allows to deactivate a constraint. For these reasons, multiple N type rows may appear. Users can specify the objective function by row name when a solver needs the MPS file and solves the LO problem. In software implementations, as default, the first N type row is usually considered to be the objective function. The other N type rows are not read and not passed to the solver.

COLUMNS section The COLUMNS section gives the information about matrix A and vector c in (1.1.4). It is composed of an indicator record and several data records too.

The Indicator Record:

It contains the section indicator COLUMNS in columns 1-7.

Columns 1-7
COLUMNS

The Data Records:

Data records specify the nonzero data of the matrix $\begin{pmatrix} c^T \\ A \end{pmatrix}$ in a column order representation. A data record includes the column name, the row name and the corresponding coefficient value.

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
blank	column name	row name	value	row name	value

For each column, the information of the column name, the row name and the coefficient is repeated for each element. It is optional to define two elements in one row, i.e., the use of Field 5 and Field 6 is optional. If used, the second

element given in Field 5 and Field 6, shares the same column name with the first element of that record.

Row index i and column index j are replaced by descriptive maximum 8 character long row and column names, respectively. All the nonzero elements a_{ij} and c_j are specified in Field 4 and Field 6, thus both the constraint matrix $A \in \mathcal{R}^{m \times n}$ and the objective function vector $c \in \mathcal{R}^n$ are stored and can be used from the COLUMN section. It is possible that some zero coefficients appear in this section due to some modelling purposes. Thus we can not assume that all given entries are nonzero. However we can remove those zero coefficients using some preprocessing techniques.

COLUMN section is column oriented. All the information related to a column must be given in one group, but within that group, data may be in any order. The order is not necessarily the same as the rows are given in the ROWS section.

RHS section: The RHS section specifies the information about the RHS data, i.e., $b \in \mathcal{R}^n$. It consists of an indicator record and several data records. The structural of the RHS section is similar to that of the BOUNDS section, because it specifies only a single column vector b, the right hand side data of the LO problem.

The Indicator Record:

It contains the section indicator RHS in columns 1-3.

Columns 1-3
RHS

The section indicator RHS must be displayed in the MPS format even if the right hand side vector b is a zero vector.

The Data Records:

These records include the RHS name, the row name and the RHS value.

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
blank	RHS name	row name	RHS value	row name	RHS value

It is optional that two different row names and the corresponding RHS values are written in the same line. In this case, Field 5 and Field 6 are used. It is allowed that multiple RHS vectors are specified with different names in a single RHS section. Since coefficient matrix A may be shared in different model variants with several objective functions, RHS, range or bound vectors, those sets of vectors may have different values based on the different vector names. Therefore, multiple RHS vectors, range vectors and bound vector may exist in one file. Users can choose the set of vectors or by default, the first vector is used automatically by an MPS reader code, if not specified otherwise.

RANGES section: The RANGES section is optional. It is desired when the constraints have both lower and upper bounds. It describes the range between the lower and upper bounds on the constraints. There is an indicator record and several data records in this section.

The Indicator Record:

It contains the section indicator RANGES in columns 1-6.

Columns 1-6
RANGES

The section indicator RANGES need not to be displayed if there is no range vector in the LO problem.

The Data Records:

Data records include the range name, the row name and the range value in an analogous format as data records in the RHS and COLUMN sections.

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
blank	range name	row name	value	range name	value

RANGES section actually gives the difference between the upper bound $\overline{b}_i$ and the lower bound $\underline{b}_i$ of constraint i.

$$\underline{b}_i \leq \sum_{j=1}^{n} a_{ij}x_j \leq \overline{b}_i,\ i = 1, \ldots, m.$$

Let b_i be the RHS value for row i given in the RHS section and r_i be the range value specified in the RANGE section. The actual lower bound $\underline{b_i}$ and upper bound $\overline{b_i}$ of RHS can be calculated according to the row type and the value of range. There are four cases as following:

Row Type	Sign of Range	RHS Lower Bound	RHS Upper Bound
G	$+$ / $-$	b_i	$b_i + \|r_i\|$
L	$+$ / $-$	$b_i - \|r_i\|$	b_i
E	$+$	b_i	$b_i + \|r_i\|$
E	$-$	$b_i - \|r_i\|$	b_i

As in the other sections, it is also optional to define two rows range values in one row. The second row name and range value are given in Field 5 and Field 6, respectively.

BOUNDS section: The BOUNDS section describes the variable's bound information. As the RANGE section, the BOUNDS section is also optional. There is an indicator record and several data records.

The Indicator Record:

It contains the section indicator BOUNDS in columns 1-6.

Columns 1-6
BOUNDS

The Data Records:

Data Records give the information about variable's bound type, bound name, column name and bound value.

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
bound type	bound name	column name	value	blank	blank

It is possible that more than one bound data record shares the same column name, e.g., when a variable has both upper and lower bounds. Different from the data records in the RHS and RANGE sections, Field 5 and Field 6 are not used in the BOUNDS section. More than one BOUNDS data set (with different bound name) can be present in the BOUNDS section. The bounds vectors may have different values based on the different bounds name. It is similar to the RHS and RANGES sections.

There are five bound types as the follows:

Bound Type	Meaning	Expression
LO/LB	lower bound	$l_j \leq x_j < +\infty$
UP/UB	upper Bound	$0 \leq x_j \leq u_j$
FX	fixed variable	$l_j = x_j = u_j$
FR	free variable	$-\infty < x_j < +\infty$
MI	unbounded below	$-\infty < x_j < 0$
PI	unbounded above	$0 \leq x_j < +\infty$

The default bounds for each variable x_i is $0 \leq x_i \leq +\infty$. If the variable is nonnegative, it needs not be defined.

ENDATA section: The ENDATA section marks the end of the MPS file. It only has a single indicator record:.

Columns 1-6
ENDATA

Comments can be included anywhere in MPS format. A comment is introduced by an * (asterisk) character. Any information starting with the asterisk is ignored.

As we mentioned, the BOUNDS and the RANGES sections are optional. That means that these two sections may not appear in an MPS file. To the contrary, the other sections are mandatory. When their content is empty, their corresponding section indicator records shall be presented in an MPS file. For instance, even if the

RHS vector b is a zero vector, then still its identifier record containing RHS must come right after the content of the COLUMNS section in the MPS file.

One peculiarity of the MPS format is that the direction of optimization: minimize or maximize is not indicated. Therefore, the solver must be advised if the objective is maximization or minimization. In software packages, minimization is the default option.

Integer Variables: As we mentioned, the MPS format is used to express linear problems and mixed integer linear problems. Integer variables are defined in the COLUMNS section too. There are two indicator records and several data records. The two indicator records are placed at the beginning and the ending of the integer variables' definition, respectively.

The Indicator Record:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
blank	marker name	'MARKER'	blank	keyword	blank

Field 1, Field 4 and Field 6 must be blank. Field 3 contains the string 'MARKER' (including the quotation marks). Field 2 contains the marker name that is different from other column names. At the beginning of an integer variable definition, Field 5 must contain the value 'INTORG' (including the quotation marks) to specify the beginning of the definition. At the end of the integer variable definition, Field 5 must contain the value 'INTEND' (including the quotation marks as well).

The Data Records:

Data records are the same as what we defined previously in the COLUMNS section.

Binary Variables: Binary variables may be defined as integer variables with their upper bounds equal to one. Additionally, binary variables can be specified in the BOUNDS section by using "BV" as bounds type. In the BOUNDS section, "BV" means binary variable, whose value equals to either 0 or 1.

QSECTION section: In the MPS file, we can also define a quadratic model. The objective function in a quadratic model is given as:

$$\min \quad c^T x + \frac{1}{2} x^T Q x,$$

where Q is a symmetric positive semidefinite matrix [20]. The QSECTION specifies the matrix Q, since Q is symmetric, only the upper triangle part of Q needs to be specified. The following three sections describe the quadratic part $x^T Q x$. They can be given in a separate MPS file. It is also possible that they are contained in the same file with the linear part we discussed before. In this case, the linear part is simply followed by the quadratic part. The three sections are:

```
NAME
QSECTION
ENDATA
```

The NAME and ENDATA sections are the same as what we defined previously. For the QSECTION section, there are an indicator record and several data records.

The Indicator Record:

It contains the section indicator QSECTION in columns 1-8.

Columns 1-8
QSECTION

The Data Records:

Data Records give two column names in Field 2 and Field 3, that actually specify the variables that form the quadratic term in the objective function. The column names in Field 2 and Field 3 need not be the same.

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
blank	column name	column name	value	column name	value

The following example may help to understand how to derive Q from the QSECTION section of an MPS file.

Example:

```
NAME
QSECTION
    x1        x1        4.0       x4        4.0
    x1        x2        0.0       x3        0.0
    x2        x1        0.0       x3        5.0
    x2        x2        5.0       x4        0.0
    x3        x3        5.0       x1        0.0
    x3        x4        0.0       x2        5.0
    x4        x1        4.0       x2        0.0
    x4        x4        4.0       x3        0.0
ENDATA
```

The QSECTION section specifies Q as:

$$Q = \begin{pmatrix} 4 & 0 & 0 & 4 \\ 0 & 5 & 5 & 0 \\ 0 & 5 & 5 & 0 \\ 4 & 0 & 0 & 4 \end{pmatrix},$$

with $x = [x_1, x_2, x_3, x_4]^T \in \mathbf{R}^4$. Then, we have

$$\frac{1}{2}x^T Q x = 2x_1^2 + \frac{5}{2}x_2^2 + \frac{5}{2}x_3^2 + 2x_4^2 + 4x_1x_4 + 5x_1x_3.$$

To close this section, we present a sample MPS file and the corresponding LO problem to facilitate the understanding of an MPS file.

Example:

```
NAME          sample problem
ROWS
 N  OBJ
 G  C1
 L  C2
 E  C3
COLUMNS
    V1        OBJ              4.5  C1               1.0
    V1        C3               2.5
    V2        OBJ              2.5  C2               1.5
    V2        C3               2.0
    V3        OBJ              4.0  C1               1.0
    V3        C2               0.5  C3               3.0
    V4        OBJ              4.0  C1               1.5
    V4        C2               0.5  C3               2.0
RHS
    RHS1      C1              40.0  C2              30.0
    RHS1      C3              95.0
RANGES
    RANGE1    C3              10.0
```

```
BOUNDS
 LO BND       V3                10.0
 UP BND       V3                20.0
 UP BND       V4                25.0
ENDATA
```

This MPS file specifies the following LO problem:

$$
\begin{aligned}
\min \quad & 4.5x_1 + 2.5x_2 + 4x_3 + 4x_4 \\
\text{s.t.} \quad & x_1 + x_3 + 1.5x_4 \geq 40, \\
& 20 \leq 1.5x_2 + 0.5x_3 + 0.5x_4 \leq 30, \\
& 2.5x_1 + 2x_2 + 3x_3 + 2x_4 = 95, \\
& x_1 \geq 0, \\
& x_2 \geq 0, \\
& 10 \leq x_3 \leq 20, \\
& 0 \leq x_4 \leq 25.
\end{aligned}
$$

Reading the MPS file precedes preprocessing that is followed by the solver and post-processing of the solution. The implementation of our preprocessing code will be discussed in Chapter 4, that is devoted to implementation issues.

1.2 Basic Concepts and Fundamental Theorems of LO

Every LO problem, called the primal problem, has another companion problem, the so-called dual problem. Duality [26] of LO problems plays an important role both in the theory and computational practice of LO. The famous duality theorem [26] characterizes the strong duality relationship.

1.2.1 The Primal Problem

Problem (1.1.2) is usually called the primal problem and denoted by (LP).

In the sequel, we list several basic concepts and notations related to (LP):

Set of primal feasible solutions: $\mathcal{F}_P = \{x \mid Ax = b,\ x \geq 0\}$.

Primal feasible solution: A vector $x \in \mathcal{F}_P$ is a *primal feasible solution.*

Primal optimal solution: A primal feasible solution x^* satisfies $c^T x^* \leq c^T x$ for all $x \in \mathcal{F}_P$, then x^* is *a primal optimal solution.*

Primal infeasible: If $\mathcal{F}_P = \emptyset$, then (LP) is *infeasible.*

Primal unbounded: If there is a sequence $\{x^k\}_{k=1}^{\infty}$ such that $x^k \in \mathcal{F}_P$ and $c^T x^k \to -\infty$ as $k \to +\infty$, then (LP) is *unbounded.*

For any LO problem, there are three possible cases. An LO problem

- is infeasible, or
- is feasible but unbounded, or

- has an optimal solution exists with final optimal objective value.

The set of primal feasible solutions defines an $n \times m$ dimensional polyhedron in an n dimensional space which is known as the feasible region. If the feasible region is not empty, then the LO problem is feasible. Otherwise, it is infeasible, i.e., there is no solution for the LO problem. When the LO problem is feasible, then we want to find a solution that can minimize the objective function when the objective is minimization. Such a solution is called the optimal solution. If such a point is found, then the LO problem is optimal. There is another case when the LO problem is feasible. If there are some solutions that make the objective value $-\infty$, then the LO problem is unbounded. Thus, from the analysis, we know that there is no other possibility except the three results.

1.2.2 The Dual Problem

Problem (LP) is associated with another LO problem which we call "dual". The dual problem of (LP) is denoted by (LD). Problem (LD) shares the same sets of data with (LP) and (LD) can be given in the following form:

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & A^T y \leq c, \\ & y \ \text{ is } \ \text{free}, \end{aligned} \tag{1.2.5}$$

where $y \in \mathcal{R}^m$ is the vector of dual variables.

By adding a nonnegative vector $s \in \mathcal{R}^n$ as slack variables to the inequality constraints, then inequalities are transformed into equality constraints. The dual

problem can be rewritten in the following form:

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & A^T y + s = c, \\ & y \text{ is free}, \ \ s \geq 0. \end{aligned} \tag{1.2.6}$$

Problem (LP) has m constraints and n variables. Problem (LD) has m dual variables y_i, $i = 1, \ldots, m$ and n dual slack variables s_j, $j = 1, \ldots, n$. A dual variable y_i is associated with constraint i in problem (LP) while a dual slack variable s_j is associated with a primal variable x_j. The roles of variables and constraints are reversed in (LP) and (LD).

Analogous to the primal problem, we give the basic terminology and notations for the dual problem.

Set of dual feasible solutions: $\mathcal{F}_D = \{(y, s) \mid A^T y + s = c, \ s \geq 0\}$.

Dual feasible solution: If there is a pair of vectors $(y, s) \in \mathcal{F}_D$, then (y, s) is a *dual feasible solution.*

Dual optimal solution: If there is a pair of vectors (y^*, s^*) such that $b^T y^* \geq b^T y$ for all $(y, s) \in \mathcal{F}_D$, then (y^*, s^*) is a dual optimal solution.

Dual Infeasible: If $\mathcal{F}_D = \emptyset$, then (LD) is *infeasible.*

Dual unbounded: If there is a sequence $\{y^k, s^k\}_{k=1}^{\infty} \in \mathcal{F}_D$ such that $(y^k, s^k) \in \mathcal{F}_D$ and $b^T y^k \to +\infty$ as $k \to +\infty$, then (LD) is *unbounded.*

1.2.3 Duality Theorems

Duality theorems are probably the most important theorems in LO. They provide optimality certificates and provide solid foundation for the investigation of the rela-

tionship between (LP) and (LD).

Theorem 1.2.1. *[Weak Duality [26]]*
Let $x \in \mathcal{F}_P$ and $(y,\ s) \in \mathcal{F}_D$. Then $c^T x \geq b^T y$, where equality holds if and only if $x^T(c - A^T y) = 0$ holds, or equivalently if

$$x_j(c - A^T y)_j = 0 \ \ holds\ for\ all\ j = 1,\ \ldots, n.$$

Proof. Since $Ax = b$, then $b^T y = (Ax)^T y = x^T A^T y$ and thus

$$c^T x - b^T y = x^T c - x^T A^T y = x^T(c - A^T y) = x^T s \geq 0. \qquad \square$$

The condition $x_j(c - A^T y)_j = x_j s_j = 0$, $j = 1,\ \ldots, n$ is called the complementarity condition. The value $c^T x - b^T y$ is called the duality gap for the solution $x \in \mathcal{F}_P$ of (LP) and $y \in \mathcal{F}_D$ of (LD).

There are several consequences of the weak duality theorem. The weak duality theorem shows that if both (LP) and (LD) admit feasible solutions, then both problems are bounded. The primal objective value of any primal feasible solution is an upper bound for the maximum objective value in (LD). On the other hand, the dual objective value of any dual feasible solution is a lower bound for the optimal objective value of (LP).

The following corollary is an obvious consequence of the Weak Duality Theorem [26].

Corollary 1.2.1. *(1) If (LP) is unbounded, then (LD) is infeasible.*
(2) If (LD) is unbounded, then (LP) is infeasible.
(3) If there is an $x^* \in \mathcal{F}_P$ and $y^* \in \mathcal{F}_D$ with $c^T x^* = b^T y^*$, then x^* is an optimal solution of (LP) and y^* is an optimal solution of (LD).

Proof. The proof of the first two statements is obvious. The proof of the third statement is as follows:

By the weak duality theorem, we have

$$c^T x \geq b^T y^* = c^T x^*,$$

and

$$b^T y \leq c^T x^* = b^T y^*.$$

Since $x^* \in \mathcal{F}_P$ and it provides the minimal objective value, it must be an optimal solution for (LP). Since $y^* \in \mathcal{F}_D$ and it provides the maximal objective value, it is an optimal solution for (LD).

□

Corollary 1.2.1 gives sufficient conditions for the infeasibility of (LP) or (LD) and for the optimality of a pair of primal and dual solutions. However, the reverse of the first two statements is not true and the reverse of the third statement is highly nontrivial.

It is possible that if either (LP) or (LD) is infeasible, then the other might be infeasible as well. Sufficiency is characterized by Farkas' lemma [22]. For further use we present both the primal and the dual forms of Farkas' Lemma.

Theorem 1.2.2 (Farkas Lemma [10], Primal Form). *Exactly one of the following two systems has a feasible solution:*
(I) $Ax = b,\ x \geq 0$;
(II) $A^T y \leq 0,\ b^T y > 0$.

Theorem 1.2.3 (Farkas Lemma, Dual Form). *Exactly one of the following two systems has a feasible solution:*

(I) $c^T x < 0,\ Ax = 0,\ x \geq 0;$
(II) $A^T y \leq c.$

We use the Farkas Lemma to prove the following result.

Corollary 1.2.2. *(1) If (LP) is infeasible, then (LD) is infeasible or unbounded. (2) If (LD) is infeasible, then (LP) is infeasible or unbounded.*

Proof. We proof the first statement. If (LP) is infeasible, then there is no $x \in \mathbf{R}^n$ such that

$$Ax = b,\ x \geq 0.$$

Then the primal form of the Farkas Lemma, Theorem 1.2.2 ensures that there exists a $y \in \mathbf{R}^m$ such that

$$A^T y \leq 0,\ \ b^T y > 0.$$

If (LD) is infeasible, then there is nothing to prove. If (LD) is feasible, then there exists a $\overline{y}$ such that

$$A^T \overline{y} \leq c.$$

For any $\alpha > 0$, we have

$$A^T(\alpha y + \overline{y}) \leq c,$$

that leads to

$$\lim_{\alpha \to +\infty} b^T(\alpha y + \overline{y}) = +\infty,$$

and thus (LD) is unbounded.

The proof of the second statement is analogous, where the dual form of the Farkas Lemma needs to be used. □

We have discussed the cases when either (LP) or (LD) is unbounded or infeasible. The remaining unsolved case is that what happens when both (LP) and (LD) have feasible solutions. We know that the optimal values are bounded. So far it is not clear if an optimal solution always exists and if both the primal and dual problems have optimal solutions, and then if the optimal objective values of the primal and dual problems are equal or not. The strong duality theorem answers the questions.

Theorem 1.2.4 (Strong Duality [22]). *Consider a pair of problems (LP) and (LD). If one of the problems has a finite optimal value, then so does the other, both problems admit an optimal solution, and the optimal objective values are equal, i.e., $c^T x^* = b^T y^*$, where x^* is a primal optimal solution and y^* is a dual optimal solution.*

The strong duality theorem provides solid foundation for designing algorithms to solve LO problems and it is especially useful to verify the optimality of candidate solutions for large-scale LO problems. The strong duality theorem guarantees that the optimal values of (LP) and (LD) are equal at optimality, and thus optimality check reduces to check primal and dual feasibility and to check whether the objective values of the two problems are equal.

In summary, there are four possible cases in LO.

(LP)	(LD)
optimal	optimal
feasible and unbounded	infeasible
infeasible	feasible and unbounded
infeasible	infeasible

If one of the two problems is unbounded, then the other is infeasible. If either of the two problems is infeasible, then the other is either infeasible or unbounded. If both have feasible solutions, then both have optimal solutions with zero duality gap.

In the next chapter, we give a comprehensive review of preprocessing techniques.

Chapter 2

Preprocessing

In spite of emerging technologies, the availability of faster and faster computers, and great improvement of LO algorithms, preprocessing is still an essential part of optimization software. In this chapter, we give a comprehensive review of preprocessing techniques for LO.

2.1 Introduction

Not only computer capacity increased significantly in the last two decades, but novel general purpose modelling systems enable users to model larger, more complex practical problems in more detail. This results in the need to solve very large scale LO problems on a regular base. These huge problems are usually not solvable in the form they were generated by general purpose modelling systems. Preprocessing is a procedure that transforms the LO problem into a properly formulated format while reduces problem size. It not only removes some redundancies but by simplifying problem formulation, it also improves the numerical characteristics of the problem. Therefore, preprocessing is an indispensable procedure in LO software implementation.

Preprocessing is not a new idea and it has been discussed by many authors. Although many techniques have been addressed, the principle is the same: use simple and fast techniques to detect various redundancies and use them repeatedly until the LO problem can not be reduced further in a short time. Usually, the time spent in preprocessing is only a small fraction of the total calculation time.

More extensive preprocessing does not necessarily imply faster overall solution time. Since some preprocessing techniques are quite time consuming, we have to balance the effect of the removed redundancies with the time spent in preprocessing. The ideal strategy is to choose only those techniques that reduce the total solution time. However, it is hard to find the optimal balance in practice. Therefore, a conservative strategy is used widely in most software packages. Only the fast and cheap techniques are usually chosen in most software implementations. Time consuming, sophisticated techniques are used only in high-end commercial products, frequently only as an optimal tool when solving very large-scale problems.

The ultimate aim of preprocessing is to speed up the solution time and improve computational efficiency. Generally, the roles of preprocessing are:

Transform the problem into a standard form. Before submitting the problem to the solver, the LO problem should be transformed into a "proper" form that the solver accepts. The "standard" form may be somewhat different among different LO solvers. Usually, the problem is transformed into the standard form:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & 0 \leq x \leq u, \end{aligned} \tag{2.1.1}$$

where matrix A has full row rank.

Detect infeasibility or unboundedness without solving the problem. If there are conflicts among the data in the original problem, problem (LP) may be detected infeasible even before submitting to the solver. Preprocessing may also detect that problem (LP) is unbounded by using the duality theorems or variants of the Farkas Lemma.

Reduce the problem size. Some data in the original problem may be redundant. Redundancy can be eliminated by removing fixed variables, redundant and forcing constraints. Moreover, some variables' bounds can be tightened that may result in conflicting bounds or fixed variables. The purpose is to reduce problem size and to improve computational efficiency.

Make A sparser. The sparsity of matrix $A \in \mathcal{R}^{m \times n}$ can reduce the complexity of the problem and make the computation faster and more reliable.

Improve numerical characteristics. Numerical instability will occur if there are some very small and/or large numbers in the problem. Scaling is used to bring those numbers to approximately the same magnitude that, in turn will improve numerical stability.

In this chapter, we give an overall survey of the preprocessing techniques. Here, we list and classify them briefly as follows:

1. Transform the LO problem into the standard form (2.1.1).

2. Basic logical analysis of the LO problem:

   (a) Infeasible variable.

   (b) Empty row.

   (c) Empty column.

   (d) Fixed variable.

   (e) Singleton row.

   (f) Singleton column.

   (g) Duplicate rows.

   (h) Duplicate columns.

3. Advanced logical analysis of the LO problem:

   (a) Redundant constraint.

   (b) Forcing constraint.

   (c) Tighten variable bounds.

   (d) Tighten dual variable bounds.

   (e) Dominated variables.

4. Make A sparser.

5. Make A to have full rank.

6. Scaling.

Preprocessing techniques are divided into several groups. The first group contains those basic transformations of the LO problem that convert the problem into a "proper" format that the solvers accept. Then two groups of logical analysis are presented. They are the essential and most widely used parts in preprocessing. The classification of "Basic" and "Advanced" logical analysis is based on the amount of calculation needed to perform that step. Much more calculations are needed for advanced analysis than for basic analysis. The more computation the techniques include, the more time they consume. Different users may choose which techniques they want to use according to their needs. Generally, in logical analysis, all the variables, the constraints, and the columns are scanned one by one. For instance, each variable is checked whether it indicates infeasibility or it is fixed based on its bounds. Some variable's bounds can even be tightened. Each row is checked whether it is empty, singleton or a duplicate of some other rows. Similarly, each column is checked whether it is empty, singleton or a duplicate of some other columns. Each constraint in both the primal problem and the dual problem is checked whether it is redundant or forcing. The operations can be summarized as elimination, substitution and removal of inherent redundancy [16]. Two more advanced techniques will be discussed as well. They are: make matrix A sparser and make A to have full rank. The aim of making matrix A sparser is to reduce the computational effort of the Cholesky factorization in Interior Point Methods (IPMs) and also to speed-up pivoting in simplex algorithms. Keeping A nonsingular is mandatory for IPM solvers. Actually some logical analysis can also help to remove the dependencies in A. We will describe them in detail. The last technique we discuss is scaling that aims to improve numerical stability.

We present these groups of preprocessing techniques one by one in the sequel.

2.2 Transform to Standard Form

One can read an LO problem from an MPS file (see Section 1.1.2) in the general form as given by (1.1.4):

$$\begin{aligned} \min \quad & c_0 + c^T x \\ \text{s.t.} \quad & \underline{b} \leq Ax \leq \overline{b}, \\ & l \leq x \leq u. \end{aligned}$$

Our aim is to transfer the problem in a form that can be used by any Interior Point Methods (IPMs) or simplex solver. LO problems in the general form (1.1.4) are transformed into the standard form (2.1.1):

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & 0 \leq x \leq u. \end{aligned}$$

The following transformations are required to bring (1.1.4) to (2.1.1):

1. Introduce a slack variable $s_i \geq 0$ to inequality constraint i:

$$\underline{b_i} \leq \sum_{i=1}^{n} a_{ij} x_j \leq \overline{b_i} \Leftrightarrow \begin{cases} \sum_{i=1}^{n} a_{ij} x_j + s_i = \overline{b_i}, \\ \sum_{i=1}^{n} a_{ij} x_j - s_i = \underline{b_i} \end{cases}$$

   with $0 \leq s_i \leq \overline{b_i} - \underline{b_i}$ if $\overline{b_i} < +\infty$ and $0 \leq s_i$ if $\overline{b_i} = +\infty$.

2. Split free variables.

   If variable x_j is a free variable, then it can be split into two nonnegative variables x_j^+ and x_j^-:

$$x_j = x_j^+ - x_j^-, \tag{2.2.2}$$

where $x_j^+ \geq 0$ and $x_j^- \geq 0$. Having the optimal value of x_j^+ and x_j^-, the value of x_j can be calculated by (2.2.2).

3. Shift the lower bound.

   If the lower bound vector l is nonzero, then a new variable vector $\hat{x}$ is introduced to shift the nonzero vector l to a zero vector:

$$\hat{x} = x - l. \tag{2.2.3}$$

   Then (2.1.1) becomes:

$$\begin{aligned} \min \quad & c^T\hat{x} + c^T l \\ \text{s.t.} \quad & A\hat{x} = b - Al, \\ & 0 \leq \hat{x} \leq u - l. \end{aligned} \tag{2.2.4}$$

   Having the optimal value of vector $\hat{x}$, the value of vector x can be recovered by (2.2.3).

2.3 Optimality Conditions

Optimality conditions provide the foundation of most preprocessing techniques. In Section 2.2, we have discussed how to transform an LO problem given in the general form (1.1.4) to the standard form (2.1.1). However, for ease of discussion, when describing the following preprocessing techniques, we allow l_j to be a nonzero vector. Thus, problem (LP) is given as:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & l \leq x \leq u, \end{aligned} \tag{2.3.5}$$

and its dual problem (LD) is:

$$\begin{aligned} \max \quad & b^T y + l^T v - u^T w \\ \text{s.t.} \quad & A^T y + v - w = c, \\ & y \ \text{ is } \ \text{free}, v \geq 0, w \geq 0. \end{aligned} \tag{2.3.6}$$

From the Strong Duality Theorem [26], if $x \in \mathcal{F}_P$ and $(y,\ v,\ w) \in \mathcal{F}_D$, then systems (2.3.5) and (2.3.6) have optimal solutions with their objective values equal, i.e.,

$$c^T x - (b^T y + l^T v - u^T w) = 0,$$

which can be written as

$$(A^T y + v - w)^T x - b^T y - l^T v + u^T w = 0.$$

Since $Ax = b$, we have

$$(x - l)^T v + (u - x)^T w = 0.$$

From $l \leq x \leq u$, $v \geq 0$ and $w \geq 0$, we get

$$\begin{aligned} (x - l)^T v = 0, \\ (u - x)^T w = 0, \end{aligned} \tag{2.3.7}$$

that is equivalent to:

$$\begin{aligned} (x_j - l_j) v_j = 0, \quad j = 1, \ldots, n, \\ (u_j - x_j) w_j = 0, \quad j = 1, \ldots, n. \end{aligned} \tag{2.3.8}$$

Further, from (2.3.8), we can see for the j-th constraint in the dual problem (2.3.6) that:

$$\begin{aligned} &\text{If } l_j = -\infty, \text{ then } v_j = 0 \text{ which leads to } \sum_{i=1}^{n} a_{ij}^T y_j \geq c_j. \\ &\text{If } u_j = +\infty, \text{ then } w_j = 0 \text{ which leads to } \sum_{i=1}^{n} a_{ij}^T y_j \leq c_j. \end{aligned} \tag{2.3.9}$$

These two conclusions are very important, they will play a crucial role in developing the preprocessing techniques about duplicate columns (see Section 2.4.8) and dominated columns (see Section 2.5.4).

In summary, the optimality conditions of the primal-dual systems (2.3.5) and (2.3.6) can be written as:

$$\begin{aligned}
Ax &= b, && (1)\\
l \le x &\le u, && (2)\\
A^T y + v - w &= c, && (3)\\
(x_j - l_j)v_j &= 0, \qquad j = 1, \ldots, n, && (4)\\
(u_j - x_j)w_j &= 0, \qquad j = 1, \ldots, n, && (5)\\
v \ge 0,\ w \ge 0,\ &y \text{ is free.} && (6)
\end{aligned} \tag{2.3.10}$$

Now we are ready to present and justify the techniques of preprocessing analysis.

2.4 Basic Logical Analysis of the LO Problem

We consider the primal LO problem in the form of (2.3.5) and its dual problem in the form of (2.3.6). In problem (2.3.5), slack variables have been added to the inequality constraints. However, to understand the preprocessing techniques in the following row related preprocessing analysis, we still consider the original row type of a constraint, that read directly from the MPS file or in other words, we distinguish slack variables from structural variables. Therefore, when we consider an inequality row, its slack variable is not included in the analysis. Moreover, if an inequality row can be removed from the LO problem, then its slack variable is removed as well.

Now, we present one by one the basic logical analysis techniques.

2.4.1 Infeasible Variable

Checking infeasibility of variables is the simplest technique in preprocessing. For each individual variable, if its lower bound is larger than its upper bound, then this constraint can not be satisfied. In this case, the variable is infeasible. It can be presented as:

variable x_j is infeasible if

$$l_j > u_j.$$

Obviously, the existence of an infeasible variable implies that problem (2.3.5) is infeasible. It is worth to detect such a possible contradiction for all the variables. The procedure of detection is easy. What we need to do is just to compare two bound vectors l and u.

2.4.2 Empty Row

If all the coefficients in a row are zero, then this row is an empty row. It can be presented as:

row i is an empty row if

$$a_{ij} = 0 \text{ for all } j = 1, \ldots, n.$$

An empty row may appear due to some zero entries in the original MPS file, as we discussed in Section 1.1.2. It also can be the result of some other reductions in the intermediate stages of preprocessing. Checking an empty row is necessary because it leads to the singularity of matrix A that is not acceptable by most IPM solvers.

For empty row i, the analysis is based on the row type of the original problem that was obtained directly from the MPS file. What is the implication of an empty row is decided by the sign of b_i. There are three cases:

1. If the row type is equal, i.e., $\sum_{j=1}^{n} a_{ij}x_j = b_i$, then:
   (a) If $b_i = 0$, then row i is redundant.
   (b) If $b_i \neq 0$, then problem (2.3.5) is infeasible.
2. If the row type is greater than or equal, i.e., $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$, then:
   (a) If $b_i > 0$, then problem (2.3.5) is infeasible.
   (b) If $b_i \leq 0$, then row i is redundant.
3. If the row type is less than or equal, i.e., $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, then:
   (a) If $b_i < 0$, then problem (2.3.5) is infeasible.
   (b) If $b_i \geq 0$, then row i is redundant.

We can see that an empty row can lead to the conclusion that either this empty row is redundant or problem (2.3.5) is infeasible.

If redundancy is detected, then this empty row can be removed from the problem. Additionally, in the case that row i is an inequality constraint, the slack variable of this row is removed as well. The removal of an empty row will reduce the LO problem size without any change in the solution of problem (2.3.5). The corresponding dual variable of this empty row can be set to any value, for instance, to zero.

2.4.3 Empty Column

If all the coefficients in a column are zero, then this column is an empty column. It can be presented as:

column j is an empty column if

$$a_{ij} = 0 \text{ for all } i = 1, \ldots, m.$$

Similar to an empty row, an empty column may exist in the original MPS file, or it may appear due to row removal.

For empty column j (variable x_j), there are three cases depending on the corresponding cost coefficient c_j and its bounds l_j and u_j. The analysis of an empty column j is as follows:

1. If $c_j = 0$:

   Because variable x_j does not appear in the objective function c and matrix A, it has no influence on the problem, thus column j can be removed from the LO problem. The solution of problem (2.3.6) is not affected by the removal of an empty column. The value of x_j can be set to any value, satisfying $l_j \leq x_j \leq u_j$.

2. If $c_j > 0$:

   (a) If $l_j = -\infty$, then $c_j l_j$ may go to $-\infty$. Obviously, either problem (2.3.5) is unbounded or dual infeasible.

   (b) If $l_j \neq -\infty$, then $c_j l_j \leq c_j x_j$ for $\forall x_j \in [l_j, u_j]$. Column j can be removed from the LO problem and x_j is set to the value of its lower bound l_j.

3. If $c_j < 0$:

(a) If $u_j = +\infty$, then $c_j u_j$ may go to $-\infty$. Obviously, either problem (2.3.5) is unbounded or dual infeasible.

(b) If $u_j \neq +\infty$, then $c_j u_j \leq c_j x_j$ for $\forall x_j \in [l_j, u_j]$. Column j can be removed from the LO problem and x_j is set to the value of its upper bound u_j.

Therefore, we know that there are two possible consequences of an empty column: either variable x_j is fixed to one of its bounds, l_j or u_j, or problem (2.3.5) is detected to be unbounded or dual infeasible. In the first case, x_j can be substituted out of the problem. Its value is stored in some array such that it can be retrieved in postprocessing. The size of the LO problem is reduced. Moreover, the objective value will be changed by an additive constant c_0 where $c_0 := c_0 + c_j x_j$ when $c_j \neq 0$.

2.4.4 Fixed Variable

Variable x_j can be fixed if there exists a j such that

$$l_j = u_j.$$

Fixed variable x_j may be defined directly in the BOUNDS section in an MPS file with bounds type definition "FX". It can also be detected at some intermediate stages of preprocessing. For instance, in the technique of detecting an empty column as described in Section 2.4.3, the value of a variable can be fixed to one of its bounds, l_j or u_j. In subsequent discussions on preprocessing, fixed variables can be produced by other techniques as well.

The treatment of a fixed variable is simple. As we referred in Section 2.4.3, if x_j is a fixed variable, then it can be substituted out of the problem and the problem size gets smaller. Before removed, the value of variable x_j must be stored in some

array so that it can be retrieved in postprocessing. Further, an additive constant in the objective value c_0 needs to be updated as

$$c_0 := c_o + c_j x_j,$$

and the corresponding RHS value needs to be updated as:

$$b_i = b_i - a_{ij} x_j,$$

where i is the row index where variable x_j occurs in matrix A.

2.4.5 Singleton Row

Row i is a singleton row if there is only one nonzero coefficient in this row, i.e., there exists a k such that

$$a_{ij} = 0 \text{ for } j = 1, \ldots, k-1, k+1, \ldots, n \text{ and } a_{ik} \neq 0.$$

A singleton row may appear directly in the MPS file. It may also occur when some columns are removed from the LO problem. If some columns are eliminated and only one nonzero coefficient is left in a row, then a new singleton row is created.

There are three cases, based on the row type of row i, to consider:

1. If the row type is equal, i.e., $\sum_{j=1}^{n} a_{ik} x_k = b_i$, then:
   (a) If $l_k \leq \frac{b_i}{a_{ik}} \leq u_k$, then row i is removed and x_k is set to $\frac{b_i}{a_{ik}}$ and variable x_k becomes a fixed variable (see Section 2.4.4).
   (b) If $\frac{b_i}{a_{ik}} < l_k$ or $\frac{b_i}{a_{ik}} > u_k$, then problem (2.3.5) is infeasible.
2. If the row type is greater than or equal to, i.e., $\sum_{j=1}^{n} a_{ik} x_k \geq b_i$, then:

(a) If $a_{ik} > 0$, then an implied lower bound is obtained as $x_k \geq \hat{l}_k = \dfrac{b_i}{a_{ik}}$.

(b) If $a_{ik} < 0$, then an implied upper bound is obtained as $x_k \leq \hat{u}_k = \dfrac{b_i}{a_{ik}}$.

3. If the row type is less than or equal to, i.e., $\sum_{j=1}^{n} a_{ik}x_k \leq b_i$, then:

(a) If $a_{ik} > 0$, then an implied upper bound is obtained as $x_k \leq \hat{u}_k = \dfrac{b_i}{a_{ik}}$.

(b) If $a_{ik} < 0$, then an implied lower bound is obtained as $x_k \geq \hat{l}_k = \dfrac{b_i}{a_{ik}}$.

If the singleton row is an inequality, then a new implied bound, $\hat{l}_k$ or $\hat{u}_k$ is obtained. The new implied bound needs to be compared with the original bounds, l_k or u_k. If they conflict, i.e., $\hat{l}_k > u_k$ or $\hat{u}_k < l_k$, then the LO problem (2.3.5) is infeasible. If the new implied bound is tighter than the original bound, i.e., $\hat{l}_k > l_k$ or $\hat{u}_k < u_k$, then the corresponding bound needs to be tightened as:

$$l_k \;=\; \max\,(l_k, \hat{l}_k) \quad \text{or} \quad u_k \;=\; \min\,(u_k, \hat{u}_k). \tag{2.4.11}$$

If the tightened new bounds, l_k and u_k, are equal, then variable x_k is treated again as a fixed variable.

If the implied bound is looser than the original one, i.e., $\hat{l}_k < l_k$ or $\hat{u}_k > u_k$, then row i is redundant and it can be removed. The corresponding dual variable y_i is set to zero. If the redundant row i is an inequality, the slack variable in row i is removed as well.

Therefore, summarizing our findings, we may say that there are three possible results of a singleton row: the variable bound is tightened, the variable is fixed or problem (LP) (2.3.5) is infeasible.

One singleton row elimination often creates new singleton row eliminations [13], thus checking singleton rows is an easy and efficient way to reduce the problem size.

2.4.6 Singleton Column

Column j is a singleton column if there is only one nonzero element in the column, i.e., there exist a k such that

$$a_{ij} = 0 \text{ for } i = 1, \ldots, k-1, k+1, \ldots, m \text{ and } a_{kj} \neq 0.$$

Variable x_j can be substituted out of the problem under the following two circumstances.

1. When variable x_j is a free variable, it can be replaced by the other variables in row k:

$$x_j = \frac{b_k - \sum_{p=1, p\neq j}^{n} a_{kp} x_p}{a_{kj}}. \tag{2.4.12}$$

   Constraint k becomes a free constraint, thus row k can be removed.

   The value of other variables x_p, $p = 1, \ldots, j-1, j+1, \ldots, n$, can be obtained from the solvers. Then the value of x_j can be calculated from (2.4.12).

2. When variable x_j is not a free variable, if its tightest computed bounds $\hat{l_j}$ or $\hat{u_j}$, derived from the lower and upper bounds of the constraints where variable x_j appears, is tighter than its original bounds l_j and u_j, i.e., $\hat{l_j} \geq l_j$ and $\hat{u_j} \leq u_j$, then variable x_j is called an implied free variable. The computed bounds of variable x_j, also called the implied bounds of variable x_j, can be calculated using the technique of "tighten bounds", that will be presented in detail in Section 2.5.2.

   x_j is an implied free variable if

$$[\hat{l}_j, \hat{u}_j] \subseteq [l_j, u_j].$$

If variable x_j is an implied free variable, then it can be treated as a free variable as outlined in case 1.

When a free (or an implied free) variable x_j is detected in a singleton column j where a_{kj} is the only nonzero coefficient in column j, variable x_j and constraint k can both be eliminated. An advantage of this is that matrix A does not change except row k is removed. The only change is in the cost coefficients c_p of all the other variables x_p, $p = 1, \ldots, j-1, j+1, \ldots, n$ that have nonzero coefficients in constraint k. The cost coefficient c_p is changed to:

$$c_p - \frac{a_{kp} c_j}{a_{kj}}.$$

The objective value will also be changed by a constant $c_0 := c_0 + \dfrac{b_k c_j}{a_{kj}}$.

Note that the removed row k specifies the relationship (2.4.12) between the removed variable x_j and the other variables x_p, $i = 1, \ldots, j-1, j+1, \ldots, n$, therefore the related information, for instance, all the nonzero coefficients in row k and the right hand side value b_k must be recorded. This information will be used to recover the value of variable x_j in postprocessing.

If x_j is a free (or an implied free) variable, then the corresponding value of y_k can be fixed to $\dfrac{c_j}{a_{kj}}$, because free variables imply equality constraints in the dual, and thus constraint j in the dual is a singleton equality row.

2.4.7 Duplicate Rows

Two rows are called duplicate rows if their corresponding coefficients are in proportion, i.e., there exist $i \neq k$ and $1 \leq i, k \leq m$, such that

$$a_{ij} = \lambda a_{kj} \text{ for all } j = 1, \ldots, n,\ \lambda \neq 0.$$

Most duplicate rows appear directly in the MPS file. It is necessary to check the duplicate rows and remove them, especially when the original row types are both equality, that imply that matrix A is singular.

There are four cases depending on the row types. The analysis is as follows:

1. If both row i and row k are equalities, then the two constraints are:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \tag{2.4.13}$$

$$\sum_{j=1}^{n} a_{kj} x_j = b_k. \tag{2.4.14}$$

   Let us multiply equation (2.4.14) by λ and substract it from equation (2.4.13). We have

$$\sum_{j=1}^{n} a_{ij} x_j - \lambda \sum_{j=1}^{n} a_{kj} x_j = b_i - \lambda b_k.$$

   The left-hand side of this equation is a zero vector, thus this constraint is an empty row (see Section 2.4.2). We list the possible results briefly as follows:

   (a) If $b_i = \lambda b_k$, then either row i or row k can be removed.

   (b) If $b_i \neq \lambda b_k$, then problem (2.3.5) is infeasible.

2. If both row i and row k are inequalities with the same row type, e.g., both are greater than or equal constraints:

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \tag{2.4.15}$$

$$\sum_{j=1}^{n} a_{kj}x_j \geq b_k. \tag{2.4.16}$$

For ease of discussion, assume that the ranges of row i and k are r_i and r_k, respectively. Note that r_i and r_k can be infinite. We have

$$b_i + r_i \geq \sum_{j=1}^{n} a_{ij}x_j \geq b_i, \tag{2.4.17}$$

$$b_k + r_k \geq \sum_{j=1}^{n} a_{kj}x_j \geq b_k. \tag{2.4.18}$$

(a) If $\lambda > 0$, then from (2.4.18), we have:

$$\lambda(b_k + r_k) \geq \lambda \sum_{j=1}^{n} a_{kj}x_j = \sum_{j=1}^{n} a_{ij}x_j \geq \lambda b_k. \tag{2.4.19}$$

There are two cases to be considered:

i. If $\lambda(b_k + r_k) < b_i$ or $b_i + r_i < \lambda b_k$, then the two constraints conflict, that imply that problem (2.3.5) is infeasible.

ii. If $\lambda(b_k + r_k) \geq b_i$ and $b_i + r_i \geq \lambda b_k$, then the two constraints can be reduced to one constraint:

$$\min(\lambda(b_k + r_k), b_i + r_i) \geq \sum_{j=1}^{n} a_{ij}x_j \geq \max(\lambda b_k, b_i). \tag{2.4.20}$$

We keep the constraint that has a tighter right-hand side $\hat{b}$, $\hat{b} = \max(\lambda b_k, b_i)$. Its corresponding range is updated as $|\min(\lambda(b_k + r_k), b_i + r_i) - \hat{b}|$. The other constraint is redundant and it can be removed.

(b) If $\lambda < 0$, then from (2.4.18), we have:

$$\lambda b_k \geq \lambda \sum_{j=1}^{n} a_{kj} x_j = \sum_{j=1}^{n} a_{ij} x_j \geq \lambda(b_k + r_k). \tag{2.4.21}$$

There are two cases to be considered:

i. If $\lambda b_k < b_i$ or $b_i + r_i < \lambda(b_k + r_k)$, then the two constraints conflict, that imply that problem (2.3.5) is infeasible.

ii. If $\lambda b_k \geq b_i$ and $b_i + r_i \geq \lambda(b_k + r_k)$, then the two constraints can be reduced to one constraint:

$$\min(\lambda b_k, b_i + r_i) \geq \sum_{j=1}^{n} a_{ij} x_j \geq \max(\lambda(b_k + r_k), b_i). \tag{2.4.22}$$

If b_i is tighter than $\lambda(b_k + r_k)$, i.e., $b_i \geq \lambda(b_k + r_k)$, then the constraint $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ is kept. Its range is updated as $|\min(b_i + r_i, \lambda b_i) - b_i|$.
If λb_k is tighter than $b_i + r_i$, i.e., $\lambda b_k \leq b_i + r_i$, then the constraint $\lambda b_k \geq \sum_{j=1}^{n} a_{ij} x_j$ is kept. Its range is updated as $|\max(\lambda(b_k + r_k), b_i) - \lambda b_k|$.

The case when both constraints have less than or equal row type can be treated analogously.

3. If both of row i and row k are inequalities with different row types, then we have:

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i,$$

$$\sum_{j=1}^{n} a_{kj} x_j \leq b_k.$$

The sign of λ may change the row type. Multiplying by λ, the row types of the two constraints may be the same or different. The analysis goes the same way as case 2 that has been discussed.

4. If one of the row types is equality while the other one is inequality, e.g.,

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,$$

$$\sum_{j=1}^{n} a_{kj}x_j \leq b_k,$$

then the analysis is similar to case 2 as well.

Detecting duplicate rows may result in either one row removal. or detecting that problem (2.3.5) is infeasible. When both of the two rows are equalities, then detecting duplicate rows is mandatory because they imply that matrix A is singular, that can be disastrous for IPM solvers.

2.4.8 Duplicate Columns

Two columns are called duplicate columns if their corresponding coefficients are in proportion, i.e., there exists a pair $j \neq k$ and $1 \leq j, k \leq n$ such that

$$a_{ij} = \lambda a_{ik} \text{ for all } i = 1, \ldots, m, \ \lambda \neq 0.$$

Using this relation, we may write

$$a_{ij}x_j + a_{ik}x_k = a_{ik}(\lambda x_j + x_k), \tag{2.4.23}$$

and

$$c_j x_j + c_k x_k = c_k(\lambda x_j + x_k) + (c_j - \lambda c_k)x_j. \tag{2.4.24}$$

Our aim is to remove variables x_j and x_k, while introducing a new variable $\hat{x}_k = \lambda x_j + x_k$. We have three cases to consider depending on the value of $c_j - \lambda c_k$.

1. If $c_j - \lambda c_k = 0$, then we can introduce the new variable $\hat{x}_k = \lambda x_j + x_k$. The bounds $\hat{l}_k$ and $\hat{u}_k$ of $\hat{x}_k$ can be derived from the bounds of x_j and x_k:

   - If $\lambda > 0$, then $l_k + \lambda l_j \leq \hat{x}_k \leq u_k + \lambda u_j$, i.e., $\hat{l}_k = l_k + \lambda l_j$ and $\hat{u}_k = u_k + \lambda u_j$.
   - If $\lambda < 0$, then $l_k + \lambda u_j \leq \hat{x}_k \leq u_k + \lambda l_j$, i.e., $\hat{l}_k = l_k + \lambda u_j$ and $\hat{u}_k = u_k + \lambda l_j$.

   In this case, variables x_j and x_k are removed from the problem and they will not appear in the matrix A and cost vector c. The number of variables in (2.3.5) is reduced by one. Further, we compare the bounds of $\hat{x}_k$. If they are in conflict, then problem (2.3.5) is infeasible. If the bounds of variable $\hat{x}_k$ are equal, i.e., $\hat{l}_k = \hat{u}_k$, then $\hat{x}_k$ is a fixed variable and can be removed from the problem as discussed in Section 2.4.4. The value of λ and the bounds of the two variables need to be recorded in order to recover the values of x_j and x_k in postprocessing. In the recovery step, the values of x_j and x_k are obtained by:

   $$\begin{aligned} \lambda x_j + x_k &= \hat{x}_k, \\ l_j \leq x_j &\leq u_j, \\ l_k \leq x_k &\leq u_k, \end{aligned} \tag{2.4.25}$$

   where $\hat{x}_k$ is already known in that stage.

2. If $c_j - \lambda c_k > 0$, then we consider the dual problem (2.3.6). Constraint j can be written as $\sum_{i=1}^{m} a_{ij} y_i + v_j - w_j = c_j$,
   that imply

   $$v_j - w_j = c_j - \sum_{i=1}^{m} a_{ij} y_i > \lambda c_k - \lambda \sum_{i=1}^{m} a_{ik} y_i = \lambda (v_k - w_k),$$

   thus we get

   $$v_j - w_j > \lambda (v_k - w_k).$$

Two special cases are of particular interest:

(a) $u_k = +\infty$ and $\lambda > 0$:

From the optimality condition (see Equations (2.3.10)), when $u_k = +\infty$, we have $w_k = 0$, and

$$v_j - w_j > \lambda v_k \geq 0,$$

implies:

$$v_j - w_j > 0.$$

Because $w_j \geq 0$, we have $v_j > 0$.

(b) If $l_k = -\infty$ and $\lambda < 0$:

Similarly, when $l_k = -\infty$, we have $v_k = 0$ and

$$v_j - w_j > -\lambda v_k \geq 0.$$

Because $w_j \geq 0$, we have $v_j > 0$.

Both cases lead to $v_j > 0$. When $v_j > 0$, then from $(x_j - l_j)v_j = 0$ in the optimality condition (2.3.10), we derive

$$x_j - l_j = 0.$$

The result is decided by the value of l_j. Therefore, because $c_j - \lambda c_k > 0$:

(a) If l_j is finite, then variable x_j can be fixed to l_j.

(b) If l_j is infinite, i.e., $l_j = -\infty$, then we should have $v_j = 0$. This conflicts with $v_j > 0$, thus the dual problem (2.3.6) is infeasible.

3. If $c_j - \lambda c_k < 0$, then an analogous analysis to case 2 can be given. The result is this case decided by the value of u_j:

(a) If u_j is finite, then variable x_j can be fixed to u_j.

(b) If u_j is infinite, i.e., $u_j = +\infty$, then the dual problem (2.3.6) is infeasible.

2.5 Advanced Logical Analysis of the LO Problem

So far, we have discussed several basic logical techniques. Usually, those techniques do not require much calculation except the case when singleton columns are analyzed. In this section, we consider advanced logical analysis techniques that require significantly more calculation.

2.5.1 Redundant Constraint and Forcing Constraint

Those two techniques may reduce the dimension of the LO problem. To find a redundant constraint or a forcing constraint, we need to calculate the lower and upper bounds of the given constraint.

For constraint i, to compute its lower bound $\underline{b_i}$ and upper bound $\overline{b_i}$, we define the index sets of the positive and negative coefficients in row i as follows:

$$\mathcal{P}_i = \{j,\ a_{ij} > 0\},$$
$$\mathcal{N}_i = \{j,\ a_{ij} < 0\}.$$

Since we know the bounds of each individual variable,

$$l_j \leq x_j \leq u_j,\ j = 1, \ldots, n,$$

we can calculate the lower bound $\underline{b_i}$ and the upper bound $\overline{b_i}$ of constraint i as follows:

$$\underline{b_i} = \sum_{j \in \mathcal{P}_i} a_{ij} l_j \ + \ \sum_{j \in \mathcal{N}_i} a_{ij} u_j,$$

$$\overline{b_i} = \sum_{j \in \mathcal{P}_i} a_{ij} u_j \ + \ \sum_{j \in \mathcal{N}_i} a_{ij} l_j.$$

Thus, for constraint i, we have

$$\underline{b_i} \leq \sum_{j=1}^{n} a_{ij}x_j \leq \overline{b_i}.$$

Definition 2.5.1. *An equality constraint i is called a forcing constraint if either its upper bound $\overline{b_i}$ or lower bound $\underline{b_i}$ is equal to its right-hand side value b_i, i.e.,*

$$\overline{b_i} = b_i \text{ or } \underline{b_i} = b_i.$$

Moreover, a less than or equal to constraint i is called a forcing constraint if its right-hand side value b_i is equal to $\underline{b_i}$, i.e., $b_i = \underline{b_i}$; A greater than or equal to constraint i is called a forcing constraint if its right-hand side value b_i is equal to $\overline{b_i}$, i.e., $b_i = \overline{b_i}$.

Definition 2.5.2. *Constraint i is called a redundant constraint in the following two cases:*

$$b_i > \overline{b_i} \text{ for } \sum_{j=1}^{n} a_{ij}x_j \leq b_i,$$

or

$$b_i < \underline{b_i} \text{ for } \sum_{j=1}^{n} a_{ij}x_j \geq b_i.$$

We can examine the relationship among b_i, $\underline{b_i}$ and $\overline{b_i}$ to find out redundant constraints and forcing constraints. There are three cases depending on the row type. For constraint i:

1. If the row type is less than or equal to, i.e., $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$:
   (a) If $b_i < \underline{b_i}$, then problem (2.3.5) is infeasible.
   (b) If $b_i = \underline{b_i}$, then constraint i is forced to its lower bound. Variable x_j is fixed at its lower bound l_j for $j \in \mathcal{P}_i$, and x_j is fixed at its upper bound u_j for $j \in \mathcal{N}_i$.

(c) If $b_i > \overline{b_i}$, then constraint i is redundant, thus row i can be removed.

(d) if $\underline{b_i} < b_i \leq \overline{b_i}$, then constraint i can not be removed. However, we may tighten the constraints range and have a chance to tighten the bounds of the participating variables. (see Section 2.5.2). Note that if the original range of row i is r_i, we can update the range $\hat{r}_i$ as $\min(b_i - \underline{b_i}, r_i)$.

2. If the row type is greater than or equal to, i.e., $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$:

(a) If $b_i > \overline{b_i}$, then problem (2.3.5) is infeasible.

(b) If $b_i = \overline{b_i}$, then constraint i is forced to its upper bound. Variable x_j is fixed at its upper bound u_j for $j \in \mathcal{P}_i$, and x_j is fixed at its lower bound l_j for $j \in \mathcal{N}_i$.

(c) If $b_i < \underline{b_i}$, then constraint i is redundant, thus row i can be removed.

(d) If $\underline{b_i} \leq b_i < \overline{b_i}$, then constraint i can not be removed. However, we may have a chance to tighten the bounds of the participating variables (see Section 2.5.2). Note that if the original range of row i is r_i, we can update the range $\hat{r}_i$ as $\min(\overline{b_i} - b_i, r_i)$.

3. If the row type is equal, i.e., $\sum_{j=1}^{n} a_{ij}x_j = b_i$:

(a) If $b_i = \overline{b_i}$, then constraint i is forced to its upper bound. Variable x_j is fixed at its upper bound u_j for $j \in \mathcal{P}_i$, and x_j is fixed at its lower bound l_j for $j \in \mathcal{N}_i$.

(b) If $b_i = \underline{b_i}$, then constraint i is forced to its lower bound. Variable x_j is fixed at its lower bound l_j for $j \in \mathcal{P}_i$, and x_j is fixed at its upper bound u_j for $j \in \mathcal{N}_i$.

(c) If $b_i > \overline{b_i}$ or $b_i < \underline{b_i}$, then problem (2.3.5) is infeasible.

(d) If $\underline{b_i} < b_i < \overline{b_i}$, then constraint i can not be removed. However, we may have a chance to tighten the bounds of the participating variables (see Section 2.5.2).

When constraint i is found to be a forcing constraint, the participating variables in this constraint can be fixed either to their upper bounds or to their lower bounds. Therefore, it is highly advantageous to find forcing constraints because both the forcing constraint and all the participating variables in that constraint can be removed. In this way the LO problem may be simplified significantly.

When constraint i is found to be a redundant constraint, it can be removed from the LO problem. The corresponding dual variable y_i is set to zero.

Note that during the preprocessing implementation, once the bounds of some variables in constraint i change, the bounds $\overline{b_i}$ and $\underline{b_i}$ of constraint i need to be recalculated again.

2.5.2 Tighten Variable Bounds

The technique of tightening variables bounds has been referred to in Section 2.4.6. Here we discuss this technique in detail.

In Section 2.5.1, we have derived the constraint bounds $\overline{b_i}$ and $\underline{b_i}$. However, we concluded that constraint i is not forcing, redundant or infeasible only if

$$\text{if } \underline{b_i} < b_i \leq \overline{b_i} \text{ when } \sum_{j=1}^{n} a_{ij}x_j \leq b_i;$$
$$\text{if } \underline{b_i} \leq b_i < \overline{b_i} \text{ when } \sum_{j=1}^{n} a_{ij}x_j \geq b_i;$$
$$\text{if } \underline{b_i} < b_i < \overline{b_i} \text{ when } \sum_{j=1}^{n} a_{ij}x_j = b_i.$$

We can use $\overline{b_i}$ and $\underline{b_i}$ to compute implied bounds $\hat{l}_k$ and $\hat{u}_k$ for the participating variables x_k in constraint i. We will discuss the various cases one by one.

1. If the row type of constraint i is less than or equal to, i.e., $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$ and all l_k for $k \in \mathcal{P}_i$ and u_k for $k \in \mathcal{N}_i$ are finite, then in this case $\underline{b_i}$ is finite as well.

   If for $\forall k \in \mathcal{P}_i$, we have $a_{ik} > 0$ and

   $$\underline{b_i} + a_{ik}(x_k - l_k) \leq \sum_{j=1}^{n} a_{ij}x_j \leq b_i,$$

   that imply

   $$x_k \leq \frac{b_i - \underline{b_i}}{a_{ik}} + l_k = \hat{u}_k.$$

   If for $\forall k \in \mathcal{N}_i$, we have $a_{ik} < 0$ and

   $$\underline{b_i} + a_{ik}(x_k - u_k) \leq \sum_{j=1}^{n} a_{ij}x_j \leq b_i,$$

   that imply

   $$x_k \geq \frac{b_i - \underline{b_i}}{a_{ik}} + u_k = \hat{l}_k.$$

   Variable x_k, $k \in \mathcal{P}_i$, gets an implied upper bound $\hat{u}_k$ if $\hat{u}_k < u_k$. Variable x_k, $k \in \mathcal{N}_i$, gets an implied lower bound $\hat{l}_k$ if $\hat{l}_k > l_k$.

2. If the row type of constraint i is greater than or equal to, i.e, $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ and all u_k for $k \in \mathcal{P}_i$ and l_k for $k \in \mathcal{N}_i$ are finite, then in this case $\overline{b_i}$ is finite as well.

   If for $\forall k \in \mathcal{P}_i$, we have $a_{ik} > 0$ and

   $$\overline{b_i} + a_{ik}(x_k - u_k) \geq \sum_{j=1}^{n} a_{ij}x_j \geq b_i,$$

   that imply

   $$x_k \geq \frac{b_i - \overline{b_i}}{a_{ik}} + u_k = \hat{l}_k.$$

If for $\forall k \in \mathcal{N}_i$, we have $a_{ik} < 0$ and

$$\overline{b}_i + a_{ik}(x_k - l_k) \geq \sum_{j=1}^{n} a_{ij}x_j \geq b_i,$$

that imply

$$x_k \leq \frac{b_i - \overline{b_i}}{a_{ik}} + l_k = \hat{u}_k.$$

Variable x_k, $k \in \mathcal{P}_i$, gets an implied lower bound $\hat{l}_k$ if $\hat{l}_k > l_k$. Variable x_k, $k \in \mathcal{N}_i$, gets an implied upper bound $\hat{u}_k$ if $\hat{u}_k > u_k$.

For variable x_k, the new implied lower and upper bounds $\hat{l}_k$ or $\hat{u}_k$ need to be compared with the original bounds. If they conflict, i.e., $\hat{l}_k > u_k$ or $\hat{u}_k < l_k$, then problem (2.3.5) is infeasible. Otherwise, we will check whether the new bounds are tighter than the original ones, i.e., $\hat{l}_k > l_k$ or $\hat{u}_k < u_k$. If tighter, then the bounds of variable x_k need to be updated by (2.4.11):

$$l_k \;=\; \max\,(l_k, \hat{l}_k), \quad u_k \;=\; \min\,(u_k, \hat{u}_k).$$

To the contrary, if the new bounds are looser than the original ones, i.e., $\hat{l}_k < l_k$ or $\hat{u}_k > u_k$, the constraint and the variables are kept without change.

Further, Gondzio [13] noticed that when $\underline{b_i}$ or $\overline{b_i}$ is infinite, it still may be possible to derive an implied bound for a variable. To compute a finite lower bound

$$\underline{b_i} = \sum_{j \in \mathcal{P}_i} a_{ij}l_j \;+\; \sum_{j \in \mathcal{N}_i} a_{ij}u_j,$$

the variables in $\mathcal{P}$ must have their finite lower bounds, while the variables in $\mathcal{N}$ must have their finite upper bounds. However, if only one variable in $\mathcal{P}$ has an infinite lower bound, or only one variable in $\mathcal{N}$ has an infinite upper bound, then we can calculate an implied bound for this variable.

Similarly, if there is an infinite upper bound

$$\overline{b_i} = \sum_{j \in \mathcal{P}_i} a_{ij} u_j \; + \; \sum_{j \in \mathcal{N}_i} a_{ij} l_j,$$

and if only one variable in $\mathcal{P}$ has an infinite upper bound, or only one variable in $\mathcal{N}$ has an infinite lower bound, then also we can calculate an implied bound for this variable.

The detailed discussion is as follows:

1. If the row type is less than or equal to, i.e., $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$, we assume that there exists only one infinite bound:

   (a) $l_k = -\infty$ for $k \in \mathcal{P}_i$, or

   (b) $u_k = +\infty$ for $k \in \mathcal{N}_i$.

   (a) For $k \in \mathcal{P}_i$, we have $a_{ik} > 0$ and

   $$a_{ik} x_k + \sum_{j \in \mathcal{P}_i - \{k\}} a_{ij} l_j + \sum_{j \in \mathcal{N}_i} a_{ij} u_j \leq \sum_{j=1}^{n} a_{ij} x_j \leq b_i,$$

   that imply

   $$x_k \quad \leq \quad \frac{b_i - \sum_{j \in \mathcal{P}_i - \{k\}} a_{ij} l_j - \sum_{j \in \mathcal{N}_i} a_{ij} u_j}{a_{ik}} = \hat{u}_k.$$

   (b) For $k \in \mathcal{N}_i$, we have $a_{ik} < 0$ and

   $$a_{ik} x_k + \sum_{j \in \mathcal{P}_i} a_{ij} l_j + \sum_{j \in \mathcal{N}_i - \{k\}} a_{ij} u_j \leq \sum_{j=1}^{n} a_{ij} x_j \leq b_i,$$

   that imply

   $$x_k \quad \geq \quad \frac{b_i - \sum_{j \in \mathcal{P}_i} a_{ij} l_j - \sum_{j \in \mathcal{N}_i - \{k\}} a_{ij} u_j}{a_{ik}} = \hat{l}_k.$$

If $k \in \mathcal{P}_i$, x_k gets an implied upper bound. If $k \in \mathcal{N}_i$, x_k gets an implied lower bound.

2. If the row type is greater than or equal to, i.e., $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$, we assume there exists only one infinite bound:

$$\text{(a) } u_k = +\infty \text{ for some } \ k \in \mathcal{P}_i, \text{ or}$$
$$\text{(b) } l_k = -\infty \ \text{ for some } \ k \in \mathcal{N}_i.$$

(a) For $k \in \mathcal{P}_i$, we have $a_{ik} > 0$ and

$$a_{ik}x_k + \sum_{j \in \mathcal{N}_i} a_{ij}l_j + \sum_{j \in \mathcal{P}_i - \{k\}} a_{ij}u_j \geq \sum_{j=1}^{n} a_{ij}x_j \geq b_i,$$

that imply

$$x_k \quad \geq \quad \frac{b_i - \sum\limits_{j \in \mathcal{N}_i} a_{ij}l_j - \sum\limits_{j \in \mathcal{P}_i - \{k\}} a_{ij}u_j}{a_{ik}} = \hat{l}_k.$$

(b) For $k \in \mathcal{N}_i$, we have $a_{ik} < 0$ and

$$a_{ik}x_k + \sum_{j \in \mathcal{N}_i - \{k\}} a_{ij}l_j + \sum_{j \in \mathcal{P}_i} a_{ij}u_j \geq \sum_{j=1}^{n} a_{ij}x_j \geq b_i,$$

that imply

$$x_k \quad \leq \quad \frac{b_i - \sum\limits_{j \in \mathcal{N}_i - \{k\}} a_{ij}l_j - \sum\limits_{j \in \mathcal{P}_i} a_{ij}u_j}{a_{ik}} = \hat{u}_k.$$

If $k \in \mathcal{P}_i$, x_k gets an implied lower bound. If $k \in \mathcal{N}_i$, x_k gets an implied upper bound.

Dealing with implied bounds is the same as what we have discussed in the case of $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$.

Gondzio's extension is especially useful for free variables. A free variable can get its implied bounds. If the implied bounds of a free variable are equal, then this variable is fixed and it can be removed from the problem.

Using "tighten variable bounds" techniques, we may find that either problem (2.3.5) is infeasible, or obtain fixed variables, or have some variable bounds tightened.

Once some variable bounds are updated, the new bounds can be used to calculate the bounds of other constraints. Then more forcing constraints or redundant constraints may be found and thus the problem dimension of the LO problem reduces.

2.5.3 Tighten Dual Variable Bounds

The analysis discussed so far are all related to the primal problem. Some techniques can be applied to the dual problem too. For instance, we can use similar techniques to the ones discussed in Section 2.5.2 to tighten the bounds of dual variables.

Similarly, for the dual constraint j, we define the following notations:

$$\mathcal{P}_j^{'} = \{i : a_{ij} > 0\},$$
$$\mathcal{N}_j^{'} = \{i : a_{ij} < 0\},$$
$$p_i \leq y_i \leq q_i,\ i = 1, \ldots, m.$$

Then, the lower and upper bounds $\underline{c_j}$ and $\overline{c_j}$ of the dual constraints j can be easily obtained by:

$$\underline{c_j} = \sum_{i \in \mathcal{P}_j^{'}} a_{ij} p_i + \sum_{i \in \mathcal{N}_j^{'}} a_{ij} q_i,$$

$$\overline{c_j} = \sum_{i \in \mathcal{N}_j^{'}} a_{ij} p_i + \sum_{i \in \mathcal{P}_j^{'}} a_{ij} q_i.$$

We will use these bounds to find the implied lower and upper bounds $\hat{p}_k$ and $\hat{q}_k$

for a participating dual variable y_k in constraint j. First, we present the cases when either $\overline{c_j}$ or $\underline{c_j}$, or both of them are finite.

1. If the type of dual constraint j is less than or equal to, i.e., $\sum_{i=1}^{m} a_{ij}y_i \leq c_j$ and all p_k for $k \in \mathcal{P}_j^{'}$ and q_k for $k \in \mathcal{N}_j^{'}$ are finite, then in this case $\underline{c_j}$ is finite as well.

   For $\forall k \in \mathcal{P}_j^{'}$, we have $a_{kj} > 0$ and

   $$\underline{c_j} + a_{kj}(y_k - p_k) \leq \sum_{i=1}^{m} a_{ij}y_i \leq c_j,$$

   that imply

   $$y_k \leq \frac{c_j - \underline{c_j}}{a_{kj}} + p_k = \hat{q}_k.$$

   For $\forall k \in \mathcal{N}_j^{'}$, we have $a_{kj} < 0$ and

   $$\underline{c_j} + a_{kj}(y_k - q_k) \leq \sum_{i=1}^{m} a_{ij}y_i \leq c_j,$$

   that imply

   $$y_k \geq \frac{c_j - \underline{c_j}}{a_{kj}} + q_k = \hat{p}_k.$$

   Variable y_k, $k \in \mathcal{P}_j^{'}$, gets an implied upper bound $\hat{q}_k$ if $\hat{q}_k < q_k$. Variable y_k, $k \in \mathcal{N}_j^{'}$, gets an implied lower bound $\hat{p}_k$ if $\hat{p}_k > p_k$.

2. If the type of dual constraint j is greater than or equal to, i.e., $\sum_{i=1}^{m} a_{ij}y_i \geq c_j$ and all p_k for $k \in \mathcal{N}_j^{'}$ and q_k for $k \in \mathcal{P}_j^{'}$ are finite, then in this case $\overline{c}_j$ is finite as well.

   For $\forall k \in \mathcal{P}_j^{'}$, we have $a_{kj} > 0$ and

   $$\overline{c}_j + a_{kj}(y_k - q_k) \geq \sum_{i=1}^{m} a_{ij}y_i \geq c_j,$$

that imply

$$y_k \geq \frac{c_j - \overline{c_j}}{a_{kj}} + q_k = \hat{p}_k.$$

For $\forall k \in \mathcal{N}_j^{'}$, we have $a_{kj} < 0$,

$$\overline{c}_j + a_{kj}(y_k - p_k) \geq \sum_{i=1}^{m} a_{ij} y_i \geq c_j,$$

that imply

$$y_k \leq \frac{c_j - \overline{c_j}}{a_{kj}} + p_k = \hat{q}_k.$$

Variable y_k, $k \in \mathcal{P}_j^{'}$, gets an implied lower bound $\hat{p}_k$ if $\hat{p}_k > p_k$. Variable y_k, $k \in \mathcal{N}_j^{'}$, gets an implied upper bound $\hat{q}_k$ if $\hat{q}_k < q_k$.

The analysis is valid when either $\underline{c}_j$ or $\overline{c}_j$ or both are finite. Further, Gondzio [13] found that it still may be possible to tighten an implied bound for a variable when either $\underline{c_j}$ or $\overline{c_j}$ is infinite. The analysis is analogous to what we did for the primal problem.

1. If the type of dual constraint j is less than or equal to, i.e., $\sum_{i=1}^{m} a_{ij} y_i \leq c_j$, we assume that there exists only one infinite bound:

   (a) $p_k = -\infty$ for some $k \in \mathcal{P}_j^{'}$, or

   (b) $q_k = +\infty$ for some $k \in \mathcal{N}_j^{'}$.

   (a) For $k \in \mathcal{P}_j^{'}$, we have $a_{kj} > 0$ and

$$a_{kj} y_k + \sum_{i \in \mathcal{P}_j^{'} - \{k\}} a_{ij} p_i + \sum_{i \in \mathcal{N}_j^{'}} a_{ij} q_i \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j,$$

   that imply

$$y_k \quad \leq \quad \frac{c_j - \sum_{i \in \mathcal{P}_j^{'} - \{k\}} a_{ij} p_i - \sum_{i \in \mathcal{N}_j^{'}} a_{ij} q_i}{a_{kj}} = \hat{q}_k.$$

(b) For $k \in \mathcal{N}_j^{'}$, we have $a_{kj} < 0$ and

$$a_{kj}y_k + \sum_{i \in \mathcal{N}_j^{'} - \{k\}} a_{ij}p_i + \sum_{i \in \mathcal{P}_j^{'}} a_{ij}q_i \leq \sum_{i=1}^{m} a_{ij}y_i \leq c_j,$$

that imply

$$y_k \quad \geq \quad \frac{c_j - \sum\limits_{i \in \mathcal{N}_j^{'} - \{k\}} a_{ij}p_i - \sum\limits_{i \in \mathcal{P}_j^{'}} a_{ij}q_i}{a_{kj}} = \hat{p}_k.$$

If $k \in \mathcal{P}_j^{'}$, y_k gets an implied upper bound $\hat{q}_k$ if $\hat{q}_k < q_k$. If $k \in \mathcal{N}_j^{'}$, y_k gets an implied lower bound $\hat{p}_k$ if $\hat{p}_k > p_k$.

2. If the type of dual constraint j is greater than or equal to, i.e., $\sum\limits_{i=1}^{m} a_{ij}y_i \geq c_j$, we assume there exists only one infinite bound:

   (a) $q_k = +\infty$ for some $k \in \mathcal{P}_j^{'}$, or

   (b) $p_k = -\infty$ for some $k \in \mathcal{N}_j^{'}$.

   (a) For $k \in \mathcal{P}_j^{'}$, we have $a_{kj} > 0$ and

$$a_{kj}y_k + \sum_{i \in \mathcal{P}_j^{'} - \{k\}} a_{ij}q_i + \sum_{i \in \mathcal{N}_j^{'}} a_{ij}p_i \geq \sum_{i=1}^{m} a_{ij}y_i \geq c_j,$$

   that imply

$$y_k \quad \geq \quad \frac{c_j - \sum\limits_{i \in \mathcal{N}_j^{'} - k} a_{ij}p_i - \sum\limits_{i \in \mathcal{P}_j^{'}} a_{ij}u_i}{a_{kj}} = \hat{p}_k.$$

   (b) For $k \in \mathcal{N}_j^{'}$, we have $a_{kj} < 0$ and

$$a_{kj}y_k + \sum_{i \in \mathcal{P}_j^{'}} a_{ij}q_i + \sum_{i \in \mathcal{N}_j^{'} - \{k\}} a_{ij}p_i \geq \sum_{i=1}^{m} a_{ij}y_i \geq c_j,$$

that imply

$$y_k \leq \frac{c_j - \sum_{i \in \mathcal{N}_j' - \{k\}} a_{ij} p_i - \sum_{i \in \mathcal{P}_j'} a_{ij} q_i}{a_{kj}} = \hat{q}_k.$$

If $k \in \mathcal{P}_j'$, y_k gets an implied lower bound $\hat{p}_k$ if $\hat{p}_k > p_k$. If $k \in \mathcal{N}_j'$, y_k gets an implied upper bound $\hat{q}_k$ if $\hat{q}_k < q_k$.

The new implied bounds $\hat{q}_k$ and $\hat{p}_k$ need to be compared with their original bounds q_k and p_k. If they are in conflict, then the dual problem (2.3.6) is infeasible. Otherwise, check whether the new bounds are tighter than the original ones. If tighter, the bounds need to be updated as:

$$p_k = \max(\hat{p}_k, p_k),$$
$$q_k = \min(\hat{q}_k, q_k).$$

2.5.4 Dominated Variables

From Section 2.5.3, we know how to calculate the lower and upper bounds $\underline{c}_j$ and $\overline{c}_j$ for dual constraint j. They have the following relationship with the dual constraint:

$$\underline{c}_j \leq \sum_{i=1}^{m} a_{ij} y_i \leq \overline{c}_j.$$

We also know that

$$\sum_{i=1}^{m} a_{ij} y_i + v_j - w_j = c_j.$$

In our subsequent discussions, the lower and upper bounds of variable x_j will decide the result of our analysis. Therefore, we consider two cases separately: $u_j = +\infty$ or $l_j = -\infty$. In both cases, the relationships among c_j, $\underline{c}_j$ and $\overline{c}_j$ are discussed.

1. If $u_j = +\infty$, the result is decided by the value of l_j.

   From (2.3.9), we know that when $u_j = +\infty$ then $w_j = 0$. We have

   $$\sum_{i=1}^{m} a_{ij} y_i + v_j = c_j. \tag{2.5.26}$$

   (a) If $c_j < \underline{c_j}$, (2.5.26) can not happen, therefore the dual problem (2.3.6) is infeasible.

   (b) If $\overline{c_j} < c_j$, from (2.5.26), we know that v_j is strictly positive, i.e., $v_j > 0$. In this case, we call column j a dominated column. From $(x_j - l_j)v_j = 0$ in (2.3.10), we know:

      i. If $l_j = -\infty$, then the dual problem (2.3.6) is infeasible.

      ii. If $l_j \neq -\infty$, then x_j is fixed at its lower bound l_j.

   (c) If $(\overline{c_j} = c_j)$, we know that (2.5.26) is satisfied when $v_j \geq 0$. In this case, column j is called a weakly dominated column. Variable x_j can be eliminated the same way as a dominated column [3]. Here, a more general condition can be used to eliminate variable x_j as follows:

   Proposition 2.5.1. *[13] Assume that $l_j > -\infty$, $u_j = +\infty$ and $\overline{c_j} = c_j$. If for all $k \in \mathcal{P}_j'$, the constraint k in (2.3.5) is of the ' $\leq$ ' type, and for all $k \in \mathcal{N}_j'$, the constraint k in (2.3.5) is of the ' $\geq$ ' type, then there exsits an optimal solution such that $x_j = l_j$ or the LO problem (2.3.5) has no optimal solution.*

   (d) If $\underline{c_j} < c_j < \overline{c_j}$, then x_j can not be eliminated, but we may use the technique presented in Section 2.5.3 to find implied bounds for the participating dual variables in dual constraint j.

2. If $l_j = -\infty$, then the result is decided by the value of u_j.

From (2.3.9), we know that $v_j = 0$ when $l_j = -\infty$. We have

$$\sum_{i=1}^{m} a_{ij} y_i - w_j = c_j. \tag{2.5.27}$$

(a) If $c_j > \overline{c_j}$, then (2.5.27) can not happen, therefore the dual problem (2.3.6) is infeasible.

(b) If $\underline{c_j} > c_j$, then we call column j a dominated column.

From (2.5.27), we know that $w_j > 0$. From $(u_j - x_j)w_j = 0$ in (2.3.10), we know:

i. If $u_j = +\infty$, then the dual problem (2.3.6) is infeasible.

ii. If $u_j \neq +\infty$, then x_j is fixed to its upper bound u_j.

(c) If $\underline{c_j} = c_j$, then column j is called a weakly dominated column. In this case, x_j can be eliminated like a dominated column. Here, a more general condition can be used to eliminate variable x_j as follows:

Proposition 2.5.2. *[13] Assume that $l_j = -\infty$, $u_j < +\infty$ and $\underline{c_j} = c_j$. If for all $k \in \mathcal{P}_j^{'}$, the constraint k in (2.3.5) is of the '$\geq$' type, and for all $k \in \mathcal{N}_j^{'}$, the constraint k in (2.3.5) is of the '$\leq$' type, then there exsits an optimal solution such that $x_j = u_j$ or the LO problem (2.3.5) has no optimal solution.*

(d) If $\underline{c_j} < c_j < \overline{c_j}$, then x_j can not be eliminated, but we may use the technique in Section 2.5.3 to find implied bounds.

2.6 Make A Sparser

Large scale LO problems usually contain relatively few nonzero elements. This feature is referred to as sparsity, or low density. For matrix A, its sparsity is defined as the

number of nonzero elements divided by the total positions in this matrix. If $\varrho(A)$ denotes the sparsity and $\nu(A)$ the number of nonzero entries, the sparsity of matrix A is given by the formula:

$$\varrho(A) = \frac{\nu(A)}{mn}.$$

Similarly, the sparsity of a vector, e.g., vector c is defined as

$$\varrho(c) = \frac{\nu(c)}{n}.$$

The sparsity of matrix A needs to be utilized in implementations and matrix A needs to be made as sparse as possible during preprocessing. Improved sparsity (i.e., reducing $\varrho(A)$) not only saves memory and decreases the cost of calculations at each iteration, but also improves numerical stability and allows to produce more accurate solutions.

Hoffman and McCormick [15, 19] defined sparsity problem as a general optimization problem and concluded that finding the exact solution of the sparsity problem is an NP-complete problem.

Definition 2.6.1. *Sparsity Problem [6]: Given $A \in R^{m\times n}$ and $b \in R^m$, that define the constraints $Ax = b$, find a nonsingular $T \in R^{m\times n}$ such that $\hat{A} \equiv TA$ is as sparse as possible, i.e.,* $\min\ \{\varrho(\hat{A}) \mid \hat{A} = TA$ *and* $T \in R^{m\times n}$ *nonsingular* $\}$.

Gondzio [13] proposed a heuristic algorithm that is simple to implement and can reduced the sparsity of matrix A effectively. The algorithm is based on the analysis of the sparsity pattern of each equality constraint in A. We try to find a row whose sparsity pattern is a superset of the sparsity pattern of another row. In this way we can make A sparser. For instance, if there are row i and row k with the following sparsity patterns:

$$\begin{array}{llll} \text{x} & \text{x} & \text{x} & \Leftarrow \text{row } i, \\ \text{x} \quad \text{x} \quad \text{x} \quad \text{x} \quad \text{x} \quad \text{x} & & & \Leftarrow \text{row } k, \end{array}$$

then we say that the sparsity pattern of row k is a superset of the sparsity pattern of row i. In this case, we select row i as a pivoting row. By choosing a suitable scalar λ, this multiple of row i is added to row k so that at least one nonzero element in row k is eliminated, i.e.,

$$\hat{a}_{kj} = a_{kj} + \lambda a_{ij},\ j = 1, \ldots, n,$$

and the corresponding RHS value is also changed to:

$$b_k = b_k + \lambda b_i.$$

The solution of problem (2.3.5) with the revised new row k is the same as that of the original one.

The main difficulty is how to find a pivoting row and the corresponding rows whose sparsity patterns are the superset of this row. To reduce the computational effort of this process, a linked list for all the equality constraint is built in an ascending order by the number of nonzero elements. We usually start with a row, e.g., row i, that has the smallest number of nonzero elements. Then we check each column that has nonzero element intersection with row i and choose the column, e.g., column j, that has the smallest number of nonzero element. The superset of row i is found by scanning each row, e.g., row k, that has nonzero element in column j. To be efficient, if the number of nonzero elements of row k is less than that of row i, then row k is rejected from our candidate superset row list. It is reported by Gondzio that if the sparsity pattern of row k is not a superset of the nonzero pattern of row i, then this fact usually can be easily found after checking the first three elements.

This algorithm can guarantee that at each elimination step at least one nonzero element is eliminated from the LO problem. If we are lucky, more nonzero elements may be eliminated at a single elimination step. And the advantage of this procedure is that no new fill in is produced and no additional storage is needed to store the updated matrix.

Observe that among the basic and the advanced logical analysis technologies, there are some, e.g., fixed variables, singleton rows, forcing rows and dominated variables that can directly or indirectly help to improve the sparsity of matrix A. Thus, in some optimization software packages, the task of making A sparser is implemented only by relying on those techniques.

2.7 Make A Full Rank

For IPMs solvers, to make matrix A full rank is mandatory, thus it is indispensable to find out the dependent rows if they exist.

Definition 2.7.1. *[2] Let $S \subseteq \{1, \ldots, m\}$ be a non-empty set. If there exists a vector $s \in R^m$ such that $s_i \neq 0$ for all $i \in S$ and $\sum_{i \in S} s_i A_i = 0$, then the rows in matrix A corresponding to the indices in S are said to be linearly dependent.*

Once dependent rows are found, then one of them can be removed from the LO problem due to the following property:

Proposition 2.7.1. *[2] Assume that the rows corresponding to the indices in S are linearly dependent. In this case, either one of these rows can be removed from the problem without changing the optimal solution or the problem is infeasible.*

Therefore, when linearly dependent rows are found in matrix A, at least one of them

can be removed from the problem if the LO problem is feasible. The removal process is repeated as long as linearly dependent rows are found. Finally a coefficient matrix A with full row rank is sent to the solver. The problem is how to find linearly dependent rows.

Theoretically, Gaussian Elimination can be used to check the dependency of the rows. For system $Ax = b$, we combine matrix $A^{m\times n}$ with vector b to get the matrix $\overline{A} = [A \mid b\,]$ with the size of $m \times (n+1)$. A sequence of elementary row operations can be applied to the rows in matrix $\overline{A}$ to identify its rank. The row operations include:

- Multiply a row by a scalar;
- Swap two rows;
- Add a scalar multiple of one row to another row.

A detailed description of Gaussian Elimination is as follows:

Examine each row starting from the first row of matrix $\overline{A}$.

1. If there are some nonzero entries in the examined row, e.g., row i, then select one of the nonzero entries, e.g., $a_{ip} \neq 0$ for some $1 \leq p \leq n$, as the pivot element. Row i is called the pivot row and column p is called the pivot column. Perform row operations to all the rows below the pivot row i, i.e., for all $i < k \leq m$ to convert the corresponding nonzero coefficient a_{kp} to zero.
2. If there are only zero entries in the examined row i. There are two cases:
   (a) If $a_{i(n+1)} = 0$, then this row is dependent. It can be removed from the problem;

(b) If $a_{i(n+1)} \neq 0$, then the LO problem is infeasible;

If matrix A is detected to be singular, then the indices of the linearly dependent rows are recorded. By removing those rows from the original matrix A, the remaining submatrix of matrix A has full row rank and it is sent to the solver. If the problem is checked to be infeasible, then we get an infeasibility certificate directly without solving the problem.

In practice, since large scale LO problems usually have relatively few nonzero elements, to do a complete Gaussian Elimination is far too expensive. It not only destroys the sparsity structure of matrix A, but also affects numerical stability. Thus, in practice a modified method is used instead. There are two factors to be considered during the implementation: avoid creating much fill in and keep numerical stability. What we want to find is just the linearly dependent rows.

1. We can find all zero rows first and check if the corresponding RHS coefficient is zero. If there is a RHS coefficient is nonzero, then the problem (2.3.5) is infeasible. Otherwise, remove these zero rows.

2. Then find all the columns that have only one nonzero entry and its corresponding row. We do not remove them. These rows are obviously linearly independent and for linear dependency, we need to check only the remaining submatrix. For instance, if the coefficient a_{kj} is the unique nonzero element in column j, then column j and row k are eliminated during the process of finding linear dependency. The singularity of the remaining matrix is the same as the original matrix A.

3. For the remaining matrix, to check the matrix singularity, we need to choose a pivot element for each row. After k pivots, let $A^{(m-k)\times(n-k)}$ denote the remaining

coefficient matrix. To preserve sparsity, the generalized Markowitz pivoting strategy is used to choose the pivot element. We only consider the rows and columns that remained in the system. The rows and columns that have already been pivoted are ignored. Let ϱ_i denote the number of remaining nonzero elements for row i and let τ_j denote the number of remaining nonzero elements for column j. The Markowitz criteria suggests to select a_{ij} as a pivot element in the remaining matrix $A^{(m-k)\times(n-k)}$ at the kth stage that minimizes

$$\min\{(\varrho_i - 1)(\tau_j - 1)\}. \tag{2.7.28}$$

Numerical stability is ensured by avoiding too small pivot elements. If the value of a_{ij} obtained by Markowitz's rule is "too small", then the first acceptable candidate that still gives a small $(\varrho_i - 1)(\tau_j - 1)$ value is chosen instead.

The implementation of Markowitz' strategy requires the knowledge of the updated sparsity pattern of the reduced $(m-k)\times(n-k)$ submatrix at the kth stage of Gaussian Elimination. For efficient search, both the row-wise and column-wise storage of the coefficient matrix are needed.

The following pseudocode shows the pivoting algorithm for large scale LO problems.

Pivoting Algorithm

Input:

row size m and column size n of matrix A;

tolerance ϵ;

begin

find row p whose nonzero element number equals to 0.

if the corresponding right-hand-side value equals to 0.

mark row p.

else

the LO problem is infeasible; **return**;

find column q whose nonzero element number equals to 1.

find the unique nonzero element a_{tq} in column q

mark row t and column q

reorder matrix A such that marked rows and columns at the end of the matrix.

$p = 1$;

while $p \leq m$ **do**

find the nonzero element a_{ij} that minimizes Equation (2.7.28)

if $(a_{ij} < \epsilon)$ or $(a_{ij} == 0)$

find the second candidate $a_{i'j'}$ minimizing Equation (2.7.28)

$i = i'; j = j'$;

end

swap row i with row p and pivoting to the rows below row i.

mark row i and column j.

end

At the end of the pivoting algorithm, find the rows whose nonzero element number is equal to zero. If the corresponding RHS equals to zero, then the row is removed from the problem. Otherwise, the LO problem is infeasible. When all the empty rows are removed from the LO problem, the matrix is made to have full rank.

Except for Gaussian Elimination, the heuristic of making A sparser and checking duplicate rows can both help to eliminate the linearly dependent rows to make A full rank.

2.8 Scaling

In LO, if there are some very large and very small numbers among the coefficients of A, b and c, then the problem may be difficult to solve numerically. Scaling is used to avoid such instability and increase the solution accuracy by adjusting the numerical characteristics of the LO problem.

Scaling is a linear transformation where the rows and/or columns are multiplied by some scalar factors. Scaling can be represented by multiplying by diagonal matrices $R^{m\times m}$ and $T^{n\times n}$, where matrix R is the row scaling factor for A and matrix T is the column scaling factor for A. If r_i is the row scalar factor for row i and t_j is the column scalar factor for column j, then $R = \text{diag}(r_1, ..., r_m)$ and $T = \text{diag}(t_1, ..., t_n)$.

If x is the solution of the problem before scaling and $\hat{x}$ is the solution of the problem after scaling, we have:

$$RAT\hat{x} = Rb \text{ and } x = T\hat{x}.$$

Every scaling needs the adjustment of the other data vectors, i.e., l, u and c in

the LO problem. After scaling, the vectors $\hat{u}$, $\hat{l}$ and $\hat{c}$ become:

$$\hat{l} = T^{-1}l, \quad \hat{u} = T^{-1}u \text{ and } \hat{c} = Tc.$$

It is clear that the objective values of the scaled problem and unscaled problem are the same, i.e., $\hat{c}^T\hat{x} = c^Tx$.

Although scaling is used quite often, it is still little understood. Some scaling methods are reputed to be more effective than others, but there is no theoretical basis of their comparisons. These comparisons and recommendations [7] are obtained just by some experimental evidence [24]. In practice, scaling may cause opposite effect on an LO problem. A good scaling can help to solve an LO problem reliably, however, a bad scaling may destroy numerical stability. We need to know how to measure the result of scaling. There are two methods [7] to measure the result of scaling by computing the magnitudes of the nonzero elements in matrix A.

1. Compute
$$\frac{\max|a_{ij}|}{\min|a_{ij}|} \text{ for all } a_{ij} \neq 0. \tag{2.8.29}$$
The matrix is well scaled if this value is not larger than a threshold value of about $10^6 - 10^8$.

2. Compute
$$\sum_{a_{ij}\neq 0} (\log|a_{ij}|)^2 \text{ for all } a_{ij} \neq 0. \tag{2.8.30}$$
Generally speaking, the smaller the value of Equation (2.8.30), the better the scaling is. It is impossible to give some numerical estimates for (2.8.30) because its value increases when the number of its entries grows. The sum of $(\log|a_{ij}|)^2$ is used here is to reduce the sensitivity to some extreme large or small values. This is the advantage of (2.8.30) over (2.8.29).

There are three frequently used scaling methods [17] that can reduce the spread of the magnitudes of the nonzero elements in the LO problem:

1. Equilibration method:

   Row i is scaled by multiplying by the reciprocal of the largest absolute element in the row:

   $$r_i^{-1} = \max\{|a_{ij}| : j = 1, \ldots, n\}, \text{ and } R = \text{diag}(r).$$

   Column j is scaled by multiplying by the reciprocal of the largest absolute element in the column:

   $$t_j^{-1} = \max\{|a_{ij}| : i = 1, \ldots, m\}, \text{ and } T = \text{diag}(t).$$

2. Geometric method:

   Row i is scaled by the factor r_i, where

   $$r_i^{-1} = \sqrt{\max\{|a_{ij}|\} \times \min\{|a_{ij}|\}}, 1 \leq j \leq n, \text{ and } R = \text{diag}(r).$$

   Column j is scaled by the factor t_j, where

   $$t_j^{-1} = \sqrt{\max\{|a_{ij}|\} \times \min\{|a_{ij}|\}}, 1 \leq i \leq m, \text{ and } T = \text{diag}(t).$$

3. Arithmetic mean method:

   Row i is scaled by the reciprocal of the arithmetic mean of the nonzero entries in the row:

   $$r_i^{-1} = \frac{\sum_{j=1}^{n} a_{ij}}{n}, \text{ and } R = \text{diag}(r).$$

   Column j is scaled by the reciprocal of the arithmetic mean of the nonzero entries in the column:

   $$t_j^{-1} = \frac{\sum_{i=1}^{m} a_{ij}}{m}, \text{ and } T = \text{diag}(t).$$

There are not many papers in the open literature with computational results comparing the different scaling methods. It is hard to say which method is the best. Based on a set of experimental results, Tomlin [24] concluded that to get a satisfactory scaling result with simple and inexpensive methods, the geometric mean method, optionally followed by Equilibration method is recommended.

Recall the two computed measures about scaling result. Based on (2.8.30), Curtis and Reid [17] designed a special approximation algorithm to find the scaling factors R and T. Assume that the nonzero coefficients a_{ij} in matrix A can be written in the normalized floating point form as:

$$a_{ij} = f_{ij}2^{e_{ij}}, \text{ where } \frac{1}{2} \leq f_{ij} \leq 1, \text{ and } e_{ij} \text{ is an integer,}$$

then the row scaling factor r_i for row i and the column scaling factor t_j for column j can be calculated as:

$$r_i = 2^{\rho_i} \text{ and } t_j = 2^{\gamma_j},$$

where ρ_i and γ_j are integers. It is shown in [7] that to get $R = \text{diag}(r_1, ..., r_m)$ and $T = \text{diag}(t_1, ..., t_n)$, one needs to solve the following equation system

$$\begin{pmatrix} M & Z \\ Z^T & N \end{pmatrix} \begin{pmatrix} \rho \\ \gamma \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix}, \tag{2.8.31}$$

where $\rho = [\rho_1, \ldots, \rho_m]^T$, $\gamma = [\gamma_1, \ldots, \gamma_n]^T$, matrix $M \in R^{m\times m}$ is a diagonal matrix whose diagonal element M_{ii} is the number of the nonzero entries in row i, and matrix $N \in R^{n\times n}$ is a diagonal matrix whose diagonal element N_{jj} is the number of the nonzero entries in column j. Vectors $s \in R^m$ and $t \in R^n$ are vectors whose elements s_i and t_j are the sums of the logarithms of the nonzero entries in row i and column j, respectively, i.e., $s_i = \sum_{j=1}^{n} e_{ij}$ and $t_j = \sum_{i=1}^{m} e_{ij}$. Matrix Z is the nonzero pattern of matrix A, where $z_{ij} = 1$ if $a_{ij} \neq 0$ and $z_{ij} = 0$ if $a_{ij} = 0$.

Curtis and Reid [7] designed an algorithm to solve the system (2.8.31) approximately with an iterative procedure. The initial solution is set with $\rho_0 = M^{-1}s$ and $\gamma_0 = 0$. The residual of the system is

$$\begin{pmatrix} 0 \\ \delta_0 \end{pmatrix}, \text{ where } \delta_0 = t - Z^T M^{-1} s.$$

By setting $h_{-1} = 0$, $\delta_{-1} = 0$, $q_0 = 1$ and $s_0 = \delta_0^T N^{-1} \delta_0$, the subsequent residuals can be calculated recursively with the following equations:

$$\delta_{j+1} = \begin{cases} -\dfrac{1}{q_j}(ZN^{-1}\delta_j + h_{j-1}\delta_{j-1}), & \text{if } j \text{ is even,} \\ -\dfrac{1}{q_j}(ZM^{-1}\delta_j + h_{j-1}\delta_{j-1}), & \text{if } j \text{ is odd,} \end{cases} \tag{2.8.32}$$

and

$$s_{j+1} = \begin{cases} \delta_{j+1}^T M^{-1} \delta_{j+1}, & \text{if } j \text{ is even,} \\ \delta_{j+1}^T N^{-1} \delta_{j+1}, & \text{if } j \text{ is odd,} \end{cases} \tag{2.8.33}$$

and

$$h_j = q_j \frac{s_{j+1}}{s_j}, \tag{2.8.34}$$

$$q_{j+1} = 1 - h_j. \tag{2.8.35}$$

When j is even, the residuals δ is $(0, \delta_j)^T$; when j is odd, the residuals δ is $(\delta_j, 0)^T$. At each iteration, one of the vectors ρ_j and γ_j is calculated by the following equations:

$$\gamma_{2k+2} = \gamma_{2k} + \frac{1}{q_{2k}q_{2k+1}}(N^{-1}\delta_{2k} + h_{2k-1}h_{2k-2}(\gamma_{2k} - \gamma_{2k-2})), \tag{2.8.36}$$

$$\gamma_{2k+1} = \gamma_{2k} + \frac{1}{q_{2k}}(N^{-1}\delta_{2k} + h_{2k-1}h_{2k-2}(\gamma_{2k} - \gamma_{2k-2})), \tag{2.8.37}$$

for $k = 0, 1, \ldots,$ with $h_{-2} = 0, \gamma_{-2} = 0$,

$$\rho_{2k+3} = \rho_{2k+1} + \frac{1}{q_{2k+1}q_{2k+2}}(M^{-1}\delta_{2k} + h_{2k-1}h_{2k}(\rho_{2k+1} - \rho_{2k-1})), \tag{2.8.38}$$

$$\rho_{2k+2} = \rho_{2k+1} + \frac{1}{q_{2k+1}}(M^{-1}\delta_{2k} + h_{2k-1}h_{2k}(\rho_{2k+1} - \rho_{2k-1})), \tag{2.8.39}$$

for $k = 0, 1, \ldots,$ with $\rho_{-1} = \rho_1 = \rho_0$.

The stopping criteria is as follows:

$$s_{j+1} \leq 0.01\nu(A), \tag{2.8.40}$$

where $\nu(A)$ is the number of nonzero entries in matrix A.

The solutions of Equation (2.8.31) are rounded to the nearest integer, to ρ_i and γ_j. Then the row scaling factor is set to $r_i = 2^{\rho_i}$ and the column scaling factor is set to $t_j = 2^{\gamma_j}$. This algorithm is proved to be efficient by experience, as it is demonstrated in [24]. It reduces the computational effort to solve the LO problem, thus, it is implemented in our code.

Scaling can be performed repeatedly based on the user's need. If scaling is performed k times, then one has

$$R_k \ldots R_1 A T_1 \ldots T_k x = R_k \ldots R_1 b,$$

and

$$\hat{x} = T_k^{-1} \ldots T_1^{-1} x, \tag{2.8.41}$$

the vectors $\hat{u}$, $\hat{l}$ and $\hat{c}$ after k scalings become:

$$\hat{l} = T_k^{-1} \ldots T_1^{-1} l, \;\; \hat{u} = T_k^{-1} \ldots T_1^{-1} u \;\text{ and }\; \hat{c} = T_k \ldots T_1 c.$$

Scaling aims to improve the accuracy of the LO solution, however, there is no performance guarantee. Sometimes scaling may cause an adverse effect and some problems may get more accurate solution without scaling [17]. In the case that when little or no numerical information is available about the LO problem, scaling is usually used to improve performance of LO algorithms [24].

Chapter 3

Postprocessing

Postprocessing is the collection of operations that, after solving the problem, undo all problem transformations that were made in preprocessing. In this chapter, we present all those postprocessing operations corresponding to preprocessing techniques that were discussed in Chapter 2.

In preprocessing, a number of techniques are used to reduce the size of the original LO problem so that it can be solved efficiently and reliably. After preprocessing, the LO problem becomes simpler and smaller. Thus the reduced LO problem, that is submitted to the solver, is not exactly the same problem as the original one. If the reduced problem is primal and dual feasible, then after solving it, the solution and the problem itself need to be transformed back to the original form as it was presented in terms of the original variables and constraints. Such a restoration procedure is called postprocessing. In postprocessing, the solution of the original LO problem is recovered.

3.1 The Order of Actions in Postprocessing

To recover the solution of the original problem, one needs the optimal values of the variables of the reduced problem that are obtained directly from the solver. Moreover, each change of the original problem made in preprocessing has to be recorded properly so that it can be found and retrieved in the proper order easily. Generally, the changes are stored in the data structure of a stack where the most recently happened operation is recorded on the top. During the recovery procedure, the information on the top of the stack is used first. Then it is deleted and the next information becomes the top. Such "undo" operations are repeated until the stack becomes empty. Thus, the restoration order in the stack is "last performed in preprocessing, first recovered in postprocessing". The operations are performed in the reversed order w.r.t. preprocessing.

Recall that the action order in preprocessing is:

1. Transformed the original LO problem (1.1.4) into the standard form (2.1.1).
2. Basic, advanced techniques and some comprehensive techniques, e.g., make matrix A sparser and full rank are used to get the revised problem.
3. Scaling is used to improve the numerical stability of matrix A.

Thus the recovery order is reversed in postprocessing:

1. Recover the problem to the unscaled problem.
2. Recover the solution of the standard form problem (2.1.1).
3. Recover the solution of the original LO problem (1.1.4).

In the sequel, we present the postprocessing operations one by one.

3.2 Unscaling

Unscale an LO problem is the first thing to do in postprocessing. As presented in Section 2.8, the LO problem is scaled as

$$RAT\hat{x} = Rb,$$

where the row scaling diagonal matrix $R = \text{diag}(r_1, ..., r_m)$ and the column scaling diagonal matrix $T = \text{diag}(t_1, ..., t_n)$ are stored in the stack. Then the value of x is recovered by (2.8.41) as

$$x = T\hat{x}.$$

The logical variables, i.e., the slack variables are not involved in column scaling. To recover the corresponding values in the unscaled LO problem, their values should be divided by the corresponding row scaling factors, i.e.,

$$x_i = \frac{\hat{x}_i}{r_i}.$$

The dual solution of the unscaled problem is obtained by multiplying by the row scaling factors, i.e., $y_i = r_i\hat{y}_i$.

3.3 Recover the Solution of the Standard Form Problem

3.3.1 Empty Row

An empty row i can either directly imply that problem (2.3.5) is infeasible or row i is redundant. If the empty row i is implied that the problem is infeasible, then we get

the result directly and the problem is not submitted to the solver and postprocessing. If row i is redundant, then the solution of problem (2.3.5) is not affected. The corresponding value of y_i in problem (2.3.6) can be set to any value, for instance, to zero.

3.3.2 Empty Column

An empty column j can either directly imply that problem (2.3.5) is unbounded or variable x_j is fixed to either its lower bound l_j or its upper bound u_j. If variable x_j is fixed, it is removed from the problem. Its value is recorded in the stack for later restoration. The solution of problem (2.3.6) is not affected by empty column removal and $s_j = c_j - \sum_{i=1}^{m} a_{ij} y_i$. We have $a_{ij} = 0$ when column j is an empty column, so $s_j = c_j \geq 0$.

3.3.3 Fixed Variable

If variable x_j is a fixed variable, its value is stored in the stack. The solution of problem (2.3.6) is not affected and the dual slack variable is calculated as $s_j = c_j - \sum_{i=1}^{m} a_{ij} y_i$.

3.3.4 Singleton Row

If a singleton row i is an equality constraint, it either implies that problem (2.3.5) is infeasible or variable x_k is fixed to $\dfrac{b_i}{a_{ik}}$ (see Section 2.4.5). When the problem is detected to be infeasible, then we directly get the result without solving the problem. When x_k is fixed, its value is stored before variable x_k and constraint i are removed from the LO problem. The value of the corresponding dual variable y_i is obtained by

$$y_i = \frac{c_k - \sum_{i=1, i \neq k}^{m} a_{ij} y_i}{a_{ik}},$$

while due to the complementarity constraint, we must set

$$s_k = 0.$$

If a singleton row i is an inequality, it results in one of the following three cases: problem (2.3.5) is infeasible, row i is redundant or a tightened bound of x_j is obtained. If row i is infeasible, we directly get the result without solving the LO problem. If row i is redundant, the value of y_i is set to zero. If a tightened upper bound u_j for x_j is obtained, then it does not affect the solution. If a tightened lower bound l_j is obtained and it is transformed into zero in preprocessing, then the value of x_j is recovered as $x_j = \hat{x}_j + l_j$ if $\hat{x}_j$ is the solution obtained from the solver.

3.3.5 Singleton Column

If column j is a singleton column and x_j is a free or an implied free variable, then constraint i and variable x_j are removed from the problem. We have

$$y_k = \frac{c_j}{a_{kj}} \text{ and } s_j = 0,$$

and

$$x_j = \frac{b_k - \sum_{p=1, p \neq j}^{n} a_{kp} x_p}{a_{kj}}.$$

3.3.6 Duplicate Rows

If row i and row k are duplicate rows, as described in Section 2.4.7, we may know that either problem (2.3.5) is infeasible or one of the two rows can be eliminated. We consider the following two cases:

1. If both of row i and row k are equalities, we need to check whether their corresponding RHS values are in proportion. If they are in proportion, i.e., $b_i = \lambda b_k$, then one of the constraints can be removed from the problem. For instance, assume that row k is eliminated, then the value of y_i can be obtained directly from the solver and the value of y_k is set to zero. The solution of problem (2.3.5) is not affected.

2. Recall that we have discussed three other cases in preprocessing: both rows are inequalities and they have the same row types, or both rows are inequalities and they are in different row types, or one of the rows is an equality and the other one is an inequality. Here, we discuss the three cases together since all of them can result in either of the following two conclusions: the two constraints conflict, or one of the constraints is removed from the problem.

   To explain the case that one of the constraints is eliminated, we assume that row i is kept and row k is removed. From the solver we get one dual variable value y_i^*. The problem is how to decide the value of y_i and y_k for the original problem.

   To find the answer, we analyze the following systems:

   Assume the primal problem is given as:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b, \\ & l \leq x \leq u. \end{aligned}$$

   By introducing slack variables, this can be rewritten as:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax - \hat{x} = b, \\ & l \leq x \leq u, \\ & \hat{x} \geq 0. \end{aligned}$$

The dual problem is:

$$\begin{aligned} \max \quad & b^T y + l^T d - u^T w \\ \text{s.t.} \quad & A^T y + d - w + s = c, \\ & -Iy + \overline{s} = 0, \\ & y \text{ is free}, d \geq 0, w \geq 0. \end{aligned}$$

From the Weak Duality Theorem (see Theorem 1.2.1), we know that $x^T s = 0$ and $\hat{x}^T \overline{s} = 0$. The optimal values of x and $\hat{x}$ are obtained directly from the solver. Depending on the value of $\hat{x}$, there are two cases as follows:

(a) If $\hat{x}_i = 0$, then by the complementarity condition $\hat{x}_i \overline{s_i} = 0$, we conclude that $\overline{s_i}$ needs not to be zero. Moreover, from $-Iy_i + \overline{s_i} = 0$, we have that $y_i = \overline{s_i}$ is a nonzero value.

(b) If $\hat{x}_i \neq 0$, then by the complementarity condition $\hat{x}_i \overline{s_i} = 0$, we conclude that $\overline{s_i}$ must be zero. Moreover, from $-Iy_i + \overline{s_i} = 0$, we have that $y_i = \overline{s_i}$ is a zero value.

Thus, we can decide the value of y_i and y_k if row i is kept and row k is eliminated:

(a) If $\hat{x_i} = 0$, then $y_i = y_i^*$ and $y_k = 0$.

(b) If $\hat{x_i} \neq 0$, then $y_i = 0$ and $y_k = y_i^*$.

3.3.7 Duplicate Columns

Recall our discussion in Section 2.4.8, if the columns j and k are duplicate columns, then $a_j = \lambda a_k$ for some $\lambda \in R$. There are two cases in our analysis depending on whether the corresponding cost coefficients of the two columns are in proportion, i.e., $c_j = \lambda c_k$.

1. If the cost coefficients are in proportion, i.e., $c_j = \lambda c_k$, then the variables x_j and x_k can be replaced by a new variable $\hat{x}_k$ and $\hat{x}_k = \lambda x_j + x_k$. Then x_j and x_k are removed from the problem and they do not appear in matrix A nor in vector c. The value of $\hat{x}_k$ can be obtained from the solver. The values of x_j and x_k are decided by (2.4.25). Any value that satisfies Equation (2.4.25) is a candidate for being the optimal values of x_j and x_k. Another thing to be considered is that the optimal values of x_j and x_k should also satisfy the complementary condition, i.e., $x_j(c_j - \sum_{p=1}^{m} y_p a_{pj}) = x_j s_j = 0$ and $x_k(c_k - \sum_{p=1}^{m} y_p a_{pk}) = x_k s_k = 0$. For instance, if $s_j = 0$, then x_j needs not to be zero. Any solution satisfying Equation (2.4.25) can be set to the value of x_j. If $s_j \neq 0$, then x_j must be zero. If $l_j \leq 0$, then x_j is set to zero. Otherwise, the problem is infeasible. Similar analysis can be given to the value of x_k.

2. If the cost coefficients are not in proportion, i.e., $c_j \neq \lambda c_k$, then there are two special cases that may lead to the conclusion that the dual problem (2.3.6) is infeasible, or to the conclusion that x_j can be fixed to either its lower bound or its upper bound (see Section 2.4.8). If the dual problem is detected infeasible, then we directly get the result. When x_j is fixed, the operation in postprocessing is the same as for a fixed variable (see Section 3.3.3).

3.3.8 Redundant Constraint

If row i is a redundant row, then it can be removed in preprocessing. The corresponding optimal value of y_i is set to zero.

3.3.9 Forcing Constraint

If row i is a forcing row, then it can be removed and all the participating variables x_j in this row are fixed. The values of those variables x_j are recorded in the stack and as fixed variables, they are substituted out of the problem. Those values can be retrieved from the stack in postprocessing.

For each participating variable x_j, its dual slack variable s_j must be nonnegative:

$$s_j = c_j - \sum_{k=1,k\neq i}^{m} y_k a_{kj} - y_i a_{ij} \geq 0 \text{ and } a_{ij} \neq 0,$$

thus, the value of y_i is be chosen such that all the corresponding dual slack variables s_j are nonnegative if problem (2.3.6) is feasible.

For simplicity, if all $a_{ij} > 0$ in constraint i, then the value of y_i is given by

$$y_i \leq \min_{1\leq j\leq n} \frac{c_j - \sum_{k=1,k\neq i}^{m} y_k a_{kj}}{a_{ij}}.$$

If all $a_{ij} < 0$ in constraint i, then the value of y_i is given by

$$y_i \geq \max_{1\leq j\leq n} \frac{c_j - \sum_{k=1,k\neq i}^{m} y_k a_{kj}}{a_{ij}}.$$

If there are some $a_{ij} > 0$ and some $a_{ij} < 0$, then the value of y_i is given by

$$\max_{1\leq j\leq n, a_{ij}<0} \frac{c_j - \sum_{k=1,k\neq i}^{m} y_k a_{kj}}{a_{ij}} \leq y_i \leq \min_{1\leq j\leq n, a_{ij}>0} \frac{c_j - \sum_{k=1,k\neq i}^{m} y_k a_{kj}}{a_{ij}}. \tag{3.3.1}$$

If there is a y_i satisfying condition (3.3.1), then the dual variable y_i is found. Otherwise, the problem (2.3.6) is infeasible.

3.3.10 Tighten Variable Bounds

If variable bounds are tightened in row i, then the solutions of problem (2.3.5) and (2.3.6) are not affected. It is possible that some variables are fixed. In this case, their values are recovered from the stack that is recorded in preprocessing. The operation is the same as fixed variables (see Section 3.3.3).

3.3.11 Tighten Dual Variable Bounds

If dual variable bounds are tightened, then the solutions of problem (2.3.5) and (2.3.6) are not affected. It is possible that some dual variables are fixed. In this case, their values are recovered from the stack.

3.3.12 Dominated Variables

If variable x_j is a dominated variable, then it can lead to either problem (2.3.6) is infeasible or x_j fixed to either l_j or u_j. If x_j can be fixed, then the value of x_j becomes a fixed variable and its optimal value is recovered from the stack (see Section 3.3.3).

3.3.13 Make A Sparser

In the heuristic algorithm [13] proposed by Gondzio, an elementary Gaussian operation is applied to row i and row k in matrix A:

$$\hat{a}_{kj} = a_{kj} + \lambda a_{ij},\ j = 1, \ldots, n,$$

that can also be expressed as $\hat{A} = MA$, where M is a nonsingular matrix. Consequently, the constraint in problem (2.3.6) $A^T y + v - w = c$ becomes

$$A^T M^T y + v - w = c,$$

thus, the dual optimal solution of the original problem is $M^T y$.

Since there are only row operations in this heuristic algorithm, the optimal solution of problem (2.3.5) is not affected.

3.4 Recover the Optimal Solution of the Original Problem

Recall that in Section 2.2, in order to transform the original problem (1.1.4) into the standard form (2.1.1), we performed the following operations:

1. Introduce slack variables to inequality constraints.
2. Split free variables.
3. Shift the lower bound l if it is a nonzero vector.

To recover the solution of the original problem (1.1.4), correspondingly, we need to execute the following operations:

1. Remove slack variables to get back the inequality constraints.
2. For free variables, x_j^+ and x_j^- are known, then we have

$$x_j = x_j^+ - x_j^-.$$

3. If the lower bound l is nonzero, then given the optimal value of $\hat{x}$, the optimal value of x is recovered by

$$x = \hat{x} + l.$$

We have completed the discussion on postprocessing operations. We continue our discussions with implementation issues in Chapter 4.

Chapter 4

Implementation Issues

The implementation consists of three subroutines: MPS reader, preprocessing and postprocessing. We use the McMaster Interior Point Method (McIPM) solver as our solver that connects the subroutines of preprocessing and postprocessing. Thus, we will discuss the implementation issues of the three subroutines and the data links among the subroutines of MPS reader, preprocessing, McIPM solver and postprocessing.

4.1 Program Structure

A large scale LO problem is solved in a standard procedure as follows: first the problem given in an MPS file is read by the subroutine MPS reader, then it is standardized and simplified by the subroutine preprocessing. Next the LO problem is solved by the solver. Here, we use the McIPM solver that was implemented by X. Zhu. The interested reader can find the detail information in her Master thesis [28] and in the papers [27] [29]. Finally, the solution obtained from the solver is sent to the postprocessing subroutine that "undo" the changes in the LO problem. The following figure

shows the flow chart of the procedure:

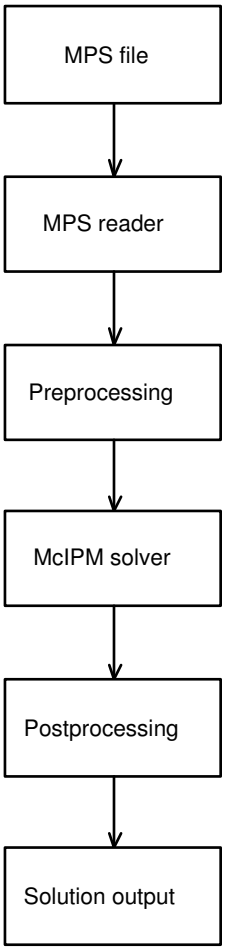

Figure 4.1: The Structure of McIPM

In the sequel, we focus our discussions on the implementation issues of our MPS reader, preprocessing and postprocessing and the communication among MPS reader, preprocessing, postprocessing and the McIPM solver.

4.2 Computational Environment

The subroutines of MPS reader, preprocessing and postprocessing are developed in the language of IBM C on an IBM RS-6000-44p-270 workstation with four processors under the operating system of the AIX 4.3. The McIPM solver is implemented in IBM C too and it uses MATLAB [1] 6.1 as the computing environment because it is suitable for numerical computation and visualization. Therefore, the subroutines of preprocessing and postprocessing are called from MATLAB.

MATLAB has an external function MEX, meaning MATLAB executable files, that can dynamically link the subroutines compiled from C with MATLAB. With MEX file, the C subroutines of preprocessing and postprocessing are called directly in MATLAB as if they are embedded. MATLAB supports the use of a variety of compilers to build MEX file. In our case, MEX file is generated from C subroutines using the IBM c/c++ complier xlc.

In addition, data files are in MAT, TXT, DAT file formats that are used to communicate date between C and the MATLAB environment.

4.3 The MPS Reader

The MPS reader is aimed to transform an external representation of an LO problem into an internal representation that can present the LO problem to the preprocessing subroutine. The main issue is how to link the names, e.g., row name, column name, bound name, with their respective indices, and then set up the structures for $A, \overline{b}, \underline{b}, c, l, u$ and the LO problem name. Thus, the process is non-numerical.

[1] MATLAB® is the product of MathWorks Inc.. It stands for "Matrix Laboratory".

Processing an MPS file is a complicated procedure. In our implementation, there are three essential tasks in reading an MPS file:

1. Recognize the section identifiers to determine what information is being read.
2. In the ROWS and COLUMNS sections, set up a hashing table that gives the correspondence of the name and its index, i.e., to link a row name with the corresponding index i and link a column name with its index j. Then, using the hashing table, find the corresponding row index i with the row name in RHS and RANGES sections and find the corresponding column index j with the column name in BOUNDS section.
3. Write the corresponding part into the data structures of $A, \overline{b}, \underline{b}, c, l$, and u. In the process of reading an MPS file, the corresponding information are stored in linked list temporarily. We choose the linked list as storage media because the size of an LO problem is unknown. Once the LO problem is finished reading, the matrix information is transferred the corresponding data structure: matrix A is stored column-wise in sparse form. The vectors $\underline{b}, \overline{b}, c, l$ and u are stored in full size.

It is essential to keep reliability and efficiency of the reading process.

1. To ensure the reliability in our implementation:
   (a) We always check for errors, inconsistence and contradictions. When there are errors or contradictions in the process of reading an MPS file, the program should not crash. For instance, if there are more than two definitions for one variable bound type, then a warning is given and the latest definition is used while the others are ignored. The warning can help to correct

the problem. If there is fatal errors, e.g., there is no required section found in an MPS file, then a corresponding message is given and the program ends.

(b) Moreover, duplicate information must be checked and eliminated in each section to avoid information redundancy. For instance, in ROWS section, it is necessary to check whether duplicate row names are defined. Further, to check whether a row type is legal or not.

2. To improve efficiency:

(a) We use the hashing table to link the names and their indices. The hashing table can quickly link the row name with its index i, and the column name with its index j. It can save considerable time especially for those huge problems with over ten thousand rows and/or columns.

(b) For each LO problem, since its size is unknown, the information obtained from the ROWS and COLUMNS section, including the information of c and A, is written into the linked list first. Once finished reading the ROWS section, we can know the row size. Based on the row size, we can allocate array space for b and $\mid \underline{b} - \overline{b} \mid$ that store the information directly from the RHS and RANGES sections. When finished reading the COLUMNS section, the column size is known too. Based on the column size, we can allocate space for vectors l and u that store the information directly from the BOUNDS section.

Writing A needs a little calculation. We can directly write A from the linked list, but because writing A is time consuming and the slack variables need to be added to the constraints later, we choose to allocate enough space at one time. Therefore, we calculate the total number of

variables, e.g. the sum of the number of the original variables and the slack variables. Then we calculate the corresponding total space needed for A. After allocating space, the information from the linked list is written into the corresponding arrays. In this way, no more space allocation is needed when the slack variables are written into A.

Next, we introduce the matrix storage.

4.4 Matrix Storage

Matrix storage is very important for sparse matrix. Due to sparsity, the sparse matrix $A^{m\times n}$ is not stored in full size, e.g., a multi-dimension array with size of $m \times n$. To improve efficiency, only the nonzero elements in A are stored. Moreover, in the process of preprocessing, matrix A is required to be accessed not only by column-wise, but also by row-wise. For instance, when a forcing row is detected in preprocessing procedure, every variable in this row needs to be substituted out of the problem. First, the indices of those variable j are needed. To retrieve such information efficiently, we need the row-wise information of matrix A. Once the variable index is known, all the rows in which the variable participates are needed for performing substitution. The corresponding values of the RHS change as well. This in turn requires the column-wise information. Thus, the column-wise and row-wise storage of matrix A are both necessary in our implementation.

We store matrix A column-wise. There are three arrays to provide the following information:

- Array JA: the start position of each column;

- Array AR: the value of each nonzero element;
- Array IA: the row index of each nonzero element.

The starting position of each column is recorded in array JA. We set $JA[0] = 1$. If there are n columns in matrix A, then the size of array JA is $n + 1$. Then the beginning position of column j is $JA[j]$ and its ending position is $JA[j+1]-1$. There are $JA[i + 1] - JA[i]$ nonzero elements in column j. The values and row indices of nonzero elements are stored consecutively in the arrays AR and IA, respectively. If the row is physically the first row, then the corresponding value in IA is 1. The size of AR and IA equals to the number of nonzero elements.

The following example may help to understand column-wise mode of matrix storage:

Example:

If $A = \begin{pmatrix} 4 & 0 & 0 & 5 \\ 0 & 7 & 9 & 0 \\ 2 & 0 & 4 & 6 \end{pmatrix}$, then we have

$JA = [1\ 3\ 4\ 6\ 8]$,

$AR = [4\ 2\ 7\ 9\ 4\ 5\ 6]$ and

$IA = [1\ 3\ 2\ 2\ 3\ 1\ 3]$.

The column-wise information can be directly obtained from the storage of A. For each column j, its starting index $AJ[j]$ and its ending index $AJ[j+1]-1$ are known, thus the nonzero entries in column j are obtained by searching the array between $AJ[j + 1] - 1$ and $AJ[j]$.

To get the row-wise information of A, we make use of matrix A^T. Like the storage

of A, matrix A^T is stored by column-wise as well. The column-wise information of A^T is exactly the row-wise information of A.

Example:

If $A^T = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 7 & 0 \\ 0 & 9 & 4 \\ 5 & 0 & 6 \end{pmatrix}$, then we have

$JA' = [1\ 3\ 5\ 8]$,

$AR' = [4\ 5\ 7\ 9\ 2\ 4\ 6]$ and

$IA' = [1\ 4\ 2\ 3\ 1\ 3\ 4]$.

For row i in A, there are $JA'[i+1] - JA'[i]$ elements. The corresponding column index and coefficient can be obtained by searching the arrays AR' and IA' between the index $JA'[i+1] - 1$ and $JA'[i]$.

In our program, the information of A is first written from the linked list directly obtained from the MPS file. Then the slack variables are added to the constraints. A is changed too if there are some fixed variables and nonzero lower bound variables in the BOUNDS section. After those fixed variables are removed and nonzero lower bounds are shifted to zero, A^T are written out and the process of preprocessing begins.

Here, we do not split the free variables at this moment because we want to keep them in the LO problem during preprocessing procedure. Therefore, when using the preprocessing techniques, there are three types of variables kept in the LO problem: $0 \leq x_j < +\infty$, $0 \leq x_j \leq u_j$ or x_j is free variable.

Further, to describe a complete LO problem, another four vectors are needed: the RHS vector b, the cost coefficient vector c, the lower bound vector l and the upper

bound vector u. For those vectors, even if they are sparse vectors, they are still stored in full size, i.e., the size b is m, the size of vectors c, l and u is n.

4.5 Preprocessing and Postprocessing

Preprocessing and postprocessing are a pair of operations that changes the LO problem and retrieves the problem changes, respectively. It is essential that the operations in preprocessing are recorded in a proper order. Postprocessing performs the "undo" operations in the reverse order.

Recall the preprocessing techniques in Section 2.1, we implement all the basic techniques and part of advanced techniques as follows: check infeasible variables, remove the fixed variable, remove the empty row and empty column, find the singleton row and singleton column, remove the duplicate rows and duplicated columns, find the forcing rows and redundant rows and tighten the bounds of the variables. There are four more complicated techniques left: tighten dual variables bounds, find dominated variables, make A sparser and make A to have full rank. The first two techniques are the operations dealing with the dual problem. The last two techniques need more computational effort. Due to the time limits, we have not implemented them.

In our program, matrix A is obtained from the linked list directly from the MPS file. Then the slack variables are added to the constraints, the fixed variables are removed from the problem and the nonzero lower bounds are shifted to zero. Here, we do not split free variables, leaving it until the last moment of the preprocessing. The flow of the preprocessing procedure is drawn as Figure 4.2.

Figure 4.2 describes the "normal" procedures when the LO problem has optimal solution. It is possible to find out that the LO problem is infeasible or unbounded.

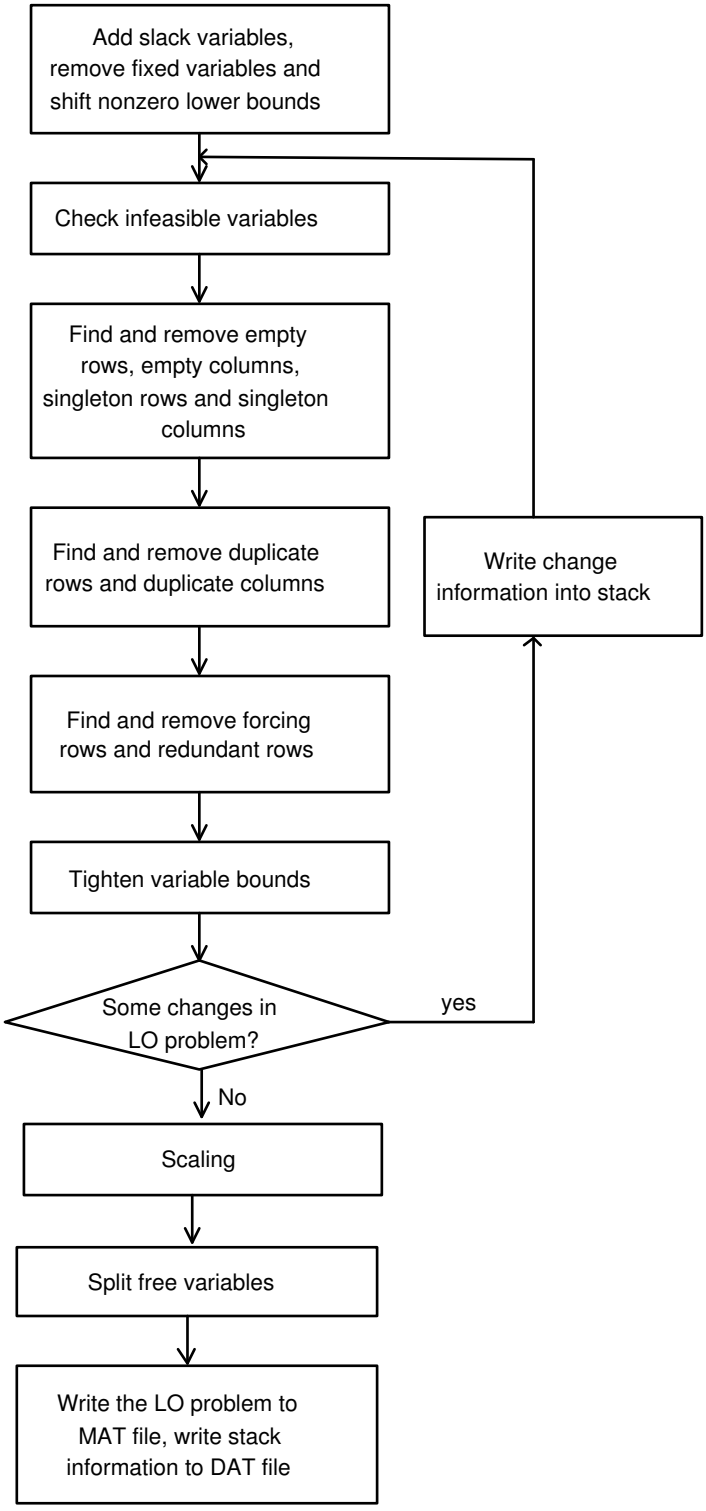

Figure 4.2: The Structure of Preprocessing

In such cases, the program just jumps out of the loop and gives the result directly.

After matrix A is added the slack variables, shifted the nonzero lower bounds, we check the infeasible variables. They can be easily checked by comparing the lower bound and upper bound of each variable. If the bounds conflict, then the LO problem is infeasible. Otherwise, A^T stored by column-wise is written and the LO problem is sent to the preprocessing procedures.

In our program, there are two arrays recording the nonzero entry numbers in each row and column, respectively. Using them, some "characterized" rows and columns can be easily found. Once the nonzero elements are changed in some rows and columns, the value in the two arrays changes too. For instance, to find the empty row and the empty column, we just find the rows and columns whose nonzero entry number is zero. Once the empty row and empty column are found, the corresponding operations are simple. Readers can refer to Sections 2.4.2 and 2.4.3.

The singleton row and singleton column are easily found by searching the rows and columns whose nonzero entry number are equal to one. The detail operation to singleton row refers to Section 2.4.5. Once the singleton column is found, we need to check whether the variable is free variable or implied free variable. If yes, then the variable can be substituted out of the LO problem, then the corresponding information is recorded. The detail operation refers to Section 2.4.6.

The operation to duplicate rows and duplicate columns are similar. We give the example how to find the duplicate row for row i. The preliminary is that row i has at least two nonzero entries. Then we find the shortest column k that has nonzero entry in row i and has the least nonzero entry number. Our target is to search each row except row i that has nonzero entry in column k. If there is row p that has the same nonzero number as row i, then row p is the candidate row. Further, we

investigate whether the position of each nonzero entry in row p and row i are the same and whether their value are in proportion. If yes, the duplicate rows are found. The corresponding operations refers to duplicate rows and duplicate columns refer to Section 2.4.7 and Section 2.4.8.

The procedure to find a forcing row need some calculation. Because there are only three types of variables left in the LO problem: $0 \leq x_j < +\infty$, $0 \leq x_j \leq u_j$ or x_j is free variable, the lower bound $\underline{b_i}$ and the upper bound $\overline{b_i}$ are easy to calculate. Further, it is also easy to find the infinity of $\underline{b_i}$ and $\overline{b_i}$ Once $\underline{b_i}$ and $\overline{b_i}$ are found infinity, the calculation stops and the search moves to the next row. The technique of tightening bounds combines with the calculation. If tightened bound can be found, then it compares with the original bound. If the new bound is tighter than the original one, then it substitutes the original one. If the lower bound are tightened, then we usually shift it to zero at once and the corresponding RHS is changed too. The corresponding operations refers to Section 2.5.1 and Section 2.5.2.

Notice in Figure 4.2, if there is a change caused by a preprocessing technique in the loop, the change information is written into the stack. Because one change or elimination may cause more changes or elimination, the loop continues until there is no more change in the LO problem. Then, the LO problem is scaled. Recall Section 2.8, there are several scaling methods. We chose to use the special algorithm proposed by Curtis and Reid [17]. The procedure seems to be complicated, but its implementation is easy. We use equation (2.8.30) to measure the scaling effect and the algorithm stops until equation (2.8.40) is satisfied. Once the row and column scaling factors are found, they are recorded in two arrays, respectively. The two arrays are used to recover the scaling in postprocessing.

The whole corresponding procedures in postprocessing is as follows:

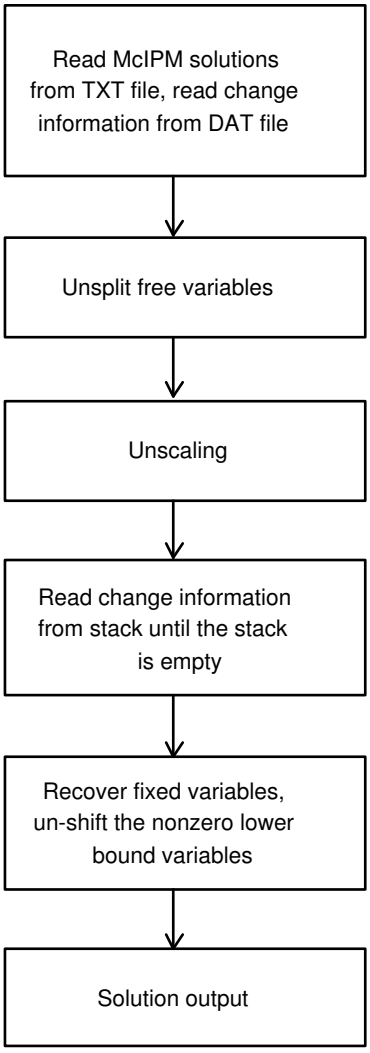

Figure 4.3: The Structure of Postprocessing

In our implementation, the main problems are what kind of data structures are used to store the information of change in the LO problem and how to retrieve those information in the given order.

The first problem is how to store the change information. There are various preprocessing techniques and different techniques require different information. For instance, an empty row is very simple structure, it only needs the index of the row that is used to set the corresponding dual variable to zero. We define the data structure as:

```
typedef struct {
    int RowIndex;
} EmptyRow,
```

where *RowIndex* is the empty row index i.

An empty column needs more information, i.e., the column index, the variable value if it is fixed. Once the variable is fixed, it is marked as "DELETED" that means it is not included in further operations. Correspondingly, the objective value will have a shift that needs to be recorded as well. The data structure is as follows:

```
typedef struct {
    int ColIndex;
    double xValue;
    double cValue;
    double cshiftObj;
} EmptyColumn,
```

where *ColIndex* is the column index j, *xValue* is the value of fixed variable x_j, *cValue* is the coefficient c_j and *cshiftObj* is the shift of the objective value $x_j c_j$.

However, more preprocessing techniques need more complicated information. The most complicated data structure is the forcing row. All the information related to the row and the information related to all the participating variables in this row need to be recorded. For instance, if row i is detected to be a forcing row, then all the variables x_j in this row are fixed either to its lower bound or to its upper bound and they are removed from the problem. Before removal, the column index j, the fixed value x_j and its coefficient in row i, i.e., a_{ij} needed to be recorded. Moreover, to recover the dual variable y_i in postprocessing, for each variable x_j in row i, all the participating row index p and the related coefficients a_{pj} in those rows are recorded as well. Therefore, the data structure for a forcing row is as follows:

First, we define a structure SparseVector for a vector because there are so many row and column vectors needed for a forcing row. The structure of SparseVector includes the information of the vector size, the index and the value of its elements.

```
typedef struct{
    int size;
    int *Index;
    double *Value;
} SparseVector,
```

then we make use structure *SparseVector* to define the structure for a forcing row:

```
typedef struct {
    int RowIndex;
    int ForcingType;
    double bValue;
    double cshiftObj;
    SparseVector *xValues;
```

```
    SparseVector *cValues;
    SparseVector *DeletedRow;
    SparseVector **DeletedColumns;
} ForcingRow,
```

the pointer *DeletedRow* records the information about the forcing row itself, i.e., the number of the variables, the column index j and its coefficient a_{ij} in row i. The double pointer *DeletedColumns* stores the information about the deleted columns for the fixed variable x_j. Because each variable may participate in other rows, once it is removed, the related value of the RHS need to be re-calculated and all the coefficients in those rows are needed as well in order to recover the solution of the dual variable. The pointer *xValues* records the information about the fixed variables, i.e., the number of the fixed variables, the column index j and the values x_j. The pointer *cValues* is similar to *xValues*, the difference is the information stored in *Value* is the coefficient c_j in the objective function.

In this way, we set up the data structure for each preprocessing technique. The next problem is how to organize those data structures and retrieve them in a certain order. The stack is a popular concept in computer science. It is specially useful when the information needs to be obtained in a certain order, normally "last come and first go". Therefore, a stack is used to control the order of the operations. In our data structure design, we refer to the design of the software package PCx. Because the stack is composed of single changes in preprocessing. Each change is exactly one of the preprocessing techniques in Section 2.1. As an example technique, let us assume we use the preprocessing techniques of empty row, empty column and forcing row, the data structure of *SingleChange* is defined as follows:

```
typedef struct{
```

```
    int ChangeType;
    EmptyRow *pEmptyRow;
    EmptyColumn *pEmptyColumn;
    ForcingRow *pForcingRow;
} SingleChange,
```

then we the stack data structure of *PresolveChanges* is composed of every *SingleChange* as follows:

```
typedef struct {
    int size;
    SingleChange **StackOfChanges;
} PresolveChanges,
```

where the number of *size* is the number of the changes in stack.

Therefore, in preprocessing, when a technique used to change somehow the LO problem, it is first written into the stack and then store the information to the corresponding data structure according to the technique. The last procedure in preprocessing is to write the stack information into a DAT files. Correspondingly, in postprocessing, the first procedure is to read the stack information from the DAT files, then "undo" each changes in the stack to recover the solution for the original problem.

4.6 Data Communication

In our implementation, the running environment is under MATLAB 6.0. The subroutines of preprocessing and postprocessing are developed in C and their executive

files are compiled in C as well. Referred to Section 4.2, the executive files are called in MATLAB via MEX file. The difficulty is how to pass the information obtained from preprocessing to the McIPM solver; from the McIPM solver to postprocessing; and from preprocessing to postprocessing.

Our solution is as follows:

1. **Preprocessing - McIPM solver:** Using the MAT file format, the matrix stored in C can be read by MATLAB.

2. **McIPM solver - Postprocessing:** Using a TXT file, the solution obtained from the McIPM solver in MATLAB can be read in our C code.

3. **Preprocessing - Postprocessing:** All the changes in preprocessing are recorded in the stack in order to be retrieved during the postprocessing subroutine. In preprocessing, the stack information are stored in the DAT file, where the information about changes can be retrieved in postprocessing.

We will describe the three files one by one.

4.6.1 MAT-Format

The data of the LO problems in MAT-format includes seven quantities:

$$A, b, c, lbounds, ubounds, BIG, NAME$$

representing the following LO problem:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & lbounds \leq x \leq ubounds, \end{aligned} \tag{4.6.1}$$

$NAME$ is the name of the LO problem, BIG is a very large number. We define $BIG = 10^{32}$. It is used when variable x_i has infinity upper bound or lower bound, i.e., $ubounds_i = BIG, lbounds_i = -BIG$. Matrix A is stored column-wise. There is a little difference from what we discussed in Section 4.4. In the MAT file, the starting point of the first row is defined as 0, and the physically first row is defined as 0 too.

To understand better, we give the example again to compare this difference in storage index:

Example:

If $A = \begin{pmatrix} 4 & 0 & 0 & 5 \\ 0 & 7 & 9 & 0 \\ 2 & 0 & 4 & 6 \end{pmatrix}$, then in MAT file we have

$JA = [0\ 2\ 3\ 5\ 7]$,

$AR = [4\ 2\ 7\ 9\ 4\ 5\ 6]$ and

$IA = [0\ 2\ 1\ 1\ 2\ 0\ 2]$.

In this way, all the information of the LO problem is stored in a MAT file and sent to the McIPM solver. By reading the MAT file, the solver receives all the necessary data of an LO problem.

4.6.2 TXT File

The data communication between the McIPM solver and the postprocessing subroutines is simple. Because the information needed by postprocessing is just the solution of the LO problem, the solution is saved in a TXT file that can be easily read by the postprocessing subroutine.

4.6.3 DAT File

The data communication between preprocessing and postprocessing is different from the above. As stated before, the information of the changes in the LO problem are stored in the stacks in preprocessing. This information is stored in C, but they can not be obtained in MATLAB. Therefore, we need some "buffer" to store the changes information temporarily and get them to postprocessing. We make use of a DAT file to store those information in preprocessing and retrieve them in postprocessing. In this way, postprocessing recalls those changes to get the solution of the original LO problem.

Each preprocessing technique has its own DAT file. The main reason is that different techniques have different requirement for data information, thus their data structure are quite different (see Section 4.5). However, no matter the data structure is, its storage and retrieval method are the same. Generally, to save the information to a DAT file, the data structure is stored first. If there is an array in the data structure, its elements need to be stored as well. Vice Versa, to retrieve the information from a DAT file, all the data structure and arrays need to be allocate space first, then read the information in the same order as what we did when saving the information to a DAT file.

UP to know, we have finished discussing the implementation issues. In the next chapter, we will go to the test procedure and give out the testing result.

Chapter 5

Computational Results

In this chapter, we first describe the testing problem set and present our computational results. Further, we compare our result with two commercial software packages CPLEX [1] *and LIPSOL* [2]*.*

5.1 Testing Problems and Results

We use the linear problems in the NETLIB set as our testing problems. The NETLIB set is a collection of practical linear problems from various sources. The problems are presented in MPS format. It consists of three testing sets:

- Netlib standard testing set, composed of 95 linear problems. All the problems have optimal solutions.
- Netlib infeasible testing set, composed of 28 linear problems. None of the problems have an optimal solution.

[1] CPLEX is the product of ILOG Inc..

[2] LIPSOL is a Matlab-based package for solving linear programs by interior-Point methods. It stands for "Linear-programming Interior-Point SOLvers".

- Netlib Kennington testing set, composed of 16 very large scale problems chosen from military aircraft applications. All the problems have optimal solutions.

Normally, the size of an LO problem is shown in the following ways: the number of rows counts the objective function, but exclude the other free rows; the number of columns excludes both the slack variables and the added split free variables; the number of nonzero elements counts the nonzero elements in the objective function, but exclude both the slack variables and the added split free variables. To find such information of an LO problem in the Netlib set, readers can refer to Appendix B in [28].

However, in our preprocessing result comparison, we do not use the same information. The reason is that such information can not present clearly the effect of preprocessing on an LO problem, thus we design the tables as follows: *Problem* is the problem name. *Row* is the row size of the matrix A, excluding all the free rows. *Col* is the column size of the matrix A including the slack variables and the added split free variables. *Nnz* is the number of nonzero elements in the matrix A including the slack variables and the split free variables as well. In this way, we show the size of a whole LO problem. Then, we list the reduced sizes in the next three columns. *Rrow* is the reduced row size, *Rcol* and *Rnnz* are the reduced column size and the nonzero element numbers, respectively. Both *Rcol* and *Rnnz* include the slack variables and split free variables. In this way, by comparing the pair of data, the effect of preprocessing is shown clearly. Further, *Tpre*, *Tpost* and *Tsol* mean the time spent in preprocessing, in postprocessing, and in solver, respectively. *Iter* means the iteration number.

The results of the Netlib testing sets are shown from Tables 5.1 to 5.5 as follows.

Table 5.1: The Netlib Standard Problem Set (I)

Problem	Row	Col	Nnz	Rrow	Rcol	Rnnz	Tpre	Tpost	Iter	Tsol
25fv47	821	1876	10705	788	1843	10538	0.11	0.01	30	4.84
80bau3b	2262	12061	23264	2183	11163	21908	0.11	0.02	41	29.67
adlittle	56	138	424	55	137	417	0.08	0.01	14	0.33
afiro	27	51	102	27	51	102	0.07	0.01	10	0.18
agg	488	615	2862	468	595	2780	0.09	0.01	22	1.87
agg2	516	758	4740	516	758	4740	0.10	0.01	21	2.16
agg3	516	758	4756	516	758	4756	0.10	0.01	22	2.29
bandm	305	472	2494	251	417	1982	0.08	0.02	18	0.81
beaconfd	173	295	3408	107	218	1888	0.09	0.02	11	0.38
blend	74	114	522	71	111	477	0.11	0.01	10	0.27
bnl1	643	1586	5532	624	1564	5498	0.10	0.01	33	3.17
bnl2	2324	4486	14996	1998	4078	14259	0.16	0.02	36	10.08
boeing1	351	726	3827	344	719	3352	0.08	0.01	24	2.17
boeing2	166	305	1358	125	264	922	0.09	0.02	15	0.71
bore3d	233	334	1448	124	217	1009	0.10	0.02	18	0.72
brandy	220	303	2202	136	246	1964	0.04	0.00	19	0.70
capri	271	496	1965	241	436	1528	0.09	0.01	19	1.15
cycle	1903	3378	21248	1496	2999	16669	0.13	0.01	39	11.39
czprob	929	3562	10708	719	3123	6209	0.10	0.01	36	5.49
d2q06c	2171	5831	33081	2132	5728	31965	0.12	0.01	52	26.76
d6cube	415	6184	37704	403	5443	34233	0.12	0.01	19	6.07
degen2	444	757	4201	444	757	4201	0.09	0.01	12	1.10
degen3	1503	2604	25432	1503	2604	25432	0.17	0.00	14	8.17
dfl001	6071	12230	35632	5984	12143	35338	0.15	0.01	45	263.27
e226	223	472	2768	214	463	2702	0.09	0.01	21	0.99
etamacro	400	816	2537	334	669	1995	0.08	0.02	27	1.96
fffff800	524	1028	6391	322	826	5164	0.08	0.00	32	2.39
finnis	497	1064	2760	456	966	2414	0.07	0.01	25	2.03
fit1d	24	1049	13427	24	1049	13427	0.11	0.01	19	2.70
fit1p	627	1677	9868	627	1677	9868	0.11	0.01	19	9.85

Table 5.2: The Netlib Standard Problem Set (II)

Problem	Row	Col	Nnz	Rrow	Rcol	Rnnz	Tpre	Tpost	Iter	Tsol
fit2d	25	10524	129042	25	10524	129042	0.15	0.01	23	31.69
fit2p	3000	13525	50284	3000	13525	50284	0.16	0.00	21	37.53
forplan	161	492	4634	122	450	4477	0.08	0.00	29	1.74
ganges	1309	1706	6937	1113	1510	6537	0.09	0.02	20	3.07
gfrd-pnc	616	1160	2445	590	1134	2393	0.07	0.02	18	1.65
greenbea	2392	5598	31070	2297	5270	29777	0.13	0.00	51	26.67
greenbeb	2393	5405	31499	2296	5258	29477	0.16	0.00	51	26.29
grow15	300	645	5620	300	645	5620	0.10	0.01	18	1.90
grow22	440	946	8252	440	946	8252	0.07	0.01	19	2.75
grow7	140	301	2612	140	301	2612	0.08	0.01	18	1.11
israel	174	316	2443	174	316	2443	0.06	0.01	22	1.71
kb2	43	68	313	43	68	313	0.11	0.01	17	0.47
lotfi	153	366	1136	133	346	867	0.09	0.01	17	0.53
maros	846	1966	10137	789	1860	9489	0.11	0.01	30	3.91
maros-r7	3136	9408	144848	2152	7440	100486	0.24	0.01	16	30.13
modszk1	687	1622	3170	665	1599	3065	0.10	0.01	30	2.56
nesm	662	3105	13470	654	2922	13244	0.12	0.01	36	9.48
perold	625	1594	7317	613	1500	6963	0.11	0.02	40	5.07
pilot	1441	4860	44375	1423	4639	42242	0.17	0.01	52	44.96
pilotja	940	2355	16216	858	1972	12900	0.12	0.03	42	8.98
pilotwe	722	3008	9801	718	2925	9494	0.10	0.00	44	8.32
pilot4	410	1211	7342	400	1117	7106	0.10	0.01	39	4.66
pilot87	2030	6680	74949	1990	6411	72258	0.26	0.01	85	156.98
pilotnov	975	2446	13331	864	2149	12037	0.08	0.01	27	5.60
recipe	91	204	687	71	137	476	0.06	0.00	12	0.46
sc105	105	163	340	104	162	339	0.10	0.02	12	0.29
sc205	205	317	665	203	315	663	0.07	0.00	12	0.36
sc50a	50	78	160	49	77	159	0.08	0.01	11	0.23
sc50b	50	78	148	48	76	146	0.07	0.00	10	0.21
scagr25	471	671	1725	469	669	1715	0.10	0.02	16	0.82

Table 5.3: The Netlib Standard Problem Set (III)

Problem	Row	Col	Nnz	Rrow	Rcol	Rnnz	Tpre	Tpost	Iter	Tsol
scagr7	129	185	465	127	183	455	0.07	0.01	13	0.35
scfxm1	330	600	2732	311	581	2497	0.08	0.02	23	1.18
scfxm2	660	1200	5469	622	1162	4999	0.07	0.01	27	2.33
scfxm3	990	1800	8206	933	1743	7501	0.11	0.01	27	3.31
scorpion	388	466	1534	361	436	1415	0.08	0.01	14	0.58
scrs8	490	1275	3288	430	1215	3067	0.09	0.01	26	1.85
scsd1	77	760	2388	77	760	2388	0.09	0.00	10	0.46
scsd6	147	1350	4316	147	1350	4316	0.08	0.02	12	0.80
scsd8	397	2750	8584	397	2750	8584	0.06	0.01	10	1.25
sctap1	300	660	1872	284	644	1802	0.09	0.01	19	0.87
sctap2	1090	2500	7334	1033	2443	7052	0.11	0.01	19	2.52
sctap3	1480	3340	9734	1408	3268	9383	0.11	0.02	20	3.84
seba	515	1036	4360	514	1033	4342	0.06	0.03	24	4.29
share1b	117	253	1179	112	248	1148	0.09	0.00	28	0.79
share2b	96	162	777	96	162	777	0.09	0.01	12	0.33
shell	536	1777	3558	487	1451	2906	0.09	0.00	24	2.44
ship04l	402	2166	6380	323	2104	4734	0.10	0.03	19	1.65
ship04s	402	1506	4400	235	1356	3044	0.07	0.00	18	1.13
ship08l	778	4363	12882	630	4231	9526	0.14	0.01	20	3.30
ship08s	778	2467	7194	358	2063	4634	0.10	0.02	17	1.52
ship12l	1151	5533	16276	756	5170	11491	0.13	0.01	25	5.29
ship12s	1151	2869	8284	384	2134	4718	0.11	0.00	21	1.94
sierra	1227	2735	9252	1212	2705	7931	0.12	0.00	18	4.66
stair	356	620	4021	356	538	3831	0.10	0.00	18	1.43
standata	359	1274	3230	346	861	1537	0.08	0.00	15	1.14
standgub	361	1383	3339	346	861	1537	0.11	0.01	15	1.12
standmps	467	1274	3878	454	1245	2953	0.08	0.00	18	1.74
stocfor1	117	165	501	102	150	421	0.08	0.00	14	0.36
stocfor2	2157	3045	9357	1980	2868	8090	0.11	0.02	30	5.57
stocfor3	16675	23541	76473	15362	22228	63608	0.29	0.01	52	86.47
truss	1000	8806	27836	1000	8806	27836	0.17	0.02	20	7.67
tuff	333	630	4563	263	586	4284	0.08	0.01	23	1.61
vtpbase	198	347	1052	163	273	692	0.09	0.00	16	0.73
wood1p	244	2595	70216	243	2474	64483	0.15	0.00	19	6.77
woodw	1098	8418	37487	1094	8414	31579	0.19	0.00	29	11.69

Table 5.4: The Netlib Infeasible Problem Set

Problem	Row	Col	Nnz	Rrow	Rcol	Rnnz	Tpre	Tpost	Iter	Tsol
bgdbg1	348	629	1662	267	548	1205	0.15	0.00	10	1.01
bgetam	400	816	2537	334	669	1995	0.09	0.00	10	0.75
bgprtr	20	40	70	20	40	70	0.10	0.01	11	0.21
box1	231	261	651	231	261	651	0.07	0.00	7	0.25
chemcom	288	744	1590	288	744	1590	0.08	0.01	8	0.61
cplex1	3005	5224	10947	3005	5224	10947	0.08	0.02	16	6.78
cplex2	224	378	1215	224	378	1215	0.07	0.00	42	2.57
ex72a	197	215	467	197	215	467	0.09	0.01	8	0.25
ex73a	193	211	668	193	211	457	0.08	0.00	7	0.23
forest6	66	131	246	66	131	246	0.08	0.01	9	0.29
galenet	8	14	22	7	13	20	0.03	0.02	7	0.16
gosh	3792	13697	100672	3533	13336	97002	0.27	0.01	150	218.99
gran	2568	2525	20111	—			0.12			
greenbeainf	2393	5600	31087	2301	5271	29883	0.17	0.00	27	14.36
itest2	9	13	26	9	13	26	0.09	0.01	7	0.12
itest6	11	17	29	10	15	26	0.08	0.01	7	0.13
klein1	54	108	750	54	108	750	0.09	0.02	20	0.49
klein2	477	531	5062	477	531	5062	0.06	0.01	17	7.84
klein3	994	1082	13101	994	1082	13101	0.11	0.00	19	765.79
mondou2	313	604	1623	259	467	934	0.08	0.01	14	1.01
pang	361	757	2973	334	692	2661	0.10	0.01	25	1.75
pilot4i	410	1211	7342	400	1117	7106	0.09	0.01	20	2.42
qual	323	464	1646	305	441	1596	0.09	0.01	38	2.85
reactor	318	808	2591	305	793	2557	0.06	0.02	22	1.98
refinery	323	464	1626	303	439	1574	0.07	0.01	16	1.09
vol1	323	464	1714	305	441	1596	0.11	0.01	32	2.26
woodinfe	35	89	140	—			0.10			

— means the problem is detected infeasible in preprocessing.

Table 5.5: The Netlib Kennington Problem Set

Problem	Row	Col	Nnz	Rrow	Rcol	Rnnz	Tpre	Tpost	Iter	Tsol
cre-a	3516	7248	18168	3081	6901	17460	0.15	0.02	31	11.44
cre-b	9648	77137	260785	7236	77133	260752	0.79	0.02	35	213.76
cre-c	3068	6411	15977	2643	6068	15248	0.14	0.01	40	14.02
cre-d	8926	73948	246614	6456	73928	246489	0.72	0.02	33	180.34
ken-07	2426	3602	8404	1437	2613	5994	0.10	0.02	17	4.93
ken-11	14694	21349	49058	10085	16740	38520	0.15	0.01	20	41.05
ken-13	28632	42659	97246	22534	36561	82698	0.26	0.01	26	126.88
ken-18	105127	154699	358171	78862	128434	298858	0.72	0.02	38	836.12
osa-07	1118	25067	144812	1081	25030	89316	0.30	0.02	23	25.12
osa-14	2337	54797	317097	2300	54760	196716	0.67	0.00	48	156.90
osa-30	4350	104374	604488	4313	104337	377404	1.09	0.01	41	271.23
osa-60	10280	243246	1408073	10243	243209	849356	2.55	0.03	42	721.57
pds-02	2953	7716	16571	2654	7417	15964	0.14	0.02	32	18.51
pds-06	9881	29351	63220	9366	28836	62100	0.19	0.02	43	149.14
pds-10	16558	49932	107605	15978	49352	106310	0.31	0.01	57	508.40
pds-20	33874	108175	232647	32947	107324	230621	0.59	0.03	79	3241.03

From Table 5.1 to 5.5, the testing results are obtained by McIPM that uses both McPre and McIPM solver. We can see that all the LO problems in the Netlib sets can be processed by preprocessor and solved successfully. Moreover, it is shown in Table 5.4 that two problems, *gran* and *woodinfe*, can be detected to be infeasible in preprocessing.

5.2 Comparison with LIPSOL

In this section, we compare our results with the academic optimization software package LIPSOL.

From Table 5.6 to 5.10, there are two groups of data. The data in the first group are the results obtained from McIPM, i.e., our preprocessor, McPre (McIPM Preprocessor) combining with the McIPM solver. The data in the other group are the results obtained from LIPSOL. In each group, *Rrow*, *Rcol* and *Rnnz* mean the reduced size of row, column and nonzero element, respectively. *Tpre*, *Tpost* and *Tsol*

mean the time spent in preprocessing, postprocessing and solver, respectively.

From Table 5.6 to 5.8, it can be seen that McPre can eliminate the same number of constraints in 25 problems. Moreover, McPre can eliminate more constraints in 70 problems. For small scale problems, e.g., *25fv47* and *adlittle*, the time that McPre spends in preprocessing is a little more than LIPSOL does. However, for the large scale problems (over 10,000 variables), e.g., *fit2d* and *greenbea*, McPre is faster than LIPSOL. The reason is that we always spend time in writing the preprocessing information to DAT files, while LIPSOL does not have such writing procedure. For small scale problems, the time spent in the reading and writing procedures is a big fraction of the whole preprocessing time while for large scale problems, that time is a relatively small fraction of the total preprocessing time. For the infeasible problems in Table 5.9, we can detect the same two infeasible problems during preprocessing. We eliminate more constraints than LIPSOL in 11 problems. For the other problems, McPre and LIPSOL remove the same number of constraints. For the problems in Table 5.10, McPre can process all the problems while LIPSOL can not process the two largest problems: *ken-18* and *pds-20*. For the remaining problems except four relatively small problems, i.e., *cre-a*, *cre-c*, *ken-07* and *pds-02*, McPre spends less time in preprocessing than LIPSOL. For the aspect of preprocessing time, McPre is superior to LIPSOL on large scale problems.

From Table 5.11 to 5.15, we compare the results with different preprocessor but with the same solver. In each table, there are two groups of data. The data in the first group are the results obtained from the McIPM, i.e., McPre combining McIPM solver, that are the same as what we have presented. The data in the other group are the results obtained by using LIPSOL Pre (LIPSOL Preprocessor) combining with McIPM solver. In each group, we present the size of row, column in the reduced matrix, the time spent in preprocessing, postprocessing and solver, and the iteration

Table 5.6: Comparison between McIPM and LIPSOL: Netlib Standard Problem Set (I)

	Results from McIPM						**Results from LIPSOL**					
Problem	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol
25fv47	788	1843	10538	0.11	0.01	4.84	798	1854	10580	0.09	0.01	1.89
80bau3b	2183	11163	21908	0.11	0.02	29.67	2235	11516	22648	0.23	0.02	9.67
adlittle	55	137	417	0.08	0.01	0.33	55	137	417	0.02	0.00	0.14
afiro	27	51	102	0.07	0.01	0.18	27	51	102	0.01	0.00	0.07
agg	468	595	2780	0.09	0.01	1.87	488	615	2862	0.03	0.00	1.06
agg2	516	758	4740	0.10	0.01	2.16	516	758	4740	0.04	0.00	1.11
agg3	516	758	4756	0.10	0.01	2.29	516	758	4756	0.05	0.00	1.10
bandm	251	417	1982	0.08	0.02	0.81	269	436	2137	0.02	0.00	0.44
beaconfd	107	218	1888	0.09	0.02	0.38	148	270	3105	0.03	0.01	0.32
blend	71	111	477	0.11	0.01	0.27	74	114	522	0.01	0.00	0.15
bnl1	624	1564	5498	0.10	0.01	3.17	632	1576	5522	0.04	0.00	1.03
bnl2	1998	4078	14259	0.16	0.02	10.08	2268	4430	14914	0.11	0.00	4.27
boeing1	344	719	3352	0.08	0.01	2.17	347	722	3819	0.03	0.00	0.94
boeing2	125	264	922	0.09	0.02	0.71	140	279	1332	0.02	0.00	0.46
bore3d	124	217	1009	0.10	0.02	0.72	199	300	1324	0.03	0.00	0.39
brandy	136	246	1964	0.04	0.00	0.70	149	259	2015	0.01	0.00	0.34
capri	241	436	1528	0.09	0.01	1.15	267	476	1905	0.04	0.00	0.57
cycle	1496	2999	16669	0.13	0.01	11.39	1801	3305	20805	0.18	0.00	5.24
czprob	719	3123	6209	0.10	0.01	5.49	737	3141	9454	0.09	0.01	2.03
d2q06c	2132	5728	31965	0.12	0.01	26.76	2171	5831	33081	0.16	0.01	9.55
d6cube	403	5443	34233	0.12	0.01	6.07	404	6184	37704	0.08	0.01	4.64
degen2	444	757	4201	0.09	0.01	1.10	444	757	4201	0.02	0.00	0.67
degen3	1503	2604	25432	0.17	0.00	8.17	1503	2604	25432	0.05	0.01	8.42
dfl001	5984	12143	35338	0.15	0.01	263.27	6071	12230	35632	0.11	0.01	372.20
e226	214	463	2702	0.09	0.01	0.99	220	469	2737	0.03	0.00	0.48
etamacro	334	669	1995	0.08	0.02	1.96	357	692	2044	0.03	0.00	0.77
fffff800	322	826	5164	0.08	0.00	2.39	501	1005	6283	0.04	0.00	1.49
finnis	456	966	2414	0.07	0.01	2.03	492	1014	2527	0.03	0.00	0.94
fit1d	24	1049	13427	0.11	0.01	2.70	24	1049	13427	0.05	0.00	1.17
fit1p	627	1677	9868	0.11	0.01	9.85	627	1677	9868	0.06	0.00	7.96

number separately. It can be seen that the effect of different preprocessor on McIPM solver is not obvious. Using McPre, some problems has less iteration number and less time spent in solver, but some problems has more iteration number and more time spent in solver.

Table 5.7: Comparison between McIPM and LIPSOL: Netlib Standard Problem Set (II)

	Results from McIPM						**Results from LIPSOL**					
Problem	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol
fit2d	25	10524	129042	0.15	0.01	31.69	25	10524	129042	0.45	0.00	13.79
fit2p	3000	13525	50284	0.16	0.00	37.53	3000	13525	50284	0.31	0.01	28.86
forplan	122	450	4477	0.08	0.00	1.74	135	463	4539	0.05	0.00	0.69
ganges	1113	1510	6537	0.09	0.02	3.07	1137	1534	6593	0.04	0.01	1.23
gfrd-pnc	590	1134	2393	0.07	0.02	1.65	600	1144	2413	0.01	0.00	0.66
greenbea	2297	5270	29777	0.13	0.00	26.67	2318	5424	30434	0.17	0.01	10.66
greenbeb	2296	5258	29477	0.16	0.00	26.29	2317	5415	30384	0.29	9.73	0.02
grow15	300	645	5620	0.10	0.01	1.90	300	645	5620	0.04	0.00	0.92
grow22	440	946	8252	0.07	0.01	2.75	440	946	8252	0.04	0.00	1.42
grow7	140	301	2612	0.08	0.01	1.11	140	301	2612	0.01	0.00	0.50
israel	174	316	2443	0.06	0.01	1.71	174	316	2443	0.02	0.00	1.08
kb2	43	68	313	0.11	0.01	0.47	43	68	313	0.01	0.00	0.21
lotfi	133	346	867	0.09	0.01	0.53	151	364	1123	0.02	0.00	0.27
maros	789	1860	9489	0.11	0.01	3.91	835	1921	10060	0.06	0.00	2.39
maros-r7	2152	7440	100486	0.24	0.01	30.13	3136	9408	144848	0.39	0.00	49.33
modszk1	665	1599	3065	0.10	0.01	2.56	686	1622	12943	0.02	0.00	0.95
nesm	654	2922	13244	0.12	0.01	9.48	654	2922	13244	0.08	0.01	3.43
perold	613	1500	6963	0.11	0.02	5.07	625	1530	7131	0.05	0.00	1.91
pilot	1423	4639	42242	0.17	0.01	44.96	1441	4657	42300	0.20	0.01	17.32
pilotja	858	1972	12900	0.12	0.03	8.98	924	2044	13339	0.10	0.00	3.84
pilotwe	718	2925	9494	0.10	0.00	8.32	722	2930	9537	0.08	0.00	2.43
pilot4	400	1117	7106	0.10	0.01	4.66	402	1173	7226	0.04	0.01	1.79
pilot87	1990	6411	72258	0.26	0.01	156.98	2030	6460	72479	0.33	0.00	51.04
pilotnov	864	2149	12037	0.08	0.01	5.60	951	2242	12460	0.07	0.00	2.24
recipe	71	137	476	0.06	0.00	0.46	85	177	249	0.02	0.00	0.16
sc105	104	162	339	0.10	0.02	0.29	105	163	340	0.01	0.00	0.12
sc205	203	315	663	0.07	0.00	0.36	205	317	665	0.00	0.00	0.14
sc50a	49	77	159	0.08	0.01	0.23	49	77	159	0.12	0.00	0.10
sc50b	48	76	146	0.07	0.00	0.21	48	76	146	0.01	0.00	0.07
scagr25	469	669	1715	0.10	0.02	0.82	471	671	1725	0.00	0.00	0.36

Table 5.8: Comparison between McIPM and LIPSOL: Netlib Standard Problem Set (III)

	Results from McIPM						**Results from LIPSOL**					
Problem	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol
scagr7	127	183	455	0.07	0.01	0.35	129	185	465	0.01	0.00	0.17
scfxm1	311	581	2497	0.08	0.02	1.18	322	592	2707	0.03	0.00	0.45
scfxm2	622	1162	4999	0.07	0.01	2.33	644	1184	5419	0.03	0.00	0.86
scfxm3	933	1743	7501	0.11	0.01	3.31	966	1776	8131	0.06	0.01	1.22
scorpion	361	436	1415	0.08	0.01	0.58	375	453	1460	0.01	0.01	0.29
scrs8	430	1215	3067	0.09	0.01	1.85	485	1270	3262	0.02	0.00	0.60
scsd1	77	760	2388	0.09	0.00	0.46	77	760	2388	0.01	0.00	0.15
scsd6	147	1350	4316	0.08	0.02	0.80	147	1350	4316	0.01	0.00	0.31
scsd8	397	2750	8584	0.06	0.01	1.25	397	2750	8584	0.02	0.00	0.47
sctap1	284	644	1802	0.09	0.01	0.87	300	660	1872	0.00	0.00	0.34
sctap2	1033	2443	7052	0.11	0.01	2.52	1090	2500	7334	0.00	0.00	0.97
sctap3	1408	3268	9383	0.11	0.02	3.84	1480	3340	9734	0.04	0.01	1.21
seba	514	1033	4342	0.06	0.03	4.29	515	1036	4360	0.02	0.00	3.34
share1b	112	248	1148	0.09	0.00	0.79	112	248	1148	0.04	0.00	0.28
share2b	96	162	777	0.09	0.01	0.33	96	162	777	0.02	0.00	0.17
shell	487	1451	2906	0.09	0.00	2.44	496	1487	2978	0.02	0.01	0.70
ship04l	323	2104	4734	0.10	0.03	1.65	356	2162	6368	0.03	0.01	0.53
ship04s	235	1356	3044	0.07	0.00	1.13	268	1414	4124	0.02	0.00	0.38
ship08l	630	4231	9526	0.14	0.01	3.30	688	4339	12810	0.07	0.00	1.10
ship08s	358	2063	4634	0.10	0.02	1.52	416	2171	6306	0.04	0.00	0.58
ship12l	756	5170	11491	0.13	0.01	5.29	838	5329	15664	0.08	0.01	1.53
ship12s	384	2134	4718	0.11	0.00	1.94	466	2293	6556	0.05	0.00	0.69
sierra	1212	2705	7931	0.12	0.00	4.66	1222	2715	7951	0.06	0.01	1.74
stair	356	538	3831	0.10	0.00	1.43	356	538	3831	0.04	0.00	0.61
standata	346	861	1537	0.08	0.00	1.14	359	1258	3173	0.02	0.00	0.53
standgub	346	861	1537	0.11	0.01	1.12	361	1366	3281	0.02	0.00	0.53
standmps	454	1245	2953	0.08	0.00	1.74	467	1258	3862	0.02	0.00	0.87
stocfor1	102	150	421	0.08	0.00	0.36	109	157	471	0.01	0.00	0.18
stocfor2	1980	2868	8090	0.11	0.02	5.57	2157	3045	9357	0.03	0.00	1.85
stocfor3	15362	22228	63608	0.29	0.01	86.47	16675	23541	76473	0.44	0.01	23.43
truss	1000	8806	27836	0.17	0.02	7.67	1000	8806	27836	0.10	0.01	2.68
tuff	263	586	4284	0.08	0.01	1.61	292	617	4549	0.03	0.00	0.77
vtpbase	163	273	692	0.09	0.00	0.73	194	325	937	0.02	0.00	0.47
wood1p	243	2474	64483	0.15	0.00	6.77	244	2595	70216	0.25	0.00	4.48
woodw	1094	8414	31579	0.19	0.00	11.69	1098	8418	37487	0.17	0.00	5.48

Table 5.9: Comparison between McIPM and LIPSOL: Netlib Infeasible Problem Set

	Results from McIPM						**Results from LIPSOL**					
Problem	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol	Rrow	Rcol	Rnnz	Tpre	Tpost	Tsol
bgdbg1	267	548	1205	0.15	0.00	1.01	348	629	1662	0.01	0.00	0.18
bgetam	334	669	1995	0.09	0.00	0.75	357	692	2044	0.03	0.00	0.22
bgprtr	20	40	70	0.10	0.01	0.21	20	40	70	0.01	0.00	0.07
box1	231	261	651	0.07	0.00	0.25	231	261	651	0.01	0.00	0.06
chemcom	288	744	1590	0.08	0.01	0.61	288	744	1590	0.01	0.00	0.18
cplex1	3005	5224	10947	0.08	0.02	6.78	3005	5224	10947	0.08	0.01	1.44
cplex2	224	378	1215	0.07	0.00	2.57	224	378	1215	0.02	0.00	1.20
ex72a	197	215	467	0.09	0.01	0.25	197	215	467	0.01	0.00	0.06
ex73a	193	211	668	0.08	0.00	0.23	193	211	457	0.02	0.01	0.05
forest6	66	131	246	0.08	0.01	0.29	66	131	246	0.01	0.00	0.13
galenet	8	14	22	0.03	0.02	0.16	8	14	22	0.02	0.00	0.05
gosh	3533	13336	97002	0.27	0.01	218.99	3718	13625	100566	0.52	0.01	15.35
gran	—			0.12			—					
greenbea	2301	5271	29883	0.17	0.00	14.36	2319	5419	30425	0.18	0.01	3.48
itest2	9	13	26	0.09	0.01	0.12	9	13	26	0.01	0.00	0.05
itest6	10	15	26	0.08	0.01	0.13	11	17	29	0.01	0.00	0.05
klein1	54	108	750	0.09	0.02	0.49	54	108	750	0.00	0.00	0.25
klein2	477	531	5062	0.06	0.01	7.84	477	531	5062	0.03	0.00	4.74
klein3	994	1082	13101	0.11	0.00	765.79	994	1082	13101	0.03	0.01	844.58
mondou2	259	467	934	0.08	0.01	1.01	259	467	934	0.02	0.00	0.18
pang	334	692	2661	0.10	0.01	1.75	357	727	2932	0.02	0.00	0.79
pilot4i	400	1117	7106	0.09	0.01	2.42	402	1173	7226	0.04	0.00	1.01
qual	305	441	1596	0.09	0.01	2.85	323	459	1633	0.02	0.00	1.20
reactor	305	793	2557	0.06	0.02	1.98	318	806	2589	0.02	0.00	0.40
refinery	303	439	1574	0.07	0.01	1.09	319	455	1600	0.02	0.00	0.44
vol1	305	441	1596	0.11	0.01	2.26	323	459	1633	0.02	0.00	0.70
woodinfe	—			0.10			—					

— means the problem is detected infeasible in preprocessing.

Table 5.10: Comparison between McIPM and LIPSOL: Netlib Kennington Problem Set

	Results from McIPM			**Results from LIPSOL**		
Problem	Rrow	Rcol	Rnnz	Rrow	Rcol	Rnnz
cre-a	3081	6901	17460	3428	7248	18168
cre-b	7236	77133	260752	7240	77137	260785
cre-c	2643	6068	15248	2986	6411	15911
cre-d	6456	73928	246489	6476	73948	246614
ken-07	1437	2613	5994	1691	2867	6640
ken-11	10085	16740	38520	11548	18203	42161
ken-13	22534	36561	82698	23393	37420	84909
ken-18	78862	128434	298858	*		
osa-07	1081	25030	89316	1118	25067	144812
osa-14	2300	54760	196716	2337	54797	317097
osa-30	4313	104337	377404	4350	104374	604488
osa-60	10243	243209	849356	*		
pds-02	2654	7417	15964	2788	7551	16230
pds-06	9366	28836	62100	9617	29087	62582
pds-10	15978	49352	106310	16239	49613	106802
pds-20	32947	107324	230621	33250	107627	231155
	Results from McIPM			**Results from LIPSOL**		
Problem	Tpre	Tpost	Tsol	Tpre	Tpost	Tsol
cre-a	0.15	0.02	11.44	0.07	0.01	4.66
cre-b	0.79	0.02	213.76	0.89	0.02	112.10
cre-c	0.14	0.01	14.02	0.06	0.00	4.59
cre-d	0.72	0.02	180.34	0.81	0.02	95.36
ken-07	0.10	0.02	4.93	0.05	0.02	2.09
ken-11	0.15	0.01	41.05	0.31	0.02	18.73
ken-13	0.26	0.01	126.88	0.59	0.04	47.97
ken-18	0.72	0.02	836.12	*		
osa-07	0.30	0.02	25.12	0.38	0.01	19.01
osa-14	0.67	0.00	156.90	0.87	0.01	55.79
osa-30	1.09	0.01	271.23	1.86	0.02	130.26
osa-60	2.55	0.03	721.57	*		
pds-02	0.14	0.02	18.51	0.09	0.00	5.14
pds-06	0.19	0.02	149.14	0.30	0.02	54.49
pds-10	0.31	0.01	508.40	0.54	0.04	244.05
pds-20	0.59	0.03	3241.03	1.38	0.10	2062.98

* means the problem can not processed in preprocessing.

Table 5.11: Preprocessor Comparison: Netlib Standard Problem Set (I)

	Results from McPre						**Results from LIPSOL Pre**					
Problem	Rrow	Rcol	Tpre	Tpost	Iter	Tsol	Rrow	Rcol	Tpre	Tpost	Iter	Tsol
25fv47	788	1843	0.11	0.01	30	4.84	798	1854	0.24	0.01	29	4.65
80bau3b	2183	11163	0.11	0.02	41	29.67	2235	11516	0.28	0.02	41	30.31
adlittle	55	137	0.08	0.01	14	0.33	55	137	0.06	0.01	14	0.34
afiro	27	51	0.07	0.01	10	0.18	27	51	0.07	0.00	10	0.19
agg	468	595	0.09	0.01	22	1.87	488	615	0.07	0.00	22	1.91
agg2	516	758	0.10	0.01	21	2.16	516	758	0.07	0.00	20	2.13
agg3	516	758	0.10	0.01	22	2.29	516	758	0.07	0.01	21	2.18
bandm	251	417	0.08	0.02	18	0.81	269	436	0.07	0.00	18	0.84
beaconfd	107	218	0.09	0.02	11	0.38	148	270	0.07	0.00	12	0.54
blend	71	111	0.11	0.01	10	0.27	74	114	0.10	0.00	11	0.28
bnl1	624	1564	0.10	0.01	33	3.17	632	1576	0.08	0.00	31	3.11
bnl2	1998	4078	0.16	0.02	36	10.08	2268	4430	0.13	0.00	38	11.41
boeing1	344	719	0.08	0.01	24	2.17	347	722	0.09	0.00	25	2.37
boeing2	125	264	0.09	0.02	15	0.71	140	279	0.07	0.00	19	0.98
bore3d	124	217	0.10	0.02	18	0.72	199	300	0.08	0.00	18	0.84
brandy	136	246	0.04	0.00	19	0.70	149	259	0.10	0.00	18	0.67
capri	241	436	0.09	0.01	19	1.15	267	476	0.07	0.00	18	1.17
cycle	1496	2999	0.13	0.01	39	11.39	1801	3305	0.25	0.00	39	13.72
czprob	719	3123	0.10	0.01	36	5.49	737	3141	0.15	0.01	36	6.62
d2q06c	2132	5728	0.12	0.01	52	26.76	2171	5831	0.22	0.01	43	23.21
d6cube	403	5443	0.12	0.01	19	6.07	404	6184	0.15	0.00	21	7.20
degen2	444	757	0.09	0.01	12	1.10	444	757	0.07	0.00	12	1.06
degen3	1503	2604	0.17	0.00	14	8.17	1503	2604	0.14	0.00	14	8.08
dfl001	5984	12143	0.15	0.01	45	263.27	6071	12230	0.18	0.00	45	234.35
e226	214	463	0.09	0.01	21	0.99	220	469	0.10	0.00	21	1.00
etamacro	334	669	0.08	0.02	27	1.96	357	692	0.10	0.00	25	1.82
ffff800	322	826	0.08	0.00	32	2.39	501	1005	0.05	0.00	27	2.59
finnis	456	966	0.07	0.01	25	2.03	492	1014	0.05	0.00	25	2.11
fit1d	24	1049	0.11	0.01	19	2.70	24	1049	0.10	0.00	25	3.77
fit1p	627	1677	0.11	0.01	19	9.85	627	1677	0.13	0.00	16	8.34

Table 5.12: Preprocessor Comparison: Netlib Standard Problem Set (II)

	Results from McPre						**Results from LIPSOL Pre**					
Problem	Rrow	Rcol	Tpre	Tpost	Iter	Tsol	Rrow	Rcol	Tpre	Tpost	Iter	Tsol
fit2d	25	10524	0.15	0.01	23	31.69	25	10524	0.51	0.00	23	32.86
fit2p	3000	13525	0.16	0.00	21	37.53	3000	13525	0.36	0.00	21	37.89
forplan	122	450	0.08	0.00	29	1.74	135	463	0.09	0.00	31	1.96
ganges	1113	1510	0.09	0.02	20	3.07	1137	1534	0.09	0.00	21	3.25
gfrd-pnc	590	1134	0.07	0.02	18	1.65	600	1144	0.10	0.01	18	1.66
greenbea	2297	5270	0.13	0.00	51	26.67	2318	5424	0.22	0.01	47	25.30
greenbeb	2296	5258	0.16	0.00	51	26.29	2317	5415	0.24	0.01	46	24.00
grow15	300	645	0.10	0.01	18	1.90	300	645	0.07	0.00	18	1.88
grow22	440	946	0.07	0.01	19	2.75	440	946	0.09	0.00	18	2.56
grow7	140	301	0.08	0.01	18	1.11	140	301	0.05	0.00	18	1.10
israel	174	316	0.06	0.01	22	1.71	174	316	0.09	0.00	22	1.68
kb2	43	68	0.11	0.01	17	0.47	43	68	0.06	0.00	17	0.49
lotfi	133	346	0.09	0.01	17	0.53	151	364	0.08	0.00	23	0.73
maros	789	1860	0.11	0.01	30	3.91	835	1921	0.13	0.00	31	4.40
maros-r7	2152	7440	0.24	0.01	16	30.13	3136	9408	0.44	0.00	16	63.46
modszk1	665	1599	0.10	0.01	30	2.56	686	1622	0.08	0.00	29	2.53
nesm	654	2922	0.12	0.01	36	9.48	654	2922	0.13	0.01	31	7.68
perold	613	1500	0.11	0.02	40	5.07	625	1530	0.09	0.01	45	6.22
pilot	1423	4639	0.17	0.01	52	44.96	1441	4657	0.25	0.01	64	51.71
pilotja	858	1972	0.12	0.03	42	8.98	924	2044	0.12	0.00	42	9.33
pilotwe	718	2925	0.10	0.00	44	8.32	722	2930	0.13	0.00	43	7.34
pilot4	400	1117	0.10	0.01	39	4.66	402	1173	0.08	0.00	36	4.43
pilot87	1990	6411	0.26	0.01	85	156.98	2030	6460	0.39	0.00	70	125.27
pilotnov	864	2149	0.08	0.01	27	5.60	951	2242	0.12	0.01	27	5.72
recipe	71	137	0.06	0.00	12	0.46	85	177	0.07	0.00	12	0.48
sc105	104	162	0.10	0.02	12	0.29	105	163	0.04	0.00	12	0.29
sc205	203	315	0.07	0.00	12	0.36	205	317	0.05	0.00	12	0.37
sc50a	49	77	0.08	0.01	11	0.23	49	77	0.07	0.00	11	0.23
sc50b	48	76	0.07	0.00	10	0.21	48	76	0.07	0.00	10	0.21
scagr25	469	669	0.10	0.02	16	0.82	471	671	0.04	0.00	16	0.81

Table 5.13: Preprocessor Comparison: Netlib Standard Problem Set (III)

	Results from McPre						**Results from LIPSOL Pre**					
Problem	Rrow	Rcol	Tpre	Tpost	Iter	Tsol	Rrow	Rcol	Tpre	Tpost	Iter	Tsol
scagr7	127	183	0.07	0.01	13	0.35	129	185	0.08	0.00	13	0.35
scfxm1	311	581	0.08	0.02	23	1.18	322	592	0.09	0.01	23	1.19
scfxm2	622	1162	0.07	0.01	27	2.33	644	1184	0.08	0.00	25	2.11
scfxm3	933	1743	0.11	0.01	27	3.31	966	1776	0.12	0.00	29	3.69
scorpion	361	436	0.08	0.01	14	0.58	375	453	0.09	0.01	14	0.60
scrs8	430	1215	0.09	0.01	26	1.85	485	1270	0.08	0.00	24	1.70
scsd1	77	760	0.09	0.00	10	0.46	77	760	0.08	0.00	10	0.46
scsd6	147	1350	0.08	0.02	12	0.80	147	1350	0.06	0.00	12	0.80
scsd8	397	2750	0.06	0.01	10	1.25	397	2750	0.08	0.00	10	1.23
sctap1	284	644	0.09	0.01	19	0.87	300	660	0.07	0.00	17	0.80
sctap2	1033	2443	0.11	0.01	19	2.52	1090	2500	0.06	0.00	13	1.79
sctap3	1408	3268	0.11	0.02	20	3.84	1480	3340	0.10	0.00	14	2.44
seba	514	1033	0.06	0.03	24	4.29	515	1036	0.07	0.00	26	4.85
share1b	112	248	0.09	0.00	28	0.79	112	248	0.08	0.00	27	0.76
share2b	96	162	0.09	0.01	12	0.33	96	162	0.06	0.00	11	0.30
shell	487	1451	0.09	0.00	24	2.44	496	1487	0.09	0.00	25	2.65
ship04l	323	2104	0.10	0.03	19	1.65	356	2162	0.09	0.00	16	1.60
ship04s	235	1356	0.07	0.00	18	1.13	268	1414	0.08	0.00	16	1.17
ship08l	630	4231	0.14	0.01	20	3.30	688	4339	0.13	0.00	19	3.66
ship08s	358	2063	0.10	0.02	17	1.52	416	2171	0.09	0.01	17	1.73
ship12l	756	5170	0.13	0.01	25	5.29	838	5329	0.15	0.00	28	7.05
ship12s	384	2134	0.11	0.00	21	1.94	466	2293	0.09	0.01	21	2.28
sierra	1212	2705	0.12	0.00	18	4.66	1222	2715	0.11	0.00	18	4.67
stair	356	538	0.10	0.00	18	1.43	356	538	0.08	0.00	18	1.41
standata	346	861	0.08	0.00	15	1.14	359	1258	0.08	0.00	17	1.47
standgub	346	861	0.11	0.01	15	1.12	361	1366	0.07	0.00	17	1.53
standmps	454	1245	0.08	0.00	18	1.74	467	1258	0.07	0.00	19	1.79
stocfor1	102	150	0.08	0.00	14	0.36	109	157	0.03	0.00	13	0.33
stocfor2	1980	2868	0.11	0.02	30	5.57	2157	3045	0.07	0.00	31	6.24
stocfor3	15362	22228	0.29	0.01	52	86.47	16675	23541	0.47	0.00	49	84.14
truss	1000	8806	0.17	0.02	20	7.67	1000	8806	0.14	0.00	20	7.58
tuff	263	586	0.08	0.01	23	1.61	292	617	0.08	0.00	20	1.49
vtpbase	163	273	0.09	0.00	16	0.73	194	325	0.08	0.00	17	0.81
wood1p	243	2474	0.15	0.00	19	6.77	244	2595	0.29	0.00	15	5.66
woodw	1094	8414	0.19	0.00	29	11.69	1098	8418	0.16	0.00	26	11.16

Table 5.14: Preprocessor Comparison: Netlib Infeasible Problem Set

	Results from McPre						**Results from LIPSOL Pre**					
Problem	Rrow	Rcol	Tpre	Tpost	Iter	Tsol	Rrow	Rcol	Tpre	Tpost	Iter	Tsol
bgdbg1	267	548	0.15	0.00	10	1.01	348	629	0.13	0.00	10	1.08
bgetam	334	669	0.09	0.00	10	0.75	357	692	0.07	0.01	9	0.70
bgprtr	20	70	0.10	0.01	11	0.21	20	40	0.07	0.00	11	0.20
box1	231	261	0.07	0.00	7	0.25	231	261	0.07	0.00	7	0.24
chemcom	288	744	0.08	0.01	8	0.61	288	744	0.05	0.01	8	0.60
cplex1	3005	5224	0.08	0.02	16	6.78	3005	5224	0.13	0.00	16	6.65
cplex2	224	378	0.07	0.00	42	2.57	224	378	0.06	0.00	41	2.55
ex72a	197	215	0.09	0.01	8	0.25	197	215	0.04	0.00	8	0.25
ex73a	193	211	0.08	0.00	7	0.23	193	211	0.06	0.00	7	0.22
forest6	66	131	0.08	0.01	9	0.29	66	131	0.06	0.00	9	0.29
galenet	7	13	0.03	0.02	7	0.16	8	14	0.04	0.00	7	0.17
gosh	3533	13336	0.27	0.01	150	218.9	3718	13625	0.57	0.02	150	237.1
gran	—		0.12				—		0.20			
greenbea	2301	5271	0.17	0.00	27	14.36	2319	5419	0.21	0.01	22	11.41
itest2	9	13	0.09	0.01	7	0.12	9	13	0.06	0.00	7	0.13
itest6	10	15	0.08	0.01	7	0.13	11	17	0.07	0.00	7	0.13
klein1	54	108	0.09	0.02	20	0.49	54	108	0.06	0.00	20	0.49
klein2	477	531	0.06	0.01	17	7.84	477	531	0.07	0.00	17	7.81
klein3	994	1082	0.11	0.00	19	765.8	994	1082	0.06	0.00	19	751.4
mondou2	259	467	0.08	0.01	14	1.01	259	467	0.10	0.00	14	1.03
pang	334	692	0.10	0.01	25	1.75	357	727	0.07	0.00	30	2.16
pilot4i	400	1117	0.09	0.01	20	2.42	402	1173	0.08	0.00	18	2.20
qual	305	441	0.09	0.01	38	2.85	323	459	0.06	0.00	31	2.24
reactor	305	793	0.06	0.02	22	1.98	318	806	0.06	0.00	20	1.82
refinery	303	439	0.07	0.01	16	1.09	319	455	0.06	0.00	17	1.23
vol1	305	441	0.11	0.01	32	2.26	323	459	0.06	0.00	27	2.01
woodinfe	—		0.10				—		0.01			

— means the problem is detected infeasible in preprocessing.

Table 5.15: Preprocessor Comparison: Netlib Kennington Problem Set

	Results from McPre			**Results from LIPSOL Pre**		
Problem	Rrow	Rcol	Tpre	Rrow	Rcol	Tpre
cre-a	3081	6901	0.15	3428	7248	0.22
cre-b	7236	77133	0.79	7240	77137	1.06
cre-c	2643	6068	0.14	2986	6411	0.11
cre-d	6456	73928	0.72	6476	73948	0.95
ken-07	1437	2613	0.10	1691	2867	0.13
ken-11	10085	16740	0.15	11548	18203	0.38
ken-13	22534	36561	0.26	23393	37420	0.66
ken-18	78862	128434	0.72	*		
osa-07	1081	25030	0.30	1118	25067	0.58
osa-14	2300	54760	0.67	2337	54797	1.12
osa-30	4313	104337	1.09	4350	104374	2.05
osa-60	10243	243209	2.55	*		
pds-02	2654	7417	0.14	2788	7551	0.13
pds-06	9366	28836	0.19	9617	29087	0.42
pds-10	15978	49352	0.31	16239	49613	0.62
pds-20	32947	107324	0.59	33250	107627	1.51

	Results from McPre			**Results from LIPSOL Pre**		
Problem	Tpost	Iter	Tsol	Tpost	Iter	Tsol
cre-a	0.02	31	11.44	0.01	28	10.72
cre-b	0.02	35	213.76	0.02	36	211.62
cre-c	0.01	40	14.02	0.01	35	13.33
cre-d	0.02	33	180.34	0.02	33	176.04
ken-07	0.02	17	4.93	0.00	17	5.46
ken-11	0.01	20	41.05	0.02	20	44.97
ken-13	0.01	26	126.88	0.04	27	133.57
ken-18	0.02	38	836.12	*		
osa-07	0.02	23	25.12	0.02	34	54.17
osa-14	0.00	48	156.90	0.01	40	153.52
osa-30	0.01	41	271.23	0.03	45	354.36
osa-60	0.03	42	721.57	*		
pds-02	0.02	32	18.51	0.01	32	18.68
pds-06	0.02	43	149.14	0.03	44	152.25
pds-10	0.01	57	508.40	0.04	55	486.79
pds-20	0.03	79	3241.03	0.11	79	3152.44

* means the problem can not processed in preprocessing.

5.3 Comparison with CPLEX

In this section, we compare our results with the commercial optimization software package CPLEX.

The comparison results are listed from Table 5.16 to 5.20. Notice that there are two columns *Erow* and *Ecol* in each table, that are different from what we have presented in other tables. *Erow* means the number of eliminated rows while *Ecol* means the number of eliminated columns. In other tables, we use *Rrow* and *Rcol* to represent the number of rows and columns after the eliminated rows and columns are removed from the matrix A. For instance, for the problem *25fv47* in Table 5.1, the number of *Row* is 821, the number of *Rrow* is 788, thus in Table 5.16, the number of *Erow* is 33. The reason we change our method to compare is that in CPLEX, the result of preprocessing is presented in this way, therefore, we adjust our data output.

For the Netlib standard problems from Table 5.16 to 5.18 and the Netlib Kennington problems in Table 5.20, CPLEX can remove more constraints than McPre for most problems. The reason is that another four preprocessing techniques (see Section 4.5) have not been implemented in McPre. For most problems, CPLEX spends less time in preprocessing and solver than McPre. Two of the reasons are: (I) McPre spends time in reading and writing procedures that CPLEX does not need. (II) CPLEX adjusts its code on different machine architecture, that can optimize system and speed up the running time. In spite of the extra data communication, for the very large scale probles, e.g., *osa-30*, *osa-60*, *pds-10* and *pds-20*, McPre spends less time than CPLEX. Moreover, another advantage of McPre is that it can read all the problems, however, there are eight problems, e.g., *blend*, *dfl001*, *forplan*, *gfrd-pnc*, *perold*, *pilotwe*, *scrs8* and *sierra* can not be read by CPLEX. CPLEX outputs the reading errors found in the RHS section.

For the Netlib infeasible problems in Table 5.19, CPLEX can detect twelve infeasible problems in preprocessing while McPre can detect two infeasible problems. The reason is that those four preprocessing techniques make it possible for CPLEX to detect more infeasibility, thus there is room to complete and improve McPre.

5.4 Computational Result on Extra Large Problems

In addition to the Netlib testing set problems, we also tested our code on some large problems from other resources. Here we list the computational results as follows:

The results in Table 5.21 are obtained by using McIPM. The comparison between McIPM and LIPSOL is shown in Table 5.22. In Table 5.23, the comparison is between the different preprocessor, i.e., McPre and LIPSOL Pre, both with McIPM solver. It is seen that McPre can remove more constraints than LIPSOL Pre does in four problems, and they remove the same number of constraints from the left six problems. McPre spends less time than LIPSOL Pre for all the problems except *l30*. This shows again that McPre is superior to LIPSOL Pre in preprocessing time for large scale problems.

Table 5.16: Comparison with CPLEX: Netlib Standard Problem Set (I)

	Results from McIPM					**Results from CPLEX**				
Problem	Erow	Ecol	Tpre	Tpost	Tsol	Erow	Ecol	Tpre	Tpost	Tsol
25fv47	33	33	0.11	0.01	4.84	50	36	0.03	0.00	0.32
80bau3b	79	898	0.11	0.02	29.67	297	1123	0.10	0.00	1.11
adlittle	1	1	0.08	0.01	0.33	3	3	0.00	0.00	0.01
afiro	0	0	0.07	0.01	0.18	12	15	0.00	0.00	0.00
agg	20	20	0.09	0.01	1.87	322	54	0.01	0.00	0.00
agg2	0	0	0.10	0.01	2.16	232	58	0.01	0.01	0.06
agg3	0	0	0.10	0.01	2.29	229	58	0.01	0.00	0.06
bandm	54	55	0.08	0.02	0.81	173	223	0.01	0.00	0.05
beaconfd	66	77	0.09	0.02	0.38	124	190	0.00	0.00	0.01
blend	3	3	0.11	0.01	0.27	*				
bnl1	19	22	0.10	0.01	3.17	81	70	0.02	0.00	0.17
bnl2	326	408	0.16	0.02	10.08	767	785	0.06	0.00	0.72
boeing1	7	7	0.08	0.01	2.17	287	418	0.02	0.00	0.10
boeing2	41	41	0.09	0.02	0.71	44	2	0.00	0.00	0.02
bore3d	109	117	0.10	0.02	0.72	159	221	0.00	0.00	0.01
brandy	84	57	0.04	0.00	0.70	100	60	0.01	0.00	0.06
capri	30	60	0.09	0.01	1.15	34	55	0.01	0.00	0.04
cycle	407	379	0.13	0.01	11.39	639	719	0.06	0.00	0.44
czprob	210	439	0.10	0.01	5.49	268	893	0.03	0.00	0.22
d2q06c	39	103	0.12	0.01	26.76	108	400	0.11	0.00	1.53
d6cube	12	741	0.12	0.01	6.07	12	741	0.08	0.00	0.93
degen2	0	0	0.09	0.01	1.10	1	0	0.02	0.00	0.13
degen3	0	0	0.17	0.00	8.17	1	1	0.05	0.00	1.24
dfl001	87	87	0.15	0.01	263.27	*				
e226	9	9	0.09	0.01	0.99	69	25	0.00	0.00	0.05
etamacro	66	147	0.08	0.02	1.96	67	171	0.01	0.00	0.11
fffff800	202	202	0.08	0.00	2.39	205	191	0.03	0.00	0.17
finnis	41	98	0.07	0.01	2.03	136	189	0.01	0.00	0.07
fit1d	0	0	0.11	0.01	2.70	0	2	0.02	0.00	0.10
fit1p	0	0	0.11	0.01	9.85	0	1254	0.05	0.00	0.18

* means the problem can not processed in preprocessing.

Table 5.17: Comparison with CPLEX: Netlib Standard Problem Set (II)

	Results from McIPM					**Results from CPLEX**				
Problem	Erow	Ecol	Tpre	Tpost	Tsol	Erow	Ecol	Tpre	Tpost	Tsol
fit2d	0	0	0.15	0.01	31.69	0	128	0.31	0.00	1.32
fit2p	0	0	0.16	0.00	37.53	0	0	0.11	0.00	1.24
forplan	39	42	0.08	0.00	1.74	*				
ganges	196	196	0.09	0.02	3.07	460	592	0.05	0.00	0.11
gfrd-pnc	26	26	0.07	0.02	1.65	*				
greenbea	95	328	0.13	0.00	26.67	461	1441	0.10	0.10	0.87
greenbeb	97	147	0.16	0.00	26.29	461	1450	0.11	0.11	0.77
grow15	0	0	0.10	0.01	1.90	0	0	0.01	0.00	0.14
grow22	0	0	0.07	0.01	2.75	0	0	0.01	0.00	0.21
grow7	0	0	0.08	0.01	1.11	0	0	0.01	0.00	0.06
israel	0	0	0.06	0.01	1.71	11	1	0.00	0.00	0.06
kb2	0	0	0.11	0.01	0.47	0	5	0.01	0.00	0.01
lotfi	20	20	0.09	0.01	0.53	36	26	0.01	0.00	0.03
maros	57	106	0.11	0.01	3.91	219	531	0.03	0.00	0.21
maros-r7	984	1968	0.24	0.01	30.13	984	2830	0.40	0.01	3.95
modszk1	22	23	0.10	0.01	2.56	29	57	0.02	0.00	0.13
nesm	8	183	0.12	0.01	9.48	16	318	0.04	0.00	0.60
perold	12	94	0.11	0.02	5.07	*				
pilot	18	221	0.17	0.01	44.96	85	436	0.11	0.08	3.33
pilotja	82	383	0.12	0.03	8.98	135	522	0.06	0.00	0.85
pilotwe	4	83	0.10	0.00	8.32	*				
pilot4	10	94	0.10	0.01	4.66	21	190	0.02	0.00	0.27
pilot87	40	269	0.26	0.01	156.98	64	312	0.19	0.00	10.74
pilotnov	111	297	0.08	0.01	5.60	132	391	0.07	0.00	0.53
recipe	20	67	0.06	0.00	0.46	30	96	0.01	0.00	0.02
sc105	1	1	0.10	0.02	0.29	1	0	0.00	0.00	0.01
sc205	2	2	0.07	0.00	0.36	2	1	0.01	0.00	0.02
sc50a	1	1	0.08	0.01	0.23	1	0	0.00	0.00	0.01
sc50b	2	2	0.07	0.00	0.21	2	0	0.00	0.00	0.00
scagr25	2	2	0.10	0.02	0.82	126	4	0.01	0.00	0.05

* means the problem can not processed in preprocessing.

Table 5.18: Comparison with CPLEX: Netlib Standard Problem Set (III)

	Results from McIPM					**Results from CPLEX**				
Problem	Erow	Ecol	Tpre	Tpost	Tsol	Erow	Ecol	Tpre	Tpost	Tsol
scagr7	2	2	0.07	0.11	0.35	36	4	0.01	0.00	0.01
scfxm1	19	19	0.08	0.02	1.18	56	7	0.01	0.00	0.04
scfxm2	38	38	0.07	0.01	2.33	112	74	0.02	0.00	0.12
scfxm3	57	57	0.11	0.01	3.31	168	111	0.03	0.00	0.19
scorpion	27	30	0.08	0.01	0.58	138	82	0.01	0.00	0.02
scrs8	60	60	0.09	0.01	1.85	*				
scsd1	0	0	0.09	0.00	0.46	0	0	0.01	0.00	0.02
scsd6	0	0	0.08	0.02	0.80	0	0	0.00	0.00	0.05
scsd8	0	0	0.06	0.01	1.25	0	0	0.01	0.00	0.09
sctap1	16	16	0.09	0.01	0.87	31	141	0.00	0.00	0.05
sctap2	1057	57	0.11	0.01	2.52	113	554	0.02	0.00	0.16
sctap3	72	72	0.11	0.02	3.84	136	713	0.03	0.00	0.23
seba	1	3	0.06	0.03	4.29	506	1020	0.01	0.00	0.01
share1b	5	5	0.09	0.00	0.79	10	17	0.01	0.00	0.03
share2b	0	0	0.09	0.01	0.33	3	0	0.00	0.00	0.01
shell	49	326	0.09	0.00	2.44	49	367	0.02	0.00	0.09
ship04l	79	62	0.10	0.03	1.65	110	228	0.02	0.00	0.10
ship04s	167	150	0.07	0.00	1.13	186	192	0.01	0.00	0.06
ship08l	148	132	0.14	0.01	3.30	308	1184	0.02	0.00	0.13
ship08s	420	404	0.10	0.02	1.52	502	819	0.02	0.00	0.05
ship12l	395	363	0.13	0.01	5.29	542	1280	0.03	0.00	0.20
ship12s	767	735	0.11	0.00	1.94	812	844	0.02	0.00	0.09
sierra	15	30	0.12	0.00	4.66	*				
stair	0	82	0.10	0.00	1.43	0	83	0.01	0.00	0.12
standata	13	413	0.08	0.00	1.14	67	658	0.01	0.00	0.03
standgub	15	522	0.11	0.01	1.12	69	767	0.01	0.00	0.03
standmps	13	29	0.08	0.00	1.74	73	64	0.01	0.00	0.07
stocfor1	15	15	0.08	0.00	0.36	36	15	0.00	0.00	0.01
stocfor2	177	177	0.11	0.02	5.57	321	177	0.04	0.00	0.22
stocfor3	1313	1313	0.29	0.01	86.47	2331	22293	0.38	0.01	3.16
truss	0	0	0.17	0.02	7.67	0	0	0.05	0.00	0.59
tuff	70	44	0.08	0.01	1.61	86	94	0.01	0.00	0.08
vtpbase	35	74	0.09	0.00	0.73	148	132	0.00	0.00	0.01
wood1p	1	121	0.15	0.00	6.77	74	866	0.10	0.00	0.51
woodw	4	4	0.19	0.00	11.69	543	4395	0.06	0.01	0.36

* means the problem can not processed in preprocessing.

Table 5.19: Comparison with CPLEX: Netlib Infeasible Problem Set

	Results from McIPM				**Results from CPLEX**			
Problem	Erow	Ecol	Tpre	Tsol	Erow	Ecol	Tpre	Tsol
bgdbg1	81	81	0.15	1.01	—		0.00	
bgetam	66	147	0.09	0.75	67	201	0.01	0.10
bgprtr	0	30	0.10	0.21	6	10	0.00	0.00
box1	0	0	0.07	0.25	15	0	0.01	0.00
chemcom	0	0	0.08	0.61	0	96	0.01	0.10
cplex1	0	0	0.08	6.78	—		0.01	
cplex2	0	0	0.07	2.57	0	0	0.01	0.14
ex72a	0	0	0.09	0.25	4	6	0.00	0.00
ex73a	0	0	0.08	0.23	3	6	0.00	0.01
forest6	0	0	0.08	0.29	0	5	0.00	0.01
galenet	1	1	0.03	0.16	—		0.00	
gosh	259	361	0.27	218.99	—		0.23	
gran	—		0.12		—		0.01	
greenbea	92	329	0.17	14.36	—		0.05	
itest2	0	0	0.09	0.12	—		0.00	
itest6	1	2	0.08	0.13	—		0.00	
klein1	0	0	0.09	0.49	0	0	0.00	0.03
klein2	0	0	0.06	7.84	1	0	0.00	86.20
klein3	0	0	0.11	765.79	1	0	0.07	191.52
mondou2	54	137	0.08	1.01	—		0.04	
pang	27	65	0.10	1.75	57	95	0.01	0.18
pilot4i	10	94	0.09	2.42	—		0.04	
qual	18	23	0.09	2.85	93	119	0.01	
reactor	13	15	0.06	1.98	—		0.01	
refinery	20	25	0.07	1.09	95	122	0.00	0.05
vol1	18	23	0.11	2.26	93	119	0.01	0.15
woodinfe	—		0.10		—		0.00	

— means the problem is detected infeasible in preprocessing.

Table 5.20: Comparison with CPLEX: Netlib Kennington Problem Set

	Results from McIPM					**Results from CPLEX**				
Problem	Erow	Ecol	Tpre	Tpost	Tsol	Erow	Ecol	Tpre	Tpost	Tsol
cre-a	435	347	0.15	0.02	11.44	735	129	0.07	0.01	0.77
cre-b	2412	4	0.79	0.02	213.76	4329	40629	0.63	0.02	8.72
cre-c	425	343	0.14	0.01	14.02	745	430	0.06	0.01	0.67
cre-d	2470	20	0.72	0.02	180.34	4842	44792	0.61	0.05	7.05
ken-07	989	989	0.10	0.02	4.93	999	1035	0.04	0.00	0.18
ken-11	4609	4609	0.15	0.01	41.05	4632	4833	0.34	0.03	1.59
ken-13	6098	6098	0.26	0.01	126.88	6115	6326	0.82	0.06	5.65
ken-18	26265	26265	0.72	0.02	836.12	26314	26395	3.17	0.29	30.63
osa-07	37	37	0.30	0.02	25.12	71	934	0.25	0.02	1.14
osa-14	37	37	0.67	0.00	156.90	71	2003	0.64	0.03	3.13
osa-30	37	37	1.09	0.01	271.23	71	3905	1.38	0.05	7.02
osa-60	37	37	2.55	0.03	721.57	71	8841	3.65	?	44.65
pds-02	299	299	0.14	0.02	18.51	357	2421	0.14	0.00	0.45
pds-06	515	515	0.19	0.02	149.14	795	3612	0.58	0.02	5.25
pds-10	580	580	0.31	0.01	508.40	1023	5097	1.07	0.04	14.78
pds-20	927	851	0.59	0.03	3241.03	1819	4630	2.84	0.10	75.51

? means the value can not be found in output.

Table 5.21: Results by McIPM

Problem	Row	Col	Nnz	Rrow	Rcol	Rnnz	Tpre	Tpost	Iter	Tsol
baxter	27441	30733	111576	23893	30184	107362	0.24	0.01	56	1129.2
data	4944	6318	38493	4944	6316	38493	0.12	0.01	18	494.8
gp20	4042	30011	284428	1691	27408	279222	0.33	0.01	29	233.7
l30	2701	18161	66339	2701	18161	66339	0.21	0.02	29	23.9
mod2	34774	66409	199836	29125	56900	157233	0.36	0.01	103	719.6
olp	9932	70891	568060	9880	25800	128000	0.20	0.02	56	13515.4
rail582	582	56097	402290	582	54917	394195	0.51	0.01	31	106.0
route	20894	43019	206782	20894	43019	206782	0.60	0.02	32	271.5
tree9	7686	7938	30618	7686	7938	30618	0.13	0.01	8	2614.2
world	34506	67147	198909	29016	58578	157431	0.40	0.01	103	743.9

Table 5.22: Comparison between McIPM and LIPSOL

	Results from McIPM			**Results from LIPSOL**		
Problem	Rrow	Rcol	Rnnz	Rrow	Rcol	Rnnz
baxter	23893	30184	107362	24386	30733	108521
data	4944	6316	38493	4944	6316	38493
gp20	1691	27408	279222	1691	27408	279222
l30	2701	18161	66339	2701	18161	66339
mod2	29125	56900	157233	34320	64304	195372
olp	9880	25800	128000	9932	25852	128104
rail582	582	54917	394195	582	56097	402290
route	20894	43019	206782	20894	43019	206782
tree9	7686	7938	30618	7686	7938	30618
world	29016	58578	157431	34081	65875	194986

	Results from McIPM			**Results from LIPSOL**		
Problem	Tpre	Tpost	Tsol	Tpre	Tpost	Tsol
baxter	0.24	0.01	1129.23	1.02	0.01	718.03
data	0.12	0.01	494.86	0.19	0.01	747.91
gp20	0.33	0.01	233.74	1.24	0.06	194.36
l30	0.21	0.02	23.98	0.20	0.00	9.06
mod2	0.36	0.01	719.65	1.70	0.09	216.20
olp	0.20	0.02	13515.46	0.92	0.04	14796.78
rail582	0.51	0.01	106.04	1.28	0.02	56.36
route	0.60	0.02	271.53	0.69	0.01	115.01
tree9	0.13	0.01	2614.20	0.12	0.00	2621.26
world	0.40	0.01	743.95	1.67	0.09	357.21

Table 5.23: Preprocessor Comparison

	Results from McPre			**Results from LIPSOL Pre**		
Problem	Rrow	Rcol	Tpre	Rrow	Rcol	Tpre
baxter	23893	30184	0.24	24386	30733	0.94
data	4944	6316	0.12	4944	6316	0.19
gp20	1691	27408	0.33	1691	27408	1.02
l30	2701	18161	0.21	2701	18161	0.19
mod2	29125	56900	0.36	34320	64304	1.88
olp	9880	25800	0.20	9932	25852	0.83
rail582	582	54917	0.51	582	56097	1.14
route	20894	43019	0.60	20894	43019	0.61
tree9	7686	7938	0.13	7686	7938	0.14
world	29016	58578	0.40	34081	65875	1.71

	Results from McPre			**Results from LIPSOL Pre**		
Problem	Tpost	Iter	Tsol	Tpost	Iter	Tsol
baxter	0.01	56	1129.23	0.02	52	840.40
data	0.01	18	494.86	0.02	18	794.22
gp20	0.01	29	233.74	0.05	29	233.78
l30	0.02	29	23.98	0.00	29	25.82
mod2	0.01	103	719.65	0.10	81	786.34
olp	0.02	56	13515.46	0.04	150	36124.10
rail582	0.01	31	106.04	0.02	31	105.37
route	0.02	32	271.53	0.01	32	288.31
tree9	0.01	8	2614.20	0.01	8	2614.63
world	0.01	103	743.95	0.10	91	792.48

Chapter 6

Conclusions and Future Work

We have presented the MPS format and a comprehensive review of preprocessing and postprocessing techniques for LO problems. We also have implemented three subroutines: MPS reader, preprocessing and postprocessing, that are essential parts of the McIPM optimization software package. In the subroutine of preprocessing, we have implemented most of the techniques discussed in Chapter 2. They are: to check infeasible variables; to remove fixed variables, empty rows, empty columns, singleton rows, singleton columns, forcing rows and redundant rows; to find duplicate rows and duplicate columns, and to tighten the bounds of variables. Detailed description of implementation issues are given as well.

We have tested our software package on the Netlib testing sets and compared our results with two commercial software packages of LIPSOL and CPLEX. Based the computational results, we may conclude that the performance of our subroutines is encouraging. The MPS reader subroutine is robust. It can read all the LO problems in the Netlib testing sets while there are some problems can not be read in the other two software packages. It can even read some incorrect MPS files in our testing. The preprocessing subroutine removes more constraints than LIPSOL, however, it can

remove less constraints than CPLEX. The reason is that there are more preprocessing techniques used in CPLEX. In infeasibility problem detection, we detect the same infeasible problems in preprocessing as LIPSOL does, however, CPLEX can detect more. The reason is that due to time limitation, we have not implemented a complete preprocessing project. There are another four preprocessing techniques needed to implement. They are: to tighten the bounds of dual variables; to find dominated variables; to make matrix A sparser, and to have matrix A full rank. Therefore, there is room to further develop our subroutines. Our advantage is the time spent in preprocessing for very large scale problems. For those large scale problems, McPre usually can spend less time than LIPSOL.

Moreover, in our implementation, a MAT file, a DAT file and a TXT file serve as the communication media to transfer the data between MATLAB and C. The reading and writing procedure takes time and it could be avoided if all the subroutines, e.g., preprocessing, postprocessing and solver, are compiled in the same environment. Therefore, to speed up, my suggestion is to integrate the subroutines of preprocessing, solver and postprocessing in one C package. Thus the subroutines can communicate directly without duplicate information.

Bibliography

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

[2] E.D. Andersen. Finding all linearly dependent rows in large-scale linear programming, *Optimization Methods and Software* 6 (1995) pp. 219-227.

[3] E.D. Andersen and K.D. Andersen. Presolving in linear programming, *Mathematical Programming* 71(2) (1995) pp. 221-245.

[4] E.D. Andersen, J. Gondzio, C. Mészáros, and X.J. Xu. Implementation of Interior-Point methods for large scale linear programs, In: T. Terlaky (ed.), *Interior Point Methods of Mathematical Programming*, Kluwer Academic Publishers (1996) pp. 189-245.

[5] A.L. Brearley, G. Mitra, and H.P. Williams. Analysis of mathematical programming problems prior to applying the Simplex Algorithm, *Mathematical Programming* 8 (1975) pp. 54-83.

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

[7] A. Chang and J.K. Reid. On the automatic scaling of matrices for Gaussian elimination, *Journal of the Institute of Mathematics and Its Applications* 10 (1972) pp. 118-124.

[8] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct methods for sparse matrices*, Clarendon Press, Oxford (1986).

[9] I.S. Duff. MA28 - a set of FORTRAN subroutines for sparse unsysmetric linear equations, A.E.R.E. Harwell report 8730, UK (1977)

[10] G. Farkas. *A Fourier-féle mechanikai elv alkalmazásai (in Hungarian).* Matematikai és Természettudományi Értesítő 12 (1894) pp. 457-472.

[11] P.E. Gill, W. Murray, and M.H. Wright. Numerical Linear Algebra and Optimization Volume 1. Redwood City, CA: Addison-Wesley (1991) pp. 337.

[12] A.J. Goldman and A.W. Tucker. Theory of linear programming. H.W. Kuhn and A.W. Tucker Editors, *Linear Inequalities and Related Systems*, Annals of Mathematical Studies, Priceton, New Jersey, No. 38 (1956) pp. 53-97.

[13] J. Gondzio. Presolve analysis of linear programs prior to applying the interior point method, *INFORMS Journal on Computing 9*, No. 1 (1997) pp. 73-91.

[14] J. Gondzio and T. Terlaky. A computational view of interior-point methods for linear programming, J. Beasley (ed.), *Advances in Linear and Integer Programming*, Oxford University Press, Oxford, England (1996) pp. 103-144.

[15] A.J. Hoffman and S.T. McCormick. A fast algorithm that makes matrices optimally sparse, Academic Press, London and New York (1984).

[16] W. Kim, S. Lim, S. Doh, S. Park, and J. Ahn. Numerical aspects in developing LP softwares, LPAKO and LPABO, *Journal of Computional and Applied Mathematics* 152 (2003) pp. 217-228.

[17] I. Maros. *Computational Techniques of the Simplex Method*, Kluwer Academic Publishers (2002).

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

[19] S.T. McCormick. A combinatorial approach to some sparse matrix problems, Ph.D. Thesis, Standford University (1983).

[20] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming*, The McGraw-Hill Companies, Inc. (1996).

[21] J.L. Nazareth. *Computer Solution of linear programs*, Monographs On Numerical Analysis, Oxford Science Publications (1987).

[22] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization, An Interior Point Approach*, John Wiley and Sons, Chichester, UK (1997).

[23] T. Terlaky (Ed.) *Interior point methods of mathematical programming*, Kluwer Academic Publishers (1996).

[24] J.A. Tomlin. On scaling linear programming problems, *Mathematical Programming Study* 4 (1975) pp. 146-166.

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

[26] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*, Kluwer Academic Publishers (1997).

[27] G. Zhang, J. Peng, T. Terlaky, and X. Zhu. Computational experience with self-regular based interior point methods, *Advanced Optimization Laboratory Report 2002/1.* http://www.cas.mcmaster.ca/~oplab/publication/report/2002-1.pdf (2001).

[28] X. Zhu. Implementing the new self-regular proximity based IPMs, *Advanced Optimization Laboratory M.Sc. Thesis* (2001).

[29] X. Zhu, J. Peng, T. Terlaky, and G. Zhang. On implementing self-regular proximity based feasible IPMs, *Advanced Optimization Laboratory Report 2004/3.* http://www.cas.mcmaster.ca/~oplab/publication/report/2004-3.pdf (2004).

[30] Z. Zlatev. *Computational methods for general sparse matrices*, Misistry of the Environment, National Environmental Research Institute, Roskilde, Denmark. Kluwer Academic Publishers (1991).