Computational methods for optimization problems

Doktori (PhD) értekezés

BEKÉNÉ RÁCZ ANETT

Debreceni Egyetem
Természettudományi Doktori Tanács
Informatikai Tudományok Doktori Iskola
Debrecen, 2012

Ezen értekezést a Debreceni Egyetem Természettudományi Doktori Tanács, Informatikai Tudományok Doktori Iskola, Az információ technológia és a sztochasztikus rendszerek elméleti alapjai és alkalmazásai programja keretében készítettem a Debreceni Egyetem természettudományi doktori (PhD) fokozatának elnyerése céljából.

Debrecen, 2012. 03. 21.

.....
Bekéné Rácz Anett
jelölt

Tanúsítom, hogy Bekéné Rácz Anett doktorjelölt 2007-2009 között a fent megnevezett Doktori Iskola, Az információ technológia és a sztochasztikus rendszerek elméleti alapjai és alkalmazásai programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Az értekezés elfogadását javasolom.

Debrecen, 2012. 03. 21.

.....
Dr. Bajalinov Erik
témavezető

Contents

List of Figures

List of Tables

Introduction

Optimization is a basic tool in applied research, engineering, business, and sciences. Operations research (OR) has become a key component of modern-day decision-making in such fields as aerospace, biotechnology, finance, forestry and agriculture, energy, telecommunication, transportation and logistics, the Internet, etc. A survey from 1993 showed that nearly 85 percent of the five hundred largest companies in the United States used optimization models [45].

First application area was the military operations during the II. World War. After the "heroic time"[1] came the "golden age"[1], when many researchers attempt to made linear programming the most frequently used optimization technique not only in military applications but also in science and industry. In those days developed G. B. Dantzig his simplex algorithm [12] which became the most widespread algorithm and also H. W. Kuhn published his Hungarian method [29] based on a research of Jenő Egerváry and Dénes Kőnig for solving assignment problems. Then came the "crisis"[1] period when researchers strived to develop different algorithms such like ellipsoid algorithm by L. G. Khachian [26] (the first polynomial time algorithm) and several variations of interior-point method (most of them based on the Karmarkar's algorithm [23]). More information about the history of linear programming can be found in [13], [27].

The next period in the history of OR is connected with the spread of computers. Performance leaps and continuous improvement of computer hardware might be a reason for the success of optimization algorithms (especially the simplex algorithm). The most important challenge is to use the well-known algorithms efficiently in computational environment by developing powerful codes. Although LP softwares and computers have became much faster, LP models have increased in size. More efficient algorithms and improved implementation techniques are therefore still very important. Large scale problems require enor-

[1] Historical classification from [27].

mous computational effort, and because of floating point arithmetic numerical inaccuracies can occur and also accumulate. These numerical errors can affect not only the final solution but also the number of iterations which can dramatically rise at any time if a small value, which is actually a numerical garbage, was wrongly chosen as a pivot element. These phenomenon has motivated several research on algorithmic techniques that are numerically more stable and also efficient.

Also, we were motivated by the fact that large scale problems to be solved correctly and efficiently usually require such special additional computational techniques as preprocessing and scaling. These techniques very often lead to considerable performance improvement of used solvers.

The main goal of this dissertation is to present the methods developed by myself during my PhD studies for preprocessing optimization problems (especially for Linear-Fractional (LFP) and Integer Programming (IP) models) in order to obtain a more stable and faster solution process.

The dissertation consists of the following parts.

Section 2.1 gives a detailed overview of the most important characteristics of large scale problems from the point of view of performance of computational processes.

In section 2.2 I consider some preprocessing techniques (presolve and scaling), and adapt them to Linear-Fractional problems. Presolve is a special technique developed especially for model reduction and is based on the use of such different operations like fixing variables, finding and removing redundant constraints, and detecting unbounded or infeasible problems. Preprocessing techniques for LP problems published by E. D. Andersen and K. D. Andersen in 1995 [1] motivated researchers to develop further adaptations (like preprocessing for quadratic programming problems by Cs. Mészáros [37]). In section 2.2.1 I discuss the main differences between the presolve and postsolve operations in LP and LFP, and show that well known LP techniques in the case of LFP can not be applied in original form and must be adapted in the corresponding way. Here I also present my rules of presolve and postsolve adapted to LFP problems.

One of the most effective technique of preprocessing is scaling. It's main goal is to transform the model into a numerically more stable form. So in the second part of chapter 2.2 I describe some scaling rules for LFP problems. During our investigations I got acquainted with different variations of scaling methods ([15], [36], [4]), which motivated me to develop more effective scaling algorithms. Section 2.2.2 starts with the theoretical background of scaling in LFP. In the following subsections I give a detailed overview of two well-known scaling rules and describe my new scaling technique. The section ends with

some implementation issue and test result.

The third part (Chapter 3) of this dissertation concerns with other type of optimization problems, namely the Integer Programming (IP) problems. The goal was the same: To improve computational efficiency of a well-known optimization algorithm, the Branch and bound (B&B) method. Section 3.1 focuses on the base frame of the B&B algorithm, describes some branching and searching strategies and also discusses the importance of these rules emphasizing the role of "bound" value. In section 3.2 I give a detailed description of my new method for calculating initial bound for branch-and-bound method. The idea is based on the "ray-method" by V. R. Khachaturov and N. A. Mirzoyan ([24]), which was developed for solving IP problems. This theoretical method in the original form has difficulties concerning computational implementation. So I used the theoretical background and the main idea of Khachaturov's ray-method to get an integer feasible solution for branch-and-bound method. The section ends with a numerical example and test results.

Chapter 1

Experiences on applications of optimization techniques

Nonlinear optimization methods with and without constraints have been frequently used in sciences and engineering for a long time. In this section I briefly overview the applications of optimization techniques in which I took part during the past few years.

Section 1.1 introduces an unconstrained minimization problem which was used as a subproblem for calculating the parameters of isobar analog resonance (IAR) when the optical potentials used are complex ([22]).

Section 1.2 presents some possible application area of LFP in economics.

As we can see in these sections the development of computational power made it possible to handle complicated problems including large matrices, where the number of floating point operations increased dramatically. Although the precision of the representation of numbers has also increased the accumulation of rounding errors sometimes causes difficulties. During these studies I also experienced that large or badly scaled problems require special computational techniques. This fact motivated my investigations on preprocessing techniques.

1.1 Physics

Cooperating with T. Vertse and R. Id Betan we used unconstrained minimization problem when expanding the complex energy shell model (CXSM) calculation for complex potentials. In this type of optimization problem we

minimize a function $F(x)$ $x \in \mathbb{R}^7$ $x = (x_1, x_2, \ldots, x_7)^T$. In our case x represents the vector of seven different parameters to be determined by minimizing the sum of squares of the differences between the complex values of the S_{cc}-matrix, and that of the single pole form $S(E)$ calculated in an adaptation for complex potentials of the framework of the Lane-model using complex energy shell model (CXSM). My role in this research was to select the proper minimum of the function $F(x)$. This minimum corresponds to the isobaric analog resonance (IAR). The IAR is calculated also by using the CXSM, where the IAR belongs to one of the complex eigenvalues resulted by the diagonalization of a complex matrix with size of several hundred.

The values of S_{cc} at each E_i energy were calculated by solving the coupled Lane-equations numerically by using 4th order Runge-Kutta method and matching the solution at a distance R_{as} to the solutions of the Coulomb differential equation. This solution is obtained the standard coupled channel (CC) method and S_{cc} are the result of this method.

Here I have to mentioned that some sort of scaling of the x variables can be performed by using the vector $ESCALE$ which is able to keep balance between the different components in vector x for computational calculations.

The application developed and performed for several IARs confirmed that the CXSM method is able to reproduce the position and the full width of the IAR even in the complex potential case.

In another work of us with P. Salamon and T. Vertse I used unconstrained optimization with a simple non-linear real function of three parameters. While in Lane-equations the nuclear potential with a Woods-Saxon (WS) form was used, which had to be cut at or before R_{as} to be able to determine the element of the S-matrix, in the potential form introduced by Salamon and Vertse (SV potential [43]) the shape approaches zero smoothly without artificial cut-off. In our paper [42] we calculated the complex poles of the S-matrix by varying the strength of the potential well, i.e. the trajectories of the poles on complex energy plane. To make SV potential resemble most to the WS form a minimization of the function $F(x)$ $x \in \mathbb{R}^3$ $x = (x_1, x_2, x_3)^T$ was performed with the same procedure used before. The squares of distances to be minimized were in this case the differences of the two potential form at meshpoints in the range of SV potential. The shapes of the pole trajectories were quite different for the two potential forms, which has an important consequences in nuclear structure calculations.

As a modification of the optimization I introduced a multi-objective minimization, which forced also the derivatives of the shapes of the two potential forms to be similar as much as possible. The results of this optimization will be

used in a coming publication of us [14].

1.2 Economics

In these section I would like to show the importance of Linear-Fractional Programming in real-world applications. Since [35] LFP problems appeared in a wide range of research papers. Researchers use the linear fractional approach in well known optimization problems like network optimization ([44]) or transportation problems ([46]).

Before I started to study this type of optimization problems I investigated the application areas ([6], [7]). While in case of LP one can optimize a result of some particular activity, like maximize the profit gained by the company or minimize the cost of production or transportation, etc. Objective of LFP problems reflects (in contrast of LP) the profit per unit of expenditure, cost per unit of produced goods or cost of transportation per unit of transported goods, etc.

We can use these two aspects on the same problem and we have to keep in our mind that these two objective functions (one of them is linear and the second one is linear-fractional) may lead to different optimal solutions.

To decide which type of objective we have to apply we can find suggestions in [7] and also in [6]. E. Bajalinov showed that although profit maximization and maximization of efficiency lead to different optimal solutions we can utilize these different optimal solutions in order to obtain some more profit. He presents a decision process in which we decide to apply the LP optimum, the LFP optimum or a combination of them depending on the cost and the amount of money the company can spend. Using these rules from [7] we can reach more profit than in case of a simple LP optimization.

Chapter 2

Large scale problems

Here and in what follows we consider the following Linear Programming (LP) and Linear-Fractional Programming (LFP) problems:

$$P(x) = \sum_{j=1}^{n} p_j x_j \to min$$

subject to

$$\sum_{j=1}^{n} \underbrace{a_{ij}}_{A=||a_{ij}||_{m\times n}} = b_i \quad i = 1, \ldots, m$$

$$x_j \geq 0 \quad j = 1, \ldots, n.$$

$$Q(x) = \frac{P(x)}{D(x)} \to min$$

subject to

$$\sum_{j=1}^{n} \underbrace{a_{ij}}_{A=||a_{ij}||_{m\times n}} = b_i \quad i = 1, \ldots, m$$

$$x_j \geq 0 \quad j = 1, \ldots, n,$$

where $P(x) = \sum\limits_{j=1}^{n} p_j x_j + p_0$, $D(x) = \sum\limits_{j=1}^{n} d_j x_j + d_0$ and $D(x) > 0$.

2.1 Main characteristics of large scale problems

The real-life LP models tend to be large in size. A reason can be the requirement to create more realistic models. Not only the size of a large scale model can make a problem "hard" but also sparsity, special structure and other properties may affect the performance and stability of algorithms and procedures used to solve them. We can also define a measure to show how "well or badly" scaled a matrix is. The aim of this chapter is to present and to mark the characteristics which can influence the computational effort required for solving a problem. The structure of this part is based on [36].

2.1.1 Size

Size can be defined by the number of the elements of constraint matrix A or the number of its columns and rows, i.e. variables and constraints. Obviously the sense of term "large-scale" is not the same in the 1960s and nowadays. While 50 years ago problems only with several hundred rows and columns could be handled, today we can read reports about solving problems with hundred of thousands of variables and constraints.

2.1.2 Sparsity

Constraint matrices of large scale problems typically have quite a lot zero entries. This feature is called sparsity. Moreover LP problems often have a special structure by having sometimes identical blocks in the constraint matrix A. Some of the operations research software tools can visualize the structure of the problem to be solved. Usually such software tools generate some graphical representation like shown in figure 2.1.

Numerically the sparsity is usually expressed as a ratio of the nonzero and the total number of elements, that is

$$\varrho(A) = \frac{\nu_A}{mn},$$

where ν is the number of nonzero elements and $A \in \mathbb{R}^{m \times n}$. Processing matrices with the same size but different sparsity does not require the same computational effort. Most of the problem collections (e.g.: NETLIB) available on the Internet provide information about problem's size (m, n) and its sparsity (ν_A) in a well defined classification.

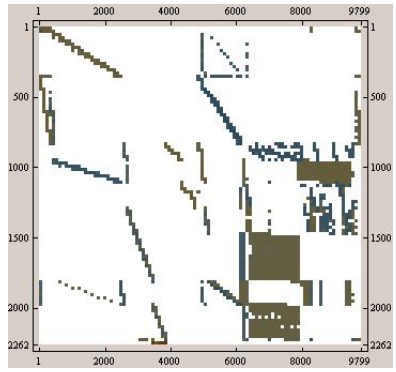


Figure 2.1: Color coded matrix plot of 80BAU3B.mps from NETLIB

2.1.3 Fill-in

Let us consider an LP problem in the following form:

$$p^T x \to min \tag{2.1}$$

subject to

$$Ax = b \tag{2.2}$$

$$x \geq 0, \tag{2.3}$$

where $A \in \mathbb{R}^{m \times n}$ contains an $m \times m$ unit matrix, so it is of full row rank and $m < n$. Any combination of m independent column vectors from A is called basis ($B = \{A_j : j \in J_B\}$) and the corresponding variables are called basic variables. A basic solution $x = (x_1, x_2, \ldots, x_n)$ can be defined in the following way:

$$\begin{cases} x_j = 0, & \text{if } x_j \text{ is not a basic variable} \\ x_j \text{ is defined from the system } \sum_{j \in J_B} A_j x_j = b & \text{if } x_j \text{ is a basic variable} \end{cases}$$

Simplex algorithm is an iterative algorithm which checks the optimality of the current basis solution moving from basis to basis until termination. Two bases, formed by columns of A, are called neighboring if they differ from one another only in one column. Changing the basis means to replace the current $B = \{A_{s_1}, A_{s_2}, \ldots, A_{s_p}, \ldots, A_{s_m}\}$ basis with a neighboring one $B' = \{A_{s_1}, A_{s_2}, \ldots,$

$A_{s_p-1}, A_k, A_{s_p+1}, \ldots, A_{s_m}\}$. Since A_k is also a column of A (m dimensional vector) it can be expressed as a linear combination of the vectors of B:

$$A_k = \sum_{i=1}^{m} \upsilon_i A_{s_i}$$

On the other hand, vector A_{s_p} in the new basis B' can be represented as $A_{s_p} = B'\eta$, where

$$\eta = (-\frac{\upsilon_1}{\upsilon_p}, \ldots, -\frac{\upsilon_{p-1}}{\upsilon_p}, \frac{1}{\upsilon_p}, -\frac{\upsilon_{p+1}}{\upsilon_p}, \ldots, -\frac{\upsilon_m}{\upsilon_p})^T$$

If we define matrix E as

$$E = [e_1, \ldots, e_{p-1}, \eta, e_{p+1}, \ldots, e_m],$$

where e_j denotes the jth column of the unit matrix, then all other basis vectors from B can be expressed in the terms of the new basis B' in the following way:

$$B = B'E \quad \text{or} \quad B'^{-1} = EB^{-1}.$$

Fill-in is such a phenomenon when nonzeros appear in positions of zero elements in matrix A during elementary transformations with matrix E after every basis change. Let us consider how matrix E transforms a column vector denoted by $A_{s_p} = (a_1, a_2, \ldots, a_m)^T$. Let us suppose that vector η is in position p in matrix E and we rewrite equality $EA_{s_p} = \overline{A}_{s_p}$ in the following form

$$\begin{bmatrix} 1 & & \eta^1 & & \\ & \ddots & \vdots & & \\ & & \eta^p & & \\ & & \vdots & \ddots & \\ & & \eta^n & & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_p \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} a_1 + a_p\eta^1 \\ \vdots \\ a_p\eta^p \\ \vdots \\ a_m + a_p\eta^m \end{bmatrix}.$$

It is obvious that if $a_p = 0$ then $A_{s_p} = \overline{A}_{s_p}$, i.e. no operation is needed to obtain the result, software can use a simple logical test to recognize this case. If A is a large and sparse matrix there is a good chance that $a_p = 0$.

On the other hand, if $a_p \neq 0$ then the operation creates new nonzeros in the positions where zeros were before. Formally, denote the index set of nonzeros in η by $\mathcal{E}$ and in A_{s_p} by $\mathcal{A}$. In this case the positions of newly created nonzeros in $\overline{A}_{s_p}$ are in set $|\mathcal{E}\backslash\mathcal{A}|$. We can see that the number of new nonzeros depends on the density of vector η (here suppose that $a_p \neq 0$).

It is obvious if there appears a small numerical inaccuracy in value of a_p it can cause many other errors during basis transformation.

2.1.4 Measure of scaling

It is quite difficult to define which matrices are well scaled. The opposite is somewhat easier. We can say a matrix is badly scaled if the magnitude of nonzero elements are hugely different. Such matrices are quite difficult to process in practice. Using the previous definition we can say that a matrix is well scaled if the magnitudes of its nonzero elements are close to each other. So the expectation for a scaling method is to reduce the spread of magnitudes as well as it can. To reach this aim we can scale columns and rows as many times as we need (or as possible).

For measuring the spread of magnitudes of nonzero elements of matrix A Fulkerson and Wolfe [15] (and also Orchard-Hays) recommended

$$\sigma(A) \;=\; \frac{max|a_{ij}|}{min|a_{ij}|}, \;\; over\ all\ a_{ij} \neq 0, \tag{2.4}$$

and say that a matrix is well scaled if quantity $\sigma(A)$ is not larger than a threshold value of about $10^6 - 10^8$.

2.2 Preprocessing in Linear-Fractional programming

The real-world LP and also LFP models tend to be large in size. In practice large-scale problems are not solved in the form they are originally formulated. There are numerous operations that can be used to increase the reliability and efficiency of solving. These operations are called preprocessing. Presolve algorithms are a part of preprocessing and they are also used for reducing the size by eliminations, improving the numerical characteristics, detecting infeasibility or unboundedness, fixing as many variables as possible without solving the problem. Here we can mention the checks being used for detecting the weaknesses of a model, such as empty rows/columns or redundant constraints which can be removed (if it is possible) in order to reduce the size. Most LP solvers contains some presolve algorithms, and provide us different options to set the parameters. For instance the most well known of them, IBM CPLEX, executes some presolve operation whereby the problem input by users is examined for logical reduction opportunities. The goal is to reduce the size, which typically translates to a reduction in total run time (even including the time spent for presolve itself). CPLEX allows us to set the *Presolve switch* parameter, CPX_PARAM_PREIND. It decides whether CPLEX applies presolve

during preprocessing or not. The following values are the possible options [50]:

Value	Meaning
0	Do not apply presolve
1	Apply presolve (default)

Table 2.1: Presolve settings in CPLEX

CPLEX also reports about the presolve operations executed when the problem has already been solved. In the output we can see information about how many rows and columns were eliminated.

Also another professional solver, LINGO, provides us an option to set on or off the model reduction. In LINGO API we can set the value of *REDUCE* parameter. In graphical user interface we can use the appropriate combobox as it can be seen on the following figure.

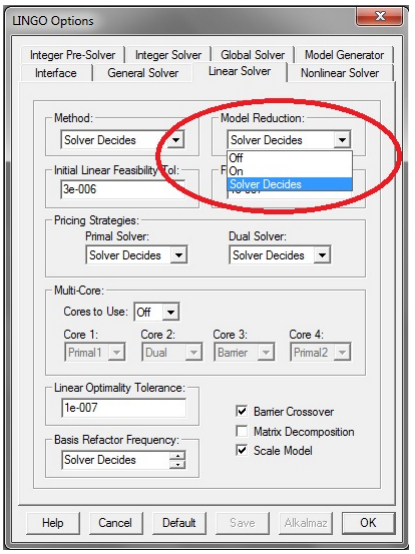


Figure 2.2: LINGO model reduction settings

2.2.1 Presolve

Preprocessing is very important for solving large optimization problems irrespectively of using interior point or simplex algorithm. Most of the professionally developed solvers (like IBM CPLEX, LINGO, GUROBI, etc.) automatically use preprocessing techniques to maintain numerical stability and improve performance. Even though computers have become even faster, the real-world models have increased in size. The reason can be the growing complexity requirements and the new sophisticated model generators too. The aim of preprocessing is to reduce the problem size, find redundancy and detect the unbounded or infeasible problems. When performing these manipulations some data from the dual problem may be lost or distorted. To restore such data some special postsolve operations are required.

In this section I describe the main results of my results connected with preprocessing techniques in LFP. My investigations are based on well-known preprocessing techniques used in LP. I adapt these techniques to LFP problems. Some of these preprocessing algorithms can be used in LFP without any changes, but the others have to be modified. In certain cases this adaptation is not trivial. Not only preprocessing but also postsolve techniques are different in the case of linear fractional programming problems. I present some preprocessing techniques with the corresponding postsolve operations based on [1], [37] and gives an overview of its adaptation into LFP.

In this section I consider some presolve techniques and present how I adapted them to LFP. First let us consider an LFP problem in the following form:

$$Q(x) = \frac{P(x)}{D(x)} \to \min \tag{2.5}$$

subject to

$$\overline{b}_i \geq \sum_{j=1}^{n} a_{ij}x_j \geq \underline{b}_i, \quad i = 1, 2, \ldots, m, \tag{2.6}$$

$$U_j \geq x_j \geq L_j, \quad j = 1, 2, \ldots, n, \tag{2.7}$$

where

$$P(x) = \sum_{j=1}^{n} p_j x_j + p_0, \qquad D(x) = \sum_{j=1}^{n} d_j x_j + d_0,$$

and $D(x) > 0, \quad \forall x = (x_1, x_2, \ldots, x_n)^T \in S$. S is the feasible set defined by constraints (2.6) and (2.7).

And its dual:

$$y_0 \to \max \tag{2.8}$$

$$d_0 y_0 - \sum_{i=1}^{m} \underline{b}_i s_i + \sum_{i=1}^{m} \overline{b}_i q_i - \sum_{j=1}^{n} L_j w_j + \sum_{j=1}^{n} U_j v_j \geq p_0 \tag{2.9}$$

$$d_j y_0 + \sum_{i=1}^{m} a_{ij} y_i + w_j - v_j = p_j \quad j = 1, 2, \ldots, n \tag{2.10}$$

$$y_i - s_i + q_i = 0, \quad i = 1, 2, \ldots, m \tag{2.11}$$

$$s, q, v, w \geq 0 \tag{2.12}$$

where $v = (v_1, \ldots, v_n)$, $w = (w_1, \ldots, w_n)$, $s = (s_1, \ldots, s_m)$ and $q = (q_1, \ldots, q_m)$ are vectors of dual variables. I={1,. . . ,m} and J={1,. . . ,n} are the index sets of constraints and variables.

2.2.1.1 Empty rows and columns

If row i_0 in (2.6) has zero entries in all positions $j = 1, 2, \ldots, n$, it can be removed from an LP model as well as from an LFP model too. The postsolve operation is very simple, because the optimal basis of the transformed problem is the optimal basis of the original problem too. We have to calculate only the corresponding slack variable. The dual values connected to this row are equal to zero. ($y_{i_0} = 0,\ p_{i_0} = 0,\ q_{i_0} = 0$)

Proof: Let x_j^*, v_j^*, w_j^* ($j \in J$) and $y_i^* = s_i^* - q_i^*$ ($i \in I \backslash i_0$), y_0^* be the optimal solution of the presolved problem (without row i_0) and its dual. It is obvious that if we insert a new row to the problem which has no nonzero entries, the optimal basis does not change. Dual constraints have to be expanded in the following a way:
Instead of

$$d_0 y_0^* - \sum_{i \in I \backslash i_0} \underline{b}_i s_i^* + \sum_{i \in I \backslash i_0} \overline{b}_i q_i^* - \sum_{j=1}^{n} L_j w_j^* + \sum_{j=1}^{n} U_j v_j^* = p_0$$

the dual model contains

$$d_0 y_0^* - \sum_{i \in I \backslash i_0} \underline{b}_i s_i^* \ - \underline{b}_{i_0} s_{i_0} \ + \sum_{i \in I \backslash i_0} \overline{b}_i q_i^* + \overline{b}_{i_0} q_{i_0} - \sum_{j=1}^{n} L_j w_j^* + \sum_{j=1}^{n} U_j v_j^* = p_0,$$

and instead of

$$d_j y_0^* + \sum_{i \in I \setminus i_0} a_{ij} y_i^* + w_j^* - v_j^* = p_j \quad j = 1, 2, \ldots, n$$

it contains

$$d_j y_0^* + \sum_{i \in I \setminus i_0} a_{ij} y_i^* + \; a_{i_0 j} y_{i_0} \; + w_j^* - v_j^* = p_j \quad j = 1, 2, \ldots, n.$$

It is obvious that the values of q_{i_0} and s_{i_0} are equal to zero. In case of dual constraints (2.10) actually it does not require any changes because $a_{i_0 j} = 0, \; \forall j \in J$.

We have to keep in mind that before deleting row i_0 we have to check primal feasibility. The problem is primal infeasible if $\underline{b}_{i_0} > 0$ or $\overline{b}_{i_0} < 0$.

It is a bit more complicated situation if a column (denoted by l) has not nonzero entries. In case of LP we have to fix the variable x_l at its upper or lower bound depending on the sign of its coefficient in the objective function. In case of LFP we have to determine which value is better from the point of view of the objective function (2.5):

$$\frac{\sum\limits_{j \in J \setminus l} p_j x_j + p_l L_l + p_0}{\sum\limits_{j \in J \setminus l} d_j x_j + d_l L_l + d_0}$$

or

$$\frac{\sum\limits_{j \in J \setminus l} p_j x_j + p_l U_l + p_0}{\sum\limits_{j \in J \setminus l} d_j x_j + d_l U_l + d_0}$$

where U_l and L_l are the individual bounds of variable x_l (see: (2.7)).

Let us denote the value

$$\sum_{j \in J \setminus l} p_j x_j + p_0 \text{ by } P'(x) \quad \text{and} \quad \sum_{j \in J \setminus l} d_j x_j + d_0 \text{ by } D'(x).$$

We have to compare the following two expressions and choose the better one:

$$\frac{P'(x) + p_l L_l}{D'(x) + d_l L_l} \quad \diamond \quad \frac{P'(x) + p_l U_l}{D'(x) + d_l U_l} \tag{2.13}$$

where $\diamond$ denotes the relational sign between the two side. (It can be $>$ or $<$ since we suppose that $L_l \neq U_l$).

Since we supposed that $D(x) > 0$, $\forall x \in S$, relation (2.13) can be written in the following form:

$$(P'(x) + p_l L_l)(D'(x) + d_l U_l) \quad \diamond \quad (P'(x) + p_l U_l)(D'(x) + d_l L_l)$$

After simplification we obtain:

$$L_l(D'(x)p_l - P'(x)d_l) \quad \diamond \quad U_l(D'(x)p_l - P'(x)d_l)$$

Since $U_l > L_l$ relation $\diamond$ depends on the sign of $D'(x)p_l - P'(x)d_l$. If it is negative, i.e. $D'(x)p_l - P'(x)d_l < 0$, relation $\diamond$ is $>$ so we have to fix the variables x_l at its upper bound, $x_l = U_l$. But if $D'(x)p_l - P'(x)d_l > 0$ then $\diamond$ is $<$, so the right way is to fix the variable at its lower bound, $x_l = L_l$. Unfortunately, when presolving we do not know the sign of $D'(x)p_l - P'(x)d_l$. This is why we have to analyze the signs of p_l and d_l and distinguish the cases described below in Theorem 2.1.

Let us introduce the following expressions:

$$\overline{P'(x)} = \sum_{j \in J \setminus l, p_j > 0} p_j U_j + \sum_{j \in J \setminus l, p_j < 0} p_j L_j + p_0$$

$$\underline{P'(x)} = \sum_{j \in J \setminus l, p_j > 0} p_j L_j + \sum_{j \in J \setminus l, p_j < 0} p_j U_j + p_0$$

$$\overline{D'(x)} = \sum_{j \in J \setminus l, d_j > 0} d_j U_j + \sum_{j \in J \setminus l, d_j < 0} d_j L_j + d_0$$

$$\underline{D'(x)} = \sum_{j \in J \setminus l, d_j > 0} d_j L_j + \sum_{j \in J \setminus l, d_j < 0} d_j U_j + d_0$$

Theorem 2.1 *Here are the following four possible cases to fix a variable (with an empty column) in problem (2.5)-(2.7) in order to minimize the objective value.*

[1] **Case** $p_l > 0$ **and** $d_l > 0$

(a) If $\overline{P'(x)}d_l < \underline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l > 0$. So we have to choose the lower bound.

(b) Or $\underline{P'(x)}d_l > \overline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l < 0$. So we have to choose the upper bound.

[2] **Case** $p_l > 0$ **and** $d_l < 0$

(a) If $\underline{P'(x)}d_l < \underline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l > 0$. So we have to choose the lower bound.

(b) Or $\overline{P'(x)}d_l > \overline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l < 0$. So we have to choose the upper bound.

[3] **Case** $p_l < 0$ **and** $d_l > 0$

(a) If $\overline{P'(x)}d_l < \overline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l > 0$. So we have to choose the lower bound.

(b) Or if $\underline{P'(x)}d_l > \underline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l < 0$. So we have to choose the upper bound.

[4] **Case** $p_l < 0$ **and** $d_l < 0$

(a) If $\underline{P'(x)}d_l < \overline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l > 0$. So we have to choose the lower bound.

(b) Or $\overline{P'(x)}d_l > \underline{D'(x)}p_l$, then $D'(x)p_l - P'(x)d_l < 0$. So we have to choose the upper bound.

After fixing a variable in the way shown in Theorem 2.1 we obtain the following problem

$$Q'(x) = \frac{\sum\limits_{j \in J \setminus l} p_j x_j + p_0 + U_l p_l}{\sum\limits_{j \in J \setminus l} d_j x_j + d_0 + U_l d_l} \to \min \tag{2.14}$$

for variable x_l fixed at upper bound and

$$Q'(x) = \frac{\sum\limits_{j \in J \setminus l} p_j x_j + p_0 + L_l p_l}{\sum\limits_{j \in J \setminus l} d_j x_j + d_0 + L_l d_l} \to \min \tag{2.15}$$

for variable x_l fixed at lower bound. Subject to

$$\begin{gathered} \overline{b}_i \geq \sum_{j \in J \setminus l} a_{ij} x_j \geq \underline{b}_i \\ U_j \geq x_j \geq L_j, \quad j \in J \setminus l \ . \end{gathered}$$

Let us suppose that problem (2.14) has the following optimal solution: $x_j^*,\ j \in J\backslash l$ and $y_i^*,\ i \in I$ and y_0^* denote the optimal solution of its dual. When postsolving we have to define only the values of dual variables w_l and v_l removed earlier.

This fixing of variable x_l results the following changes in the dual problem:

[1] Instead of original

$$d_0 y_0^* - \sum_{i\in I} \underline{b}_i s_i^* + \sum_{i\in I} \overline{b}_i q_i^* - \sum_{j\in J\backslash l} L_j w_j^* - L_l w_l + \sum_{j\in J\backslash l} U_j v_j^* + U_l v_l = p_0.$$

we have

$$(d_0 + U_l d_l) y_0^* - \sum_{i\in I} \underline{b}_i s_i^* + \sum_{i\in I} \overline{b}_i q_i^* - \sum_{j\in J\backslash l} L_j w_j^* + \sum_{j\in J\backslash l} U_j v_j^* = p_0 + U_l p_l$$

[2] Also constraint

$$d_l y_0^* + \sum_{i\in I} a_{il} y_i + w_l - v_l = p_l$$

was removed.

According to the complementary slackness theorem $w_l = 0$, because its primal constraint pair, $x_l^* > L_l$ and the other variable $v_l = d_l y_0^* - p_l$. In the case of problem (2.15) we have $w_l = p_l - d_l y_0^*$ and $v_l = 0$.

2.2.1.2 Singleton rows

Row i_0 in (2.6) is said to be singleton if it has only one nonzero entry (let us denote this position by l). In fact it results a new individual bound for variable x_l.

If this new bound is not stricter then the original individual bound for variable x_l we can remove this constraint without any other changes in the model. In this case the constraint is redundant, i.e.

$$x_l \geq L_l \geq \frac{\underline{b}_{i_0}}{a_{i_0 l}} \text{ or } x_l \leq U_l \leq \frac{\overline{b}_{i_0}}{a_{i_0 l}}.$$

Otherwise, we remove the singleton row from the problem and replace the individual bounds of x_l with L_l' and U_l'. After solving the preprocessed problem

we obtain x_j^*, v_j^*, w_j^*, y_i^*,s_i^*, q_i^* as optimal values, where $j \in J$ and $i \in I \backslash i_0$. During postsolve we have to define the value of dual variable y_{i_0}, and also s_{i_0}, q_{i_0} and the values of v_l and w_l are also different from the preprocessed ones. The following table shows how we can construct the new individual bounds for x_l in the preprocessed problem and also the rules of postsolve operation.

Individual bound in preprocessed problem	Rules of postsolve
$L_l' = \begin{cases} L_l \text{ if } L_l \geq \dfrac{\underline{b}_{i_0}}{a_{i_0 l}} \\ \dfrac{\underline{b}_{i_0}}{a_{i_0 l}} \text{ if } L_l < \dfrac{\underline{b}_{i_0}}{a_{i_0 l}} \end{cases}$	$s_{i_0} = 0$ $s_{i_0} = \dfrac{w_l^*}{a_{i_0 l}} \quad w_l = 0$
$U_l' = \begin{cases} U_l \text{ if } U_l \leq \dfrac{\overline{b}_{i_0}}{a_{i_0 l}} \\ \dfrac{\overline{b}_{i_0}}{a_{i_0 l}} \text{ if } U_l > \dfrac{\overline{b}_{i_0}}{a_{i_0 l}} \end{cases}$	$q_{i_0} = 0$ $q_{i_0} = \dfrac{v_l^*}{a_{i_0 l}} \quad v_l = 0$

There are some special cases. During this step of preprocessing we can detect infeasibility or we can fix the corresponding variable.

$$\text{Problem is infeasible, if: } \begin{cases} \dfrac{\underline{b}_{i_0}}{a_{i_0 l}} > U_l \text{ or} \\ \dfrac{\overline{b}_{i_0}}{a_{i_0 l}} < L_l \text{ or} \\ \dfrac{\underline{b}_{i_0}}{a_{i_0 l}} = U_l \text{ and } \dfrac{\overline{b}_{i_0}}{a_{i_0 l}} = L_l \text{ but } L_l \neq U_l \end{cases}$$

$$\text{We can fix variable } x_l \text{ if: } \begin{cases} \dfrac{\underline{b}_{i_0}}{a_{i_0 l}} = U_l & \text{at its upper bound} \\ \dfrac{\overline{b}_{i_0}}{a_{i_0 l}} = L_l & \text{at its lower bound} \end{cases}$$

For these special cases I defined the following postsolve operations:

- In case of fixing at upper bound

$$s^*_{i_0} = \frac{p_l - d_l y^*_0 - \sum_{i \in I \setminus i_0} a_{il} y^*_i}{a_{i_0 l}}$$
$$q^*_{i_0} = 0$$
$$w^*_l = 0, v^*_l = 0$$

- In case of fixing at lower bound

$$q^*_{i_0} = \frac{-p_l + d_l y^*_0 + \sum_{i \in I \setminus i_0} a_{il} y^*_i}{a_{i_0 l}}$$
$$s^*_{i_0} = 0$$
$$v^*_l = 0, w^*_l = 0$$

2.2.1.3 Primal feasibility test

This test is exactly the same in LFP as in LP, because it does not depend on the objective function. It is based only on the individual bounds and constraints. Let us define

$$\overline{b}'_i = \sum_{k \in J_i^+} a_{ik} U_k + \sum_{k \in J_i^-} a_{ik} L_k$$

$$\underline{b}'_i = \sum_{k \in J_i^+} a_{ik} L_k + \sum_{k \in J_i^-} a_{ik} U_k$$

where $J_i^+ = \{j | a_{ij} > 0\}$ and $J_i^- = \{j | a_{ij} < 0\}$. U_k, L_k are individual bounds (see (2.7)).

Primal infeasibility is detected if $\exists i \in I$, where $\underline{b}'_i > \overline{b}_i$ or $\overline{b}'_i < \underline{b}_i$.

Redundancy is detected if $\overline{b}_i \geq \overline{b}'_i \geq \underline{b}'_i \geq \underline{b}_i$. In this case we simply remove the redundant constraint, and the corresponding postsolve operation is $s^*_i = 0,\ q^*_i = 0,\ y^*_i = 0$.

Variables included by row i_0 can be fixed at their individual bounds if $\underline{b}'_{i_0} = \overline{b}_{i_0}$ or $\overline{b}'_i = \underline{b}_{i_0}$. The preprocessed problem does not contain row i_0, variables $x_j, j \in J_{i_0}^+ \cup J_{i_0}^-$, their individual bounds and therefore the dual constraints

connected to these variables any longer. Let us denote $J_{i_0} = \{j | a_{i_0 j} \neq 0\}$, i.e. $J_{i_0} = J_{i_0}^- \cup J_{i_0}^+$. If J_{i_0} has only one item, it is exactly the case of singleton row (see in section 2.2.1.2). Otherwise, the changes in the preprocessed problem are:

$$\begin{aligned}
p'_0 &= p_0 + \sum_{j \in J^+} p_j U_j + \sum_{j \in J^-} p_j L_j \\
d'_0 &= d_0 + \sum_{j \in J^+} d_j U_j + \sum_{j \in J^-} d_j L_j \\
\overline{b}'_i &= \overline{b}_i - \sum_{j \in J^+} a_{ij} U_j - \sum_{j \in J^-} a_{ij} L_j \quad i \in I \backslash i_0 \\
\underline{b}'_i &= \underline{b}_i - \sum_{j \in J^+} a_{ij} U_j - \sum_{j \in J^-} a_{ij} L_j \quad i \in I \backslash i_0
\end{aligned}$$

Optimal solution of the preprocessed problem and its dual are $y_0^*, s_i^*, q_i^*, w_j^*, v_j^*$ and $y_i^*,\ i \in I \backslash i_0, j \in J \backslash J_{i_0}$.

During postsolve we define dual variables in the following way: $y_{i_0}^* = s_{i_0}^* = q_{i_0}^* = 0$.

- In case of $\overline{b}'_{i_0} = \underline{b}_{i_0}$:

$$w_j = 0 \text{ and } v_j = d_j y_0^* + \sum_{i \in I \backslash i_0} a_{ij} y_i^* - p_j \text{ if } j \in J_{i_0}^+ \tag{2.16}$$

$$v_j = 0 \text{ and } w_j = -d_j y_0^* - \sum_{i \in I \backslash i_0} a_{ij} y_i^* + p_j \text{ if } j \in J_{i_0}^- \tag{2.17}$$

- In case of $\underline{b}'_{i_0} = \overline{b}_{i_0}$

$$v_j = 0 \text{ and } w_j = -d_j y_0^* - \sum_{i \in I \backslash i_0} a_{ij} y_i^* + p_j \text{ if } j \in J_{i_0}^+ \tag{2.18}$$

$$w_j = 0 \text{ and } v_j = d_j y_0^* + \sum_{i \in I \backslash i_0} a_{ij} y_i^* - p_j \text{ if } j \in J_{i_0}^- \tag{2.19}$$

2.2.1.4 Cheap dual test

Using cheap dual test we can remove further columns from the model. I use the cases (1-4) and subcases signed by (a) and (b) from Theorem 2.1 in order to remove column l as follows:

If any subcase signed by (a) is true and $\begin{cases} \overline{b}_i = +\infty \text{ if } a_{il} < 0 \\ \underline{b}_i = -\infty \text{ if } a_{il} > 0 \end{cases}$ $i \in I$,

variable x_l can be fixed at its lower bound.

If any subcase signed by (b) is true and $\begin{cases} \overline{b}_i = +\infty \text{ if } a_{il} > 0 \\ \underline{b}_i = -\infty \text{ if } a_{il} < 0 \end{cases}$ $i \in I$.

we fix variable x_l at its upper bound.

In these cases rows with nonzeros in position l can be removed from the original problem and the corresponding postsolve operations are similar to that in primal feasibility test. The dual variable for the rows removed will be $y_i^* = 0,\ i \in I_j$, where $I_j = \{i \in I | a_{il} \neq 0\}$. In case of fixing variable x_l at its upper bound formula (2.16) is used. In case of lower bound formula (2.17) defines how to calculate w_l^* and v_l^*.

It may happen that the appropriate bound is not finite, then the problem is dual infeasible, because the primal objective function is not finite on the feasible set.

There is a special case when the appropriate bound of x_l is not finite but coefficients p_l and d_l are zeros. In this case the constraints which have nonzero entry in position l can be removed and the value of variable x_l is undefined.

2.2.1.5 Duplicate rows

If there are such two rows (row i and k) in (2.6) that $a_{ij} = \alpha a_{kj}, \forall j \in J$ and $i \neq k$ then rows i and k are identical up to a scalar multiplier α. Depending on bounds primal infeasibility or redundancy can be detected.

Infeasibility is detected if $\overline{b}_i < \dfrac{\underline{b}_k}{\alpha}$ or $\dfrac{\overline{b}_k}{\alpha} < \underline{b}_i$.

Otherwise redundancy is detected, and a new row i_0 is need to be created instead of the two original rows i and k by tightening bounds in the following way:

$$\underline{b}_{i_0} = \max\{\underline{b}_i\ ;\ \frac{\underline{b}_k}{\alpha}\}$$
$$\overline{b}_{i_0} = \min\{\overline{b}_i\ ;\ \frac{\overline{b}_k}{\alpha}\}$$

Postsolve in this situation is obvious. We have to define the following dual variables $y_i = s_i - q_i$ and $y_k = s_k - q_k$ using the following formulas:

- $s_i^* = s_{i_0}^*$ and $s_k^* = 0$ if $\underline{b}_{i_0} = \underline{b}_i$,
- $s_k^* = s_{i_0}^*$ and $s_i^* = 0$ otherwise.
- $q_i^* = q_{i_0}^*$ and $q_k^* = 0$ if $\overline{b}_{i_0} = \overline{b}_i$,
- $q_k^* = q_{i_0}^*$ and $q_i^* = 0$ otherwise.

where $s_{i_0}^*$ and $q_{i_0}^*$ are the optimal dual values of the new row i_0.

2.2.1.6 Presolve Linear-Fractional or Linear Analogue Problem

In this section I show the differences between presolving an LFP problem and its Linear Analogue (LA) form obtained by Charles and Cooper transformation [11]. I reveal the reasons why it is more efficient to presolve the original form of an LFP problem (before transformation) instead of presolve the LA form after transformation.

First, let us consider the linear analogue form of LFP problem (2.5)-(2.7):

$$C(t) = \sum_{j=0}^{n} p_j t_j \to min \tag{2.20}$$

subject to

$$\begin{array}{rl} & \sum_{j=1}^{n} a_{ij} t_j - \overline{b}_i t_0 \quad \leq 0 \quad i \in I \\ 0 \leq & \sum_{j=1}^{n} a_{ij} t_j - \underline{b}_i t_0 \qquad\quad i \in I \end{array} \tag{2.21}$$

$$\begin{array}{rl} & t_j - U_j t_0 \quad \leq 0 \quad j \in J \\ 0 \leq & t_j - L_j t_0 \qquad\quad j \in J \end{array} \tag{2.22}$$

$$\sum_{j=0}^{n} d_j t_j = 1 \tag{2.23}$$

where index set $I = \{1, 2, \ldots, m\}$, $J = \{1, 2, \ldots, n\}$ and $t_j = \dfrac{x_j}{D(x)}, \quad j = 1, 2, \ldots, n$, $t_0 = \dfrac{1}{D(x)}$. Since we supposed in (2.5)-(2.7) that $D(x) > 0$ here

$t_0 > 0$, which can be rewritten as $t_0 \geq 0$, since (2.22) and (2.23) exclude the case $t_0 = 0$. Hence, LA model can be extended with an individual bound for t_0:

$$0 \leq t_0 \tag{2.24}$$

Here I examine the presolve operations (discussed in section 2.2.1) one by one and show the differences between the two cases: presolve before or after transformation o demonstrate the advantages of LFP presolve relative to LP presolve on LA model.

First suppose LFP problem (2.5)-(2.7) has an **"empty row"** indexed by i_0 (i.e. $a_{i_0 j} = 0$, $j = 1, 2, \ldots, n$). As I mentioned above, this row can be removed form the model. Now, let us see what happens if we transform such a model into LA form and after that try to apply presolve operations to LA. It is obvious the empty row does not affect the objective function (2.20) and constraints (2.22), (2.23) and (2.24) in LA form. Row i_0 results the following constraints instead of (2.21) in linear analogue model:

$$\begin{aligned}
& \sum_{j=1}^{n} a_{ij} t_j - \overline{b}_i t_0 && \leq 0 && i \in I \backslash i_0 \\
0 \leq & \sum_{j=1}^{n} a_{ij} t_j - \underline{b}_i t_0 && && i \in I \backslash i_0 \\
& -\overline{b}_{i_0} t_0 && \leq 0 && \\
0 \leq & -\underline{b}_{i_0} t_0 && &&
\end{aligned}$$

We can see row i_0 is not an empty, but a singleton row in LA model. Constraints $-\overline{b}_{i_0} t_0 \leq 0$ and $0 \leq -\underline{b}_{i_0} t_0$ are redundant because of (2.24), and they can be removed from the model unless primal infeasibility is detected. As we mentioned in subsection 2.2.1.1 if $\overline{b}_{i_0} < 0$ or $\underline{b}_{i_0} > 0$ the model is primal infeasible. So we see that in case of LFP "empty row" test simply removes row i_0, while in LA model we have to apply "singleton row" test.

According to **"empty column"** test a variable (x_l) with zero coefficients in all constraints (2.6) can be fixed at its lower or upper bound if it fulfills one of the conditions in theorem 2.1. As individual bound restrictions (2.7) of LFP became two-variable constraints in LP (2.22) since there appear some more constraint coefficients of the corresponding variable (2.22), (2.23). Considering the constraint matrix of LA model we can see variable x_l is not associated with an empty column. Hence, if we do not remove an empty column (j_0) which satisfies one of the conditions in theorem 2.1 from LFP model before transformation, in

LA model will not be any chance for it, because column j_0 in LA model is no longer an empty column.

Case of **"singleton rows"** is quite similar: When transforming an LFP model with a singleton row (i_0, where $a_{i_0 j} = 0$, $\forall j \in J \backslash l$ and $a_{i_0 l} \neq 0$) into LA form, row i_0 became a constraint as follows:

$$\underline{b_{i_0}} \leq \underbrace{\sum_{j \in J \backslash l} a_{i_0 j} x_j}_{0} + a_{i_0 l} x_l \leq \overline{b_{i_0}} \quad \rightarrow \quad \begin{cases} \underbrace{\sum_{j \in J \backslash l} a_{i_0 j} t_j}_{0} + a_{i_0} t_l - \overline{b_{i_0}} t_0 \leq 0 \\ \underbrace{-\sum_{j \in J \backslash l} a_{i_0 j} t_j}_{0} + a_{i_0} t_l + \underline{b_{i_0}} t_0 \leq 0 \end{cases}$$

$$A^{(LFP)} = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & a_{i_0 l} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\downarrow$$

$$A^{(LA)} = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & a_{i_0 l} & 0 & \dots & 0 & \{-\overline{b_{i_0}} \mid \underline{b_{i_0}}\} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

After Charnes-Cooper transformation of LFP model in row i_0 of LA model appears a constraint with two nonzero coefficients. Presolve operations mentioned above can not reduce such a model.

As LA model (2.20)-(2.24) has bound restriction only for variable t_0 the **"primal feasibility"** test can be used only in some special cases when constraints contain only the variable t_0. This kind of constraints appear if the transformed LFP model has an empty row as we examined above.

Considering the **"cheap dual"** test we can realize that if variable x_j in LFP problem (2.5)-(2.7) satisfies the conditions in section 2.2.1.4 then the corresponding t_j in LA satisfies constraints (2.21). However constraints (2.22) also contain variable t_j and therefore conditions of cheap dual test are not fulfilled for all constraints of LA model.

The last presolve operation considered in section 2.2.1 was the **"duplicate rows"** which may be formulated as follows: If coefficients of row k can be obtained by a simple multiplication of coefficients in constraint i, i.e.:

$$\begin{array}{lll} \text{Row } i: & \sum_{j=1}^{n} a_{ij}x_j \le b_i & \\ \text{Row } k: & \sum_{j=1}^{n} \underbrace{\alpha a_{ij}}_{a_{kj}} x_j \le b_k & \end{array} \quad \rightarrow \quad \begin{array}{ll} \text{Row } i': & \sum_{j=1}^{n} a_{ij}t_j - b_i t_0 \le 0 \\ \text{Row } k': & \sum_{j=1}^{n} \underbrace{\alpha a_{ij}}_{a_{kj}} t_j - b_k t_0 \le 0, \end{array}$$

row i and k in LFP model can be simplified as redundant constraints (unless primal infeasibility is detected), but row i' and k' in LA model are linearly independent rows, so LA model can not be reduced (except if $b_i = \alpha b_k$).

2.2.2 Scaling

Solving a large-scale problem requires hundreds of thousands to millions of floating-point arithmetic operations. Because of the finite precision inherent in computer arithmetic small numerical errors occur during these calculations. These errors typically have a cumulative effect, which often leads to a numerically unstable problem and possibly large errors in the "solution" obtained.

To avoid such problems all well-made industrial solvers include special sophisticated techniques that dramatically reduce the cumulative effect of rounding and often lead to considerable improvement in the solvers' performance. One of the most simple, relatively effective and widespread techniques of this type is scaling. This is a well-defined transformation of the constraint-matrix A that attempts to make the magnitudes of the data as close to each other as possible. That means those rows or columns of matrix A which are badly scaled must be transformed with their own scale factors ρ. The main difference between several scaling algorithms is the factor-defining method. In most cases scaling improves the numerical characteristics of a problem and sometimes it can also reduces the number of iterations needed during the simplex method.

Most professionally developed solvers automatically use scaling methods to maintain numerical stability and improve the performance. Normally, we can choose among such options as: "No Scaling", "Row Scaling", "Column Scaling", or "Row and Column Scaling" with or without scaling the objective function. For instance the well-known CPLEX optimizer lets us to set the so called *scale parameter* (CPX_PARAM_SCAIND) which can get three different values:

LINGO also provides us the opportunity to set scaling on or off:

Value	Meaning
-1	No scaling
0	Equilibration scaling (default)
1	More aggressive scaling

Table 2.2: Scaling settings in CPLEX

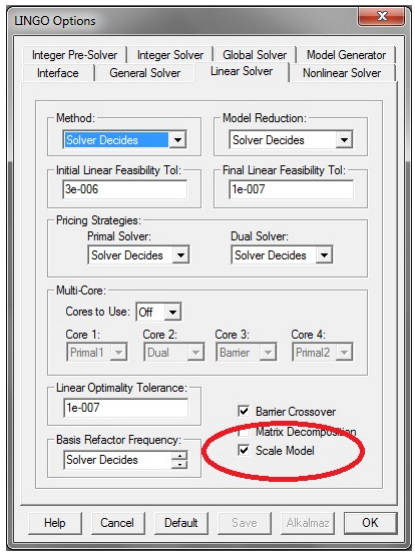


Figure 2.3: LINGO scaling settings

In this section I present our adaptation of scaling techniques into LFP. The section mainly consists of the following two part.

In the first one I show our investigations connected with differences and connections between the optimal solution of the original problem and that of the scaled problem in case of LFP, and present the re-scaling operation needed to obtain the optimal solution of the original problem from the optimal solution of the scaled problem.

In the second part I briefly overview three different rules for calculating scaling factors, and illustrate with a numerical example how these methods work. It is also considered how can they improve the $\sigma(A)$ see (2.4). The first

two rules have been implemented in the linear programming codes developed at Edinburgh University, Department of Mathematics and Statistics, Scotland. The third rule has been developed by me.

Finally, I compare my new scaling rule to the two other by conducting several computational test.

2.2.2.1 Theoretical background

Before turning our attention to the scaling methods, I briefly overview what kind of changes arise from scaling transformations. Let us consider the following canonical LFP problem:

$$Q(x) = \frac{P(x)}{D(x)} = \frac{\sum_{j=1}^{n} p_j x_j + p_0}{\sum_{j=1}^{n} d_j x_j + d_0} \to \max \tag{2.25}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, 2, \ldots, m, \tag{2.26}$$

$$x_j \geq 0, \quad j = 1, 2, \ldots, n, \tag{2.27}$$

where $D(x) > 0, \quad \forall x = (x_1, x_2, \ldots, x_n)^T \in S, \quad S$ - is a feasible set defined by constraints (2.26) and (2.27).

Scaling may include coefficients of the objective function. In case of LP problems scaling a constraint matrix A, a right-hand-side vector b or an objective function $P(x)$ does not lead to any difficulties because of linearity of constraints and objective function.

In case of scaling LFP problems we distinguish the following cases:

I. Scaling constraints:

- right-hand side vector $b = (b_1, b_2, \ldots, b_m)^T$;
- columns $A_j \ \ (j = 1, 2, \ldots, n)$ of matrix A;
- rows of matrix A;

II. Scaling objective function:

- only the vector $p = (p_0, p_1, p_2, \ldots, p_n)$ connected to numerator $P(x)$;

- only the vector $d = (d_0, d_1, d_2, \ldots, d_n)$ connected to denominator $D(x)$;
- both vectors p and d that define the objective function $Q(x)$.

Below we investigate all these possible cases.

Case 1 *Right-hand side column-vector*

Suppose that vector x^* is an optimal solution for LFP problem (2.25)-(2.27), so

$$\sum_{j=1}^{n} A_j x_j^* = b \quad \text{and} \quad x^* \geq 0,$$

and matrix $B = (A_{s_1}, A_{s_2}, \ldots, A_{s_m})$ is its basis.

Let us replace RHS vector b with some other vector $b' = \rho\, b$, where $\rho > 0$. Consider the new vector $x' = \rho x^*$. It is obvious that this vector x' satisfies the constraints:

$$\sum_{j=1}^{n} A_j(\rho x_j^*) = \rho\, b \quad \text{and} \quad x' = \rho x^* \geq 0,$$

so vector x' is a feasible solution of LFP problem

$$Q(x) = \frac{P(x)}{D(x)} = \frac{\displaystyle\sum_{j=1}^{n} p_j x_j + p'_0}{\displaystyle\sum_{j=1}^{n} d_j x_j + d'_0} \to \max \tag{2.28}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = \rho b_i, \quad i = 1, 2, \ldots, m; \tag{2.29}$$

$$x_j \geq 0, \quad j = 1, 2, \ldots, n. \tag{2.30}$$

Now we have to check vector x' whether it is an optimal solution of problem (2.28)-(2.30). Since vector x^* is an optimal solution of the original LFP problem (2.25)-(2.27) it means that

$$\Delta_j(x^*) \geq 0, \;\; j = 1, 2, \ldots, n, \tag{2.31}$$

where

$$\begin{aligned}\Delta_j(x^*) &= D(x^*)\Delta'_j - P(x^*)\Delta''_j, \;\; j = 1, 2, \ldots, n,\\ \Delta'_j &= \sum_{i=1}^{m} p_{s_i} x_{ij} - p_j, \;\; j = 1, 2, \ldots, n,\\ \Delta''_j &= \sum_{i=1}^{m} d_{s_i} x_{ij} - d_j, \;\; j = 1, 2, \ldots, n,\end{aligned}$$

coefficient x_{ij} are defined by the systems

$$\sum_{i=1}^{m} A_{s_i} x_{ij} = A_j, \;\; j = 1, 2, \ldots, n; \tag{2.32}$$

where A_j denotes the column-vectors $A_j = (a_{1j}, a_{2j}, \ldots, a_{mj})^T, \; j = 1, 2, \ldots, n,$ of matrix $A = \|a_{ij}\|_{m\times n}$.

Observe that reduced costs Δ'_j and Δ''_j do not depend on RHS vector b, so substitution $b \to \rho\, b$ does not affect values of Δ'_j and Δ''_j. However, values of functions $P(x)$ and $D(x)$ depend on RHS vector b, so we have to consider the new reduced costs $\Delta_j(x')$, where $x' = \rho x^*$, for LFP problem (2.28)-(2.30). We have

$$\begin{aligned}\Delta_j(\rho x^*) &= D(\rho x^*)\; \Delta'_j - P(\rho x^*)\; \Delta''_j =\\ &= (\sum_{j=1}^{n} d_j(\rho x_j^*) + d'_0)\; \Delta'_j - (\sum_{j=1}^{n} p_j(\rho x_j^*) + p'_0)\; \Delta''_j =\\ &= (\sum_{j=1}^{n} d_j(\rho x_j^*) + d'_0 + \rho d_0 - \rho d_0)\; \Delta'_j -\\ &\quad -(\sum_{j=1}^{n} p_j(\rho x_j^*) + p'_0 + \rho p_0 - \rho p_0)\; \Delta''_j =\\ &= \rho D(x^*)\; \Delta'_j + (d'_0 - \rho d_0)\; \Delta'_j - \rho P(x^*)\; \Delta''_j - (p'_0 - \rho p_0)\; \Delta''_j =\\ &= \rho\Delta_j(x^*) + (d'_0 - \rho d_0)\; \Delta'_j - (p'_0 - \rho p_0)\; \Delta''_j =\\ &= \rho\Delta_j(x^*) - G_j, \end{aligned}\tag{2.33}$$

where

$$G_j = (p'_0 - \rho p_0)\; \Delta''_j - (d'_0 - \rho d_0)\; \Delta'_j \; .$$

Formula (2.33) means that if p'_0 and d'_0 are such that

$$\rho\Delta_j(x^*) - G_j \geq 0, \;\; j = 1, 2, \ldots, n,$$

or, in particular, if $p'_0 = \rho p_0$ and $d'_0 = \rho d_0$, then

$$\Delta_j(\rho x^*) \overset{(2.33)}{=} \rho\Delta_j(x^*) \overset{(2.31)}{\geq} 0, \;\; \forall j = 1, 2, \ldots, n,$$

and hence, vector x' is an optimal solution of LFP problem (2.28)-(2.30).

So, if we substitute RHS vector b with some other vector $b' = \rho b$, $\rho > 0$, we have simultaneously to replace coefficients p_0 and d_0 in the original objective function $Q(x)$ with $p'_0 = \rho p_0$ and $d'_0 = \rho d_0$, respectively. These two substitutions will guarantee the equivalency between the original problem (2.25)-(2.27) and the new scaled LFP problem (2.28)-(2.30).

It is obvious that if vector x' is an optimal solution of the new (scaled) LFP problem (2.28)-(2.30), then vector $x^* = x'/\rho$ is an optimal solution of the original LFP problem (2.25)-(2.27).

Case 2 *Left-hand side column-vectors*

In this case we scale columns A_j, $j = 1, 2, \ldots, n$, of matrix $A = \|a_{ij}\|_{m\times n}$. We suppose that vector x^* is an optimal solution for the original LFP problem (2.25)-(2.27), so

$$\sum_{j=1}^{n} A_j \, x_j^* = b \quad \text{and} \quad x_j^* \geq 0, \; j = 1, 2, \ldots, n,$$

and matrix $B = (A_{s_1}, A_{s_2}, \ldots, A_{s_m})$ is its basis.

Let us replace vector A_r, $r \in J = \{1, 2, \ldots, n\}$, with some other vector $A'_r = \rho A_r$, where $\rho > 0$. It is obvious that, the new vector

$$x' = (x_1^*, \, x_2^*, \, \ldots, \, x_{r-1}^*, \, \frac{x_r^*}{\rho}, \, x_{r+1}^*, \, \ldots, \, x_n^*)$$

will satisfy constraints

$$\sum_{\substack{j=1 \\ j\neq r}}^{n} A_j x_j^* + \rho A_r \frac{x_r^*}{\rho} = b,$$

$$x'_j \geq 0, \;\; j = 1, 2, \ldots, n,$$

and hence vector x' is a feasible solution of the new scaled LFP problem

$$Q(x) = \frac{P'(x)}{D'(x)} = \frac{\sum\limits_{\substack{j=1\\ j\neq r}}^{n} p_j x_j + p'_r x_r + p_0}{\sum\limits_{\substack{j=1\\ j\neq r}}^{n} d_j x_j + d'_r x_r + d_0} \to \max \tag{2.34}$$

subject to

$$\sum_{\substack{j=1\\ j\neq r}}^{n} A_j x_j + A'_r x_r = b, \tag{2.35}$$

$$x_j \geq 0, \quad j = 1, 2, \ldots, n. \tag{2.36}$$

Our aim is now to detect whether vector x' is an optimal solution of the scaled LFP problem (2.34)-(2.36). Since vector x^* is an optimal solution of the original problem (2.25)-(2.27), we have that

$$\Delta_j(x^*) \;=\; D(x^*)\Delta'_j - P(x^*)\Delta''_j \;\geq 0, \quad j = 1, 2, \ldots, n. \tag{2.37}$$

Here we have to distinguish the following two cases: A_r is a basic vector, and vector A_r is a non-basic vector.

Let us suppose that A_r is a basic vector, i.e. $r \in J_B = \{s_1, s_2, \ldots, s_m\}$. In this case, keeping in mind that

$$P'(x') = \sum_{\substack{j=1\\ j\neq r}}^{n} p_j x^*_j + p'_r \frac{x^*_r}{\rho} + p_0 \;, \qquad D'(x') = \sum_{\substack{j=1\\ j\neq r}}^{n} d_j x^*_j + d'_r \frac{x^*_r}{\rho} + d_0 \;,$$

$$\Delta'_j = \sum_{\substack{i=1\\ s_i\neq r}}^{m} p_{s_i} x_{ij} + p'_r \frac{x_{rj}}{\rho} - p_j \;, \qquad \Delta''_j = \sum_{\substack{i=1\\ s_i\neq r}}^{m} d_{s_i} x_{ij} + d'_r \frac{x_{rj}}{\rho} - d_j \;,$$

for the new scaled problem (2.34)-(2.36) we have

$$\begin{aligned} \Delta_j(x') &= D'(x')\Delta'_j - P'(x')\Delta''_j = \\ &= (D(x^*) - d_r x^*_r + d'_r \frac{x^*_r}{\rho})(\Delta'_j - p_r x_{rj} + p'_r \frac{x_{rj}}{\rho}) - \\ &- (P(x^*) - p_r x^*_r + p'_r \frac{x^*_r}{\rho})(\Delta''_j - d_r x_{rj} + d'_r \frac{x_{rj}}{\rho}) \end{aligned} \tag{2.38}$$

Equation (2.38) makes it obvious that if $p'_r = p_r\rho$ and $d'_r = d_r\rho$, then

$$\Delta_j(x') \overset{(2.38)}{=} \Delta_j(x^*) \overset{(2.37)}{\geq} 0, \quad j = 1, 2, \ldots, n.$$

The latter means that in this case vector x' is an optimal solution of the scaled LFP problem (2.34)-(2.36). So, if we substitute some basic vector A_r with some other vector $A'_r = \rho\, A_r$, $\rho > 0$, we simultaneously have to replace coefficients p_r and d_r in the original objective function $Q(x)$ with $p'_r = \rho p_r$ and $d'_r = \rho d_r$, respectively. These two substitutions will guarantee the equivalency between the original problem (2.25)-(2.27) and the new scaled LFP problem (2.34)-(2.36).

It is obvious that if vector x' is an optimal solution of the new (scaled) LFP problem (2.34)-(2.36), then vector $x^* = (x'_1, x'_2, \ldots, x'_{r-1}, x'_r\rho, x'_{r+1}, \ldots, x'_n)$ will be an optimal solution of the original LFP problem (2.25)-(2.27).

Now, we have to consider the case when substituted vector A_r is a non-basic vector, i.e. $r \in J_N = J \setminus J_B$. As in the previous case, we simultaneously replace original coefficients p_r and d_r with ρp_r and ρd_r, respectively. Since index r is non-basic and $x^*_r = 0$, it is obvious that

$$x' = x^*, \quad P'(x') = P(x^*), \quad D'(x') = D(x^*) \quad \text{and hence} \quad Q'(x') = Q(x^*).$$

So replacement $A_r \rightarrow \rho A_r$, $r \in J_N$, affects only values of Δ'_r, Δ''_r, and $\Delta_r(x')$. Indeed, if in the original LFP problem (2.25)-(2.27) for non-basic vector A_r we had (see (2.32)), that

$$\sum_{i=1}^{m} A_{s_i} x_{ir} = A_r, \quad r \in J_N,$$

then after replacement $A_r \rightarrow A'_r$, where $A'_r = \rho A_r$, we obtain the following representation of the new vector A'_r in the same basis B:

$$\sum_{i=1}^{m} A_{s_i} (\rho x_{ir}) = \rho A_r, \quad r \in J_N.$$

If in case of replacing $A_r \rightarrow \rho A_r$, we simultaneously substitute $p_r \rightarrow p'_r$, where $p'_r = \rho p_r$, and $d_r \rightarrow d'_r$, where $d'_r = \rho d_r$, then for new $\tilde{\Delta}'_r$, $\tilde{\Delta}''_r$, and $\tilde{\Delta}_r(x')$ we have

$$\begin{aligned} \tilde{\Delta}'_r &= \sum_{i=1}^{m} p_{s_i} (\rho x_{ir}) - (\rho p_r) = \rho \Delta'_r, \\ \tilde{\Delta}''_r &= \sum_{i=1}^{m} d_{s_i} (\rho x_{ir}) - (\rho d_r) = \rho \Delta''_r. \end{aligned}$$

Thus,

$$\begin{aligned}\tilde{\Delta}_r(x') &= D(x^*)\,\tilde{\Delta}'_r - P(x^*)\,\tilde{\Delta}''_r \\ &= D(x^*)\,(\rho\Delta'_r) - P(x^*)\,(\rho\Delta''_r) \\ &= \rho\Delta_r(x^*) \overset{(2.37)}{\geq} 0.\end{aligned}$$

The latter means that in this case vector x^* is an optimal solution of the scaled LFP problem (2.34)-(2.36).

So, if we substitute some non-basic vector A_r with some other vector $A'_r = \rho\, A_r,\ \rho > 0,$ we have simultaneously to replace coefficients p_r and d_r in the original objective function $Q(x)$ with $p'_r = \rho p_r$ and $d'_r = \rho d_r,$ respectively. These two substitutions will guarantee the equivalency between the original problem (2.25)-(2.27) and the new scaled LFP problem (2.34)-(2.36). Moreover, it will guarantee that $x^*_r = x'_r = 0.$

Case 3 *Rows of constraints*

Let us replace row-vector $a_r = (a_{r1}, a_{r2}, \ldots, a_{rn})$ of matrix $A = ||a_{ij}||_{m\times n}$ in LFP problem (2.25)-(2.27) with some other row-vector $a'_r = \rho a_r$. If we simultaneously with replacement $a_r \to \rho a_r$ substitute the r-th element of RHS vector b, i.e. $b_r \to b'_r = \rho b_r$, we obtain instead of the original constraint in the r-th row

$$\sum_{j=1}^{n} a_{rj}x_j = b_r, \quad \text{we have} \quad \sum_{j=1}^{n}(\rho a_{rj})x_j = (\rho b_r).$$

It is well-known that such scaling does not affect the structure of feasible set S. So the new scaled problem is absolutely equivalent with the original one.

It is obvious, if we do not modify RHS vector b, it leads to unpredictable deformations in feasible set S, so we cannot provide any guarantee that the optimal basis of the scaled problem will be the same as in the original one. So, the only negotiable method of scaling rows in matrix A is the following $\tilde{a}_r \to \tilde{a}'_r$ where $\tilde{a}_r = (a_{r1},\ a_{r2},\ \ldots,\ a_{rn},\ b_r)$ and $\tilde{a}'_r = (\rho a_{r1},\ \rho a_{r2},\ \ldots,\ \rho a_{rn},\ \rho b_r)$.

Obviously, the optimal solution x' of the scaled LFP problem is absolutely identical with the optimal solution x^* of the original LFP problem. So we need not any postsolve operation in this case.

Note that in the simplex method, when performing θ-ratio test, the elements of the pivotal column take part in the calculations. Hence, the choice of pivotal row depends on the row scaling. Since a bad choice of pivots can lead to large errors in the computed solution, it means that a proper row scaling is very important.

Case 4 *Objective function - Numerator $P(x)$*

Let us replace the coefficient vector $p = (p_0, p_1, \ldots, p_n)$ in the numerator $P(x)$ with some other vector $p' = (p'_0, p'_1, \ldots, p'_n)$, where $p'_j = \rho p_j,\ j = 0, 1, 2, \ldots, n$.

It is clear that this replacement does not affect either the optimal value of denominator $D(x)$ or the values of reduced costs $\Delta''_j,\ j = 1, 2, \ldots, n$, but it changes the optimal values of functions $P(x)$ and $Q(x)$, and in this way affects the values of reduced costs Δ'_j and $\Delta_j(x),\ j = 1, 2, \ldots, n$.

So, for the new values $\tilde{\Delta}'_j,\ P'(x^*),\ Q'(x^*),$ and $\tilde{\Delta}_j(x^*),\ j = 1, 2, \ldots, n$, we have:

$$\begin{aligned}
\tilde{\Delta}'_j &= \sum_{i=1}^{m} p'_{s_i} x_{ij} - p'_j = \sum_{i=1}^{m} (\rho\, p_{s_i}) x_{ij} - (\rho\, p_j) = \rho\, \Delta'_j,\ \ j = 1, 2, \ldots, n, \\
P'(x^*) &= \sum_{j=1}^{n} p'_j x^*_j + p'_0 = \sum_{j=1}^{n} (\rho\, p_j) x^*_j + (\rho\, p_0) = \rho\, P(x^*), \\
Q'(x^*) &= P'(x^*)/D(x^*) = \rho\, P(x^*)/D(x^*) = \rho\, Q(x^*),
\end{aligned}$$

and hence,

$$\begin{aligned}
\tilde{\Delta}_j(x^*) &= D(x^*)\tilde{\Delta}'_j - P'(x^*)\Delta''_j = \\
&= D(x^*)(\rho\, \Delta'_j) - (\rho\, P(x^*))\Delta''_j = \rho\, \Delta_j(x^*),\ \ j = 1, 2, \ldots, n.
\end{aligned}$$

Since $\rho > 0$ and $\Delta_j(x^*) \geq 0,\ j = 1, 2, \ldots, n,$ the latter means that

$$\tilde{\Delta}_j(x^*) = \rho\, \Delta_j(x^*) \overset{(2.37)}{\geq} 0,\ \ j = 1, 2, \ldots, n.$$

Finally, we have to note that replacement $p \to \rho p$ does not lead to any changes in the optimal basis or in optimal solution x^*. So, if we have solved the scaled LFP problem, in order to "un-scale" the optimal solution obtained we have to use the following formula $Q(x^*) = \dfrac{1}{\rho}\, Q'(x^*)$, because the optimal solution x' of the scaled problem is exactly the same as optimal solution x^* of the original problem.

Case 5 *Objective function - Denominator $D(x)$*

Let us replace vector $d = (d_0, d_1, \ldots, d_n)$ in the denominator $D(x)$ with another vector $d' = (d'_0, d'_1, \ldots, d'_n)$, where $d'_j = \rho d_j,\ j = 0, 1, \ldots, n$.

It is obvious that such replacement leads to some changes in the optimal values of denominator $D(x)$, objective function $Q(x)$ and values $\Delta''_j,\ \Delta_j(x),\ j = 1, 2, \ldots, n$; but does not affect the optimal value of numerator $P(x)$ or the values of reduced costs $\Delta'_j,\ j = 1, 2, \ldots, n$.

So, for new values $\tilde{\Delta}''_j,\ D'(x^*),\ Q'(x^*)$, and $\tilde{\Delta}_j(x^*),\ \ j = 1, 2, \ldots, n$, we have

$$\begin{aligned}
\tilde{\Delta}''_j &= \sum_{i=1}^{m} d'_{s_i} x_{ij} - d'_j = \sum_{i=1}^{m} (\rho\, d_{s_i}) x_{ij} - (\rho\, d_j) = \rho\, \Delta''_j,\ \ j = 1, 2, \ldots, n, \\
D'(x^*) &= \sum_{j=1}^{n} d'_j x^*_j + d'_0 = \sum_{j=1}^{n} (\rho\, d_j) x^*_j + (\rho\, d_0) = \rho\, D(x^*), \\
Q'(x^*) &= P(x^*)/D'(x^*) = P(x^*)/(\rho\, D(x^*)) = Q(x^*)/\rho,
\end{aligned}$$

and hence,

$$\begin{aligned}
\tilde{\Delta}_j(x^*) &= D'(x^*)\Delta'_j - P(x^*)\tilde{\Delta}''_j = \\
&= (\rho\, D(x^*))\Delta'_j - P(x^*)(\rho\, \Delta''_j) = \rho\, \Delta_j(x^*),\ \ j = 1, 2, \ldots, n.
\end{aligned}$$

Thus, we obtain

$$\tilde{\Delta}_j(x^*) = \rho\, \Delta_j(x^*) \overset{(2.37)}{\geq} 0,\ \ j = 1, 2, \ldots, n.$$

Finally, we have to note that replacement $d \to \rho d$ does not lead to any changes in optimal basis B or in optimal solution x^*. So once we have solved the scaled LFP problem, in order to postsolve the optimal solution obtained we have to use the following formula $Q(x^*) = \rho\, Q'(x^*)$, because the optimal solution x' of the scaled problem is exactly the same as optimal solution x^* of the original problem.

2.2.2.2 Geometric rule

In accordance with this rule we define the following column-vector ρ^r as scaling factors for rows

$$\rho^r = (\rho^r_1, \rho^r_2, \ldots, \rho^r_m)^T, \tag{2.39}$$

where

$$\rho_i^r \;=\; (\prod_{j\in J_i^+} a_{ij})^{1/K_i^r}, \quad i = 1, 2, \ldots, m;$$

$J_i^+ = \{j : \; a_{ij} \neq 0\}, \;\; i = 1, 2, \ldots, m$, is a row related set of indices j of non-zero entries a_{ij} in row i, and K_i^r denotes the number of non-zero entries a_{ij} in row i.

Analogically, to scale columns we use the following factors organized in a row-vector ρ^c

$$\rho^c \;=\; (\rho_1^c, \rho_2^c, \ldots, \rho_n^c), \tag{2.40}$$

where

$$\rho_j^c \;=\; (\prod_{i\in I_j^+} a_{ij})^{1/K_j^c}, \quad j = 1, 2, \ldots, n;$$

$I_j^+ = \{i : \; a_{ij} \neq 0\}, \;\; j = 1, 2, \ldots, n$, is a column related set of indices i of non-zero entries a_{ij} in column j, and K_j^c denotes the number of non-zero entries a_{ij} in column j.

2.2.2.3 Mean rule

As an alternative to the scaling factors calculated in accordance with Geometric rule, the Mean rule defines the following column-vector ρ^r as scaling factors for rows

$$\rho^r \;=\; (\rho_1^r, \rho_2^r, \ldots, \rho_m^r)^T, \tag{2.41}$$

where

$$\rho_i^r \;=\; \sqrt{r_i' \, r_i''}, \quad i = 1, 2, \ldots, m;$$

and

$$r_i' = \max_{j:\, a_{ij}\neq 0} |a_{ij}|, \quad r_i'' = \min_{j:\, a_{ij}\neq 0} |a_{ij}|, \quad i = 1, 2, \ldots, m.$$

Analogically to row scaling factors, for columns we have to define the following row-vector ρ^c as scaling factors

$$\rho^c \;=\; (\rho_1^c, \rho_2^c, \ldots, \rho_n^c), \tag{2.42}$$

where

$$\rho_j^c \;=\; \sqrt{c_j' \, c_j''}, \quad j = 1, 2, \ldots, n;$$

and

$$c_j' = \max_{i:\, a_{ij}\neq 0} |a_{ij}|, \quad c_j'' = \min_{i:\, a_{ij}\neq 0} |a_{ij}|, \quad j = 1, 2, \ldots, n.$$

2.2.2.4 Min-Max rule

As an alternative to the previous two scaling rules I introduce my new method for calculating scaling factors. First, define the smallest $a_{i_0 j_0}$ and the largest $a_{i_1 j_1}$ absolute values for non-zero entries in the matrix and then calculate the following four scaling factors:

- ρ'_c – for column j_0 containing the minimal value,
- ρ''_c – for column j_1 containing the maximal value,
- ρ'_r – for row i_0 containing the minimal value,
- ρ''_r – for row i_1 containing the maximal value,

using the following formulas:

$$\rho'_c = \frac{\max\limits_i |a_{ij_0}| + a_{i_1 j_1}}{2 \times \max\limits_i |a_{ij_0}|}, \qquad \rho''_c = \frac{\min\limits_{i:\, a_{ij} \neq 0} |a_{ij_1}| + a_{i_0 j_0}}{2 \times \min\limits_{i:\, a_{ij} \neq 0} |a_{ij_1}|},$$

$$\rho'_r = \frac{\max\limits_j |a_{i_0 j}| + a_{i_1 j_1}}{2 \times \max\limits_j |a_{i_0 j}|}, \qquad \rho''_r = \frac{\min\limits_{j:\, a_{ij} \neq 0} |a_{i_1 j}| + a_{i_0 j_0}}{2 \times \min\limits_{j:\, a_{ij} \neq 0} |a_{i_1 j}|}.$$

It is obvious that when applying such scaling factors the 'distance' between the largest and the smallest absolute values in the matrix decreases.

In case of large problems searching the maximum and minimum element of the matrix in each iteration which modifies only one row/column may be a computationally very expensive operation. Therefore I suggest the following modification in case of scaling large problems in order to calculate a factor for each row/column in an iteration.

$$\rho_i^{r'} = \frac{\max\limits_j |a_{ij}| + a_{i_1 j_1}}{2 \times \max\limits_j |a_{ij}|}, \qquad \rho_i^{r''} = \frac{\min\limits_{j:\, a_{ij} \neq 0} |a_{ij}| + a_{i_0 j_0}}{2 \times \min\limits_{j:\, a_{ij} \neq 0} |a_{ij}|},$$

$$\rho_j^{c'} = \frac{\max\limits_i |a_{ij}| + a_{i_1 j_1}}{2 \times \max\limits_i |a_{ij}|}, \qquad \rho_j^{c''} = \frac{\min\limits_{i:\, a_{ij} \neq 0} |a_{ij}| + a_{i_0 j_0}}{2 \times \min\limits_{i:\, a_{ij} \neq 0} |a_{ij}|},$$

where $\rho_i^{r'}$ and $\rho_i^{r''}$ are factors for row i and $\rho_j^{c'}$ and $\rho_j^{c'}$ are factor for column j. I permute these factors during the iterations.

2.2.2.5 Numerical example for scaling methods

All three rules are suitable to be used for automatic scaling in programming packages and allow relatively easy achievement of a well-scaled problem.

To illustrate how these scaling factors work, we consider the following rectangular matrix of size 7×5:

$$A = \begin{pmatrix} 0.0005 & 3.000 & 0.340 & 234.000 & 34.000 \\ 2.0000 & 4.000 & 345.000 & 1234.000 & 234.000 \\ 30000.0000 & 5.000 & 4565643.000 & 34.000 & 234.000 \\ 9.0000 & 6.000 & 0.001 & 567.000 & 4.000 \\ 567.0000 & 7.000 & 234.000 & 24.000 & 234.000 \\ 56.0000 & 8.000 & 345.000 & 0.001 & 3.000 \\ 45000.0000 & 9.000 & 4.000 & 3.000 & 123.000 \end{pmatrix}. \quad (2.43)$$

This matrix can be said to be **badly scaled** since

$$\begin{aligned} \max_{i,j \in J_+} (|a_{ij}|) &= a_{33} = 4565643.000 = 4.565643E+06, \\ \min_{i,j \in J_+} (|a_{ij}|) &= a_{11} = 0.0005 = 5.000000E-04; \end{aligned}$$

and

$$\sigma(A) = \frac{\max\limits_{i,j \in J_+} (|a_{ij}|)}{\min\limits_{i,j \in J_+} (|a_{ij}|)} = \frac{4.565643E+06}{5.00E-04} = 9.13E+09,$$

i.e. the magnitude between the largest and the smallest absolute values of non-zero entries a_{ij} is of order 10 ($\sigma(A) = 9.13E+09 \approx 1.0E+10$). Here $J_+ = \{i, j : a_{ij} \neq 0\}$.

1 *Geometric factors*

First, we apply successively Geometric rule factors for rows and columns to scale matrix A. The results of scaling are as follows.

Original matrix: In accordance with rule (2.39) we calculate vector ρ^r of row scaling factors

$$\rho^r = (1.3233, 60.2959, 1403.6460, 2.6158, 87.7940, 3.4138, 56.9257)^T.$$

Perform row scaling.

After 1st scaling of rows For the modified matrix we calculate $\sigma(A)$ measure of scaling:

$$\max_{i,j\in J_+}(|a_{ij}|) \;=\; 3.25E+03; \quad \min_{i,j\in J_+}(|a_{ij}|) \;=\; 2.93E-04;$$

$$\sigma(A) \;=\; \frac{3.25E+03}{2.93E-04} \;=\; 1.11E+07.$$

Row-vector ρ^c of column scaling factors calculated in accordance with rule (2.40)

$$\rho^c = (1.8606, 0.2321, 1.6591, 0.6973, 2.0014).$$

Perform column scaling.

After 1st scaling of columns For the modified matrix we calculate $\sigma(A)$ measure of scaling:

$$\max_{i,j\in J_+}(|a_{ij}|) \;=\; 1.96E+03; \quad \min_{i,j\in J_+}(|a_{ij}|) \;=\; 2.03E-04;$$

$$\sigma(A) \;=\; \frac{1.96E+03}{2.03E-04} \;=\; 9.65E+06.$$

Column-vector ρ^r of row scaling factors will be

$$\rho^r = (1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000)^T.$$

Moreover, rule (2.40) used to calculate new row-vector ρ^c of column scaling factors gives

$$\rho^c = (1.0000, 1.0000, 1.0000, 1.0000, 1.0000).$$

After performing two successive scaling operations for rows and columns, we obtain scaling factors both for rows and columns with values exactly equal to 1. Hence, there is no reason to continue this process, since further improvement of $\sigma(A)$ for matrix A, using this rule is impossible.

So, starting from the original matrix A with $\sigma(A) = 9.13E+09$ we obtained its scaled modification with $\sigma(A) = 9.65E+06$. As we can see, the improvement of magnitude achieved is of order $4 = 10 - 6$.

2 *Mean factors*

Now, let us apply Mean rule factors to scale the same matrix A given in (2.43). We have the following results.

Original matrix In accordance with rule (2.41), vector ρ^r of row scaling factors for original matrix A is

$$\rho^r = (0.3421, 49.6790, 4777.8881, 0.7530, 63.0000, 0.5874, 367.4235)^T.$$

Perform row scaling.

After 1st scaling of rows For modified matrix we calculate $\sigma(A)$ measure of scaling:

$$\max_{i,j\in J_+}(|a_{ij}|) \;=\; 9.56E+02; \quad \min_{i,j\in J_+}(|a_{ij}|) \;=\; 1.05E-03;$$

$$\sigma(A) \;=\; \frac{9.56E+02}{1.05E-03} \;=\; 9.13E+05.$$

We use rule (2.42) to calculate vector ρ^c of column scaling factors:

$$\rho^c = (0.4231, 0.1194, 1.1265, 1.1322, 2.2064).$$

Perform column scaling.

After 1st scaling of columns For the modified matrix we calculate $\sigma(A)$ measure of scaling:

$$\max_{i,j\in J_+}(|a_{ij}|) \;=\; 8.48E+02; \quad \min_{i,j\in J_+}(|a_{ij}|) \;=\; 1.18E-03;$$

$$\sigma(A) \;=\; \frac{8.48E+02}{1.18E-03} \;=\; 7.20E+05.$$

Vector ρ^r of row scaling factors:

$$\rho^r = (1.4448, 1.4448, 2.3090, 0.8854, 2.6752, 0.8854, 1.4448)^T.$$

Perform row scaling.

After 2nd scaling of rows For the modified matrix we calculate $\sigma(A)$ measure of scaling:

$$\max_{i,j\in J_+}(|a_{ij}|) \;=\; 7.51E+02; \quad \min_{i,j\in J_+}(|a_{ij}|) \;=\; 1.33E-03;$$

$$\sigma(A) \;=\; \frac{7.51E+02}{1.33E-03} \;=\; 5.64E+05.$$

Vector ρ^c of column scaling factors:

$$\rho^c = (0.7801, 0.6994, 0.8854, 1.1294, 0.5475).$$

Perform column scaling.

After 2nd scaling of columns For the modified matrix we calculate $\sigma(A)$ measure of scaling:

$$\max_{i,j\in J_+}(|a_{ij}|) \;=\; 6.65E+02; \quad \min_{i,j\in J_+}(|a_{ij}|) \;=\; 1.50E-03;$$

$$\sigma(A) \;=\; \frac{6.65E+02}{1.50E-03} \;=\; 4.42E+05.$$

Vector ρ^r of row scaling factors:

$$\rho^r = (1.0654, 1.0654, 1.0000, 1.0000, 1.0654, 1.0000, 1.0654)^T.$$

Perform row scaling.

After 3rd scaling of rows For the modified matrix we calculate $\sigma(A)$ measure of scaling:

$$\max_{i,j\in J_+}(|a_{ij}|) \;=\; 6.65E+02; \quad \min_{i,j\in J_+}(|a_{ij}|) \;=\; 1.50E-03;$$

$$\sigma(A) \;=\; \frac{6.65E+02}{1.50E-03} \;=\; 4.42E+05.$$

Vector ρ^c of column scaling factors:

$$\rho^c = (0.9688, 1.0000, 1.0000, 1.0000, 0.9688).$$

After performing multiple successive scaling operations for rows and columns, we obtain scaling factors both for rows and columns with values close to 1. Hence, there is no reason to continue this process, since the further improvement of $\sigma(A)$ for matrix A becomes more and more expensive.

So, starting from the original matrix A with $\sigma(A) = 9.13E+09$ we obtained its scaled modification with $\sigma(A) = 4.42E+05$. As we can see, the improvement of magnitude achieved is of order $5 = 10 - 5$.

3 *Min-Max factors*

Finally, let us apply my Min-Max factors to scale the same matrix A given in (2.43).

First, define the smallest entry $a_{11} = 0.0005$ and the largest entry $a_{33} = 4565643.000$, then in the row 1 (which contains the smallest entry) we find the largest entry $a_{14} = 234.00$ and calculate factor ρ'_r for row 1 as follows

$$\rho'_r \;=\; \frac{\max\limits_j |a_{1j}| + a_{33}}{2 \times \max\limits_j |a_{1j}|} \;=\; \frac{234.00 + 4565643.000}{2 \times 234.00} = 9756.1475.$$

Using the factor obtained we scale row 1 and obtain the following

$$\text{Row 1} = \quad 4.8781 \quad 29268.4414 \quad 3317.0901 \quad 2282938.5 \quad 331709.000$$

Now, the smallest non-zero entry of the matrix is $a_{43} = 0.001$, so

$$\sigma(A) \;=\; \frac{\max\limits_{i,j \in J_+} (|a_{ij}|)}{\min\limits_{i,j \in J_+} (|a_{ij}|)} \;=\; \frac{4.565643E+06}{1.00E-03} \;=\; 4.565643E+09,$$

Note, that in the scaled matrix the smallest entry $a_{43} = 0.001$ is in the same column 3 as the largest entry $a_{33} = 4565643.00$, so there is no reason for scaling this column since in this case the value for scaling factors ρ'_c and ρ''_c will be exactly 1. Indeed,

$$\rho'_c \;=\; \frac{\max\limits_i |a_{i3}| + a_{33}}{2 \times \max\limits_i |a_{i3}|} \;=\; \frac{a_{33} + a_{33}}{2 \times a_{33}} \;=\; 1,$$

and

$$\rho''_c \;=\; \frac{\min\limits_{i:\, a_{i3} \neq 0} |a_{i3}| + a_{43}}{2 \times \min\limits_{i:\, a_{i3} \neq 0} |a_{i3}|} \;=\; \frac{a_{43} + a_{43}}{2 \times a_{43}} \;=\; 1.$$

This is why in the next scaling operation we have to scale a row. Arbitrarily, we choose the row of the largest entry and calculate the following factor

$$\rho''_r \;=\; \frac{\min\limits_{j:\, a_{3j} \neq 0} |a_{3j}| + a_{43}}{2 \times \min\limits_{j:\, a_{3j} \neq 0} |a_{3j}|} \;=\; \frac{a_{32} + a_{43}}{2 \times a_{32}} \;=\; \frac{5.00 + 0.001}{2 \times 5.00} \;=\; 0.5001$$

Performing the scaling we obtain

$$\text{Row } 3 = \quad 15003.0000 \quad 2.5005 \quad 2283278.0643 \quad 17.0034 \quad 117.0234$$

and new value for the measure of ill-scaling

$$\sigma(A) \;=\; \frac{\max\limits_{i,j\in J_+}(|a_{ij}|)}{\min\limits_{i,j\in J_+}(|a_{ij}|)} \;=\; \frac{2283278.0643}{0.0010} \;=\; 2.28328E+09,$$

so the matrix obtained is as follows

$$A \;=\; \begin{pmatrix} 4.8781 & 29268.4414 & 3317.0901 & 2282938.5 & 331709.000 \\ 2.0000 & 4.000 & 345.000 & 1234.000 & 234.000 \\ 15003.0000 & 2.5005 & 2283278.0643 & 17.0034 & 117.0234 \\ 9.0000 & 6.000 & 0.001 & 567.000 & 4.000 \\ 567.0000 & 7.000 & 234.000 & 24.000 & 234.000 \\ 56.0000 & 8.000 & 345.000 & 0.001 & 3.000 \\ 45000.0000 & 9.000 & 4.000 & 3.000 & 123.000 \end{pmatrix}.$$

Note, that the largest and the smallest entries of the matrix are in the same column, so in the next scaling operation we have to scale a row. After performing 9 such scaling operations we obtain a matrix with $\sigma(A) = 4.92E+05$.

Observe, that when using this scaling rule in each step we scale only one row or column, so the rule is not so expensive as the previous two scaling rules.

2.2.2.6 Implementation issues

To scale an LFP problem we have to calculate and then to store scaling factors for rows, columns and objective function (separately for numerator and denominator). One of the possible ways to store factors is to expand the matrix of the problem as follows

$$\tilde{A} = \left(\begin{array}{c|cccc|c} \rho_1^r & a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ \rho_2^r & a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ \rho_m^r & a_{m1} & a_{m2} & \dots & a_{mn} & b_n \\ \hline \rho_{m+1}^r & p_1 & p_2 & \dots & p_n & p_0 \\ \rho_{m+2}^r & d_1 & d_2 & \dots & d_n & d_0 \\ \hline & \rho_1^c & \rho_2^c & \dots & \rho_n^c & \rho_{n+1}^c \end{array}\right).$$

During the calculation of Geometric factors ((2.39) and (2.40)) I experienced that overflow had appeared in case of large scale matrices. The reason is the lot of multiplication before the Kth root in formula (2.39) and (2.40). Therefore I propose the following algorithm for calculating Geometric factors:

Initialization;
$\rho_r^{i_0} := 1$;
K := 0;
Determine the number of nonzeros;
for *j:=1* **to** n **do**
begin
  if $a_{i_0 j} \neq 0$ **then**
    K := K+1;
  end
end
Calculate the factor;
for *j:=1* **to** n **do**
begin
  if $a_{i_0 j} \neq 0$ **then**
    $\rho_r^{i_0} := \rho_r^{i_0} * (a_{i_0 j})^{1/K}$;
  end
end

Algorithm 1: Calculating Geometric factor for row i_0 of matrix $||a_{ij}||_{mxn}$

We have to note that instead of precisely calculated values of scaling factors ρ several linear programming codes usually use the nearest powers of two as a "binary approximation" of these values. The reason is that for computers based on the binary system, it may dramatically improve the performance of scaling since in this case the relatively expensive operation of multiplication may be implemented as very fast shifting of data to the left or right, depending on the power of 2 used for such "approximation".

If we scale rows and columns multiple times we have to accumulate scaling factors for post-optimization un-scaling as shown in the following algorithm:

Initialization;
for *i:=1* **to** *m+2* **do**
$\rho_i^r := 1.0$;
for *j:=1* **to** *n+1* **do**
$\rho_j^c := 1.0$;
Repeat scaling several times;
repeat
  Scaling rows;
  for *i:=1* **to** *m+2* **do**
  begin
    temp := get_row_factor(i) ; /* Calculate row factors */
    $\rho_i^r := \rho_i^r$*temp ; /* Update factor */
    scale_row(i, temp) ; /* Row scaling */
  end
  Scaling columns;
  for *j:=1* **to** *n+1* **do**
  begin
    temp := get_col_factor(j) ; /* Calculate column factors */
    $\rho_j^c := \rho_j^c$*temp ; /* Update factor */
    scale_row(i, temp) ; /* Column scaling */
  end
until *Termination condition*;

Algorithm 2: Scaling a LFP problem

2.2.2.7 Test results and comparison

In this section I present computational results produced by my test code developed in `C++`. For this performance evaluation I used 69 test problems from NETLIB collection. According to algorithm (2) I implemented scaling rules with the following `Termination condition` : $\sigma(A) \geq SIGMA\_LIMIT$. Defining $SIGMA\_LIMIT$ as 10^3 I experienced that the rules tested reach this value not in every case:

- Mean rule reached the 10^3 $SIGMA\_LIMIT$ in case of 78 percent of test problems.
- Min-Max rule reached the 10^3 $SIGMA\_LIMIT$ in case of 78 percent of test problems.
- Geometric rule reached the 10^3 $SIGMA\_LIMIT$ in case of 33 percent of test problems.

There were problems on which methods can not reach the 10^3 in value of $\sigma(A)$, because the factors converge into one so the $\sigma(A)$ decreases even less in every iteration until it fixed at a particular value. I performed the code with termination condition containing a higher 10^4 and 10^5 value for $SIGMA\_LIMIT$ in these cases.

The following tableaus specify names, dimensions and initial $\sigma(A)$ value of NETLIB problems. The other columns show the results for different scaling rules which are characterized by the time required for scaling and the final value of $\sigma(A)$ obtained. Symbol "-" shows that the given rule could not reach the predefined $\sigma(A)$.

Test result with termination condition $\sigma(A) \geq 10^3$:

name	size (var-const)	$\sigma(A)$	MinMax_modified $\sigma(A^{\text{scaled}})$	Time (sec)	Mean $\sigma(A^{\text{scaled}})$	Time (sec)	Geometric $\sigma(A^{\text{scaled}})$	Time (sec)
BEACONFD	262-173	1,578E+06	9,023E+02	0,009	6,961E+02	0,000	-	-
CZPROB	3523-929	3,000E+06	8,380E+02	0,651	7,843E+02	0,187	-	-
DEGEN3	1818-1503	4,700E+03	7,907E+02	0,125	2,285E+02	0,109	-	-
FFFFF800	854-524	2,823E+07	9,212E+02	0,202	6,479E+02	0,047	-	-
FIT1P	1677-627	1,890E+05	7,198E+02	0,046	4,018E+02	0,047	9,476E+02	0,047
FIT2D	10500-25	1,626E+05	9,921E+02	0,000	8,703E+02	0,016	-	-
FIT2P	13525-3000	5,128E+04	9,654E+02	5,897	5,510E+02	2,278	-	-
PEROLD	1376-625	6,940E+08	9,915E+02	0,722	9,941E+02	0,172	-	-
SCAGR7	140-129	3,450E+04	6,440E+02	0,003	7,980E+02	0,016	6,686E+02	0,001
SHIP04L	2118-402	5,304E+05	8,634E+02	0,043	5,603E+02	0,062	-	-
SHIP04S	1458-402	5,304E+05	8,634E+02	0,030	5,603E+02	0,047	-	-
SHIP12L	5427-1151	9,006E+05	7,973E+02	1,393	8,000E+02	0,452	-	-
STOCFOR2	2031-2157	7,063E+05	8,655E+02	0,098	2,741E+02	0,187	8,092E+02	0,187

Test result with termination condition $\sigma(A) \geq 10^4$:

name	size (var-const)	$\sigma(A)$	MinMax_modified $\sigma(A^{\text{scaled}})$	Time (sec)	Mean $\sigma(A^{\text{scaled}})$	Time (sec)
BOEING2	143-166	1,000E+07	5,777E+03	0,051	8,069E+03	0,016
CAPRI	353-271	3,162E+07	9,823E+03	0,026	9,972E+03	0,531
D6CUBE	6184-415	1,152E+04	8,524E+03	0,047	4,812E+03	0,062
MAROS	1443-846	4,009E+08	7,558E+03	0,515	4,295E+03	0,047
PILOT4	1000-410	1,067E+09	9,867E+03	0,156	8,401E+03	0,047
STAIR	467-356	8,984E+06	9,670E+03	0,047	9,645E+03	0,011

Tests were performed in the following environment:

- Operating system: Microsoft Windows 7
- CPU: Intel Core i3 2120
- RAM: 8 GB DDR3

Table 1 (in appendix) presents all of the test problems.

One can see from the table that my Min-Max method is more efficient than the Geometric rule, because it can reach the predefined $\sigma(A)$ for every test problem. The required time was almost the same in case of our Min-Max method and Mean rule, but there are several test problem (for instance STOCFOR2, SHIP12L) on which the one or the other method was a bit faster.

I proposed an easy to implement scaling method which is more efficient than Geometric rule. Compering it with the Mean rule we can see that both rules are quite similar from the point of view of the run time and improvement achieved in $\sigma(A)$, but there are some special cases when a little difference appears.

Chapter 3

Integer Programming and Branch and Bound method

3.1 Overview of Branch and Bound algorithm

The Branch and Bound (B&B) algorithm, first proposed by Land and Doig [31], is a well known and efficient algorithm for solving Integer Programming (IP) or Mixed Integer Programming (MIP) problems, it is used by all commercial solvers. In this section first I outline the main steps and mechanism of B&B method, after that I present our method proposed for calculating an initial bound. In the following sections I show how an initial bound, defined by our new method, can improve the efficiency of B&B algorithm.

Let us consider an IP problem in a general form:

$$f(x) \to \min \tag{3.1}$$

subject to

$$g_i(x) \leq b_i \;\; i = 1, \ldots, m \tag{3.2}$$

$$x = (x_1, x_2, \ldots, x_n), \quad x_j - \text{ integer}, \;\; j = 1, 2, \ldots, n, \tag{3.3}$$

where $x \in \mathbb{R}^n$ and $S = \{x \mid g_i(x) \leq b_i, \quad \forall i = 1, \ldots, m\}$. Denote the set of indices by $J = \{1, 2, \ldots, n\}$. The problem obtained by removing restrictions (3.3) is called *relaxation problem* (3.1)-(3.2).

B&B algorithm for IP can be described by the following general pseudo code:

Initialization;
N := $\emptyset$; /* Set of active subproblems/nodes */
OPTSOL := NULL ; /* Best node */
BOUND := ∞ ; /* In case of maximization it is $-\infty$ */
solve(S^0) $\rightarrow$ $x^{*^{(0)}}$;
if $x_j^{*^{(0)}} \in \mathbb{Z}\ \forall j \in J$ **then**
  return $x^{*^{(0)}}$; /* Optimal solution found */
else
  N := N $\bigcup\{S^0\}$;
end
while $N \neq \emptyset$ **do**
  V:= $\emptyset$; /* Initialize the set of non-integer variables */
  Determine the branching variable;
  S^{act} := searching_strategy(N) ; /* see subsection (3.1.2) */
  N := N $\setminus$ S^{act};
  solve(S^{act}) $\rightarrow$ $x^{*^{(act)}}$;
  if $x^{*^{(act)}} \neq$ *NULL ;* /* Optimal solution exists */
  then
    if $x_j^{*^{(act)}} \in \mathbb{Z}\ \forall j \in J$ **then**
      if *obj(S^{act})* $\leq$ *BOUND* **then**
        BOUND := obj(S^{act}) ; /* Update BOUND */
        OPTSOL := $x^{*^{(act)}}$;
      end
    else
      if *obj(S^{act})* $<$ *BOUND* **then**
        V := $\{j | x_j^{*^{(act)}} \notin \mathbb{Z}\}$;
        b := branching_rule(V) ; /* see subsection (3.1.1) */
        S^{act_left} := add_constraint(S^{act}, $x_b \leq [x_b^{*^{(act)}}]$);
        S^{act_right} := add_constraint(S^{act}, $x_b \geq [x_b^{*^{(act)}}] + 1$);
        N := N $\bigcup$ $\{S^{act\_left}, S^{act\_right}\}$;
      end
    end
  end
end
return *OPTSOL*;

Algorithm 3: B&B algorithm

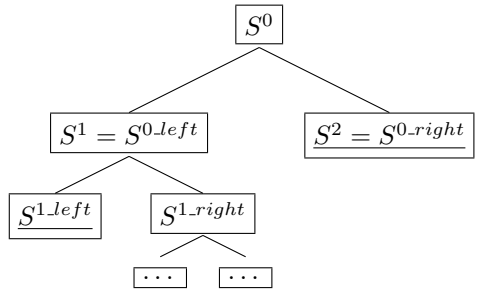
where S^0 is the relaxation problem, $x^{*^{(k)}} = (x_1^{*^{(k)}}, x_2^{*^{(k)}}, \ldots, x_n^{*^{(k)}})$ is the optimal solution of subproblem S^k obtained by simplex or dual simplex algorithm and obj(S^k) gives the optimal objective value of subproblem S^k. Here and in what follows we suppose that relaxation problem S^0 has an optimal solution, if it has not it is obvious that the original problem also has no solution.

B&B solves an IP problem by splitting the feasible set and generating subproblems. The set of subproblems can be represented by a binary tree where every node identifies a subproblem which is derived from its parent node, i.e. the previous subproblem. The first subproblem, the root node, is the relaxation of the original optimization problem.

Branching procedure means if a subproblem S^k has a solution that does not satisfy the integer restrictions (3.3), we choose one of the variables which have a non-integer optimal value. This variable is called branching variable. There are several rules and advices on selection of branching variables (see section 3.1.1). Using the branching variable chosen (x_{i_0}) we create two subproblems S^{left}, S^{right}. Both subproblems "inherit" all of the constraints from their parent node and they are also extended with new linear constraints: $x_{i_0} \leq \lfloor x_{i_0}^{*^k} \rfloor$ is added to S^{left} and $x_{i_0} \geq \lceil x_{i_0}^{*^k} \rceil$ is added to S^{right}, where $\lfloor x \rfloor$ means the lower integer part of value x, and $\lceil x \rceil$ means the upper integer part of value x. This type of constraints means only a new upper or lower bound for variable x_{i_0}, so it is not necessary to increase the size of the constraint matrix, it is enough to update the bound section.

In the first level, in case of the root problem, bound is $-\infty$ in case of maximization or ∞ in case of minimization. As we have found a feasible solution we can redefine the bound if the solution found has a better objective value than the current bound. In any case when we find a feasible solution we have to compare the corresponding objective value with the current bound and update the bound if it is necessary. This value has an important role during the B&B process. Branching is needed if and only if the current objective value is not worse (smaller in case of maximization and larger in case of minimization) than the current bound. It is obvious that further branching in a subproblem which already has a worse objective value than the current bound does not result a better feasible solution than that one which objective is equal to the current bound.

Tree building process based on branching procedure can be illustrated by the following figure:



Underlined subproblems are leaf nodes. The reason why a subproblem does not have any descendants can be the following:

- The subproblem has an integer optimal solution.
- The subproblem has no feasible solution.
- When we have to decide if the current subproblem has to be split or not, the current bound is better than the optimal objective value of the current subproblem.

It is obvious this algorithm after simple modification in feasibility-checking and choosing branching variable can be used for Mixed Integer Programming (MIP) problems too.

3.1.1 Branching rules

Suppose that an active node, subproblem S^k, has been selected for branching. Denote the indices of variables with non-integer optimal values by $V = \{j \mid x_j^{*^k} \notin \mathbb{Z}\}$. Practical experience shows that selection of branching variable from set V can dramatically affect the size of binary tree and, hence, also the running time of the algorithm. The most of professional solvers provide users a capability to set the branching variable selection mode. For instance the following table contains the available settings of *variable selection parameter*, CPX_PARAM_VARSEL, in CPLEX.

Value	Meaning
-1	Branch strictly at the nearest integer value which is closest to the fractional variable.
1	Branch strictly at the nearest integer value which is furthest from the fractional variable.
0	CPLEX automatically decides each branch direction.
2	Use pseudo costs, which derive an estimate about the effect of each proposed branch from duality information.
3	Use strong branching, which invests considerable effort in analyzing potential branches in the hope of drastically reducing the number of nodes that will be explored.
4	Use pseudo reduced costs, which are a computationally less intensive form of pseudo costs.

Table 3.1: Branching variable choice settings in CPLEX ([50])

LINGO, as many other solvers, has fewer possible options. It allows only to specify a priority, it means that an ordering of variables can be defined and the branching variable is chosen according to this order. In case of LINGO it is a very simple ordering, we have only one option, namely we can give higher priority for binary variables ([34]).

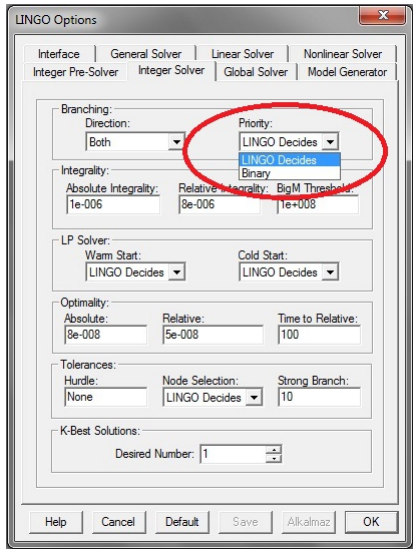


Figure 3.1: LINGO branching priority settings

3.1.2 Searching strategies

Given a set N of active subproblems, i.e. set of unpruned nodes of the current B&B tree. There are several rules for selecting the node should be processed next. Two basic categories of these rules can be mentioned: rules based on the place of nodes in the tree (Last-In-First-Out) and rules that use other information (estimations, reduced cost, etc.). According to this CPLEX has the following possible alternatives, which can be set by define the value of *node selection parameter*, CPX_PARAM_NODESEL.

Value	Meaning
1	Best Bound search, which means that the node with the best objective function will be selected, generally near the top of the tree.
2	Best Estimate search, whereby CPLEX will use an estimate of a given node's progress toward integer feasibility relative to its degradation of the objective function. This setting can be useful in cases where there is difficulty in finding feasible solutions or in cases where a proof of optimality is not crucial.
3	A variation on the Best Estimate search.
0	Depth First search will be conducted. In many cases this amounts to a brute force strategy for solving the combinatorial problem, gaining a small amount of tactical efficiency due to a variety of reasons, and it is rare that it offers any advantage over other settings.

Table 3.2: CPLEX node selection strategies ([50])

LINGO has similar settings, in case of API the parameter NODESL should be defined ([34]), in case of graphical interface the following figure shows the appropriate combobox.

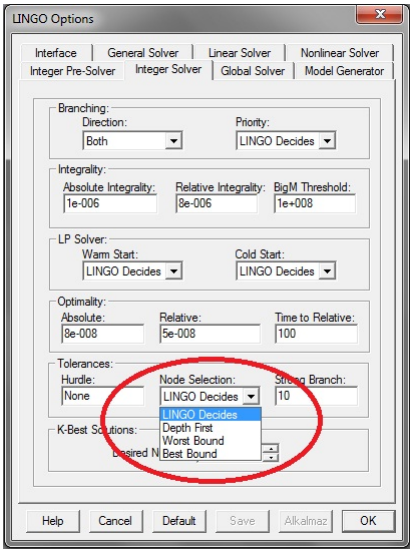


Figure 3.2: LINGO node selection settings

3.1.3 Initial bound

It is well known that the performance of the B&B method mainly depends on the following three main factors:

- the rule used for choosing the branching variable,
- the strategy used for generating binary search tree and
- the value of the initial bound.

Numerous efforts have been made in past decades to investigate general properties and behavior of B&B method, e.g. [9], [17], [18], [30], [32], [38], [47], [49], to improve its computational efficiency, e.g. [10], [19], [20], [21], [33], to maximize its performance in different computational environments, see for example [16], [48], etc.

Generally speaking, while branching variable and searching strategy determine the size of binary tree to be generated, getting a "good" bound as soon as possible can dramatically reduce the size of the tree to be considered, since bound is used to prune those parts of a tree where the value of the objective function cannot be better than the bound. My aim was to give a general algorithm which can provide a bound before we start the tree-building. For this I

give a procedure which can define a feasible integer solution aspiring to the best possible objective value using minimal computational effort and time. In the following section I overview the algorithm which inspired my work after that I present my new procedure for searching an initial bound.

3.2 Ray method

Nowadays many researchers in different countries investigates IP problems in order to develop alternative algorithms for problems with discrete variables. One of such investigations was performed in the Computer Center of the Russian Academy of Sciences [24], [25]. Schematically, the method developed by Khachaturov et al. is based on the search of "better" feasible integer solutions in some special set along some direction called "ray". In this section I present my new algorithm for determining initial bound for the Branch and Bound (B&B) method. The idea of this algorithm is based on the use of "ray" as introduced in the "ray-method" developed for solving integer linear programming problems [24], [25]. Instead of solving an integer programming problem I use the main idea of the ray-method to find an integer feasible solution of an integer linear programming problem along the ray as close to an optimal solution of the relaxation problem as possible. The objective value obtained in this manner is used as an initial bound for B&B method. As it was mentioned in the previous section, getting a "good bound" as soon as possible can often significantly increase the performance of B&B method.

3.2.1 Original ray-method

Let us briefly overview the original ray-method - its mathematical background, its general scheme, different ways to define the ray, and implementation issues.

Originally, the method was developed for a non-linear integer programming problem of the following form:

$$f(x) \to \min \tag{3.4}$$

st.

$$x \in S \subset R^n, \tag{3.5}$$

$$x = (x_1, x_2, \ldots, x_n), \quad x_j - \text{ integer}, \ j = 1, 2, \ldots, n, \tag{3.6}$$

where feasible set

$$S = \{x \in R^n \mid g_i(x) \leq b_i,\ i = 1, 2, \ldots, m\}$$

is a convex, bounded and non-empty set, objective function $f(x)$ is non-linear differentiable $\forall x \in S$ and bounded from below on S, functions $g_i(x),\ i = 1, 2, \ldots, m$, are non-linear and differentiable.

3.2.1.1 Mathematical background

Definition 1 *Let be given a feasible integer point $x^0 \in R^n$. We say that integer point x' belongs to neighborhood set $O(x^0)$, i.e. $x' \in O(x^0)$ if values*

$$\left|x_1^0 - x_1'\right|, \quad \left|x_2^0 - x_2'\right|, \quad \ldots, \quad \left|x_n^0 - x_n'\right|$$

are relative primes.

Note that $O(x^0)$ has the following obvious properties.

Property 1 $x^0 \notin O(x^0)$.

Property 2 *If point $x' \in O(x^0)$, then on the straight line open segment (x^0, x') there is no integer point, i.e. there is no integer point x such that*

$$x \ = \ x^0 + \lambda(x' - x^0), \quad 0 < \lambda < 1\ .$$

Let $S(x^0)$ denote the subset of feasible set S, where $f(x)$ is strictly less than $f(x^0)$, i.e.

$$S(x^0) \ = \ \{x \in S \mid f(x) < f(x^0)\}.$$

Using this notation we can formulate the following optimization criteria.

Theorem 3.1 *Feasible integer point x^0 is an optimal solution for problem (3.4)-(3.6) if and only if*

$$O(x^0) \ \cap \ S(x^0) \ = \ \emptyset \tag{3.7}$$

For **Proof** see [24], [25].

3.2.1.2 General scheme

Let x^0 be an integer feasible solution for problem (3.4)-(3.6). In accordance with the main idea of the method we have to find such an integer point $x' \in O(x^0)$ that $x' \in O(x^0) \cap S(x^0)$. If $O(x^0) \cap S(x^0) = \emptyset$, then x^0 is an optimal solution for problem (3.4)-(3.6). The problem is solved.

Otherwise, i.e. if $O(x^0) \cap S(x^0) \neq \emptyset$, we solve the following one-variable minimization problem:

$$f(\lambda) = f(x^0 + \lambda(x' - x^0)) \to \min \tag{3.8}$$

$$x^0 + \lambda(x' - x^0) \in S \tag{3.9}$$

$$\lambda \geq 0. \tag{3.10}$$

Since both points x^0 and x' belong to the convex bounded non-empty feasible set S, constraint (3.9) defines the segment of straight line $x^0 + \lambda(x' - x^0)$, which belongs to S. So constraints (3.9) and (3.10) determine a non-empty bounded subset of S. Since original objective function $f(x)$ is continuous $\forall x \in S$ and bounded from below, it means that function $f(\lambda)$ is continuous and bounded too on any subset of S. The latter means that problem (3.8)-(3.10) is solvable and may be solved by any suitable numerical method. Let λ_{min} be its optimal solution and $[\lambda_{min}]$ denote its integer part. Concerning value $[\lambda_{min}] + 1$ it may occur that

$$x^0 + ([\lambda_{min}] + 1)(x' - x^0) \notin S, \tag{3.11}$$

or

$$f(x^0 + ([\lambda_{min}] + 1)(x' - x^0)) \geq f(x^0 + [\lambda_{min}](x' - x^0)) \,. \tag{3.12}$$

Now we construct the following point:

$$x'' = \begin{cases} x^0 + [\lambda_{min}](x' - x^0), & \text{if (3.11) or (3.12) takes place,} \\ x^0 + ([\lambda_{min}] + 1)(x' - x^0), & \text{otherwise.} \end{cases}$$

In other words, first, solving problem (3.8)-(3.10) we search the minimal value of function (3.8) on the ray beginning from point x^0 and passing through point x'. Then on this ray we have to find an integer feasible point x'' for which $f(x'')$ is most close to value $f(\lambda_{min})$. In the next step we denote point x'' by x^0 and repeat the process. The new set $S(x^0)$ differs from the previous one by only one constraint $f(x) <= f(x^0) = f(x'')$.

Since the number of integer points in feasible set S is bounded, the process will terminate in a finite number of iterations.

3.2.1.3 Different rules to define the ray

Let point x^0 be an integer feasible solution for problem (3.4)-(3.6), i.e. $x^0 \in S$. We will say that $L = \{x \in R^n \mid x = x^0 + \lambda l,\ \lambda \geq 0\}$, where $l = (l_1, l_2, \ldots, l_n) \in R^n$, is a ray, if there exists $\lambda' > 0$ such that $f(x^0 + \lambda' l) < f(x^0)$. Using this notation, we describe here the following three ways by [25] for constructing a ray.

Procedure 1: Let x^* be the optimal solution of the relaxation problem (i.e. the problem without the integrality constraints) (3.4)-(3.5). Define ray L as

$$L \ = \ \{x \in R^n \mid x = x^0 + \lambda\,(x^* - x^0),\ \lambda \geq 0\}. \tag{3.13}$$

Procedure 2: Calculate the gradients $\nabla g_i(x)$ at the point x^0 for all $g_i(x),\ i = 1, 2, \ldots, m,$ functions, and introduce rays $L_i,\ i = 1, 2, \ldots, m$ in the following way:

$$L_i \ = \ x^0 + \lambda\,\nabla g_i(x^0), \quad \lambda \geq 0,$$

if exists such $\lambda' > 0$, that $f(x^0 + \lambda'\,\nabla g_i(x^0)) < f(x^0)$. And

$$L_i \ = \ x^0 - \lambda\,\nabla g_i(x^0), \quad \lambda \geq 0,$$

otherwise.

Procedure 3: Choose the following formula for the ray.

$$L \ = \ x^0 - \lambda\,\nabla f(x^0), \quad \lambda \geq 0. \tag{3.14}$$

3.2.1.4 Implementation issues

In contrast to the transparency of theoretical background for the method there are serious difficulties with its implementation and computational efficiency. The main and most serious of them is checking optimality criteria for a given feasible integer point x^0 since set $O(x^0)$ may contain a huge number of integer points. Note that, as it was mentioned above, these integer points were selected on the basis of usage of relative prime numbers, so determining these integer points may be a very hard and computationally very expensive problem.

3.2.2 The new method proposed

Here I explain the main idea of my algorithm proposed: first of all I define the ray, and then describe the main steps of the procedure. Finally, using a small illustrative numerical example, I show how my new algorithm is utilized.

As it was mentioned above, the ray-method was originally developed as a method for solving non-linear integer programming problems. Using the main idea of the method, below I propose a new algorithm which is used for determining an initial bound in B&B method when solving integer linear programming problems.

3.2.2.1 Preliminaries

Consider the following pure integer linear programming minimization problem:

$$f(x) = \sum_{j=1}^{n} c_j x_j \to \min \tag{3.15}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1, 2, \ldots, m, \tag{3.16}$$

$$x_j \ge 0, \quad \text{integer}, \quad j = 1, 2, \ldots, n. \tag{3.17}$$

Here and in what follows we assume that relaxation problem (3.15)-(3.17) is solvable (i.e. has a non-empty feasible set and objective function $f(x)$ over the feasible set has a finite lower bound) and vector

$$x^{min} = (x_1^{min}, x_2^{min}, \ldots, x_n^{min})$$

is its relaxation non-integer solution. Furthermore, we suppose that the corresponding maximization relaxation problem is solvable too, and

$$x^{max} = (x_1^{max}, x_2^{max}, \ldots, x_n^{max})$$

is its optimal solution. These two assumptions play a very important role in my new algorithm since I use points x^{min} and x^{max} to determine the ray for indicating the direction of the search. Moreover we assume that $x^{min} \ne x^{max}$.

3.2.2.2 Main steps

Using the given notation, my new algorithm proposed is described in the following steps.

0. Initial point: Let us denote point x^{min} by x^0, and $l = (l_1, l_2, \ldots, l_n)$, where $l_j = x_j^{max} - x_j^{min}$, $j = 1 \ldots n$.

1. Ray: Define the ray in the following way:

$$L \;=\; x^0 + \lambda(x^{max} - x^0),\;\; 0 \le \lambda \le 1\;.$$

Note that since feasible set S is convex, it means that all points of L are elements of set S.

2. Constructing set $O(x^0)$: Let J^0 be a set of indexes of integer components of x^0, i.e. $J^0 = \{j \in J |\; x_j^0 = [x_j^0]\}$, where $J = \{1, 2, \ldots, n\}$. Define set $O(x^0)$ as the set of such points x which satisfy the following constraints:

$$\left.\begin{array}{lcll} [x_j^0] & \le\; x_j \;\le & [x_j^0] + 1, & \text{if } j \notin J^0\;, \\ [x_j^0] & \le\; x_j \;\le & [x_j^0] + 1, & \text{if } j \in J^0 \text{ and } l_j > 0\;, \\ [x_j^0] - 1 & \le\; x_j \;\le & [x_j^0], & \text{if } j \in J^0 \text{ and } l_j < 0\;, \\ & x_j \;= & [x_j^0], & \text{if } j \in J^0 \text{ and } \; l_j = 0\;. \end{array}\right\} \tag{3.18}$$

Generally speaking $O(x^0)$ is the unit-cube containing the point x^0. If $J^0 = \emptyset$, then the dimension of this unit-cube is n.

Before starting the iterations define the point $x' := x^0$ and calculate the first perforation point $P_{act} = (p_1, p_2, \ldots, p_n)$ solving the following optimization problem:

$$\lambda \to \max \tag{3.19}$$

subject to

$$p_j = x_j^0 + \lambda l_j \quad j = 1, 2, \ldots, n, \tag{3.20}$$

$$\left.\begin{array}{lcll} [x_j^0] & \le\; p_j \;\le & [x_j^0] + 1, & j \notin J^0\;, \\ [x_j^0] & \le\; p_j \;\le & [x_j^0] + 1, & j \in J^0 \text{ and } l_j > 0\;, \\ [x_j^0] - 1 & \le\; p_j \;\le & [x_j^0], & j \in J^0 \text{ and } l_j < 0\;, \\ & p_j \;= & [x_j^0], & j \in J^0 \text{ and } l_j = 0\;. \end{array}\right\} \tag{3.21}$$

3. Shifting: Enter new variables $y_j = x_j - [x_j']$, $j = 1, 2, \ldots, n$, and construct new feasible set S' in the following way

$$S' : \qquad \sum_{j=1}^{n} a_{ij} y_j \le b_i',\;\; i = 1, 2, \ldots, m\;,$$

where

$$b'_i \;=\; b_i - \sum_{j=1}^{n} a_{ij}[x'_j], \;\; i = 1, 2, \ldots, m \; .$$

Obviously, set S' is the intersection of the current set $O(x')$ and feasible set S shifted to point 0. Then solve the following 0-1 LP problem

$$f'(y) = \sum_{j=1}^{n} c_j y_j + c_0 \rightarrow \max \tag{3.22}$$

subject to

$$\sum_{j=1}^{n} a_{ij} y_j \leq b'_i, \;\; i = 1, 2, \ldots, m, \tag{3.23}$$

$$y_j = 0/1, \quad j = 1, 2, \ldots, n, \tag{3.24}$$

where

$$c_0 = \sum_{j=1}^{n} c_j [x'_j] \; ,$$

using **Balas'** additive algorithm (implicit enumeration) [8]. If problem (3.22)-(3.24) has 0-1 optimal solution y^*, then vector $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is determined, where

$$x_j^* = y_j^* + [x'_j], \;\; j = 1, 2, \ldots, n \; ,$$

and value $f(x^*) = f'(y^*)$ is used as an initial bound for the branch and bound method. **Stop**.
Otherwise,

4. Perforation point: Determine point P_{next} where the ray "perforates" the hull of the next unit-cube along the ray solving the following optimization problem:

$$\lambda \rightarrow \max \tag{3.25}$$

subject to

$$x_j = x_j^0 + \lambda l_j \quad j = 1, 2, \ldots, n \; , \tag{3.26}$$

$$\left.\begin{array}{lcll} [p_j] & \leq \; x_j \; \leq & [p_j] + 1, & j \notin J' \; , \\ [p_j] & \leq \; x_j \; \leq & [p_j] + 1, & j \in J' \text{ and } l_j > 0 \; , \\ [p_j] - 1 & \leq \; x_j \; \leq & [p_j], & j \in J' \text{ and } l_j < 0 \; , \\ & x_j \; = & [p_j], & j \in J' \text{ and } l_j = 0 \; , \end{array}\right\} \tag{3.27}$$

where $J' = \{j \in J |\ p_j = [p_j]\}$. Here constraints (3.20) and (3.21) provide $P_{next} \in L$ and $P_{next} \in O(P_{act})$, correspondingly. Obviously, this problem is solvable, i.e. has a non-empty feasible set and its objective function is bounded from above. Let $P_{next} = (p'_1, p'_2, \ldots, p'_n)$ solve problem (3.19)-(3.21) and λ' be the maximal value of objective function (3.19).

Now, define the middle point x' of section $[P_{act};\ P_{next}]$:

$$x'_j = \frac{p_j + p'_j}{2} \quad j = 1, 2, \ldots, n\ . \tag{3.28}$$

Furthermore, point P_{act} is not needed any more, so I overwrite it with the value of P_{next}, i.e. $P_{act} := P_{next}$.

5. Next unit-cube: Having point x' I determine the next unit-cube along the ray using the following rule:

$$\left.\begin{array}{lcll} [x'_j] & \leq x_j \leq & [x'_j]+1, & \text{if } j \notin J'\ , \\ [x'_j] & \leq x_j \leq & [x'_j]+1, & \text{if } j \in J' \text{ and } l_j > 0\ , \\ [x'_j]-1 & \leq x_j \leq & [x'_j], & \text{if } j \in J' \text{ and } l_j < 0\ , \\ & x_j = & [x'_j], & \text{if } j \in J' \text{ and } l_j = 0\ , \end{array}\right\} \tag{3.29}$$

where $J' = \{j \in J |\ x'_j = [x'_j]\}$. Go to step 3.

Since the number of unit-cubes "perforated" by the ray is finite, the process will terminate in a finite number of iterations. It may occur that when determining next perforation point x' $\lambda' > 1$ is obtained. It means that unit-cubes constructed along the ray do not contain any integer feasible point for original problem (3.15)-(3.17). It means the method fails.

3.2.3 Adaptation for MIP problems

Here I give the appropriate modification of my algorithm proposed for mixed integer programming (MIP) problems too. Consider the following MIP problem in general form:

$$f(x) = \sum_{j=1}^{n} c_j x_j \to \min \tag{3.30}$$

subject to

$$\sum_{j=1}^{n} a_{ij}x_j \le b_i, \quad i = 1, 2, \ldots, m, \tag{3.31}$$

$$x_j \ge 0, \ j = 1, 2, \ldots, n. \tag{3.32}$$

$$x_j - \text{ integer}, \ j \in J^i \subset J = \{1, 2, \ldots, n\} \tag{3.33}$$

0. Initial point: Let us denote point x^{min} by x^0, and $l = (l_1, l_2, \ldots, l_n)$, where $l_j = x_j^{max} - x_j^{min}$, $j = 1 \ldots n$.

1. Ray: Define the ray as above:

$$L \ = \ x^0 + \lambda(x^{max} - x^0), \ \ 0 \le \lambda \le 1 \ .$$

2. Constructing set $O(x^0)$: J^0 is a set of indexes of integer components of x^0, i.e. $J^0 = \{j \in J |\ x_j^0 = [x_j^0]\}$, where $J = \{1, 2, \ldots, n\}$. I define set $O(x^0)$ as the set of such points x which satisfy the following constraints:

$$\left.\begin{array}{lcll} [x_j^0] & \le \ x_j \ \le & [x_j^0] + 1, & \text{if } j \in J^i \setminus J^0, \\ [x_j^0] & \le \ x_j \ \le & [x_j^0] + 1, & \text{if } j \in J^i \cap J^0 \text{ and } l_j > 0, \\ [x_j^0] - 1 & \le \ x_j \ \le & [x_j^0], & \text{if } j \in J^i \cap J^0 \text{ and } l_j < 0, \\ & x_j \ = & [x_j^0], & \text{if } j \in J^i \cap J^0 \text{ and } \ l_j = 0. \end{array}\right\} \tag{3.34}$$

Define the point $x' := x^0$ and calculate the first perforation point $P_{act} = (p_1, p_2, \ldots, p_n)$ solving the following optimization problem:

$$\lambda \to \max \tag{3.35}$$

subject to

$$p_j = x_j^0 + \lambda l_j \quad j = 1, 2, \ldots, n, \tag{3.36}$$

$$\left.\begin{array}{lcll} [x_j^0] & \le \ p_j \ \le & [x_j^0] + 1, & j \in J^i \setminus J^0 \ , \\ [x_j^0] & \le \ p_j \ \le & [x_j^0] + 1, & j \in J^i \cap J^0 \text{ and } l_j > 0 \ , \\ [x_j^0] - 1 & \le \ p_j \ \le & [x_j^0], & j \in J^i \cap J^0 \text{ and } l_j < 0 \ , \\ & p_j \ = & [x_j^0], & j \in J^i \cap J^0 \text{ and } l_j = 0 \ . \end{array}\right\} \tag{3.37}$$

3. Shifting: Enter new variables $y_j = x_j - [x'_j],\ j = 1, 2, \ldots, n,$ and construct new feasible set S' in the following way

$$S' : \qquad \sum_{j=1}^{n} a_{ij} y_j \le b'_i,\ \ i = 1, 2, \ldots, m\ ,$$

where

$$b'_i \ = \ b_i - \sum_{j=1}^{n} a_{ij}[x'_j],\ \ i = 1, 2, \ldots, m\ .$$

Then solve the following mixed binary LP problem

$$f'(y) = \sum_{j=1}^{n} c_j y_j + c_0 \to \max \tag{3.38}$$

subject to

$$\sum_{j=1}^{n} a_{ij} y_j \le b'_i,\ \ i = 1, 2, \ldots, m, \tag{3.39}$$

$$y_j = 0/1, \quad j \in J^i \tag{3.40}$$

$$0 \le y_j \le 1, \quad j \in J \setminus J^i \tag{3.41}$$

where

$$c_0 = \sum_{j=1}^{n} c_j [x'_j]\ ,$$

If problem (3.22)-(3.24) has an optimal solution y^*, then vector $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is determined, where

$$x_j^* = y_j^* + [x'_j],\ \ j = 1, 2, \ldots, n\ ,$$

and value $f(x^*) = f'(y^*)$ is used as an initial bound for the branch and bound method. **Stop**.
Otherwise,

4. Perforation point: I determine point P_{next} as follows:

$$\lambda \to \max \tag{3.42}$$

subject to

$$x_j = x_j^0 + \lambda l_j \quad j = 1, 2, \ldots, n\ , \tag{3.43}$$

$$\left.\begin{array}{lcllll} [p_j] & \leq & x_j & \leq & [p_j]+1, & j \in J^i \setminus J' \ , \\ [p_j] & \leq & x_j & \leq & [p_j]+1, & j \in J^i \cap J' \text{ and } l_j > 0 \ , \\ [p_j]-1 & \leq & x_j & \leq & [p_j], & j \in J^i \cap J' \text{ and } l_j < 0 \ , \\ & & x_j & = & [p_j], & j \in J^i \cap J' \text{ and } l_j = 0 \ , \end{array}\right\} \tag{3.44}$$

where $J' = \{j \in J |\ p_j = [p_j]\}$. Suppose that $P_{next} = (p'_1, p'_2, \ldots, p'_n)$ solve problem (3.19)-(3.21) and λ' be the maximal value of objective function (3.19).

Let us define middle point x' of section $[P_{act};\ P_{next}]$:

$$x'_j = \frac{p_j + p'_j}{2} \quad j = 1, 2, \ldots, n \ . \tag{3.45}$$

and $P_{act} := P_{next}$.

5. Next unit-cube: Having point x' determine the next unit-cube along the ray using the following rule:

$$\left.\begin{array}{lcllll} [x'_j] & \leq & x_j & \leq & [x'_j]+1, & \text{if } j \in J^i \setminus J' \ , \\ [x'_j] & \leq & x_j & \leq & [x'_j]+1, & \text{if } j \in J^i \cap J' \text{ and } l_j > 0 \ , \\ [x'_j]-1 & \leq & x_j & \leq & [x'_j], & \text{if } j \in J^i \cap J' \text{ and } l_j < 0 \ , \\ & & x_j & = & [x'_j], & \text{if } j \in J^i \cap J' \text{ and } \ l_j = 0 \ , \end{array}\right\} \tag{3.46}$$

where $J' = \{j \in J |\ x'_j = [x'_j]\}$. Go to step 3.

3.2.4 An illustrative numerical example

Here I illustrate the main steps of the method proposed using a small numerical example. My method proposed was partially implemented in the frame of the educational linear and linear-fractional package WinGULF [4]. The package has numerous options for the B&B method - we can choose the direction of the search (first left node and then right one or vice versa), different rules for selecting a branching variable (for example, "fractional part most close to 0.5", "smallest fractional part", "biggest fractional part", "smallest value", "biggest value", etc.), user defined initial bound, etc. When testing the method proposed, most of the options built in were used. For testing on large MIPLIB problems I implemented my ray-method in C++ using CPLEX callable library

for solving subproblems.

Consider the following numerical example:

$$\begin{array}{rcrrrrrr} f(x) & = & 20x_1+ & 21x_2+ & 18x_3 & \to & & \min \\ \text{subject to} & & & & & & & \\ & & 9x_1+ & 1.5x_2+ & 7x_3 & \leq & 1350\,, \\ & & 5.5x_1+ & 1x_2+ & 9x_3 & \geq & 1250\,, \\ & & -4.5x_1- & 10x_2+ & 2.5x_3 & \leq & -1050\,, \end{array}$$

$$x_1,\ x_2,\ x_3\ -\ \text{integer.}$$

Solving both (minimization and maximization) relaxation problems we obtain the following:

$$\begin{array}{rcrrr} x^{min} & = & (\ 65.042, & 97.799, & 88.274)\,, \\ x^{max} & = & (\ 0.000, & 523.076, & 80.769)\,, \\ L & = & (-65.042, & 425.277, & -7.504)\,. \end{array}$$

Let us denote x^{min} with x^0 and construct set $O(x^0)$, i.e. the following unit-cube:

$$O(x^0): \begin{cases} 65 \leq x_1 \leq 66\,, \\ 97 \leq x_2 \leq 98\,, \\ 88 \leq x_3 \leq 89\,. \end{cases}$$

So the first shifted problem is constructed:

$$\begin{array}{rcrrrrrr} f'(y) & = & 20y_1+ & 21y_2+ & 18y_3 & + & 4921 & \to \max \\ \text{st.} & & & & & & & \\ & & 9y_1+ & 1.5y_2+ & 7y_3 & \leq & 3.5\,, \\ & & 5.5y_1+ & 1y_2+ & 9y_3 & \geq & 3.5\,, \\ & & -4.5y_1- & 10y_2+ & 2.5y_3 & \leq & -7.5\,, \end{array}$$

$$y_1,\ y_2,\ y_3\ -\ 0/1$$

and solve it. Since the problem is infeasible, next unit-cube along the ray have to be determined. In order to obtain the next unit-cube first solve the following problem (see (3.19)-(3.21)):

$$\lambda \to \max$$

subject to

$$\left.\begin{aligned} x_1 &= 65.042 + \lambda(-65.042)\ , \\ x_2 &= 97.799 + \lambda(425.277)\ , \\ x_3 &= 88.274 + \lambda(-7.504)\ , \end{aligned}\right\}$$

$$\left.\begin{aligned} 65 \leq\ & x_1\ , & \\ & x_2 & \leq 98\ , \\ 88 \leq\ & x_3 & \end{aligned}\right\}$$

and obtain the first perforation point P_1:

$$\lambda' \ =\ 0.00047, \quad P_1 \ =\ (65.011,\ 98,\ 88.270)\ .$$

To find the next perforation point P_2 the following problem has to be solved:

$$\lambda \rightarrow \max$$

subject to

$$\left.\begin{aligned} x_1 &= 65.042 + \lambda(-65.042)\ , \\ x_2 &= 97.799 + \lambda(425.277)\ , \\ x_3 &= 88.274 + \lambda(-7.504)\ , \end{aligned}\right\}$$

$$\left.\begin{aligned} 65 \leq\ & x_1\ , & \\ & x_2 & \leq 99\ , \\ 88 \leq\ & x_3\ , & \end{aligned}\right\}$$

so I obtain

$$\lambda'' \ =\ 0.00064, \quad P_2 \ =\ (65,\ 98.07,\ 88.26)\ .$$

Using these points P_1 and P_2 I find the middle point

$$x' \ =\ (65.006,\ 98.03,\ 88.26)\ .$$

This point allows us to construct the next shifted problem:

$$\begin{array}{rcrrrrrrl} f'(y) & = & 20y_1+ & 21y_2+ & 18y_3 & + & 4942 & \to \max \\ \text{st.} & & & & & & & \\ & & 9y_1+ & 1.5y_2+ & 7y_3 & \le & 2\,, & \\ & & 5.5y_1+ & 1y_2+ & 9y_3 & \ge & 2.5\,, & \\ & & -4.5y_1- & 10y_2+ & 2.5y_3 & \le & -2.5\,, & \end{array}$$

$$y_1,\ y_2,\ y_3\ -\ 0/1\ .$$

This problem has no feasible solution.

Proceeding to the next perforation point P_3, we obtain the following optimization problem to solve

$$\lambda \to \max$$

subject to

$$\left.\begin{array}{l} x_1 = 65.042 + \lambda(-65.042)\,, \\ x_2 = 97.799 + \lambda(425.277)\,, \\ x_3 = 88.274 + \lambda(-7.504)\,, \end{array}\right\}$$

$$\left.\begin{array}{rrl} 64 \le & x_1\,, & \\ & x_2 & \le 99\,, \\ 88 \le & x_3\,. & \end{array}\right\}$$

We obtain $\lambda = 0.0028$ and the next perforation point $P_3 = (64.85,\ 99,\ 88.25)$. Therefore the next middle point is $x' = (64.93,\ 98.53,\ 88.26)$ and the shifted problem is as follows:

$$\begin{array}{rcrrrrrrl} f'(y) & = & 20y_1+ & 21y_2+ & 18y_3 & + & 4922 & \to \max \\ \text{subject to} & & & & & & & \\ & & 9y_1+ & 1.5y_2+ & 7y_3 & \le & 11\,, & \\ & & 5.5y_1+ & 1y_2+ & 9y_3 & \ge & 8\,, & \\ & & -4.5y_1- & 10y_2+ & 2.5y_3 & \le & -2\,, & \end{array}$$

$$y_1,\ y_2,\ y_3\ -\ 0/1\ .$$

The optimal solution of this problem is $y^* = (0,\ 1,\ 1)$ and $f'(y^*) = 4961$. This value was used as an initial bound. The corresponding integer point is $x^* = (64,\ 99,\ 89)$.

Note that the initial bound obtained (4961) is very close to the optimal value (after solving the problem we obtain 4959). Below is presented a table with results obtained from WinGULF after running B&B method on this numerical example using different strategies (from left to right and vice versa) and different branching rules. *"Wo.I.B."* means without an initial bound and *"W.I.B."* means with an initial bound.

Branching variable	Left → Right		Right → Left	
	Wo.I.B.	W.I.B.	Wo.I.B.	W.I.B.
Minimal index	291	37	37	33
Maximal index	31	23	243	23
Max. value	33	25	245	25
Min. value	31	25	25	25
Max. fract. part	105	41	37	37
Min. fract. part	31	23	23	23
Most close to 0.5	31	25	25	25

In column "Wo.I.B." are the size of the tree (number of its nodes) built while solving the problem without my initial bound, and in column "W.I.B." number of nodes for solving the same problem using my initial bound can be seen. As we can see also on this simple example using my initial bound defining technique we can reach an enormous reduction in size of binary tree.

3.2.5 Implementation issues

Considering the optimization subproblems (3.19)-(3.21) (and also (3.35)-(3.37)) I suggest the following solution instead of this optimization problem. Here I show that these optimization subproblems can be replaced with a simple formula.

First of all I replace variables x_j in (3.21) with the right-hand-side expressions of (3.20). This replacement allows us to transform the original $n+1$ variable optimization problem to the following one-variable system of constraints:

$$\left\{\begin{array}{lcll} [x_j^0] & \leq & x_j^0+\lambda l_j \leq [x_j^0]+1, & \text{if } j\notin J^0\ , \\ [x_j^0] & \leq & x_j^0+\lambda l_j \leq [x_j^0]+1, & \text{if } j\in J^0 \text{ and } l_j>0\ , \\ [x_j^0]-1 & \leq & x_j^0+\lambda l_j \leq [x_j^0], & \text{if } j\in J^0 \text{ and } l_j<0\ , \\ & & x_j^0+\lambda l_j = [x_j^0], & \text{if } j\in J^0 \text{ and } \ l_j=0\ . \end{array}\right. \tag{3.47}$$

Now consider the left-hand-side constraints of (3.47), i.e.

$$\left\{\begin{array}{lcll} [x_j^0] & \leq & x_j^0+\lambda l_j, & \text{if } j\notin J^0\ , \\ [x_j^0] & \leq & x_j^0+\lambda l_j, & \text{if } j\in J^0 \text{ and } l_j>0\ , \\ [x_j^0]-1 & \leq & x_j^0+\lambda l_j, & \text{if } j\in J^0 \text{ and } l_j<0\ , \end{array}\right. \tag{3.48}$$

and rewrite it in the following form

$$\lambda l_j \geq [x_j^0]-x_j^0, \ \text{ if } j\notin J^0\ , \tag{3.49}$$

$$\lambda l_j \geq [x_j^0]-x_j^0-1, \ \text{ if } j\in J^0 \text{ and } l_j>0\ , \tag{3.50}$$

$$\lambda l_j \geq [x_j^0]-x_j^0, \ \text{ if } j\in J^0 \text{ and } l_j<0\ , \tag{3.51}$$

Introducing the following notations for index j:

$$J^*=\{j\in J|\, x_j^0\neq[x_j^0]\},\quad J_+^*=\{j\in J^*|\, l_j>0\},\quad J_-^*=\{j\in J^*|\, l_j<0\},$$

$$J_+^0=\{j\in J^0|\, l_j>0\},\quad J_-^0=\{j\in J^0|\, l_j<0\},\quad J_0^0=\{j\in J^0|\, l_j=0\}.$$

Now consider constraints (3.49), (3.50), (3.51) separately. From (3.49) I have

$$\lambda\geq\frac{[x_j^0]-x_j^0}{l_j},\quad \forall j\in J_+^* \tag{3.52}$$

and

$$\lambda\leq\frac{[x_j^0]-x_j^0}{l_j},\quad \forall j\in J_-^* \tag{3.53}$$

Note, that constraints (3.52) may be used to determine the lower bound for value λ and only constraints (3.53) can be used to determine the upper bound for value

λ. Since problem (3.19)-(3.21) is a maximization one constraints (3.52) is not considered and I use only constraints (3.53). So, let K_1 denote the following

$$K_1 = \min_{j \in J_-^*} \frac{[x_j^0] - x_j^0}{l_j}$$

Hence, (3.49) gives the following upper bound for λ

$$\lambda \leq K_1 \tag{3.54}$$

Analogously, from (3.50) I obtain

$$\lambda \leq \frac{[x_j^0] - x_j^0 - 1}{l_j}, \quad \forall j \in J_-^0 \tag{3.55}$$

Note, in accordance with definition of set J^0 I have, that $x_j^0 = [x_j^0]$, $\forall j \in J^0$, so (3.55) may be rewritten in the following form

$$\lambda \leq -\frac{1}{l_j}, \quad \forall j \in J_-^0 \tag{3.56}$$

Let us denote

$$K_2 = \min_{j \in J_-^0} \frac{-1}{l_j}$$

So I obtain the following new upper bound for λ

$$\lambda \leq K_2 \tag{3.57}$$

Constraints (3.51) is not considered since these constraints do not provide any upper bound for value λ.

Processing in the same manner the right-hand-side conditions of system (3.47), I obtain

$$\lambda \leq K_3 \text{ and } \lambda \leq K_4, \tag{3.58}$$

where

$$K_3 = \min_{j \in J_+^*} \frac{[x_j^0] + 1 - x_j^0}{l_j}$$

$$K_4 = \min_{j \in J_+^0} \frac{1}{l_j}$$

Finally, combining (3.54), (3.57) and (3.58) I obtain the least upper bound for value λ subject to constraints (3.19)-(3.21)

$$\lambda \;=\; \min\{K_1,\; K_2,\; K_3,\; K_4\}$$

Using this simple formula when implementing the ray-method I could replace the optimization subproblem (3.19)-(3.21) (and also (3.35)-(3.37)), and therefore reach some more run time reduction.

3.2.6 Test results

The method proposed was implemented in WinGulf[1]. These test results show not only the efficiency of the method proposed but also the dependence of the size of B&B tree on node and variable selection strategies. For preliminary tests we implemented several options in WinGulf which provide different strategies for users, as the following figures show.

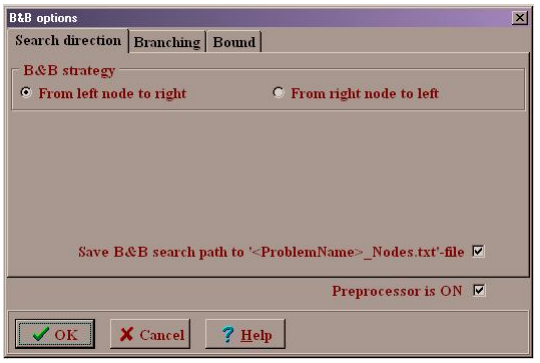


Figure 3.3: WinGulf node selection settings

Our pure integer programming (IP) test files were generated automatically in mps format in different size (25-50 variables and 25-50 constraints). The different selection strategies are separately listed in Table 2, 3 and 4 (in appendix). "Total" means the total number of the nodes in the B&B tree, "Best" is the node of the optimal solution. In column "Difference" are the differences between the two tree built by B&B method with and without our initial bound.

[1]General User-friendly Linear and linear-Fractional programming package for educational purposes. Developed at the University of Debrecen. See: http://zeus.nyf.hu/ bajalinov/WinGulf/wingulf.html

Figure 3.4: WinGulf branching variable selection settings

I used two different node selection strategies:

- "Left to right" : Preorder tree-traversal. (Visit the root, traverse the left subtree, traverse the right subtree.)
- "Right to left" : Visit the root, traverse the right subtree, traverse the left subtree.

I used the following branching variable selection strategies:

- "Min. fract. part" : The noninteger variable with minimal fractional part is chosen as branching variable.
- "Max. fract. part" : The noninteger variable with maximal fractional part is chosen as branching variable.
- "Most close to 0.5" : The noninteger variable with fractional part most close to 0.5 is chosen as branching variable.

These results are presented in tableaus in appendix.

One can see on these relatively small examples my ray-method can produce an initial bound which reduces the size of B&B tree in an average of 16%.

Encouraged by this results I conducted further tests now using professional solver CPLEX and official test files form MIPLIB. Therefore I implemented the MIP adaptation of ray-method discussed in section 3.2.3 using CPLEX callable

library. The structure of test code is as follows:

1. Read .mps or .lp input file.
2. CPLEX solves the problem → *Time without bound*
3. Ray method calculates an initial bound. → *Time to define bound*
4. CPLEX solves the problem using the bound defined in previous step. → *Time with bound*

The following table shows these results. Most of the files can be found in MIPLIB library, contributor is A. Atamtürk [2], [3].

Name	Rows	Cols	Int	Nz	Time without bound (sec)	Time with bound (sec)	Time to define bound (sec)	Improvement
n4-3	1236	3596	174	14036	361,14	288,459	0,254	20%
n12-3	2430	6120	216	24048	69,689	42,633	0,568	38%
n5-3	1062	2550	150	9900	46,335	35,434	0,143	23%
n7-3	2336	5626	174	22156	24,586	21,502	0,31	11%
pr9	2220	7350	42	22176	108,52	86,464	11,618	10%
pr12	2313	5868	36	17712	2,604	1,604	0,4	23%

Table 3.3: Ray method test results on MIP files

The previous table shows the corresponding information about the test problems in the first five column (name, number of rows and columns, number of integer variables and nonzero coefficients). The other columns present the test results. In column "Time without bound" you can find the time in sec which was needed for IBM CPLEX for solving the IP problem without my bound. In column "Time to define bound" is the elapsed time while my method was calculating the initial bound. Finally, column "Time with bound" shows CPLEX's reduced run time while solving the same IP problem but using initial bound calculated by ray-method.

As we can see an overall 20% improvement on run time can be observed in case of these test files. Spending a very little time to calculating an initial bound using my method a significant improvement can be reached on solving IP problems.

Summary and conclusion

Nowadays OR applications highly depend on performance of computers. One of the most urgent challenges is to develop the most efficient implementations of the well-known algorithms whereby we can obtain a faster, numerically more stable and more reliable version. However, computers have even larger performance, real-world OR models are also even more complex so their computational requirement is growing fast.

I was motivated by the reasons described above when started my investigations of different ways to improve stability and performance of computer codes. Namely I focused my research on the following three different domains:

[1] preprocessing and postsolving in LFP

[2] scaling LFP problems and re-scaling solutions obtained, and

[3] getting better initial bound for branch-and-bound method in ILP and MILP

In the first part of the dissertation I consider the different presolve methods which are used for reducing the size of models by removing redundant constraints or fixing variables. When performing these manipulations some data from the dual problem may be lost or distorted. To restore such data some special postsolve operations are required. Based on [1] I developed the corresponding adaptation of presolve methods for Linear-Fractional problems. In this part I also showed the advantages of using my LFP presolve adaptation instead of transforming the LFP model into LA form by Charnes-Cooper transformation and using LP presolve on LA model.

The topic of the second part of section 2.2 discusses another preprocessing technique, the scaling. Scaling methods are used for decreasing the numerical inaccuracies caused by floating-point arithmetic. Scaling also needs some

postsolve ("re-scale") operation to obtain the optimal solution of the original problem from the optimal solution of the scaled, and therefore numerically more stable problem. Here I aimed to adapt scaling for LFP at first, than I developed my scaling rule ([5]) and compared with two other scaling method ([4], [36]) in point of view of efficiency and run time on NETLIB test files. Tests show my new scaling rule can be as efficient as the well known rules and compering with geometric rule my algorithm reached better results on most of the test problems.

Finally, in the third part I discussed my method for improving solving of IP and MIP problems. Our method based on [25] which presents theoretical contexts of optimal solution of IP problems. Using these theorems I developed a new method, called ray-method. Ray-method is used for finding not optimal but a feasible solution of IP problem. Using the objective value of this feasible solution I provide a bound value for B&B method when starting. The requirement for this kind of procedure is to be fast and provide a feasible solution as close to the optimal point as possible. I tested my ray-method on random generated IP problems using WINGULF at first, than I adapted it for MIP problems and developed a test environment using CPLEX Callable Library for comparing the run-time of B&B algorithm with the initial bound found by ray-method or without any initial bound. So I had the opportunity to test my method on large problems from MIPLIB test collection library. According to these test results my new method can improve the efficiency of the professional CPLEX solver by about 21% on large IP and MIP problems.

Összefoglaló

Manapság az operációkutatás gyakorlati alkalmazása nagyban függ a számítógépek teljesítményétől. Egyik legfontosabb probléma a meglevő algoritmusok hatékony alkalmazása számítógépes környezetben, ezért fontos az olyan irányú továbbfejlesztésük, mely során egy gyorsabb, pontosabb és megbízhatóbb implementációt kapunk. Bár a számítógépek egyre nagyobb teljesítményre képesek, az ipari alkalmazások során keletkező modellek is egyre összetettebbek, így erőforrásigényük is egyre növekszik.

Ezen motivációból eredően kezdtem foglalkozni különböző stabilitás- és teljesítmény javító lehetőségeken. Ezen belül három különböző területtel, melyeknek közös célja, hogy gyorsítsák vagy pontosítsák a különböző optimalizációs feladatok számítógépes megoldását.

Az első körben az un. presolve eljárásokkal foglalkoztam, melyek a feladat méretének csökkentését szolgálják azáltal, hogy redundáns feltételeket vagy rögzíthető változókat szűrnek ki és távolítanak el a modellből. Ezen változások miatt azonban duális információk is elvesznek. Ezek helyreállításához szükségesek a kapcsolódó postsolve műveletek. A [1] munkán alapulva kidolgoztam a megfelelő adaptációt hiperbolikus programozási problémákra. A dolgozat ezen részében azt is megmutatom, hogy miért előnyösebb a hiperbolikus presolve adaptáció alkalmazása azzal szemben, hogy a modellt Charnes-Cooper transzformációval lineáris analóg formára alakítsuk és ezen lineáris presolve-ot alkalmazzunk.

A második témakör szintén nevezhető "preprocessing" eljárásnak, un. skálázó eljárásokkal foglalkoztam, melyek a lebegőpontos számábrázolásból adódó pontatlanságok csökkentését szolgálják. Szintén szükségesek bizonyos postsolve (reskálázó) műveletek, hogy a skálázott, ezáltal jobb numerikus tulajdonságokkal rendelkező, feladat megoldásából visszanyerjük az eredeti feladat megoldását. A cél kezdetben itt is a hiperbolikus környezetre való adaptáció volt, majd később egy saját skálázási módszert ([5]) is készítettem és összehasonlítottam két már

ismert eljárással ([4], [36]) teljesítmény és futási idő szempontjából. A tesztek alapján a saját skálázási algoritmusom felveszi a versenyt a meglévő ismert algoritmusokkal, a geometriai közép szabályt tekintve pedig az esetek többségénél jobbnak bizonyult.

Végezetül az egészértékű feladatok megoldásának gyorsítását tűztem ki célul. Az alapötlet a [25] munkán alapult, mely elméleti összefüggéseket mutat be az egészértékű feladatok optimális megoldását illetően. Ezt felhasználva dolgoztam ki egy eljárást sugár-módszer néven, amellyel nem az optimális megoldást sokkal inkább egy lehetséges megoldást szerettünk volna előállítani, amely a B&B módszer számára már a kiinduló pontban egy elérhető korlát értékül szolgál. Az elvárás egy ilyen előkészítő algoritmustól, hogy gyors legyen és lehetőleg minél jobb kiinduló korlátot adjon. A sugár módszert először véletlenszerűen generált tiszta egészértékű feladatokon teszteltem a WINGULF rendszerbe beépítve ([41]), majd átdolgoztam vegyes egészértékű feladatokra is valamint egy tesztkörnyezetet fejlesztettem, amely a CPLEX Callable Library-t használva összehasonlítja a futási időket, amikor a CPLEX használja a sugármódszer által generált korlátot és amikor nem. Így lehetőség adódott az internetes könyvtárak nagy méretű tesztfájljain is kipróbálni a sugár módszert. Ezen tesztek alapján elmondható, hogy a saját módszerem átlagosan 21%-kal képes javítani a professzionális CPLEX solver teljesítményét nagyméretű IP és MIP problémák esetén.

Acknowledgement

Here I would like to thank all of those who have, directly or indirectly, contributed to my dissertation.

To my supervisor, Dr. Erik Bajalinov, for his encouragement, guidance and support from the initial to the final level.

I am also grateful to Prof. Tamás Vertse for valuable discussions.

I am indebted to many of my colleagues to support me.

Ezúton szeretnék köszönetet mondani mindazoknak, akik közvetve vagy közvetlenül segítették a dolgozatom elkészítését.

Témavezetőmnek, Dr. Bajalinov Eriknek az immár több éve tartó bátorításáért, útmutatásaiért és támogatásáért.

Szintén hálás vagyok Prof. Vertse Tamásnak a hasznos konzultációkért.

Köszönet a kollégáimnak, akik támogattak.

Tables

name	size (var-const)	$\sigma(A)$	MinMax_modified $\sigma(A^{\text{scaled}})$	MinMax_modified Time (sec)	Mean $\sigma(A^{\text{scaled}})$	Mean Time (sec)	Geometric $\sigma(A^{\text{scaled}})$	Geometric Time (sec)
ADLITTLE	97-56	2,758E+06	8,941E+02	0,001	6,356E+02	0,000	9,274E+02	0,001
AFIRO	32-27	4,673E+03	5,000E+02	0,000	5,000E+02	0,000	3,105E+01	0,000
AGG	163-488	3,071E+11	9,098E+02	0,059	7,248E+02	0,016	9,822E+03	0,016
BANDM	472-305	2,000E+06	9,551E+02	0,040	9,705E+02	0,016	-	-
BEACONFD	262-173	1,578E+06	9,023E+02	0,009	6,961E+02	0,000	8,764E+03	0,016
BNL1	1175-643	8,000E+06	9,363E+02	0,513	4,000E+02	0,047	8,977E+02	0,094
BNL2	3489-2324	6,000E+07	8,502E+02	10,735	9,946E+02	1,139	9,999E+03	4,727
BOEING1	384-351	2,741E+05	6,791E+02	0,049	4,875E+02	0,016	8,254E+02	0,000
BOEING2	143-166	1,000E+07	5,777E+03	0,051	8,069E+03	0,016	-	-
BRANDY	249-220	2,546E+05	9,476E+02	0,128	7,094E+02	0,018	7,938E+03	0,000
CAPRI	353-271	3,162E+07	9,823E+03	0,026	9,972E+03	0,531	-	-
CZPROB	3523-929	3,000E+06	8,380E+02	0,651	7,843E+02	0,187	-	-
D2Q06C	5167-2171	2,500E+07	8,184E+03	2,760	7,765E+03	0,062	-	-
D6CUBE	6184-415	1,152E+04	8,524E+03	0,047	4,812E+03	0,062	8,764E+04	0,088
DEGEN2	534-444	4,757E+03	7,546E+02	0,016	2,643E+02	0,000	-	-
DEGEN3	1818-1503	4,700E+03	7,907E+02	0,125	2,285E+02	0,109	-	-
DFL001	12230-6071	4,808E+08	7,750E+02	26,489	6,666E+02	3,541	9,601E+03	6,693
E226	282-223	5,716E+06	8,285E+02	0,015	9,370E+02	0,000	4,524E+04	0,005
ETAMACRO	688-400	1,200E+06	7,030E+02	0,063	3,406E+02	0,000	7,281E+02	0,010
FFFFF800	854-524	2,823E+07	9,212E+02	0,202	6,479E+02	0,047	9,870E+03	0,093
FINNIS	614-497	4,008E+08	8,385E+03	0,171	5,987E+03	0,015	-	-
FIT1D	1026-24	1,890E+05	9,784E+02	0,000	6,600E+02	0,000	8,309E+02	0,000
FIT1P	1677-627	1,890E+05	7,198E+02	0,046	4,018E+02	0,047	9,476E+02	0,047
FIT2D	10500-25	1,626E+05	9,921E+02	0,000	8,703E+02	0,016	1,171E+04	0,010
FIT2P	13525-3000	5,128E+04	9,654E+02	5,897	5,510E+02	2,278	3,402E+03	1,809
GANGES	1681-1309	7,143E+07	9,611E+02	0,608	7,008E+02	0,078	3,766E+02	0,094
GFRD-PNC	1092-616	4,318E+05	5,276E+02	0,265	6,571E+02	0,032	6,359E+02	0,046
GROW15	645-300	1,167E+06	7,230E+04	0,000	3,285E+04	0,006	-	-
GROW22	946-440	1,167E+06	7,230E+04	0,016	3,285E+04	0,015	-	-
GROW7	301-140	1,167E+06	7,230E+04	0,000	3,285E+04	0,001	-	-
ISRAEL	142-174	9,170E+08	9,186E+04	0,000	3,217E+04	0,015	-	-
KB2	41-43	1,290E+03	3,875E+02	0,000	3,608E+02	0,000	8,984E+02	0,000
LOTFI	308-153	1,114E+06	8,385E+02	0,000	1,573E+02	0,000	8,611E+02	0,005
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

name	size (var-const)	$\sigma(A)$	MinMax_modified $\sigma(A^{\text{scaled}})$	MinMax_modified Time (sec)	Mean $\sigma(A^{\text{scaled}})$	Mean Time (sec)	Geometric $\sigma(A^{\text{scaled}})$	Geometric Time (sec)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
MAROS-R7	9408-3136	1,879E+07	9,308E+03	12,137	4,191E+03	1,560	4,715E+04	1,242
MAROS	1443-846	4,009E+08	7,558E+03	0,515	4,295E+03	0,047	9,877E+04	0,040
MODSZK1	1620-688	1,873E+08	6,736E+02	0,390	3,723E+02	0,046	6,499E+02	0,036
NESM	2923-662	1,283E+07	8,954E+02	1,142	7,928E+02	0,172	-	-
PEROLD	1376-625	6,940E+08	9,915E+02	0,722	9,941E+02	0,172	5,500E+04	0,064
PILOT-JA	1988-940	2,926E+12	9,624E+04	0,748	7,247E+04	0,125	-	-
PILOT-WE	2789-722	2,135E+10	9,894E+02	2,909	9,483E+02	0,499	-	-
PILOT4	1000-410	1,067E+09	9,867E+03	0,156	8,401E+03	0,047	8,355E+04	0,045
PILOTNOV	2172-975	2,926E+12	9,436E+04	0,858	8,192E+04	0,098	-	-
RECIPELP	180-91	1,450E+05	9,965E+02	0,000	7,464E+01	0,000	2,671E+02	0,001
SC105	103-105	2,000E+03	2,000E+02	0,000	2,000E+02	0,000	2,754E+02	0,001
SC205	203-205	2,000E+03	2,000E+02	0,000	2,000E+02	0,000	2,754E+02	0,002
SC50A	48-50	1,700E+03	1,700E+02	0,000	1,700E+02	0,000	2,322E+02	0,001
SC50B	48-50	1,000E+03	3,000E+02	0,000	3,000E+02	0,000	3,000E+02	0,001
SCAGR25	500-471	3,450E+04	6,440E+02	0,025	7,980E+02	0,015	6,686E+02	0,006
SCAGR7	140-129	3,450E+04	6,440E+02	0,003	7,980E+02	0,016	6,686E+02	0,001
SCFXM1	457-330	3,600E+06	6,841E+02	0,020	8,504E+02	0,000	8,549E+03	0,015
SCFXM2	914-660	3,600E+06	9,221E+02	0,212	5,966E+02	0,031	8,155E+04	0,056
SCFXM3	1371-990	3,600E+06	9,221E+02	0,492	5,966E+02	0,078	9,655E+04	0,099
SCORPION	358-388	1,005E+05	5,525E+02	0,011	5,758E+02	0,000	5,840E+02	0,006
SCRS8	1169-490	2,190E+07	9,475E+02	0,417	5,726E+02	0,047	8,733E+03	0,124
SEBA	1028-515	2,406E+05	9,736E+02	0,156	8,076E+02	0,016	7,744E+03	0,031
SHARE1B	225-117	2,936E+07	9,600E+04	0,000	9,802E+04	0,001	-	-
SHARE2B	79-96	1,030E+04	2,443E+02	0,000	1,433E+02	0,000	2,064E+02	0,001
SHELL	1775-536	4,750E+04	9,088E+02	0,215	2,179E+02	0,031	2,543E+02	0,022
SHIP04L	2118-402	5,304E+05	8,634E+02	0,043	5,603E+02	0,062	5,636E+03	0,000
SHIP04S	1458-402	5,304E+05	8,634E+02	0,030	5,603E+02	0,047	5,559E+03	0,000
SHIP08L	4283-778	7,235E+05	8,624E+02	0,420	6,244E+02	0,219	8,818E+03	0,031
SHIP08S	2387-778	7,235E+05	8,624E+02	0,230	6,244E+02	0,125	5,127E+03	0,015
SHIP12L	5427-1151	9,006E+05	7,973E+02	1,393	8,000E+02	0,452	4,613E+04	0,062
SHIP12S	2763-1151	9,006E+05	7,973E+02	0,705	8,000E+02	0,234	4,408E+04	0,033
STAIR	467-356	8,984E+06	9,670E+03	0,047	9,645E+03	0,011	-	-
STOCFOR1	111-117	5,379E+03	5,736E+02	0,000	5,394E+02	0,000	6,403E+02	0,000
STOCFOR2	2031-2157	7,063E+05	8,655E+02	0,098	2,741E+02	0,187	8,092E+02	0,187
STOCFOR3	15695-16675	7,780E+08	9,520E+02	257,73	9,174E+02	58,906	out of runtime	
WOODW	8405-1098	1,000E+05	7,327E+02	1,316	7,071E+02	0,328	5,000E+03	0,092

Table 1: Benchmark of scaling methods on NETLIB files. Termination condition: SIGMA_LIMIT = 10^3, SIGMA_LIMIT = 10^4, SIGMA_LIMIT = 10^5.

File	Var. Sel. Strat.	Without Bound		With Bound		Improvement	
		Total	Best	Total	Best	Node	%
pr0	Min. fract. part	4891	4608	4797	4514	94	1,92%
pr1	Min. fract. part	29649	28921	29493	28765	156	0,53%
pr2	Min. fract. part	14929	12908	14929	12908	0	0,00%
pr3	Min. fract. part	4425	4424	3761	3760	664	15,01%
pr4	Min. fract. part	11509	9059	9263	6813	2246	19,52%
pr0	Max. fract. part	1007	805	1007	805	0	0,00%
pr1	Max. fract. part	4339	3921	4339	3921	0	0,00%
pr2	Max. fract. part	7255	6125	7255	6125	0	0,00%
pr3	Max. fract. part	1257	1114	297	154	960	76,37%
pr4	Max. fract. part	5295	5025	4415	4145	880	16,62%
pr0	Most close to 0.5	4181	4033	4181	4033	0	0,00%
pr1	Most close to 0.5	7237	7034	7237	7034	0	0,00%
pr2	Most close to 0.5	7409	6446	7255	6125	154	2,08%
pr3	Most close to 0.5	4285	4284	2785	2784	1500	35,01%
pr4	Most close to 0.5	5215	4832	5215	4832	0	0,00%
						Average:	11,14%

Table 2: Problems in size 50x50, node selection strategy: Left to right

File	Var. Sel. Strat.	Without Bound		With Bound		Improvement	
		Total	Best	Total	Best	Node	%
pr0	Min. fract. part	4891	4608	4797	4514	94	1,92%
pr1	Min. fract. part	29649	28921	29493	28765	156	0,53%
pr2	Min. fract. part	14929	12908	14929	12908	0	0,00%
pr3	Min. fract. part	4425	4424	3761	3760	664	15,01%
pr4	Min. fract. part	11509	9059	9263	6813	2246	19,52%
pr0	Max. fract. part	1007	805	1007	805	0	0,00%
pr1	Max. fract. part	4339	3921	4339	3921	0	0,00%
pr2	Max. fract. part	7255	6125	7255	6125	0	0,00%
pr3	Max. fract. part	1257	1114	297	154	960	76,37%
pr4	Max. fract. part	5295	5025	4415	4145	880	16,62%
pr0	Most close to 0.5	4181	4033	4181	4033	0	0,00%
pr1	Most close to 0.5	7237	7034	7237	7034	0	0,00%
pr2	Most close to 0.5	7409	6446	7023	6060	386	5,21%
pr3	Most close to 0.5	4285	4284	2785	2784	1500	35,01%
pr4	Most close to 0.5	3321	1486	3321	1486	0	0,00%
						Average:	11,35%

Table 3: Problems in size 50x50, node selection strategy: Right to left

File	Var. Sel. Strat.	Without Bound		With Bound		Improvement	
		Total	Best	Total	Best	Node	%
pr0	Min. fract. part	2019	2011	1257	1249	762	37,74%
pr1	Min. fract. part	3625	3588	625	588	3000	82,76%
pr2	Min. fract. part	2047	1563	1443	959	604	29,51%
pr3	Min. fract. part	1999	1861	1589	1451	410	20,51%
pr4	Min. fract. part	1785	1784	1589	1588	196	10,98%
pr0	Max. fract. part	299	104	299	104	0	0,00%
pr1	Max. fract. part	611	601	255	245	356	58,27%
pr2	Max. fract. part	1861	1568	1369	1076	492	26,44%
pr3	Max. fract. part	355	199	261	105	94	26,48%
pr4	Max. fract. part	983	942	813	772	170	17,29%
pr0	Most close to 0.5	905	889	809	793	96	10,61%
pr1	Most close to 0.5	915	913	563	561	352	38,47%
pr2	Most close to 0.5	1121	888	1121	888	0	0,00%
pr3	Most close to 0.5	887	801	887	801	0	0,00%
pr4	Most close to 0.5	1603	1559	1393	1349	210	13,10%
						Average:	24,81%

Table 4: Problems in size 25x25, node selection strategy: Left to right

Bibliography

[1] Andersen, E. D., Andersen K. D., *"Presolving in linear programming"*, Mathematical Programming, Vol. 71, pp. 221-245, 1995.

[2] Atamtürk, A., *On Capacitated Network Design Cut-Set Polyhedra*, Mathematical Programming, Vol. 92, pp. 425-437, 2002.

[3] Atamtürk, A., Rajan D., *On Splittable and Unsplittable Capacitated Network Design Arc-Set Polyhedra*, Mathematical Programming, Vol. 92, pp. 315-333, 2002.

[4] Bajalinov, E. B., *"Linear-Fractional Programming: Theory, Methods, Applications and Software"*, Kluwer Academic Publishers, 2003.

[5] Bajalinov, E., Rácz, A., *"Scaling problems in linear-fractional programming"*, Acta Mathematica Academiae Paedagogicae Nyiregyhaziensis, Vol. 25, no. 2, pp. 283-301, 2009.

[6] Bajalinov, E., *"Taxes, subsidies and unemployment: a unified optimization approach"*, Croatian Operational Research Review (CRORR), Vol. 1, 2010.

[7] Bajalinov E., *"On an approach to the modeling of conflicting economic interests"*, European Journal of Operations Research, 116, 477-486, 1999.

[8] Balas, E., *"An additive algorithm for solving linear programs with zero-one variables"*, Operations Research, vol. 13, pp. 517-46, 1965.

[9] Berliner, H., *"The B tree search algorithm: A best-first proof procedure"*, Artificial Intell., vol. 12, no. 1, pp. 23-40, 1979.

[10] Borchers, B., Mitchell, J.E., *"An improved branch and bound algorithm for mixed integer nonlinear programs"*, Computers and Operations Research, vol. 21, issue 4, pp. 359-367, 1994.

[11] Charnes, A., Cooper, W. W., *"Programming with Linear Fractional Functionals"*, Naval Research Logistics Quarterly 9, pp. 181-186, 1962.

[12] Dantzig, G. B., *"Linear Programming and Extensions"*, Princeton University Press, 1963.

[13] Dantzig, G. B., *"Linear programming"*, Operations Research 50: 42-47, INFORMS, 2002.

[14] Darai, J., Rácz, A., Salamon, P., Lovas, R. G., *Antibound poles in cutoff Woods-Saxon and in Salamon-Vertse potentials*, *submitted*

[15] Fulkerson, D., Wolfe, P., *"An algorithm for scaling matrices"*, SIAM Review, 4, pp. 142–146, 1962.

[16] Gendron, B., Crainic, T.G., *"Parallel Branch-and-Bound Algorithms: Survey and Synthesis."*, Opererations Research, vol. 42 issue 6, pp. 1042-1066, 1994.

[17] Gupta, O. K., Ravindran, V., *"Branch and Bound Experiments in Convex Nonlinear Integer Programming"*, Management Science, vol. 31, no. 12, 1533-1546, 1985.

[18] Hawkins, D. M., *"Branch-and-Bound method"*, Encyclopedia of Statistical Sciences, John Wiley and Sons, 2006.

[19] Ibaraki, T., *"Computational efficiency of approximate branch-and-bound algorithms"*, Math. Oper. Res., vol. 1, no. 3, pp. 287-298, 1976.

[20] Ibaraki, T., *"Theoretical comparisons of search strategies in branch-and-bound algorithms"*, Int. J. Computer and Information Sciences, vol. 5, no. 4, pp. 315-343, 1976.

[21] Ibaraki, T., *"The Power of Dominance Relations in Branch-and-Bound Algorithms"*, Journal of the ACM (JACM), vol. 24, issue 2, pp. 264-279, 1977.

[22] Id Betan, R., Rácz, A., Vertse, T., *"Calculation of the Isobaric Analogue Resonance Using Shell Model in the Complex Energy Plane"*, International Journal of Theoretical Physics, Vol. 50, no. 7, pp. 2222-2226, 2011.

[23] Karmarkar, N., *"A New Polynomial Time Algorithm for Linear Programming"*, Combinatorica, vol. 4, no. 4, pp. 373-395, 1984.

[24] Khachaturov, V.R., Mirzoyan, N.A., *"Solving problems of integer programming with ray-method."*, Notes on applied mathematics, Computer Center of Soviet Academy of Science, 1987.

[25] Khachaturov, V.R., *"Combinatorial methods and algorithms for solving large-scale discrete optimization problems"*, Moscow, Nauka, 2000.

[26] Khachian, L. G., *"A polynomial algorithm in linear programming"*, Soviet Mathematics Doklady, vol. 20, pp. 1093-1096, 1979.

[27] Kirby, M. W., *"Operational research in war and peace - The british Experience from the 1930s to 1970"*, Imperial College Press and the Operation Research Society, 2003.

[28] Koberstein, A., *"The Dual Simplex Method, Techniques for a fast and stable implementation"*, Dissertation, Faculty of Business Administration and Economics, University of Paderborn, 2005.

[29] Kuhn, H. W., *"The Hungarian method for the assignment problem"*, Naval Research Logistics Quarterly, vol. 2, pp. 83-87, 1955.

[30] Kumar, V., Kanal, L. N., *"A general branch and bound formulation for understanding and synthesizing and/or tree search procedures"*, Artificial Intell., vol. 21, no. 1-2, pp. 179-198, 1983.

[31] Land, A. H., Doig, A. G., *"An Automatic Method for Solving Discrete Programming Problems"*, Econometrica, vol. 28, pp.497-520, 1960.

[32] Lawler, E. L., Wood, D. E., *"Branch-And-Bound Methods: A Survey"*, Operations Research, vol. 14, no. 4, pp. 699-719, 1966.

[33] Linderoth, T. Savelsbergh, M. W. P., *"A computational study of branch and bound search strategies for mixed integer programming"*, INFORMS J. Computing, vol. 11, no. 2, pp. 173-187, 1999.

[34] LINDO Systems Inc. *"LINGO the modelling language and optimizer"*, Chicago, 2006.

[35] Martos, B., *"Hyperbolic programming"*, Naval Research Logistics Quarterly, vol. 11., pp. 135-155, 1964.

[36] Maros, I., *"Computational Techniques of the Simplex Method"*, Kluwer Academic Publishers, 2003.

[37] Mészáros, Cs., Suhl, U. H., *"Advanced preprocessing techniques for linear and quadratic programming"*, OR Spectrum 25: 575-595, Springer Berlin, 2003.

[38] Mitten, L., *"Branch and bound methods: General formulation and properties"*, Operation Research, vol. 18, pp. 24-34, 1970.

[39] Nemhauser, G. L., Wolsey, L. A., *"Integer and combinatorial optimization"*, John Wiley & Sons, 1999.

[40] Pardalos, P. M., Resende M.G.C., *"Handbook of Applied Optimization"*, Oxford University Press, 2002.

[41] Rácz, A., *"Determining initial bound by "ray-method" in branch and bound procedure"*, Acta Cybernetica, Vol. 19, no. 1, pp. 135-146, 2009

[42] Rácz, A., Salamon, P., Vertse, T., *"Trajectories of S-matrix poles in a new finite-range potential"*, Physical Review C 84 (3), art. no. 037602, 2011.

[43] Salamon, P., Vertse, T., *New simple form a phenomenological nuclear potential*, Physical Review C 77, art. no. 037302, 2008.

[44] Sherali, H. D., *"On a fractional minimal cost flow problem on networks"*, Optimization Letters, pp. 1-5, 2011.

[45] Shu-Cherng Fang, Sarat Puthenpura *"Linear optimization and extensions: theory and algorithms"*, Prentice Hall, 1993.

[46] Sivri, M. , Emiroglu, I. , Guler, C. , Tasci, F., *"A solution proposal to the transportation problem with the linear fractional objective function"*, 4th International Conference on Modeling, Simulation and Applied Optimization, ICMSAO 2011, Article number: 5775530

[47] Smith, D.R., *"Random trees and the analysis of branch and bound procedures"*, Journal of the Association for computing machinery, vol.31, no.1, pp.163-188, 1984.

[48] Yu, C.-F., Wah, B.W., *"Efficient Branch-and-Bound Algorithms on a Two-Level Memory System"*, IEEE Transactions on Software Engineering, vol. 14, no. 9, pp. 1342-1356, 1988.

[49] Yu, C.-F., Wah, B.W., *"Stochastic modeling of branch-and-bound algorithms with best-first search"*, IEEE Transactions on Software Engineering, vol. SE-11, no. 9, pp. 922-934, 1985.

[50] *"IBM CPLEX documentation"*
http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

List of publications

[1] Rácz, A., Salamon, P., Vertse, T., *"Trajectories of S-matrix poles in a new finite-range potential"*, Physical Review C 84 (3), art. no. 037602, 2011 (IF: 3.416) *(Scopus, Web of knowledge)*

[2] Id Betan, R., Rácz, A., Vertse, T., *"Calculation of the Isobaric Analogue Resonance Using Shell Model in the Complex Energy Plane"*, International Journal of Theoretical Physics, Vol. 50, no. 7, pp. 2222-2226, 2011 (IF: 0.670) *(Scopus, Web of knowledge)*

[3] Bajalinov, E., Rácz, A., *"Scaling problems in linear-fractional programming"*, Acta Mathematica Academiae Paedagogicae Nyiregyhaziensis, Vol. 25, no. 2, pp. 283-301, 2009 *(Scopus, MathSciNet)*

[4] Rácz, A., *"Determining initial bound by "ray-method" in branch and bound procedure"*, Acta Cybernetica, Vol. 19, no. 1, pp. 135-146, 2009 *(Scopus, MathSciNet)*

[5] Bajalinov, E., Rácz, A., *"Scaling problems in linear-fractional programming"*, Proceedings of the International Conference on Information Technology Interfaces, ITI , art. no. 1708531, pp. 495-499, 2006 *(Scopus, Web of knowledge, IEEE Xplore)*

Közlésre benyújtott cikkek:

[1] J. Darai, A. Rácz, P. Salamon, and R. G. Lovas, *Antibound poles in cutoff Woods-Saxon and in Salamon-Vertse potentials*, Physical Review C

Oktatási segédanyag:

[1] Bajalinov Erik, Rácz Anett, *Operációkutatás I.*, TÁMOP-4.1.2-08/1/A-2009-0046

[2] Bajalinov Erik, Rácz Anett, *Operációkutatás II.*, TÁMOP-4.1.2-08/1/A-2009-0046

Conference Talks

[1] Anett Bekéné Rácz *"Stabilitás- és teljesítményjavító eljárások optimalizációs feladatokban"*, XXIX. Magyar Operációkutatási Konferencia, Balatonőszöd, 2011. 09. 28.

[2] Zoltán Megyesi, Anett Bekéné Rácz *"Optimalizációs feladatok megoldása GPU segítségével"*, XXIX. Magyar Operációkutatási Konferencia, Balatonőszöd, 2011. 09. 30.

[3] Veronika Mátyus, Zoltán Megyesi, Anett Bekéné Rácz *"MPS processing for optimization problems"*, (conference poster) CSMA 2011, Debrecen

[4] Anett Rácz *"Methods for increasing computational efficiency and stability in solving optimization problems"*, 7th Conference on Applied Mathematics and Scientific Computing - ApplMath11, Trogir, 2011.06.17.

[5] Anett Rácz *"Preprocessing in Linear-Fractional Programming"*, 24th European Conference on Operational Research - EURO XXIV 2010, Lisbon, 2010.07.14.

[6] Anett Rácz *"Preprocessing in Linear-Fractional Programming"*, Conference of PhD Students in Computer Science 2010 - CSCS, Szeged, 2010.07.01.

[7] Anett Rácz *"Adapting LP preprocessing techniques to LFP problems"*, 8th International Conference on Applied Informatics - ICAI 2010, Eger, 2010.01.29.

[8] Erik Bajalinov, Anett Rácz *"On the Ray-based procedure for determining initial bound in branch and bound method"*, Veszprém Optimization Conference: Advanced Algorithms - VOCAL, 2008.12.16.

[9] Anett Rácz *"Determining Initial Bound by "Ray-method" in Branch and Bound Procedure"*, The Sixth Conference of PhD Students in Computer Science - CSCS, Szeged, 2008.07.03.

[10] Erik Bajalinov, Anett Rácz *"A sugár módszer lineáris és hiperbolikus programozásban"*, 9. Gyires Béla Informatikai Nap, Debreceni Egyetem, 2007.11.23.

[11] Anett Rácz *"Skálázási eljárások értékelése"*, XXVIII. Országos Tudományos Diákköri Konferencia - OTDK, Miskolc, 2007. 04. 26.

[12] Anett Rácz *"Scaling Problems in Linear-Fractional Programing"*, 28th International Conference on Information Technology Interfaces - ITI, Cavtat/Dubrovnik, 2006.06.20.

Computational methods for optimization problems

Optimalizációs feladatok számítógépes feldolgozása

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében
az informatika tudományágban

Írta: Bekéné Rácz Anett okleveles programtervező matematikus

Készült a Debreceni Egyetem Informatikai Tudományok doktori iskolája
(Az információ technológia és a sztochasztikus rendszerek elméleti alapjai és
alkalmazásai programja) keretében

Témavezető: Dr. Bajalinov Erik

A doktori szigorlati bizottság:
elnök: Dr. Sztrik János
tagok: Dr. Pokorádi László
Dr. Csendes Tibor

A doktori szigorlat időpontja: 2010. December 10.

Az értekezés bírálói:
Dr. .....
Dr. .....

A bíráló bizottság:
elnök: Dr. .....
tagok: Dr. .....
Dr. .....
Dr. .....
Dr. .....

Az értekezés védésének időpontja: 2012. .....