A Decision Support System for Solving Linear Programming Problems

Nikolaos Ploskas, Nikolaos Samaras
Department of Applied Informatics, School of Information Sciences, University of Macedonia
156 Egnatia Str., 54006 Thessaloniki, Greece
ploskas@uom.gr, samaras@uom.gr

Jason Papathanasiou
Department of Business Administration, School of Business Administration, University of Macedonia
156 Egnatia Str., 54006 Thessaloniki, Greece
jasonp@uom.gr

ABSTRACT

Linear programming algorithms have been widely used in Decision Support Systems. These systems have incorporated linear programming algorithms for the solution of the given problems. Yet, the special structure of each linear problem may take advantage of different linear programming algorithms or different techniques used in these algorithms. In this paper, we propose a web-based DSS that assists decision makers in the solution of linear programming problems with a variety of linear programming algorithms and techniques. Two linear programming algorithms have been included in the DSS: (i) revised simplex algorithm and (ii) exterior primal simplex algorithm. Furthermore, ten scaling techniques, five basis update methods and eight pivoting rules have been incorporated in the DSS. All linear programming algorithms and methods have been implemented using MATLAB and converted to Java classes using MATLAB Builder JA, while the web interface of the DSS has been designed using Java Server Pages.

Keywords: Decision Making, Web-based Decision Support Systems, Linear Programming, Revised Simplex Algorithm, Exterior Primal Simplex Algorithm.

1. INTRODUCTION

Web-based Decision Support Systems (DSS) are computerized information systems that provide decision support tools to managers or business analysts using only a thin-client Web Browser (Power & Kaparthi, 2002). Web-based DSS can assist a decision maker to: (i) retrieve, analyze and display data from large databases, (ii) provide access to a model, and (iii) establish communication and decision making in distributed teams (Power, 2000). In general, all types of DSS, communication-driven, knowledge-driven and document-driven (Bhargava et al., 2007), can be implemented as a web-based DSS (Power, 2000).

Linear programming algorithms have been widely used in DSS for supplier selection (Ghodsypour & O'Brien, 1998), forest management planning systems (Lappi et al., 1996), assignment of parking spaces (Venkataramanan & Bornstein, 1991), schedule of student attendants (Lauer et al., 1994), portfolio robustness evaluation (Lourenço et al., 2012), optimality in open air reservoir strategies (Van Vuuren & Grundlingh, 2002), energy planning (Mavrotas, 2000) and water resource management (Faye et al., 1998) among

others. However, the special structure of each linear problem should be taken into consideration in order to take advantage of different linear programming algorithms and methods.

This paper presents a web-based DSS that provides decision support tools to decision makers that want to solve their linear programming problems. The paper builds on the work of Ploskas et al. (2013). Ploskas et al. (2013) have implemented a web-based DSS that assists decision makers in the selection of the linear programming algorithm and basis update method for solving their linear programming problems. In this paper, we do not only take into consideration the basis update step, but go further to explore all different steps of the linear programming algorithms. The main difference from our previous paper (Ploskas et al., 2013) is that here we include ten scaling techniques, five basis update methods and eight pivoting rules; the user can select any combination of these methods to be included in the execution of the linear programming algorithm or let the DSS select the best combination for the selected linear programming problem.

Two linear programming algorithms are incorporated in the DSS: (i) Revised Simplex Algorithm (Dantzig, 1953) and (ii) Exterior Primal Simplex Algorithm (Paparrizos et al., 2003). The DSS also includes a variety of different methods for the different steps of these algorithms. More specifically, ten scaling techniques, five basis update methods and eight pivoting rules have been implemented in the DSS. The decision maker can either select the algorithm and the appropriate methods to solve a linear programming problem or perform a thorough computational study with all combinations of algorithms and methods in order to gain an insight on its linear programming problem.

There are already linear programming solvers in the market that efficiently solve linear programming problems (LPs), but either they do not include so many scaling techniques, basis update methods and pivoting rules, either they do not allow the user to choose some of them. To the best of our knowledge, this is the first paper that implements a DSS for solving linear programming problems that include all these different methods for scaling, basis update and pivoting, and lets the user select the different combinations of the methods to be included in the execution of the linear programming algorithm.

The rest of this paper is organized as follows. Section 2 presents the background of our work. In Section 3, ten widely-used scaling techniques that incorporated in the DSS are presented. Section 4 includes the presentation of the five basis update methods implemented in the DSS, while in Section 5 eight well-known pivoting rules that incorporated in the DSS are presented. Section 6 includes the analysis and design of the DSS, while in Section 7 the DSS is presented. Finally, the conclusions of this paper are outlined in Section 8.

2. BACKGROUND

Linear programming

Linear programming is the process of minimizing or maximizing a linear objective function $z = \sum_{i=1}^{n} c_i x_i$ to a number of linear equality and inequality constraints. Several methods are available for solving linear programming problems, among which the simplex algorithm is the most widely used. We assume that the problem is in its general form. Formulating the linear problem, we can describe it as shown in equation 1:

$$\begin{aligned} \min \quad & c^T x \\ \textit{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{1}$$

where $A \in R^{mxn}$, $(c, x) \in R^n$, $b \in R^m$, and T denotes transposition. We assume that A has full rank (rank(A)=m, m < n). The simplex algorithm searches for an optimal solution by moving from one feasible solution to another, along the edges of the feasible set. The dual problem associated with the linear problem in equation 1 is shown in equation 2:

$$\begin{aligned} \min \quad & b^T w \\ \textit{subject to} \quad & A^T w + s = c \\ & s \geq 0 \end{aligned} \tag{2}$$

where $w \in R^m$ and $s \in R^n$. Using a partition (B, N) equation 1 can be written as shown in equation 3:

$$\begin{aligned} \min \quad & c_B^T x_B + c_N^T x_N \\ \textit{subject to} \quad & A_B x_B + A_N x_N = b \\ & x_B, x_N \geq 0 \end{aligned} \tag{3}$$

In the above equation, B is an mxm non-singular sub-matrix of A, called basic matrix or basis. The columns of A which belong to subset B are called basic and those which belong to N are called non basic. The solution of the linear problem $x_B = B^{-1}b, x_N = 0$ is called a basic solution. A solution $x = (x_B, x_N)$ is feasible if x > 0. Otherwise the solution is infeasible. The solution of the linear problem in equation (2) is computed by the relation $s = c - A^T w$, where $w = (c_B)^T B^{-1}$ are the simplex multipliers and s are the dual slack variables. The basis B is dual feasible if $s \geq 0$.

Two linear programming algorithms have been implemented using MATLAB and incorporated in the DSS: (i) Revised Simplex Algorithm proposed by Dantzig (1953) and (ii) Exterior Primal Simplex Algorithm (EPSA) proposed by Paparrizos et al. (2003).

Revised simplex algorithm

A formal description of the revised simplex algorithm (Dantzig, 1953) is given below.

Table 1: Revised Simplex Algorithm

Step 0. (*Initialization*).

Start with a feasible partition (B, N). Compute A_B^{-1} and vectors x_B, w and s_N.

Step 1. (*Test of optimality*).

if $s_N \geq 0$ then STOP. The linear problem (equation 3) is optimal.

Step 2. (*Choice of the entering/leaving variable*).

Choose the index l of the entering variable using a pivoting rule. Variable x_l enters the basis.

Compute the pivot column $h_l = A_B^{-1} A_l$.

if $h_l \leq 0$ then STOP. The linear problem (equation 3) is unbounded.

else

Choose the leaving variable $x_{B[r]} = x_k$ using the following equation:

$$x_{B[r]} = \frac{x_{B[r]}}{h_{il}} = \min\left\{\frac{x_{B[i]}}{h_{il}} : h_{ll} < 0\right\}$$

Step 3. (*Pivoting*).

Swap indices k and l. Update the new basis inverse $\overline{A_B}^{-1}$. Go to Step 1.

Exterior primal simplex algorithm

The algorithm starts with a primal feasible basic partition (B, N). Then, the following sets of indexes are computed:

$$P = \{ j \in N : s_j < 0 \} \quad (4)$$

$$Q = \{ j \in N : s_j \geq 0 \} \quad (5)$$

If P = ∅ then the current basis B and the corresponding solution $x^T = (x_B, x_N)$ is optimal for the primal problem. EPSA firstly defines the leaving and afterwards the entering variable. The leaving variable $x_{B[r]} = x_k$ is computed using equation 6:

$$a = \frac{x_{B[r]}}{-d_{B[r]}} = \min\left\{\frac{x_{B[i]}}{-d_{B[i]}} : d_{B[i]} < 0\right\} \quad (6)$$

where d is an improving direction. This direction is constructed in such way that the ray {x + td : t > 0} crosses the feasible region of equation 1. The notation d_B denotes those components from d which correspond to the basic variables. The d_B is computed as shown in equation 7:

$$\mathbf{d_B} = -\sum_{j \in P} \mathbf{h_j} \quad (7)$$

where $h_j = B^{-1} A_{.j}$. If $d_B \geq 0$, then the problem is unbounded.

In order to compute the entering variable x_l, the following ratios must first be calculated using equations 8 and 9:

$$\theta_1 = -\frac{-s_Q}{H_{rQ}} = \min\left\{\frac{-s_j}{H_{rj}} : H_{rj} > 0 \ \wedge \ j \in P\right\} \quad (8)$$

and

$$\theta_2 = -\frac{-s_Q}{H_{rQ}} = \min\left\{\frac{-s_j}{H_{rj}} : H_{rj} < 0 \ \wedge \ j \in Q\right\} \quad (9)$$

If $\theta_1 \leq \theta_2$ then l = p, otherwise (e.g., $\theta_1 > \theta_2$) l = q. The non-basic variable x_l enters the basis. A formal description of the EPSA is given below (Paparrizos et al., 2003).

Table 2: Exterior Primal Simplex Algorithm

Step 0. (*Initialization*).

Start with a feasible partition (B, N). Compute B^{-1} and vectors x_B, w and s_N. Find the sets of indices P and Q using relations 4 and 5. Define an arbitrary vector $\lambda = (\lambda_1, \lambda_2, ..., \lambda_{|P|}) > 0$ and compute s_0 as follows:

$$s_0 = \sum_{j \in P} \lambda_j s_j \quad (10)$$

and the direction d_B from equation 7.

Step 1. (*Termination test*).

(*Optimilaty test*). If P = ∅, STOP. The problem is optimal.

(*Leaving variable selection*). If $d_B \geq 0$, STOP. If $s_0 = 0$ the problem is optimal. If $s_0 < 0$ the problem is unbounded. Otherwise choose the leaving variable $x_{B[r]} = x_k$ using equation 6.

Step 2. (*Entering variable selection*).

Compute the row vectors:

$$H_{rP} = (B^{-1})_{r.}A_P \text{ and } H_{rQ} = (B^{-1})_{r.}A_Q \quad (11)$$

Compute the ratios θ_1 and θ_2 using relations 8 and 9. Determine the indices t_1 and t_2 such that $P[t_1] = p$ and $Q[t_2] = q$. If $\theta_1 \leq \theta_2$, set $l = p$, otherwise ($\theta_1 > \theta_2$) $l = q$. The non-basic variable x_l enters the basis.

Step 3. (*Pivoting*)

Set $B[r] = l$. If $\theta_1 \leq \theta_2$, set $P \leftarrow P\backslash\{l\}$ and $Q \leftarrow Q \cup \{k\}$. Otherwise, set $Q[t_2] = k$. Using the new partition (B, N) where N = (P, Q), update the matrix B^{-1} and the vectors x_B, w and s_N. Also update $\overline{d}_B$ as follows:

$$\overline{d_B} = E^{-1}d_B \quad (12)$$

If $l \in P$ set $d_{B[r]} \leftarrow d_{B[r]} + \lambda_l$. Go to step 1.

3. SCALING TECHNIQUES

Preconditioning techniques can be applied to linear programming problems prior to the application of an optimization solver in order to improve their computational properties. Scaling is the most well-known and widely used preconditioning technique. Scaling is an operation in which the rows and columns of a matrix are multiplied by positive scalars; this operation leads to nonzero numerical values of similar magnitude. Scaling is used for the following reasons for (Tomlin, 1975): (i) the production of a compact representation of the bounds of the variables, (ii) the reduction of the number of the iterations, (iii) the simplification of the setup of the tolerances, (iv) the reduction of the condition number of the constraint matrix, and (v) the improvement of the numerical behavior of the linear programming algorithms.

In the proposed DSS, we have implemented ten widely used scaling techniques: (i) arithmetic mean, (ii) de Buchet for the case p = 1, (iii) de Buchet for the case p = 2, (iv) entropy (Larsson, 1993), (v) equilibration, (vi) geometric mean, (vii) IBM MPSX (Benichou et al., 1977), (viii) Lp-norm for the case p = 1, (ix) Lp-norm for the case p = 2, and (x) L_p-norm for the case p = ∞ and de Buchet for the case p = ∞.

Prior to the presentation of the aforementioned scaling techniques, some mathematical preliminaries and notations should be introduced. Let A be an m x n matrix. Let r_i be the row scaling factor for row i and s_j be the column scaling factor for column j. Let $N_i = \left\{ j \mid A_{ij} \neq 0 \right\}$, where i = 1, ...,m, and $M_j = \left\{ i \mid A_{ij} \neq 0 \right\}$, where j = 1, ..., n. Let n_i and m_j be the cardinality numbers of the sets N_i and M_j, respectively. The scaled matrix is expressed as X = RAS, where $R = diag(r_1...r_m)$ and $S = diag(s_1...s_n)$.

Arithmetic mean

Arithmetic mean scaling method aims to reduce the variance between the nonzero elements of the coefficient matrix A. Each row and column is divided by the arithmetic

mean of the elements in that specific row and column, respectively. The row and column scaling factors are presented in equation 13 and equation 14, respectively:

$$r_i = \left(\frac{n_i}{\sum_{j \in N_i} A_{ij}} \right) \tag{13}$$

$$s_j = \left(\frac{m_j}{\sum_{i \in M_j} A_{ij}} \right) \tag{14}$$

de Buchet

The de Buchet scaling model is formulated as shown in equation 15:

$$\min_{(r,s>0)} \left(\sum_{(i,j) \in \overline{Z}} \left\{ A_{ij} r_i s_j + 1/\left(A_{ij} r_i s_j \right) \right\}^p \right)^{1/p} \tag{15}$$

where p is a positive integer and $\overline{Z}$ is the number of the nonzero elements of matrix A. For the case p = 1, equation 15 is formulated as shown in equation 16.

$$\min_{(r,s>0)} \sum_{(i,j) \in \overline{Z}} A_{ij} r_i s_j + 1/\left(A_{ij} r_i s_j \right) \tag{16}$$

The row and the column scaling factors for the case p = 1 are shown in equations 17 and 18, respectively:

$$r_i = \left\{ \left(\sum_{j \in N_i} 1/\left| A_{ij} \right| \right) \left(\sum_{j \in N_i} \left| A_{ij} \right| \right) \right\}^{1/2} \tag{17}$$

$$s_j = \left\{ \left(\sum_{i \in M_j} 1/\left| A_{ij} \right| \right) \left(\sum_{i \in M_j} \left| A_{ij} \right| \right) \right\}^{1/2} \tag{18}$$

For the case p = 2, equation 15 is formulated as shown in equation 19.

$$\min_{(r,s>0)} \left(\sum_{(i,j) \in \overline{Z}} \left\{ A_{ij} r_i s_j + 1/\left(A_{ij} r_i s_j \right) \right\}^2 \right)^{1/2} \tag{19}$$

The row and the column scaling factors for the case p = 2 are shown in equations 20 and 21, respectively:

$$r_i = \left\{ \left(\sum_{j \in N_i} \left(1/\left| A_{ij} \right| \right)^2 \right) \left(\sum_{j \in N_i} \left(\left| A_{ij} \right| \right)^2 \right) \right\}^{1/4} \tag{20}$$

$$s_j = \left\{ \left(\sum_{i \in M_j} \left(1/\left| A_{ij} \right| \right)^2 \right) \left(\sum_{i \in M_j} \left(\left| A_{ij} \right| \right)^2 \right) \right\}^{1/4} \tag{21}$$

Finally, for the case p = ∞, equation 15 is formulated as shown in equation 22.

$$\min_{(r,s>0)} \max_{(i,j) \in \overline{Z}} \left| \log\left(A_{ij} r_i s_j \right) \right| \tag{22}$$

The row and the column scaling factors for the case p = ∞ are shown in equations 23 and 24, respectively:

$$r_i = 1/\left\{\left(\max_{j\in N_i}\left|A_{ij}\right|\right)\left(\min_{j\in N_i}\left|A_{ij}\right|\right)\right\}^{1/2} \tag{23}$$

$$s_j = 1/\left\{\left(\max_{i\in M_j}\left|A_{ij}\right|\right)\left(\min_{i\in M_j}\left|A_{ij}\right|\right)\right\}^{1/2} \tag{24}$$

Entropy

The entropy model was first presented by Larsson (1993). This technique solves the model presented in equation 25, in order to identify a scaling X with all $x_{ij} \neq 0$ of magnitude one:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in\overline{Z}} X_{ij}\left(\log\left(X_{ij}/A_{ij}\right)-1\right) \\ \textit{subject to} \quad & \sum_{j\in N_i} X_{ij} = n_i \qquad i = 1,\ldots,m \\ & \sum_{i\in M_j} X_{ij} = m_j \qquad \mathrm{j} = 1,\ldots,n \\ & \mathrm{X}_{ij} \geq 0 \qquad\qquad \forall(i,j)\in\overline{Z} \end{aligned} \tag{25}$$

The row and column scaling factors are presented in equations 26 and 27, respectively:

$$r_i = n_i / \sum_{j\in N_i}\left|A_{ij}\right| \tag{26}$$

$$s_j = m_j / \sum_{i\in M_j}\left|A_{ij}\right| \tag{27}$$

Equilibration

In this scaling technique, for each row of the coefficient matrix A the largest element in absolute value is found. Then, the specified row of matrix A and the corresponding element of vector b are multiplied by the inverse of the largest element. Then, for each column of the coefficient matrix A that does not include 1 as the largest element in absolute value, the largest element in absolute value is found, and the specified column of matrix A and the corresponding element of vector c is multiplied by the inverse of the largest element. Consequently, all the elements of matrix A will have values between -1 and 1.

Geometric mean

Like the arithmetic mean scaling method, geometric mean also aims to reduce the variance between the nonzero elements of the coefficient matrix A. Each row and column is divided by the product of the square root of the maximum and minimum element in that row and column, respectively. The row and column scaling factors are presented in equation 28 and equation 29, respectively:

$$r_i = \left(\max_{j\in N_i} A_{ij} \min_{j\in N_i} A_{ij}\right)^{-1/2} \tag{28}$$

$$s_j = \left(\max_{i\in M_j} A_{ij} \min_{i\in M_j} A_{ij}\right)^{-1/2} \tag{29}$$

IBM MPSX

The method was proposed by Benichou et al. (1977) and was later adopted by IBM, which used this method in IBMs MPSX linear optimization solver. This method combines geometric mean and equilibration scaling techniques. Initially, geometric mean is performed four times or until the relation 30 is true.

$$\frac{1}{\left|\overline{Z}\right|}\left(\left(\sum_{(i,j)\in\overline{Z}}\left(A_{ij}\right)^2\right)-\left(\sum_{(i,j)\in\overline{Z}}\left(A_{ij}\right)^2\right)^2/\left|\overline{Z}\right|\right)<\varepsilon \tag{30}$$

where $\left|\overline{Z}\right|$ is the cardinality number of nonzero element of matrix A and ε is a tolerance, which is often set below ten. Then, the equilibration scaling technique is applied.

L_p-norm

The L_p-norm scaling model is formulated as shown in equation 31:

$$\min_{(r,s>0)}\left(\sum_{(i,j)\in\overline{Z}}\left|\log\left(A_{ij}r_is_j\right)\right|^p\right)^{1/p} \tag{31}$$

where p is a positive integer and $\left|\overline{Z}\right|$ is the cardinality number of nonzero element of matrix A.

For the case p = 1, equation 31 is formulated as shown in equation 32.

$$\min_{(r,s>0)}\sum_{(i,j)\in\overline{Z}}\left|\log\left(A_{ij}r_is_j\right)\right| \tag{32}$$

The row and the column scaling factors for the case p = 1 are shown in equations 33 and 34, respectively:

$$r_i=1/\,median\left\{A_{ij}\mid j\in N_i\right\} \tag{33}$$

$$s_j=1/\,median\left\{A_{ij}\mid i\in M_j\right\} \tag{34}$$

For the case p = 2, equation 32 is formulated as shown in equation 35.

$$\min_{(r,s>0)}\left(\sum_{(i,j)\in\overline{Z}}\left|\log\left(A_{ij}r_is_j\right)\right|^2\right)^{1/2} \tag{35}$$

The row and the column scaling factors for the case p = 2 are shown in equations 36 and 37, respectively:

$$r_i=1/\prod_{j\in N_i}\left(A_{ij}\right)^{1/n_i} \tag{36}$$

$$r_i=1/\prod_{i\in M_j}\left(A_{ij}\right)^{1/m_j} \tag{37}$$

Finally, for the case p = ∞, the model and the row and scaling factors are equivalent to the de Buchet for the case p = ∞.

4. BASIS UPDATE METHODS

The computation of the basis inverse is the most time-consuming step in linear programming algorithms and if these methods are not properly designed and implemented the basis inverse step can dictate the total execution time of the algorithm. However, this inverse does not have to be computed from scratch at each iteration, but updating methods can be applied. In the proposed DSS, we have implemented five widely used basis update methods: (i) Gaussian elimination, (ii) MATLAB's built-in function called inv, (iii) LU decomposition (Markowitz, 1957), (iv) Modification of the Product Form of the Inverse (Benhamadou, 2002), and (v) Product Form of the Inverse (Dantzig & Orchard-Hays, 1954).

Gaussian elimination

Gaussian elimination is a method for solving systems of linear equations that can be used to compute the inverse of a matrix in simplex type algorithms. Gaussian elimination performs a forward substitution, which reduces the given matrix to a triangular or echelon form, and a back substitution, which calculates the solution of the given system of linear equations. Gaussian elimination with partial pivoting requires $O(n^3)$ time complexity.

Gaussian elimination has been implemented using the mldivide operator of MATLAB. The new basis inverse using Gaussian elimination can be found using equation 38:

$$\left(A_B\right)^{-1} = A_B \setminus I \tag{38}$$

Built-in function inv of MATLAB

The basis inverse can be computed using the built-in function of MATLAB called inv, which uses LAPACK routines to compute the basis inverse. This function is already compiled and optimized for MATLAB, so its execution time is smaller compared with the other relevant methods that compute the explicit basis inverse. The time-complexity, though, remains $O(n^3)$.

LU decomposition

LU decomposition method factorizes a matrix as the product of a lower L and an upper U triangular factors that can be used to compute the inverse of a matrix. In order to compute the L and U factors, the built-in function of MATLAB called lu has been used. LU decomposition requires $O(n^3)$ time complexity.

Modification of the product form of the inverse

The Modification of the Product Form of the Inverse (MPFI) updating scheme has been presented by Benhamadou (2002). The new basis inverse $(A_{\bar{B}})^{-1}$ can be computed from the previous basis inverse $(A_B)^{-1}$ using an outer product of two vectors and one matrix addition, as shown in equation 39:

$$\left(A_{\bar{B}}\right)^{-1} = \left(A_{\bar{B}}\right)^{-1}_{r.} + v \otimes \left(A_B\right)^{-1}_{r.} \tag{39}$$

The outer product of equation 39 requires m^2 multiplications and the addition of two matrices requires m^2 additions. Hence, the time complexity of this basis updating scheme is $\Theta(m^2)$.

Product form of the inverse

The Product Form of the Inverse (PFI) updating scheme uses information only about the entering and leaving variables along with the current basis $(A_B)^{-1}$ in order to update the new basis $(A_{\bar{B}})^{-1}$. The new basis inverse can be updated at any iteration using equation 40:

$$\left(A_{\bar{B}}\right)^{-1} = \left(A_B E\right)^{-1} = E^{-1}(A_B)^{-1} \tag{40}$$

where E^{-1} is the inverse of the eta-matrix and can be computed by the equation 41:

$$\mathrm{E}^{-1} = \mathrm{I} - \frac{1}{\mathrm{h_{rl}}}(\mathrm{h_l} - \mathrm{e_l})\mathrm{e_l^T} = \begin{bmatrix} 1 & & -\mathrm{h_{1l}}/\mathrm{h_{rl}} & & \\ & \ddots & \vdots & & \\ & & 1/\mathrm{h_{rl}} & & \\ & & \vdots & \ddots & \\ & & -\mathrm{h_{ml}}/\mathrm{h_{rl}} & & 1 \end{bmatrix} \tag{41}$$

If the current basis inverse is computed using regular multiplication, then the time complexity of the PFI basis updating scheme is $\Theta(m^3)$.

5. PIVOTING RULES

A critical step in the solution of a linear programming problem is the selection of the entering variable in each iteration, called pivoting or pricing. The key factor that will determine the number of the iterations that the linear programming algorithm performs is the pivoting rule (Maros & Khaliq, 2002). Good choices of the entering variable can lead to fast convergence to the optimal solution, while poor choices lead to more iterations. In the proposed DSS, we have implemented eight widely used pivoting rules: (i) Bland's rule (Bland, 1977), (ii) Dantzig's rule (Dantzig, 1963), (iii) Greatest Increment Method (Klee & Minty, 1972), (iv) Least Recently Considered Method (Zadeh, 1980), (v) Partial Pricing rule, (vi) Queue rule, (vii) Stack rule, and (viii) Steepest Edge rule (Goldfarb & Reid, 1977).

Bland's rule

Bland's rule (Bland, 1977) selects as entering variable the first among the eligible ones, that is the leftmost among columns with negative relative cost coefficient. Although Bland's rule avoids cycling, it has been observed in practice that this pivoting rule can lead to stalling, a phenomenon where long degenerate paths are produced.

Dantzig's rule

The first pivoting rule that was used in the simplex algorithm is Dantzig's rule or largest coefficient rule (Dantzig, 1963). In this pivoting rule, the column A_l with the most negative $\overline{c_l}$ is selected as the entering variable. Dantzig's rule guarantees the largest reduction in the objective value per unit of non-basic variable $\overline{c_l}$ increase. Its worst-case complexity is exponential (Klee & Minty, 1972). However, Dantzig's rule is claimed as simple but powerful enough to guide simplex algorithm into short paths (Thomadakis, 1994).

Greatest increment method

Greatest Increment Method (Klee & Minty, 1972) selects as entering variable the variable with the largest total objective value improvement. Greatest Increment Method calculates the improvement of the objective value for each non-basic variable and then selects the variable that offers the largest improvement in the objective value. Although this pivoting rule can lead to fast convergence to the optimal solution, this advantage is eliminated by the additional computational cost per iteration. Gärtner (1995) constructed LPs that Greatest Increment Method showed exponential complexity.

Least recently considered method

In the first iteration of Least Recently Considered Method (Zadeh, 1980), the entering variable l is selected according to Bland's rule, i.e. the leftmost among columns with negative relative cost coefficient. In the next iterations, Least Recently Considered Method starts searching for the first eligible variable with index greater than l. If $l = n$ then Least Recently Considered Method starts searching from the first column again. Least Recently Considered Method prevents stalling and it has been observed that it performs fairly well in practice (Thomadakis, 1994). However, its worst-case complexity has not been proved yet.

Partial pricing rule

Partial Pricing methods are variants of the standard rules that take only a part of non-basic variables into account when searching for the entering variable. In the DSS presented in Section 7, we have implemented the partial pricing rule as variant for Dantzig's rule using static partial pricing, i.e. non-basic variables are divided into equal segments with predefined size and the pricing operation is carried out segment by segment.

Queue rule

Queue is a FIFO (First-In-First-Out) data structure, where the first element added to the queue is the first one to be removed. In the pivoting rule of queue, two queues are constrcuted; the first one holds the indices of the basic variables, while the other the indices of the non-basic variables. The entering and leaving variables are selected from the front of the corresponding queue. The variable, which is extracted from the front of the queue that holds the basic variables, is inserted to the end of the queue that holds the non-basic variables. Respectively, the variable, which is extracted from the front of the queue that holds the non-basic variables, is inserted to the end of the queue that holds the basic variables.

Stack rule

Stack is a LIFO (Last-In-First-Out) data structure, where the last element added to the stack is the first one to be removed. In the stack rule, the entering and leaving variables are selected from the top of the corresponding stack. The variable, which is extracted from the top of the stack that holds the basic variables, is inserted to the top of the stack that holds the non-basic variables. Respectively, the variable, which is extracted from the top of the stack that holds the non-basic variables, is inserted to the end of the stack that holds the basic variables.

Steepest edge rule

Steepest Edge Rule or All-Variable Gradient Method (Goldfarb & Reid, 1977) selects as entering variable the variable with the most objective value reduction per unit distance. Although this pivoting rule can lead to fast convergence to the optimal solution, this advantage is debatable due to the additional computational cost per iteration.

6. DECISION SUPPORT SYSTEM ANALYSIS AND DESIGN

The decision making process that the policy maker can perform using the proposed DSS is presented in Figure 1. Initially, the decision maker formulates the given problem as a linear programming problem. In step 2, the decision maker gathers, validates and verifies the adequate data and the input data are uploaded to the DSS. Then, the decision maker either selects the desired algorithm and the appropriate methods to solve a linear programming problem or selects the option to perform a computational study with all combinations of algorithms and methods. Then, the algorithms' evaluation and execution step follows. In the last step, the results are presented and analyzed. Finally, the decision maker validates the results and considers if the provision of further feedback on the operation of the DSS is necessary; if so, the updated decision making process is performed again.

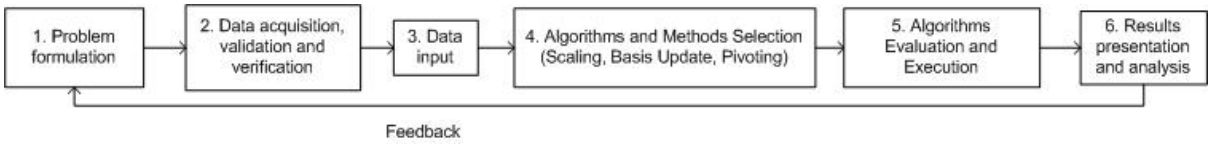

Figure 1: Decision Making Process

The interaction between the decision maker and the DSS is presented in Figure 2. The decision maker uploads the input file in the standardized mps format, selects the algorithms (RSM, EPSA), the scaling methods (arithmetic mean, de Buchet for the case p = 1, de Buchet for the case p = 2, entropy, equilibration, geometric mean, IBM MPSX, Lp-norm for the case p = 1, Lp-norm for the case p = 2, Lp-norm for the case p = ∞, de Buchet for the case p = ∞), the basis update methods (Gaussian elimination, MATLAB's built-in function called inv, LU decomposition, Modification of the Product Form of the Inverse, Product Form of the Inverse), the pivoting rules (Bland's rule, Dantzig's rule, Greatest Increment Method, Least Recently Considered Method, Partial Pricing Rule, Queue Rule, Stack Rule, Steepest Edge Rule) and presses the 'Report' button. Then, the DSS validates the input data and executes the algorithms for each combination of methods (scaling methods, basis update methods and pivoting rules). Then, it collects: (i) the total execution time, (ii) the time to perform the scaling, (iii) the time to perform the basis inverse, (iv) the time to perform the pivoting, (v) the number of iterations, and (vi) the objective value; and presents these results to the decision maker. Finally, the decision maker can export the results as a pdf file for further analysis.

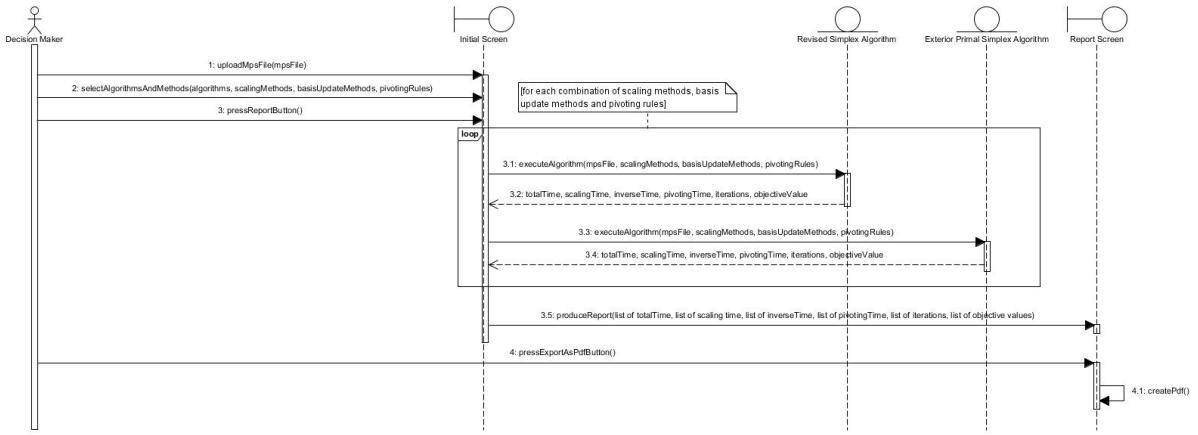

Figure 2: Sequence Diagram

Figure 3 presents the class diagram of the proposed DSS. InitialScreen is a boundary class that includes three methods that respond to the decision maker's action events: (i) upload input file, (ii) select algorithms, scaling methods, basis update methods and pivoting rules, and (iii) press 'Report' button. SimplexAlgorithm is an abstract class that includes the common attributes and methods of RevisedSimplexAlgorithm and ExteriorPrimalSimplexAlgorithm. Matrix A contains the constraints coefficients, vector c the objective function coefficients, vector b the right-hand side values, vector Eqin the type of constraints (equality or inequality), and variable minMax the type of the linear programming problem (minimization or maximization). Furthermore, SimplexAlgorithm class includes three methods that perform the scaling, the basis inverse and the pivoting according to the selected methods. RevisedSimplexAlgorithm and ExteriorPrimalSimplexAlgorithm classes override the abstract method executeAlgorithm of the SimplexAlgorithm and perform their unique steps for the solution of the linear programming problem.

Scaling is an abstract class that includes the common attributes and methods of all different scaling methods. Matrix A again contains the constraints coefficients, vector c the objective function coefficients, vector b the right-hand side values, vector r the row scaling factors and vector s the column scaling factors. All the derived scaling classes override the abstract method scaling of the Scaling class and perform their steps to scale the linear programming problem.

BasisUpdateMethod is an abstract class that includes the common attributes and methods of all different basis update methods. Matrix Ab contains the previous basis inverse, vector hl the pivot column, k the index of the leaving variable and m the number of the constraints. All the derived basis update classes override the abstract method inverse of the BasisUpdateMethod class and perform their steps to update the basis matrix.

PivotingRule is an abstract class that includes the common attributes and methods of all different pivoting rules. Vector Sn contains the cost coefficients. All the derived pivoting classes override the abstract method pivoting of the PivotingRule class and perform their steps to make the pivoting step. Finally, some of the derived pivoting classes, like Steepest, contain some unique attributes, i.e. Steepest contains vector nonBasicList that holds the indices of the non-basic variables, matrix A the constraints coefficients and matrix Ab the basis matrix.

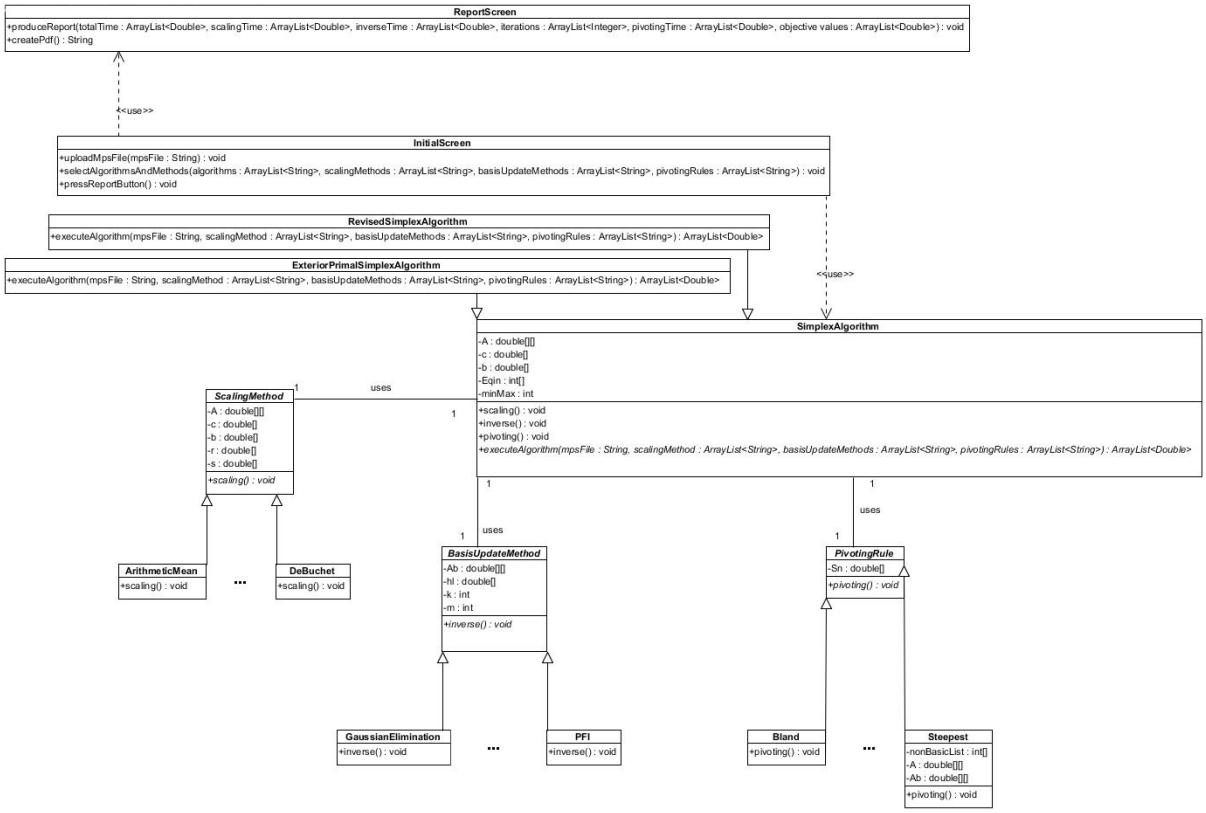

Figure 3: Class Diagram

7. DECISION SUPPORT SYSTEM PRESENTATION

Simplex type algorithms, scaling methods, basis update methods and pivoting rules have been implemented using MATLAB. Then, these algorithms and methods were converted to Java classes using the MATLAB Builder JA. The web interface of the DSS was designed using Java Server Pages (JSP). The DSS is a freeware and can be used mainly for academic purposes. The proposed DSS can be used to solve large-scale LPs. We have achieved to run a 15,000 x 15,000 dense linear programming problem on a quad-processor Intel Core i7 3.4 GHz with 32 Gbyte of main memory and 8 cores, a clock of 3700 MHz, running under Microsoft Windows 7 64-bit. Furthermore, we also managed to solve many medium- and large-scale Netlib problem set (optimal, Kennington and infeasible LPs) (Gay, 1985).

The initial screen of the DSS is presented in Figure 4. The decision maker presses the 'Browse' button in order to upload the file containing the LP in mps format. MPS is a well-known file format for mathematical programming. After the upload of the input file, the decision maker can view useful information of the selected LP, like: (a) the filename, (b) the number of the constraints, (c) the number of the variables, (d) the number of the nonzeros in matrix A, and (e) the density of matrix A. Moreover, the decision maker selects the algorithms, the scaling methods, the basis update methods and the pivoting rules that will be included in the comparison. By pressing the 'Report' button a screen with a thorough report is presented (Figure 5). This screen includes the objective value, the number of the iterations, the total time, the times needed to perform the scaling and the basis update, and the number of iterations for each pivoting rule. Finally, the decision maker may export the report as a pdf file for further analysis.

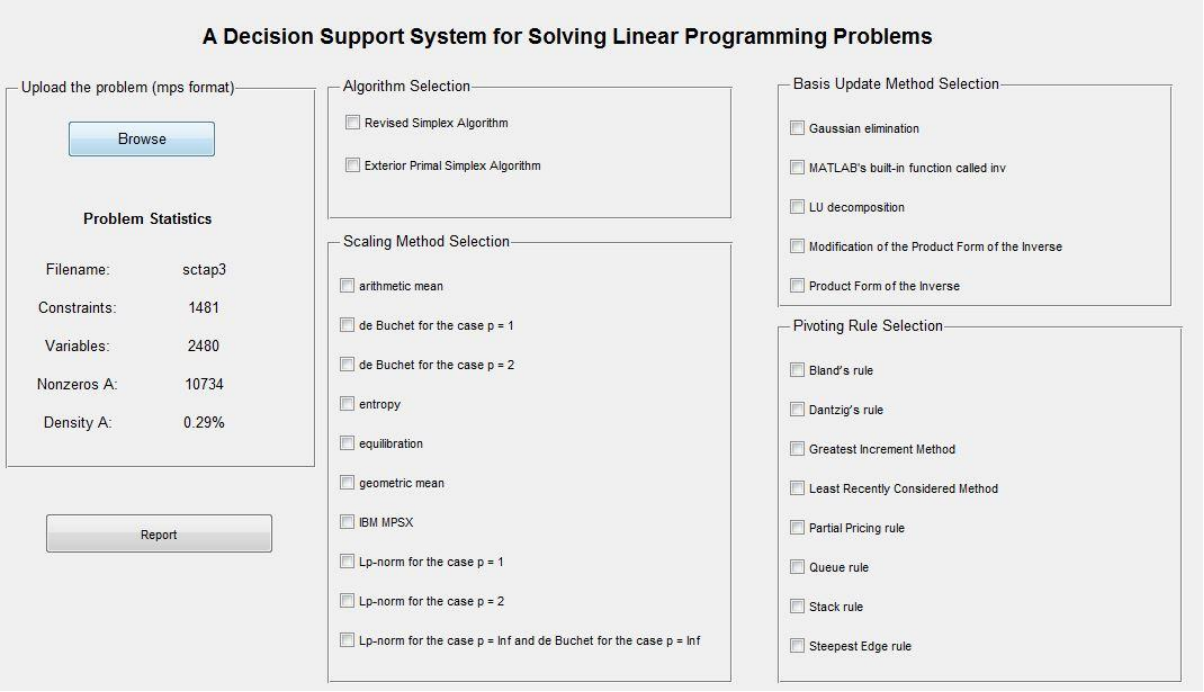

Figure 4: Initial Screen of the proposed DSS

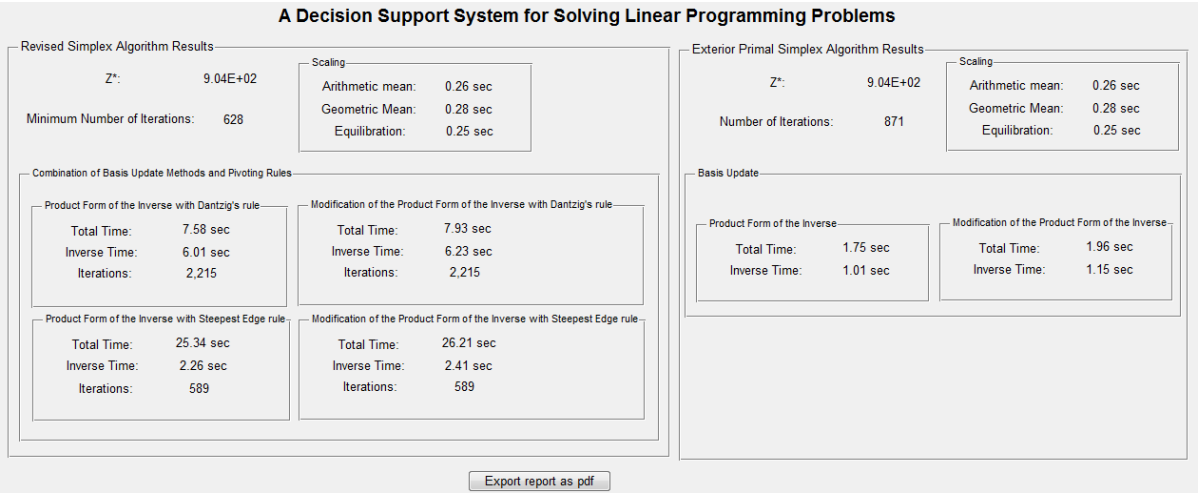

Figure 5: Report Screen

Many LPs of the Netlib set are real-world problems. Figures 4 and 5 present a case study for SCTAP3, which is a problem of the Netlib set. SCTAP3 is a problem in the optimization of the dynamic flow over a traffic network where congestion is modelled explicitly in the flow equations (Ho & Loute, 1981). This model, originally formulated in Merchant and Nemhauser (1978), is further studied in Ho (1980). SCTAP3 includes 2,480 variables with 1,481 constraints. From the results that are presented in Figure 5, it is concluded that EPSA with the PFI updating method and the equilibration scaling technique is the best choice for the solution of this problem.

The proposed DSS offers important managerial implications. Initially, the decision maker can formulate its problem as a linear programming problem. Problems that can be formulated as linear programming problems might refer to telecommunications, bio-informatics, supply

chain management, water management, resource allocation, etc. Furthermore, the decision-policy maker can gain an insight of the best algorithm, scaling method, basis update method and pivoting rule that best suits the given problem. On the other hand, a limitation that exists on the proposed DSS is that some problems cannot be formulated as linear programming problems.

8. CONCLUSIONS

Many problems from different scientific fields can be formulated as linear programming problems. Many DSS that utilize linear programming algorithms exist, but they do not take into consideration the structure of the problem in order to suggest the best combination of the linear programming algorithm and the appropriate methods for each step of the algorithm. In this paper, we presented a web-based DSS that supports decision makers in the solution of linear programming problems with a variety of linear programming algorithms and techniques. More specifically, the decision maker has two choices: (i) either select which linear programming algorithm, scaling method, basis update method and pivoting rule will be used to solve the given problem, or (ii) perform a computational study with all combinations of algorithms and methods in order to export a detailed report and find the combination of algorithms and methods that best suits the given problem.

In future work, we plan to enhance the DSS with an option that can exploit the structure of the input problem prior of the execution of the algorithms and propose to the decision maker the best suitable combination of algorithms and methods. Finally, we plan to present real application case studies on which the proposed DSS can be utilized.

REFERENCES

Benhamadou, M. (2002). On the simplex algorithm ‘revised form’. *Advances in Engineering Software*, 33(11), 769-777.

Benichou, M., Gauthier, J. M., Hentges, G., & Ribiere, G. (1977). The efficient solution of large-scale linear programming problems—some algorithmic techniques and computational results. *Mathematical Programming*, 13(1), 280-322.

Bhargava, H. K., Power, D. J., & Sun, D. (2007). Progress in Web-based decision support technologies. *Decision Support Systems*, 43(4), 1083-1095.

Bland, R. G. (1977). New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 103-107.

Dantzig, G. B. (1953). *Computational Algorithm of the Revised Simplex Method*. RAND Report RM-1266, The RAND Corporation, Santa Monica, CA.

Dantzig, G. B., & Orchard-Hays, W. (1954). *The product form for the inverse in the simplex method*. Mathematical Tables and Other Aids to Computation, 64-67.

Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press.

Faye, R. M., Mora-Camino, F., Sawadogo, S., & Niang, A. (1998, October). An intelligent decision support system for irrigation system management. In *1998 IEEE International Conference on Systems, Man, and Cybernetics* (Vol. 4, pp. 3908-3913). IEEE.

Gärtner, B. (1995). *Randomized optimization by Simplex-type methods*. Doctoral dissertation, Freie Universität, Germany.

Gay, D. M. (1985). Electronic mail distribution of linear programming test problems. *Math. Program. Soc. COAL News*, 13, 10-12.

Ghodsypour, S. H., & O'brien, C. (1998). A decision support system for supplier selection using an integrated analytic hierarchy process and linear programming. *International Journal of Production Economics*, 56, 199-212.

Goldfarb, D., & Reid, J. K. (1977). A practicable steepest-edge simplex algorithm. *Mathematical Programming*, 12(1), 361-371.

Ho, J. K. (1980). A successive linear optimization approach to the dynamic traffic assignment problem. *Transportation Science*, 14(4), 295-305.

Ho, J. K., & Loute, E. (1981). A set of staircase linear programming test problems. *Mathematical Programming*, 20(1), 245-250.

Klee, V., & Minty, G. J. (1972). How Good is the Simplex Algorithm. In O. Shisha (Ed.), *Inequalities – III*, New York and London: Academic Press Inc.

Lappi, J., Nuutinen, T., & Siitonen, M. (1996). A linear programming software for multilevel forest management planning. In *Management systems for a global economy with global resource concerns*, 470-482. Proceedings of the symposium on system analysis in forest resources. Pacific Grove, CA.

Larsson, T. (1993). On scaling linear programs—Some experimental results. *Optimization*, 27(4), 355-373.

Lauer, J., Jacobs, L. W., Brusco, M. J., & Bechtold, S. E. (1994). An interactive, optimization-based decision support system for scheduling part-time, computer lab attendants. *Omega*, 22(6), 613-626.

Lourenço, J. C., Morton, A., & Bana e Costa, C. A. (2012). PROBE–A multicriteria decision support system for portfolio robustness evaluation. *Decision Support Systems*, 54, 534-550.

Markowitz, H. M. (1957). The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3), 255-269.

Maros, I., & Haroon Khaliq, M. (2002). Advances in design and implementation of optimization software. *European Journal of Operational Research*, 140(2), 322-337.

Mavrotas, G. (2000). *Multiple Objective Linear Programming under Uncertainty: Development of a DSS and Application in Energy Planning*. Unpublished doctoral dissertation, National Technical University of Athens, Greece.

Merchant, D. K., & Nemhauser, G. L. (1978). A model and an algorithm for the dynamic traffic assignment problems. *Transportation Science*, 12, 183-199.

Paparrizos, K., Samaras, N., & Stephanides, G. (2003). An efficient simplex type algorithm for sparse and dense linear programs. *European Journal of Operational Research*, 148(2), 323-334.

Ploskas, N., Samaras, N., & Papathanasiou, J. (2013). A Web-Based Decision Support System Using Basis Update on Simplex Type Algorithms. In *Decision Support Systems II-Recent Developments Applied to DSS Network Environments* (pp. 102-114). Springer Berlin Heidelberg.

Power, D. J. (2000). Web-based and model-driven decision support systems: concepts and issues. In *Americas Conference on Information Systems*, Long Beach, California (Vol. 3, pp. 27-002).

Power, D. J., & Kaparthi, S. (2002). Building Web-based decision support systems. *Studies in Informatics and Control*, 11(4), 291-302.

Thomadakis, M. E. (1994). *Implementation and Evaluation of Primal and Dual Simplex Methods with Different Pivot-Selection Techniques in the LPBench Environment A Research Report*. Texas: Texas A&M University.

Tomlin, J. A. (1975). On scaling linear programming problems. *Math. Program. Stud.*, 4, 146-166.

Van Vuuren, J. H., & Grundlingh, W. R. (2001). An active decision support system for optimality in open air reservoir release strategies. *International Transactions in Operational Research*, 8(4), 439-464.

Venkataramanan, M. A., & Bornstein, M. (1991). A decision support system for parking space assignment. *Mathematical and Computer Modelling*, 15(8), 71-76.

Zadeh, N. (1980). What is the worst case behavior of the simplex algorithm. *Polyhedral Computation*, 48, 131-143.