LINEAR PROGRAMMING FOR DATABASE ENVIRONMENT

Akira Kawaguchi and Jose Alfredo Perez
Department of Computer Sciences, The City College of New York, New York, New York 10031, U.S.A.

Keywords: Linear programming, simplex method, revised simplex method, database, stored procedure.

Abstract: Solving large-scale optimization problems requires an integration of data-analysis and data-manipulation capabilities. Nevertheless, little attempt has been made to facilitate general linear programming solvers for database environments. Dozens of sophisticated tools and software libraries that implement linear programming model can be found. But, there is no database-embedded linear programming tool seamlessly and transparently utilized for database processing. The focus of this study is to fill out this kind of technical gap of data analysis and data manipulation, in the event of solving large-scale linear programming problems for the applications built on the database environment. Specifically, this paper studies the representation of the linear programming model in relational structures and the computational method to solve the linear programming problems. Foundations for and preliminary experimental results of this study are presented.

1 INTRODUCTION

Linear programming is a powerful technique for dealing with the problem of allocating limited resources among competing activities, as well as other problems having a similar mathematical formation (Winston, 1994; Richard, 1991; Walsh, 1985). It has become an important field of optimization in the areas of science and engineering. It has become a standard tool of great importance for numerous business and industrial organizations. In particular, large-scale linear programming problems arise in practical applications such as logistics for large spare-parts inventory, revenue management and dynamic pricing, finance, transportation and routing, network design, and chip design (Hillier and Lieberman, 2001).

While these problems inevitably involve the analysis of a large amount of data, there has been relatively little work in the context of database processing. Little attempt has been made to facilitate data-driven analysis with data-oriented techniques. In today's marketplace, dozens of sophisticated tools and software libraries that implement linear programming model can be found. Nevertheless, these products do not work with database systems seamlessly. They rather require additional software layers built on top of databases to extract and transfer data in the databases. The focus of our study gathered here is to fill out this kind of technical gap of data analysis and data manipulation, in the event of solving large-scale linear programming problems for the applications built on the database environment.

In mathematics, linear programming problems are optimization problems in which the objective function to characterize an optimality of a problem and the constraints to express specific conditions for that problem are all *linear* (Hillier and Lieberman, 2001; Thomas H. Cormen and Stein, 2001). Two families of solution methods, so-called *simplex methods* (Dantzig, 1963) and *interior-point methods* (Karmarkar, 1984), are in wide use and available as computer programs today. Both methods progressively improve series of trial solutions by visiting edges of the feasible boundary or the points within the interior of feasible region, until a solution is reached to satisfy the conditions for an optimum. In fact, it is known that large problem instances render even the best of codes nearly unusable (Winston, 1994). Furthermore, the program libraries available today are found outside the standard database environment, thus mandat-

Kawaguchi A. and Alfredo Perez J. (2007).
LINEAR PROGRAMMING FOR DATABASE ENVIRONMENT.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 186-191
DOI: 10.5220/0001652701860191

ing the use of a special interface to interact with these tools for linear programming computations.

This paper presents a detailed account on the methodology and technical issues to realize a general linear programming method for the relational database environment. In general, relational database is not for matrix operations for solving linear programming problems. Indeed, realizing matrix operations on top of the standard relational structure is non-trivial. In this paper, implementation techniques and key issues for this development are studied extensively, and a model suitable to capture the dynamics of linear programming computations is incorporated into the aimed development, by way of realizing a set of procedural interfaces that enables a standard database language to define problems within a database and to derive optimal solutions for those problems without requiring users to write detailed program statements.

Specifically, we develop a set of ready to use *stored procedures* to solve general linear programming problems. A stored procedure is a group of SQL statements, precompiled and physically stored within a database (Gulutzan and Pelzer, 1999; Gulutzan, 2007). It forms a logical unit to encapsulate a set of database operations, defined with application program interface to perform a particular task, thereby having complex logic run inside the database via a stored procedure. The exact implementation of a stored procedure varies from one database to another, but is supported by most major database vendors. To this end, we will show an implementation using MySQL open-source database system.

As a summary, contributions of this paper are threefold: First, we present a detailed account on the methodology and technical issues to integrate a general linear programming method into relational databases. Second, we present the development as forms of stored procedures for today's representative database systems. Third, we present an experimental performance study based on a comprehensive system that implements all these concepts. This is to investigate applications of business decision problems to a database environment for practical use.

2 FUNDAMENTALS

Consider the matrix notation expressed in the set of equations (1) below. The standard form of the linear programming problem is to maximize an objective function $Z = \mathbf{c}^T\mathbf{x}$, subject to the functional constraints of $\mathbf{Ax} \leq \mathbf{b}$ and non-negativity constraints of $\mathbf{x} \geq \mathbf{0}$, with $\mathbf{0}$ in this case being the n-dimensional zero column vector. A coefficient matrix $\mathbf{A}$ and column vectors $\mathbf{c}$, $\mathbf{b}$, and $\mathbf{x}$ are defined in the obvious manner such that each component of the column vector $\mathbf{Ax}$ is less than or equal to the corresponding component of the column vector $\mathbf{b}$.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix},$$

$$\mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad (1)$$

The goal is to find an optimal solution, that is, the most favorable values of the objective function among feasible ones for which all the constraints are satisfied. The *simplex method* (Dantzig, 1963) is an algebraic iterative procedure where each round of computation involves solving a system of equations to obtain a new trial solution for the optimality test. The simplex method relies on the mathematical property that the objective function's maximum must occur on a corner of the space bounded by the constraints of the feasible region.

To apply the simplex method, linear programming problems must be converted into augmented form, by introducing non-negative *slack variables* to replace non-equalities with equalities in the constraints. The problem can then be rewritten on the following form:

$$\mathbf{x_s} = \begin{bmatrix} x_{n+1} \\ x_{n+2} \\ \vdots \\ x_{n+m} \end{bmatrix}, [\mathbf{A}, \mathbf{I}] \begin{bmatrix} \mathbf{x} \\ \mathbf{x_s} \end{bmatrix} = \mathbf{b}, \begin{bmatrix} \mathbf{x} \\ \mathbf{x_s} \end{bmatrix} \geq \mathbf{0},$$

$$\begin{bmatrix} 1 & -\mathbf{c}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \mathbf{I} \end{bmatrix} \begin{bmatrix} Z \\ \mathbf{x} \\ \mathbf{x_s} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix} \quad (2)$$

In equations (2) above, $\mathbf{x} \geq \mathbf{0}$, a column vector of slack variables $\mathbf{x_s} \geq \mathbf{0}$, and $\mathbf{I}$ is the $m \times m$ identity matrix. Following the convention, the variables set to zero by the simplex method are called *nonbasic variables* and the others are called *basic variables*. If all of the basic variables are non-negative, the solution is called *a basic feasible solution*. Two basic feasible solutions are adjacent if all but one of their nonbasic variables are the same. The spirit of the simplex method utilizes a rule for generating from any given basic feasible solution a new one differing from the old in respect of just one variable.

Thus, moving from the current basic feasible solution to an adjacent one involves switching one vari-

able from nonbasic to basic and vice versa for one other variable. This movement involves replacing one nonbasic variable (called *entering* basic variable) by a new one (called *leaving* basic variable) and identifying the new basic feasible solution. The simplex algorithm is summarized as follows:

Simplex Method:

1. Initialization: transform the given problem into an augmented form, and select original variables to be the nonbasic variables (*i.e.*, $\mathbf{x} = \mathbf{0}$), and slack variable to be the basic variables (*i.e.*, $\mathbf{x_s} = \mathbf{b}$).
2. Optimality test: rewrite the objective function by shifting all the nonbasic variables to the right-hand side, and see if the sign of the coefficient of every nonbasic variable is positive, in which case the solution is optimal.
3. Iterative Step:
   - 3.1 Selecting an entering variable: as the nonbasic variable whose coefficient is largest in the rewritten objective function used in the optimality test.
   - 3.2 Selecting a leaving variable: as the basic variable that reaches zero first when the entering basic variable is increased, that is, the basic variable with the smallest upper bound.
   - 3.3 Compute a new basic feasible solution: by applying the Gauss-Jordan method of elimination, and apply the above optimality test.

2.1 Revised Simplex Method

The computation of the simplex method can be improved by reducing the number of arithmetic operations as well as the amount of round-off errors generated from these operations (Hillier and Lieberman, 2001; Richard, 1991; Walsh, 1985). Notice that n nonbasic variables from among the $n+m$ elements of $\left[\mathbf{x}^T, \mathbf{x_s}^T\right]^T$ are always set to zero. Thus, eliminating these n variables by equating them to zero leaves a set of m equations in m unknowns of the basic variables. The spirit of the *revised simplex method* (Hillier and Lieberman, 2001; Winston, 1994) is to preserve the only pieces of information relevant at each iteration, which consist of the coefficients of the nonbasic variables in the objective function, the coefficients of the entering basic variable in the other equations, and the right-hand side of the equations.

Specifically, consider the equations (3) below. The revised method attempts to derive a basic (square) matrix $\mathbf{B}$ of size $m \times m$ by eliminating the columns corresponding to coefficients of nonbasic variables from $[\mathbf{A}, \mathbf{I}]$ in equations (2). Furthermore, let $\mathbf{c_B}^T$ be the vector obtained by eliminating the coefficients of nonbasic variables from $\left[\mathbf{c}^T, \mathbf{0}^T\right]^T$ and reordering the elements to match the order of the basic variables. Then, the values of the basic variables become $\mathbf{B}^{-1}\mathbf{b}$ and $Z = \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{b}$. The equations (2) become equivalent with equations (3) after any iteration of the simplex method.

$$\mathbf{B} = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1m} \\ B_{21} & B_{22} & \cdots & B_{2m} \\ \vdots & \vdots & & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mm} \end{bmatrix},$$

$$\begin{bmatrix} 1 & \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A} - \mathbf{c}^T & \mathbf{c_B}^T\mathbf{B}^{-1} \\ \mathbf{0} & \mathbf{B}^{-1}\mathbf{A} & \mathbf{B}^{-1} \end{bmatrix} \begin{bmatrix} Z \\ \mathbf{x} \\ \mathbf{x_s} \end{bmatrix} = \begin{bmatrix} \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{b} \\ \mathbf{B}^{-1}\mathbf{b} \end{bmatrix} \quad (3)$$

This means that only $\mathbf{B}^{-1}$ needs to be derived to be able to calculate all the numbers used in the simplex method from the original parameters of $\mathbf{A}, \mathbf{b}, \mathbf{c_B}$—providing efficiency and numerical stability.

2.2 Relational Representation

A *relational model* provides a single way to represent data as a two-dimensional table or a *relation*. An n-ary relation being a subset of the Cartesian product of n domains has a collection of rows called *tuples*. Implementing the simplex and revised simplex methods must locate the exact position of the values for the equations and variables of the linear programming problem to be solved. However, the position of the tuples in the table is not relevant in the relational model. Tuple ordering and matrix handling are beyond the standard relational features, and these are the important issues addressed to implement the linear programming solver using the simplex method.

Simplex calculations are most conveniently performed with the help of a series of tables known as *simplex tableaux* (Dantzig, 1963; Hillier and Lieberman, 2001). A simplex tableaux is a table that contains all the information necessary to move from one iteration to another while performing the simplex method. Let $\mathbf{x_B}$ be a column vector of m basic variables obtained by eliminating the nonbasic variables from $\mathbf{x}$ and $\mathbf{x_s}$. Then, the initial tableaux can be expressed as,

$$\begin{bmatrix} Z & 1 & -\mathbf{c}^T & \mathbf{0} & 0 \\ \mathbf{x_B} & \mathbf{0} & \mathbf{A} & \mathbf{I} & \mathbf{b} \end{bmatrix} \quad (4)$$

The algebraic treatment based on the revised simplex method (Hillier and Lieberman, 2001; William

H. Press and Flannery, 2002) derives the values at any iteration of the simplex method as,

$$\begin{bmatrix} Z & 1 & \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{A}-\mathbf{c}^T & \mathbf{c_B}^T\mathbf{B}^{-1} & \mathbf{c_B}^T\mathbf{B}^{-1}\mathbf{b} \\ \mathbf{x_B} & \mathbf{0} & \mathbf{B}^{-1}\mathbf{A} & \mathbf{B}^{-1} & \mathbf{B}^{-1}\mathbf{b} \end{bmatrix} \quad (5)$$

For the matrices expressed (4) and (5), the first two column elements do not need to be stored into persistent memory. Thus, the simplex tableaux can be a table using the rest of the three column elements in the relational model. Specifically, a linear programming problem in the augmented form (equations (2)) can be seen as a relation:

$$\texttt{tableaux}(\underline{\texttt{id}}, x_1, x_2, \cdots x_n, \texttt{rhs})$$

Every variable of the constraints and the objective function becomes an attribute of the relation, together with the right hand that becomes the `rhs` column on the table. The `id` column serves a *key* that can uniquely determine every variable of the constraints and of the objective function. When the LP problem in augmented form is inputted into the database, every constraint is identified by a unique integer value going from 1 to n in the `id` column, where n is the number of constraints for the problem plus the objective function. Thus by applying relational operations, it is feasible to know the position of every constraint and variable for a linear programming problem, and to proceed with the matrix operations necessary to implement the simplex algorithm.

3 SYSTEM DEVELOPMENT

The availability of real-time databases capable of accepting and solving linear programming problems helps us examine the effectiveness and practical usability in integrating linear programming tools into database environment. Towards this end, a general linear programming solver is developed on top of the *de facto* standard database environment, with the combination of a PHP application for the front-end and a MySQL application for the back-end. Note that the implementation of this linear programming solver is strictly within the database technology, not relying on any outside programming language.

The PHP front-end enables the user to input the number of variables and number of constraints of the linear programming problem to solve. With these values, it generates a dynamic Web interface to accept the values of the objective function and the values of the constraining equations. The Web interface also allows the user to upload a file in a MPS (Mathematical Programming System) format that defines a linear programming problem. The MPS file format serves a standard for describing and archiving linear programming and mixed integer programming problems (Organization, 2007). A special program is built to convert MPS data format into SQL statements for populating an linear programming instance—the main objective of this development is to obtain benchmark performances for large-scale linear programming problems.

The MySQL back-end performs iterative computations by the use of a set of stored procedures precompiled and integrated into the database structure. The system encapsulates an API for processing a simplex method that requires the execution of several SQL queries to produce a solution.

The output of the system is shown in Figure 1, in which each table represents the tableaux containing the values resulted at each iteration. The system presents successive transformations and optimal solution if it exists.

Linear Program Solver

x1	x2	x3	x4	x5	x6	x7	RHS
3	-2	1	1	0	0	0	5
1	3	-4	0	1	0	0	9
0	1	5	0	0	-1	0	1
1	1	1	0	0	0	1	6
-2	3	-1	0	0	0	0	0

x1	x2	x3	x4	x5	x6	x7	RHS
3	0	11	1	0	-2	0	7
1	0	-19	0	1	3	0	6
0	1	5	0	0	-1	0	1
1	0	-4	0	0	1	1	5
-2	0	-16	0	0	3	0	-3

x1	x2	x3	x4	x5	x6	x7	RHS
3.66666666	0	-1.66666666	1	0.66666666	0	0	11
0.33333333	0	-6.33333333	0	0.33333333	1	0	2
0.33333333	1	-1.33333333	0	0.33333333	0	0	3
0.66666667	0	2.33333333	0	-0.33333333	0	1	3
-2.99999999	0	2.99999999	0	-0.99999999	0	0	-9

x1	x2	x3	x4	x5	x6	x7	RHS
4.14285714	0	0	1	0.42857143	0	0.71428571	13.14285714
2.14285717	0	0	0	-0.57142856	1	2.71428572	10.14285717
0.71428572	1	0	0	0.14285714	0	0.57142857	4.71428572
0.28571429	0	1	0	-0.14285714	0	0.42857143	1.28571429
-3.85714286	0	0	0	-0.57142857	0	-1.28571429	-12.85714286

Result: Optimal Solution.

Figure 1: Simplex method iterations and optimal solution.

3.1 Stored Procedure Implementation

Stored procedures can have direct accesses to the data in the database, and run their steps directly and entirely within the database. The complex logic runs inside the database engine, thus faster in processing requests because numerous context switches and a great deal of network traffic can be eliminated. The database system only needs to send the final results back to the user, doing away with the overhead of communicating potentially large amounts of interim

Table 1: Experimental set and measured execution time.

name	m	n	nonzeros	optimal value	time	standard deviation
ADLITTLE	57	97	465	2.2549496316E+05	1 min. 25 sec.	2.78 sec.
AFIRO	28	32	88	-464.7531428596	35 sec.	1.67 sec.
BLEND	75	83	521	-3.0812149846E+01	1 min. 5 sec.	3.20 sec.
BRANDY	221	249	2150	1.5185098965E+03	2 min. 50 sec.	4.25 sec.

data back and forth (Gulutzan, 2007; Gulutzan and Pelzer, 1999).

Stored procedures are supported by most DBMSs, but there is a fair amount of variation in their syntax and capabilities even their internal effects are almost invisible. Our development uses MySQL version 5.0.22 at the time of this paper writing (as for MySQL version 5, stored procedures are supported). The next code listing is the stored procedure used to create the table to store the linear programming problem to be solved by the application. The first part of the stored procedure consists of the prototype of the function and the declaration of the variables to be used in the procedure.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS
  `lpsolver`.`createTable` $$
CREATE PROCEDURE `lpsolver`.`createTable`
   (constraints INT, variables INT)
BEGIN
  DECLARE i INT;
  DECLARE jiterator VARCHAR(50);
  DECLARE statement VARCHAR(1000);
  DROP TABLE IF EXISTS tableaux;
```

Because of the dynamic nature of the calculations for solving linear programming problems, our stored procedure relies on the extensive use of prepared SQL statements. In the next code block, the SQL statement to create a table is generated on the fly, based on the number of variables and constraints of the problem to solve. The generated procedure is then passed to the database for execution.

```
SET statement = 'CREATE TABLE
    tableaux(id INT(5) PRIMARY KEY, ';
SET i = 1;
table_loop:LOOP
  IF i > constraints + variables + 1 THEN
    LEAVE table_loop;
  END IF;
  SET jiterator = CONCAT('j',i);
  SET statement = CONCAT(statement,
      jiterator);
  SET statement = CONCAT(statement,
      ' DOUBLE DEFAULT 0');
  IF i <= constraints + variables THEN
     SET statement = CONCAT(statement, ', ');
  END IF;
  SET i = i + 1;
  END LOOP table_loop;
  SET statement = CONCAT(statement, ')');
  SET @sql_call = statement;
  PREPARE s1 FROM @sql_call;
  EXECUTE s1;
  DEALLOCATE PREPARE s1;
END $$
DELIMITER ;
```

3.2 Experimental Results

To see the effectiveness of the implementation, various linear programming problems were selected from commonly available Netlib linear programming library (Organization, 2007). As one case, see Table 1 for a sufficiently large problem set. The values m and n indicate the size, $m \times n$, of the coefficient matrix $\mathbf{A}$ in equations (1), or equivalently, m is the number of constraints and n is the number of decision variables.

All experiments were performed by an Intel 586 based standalone machine with 1.2 GHz CPU and 512 MB memory that running MySQL 5.0.22. The data values were extracted from Netlib MPS files to populate the problems into the database prior to run the simplex method. The time measured does not include this data preparation process, but only the execution of the stored procedure to produce a solution. The time listed in the following table is the average of ten executions of each problem. The set of results shown in Table 1 are based on the implementation of the revised simplex method contained in the stored procedures.

One limiting factor is the fact that MySQL allows to have up to 1000 columns on a table. Given that our implementation is based on mapping of a simplex tableaux into a database relation, the number of variables plus the number of constraints cannot exceed the number of columns allowed for a MySQL table. This prohibited us from testing the problems in the Netlib library that exceed the column size of 1000. Finally, we observed one problem when trying to find optimal solutions for larger problems with higher numbers of columns, variables and zero elements. The computation never came to an end, indicating that the problem had become unbounded, which can be attributed to the tableaux becoming *ill-conditioned* as a consequence of truncation errors resulted from repeated matrix operations.

4 RELATED WORK

A vast amount of effort for the establishment of theory and practice is observed today. Certain special cases of linear programming, such as network flow problems and multi-commodity flow problems are considered important enough to have generated much research on specialized algorithms for their solution (Winston, 1994; Thomas H. Cormen and Stein, 2001; Hillier and Lieberman, 2001). A number of algorithms for other types of optimization problems work by solving linear programming problems as subproblems. Historically, ideas from linear programming have inspired many of the central concepts of optimization theory, such as duality, decomposition, and the importance of convexity and its generalizations (Hillier and Lieberman, 2001).

There are approaches considered to fit a linear programming model, such as integer programming and nonlinear programming (Alexander, 1998; Richard, 1991; Hillier and Lieberman, 2001). But, our research focuses on the area of iterative methods for solving linear systems. Some of the most significant contributions and the chain of contributions building on each other are summarized in (Saad and van der Vorst, 2000), especially a survey of the transition from simplex methods to interior-point methods is presented in (Wang, 99). In terms of implementation techniques, the work of (Morgan, 1976; Shamir, 1987) provided us with introductory sources for reference. There are online materials such as (Optimization Technology Center and Laboratory, 2007; Organization, 2007) to help us understand the details and plan for experimental design.

5 CONCLUSION

The subject of this research is to respond a lack of database tools for solving a linear programming problem defined within a database.

We described the aim and approach for integrating a linear programing method into today's database system, with our goal in mind to establish a seamless and transparent interface between them. As demonstrated, this is feasible by the use of stored procedures, the emerging database programming standard that allows for complex logic to be embedded as an API in the database, thus simplifying data management and enhancing overall performance.

We implemented a general linear programming solver on top of the PHP and MySQL software layers. The experiments with several benchmark problems extracted from the Netlib library showed its correct optimal solutions and basic performance measures.

The contents of this paper are the first dissemination of our upcoming series of publications—we plan to extend this research into several directions. First, the limiting factor of MySQL's table column size for the implementation needs to be addressed. Also, the code must be optimized to reduce the overall execution time. Second, more experiments must be done to collect additional performance measures. Furthermore, other commercial databases such as Oracle should be included for comparative study, and nonlinear and other optimization methods should also be explored.

REFERENCES

Alexander, S. (1998). *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, NY.

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

Gulutzan, P. (2007). *MySQL 5.0 New Features: Stored Procedures*. MySQL AB, http://www.mysql.com.

Gulutzan, P. and Pelzer, T. (1999). *SQL-99 Complete, Really*. CMP Books.

Hillier, F. S. and Lieberman, G. J. (2001). *Introduction to Operations Research*. McGraw-Hill, 8th edition.

Karmarkar, N. K. (1984). A new polynomial-time algorithm for linear programming and extensions. *Combinatorica*, 4:373–395.

Morgan, S. S. (1976). A comparison of simplex method algorithms. Master's thesis, University of Florida.

Optimization Technology Center, N. U. and Laboratory, A. N. (2007). The linear programming frequently asked questions.

Organization, T. N. (2007). The netlib repository at utk and ornl.

Richard, B. D. (1991). *Introduction To Linear Programming: Applications and Extensions*. Marcel Dekker, New York, NY.

Saad, Y. and van der Vorst, H. (2000). Iterative solution of linear systems in the 20-th century. *JCAM*.

Shamir, R. (1987). The efficiency of the simplex method: a survey. *Manage. Sci.*, 33(3):301–334.

Thomas H. Cormen, Charles E. Leiserson, R. L. R. and Stein, C. (2001). *Introduction to Algorithms, Chapter29: Linear Programming*. MIT Press and McGraw-Hill, 2nd edition.

Walsh, G. R. (1985). *An Introduction to Linear Programming*. John Wiley & Sons, New York, NY.

Wang, X. (99). From simplex methods to interior-point methods: A brief survey on linear programming algorithms.

William H. Press, Saul A. Teukolsky, W. T. V. and Flannery, B. P. (2002). *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University.

Winston, W. L. (1994). *Operations Research, Applications and Algorithms*. Duxbury Press.