We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

19

Linear Programming in Database

Akira Kawaguchi and Andrew Nagel
Department of Computer Science, The City College of New York. New York, New York
United States of America

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

Abstract

Linear programming is a powerful optimization technique and an important field in the areas of science, engineering, and business. Large-scale linear programming problems arise in many practical applications, and solving these problems requires an integration of data-analysis and data-manipulation capabilities. Database technology has become a central component of today's information systems. Almost every type of organization is now using database systems to store, manipulate, and retrieve data. 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 models can be found. But, there is no database-embedded linear programming tool seamlessly and transparently utilized for database processing. The focus of the study in this chapter is to fill this technical gap between data analysis and data manipulation, by solving large-scale linear programming problems with applications built on the database environment. Specifically, this chapter studies the representation of the linear programming model in relational structures, as well as the computational method to solve the linear programming problems. We have developed 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, thereby having complex logic run inside the database. We show versions of procedures in the open-source MySQL database and commercial Oracle database system. The experiments are performed with several benchmark problems extracted from the Netlib library. Foundations for and preliminary experimental results of this study are presented.*

1. Introduction

* This work has been partly supported by New York State Department of Transportation and New York City Department of Environment Protection.

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 formulation (Winston, 1994, Richard, 1991, Walsh, 1985). It has become an important field of optimization in the areas of science and engineering and 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 addressing this in the database context. Little serious 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 models 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 this technical gap between data analysis and data manipulation by solving large-scale linear programming problems with applications built on the database environment.

In mathematics, linear programming problems are optimization problems in which the objective function to characterize 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 the feasible region, until a solution is reached that satisfies the constraints and cannot be improved. 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 mandating the use of a special interface to interact with these tools for linear programming computations.

This chapter gives a detailed account of the methodology and technical issues related to general linear programming in the relational (or object-relational) database environment. Our goal is to find a suitable software platform for solving optimization problems on the extension of a large amount of information organized and structured in the relational databases. In principle, whenever data is available in a database, solving such problems should be done in *a database way*, that is, computations should be closed in the world of the database. There is a standard database language, ANSI SQL, for the manipulation of data in the database, which has grown to a level comparable to most ordinary programming or scripting languages. Eliminating reliance on a commercial linear programming package, thus eliminating the overhead of data transfer between database and package is what we hope to achieve.

There are also the issues of trade-off. A basic nature of linear programming is a collection of matrices defining a problem and a sequence of algebraic operations repeatedly applied to

these matrices, hence giving a perfect match for array-based programming in scientific computations. In general, the relational database is not designed for matrix operations like solving linear programming problems. Indeed, realizing matrix operations on top of the standard relational (or object-relational) structure is non-trivial. On the other hand, at the heart of the database system is the ability to effectively manage resources coupled with an efficient data access mechanism. The response to user is made by the best available sequence of operations, or so-called optimized queries, on the actual data. When handling extremely large matrices, the system probably gives a performance advantage over the unplanned or ad hoc execution of the program causing an insatiable use of virtual memory (thus causing thrashing) for the disposition of arrays.

In this chapter, implementation techniques and key issues for this development are studied extensively. 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 two sets 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 an application program interface to perform a particular task, thereby having complex logic run inside the database. 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 implementations using MySQL open-source database system and freely available Oracle Express Edition selected from the commercial domain. Our choice of these popular database environments is to justify the feasibility of concepts and to draw comparisons of their usability.

The rest of this chapter is organized as follows: Section 2 defines the linear programming model and introduces our approach to express the model in the relational database. Section 3 presents details of developed simulation system and experimental performance studies. Section 4 discusses related work, and Section 5 concludes our work gathered in this chapter.

2. Fundamentals

A linear programming problem consists of a collection of linear inequalities on a number of real variables and a fixed linear function to maximize or minimize. In this section, we summarize the principle technical issues in formulating the problem and some solution method in the relational database environment.

2.1 Linear Programming Principles

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 **Ax** is less than or equal to the corresponding component of the column vector **b**. But all forms of linear programming problems arise in practice, not just ones in the standard form, and we must deal with issues such as minimization objectives, constraints of the form $\mathbf{Ax} \geq \mathbf{b}$ or $\mathbf{Ax} = \mathbf{b}$, variables ranging in negative values, and so on. Adjustments can be made to transform every non-standard problem into the standard form. So, we limit our discussion to the standard form of the problem.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$
$$\mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{1}$$

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 a so-called *augmented form*, by introducing non-negative *slack variables* to replace non-equalities with equalities in the constraints. The problem can then be rewritten in the following form:

$$\mathbf{x_s} = \begin{bmatrix} x_{n+1} \\ x_{n+2} \\ \vdots \\ x_{n+m} \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_s \end{bmatrix} = \mathbf{b}, \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_s \end{bmatrix} \geq \mathbf{0}, \begin{bmatrix} 1 & -\mathbf{c}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \mathbf{I} \end{bmatrix} \begin{bmatrix} Z \\ \mathbf{x} \\ \mathbf{x}_s \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix} \tag{2}$$

In equations (2) above, $\mathbf{x} \geq \mathbf{0}$, a column vector of slack variables $\mathbf{x}_s \geq \mathbf{0}$, and **I** is the m × 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 variable 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.

2.2 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 $[\mathbf{x}^T, \mathbf{x}_S^T]^T$ 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 only the pieces of information relevant at each iteration, which consists 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 × 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 $[\mathbf{c}^T, \mathbf{0}^T]^T$ 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.

2.3 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*. Implementions of 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 solve. However, the position of the tuples in the table is not relevant in the relational model. By definition, tuple ordering and matrix handling are beyond the standard relational features, and these are the most important issues that need to be addressed to implement the linear programming solver within the database using the simplex method. We will explore two distinct methods for representing matrices in the relational model.

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 tableau is a table that contains all the information necessary to move from one iteration to another while performing the simplex method. Let **xB** be a column vector of *m* basic variables obtained by eliminating the nonbasic variables from **x** and **xs**. Then, the initial tableau can be expressed as,

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

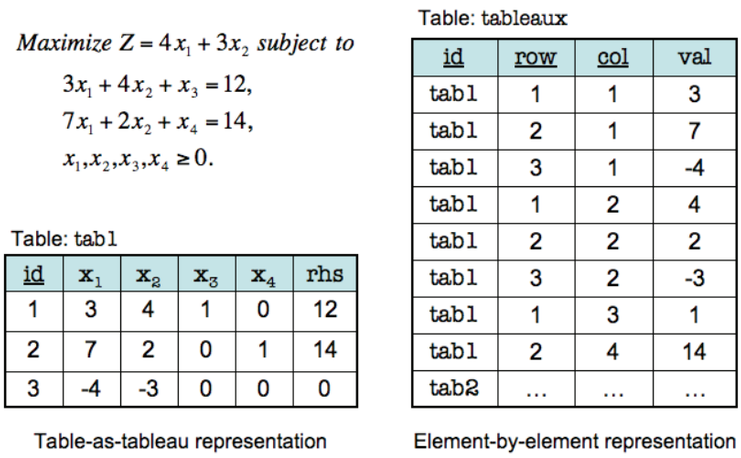
The algebraic treatment based on the revised simplex method (Hillier and Lieberman, 2001, William H. Press and Flannery, 2002) derives the values at any iteration of the simplex method as,

$$\begin{bmatrix} Z & 1 & \mathbf{c}_\mathbf{B}^\mathrm{T}\mathbf{B}^{-1}\mathbf{A} - \mathbf{c}^T & \mathbf{c}_\mathbf{B}^\mathrm{T}\mathbf{B}^{-1} & \mathbf{c}_\mathbf{B}^\mathrm{T}\mathbf{B}^{-1}\mathbf{b} \\ \mathbf{x_B} & \mathbf{0} & \mathbf{B}^{-1}\mathbf{A} & \mathbf{B}^{-1} & \mathbf{B}^{-1}\mathbf{b} \end{bmatrix} \tag{5}$$

For the matrices expressed (4) and (5), the first two column elements do not need to be stored in persistent memory. Thus, the simplex tableau can be a table using the rest of the three column elements in the relational model. Creating table instances as simplex tableaux is perhaps the most straighforward way. Indeed, our MySQL implementation in the next section uses this representation, in which a linear programming problem in the augmented form (equations (2)) can be seen as a relation:

$$\text{tableau}(\underline{\text{id}}, x1, x2, \cdots xn, \text{rhs})$$

A variable of the constraints and the objective function becomes an attribute of the relation, together with the right hand side that becomes the rhs column on the table. The id column serves as a *key* that can uniquely determine every variable of the constraints and of the objective function in the tuple. A constraint of the linear programming problem in the augmented form is identified by a unique positive integer value ranging 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. See Figure 1 for the table instance populated with a simple example.



$$Maximize\ Z = 4x_1 + 3x_2\ subject\ to$$
$$3x_1 + 4x_2 + x_3 = 12,$$
$$7x_1 + 2x_2 + x_4 = 14,$$
$$x_1, x_2, x_3, x_4 \geq 0.$$

Table: tabl

id	x_1	x_2	x_3	x_4	rhs
1	3	4	1	0	12
2	7	2	0	1	14
3	-4	-3	0	0	0

Table-as-tableau representation

Table: tableaux

id	row	col	val
tabl	1	1	3
tabl	2	1	7
tabl	3	1	-4
tabl	1	2	4
tabl	2	2	2
tabl	3	2	-3
tabl	1	3	1
tabl	2	4	14
tab2	...	...	...

Element-by-element representation

Fig. 1. Two representations of the initial simplex tableau in the relational database

The main drawback of this design is a fixed structure of table. An individual table needs to be created for each problem, and the cost of defining (and dropping) table becomes part of the process implementing the simplex method. The number of tables in the database will increase as the collection of problems to solve accumulates. This may cause administrative strain for database management. Subtle issues arise in the handling of large-scale problems. The table maps to the full instance of the matrix even if the problem has sparsely populated non-zero data. Thousands of zero values (or null values specific to database) held in a tuple pile up a significant amount of space. Besides, a tuple of a large number of non-zero values is problematic because the physical record holding such a tuple may not fit into a disk block. Accessing a spanned record over multiple disk blocks is time-consuming.

As an alternative to the table-as-tableaux structure, element-by-element represenation can be considered. The simplex tableaux are decomposed to a collection of values, each of which is a tuple consisting of a tableau id, a row position, a column position, and a value in the specified position. This is to say that the table no longer possesses the shape of a tableau but has the information to locate every element in the tableau. Missing elements are zeros, thus space efficient for sparse contents. A single table can gather all the problem instances, in that the elements in the specific tableau are found by the use of the tableau id. The size of the tuple is small because there are only four attributes in the tuple. In return, there is a time

overhead for finding a specific element in the table. Our Oracle XE implementation detailed in the next section utilizes this representation.

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 the 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 or Oracle application for the backend. Note that the implementation of this linear programming solver is strictly within the database technology, not relying on any outside programming language.

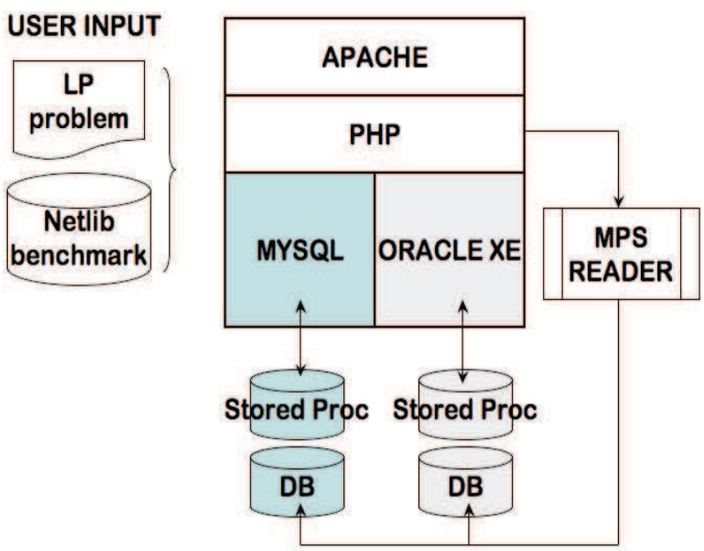


Fig. 2. Architecture of the implemented linear programming solver

The systems architecture is summarized in Figure 2. 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 as 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 a linear programming instance. The main objective of this development is to obtain benchmark performances for large-scale linear programming problems.

The MySQL and Oracle back-ends perform iterative computations by the use of a set of stored procedures precompiled and integrated into the database structure. The systems encapsulate an API for processing a simplex method that requires the execution of several SQL queries to produce a solution. The input and output of the system are shown in Figure 3 in which each table of the right figure represents the tableau containing the values resulted from each iteration of the simplex method. The system presents successive transformations and optimal solution if it exists.

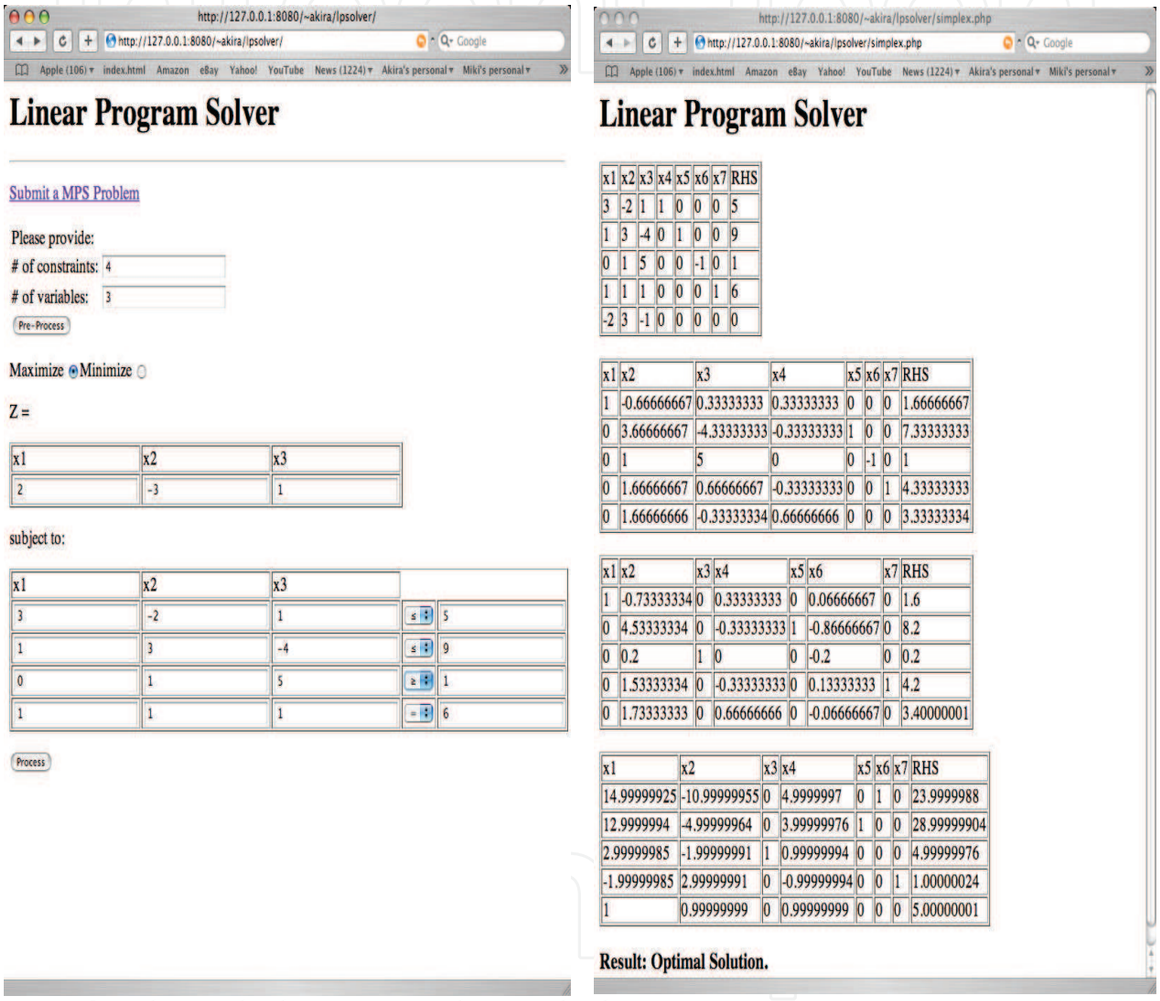


Fig. 3. Simplex method iterations and optimal solution

3.1 Stored Procedure Implementation MySQL

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

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.

Table 1. MySQL Experimental set and measured execution time

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 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 (Perez, 2007). 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 MySQL

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 was 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 Table 1 is the average of ten executions of each problem. The results 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 this implementation is based on mapping of a simplex tableau 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 tableau becoming ill-conditioned as a consequence of truncation errors resulted from repeated matrix operations (Kawaguchi and Perez, 2007).

3.3 Stored Procedure Implementation in Oracle XE

A Linear Programming Solver was implemented in Oracle XE stored procedures with a simple web interface built in PHP. Oracle XE is a free version of the Oracle database system subject to some restrictions. Notably, Oracle XE will only utilize a single processor, and total user data is limited to 4 GB. Still, stored procedures are entirely supported, as well as advanced indexing techniques, making Oracle XE an attractive alternative.

The web interface shown in Figure 4 provides for creation, editing, and display of large matrices, and allows the user to perform elementary matrix operations. The "Linear

Programming" menu provides options for uploading and parsing standard MPS files, for solving the problem automatically, and for viewing performance data. The "Work Tableau" option is shown, and provides an interface where the user can view the tableau, or any portion of it. They can choose an element to pivot on, or by clicking the "suggest" button, the column and row selected by the simplex method is high-lighted and displayed. This was useful for debugging, but also serves as a good educational tool since a student can go through the algorithm step by step.

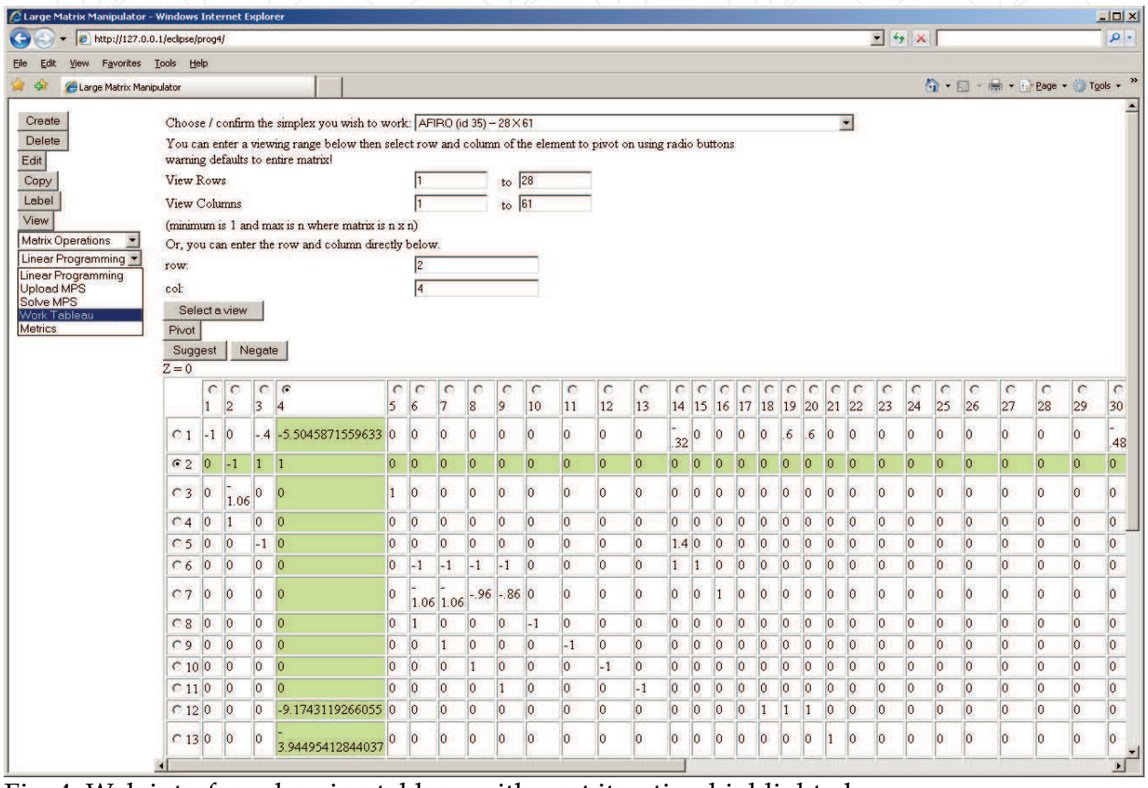


Fig. 4. Web interface showing tableau with next iteration highlighted

The data model was also changed in this implementation to address the limitation of max number of columns allowed in a table. Rather than creating a table with enough columns to contain the simplex tableau, a matrix is represented by two tables, one describing the properties, and one containing the row position, column position, and value of each element. This has the added benefit of simplifying matrix operations. Since the tables described below hold any matrices stored in the system, creation, deletion, and alteration of matrices relies only on INSERT, DELETE, and UPDATE statements, with no need for 'on the fly' table creation or procedure compilation.

matrix_property(matrix_id, name, row_size, column_size)
matrix_values(matrix_id, row_position, column_position, value)

Although allowing for an indefinite number of columns in a stored matrix, this model introduces a problem in data look up. In the MySQL implementation, since each row of the

matrix was a tuple in the relation, once a row is retrieved by the database, every element in the row is either in main memory or efficiently buffered since the tuple is stored contiguously on disk. But in the Oracle implementation, getting each element in a row could require a new disk read. Fortunately, Oracle XE supports Hash Index for fast retrieval of tuples that share a common hash value. Since in the Oracle model, any two elements in the same row share the same matrix_id and row_position, we can build such an index.

In the Oracle XE environment, this is achieved by first creating a hash cluster (equivalent to hash buckets), then creating the table that is designated to be stored in the cluster according to a hash of at least one of its attributes.

```
CREATE CLUSTER Matrix_index (matid NUMBER, rowpos NUMBER) SIZE
512 SINGLE TABLE HASHKEYS 1000;

CREATE TABLE "MVALUE"
(
"MATID" NUMBER NOT NULL ENABLE,
"ROWPOS" NUMBER NOT NULL ENABLE,
"COLPOS" NUMBER NOT NULL ENABLE,
"CELL_VALUE" NUMBER(26,14) NOT NULL ENABLE,
CONSTRAINT "MVALUE_UK1" UNIQUE ("MATID", "ROWPOS", "COLPOS")
ENABLE,
CONSTRAINT "MVALUE_FK" FOREIGN KEY ("MATID") REFERENCES
"MPROPERTY" ("MATID") ON DELETE CASCADE ENABLE
)
CLUSTER "Matrix_index" ("MATID", "ROWPOS");
```

To actually benefit from this index, it is necessary to make use of Cursors in the stored procedures that operate on the matrices. Cursors are featured in many database systems, and provide an interface to declare complex SELECT statements and iterate over the results. By declaring cursors, rather than using a SELECT statement inside a FOR loop, the query optimizer is better able to take advantage of the hash index and retrieve entire rows of the matrix with minimal disk I/O. Implementing the solver in this way resulted in more than 50% performance increase, particularly as problem size was increased.

Name	*m*	*n*	Non zeros	Optimal Value (calculated)	Std. Err.	Iterations	Avg. Time (sec)	Std. Dev
ISRAEL	175	143	2358	-896641.4612	0.0004%	308	784	1.41
LOTFI	154	309	1086	-25.26470426	0.0000%	203	79.9	9.18
AFIRO	28	33	88	-464.7531429	0.0000%	11	0.4	0.52
SC105	106	104	281	-52.20206121	0.0000%	110	77.9	1.1
SC205	206	204	552	-52.20206121	0.0000%	257	702.2	2.9
ADLITTLE	57	98	465	225494.9384	0.0000%	146	38.1	1.73
BLEND	75	84	521	-30.82213945	0.0324%	118	41.2	0.63
BRANDY	221	250	2150	557.6518123	63.2764%	659	1433.8	7.11

Table 2. Oracle Experimental set and measured execution time

3.4 Experimental Results Oracle XE

As before, a set of linear programming problems was selected from the Netlib library (Organization, 2007). The results are presented in Table 2, where again, *m* is the number of constraints and *n* is the number of decision variables. The standard error is calculated based on the optimal value our solver returned compared to the optimal values published by Netlib.The Oracle experiments were run on an Intel D865GV board with 3 GHz Pentium 4 CPU and 2 GB memory running Oracle XE 10g. Other parameters and timing of the experiment are as described in 3.2.

While the model used in the Oracle implementation does allow for storage and simple manipulation of matrices larger than 1000 x 1000, it did not solve all the problems experienced in the MySQL version. Truncation and rounding errors created deviance from the published optimal value, hence the inclusion of standard error in Table 2. For larger problems, this sometimes degenerated to an 'ill-conditioned' state, as with the MySQL implementation and the algorithm may not finish or may report incorrect results as with BRANDY.

Rounding errors were sometimes more of an issue in the Oracle implementation because it uses the Big M variant of the Simplex method when dealing with problems in non-standard form. Briefly, this involves introducing artificial variables to make each constraint feasible for the basic solution and penalizing those artificial variables with a large coefficient in the objective function, this penalty being the Big M. While this method is easy to implement, it requires more iterations, which introduces more potential for rounding/truncation errors.

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 sub-problems. 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.

The contents of this chapter are extended from the work gathered in (Kawaguchi and Perez, 2007), in which the experimental performance of MySQL implementation is shown. A more detailed implementation of MySQL stored procedures can be found in (Perez, 2007).

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 programming 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. As a summary, contributions of the discussions presented in this chapter 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.

Our implementation of general linear programming solvers is on top of the PHP, MySQL, and Oracle software layers. The experiments with several benchmark problems extracted from Netlib library showed its correct optimal solutions and basic performance measures. However, due to the methods used, rounding errors were still an issue for large problems despite the system having the capacity to work with large matrices. We thus plan to continue this research in several directions. Although the Oracle system can work with large matrices, both implementations have too much rounding error to solve linear programming problems that would be considered large by commercial standards. This should be addressed by the implementation of a more robust method. Overall, the code must be optimized to reduce the execution time, which could also be improved by tuning the size and number of hash buckets in the index. We will perform more experiments to collect additional performance measures. Non-linear and other optimization methods should also be explored.

6. 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.

Kawaguchi, A. and Perez, A. J. (2007). Linear programming for database environment. *ICINCO-ICSO* 2007: 186-191

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.

Perez, A. J. (2007). Linear programming for database environment. Master's thesis, City College of New York.

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.

New Developments in Robotics Automation and Control

Edited by Aleksandar Lazinica

ISBN 978-953-7619-20-6
Hard cover, 450 pages
Publisher InTech
Published online 01, October, 2008
Published in print edition October, 2008

This book represents the contributions of the top researchers in the field of robotics, automation and control and will serve as a valuable tool for professionals in these interdisciplinary fields. It consists of 25 chapter that introduce both basic research and advanced developments covering the topics such as kinematics, dynamic analysis, accuracy, optimization design, modelling , simulation and control. Without a doubt, the book covers a great deal of recent research, and as such it works as a valuable source for researchers interested in the involved subjects.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Akira Kawaguchi and Andrew Nagel (2008). Linear Programming in Database, New Developments in Robotics Automation and Control, Aleksandar Lazinica (Ed.), ISBN: 978-953-7619-20-6, InTech, Available from: http://www.intechopen.com/books/new_developments_in_robotics_automation_and_control/linear_programming_in_database

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen