4-1997

MLPQ: A LINEAR CONSTRAINT DATABASE SYSTEM WITH AGGREGATE OPERATORS

YiMing Li
University of Nebraska - Lincoln

MLPQ: A LINEAR CONSTRAINT DATABASE SYSTEM WITH AGGREGATE OPERATORS

by

YiMing Li

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Peter Revesz

Lincoln, Nebraska

April, 1997

MLPQ: A LINEAR CONSTRAINT DATABASE SYSTEM WITH AGGREGATE OPERATORS

YiMing Li, M.S.

University of Nebraska, 1997

Adviser: Peter Revesz

In this project report, I will discuss a Multiple Linear Programming Query (MLPQ) system and the theoretical background of this system.The MPLQ system is developed to solve some realistic problems involving both linear programming (UP) techniques and linear constraint databases (LCDBs) theory. The MLPQ system is aimed at providing a mechanism of bridging these two important areas. system basically consists of three parts which are a linear constraint database, an LP solver, and an interface between the LCDB and the LP solver. The LCDB of the MLPQ system contains multiple linear programming problems. The LP solver used in the MPLQ is an implementation Of the SIMPLEX method.An important feature of the MLPQ system is that it can handle the SQL aggregate Operators, such as minimum *Min*, maximum *Max*, summation *Sum*, and average *Avg*. The MLPQ system provides an efficient way of evaluation of aggregate operators for linear constraint databases.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor Dr. Peter Revesz for his guidance and encouragement on this research project. I also thank Dr.Istvan Bogardi and Dr.Jean-Camille Birget for serving on my advisory committee and for their time and counsel.

I would like to dedicate my thesis to my parents, my wife Ping and my son George for their love, encouragement and support throughout my life. They always believed in me and I can never be grateful enough.

Table of Contents

Chapter 1

Introduction

1.1 Constraint Database

The databases most widely used today are the unrestricted relational databases in which the databases are presented in a set of tables. The limitation of the relational databases is that they cannot store the data presented as arithmetical expressions. However, in real world practice it is often necessary to present the data containing arithmetical expressions. For instance, if we want to store the information of an area occupied by a house in a database, it will be quite convenient to present the area using four arithmetical line equations for the four sides of the house. To store such arithmetical expression,we need a new type of database called constraint database [6], [11]. In the following I will use an example to discuss why we need constraint databases to handle the data involving arithmetical computation.

Let us consider a postage example. Mr. Johnson is a sales man for an electronic hardware company in Omaha, Nebraska. He wants to send three packages of electronic components to three customers who live in Boston, Chicago, and Dallas. The weights of the packages are 12.6, 27.3 and 37.5 ounces respectively. Mr. Johnson would like to know how much he should pay for his packages if he sends those pack-

ages through a post office.

Assuming a postage fee charged for a domestic package can be computed based on the weight of a package and the postage rate associated with the weight, we may Construct two relational database tables and one SQL query to present this problem. The package information including weight, origin and destination of the package is stored in a relational table called Package.

Package

Origin	Destination	Weight
Omaha	Boston	12.6
Omaha	Chicago	27.3
Omaha	Dallas	37.5

Assuming the postage charge is $0.53 per ounce for a package up to 5 ounce, and $0.45 per ounce for the weight above 5 ounce but below 15 ounce, and so on. The maximum be sent is 50 ounce. The postage rate information based on package weight can be stored in the table called Postage such that

Postage

Weight	Fee
5	2.65
...	...
50	16.65

Mr. Johnson's query can be expressed in SQL such that

```
SELECT      Sum(Fee)
FROM        Package, Postage
```

There is a serious problem with the above database construction. The weight included in the Package can be any real number between 0 and 50 ounces. To match the weight in the Package, the Postage table has to include an infinite number of tuples. So it will be unreasonable to present the postage information using the traditional relational database. One approach to this problem is to allow some mathematical expressions in the stored data, For instance, to present the Postage table in a finite format, we can construct a new postage rate relation called Postage1. In Postage1, the attributes of the relation are specified by variables and constraints involving arithmetical computations Assuming that postage' rate increases with weight piecewise linearly, the relation Postage1 will be presented as follows.

Postage1

Weight	Fee	
w	f	0 <=w, w <=5, f=w * 0.53
w	f	5 <w, w <=15, f=2.65 + (w - 5) * 0.45
w	f	15 <w, w <=30, f=7.15 + (w - 15) * 0.3
w	f	30 <w, w <=50, f=11.65 + (w - 30) * 0.25

Mr. Johnson's query can be constructed the same way as before by substitution of relation Postage1 for table Postage. The query will return a total fee of 27.785 for the three packages.

The database presented by the relation Postage1 is called constraint database [6], [11] Constraint databases are the databases in which any attributes are specified using variables and constraints. One specific type Of constraint databases is called linear constraint database (LCDB). LCDB was developed in recent years to include linear constraint mathematical expressions in relations and queries [1], [5]. The linear constraint database knowledge is particularly important in enhancing the capabil-

ity of traditional databases, constraint databases and constraint logic programming systems.

1.2 Linear Programming

Linear programming is a mathematical technique widely used in engineering, management, planning and economics to optimize (minimize or maximize) single or objective functions based on given number of linear constraints. The linear programming problem can be described in its standard format such that

Minimize z = cx

Subject to Ax = b

and x $\geq$ 0

where z is an objective function, x is the vector of variables need to be solved for, A is a coefficient matrix, and b and c are vectors of known constants. The most often used LP solving technique is called the SIMPLEX method, which was developed in the 1940s and thereafter. The method is a very efficient and uses only basic arithmetical operations [14]. It also has advantage to produce the dual values for the constraints along with the best values of the decision variables. The problem with the SIMPLEX method is that it has difficulty to handle problems in which the variables must be integer values [13]. Many other LP solving methods developed since 1950s include "cutting-plan algorithms" for solving integer problem [4], "branch-and-bound method [8]," and the polynomial time algorithm developed by Kachiyan in 1979 [7]. A family of LP techniques called Interior-Point method has been developed in late 80s. The methods using nonlinear programming approach can be used to solve many large scale problems. Different from SIMPLEX method which always stays on the boundaries,

the Interior-Point methods construct a sequence of trial solutions that go through the interior of the solution space. "Although more and more algorithms have been developed in the past, for practical, real life problems, the SIMPLEX method is still remain the dominant linear programming algorithm for at least the near future [2]."

The theory behind the SIMPLEX method is that only the corner points Of the feasible region can be optima. No point in the feasible region can ever be better than all corner points. Those corner points will give basic feasible solution to the problem. The basic procedure of the SIMPLEX method is to obtain any basic feasible solution to start with. Then it checks the neighboring solutions to see if they are better. If there is better solution, move to it. It repeats the procedure until no improvement can be found.

1.3 Objectives

This project is aimed at producing a system combining the linear programming techniques and the linear constraint databases together through an interface. The system is required to handle some SQL aggregator operators over linear constraint databases which contain multiple linear programming problems.

Chapter 2

Linear Programming and Constraint Databases

2.1 The Motivation for Linear Programming in Databases

The motivation for using linear programming in linear constraint databases can be traced to Brodsky et al. [3]. In their paper, they suggested a merger of linear programming techniques and linear constraint databases. What they found is that the linear programming technique will be very helpful in retrieve boundaries of variables in LCDBs. However, they did not detail how they use the linear programming technique in LCDBs and they also did not talk about the aggregator operators in the constraint SQL.

The following is a motivation example [12] to answer why we need linear programming in constraint databases. Let us consider a food production company which has manufacturing plants in four cities A, B, C, and D around the world. The company produces candies, chocolate bars, ice cream, and yogurt. For producing a unit of each of these four items in the city A, the company needs 15, 8, 10, and 15 units of sugar, 30, 25, 5, and 10 units of milk, and 0, 50, 25, 0 units of chocolate respectively. Further, each unit of these four items yields a profit of 300, 250, 100, and 150 respectively. In city A the company has on store 3,000 units of sugar, 8,000 units of milk, and 2,000 unit of chocolate, Similar data is also available for the other three cities.

LCDBs can conveniently represent the above sort of data in a constraint relation called Food. In relation Food, chocolate-bar and ice cream are abbreviated as C-B and I-C.

Food

City	Candy	C-B	I-S	Yogurt	Profit	
A	x1	x2	x3	x4	z	300x1+250x2+100X3+150x4=z 15x1+8x2+10x3+15x4 $\leq$ 3000 30x1+25x2+5x3+10x4 $\leq$ 8000 50x2+25x3 $\leq$ 2000
B	x1	x2	x3	x4	z	170x1 + 230x2 + 100x3 = z 20x1 + 30x2 = 10000 14x1 + 12x2 + 30x3 $\leq$ 10000
C	x1	x2	x3	x4	z	290x2+160x3+200x4 = z 30x2 + 10x3 + 25x4 $\leq$ 6500 35x2 + 16x3 $\leq$ 2000
D	x1	x2	x3	x4	z	230x1+150x2+190X3+350x4=z 25x1 + 18x2 + 23x3 + 9x4 $\leq$ 2300 36x1 + 10x2 + 20x3 + 5x4 = 4200 75x2 + 25x3 $\leq$ 3800

There are some very natural questions that one may ask considering the above data. For example, an investor may want to know what is the maximum amount of total profit that the company can produce in a particular city or in all the cities. The CEO of the company may want to expand the company, and he/she would like to know the location of the most profitable company plant? The above problem can be considerably more complicated in real life examples. For example, we would have to

consider taxes, tariffs, labor wages, costs of supplies and plant facilities, plant storage capacity, etc. However, the above example already illustrates, we believe, the main benefits of the query language extension that we propose in this paper and that we implemented in the MLPQ constraint database system. The chief advantage is the ability to use aggregate operators, which is needed in expressing both the investor's or the CEO's query,

To answer the previously mentioned investor's query, we need to find the maximum profit Max(profit) for each plant first, then we will have sum up all the maximum values to find the maximum of total profits. We construct a constraint SQL as follows:

```
SELECT    Sum(Profit)
FROM      Food
WHERE     Profit = (SELECT Max(Profit)
                    FROM Food)
```

The CEO's query is similar to the investor's query. The only difference it that the query should return the city name associated with the maximum profit found in above query. Thus, the CEO's query can be written as follows:

```
SELECT    City
FROM      Food
WHERE     Profit = (SELECT Max(Profit)
                    FROM Food)
```

The food relation provides a perfect example of linear programming type of the problems in a constraint database domain. To solve such a problem we need the linear programming technique There is currently no database system solving such a

problem.

2.2 The Motivation for Databases in Linear Programming

In the previous section, I have presented a motivation example for linear programming in LCDBs. In this section, I will present the motivation for extending LCDBs in the linear programming domain. Let us consider a real world example. The following problem is a simplified version Of a management problem for the street maintenance department in the city of Lincoln. It is a typical linear programming problem. There are currently three facilities f1, f2, and f3 in Lincoln offering three maintenance services s1, s2, and s3. Each service will be performed by a certain number of crews $(x_1, \ldots, x_9)$ from the three facilities and each facility has its own capacity for stationing a number of crews. Facility f1 has a capacity of 3 crews, f2 has a capacity of 3 crews and f3 has a capacity of 2. Service s1 is performed by 3 crews, s2 is performed by 2 crews and s3 is performed by 3 crews. There is a service cost associated with each service performed from each facility. The number of crews from a facility to perform a service needs to be decided. The aim is to find the minimum service cost based on the facility and service constraints. Below is a table representation of the problem:

Maintenance

Weight	s1	s2	s3	constraint
f1	x_1	x_2	x_3	≤ 3
f2	x_4	x_5	x_6	≤ 3
f3	x_7	x_8	x_9	≤ 2
constraint	=3	=2	=3	

In the maintenance table, $x_1, \ldots, x_9$ are variables representing number of crews from a facility to perform a service. Due to different travel distances and other factors, costs for crews performing s1, s2 and s3 are different among the three facilities. By introducing cost data associated with crew variables, $x_1, \ldots, x_9$ we can obtain an LP problem.

Minimize

$$z = x_1 + x_2 + 1.1x_3 + 1.1x_4 + 1.2x_5 + 1.2x_6 + x_7 + 1.3x_8 + x_9$$

subject to the constraints:

$$x_1 + x_2 + x_3 \leq 3$$

$$x_4 + x_5 + x_6 \leq 3$$

$$x_7 + x_8 + x_9 \leq 2$$

$$x_1 + x_4 + x_7 = 3$$

$$x_2 + x_5 + x_8 = 2$$

$$x_3 + x_6 + x_9 = 3$$

$$x_1, \ldots, x_9 \geq 0$$

This linear programming problem can also be presented in constraint databases. The advantage of doing so will be shown later on. For instance, we may have a facility-constraint relation to present the number of crews that can be provided from each facility.

Facility-Constraint

Facility	F1-S1	F1-S2	F1-S3	F2-S1	F2-S2	F2-S3	F3-S1	F3-S2	F3-S3	
f1	x1	x2	x3	x4	x5	x6	x7	x8	x9	x1+x2+x3≤3 x4, ..., x9 = 0
f2	x1	x2	x3	x4	x5	x6	x7	x8	x9	x4+x5+x6≤3 x1,x2,x3=0 x7,x8,x9 = 0
f3	x1	x2	x3	x4	x5	x6	x7	x8	x9	x7+x8+x9≤2 x1, ..., x6 = 0

The attribute F1-S1 means the number of crew from facility f1 to perform a service s1, and same for the Other attributes. Similarly, a service-constraint relation gives us the demand constraint for each service.

Service-Constraint

Service	F1-S1	F1-S2	F1-S3	F2-S1	F2-S2	F2-S3	F3-S1	F3-S2	F3-S3	
s1	x1	x2	x3	x4	x5	x6	x7	x8	x9	x1+x4+x7≤3 x2,x3,x5,x6,x8,x9 = 0
s2	x1	x2	x3	x4	x5	x6	x7	x8	x9	x2+x5+x8≤2 x1,x3,x4,x6,x7,x9 = 0
s3	x1	x2	x3	x4	x5	x6	x7	x8	x9	x3+x6+x9≤3 x1,x2,x4,x5,x7,x8 = 0

A cost relation presents a way to calculate the total service cost based on the cost coefficient associated and the corresponding of crew variables.

Maintenance-Cost

F1-S1	F1-S2	F1-S3	F2-S1	F2-S2	F2-S3	F3-S1	F3-S2	F3-S3	Cost	
x1	x2	x3	x4	x5	x6	x7	x8	x9	z	z=x1+x2+1.1x3+ 1.1x4+1.2x5+1.2x6 +x7+1.3x8+x9

Now, we may use constraint SQL to construct a query to find the minimal value of cost when we use all three facilities to perform the three services such that

SELECT Min(cost)
FROM Facility-Constraint, Service-Constraint, Maintenance-Cost
WHERE Service=s1
AND Service=s2
AND Service=s2
AND Service=s3
AND Facility=f1
AND Facility=f2
AND Facility=f3

This database presentation is equivalent to the LP formulation discussed previously. However, since we present the above maintenance management problem in linear constraint databases, we can construct many more queries. For instance, we may want to eliminate one facility to increase economy of scale. We can easily construct a different constraint SQL query which only includes two facilities. Or, later on the city may decide to contract a service to outside source, the maintenance department will only provide two services. In that case, the constraint SQL query can be constructed that only two services are included. Hence, we turn the original single objective linear programming problem into a wide range linear programming problems

This example shows that linear constraints databases can be a powerful extension of the traditional linear programming techniques.

2.3 Combining Constraint Databases and Linear Programming

The previous two sections stress the importance and advantage of combining linear programming and linear constraint database. Although combining these two domain seems important in applications, the area has not been seriously investigated. Recent linear constraint database research focuses on developing strategies for elimination and simplification of redundant linear arithmetic constraints [9], query optimization [3], and evaluation efficiency [5]. Although Brodsky et al. [3] consider some SQL like queries for LCDBs, they do not consider aggregate operators within the query language. This turns out to be an important issue, which can be solved very naturally by calling upon existing techniques in the area of linear programming.

Chapter 3

Constraint SQL Query Language in the MLPQ System

3.1 Constraint SQL

As discussed by Brodsky et al. [3], the constraint tuples of LCDBs have the form:

$(x_1,...,x_n)$ WHERE Con

where the x_i's are variables or constants and Con is a set of constraints associated with the variables x_i's. Con will take a form:

$a_1x_1 + ... + a_nx_n\theta p$

where a_i and p are constants, θ is an operator such as $=, <, >, \leq, \geq$, and the x_i are variables larger than or equal to zero. The constraint SQL queries in LCDBs can be generally presented as follows.

SELECT $x_1, ..., x_n$

FROM $B_1, ..., B_j$

WHERE Con1

OR

WHERE $C_1, ..., C_j$

WHERE Con2

OR ...

where the set $\{x_1, ..., x_n\}$ is a subset of $\{x_1, ..., x_n, ..., x_m\}$ for relations $B_1, ..., B_j$ and $C_1, ..., C_j$. This query is to select a number of tuples of a set of variables of $(x_1, ..., x_n)$ that occur in relations B'S and satisfy Con1, or occur in relations C's and satisfy Con2, and so on.

Implementation of basic SQL operators such as selection, projection and join in LCDBs are described by Brodsky et al. [3]. The constraint SQL for those basic operators are as follow.

For a selection, the query of constraint SQL will be of the form

SELECT $x_1, ..., x_n$
FROM B
WHERE Con

where relation B and Con have variables $(x_1, ..., x_i, ..., x_n)$. Each tuple of variables $(x_1, ..., x_n)$ is selected by taking a conjunction of a constraint tuple from B and Con, testing whether it is satisfiable.

For a simple projection, the query is of the form:

SELECT $x_1, ..., x_n$
FROM B$(x_1, ..., x_n, ..., x_m)$

where relation B has variables of $(x_1, ..., x_n, ..., x_m)$. The projection will involve the elimination Of variables $(x_{n+1}, ..., x_m)$.

For a constraint SQL join, the query can be constructed as follows.

SELECT $x_1, ..., x_i, ..., x_j, ..., x_n$

FROM B,C

where relation B has variables $(x_1, ..., x_j)$ and relation C has variables $(x_i, ..., x_n)$. In the join query, each tuple is computed from conjunction of a tuple from B and a tuple from C.

3.2 The Query Language of MLPQ

The aggregator operators in constraint SQL of LCDBs are not mentioned by Brodsky et al. The query language in MLPQ is constructed to include those aggregate operators such as *Mar*, *Min*, *Avg*, and *Sum*. The General format of MLPQ query language may be expressed as follows.

SELECT opt(z)

FROM A

WHERE z = (SELECT opt(z)

FROM A)

where *opt* is an aggregate operator such as *Mar*, *Min*, *Avg*, or *Sum*, A is a relation that contains a set of variables $\{x_1, ..., x_i, z\}$, and z represents the objective function. The above query selects an objective value of z from a relation A. The value of z may be obtained by satisfying a number of constraints stored in a linear constraint database. To find the objective value of z satisfying each set of constraints, we need some optimization technique. Here, we use a linear programming method to find such an optimum value of the objective function.

3.3 The Constraint Databases of MLPQ

The linear constraint database Of MLPQ is basically a number Of linear programs stored in a predefined format. The LCDB will treat each linear programming problem as a tuple Of the database. The LCDB of MLPQ can also store some attribute information which do not involving with linear arithmetic computation. These attributes are equally important since they will be used to distinguish different tuples. The constraint SQL aggregator operators are performed for those tuples. The LCDB of MLPQ will connect with LP solver through an interface. The format of storing linear constraint data can be improved late on to enable some basic constraint SQL queries.

Chapter 4

Implementation of the MLPQ System

4.1 Linear Programming Method in the MLPQ

As described previously, the optimization of objective value in MLPQ is implemented using a linear programming technique. The linear programming method used in the MLPQ is the SIMPLEX method. Although the SIMPLEX method may not be suitable for linear programming problems with a large amount of variables, it should be good enough for dealing with a normal constraint database query. In fact, in a linear constraint database System, we may not require to solve for huge amounts of variables. Often we are only interested in finding solutions for relative small scale LP problems and performing necessary queries, Thus the SIMPLEX method is adequate to serve as a computational mechanism for n linear constraint database.

There are various source codes available for the SIMPLEX method. Most of them are written in FORTRAN programming language. The source code used in the MLPQ system is obtained from Press et al.'s book "Numerical Recipe [10]." The source code is written in C which include simp1.c, simp2.c, simp3.c, and simplx.c . I have written an LP computation routine called compute() to call those SIMPLEX programs to optimize (maximize or minimize) the objective function of each LP problem in the linear constraint database.

4.2 Interface of LP with LCDB in the MLPQ System

To use linear programming method in order to solve a constraint database query, we need a good interface Of the two, First one needs to define the syntax of a linear constraint database which contains multiple linear programming problems. Second, an interface program needs to be written so the linear programming solver will recognize the input of the database.

The syntax of the MLPQ constrain database is similar to the linear mathematical expressions. Let's use the previously discussed relation Food as an example. The tuples where City = "A" and "B" can be stored as such

%filename.out%

Qmax()

$City=A

O(x1,x2,x3,x4):-max(300,250,100,150)

C(x1,x2,x3,x4):- (15,8,10,15) <3000,

(30,25,5,10) <8000,

(0,50,25,0) <2000)

$City=B

O(x1,x2,x3):-max(170,230,100)

C(x1,x2,x3):-{(20,30,0)=10000,(14,12,30)¡5200}

%end%

where

%filename.out%=output file name.

%end% = end of file.

$ = attribute name.

Qmax(), Qmin(), Qavg(), Qsum() = aggregator operator.

O(...) = objective function.

C(...) = linear constraints.

The data after the "$" sign (e.g. City = A) will Store attributes do not involve with linear arithmetical computation. The variables names in objective function and constraint functions will be include in the bracket after "O" Or "C" (e.g. O(x1,x2,x3,x4), C(x1,x2,x3,x4)). The *max* or *min* in the objective function specifies what kind of linear programming optimization is expected, The linear arithmetical expressions in MLPQ are expressed with taking the variables out (e.g. 15x1+8x2+10x3+5x4 $\leq$ 3000 is equivalent to (15,8,10,5)<3000).

Currently the aggregator operator is stored on the top of the LCDB. The syntax of the aggregator operators are Qmax() for maximum query, Qmin() for minimum query, Qavg() for average query and Qsum() for summation query. The aggregator operator can join other basic constraint SQL to build a complete query language for the MLPQ system.

The syntax of the constraint database can be changed late on in order to store more information, It could also be changed to provide convenience for the join, selection, and projection operations. The current syntax is relative easy to be converted into a matrix format which is required by the SIMPLEX method used as a LP solver.

Since the LP solver only recognize the matrix input, an interface between the linear constraint database and the LP solver need to be created. The interface is programmed in C++ code. program name is mlpq.C. The program will convert the previous discussed linear constraint database format into a matrix and call the function compute() which will calculate the objective function for each individual linear program, For each LP problem, the LP solver will compute the optimized objective function and solutions for all variables. Then, the aggregator operator query will be made on the values cf objective functions.

4.3 Testing Results

A compiled executable file called mlpq is created. It can run on both UNIX and PC platforms. TO run this program on a constraint database input, one only need type in the filename after the mlpq command. The testing results of the MLPQ system are quite encouraging. Let's 100k at the previously discussed Food example, the investor's query is Sum(Max(z)) and the CEO's query is the name of city WHERE Max(z). They both could be conveniently run in the MLPQ system. To satisfy the investor's query on Food, the system will first find the maximum profit at each plant using the SIMPLEX LP solver. The system will compute the values of z's for the plants in the four cities, A, B, C, and D. The results are 63,600.00, 80,667.67, 62,000.00, and 61,577.78, respectively, Then, the MLPQ will perform the query of Sum() which will return the summation of the four values, Here, the maximum of total profit will be 267,844.45, For CEO's query, the MLPQ system will return the name of city where the plant produce the maximum profit. In this case, the CEO query will give us the city name "B" because the plant in this city has the highest profit among all four cities.

It would be more interesting to see how the MLPQ system functions when dealing with large quantity of linear constraint problems. For instance, we may want to know if there are 400 or 1000 LP problems in a linear constraint database, how long it is going to take to perform a query such as *Max*, *Min*, etc. To make such an experiment, we uniformly randomly generate values of coefficients and constants for a four-variable with two-constraint linear programming problem. The linear programming problem can be expressed as follows.

Maximize $z = c1x1 + c2x2 + c3x3 + c4x4$

Subject to $a1x1 + a2x2 + a3x3 + a4x4 \leq q1$

$b1x1 + b2x2 + b3x3 + b4x4 \leq q2$

all subject to $x1, x2, x3, x4 \geq 10$

The ranges of those random generated values are presented in following table.

Range of Random Generated Coefficients

a1 (100,350)	a2 (150,300)	a3 (0,100)	a4 (50,200)
b1 (0,35)	b2 (0,45)	b3 (0,55)	b4 (0,35)
c1 (0,65)	c2 (0,55)	c3 (0,25)	c4 (0,45)
q1 (2000,5000)	q2 (4000,8000)		

Seven linear constraint databases are generated to test the MLPQ system. Each of them includes 50, 100, 200, 300, 400, 700, and 1000 number of LP problems, respectively. The run-time results are reported in the following table.

Run-Time Test

Number of LPs	Time Used (s)
50	2
100	3
200	6
300	9
400	12
700	20
1000	28

Plotting the run-time test results shows that the run-time increases linearly with the number Of LP problems in its input table. It is easy to see that using LP solving mechanism to perform such kind query is extremely efficient.

Chapter 5

Discussion and Conclusion

At the present time, there is no query language that allows solutions Of multiple linear programming problems, although it occurs frequently in practice. There are many algorithms for one single instance of linear programming, but not for multiple ones. The traditional LP data does not allow joining and union, which are common practices in database operations, The MLPQ system is a contribution for the problem of convenient expression and evaluation of these problems. Since it accepts query language, it can be completely integrated with other query capabilities, and it is convenient to take joins, etc. To extend the MLPQ query language to include a join operation will make it looks as follows:

```
SELECT      opt(z)
FROM        A,B
WHERE       z = (SELECT opt(z)
                 FROM A,B)
```

The query will join two relations A and B and then perform an aggregate operator opt on the objective functions. Note how convenient it was here to extend the problem from considering one to four different subclasses of sugars. This would be

simply impossible to do in the LP standard format representation. The current MLPQ system can only perform aggregator operator on a linear constraint database. What the MLPQ system lacks now is the implementation of basic SQL operations such as join, projection, and selection as discussed previously. Those operations are certainly can be implemented in the MLPQ. One approach is to perform those queries on the LCDB relations first to generate some new relations. Then allow LP solver to be interfaced with the newly created new relations. To include those basic SQL operations on top of the aggregator operators will make the MLPQ a more complete linear constraint database system.

Another subject that needs to be studied further is the input format of the linear programming problem. The currently used SIMPLEX LP solver recognize the matrix input format. The most widely used LP solvers take the MPS format as their input format, The MPS format is an old format and is column oriented. It is set up as though you were using punch cards. More importantly, it is not a free format. Some other commercialized codes have their own input formats. If we want to built a more powerful LP algorithm in a constraint database query system, we need put some efforts to make the interface work between the LP algorithm and the constraint database input format.

The research discussed in this project report mainly focuses on the significance of combining linear programming and linear constraint databases technologies. Combining linear constraint database and linear programming techniques will make the LP technique more flexible to accommodate the combination of the variables and variation of queries in wide ranging applications. The traditional linear programming targets to solve single LP problem. Combination of linear programming and linear constraint database will make it more efficient in solving multiple LP problems and making multiple queries. On the other hand, using linear programming in linear con-

straint databases will also strongly enhance capability of a database to handle linear constraint data information.

Bibliography

[1] F. Afrati, S. Cosmadakis, S. Grumbach, and G. Kuper. Linear versus polynomial constraints in database query languages. In A. Borning, editor, *International Workshop on Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*, page 181192. Springer, 1994.

[2] B. Aspvall and R.E. Stone. Khachian's linear programming algorithm. *Journal of Algorithms*, 1(1):113, 1980.

[3] A. Brodsky, J. Jaffar, and M.J. Maher. Towards Practical Constraint Databases. In *International Conference on Very Large Data Bases*, pages 567–580, Dublin, Ireland, 1993.

[4] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[5] S. Grumbach, J. Su, and C. Tollu. Linear constraint query languages: Expressive power and complexity. In D. Leivant, editor, *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 426–46. Springer, 1995.

[6] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

[7] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSR*, 20:191–4, 1979.

[8] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[9] J-L. Lassez, T. Huynh, and K. McAloon. Simplification and elimination of redundant linear arithmetic constraints. In E. L. Lusk and R. A. Overbeek, editors, *Proc. North American Conference on Logic Programming*, pages 35–51. MIT Press, 1989.

[10] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 1992.

[11] P. Z. Revesz. *Introduction to Constraint Databases*. Springer, New York, NY, 2002.

[12] P. Z. Revesz and Y. Li. MLPQ: A linear constraint database system with aggregate operators. In *Proc. 1st International Database Engineering and Applications Symposium*, pages 132–7. IEEE Press, 1997.

[13] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[14] M. Wilkes. *Operational Research Analysis and Applications*. McGraw-Hill, 1989.