MLPQ: A Linear Constraint Database System with Aggregate Operators*

Peter Z. Revesz and Yiming Li
University of Nebraska–Lincoln
Dept. of Computer Science and Engineering
Lincoln, NE 68588, USA
{revesz,yli}@cse.unl.edu

Abstract

This paper describes the MLPQ constraint database system. The query language of MLPQ is SQL extended with linear arithmetic constraints. The input and output databases are Linear Constraint Databases (LCDBs). An important feature of the MLPQ system is that it can handle aggregate operators, Min, Max, Sum, Avg, etc. In MLPQ, these operators are evaluated for a series of linear programming (LP) problems. This approach provides an efficient way of evaluation of SQL queries with aggregate operators on linear constraint databases.

1. Introduction

Relational databases are limited in dealing with problems that involve numerical computations. For instance, consider the following problem. Mr. Johnson who lives in Omaha, Nebraska wants to send three packages to three clients who live in Atlanta, Boston, and Chicago. He wants to know how much he should pay for his packages. The weight of each package can be represented in the following relational database table.

Package

Serial No.	From	Destination	Weight
101	Omaha	Chicago	12.6
102	Omaha	Atlanta	27.3
103	Omaha	Boston	37.5

The postage fee charged for packages could be computed based on the weight of the package and the postage rate associated with the weight. Up to five ounces, the fee is the weight of the package times 0.53, between 5 and 15 ounces the fee is \$2.65 plus 0.45 times the weight over 5 ounces, etc. In other words, the postage rate increases piecewise linearly as the ounces increase. Ideally, we would like to represent the Postage relation as the table:

Postage

Weight	Fee
5	2.65
⋮	⋮
50	16.65

The above representation would be very convenient, because, then we could express Mr. Johnson's query in SQL as follows:

SELECT Sum(Fee)
FROM Package, Postage

Unfortunately, in relational databases we cannot do that. The reason is that Postage is an infinite relation, hence we cannot represent it in relational databases.

Fortunately, constraint databases can rescue the situation. Constraint databases combine relational databases with numerical constraints. Within the tables we can use attribute variables and constraints appended to each tuple. The Postage1 relation can be represented using a constraint table as follows:

Postage1

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

*This work was supported in part by NSF grants IRI-9625055 and IRI-9632871. The second author was also supported by a grant from the Mid-America Transportation Center.

The meaning of the constraint databases is the set of instantiations into the attribute variables of any tuple that make the appended constraints true. Hence, the Postage1 table will represent finitely the same table as Postage represents using an infinite number of tuples.

In constraint databases, it is permissible to use the same SQL queries as in relational databases. Constraint SQL queries are computed and the compuation returns constraint tables as answers, i.e., other tables that may contain attribute variables and constraints. Mr. Johnson's query can now be computed, and it will return a total fee of 27.785 for the three packages.

Although the basic idea of extending relational databases and query languages to constraint databases and query languages was presented by Kanellakis et al. [?] in 1990, the implementation issues behind constraint databases still need to be explored in detail.

In particular, an essential issue that was not dealt with in earlier papers [?, ?, ?, ?] on the topic of linear constraint databases (LCDBs for short) is how to implement efficiently the aggregate operators of SQL, in particular the maximum Max, minimum Min, and sum Sum aggregate operators.

[?, ?] considered the efficient implementation of relational algebra operators select, project, join, intersection, and cross product. [?] considered query optimization for SQL without the aggregate operators. [?] studied strategies for variable elimination and simplification of redundant linear arithmetic constraints. [?] compared the expressive power of relational algebra queries over linear vs. polynomial constraint databases. None of these papers consider aggregate operators within the query language.

In this paper we present a solution to the problem of computing aggregate operators. The solution is based on efficient reduction of each aggregate operation to a series of linear programming instances. We describe the ideas behind the evaluation method. We also present running time results for our prototype implementation of SQL queries over LCDBs. The running time results show that the speed of the system is satisfying even for queries that reduce to hundreds of linear programming problems.

We call the system MLPQ (an abbreviation for multiple linear programming queries). We hope that MLPQ will be very useful for a number of application domains. Basically, it is surprising that while there are numerous packages available for linear programming (based on either the SIMPLEX method, or more recent methods such as the ellipsoid method) we found no packages that are concerned with efficient solving of a series of closely related linear programming problems. It is even difficult to express such problems, even though they occur naturally as our examples will illustrate later in the paper. MLPQ will provide a easy way of expressing such problems with all the conveniences of SQL. For example, with MLPQ it is easy to reformulate problems using joins and more complex queries. It would be very hard to reformulate problems using the standard input format for linear programming problems.

This paper is organized as follows. In Section 2 we describe a complex motivating example and the syntax of constraint SQL. In Section 3 we describe the implementation of MLPQ, (in section 3.1 for the relational algebra operators, and in section 3.2 for the aggregate operators). In Section 4 we present running time results for MLPQ queries. In Section 5 we discuss further possible extensions of the MLPQ system.

2. Constraint SQL

2.1. A motivating example for constraint SQL

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 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 table called Food. In this table, attributes chocolate-bar, ice cream, sugar, milk, and chocolate are abbreviated as C-B, I-C, S, M, and C, respectively.

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 all the cities. The CEO of the company may want to expand the company, and he/she would like to know where is 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 the main benefits of constraint databases and constraint SQL queries that we propose in this paper and that we implemented in the MLPQ constraint database system.

City	Candy	C-B	I-S	Yogurt	S	M	C	Profit	
A	x_1	x_2	x_3	x_4	s	m	c	z	$300x_1 + 250x_2 + 100x_3 + 150x_4 = z$ $15x_1 + 8x_2 + 10x_3 + 15x_4 \leq s$ $30x_1 + 25x_2 + 5x_3 + 10x_4 \leq m$ $50x_2 + 25x_3 \leq c$ $s = 3000, m = 8000, c = 2000$
B	x_1	x_2	x_3	x_4	s	m	c	z	$170x_1 + 230x_2 + 100x_3 = z$ $20x_1 + 30x_2 \leq s$ $14x_1 + 12x_2 + 30x_3 \leq m$ $s = 10000, m = 5200$
C	x_1	x_2	x_3	x_4	s	m	c	z	$290x_2 + 160x_3 + 200x_4 = z$ $30x_2 + 10x_3 + 25x_4 \leq m$ $35x_2 + 16x_3 \leq c$ $m = 6500, c = 2000$
D	x_1	x_2	x_3	x_4	s	m	c	z	$230x_1 + 150x_2 + 190x_3 + 350x_4 = z$ $25x_1 + 18x_2 + 23x_3 + 9x_4 \leq s$ $36x_1 + 10x_2 + 20x_3 + 5x_4 \leq m$ $75x_2 + 25x_3 \leq c$ $s = 2300, m = 4200, c = 3800$

Figure 1. The Food relation

The chief advantage is the ability to use aggregate operators, which is needed in expressing both the investor's or the CEO's query.

For example let's try to express the investor's query mentioned above. Here we need to find the maximum profit for each plant first, then we have to find the sum of the maximums. We may use constraint SQL to write the query as follows:

```
SELECT   Sum(Profit)
FROM     Food
WHERE    Profit IN     (SELECT max(Profit)
                       FROM Food)
```

Let's try now to express the CEO's query. The query will be similar to investor's query. The only difference is 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 IN   (SELECT Max(Profit)
                     FROM Food)
```

2.2. A formal definition of constraint SQL

A basic constraint SQL query is of the form

SELECT $a_{i_1}, ..., a_{i_l}$
FROM $R_1, R_2, \ldots, R_m$
WHERE $Con_1, \ldots, Con_k$

Here the Rs are relation names, as are attribute names, and Cons are atomic constraints. Suppose that the set of attribute names appearing in the relations $R_1, \ldots, R_m$ is $S = \{a_1, \ldots, a_n\}$. The set of attribute names appearing in the SELECT line is some subset $S_i = \{a_{i_1}, \ldots, a_{i_l}\}$ of S. Each atomic constraint is of the form

$$c_1 a_1 + \ldots + c_l a_n \Theta b$$

where cs and b are rational constants and some of them may be 0, and Θ is one of the comparison operators $=, <, \leq, >, \geq$.

Constraint SQL also allows the aggregate operators Max, Min, Avg, and Sum within the SELECT line. The Max and the operators are allowed on linear constraint constraint databases while the Avg and Sum operators are allowed only on relational databases. (Every relational database is a LCDB, but not every LCDB is a relational database. This is because an infinite number of tuples cannot be represented in a relational database.) Hence a basic constraint SQL query with aggregation looks like the following:

SELECT $a_{i_1}, ..., a_{i_l}, opt(f(a_1, \ldots, a_n)$
FROM $R_1, R_2, \ldots, R_m$
WHERE $Con_1, \ldots, Con_k$

where opt is one of Max, Min, Avg, Sum with the restriction mentioned above, and f is a linear constraint of the form $c_1 a_1 + \ldots + c_n a_n$. The function f is called an objective function.

Basic SQL queries can be combined together to form bigger queries. Two types of combinations are allowed within the MLPQ system. The first combination is the $Union$ operation which has the form:

BASIC Constraint SQL Query Q_1
UNION
BASIC Constraint SQL Query Q_2

where the set of attribute names in the SELECT lines of Q_1 and Q_2 the same.

The second combination allowed is more interesting. It looks as follows:

SELECT $opt(f_2(...))$
FROM $R'_1, \ldots, R'_{m'}$
WHERE a_i IN (SELECT $opt(f_1(...))$ FROM $R_1, \ldots, R_m$)

In the above, the result of the inner SQL query is a LCDB with one attribute. The relation names R_i in the inner and R'_j in the outer basic SQL query may be equal. The aggregation in the inner and the outer basic SQL query is optional, it could be instead an attribute value.

3. Implementation Issues

3.1. Implementation of basic constraint SQL queries

Each basic SQL query can be translated to a relational algebra expression on LCDBs involving the project π, select σ and Cartesian product $\times$ operators as follows:

$$\pi_{a_{i_1},\ldots,a_{i_l}}\ (\sigma_{Con_1}(\ldots(\sigma_{Con_k}\ (R_1 \times R_2 \times \ldots \times R_m))\ldots))$$

The basic relational algebra operators on LCDBs are implemented as described by Grumbach and Lacroix [?]. Let us assume that each R_i has the following form:

$a_{i,1}$	$\ldots$	a_{i,n_i}	
$R_i.x_1$	$\ldots$	$R_i.x_{n_i}$	$Con_{i,1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$R_i.x_1$	$\ldots$	$R_i.x_{n_i}$	Con_{i,m_i}

Then the cross product of R_i and R_j will be the following:

$R_i.a_{i,1}$	$\ldots$	$R_j.a_{j,n_j}$	
$R_i.x_1$	$\ldots$	$R_j.x_{n_j}$	$Con_{i,1}, Con_{j,1}$
$R_i.x_1$	$\ldots$	$R_j.x_{n_j}$	$Con_{i,1}, Con_{j,2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$R_i.x_1$	$\ldots$	$R_j.x_{n_j}$	Con_{i,m_1}, Con_{j,m_j}

Note that $R_1.x_1$ and $R_2.x_1$ are two different variables. (If we did not have that, then we would have a join operation instead of a cross product.)

The selection $\sigma_{Con}R_i$ will yield the table:

$a_{i,1}$	$\ldots$	a_{i,n_i}	
$R_i.x_1$	$\ldots$	$R_i.x_{n_i}$	$Con_{i,1}, Con$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$R_i.x_1$	$\ldots$	$R_i.x_{n_i}$	Con_{i,m_i}, Con

Note that the selection condition Con is conjoined with the selection condition within each constraint tuple. Some of the constraint tuples become unsatisfiable. The satisfiability of constraint tuples is checked in MLPQ and the unsatisfiable ones are removed.

The projection $\pi_{a_{i_1},\ldots,a_{i_l}}R_j$ is implemented using Fourier-Motzkin elimination. We delete from R_i each column that is headed with an attribute name a that does not occur in the index of the projection operation. Then we eliminate $R_j.a$ from each constraint in the rightmost column. We do the variable elimination one by one for each variable that does not occur in the selection condition.

3.2. Implementation of aggregate operators

Linear programming is widely used in engineering, management, planning and economics to optimize (minimize or maximize) single or multiple objective functions based on given linear constraints [?]. The LP problem can be presented in its standard format such that

Minimize $z = \overline{c}\,\overline{x}$

Subject to $A\overline{x} = \overline{b}$

All subject to $\overline{x} \geq 0$

where z is an objective function, $\overline{x}$ is the vector of variables that need to be solved, A is the coefficient matrix, and $\overline{b}$ and $\overline{c}$ are known vectors of constants.

Finding the maximum or minimum of an objective function over a single constraint tuple is easily expressible as a LP problem. Finding the maximum or minimum of an objective function over a relation with n constraint tuples can be reduced to n separate LP problems followed by finding the maximum or minimum of the values of the objective function returned by the n separate LP problems. Alternatively, the sum or the average of the values can be also taken.

In MLPQ linear programming is implemented based on the SIMPLEX method developed in the 1940s. The method is very efficient and uses only basic arithmetical operations [?]. The theory behind the SIMPLEX method is that only the corner points of the feasible region can be unique 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 a better solution, then it moves to it. It repeats the procedure until no improvement can be found.

A family of alternative LP techniques called Interior-Point methods have been developed in the late 1980s. These are non-linear programming techniques that can be used to solve many large scale linear programming problems. Interior-Point methods are different from the SIMPLEX method in that they always stays within the boundaries of the solution space. We could have also used Interior-Point based algorithms, branch-and-bound methods [?], or the polynomial time algorithm developed by Hacijan in 1979 [?] or other methods. However, since the number of variables are small in normal constraint SQL queries and therefore also in each LP problem that need to be solved, the SIMPLEX method works well. This will be described in the next section.

3.3. Implementation of complex constraint SQL queries

The UNION operation takes the union of all constraint tuples in the two input relations. We did not implement at present other operators similar to the UNION operator, such as INTERSECTION and DIFFERENCE, but they can be also implemented as described in [?, ?]. Nested SQL queries were implemented by evaluating the innermost and then the outer constraint SQL queries.

4. Testing Results

Our testing results were very encouraging. We could run the investor's and the CEO's queries quite conveniently in the MLPQ system. For instance, we run in MLPQ the investor's query on the input database presented in table Food. The MLPQ system found first the maximum profits that each plant can achieve using the SIMPLEX method. MLPQ found the values of the z's for the plants in the four cities, A B, C, and D to be 63,600.00, 80,667.67, 62,000.00, and 61,577.78, respectively. MLPQ then summed up these four values and returned the maximum of total profit value which was 267,844.45. For the CEO's query, the MLPQ system returned the name of city B which is correct because the plant in city B could produce the highest maximum profit.

It is more interesting to see how the MLPQ system functions when dealing with larger LCDBs. Therefore we made some computer experiments. We generated a Food LCDB relation with 50, 100, 200, 300, 400, 700, and 1000 number of constraint tuples. The coefficients of the four variables x_1, x_2, x_3, and x_4 for the objective function as well as for the four linear constraints in the constraints tuple were generated using a uniformly random distribution.

We then ran the investor's query on the randomly generated databases. and measured the running time of the query which included input and output time. 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	27

The run-time increases linearly with the number of LP problems in its input table. Further, the table demonstrates that the MLPQ system can be extremely efficient in performing simple constraint SQL queries.

5. Discussions 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 MLPQ system is a contribution for the problem of convenient expression and evaluation of these problems. Since it is a query language, it can be completely integrated with other query capabilities, and it is convenient to take select, project, join, and other operators that are common even in the simple database queries. This is quite a contrast with linear programming data as it is usually represented (even) today. The traditional LP data input format, called MPS, is close to the punch card format and originated at the same time when punch cards were widely used. As a data storage format it is a far cry from the convenience of relational databases, and constraint databases.

For example, a selection operation may be needed if we are interested only in city A. A join operation may be needed if we get into consideration that instead of sugar, we can substitute several different types

of sugars, for instance cane sugar, corn syrup, beets, molasses, and honey. For instance, the sugar can be substituted by 1 unit of cane sugar, 4 units of beets sugar, 2 units of molasses, and 5 units of honey. Let us assume that in city A the company has on store 300 units of cane sugar, 200 units of beets, 500 units of molasses, and 100 units of honey. We can construct an LCDB relation called Sugar to present this relation as follows:

Sugar

City	Cane	Beet	Molass	Honey	S	
A	y_1	y_2	y_3	y_4	s	$y_1 + 1/4y_2 + 1/2y_3 + 1/5y_4 = s$ $y_1 \leq 300$ $y_2 \leq 200$ $y_3 \leq 500$ $y_4 \leq 100$

Suppose that we would like to find the maximum profit in city A. The entire query, which involves select, join, and aggregate operations can be written as:

SELECT Max(Profit)
FROM Food, Sugar
WHERE City = "A" and Food.S = Sugar.S

Note that the entire query reduces to a single instance of a linear programming problem. Hence the query seems trivial. However, it is interesting because LP packages do not contain algorithms for select and join. The MLPQ package does provide these operations.

The problem with the SIMPLEX method, and MLPQ currently, is that it does not handle problems in which the variables range over the integer numbers[?]. Therefore, in a future version of MLPQ we will look at integer programming methods, including cutting-plan algorithms [?].

References

[1] F. Afrati, S. Cosmadakis, S. Grumbach, and G. Kuper, "Linear versus Polynomial Constraints in Database Query Languages," In A. Borning editor, Proc. 2nd International Workshop on Principles and Practice of Constraint Programming, pp. 181-192, Springer Verlag LNCS 874, 1994.

[2] A. Brodsky, J. Jaffar and M. Maher, "Towards Practical Constraint Databases," In Proc. Conf. on Very Large Databases, Dublin, Ireland, pp. 567-580, 1993

[3] R.E. Gomory, "Outline of an Algorithm for Integer Solutions to Linear Programs," Bulletin of the American Mathematical Society, Vol. 64, pp. 275-278, 1958.

[4] S. Grumbach and Z. Lacroix, "Computing Queries on Linear Constraint Databases", Proc. Conf. on Database Programming Languages, 1995.

[5] S. Grumbach, J. Su, and C. Tollu, "Linear Constraint Database," In D. Leivant, editor, Proc. Workshop on Logic and Computational Complexity, Indianapolis, Springer-Verlag LNCS 960, 1994.

[6] L.G. Hacijan, "A Polynomial Algorithm in Linear Programming," Doklady Akademii Nauk SSSR, Vol. 244, No5, pp. 192-194, 1979.

[7] J. P. Ignizio, Linear Programming in Single and Multiple Objective Systems, Prentice-Hall Inc., pp. 1-17, 1982.

[8] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz, "Constraint Query Language," Journal of Computer and System Sciences, 51(1): 26-52, August, 1995.

[9] A.H. Land, and A.G. Doig, "An Automatic Method of Solving Discrete Programming Problems," Econometrica, Vol. 28, No. 3, pp. 497-520, 1960.

[10] J. L. Lassez, T. Huynh, and K. McAloon, "Simplification and Evaluation of Redundant Linear Arithmetic Constraints," In Proc. North American Conference on Logic Programming, Cleveland, pp. 35-51, 1989.

[11] C.D. Pegden, and J.P. Ignizio, "Khachian's Polynomial Algorithm for Linear Programming," AIIE News: Operations Research, Vol. 14, No. 4, pp. 1-4, 1980.

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

[13] M. Wilkes, Operational Research Analysis and Applications, McGraw-Hill Book Company, pp. 47- 76, 1989.