MLPQ/GIS: A GIS using Linear Constraint Databases*

Pradip Kanjamala, Peter Z. Revesz, Yonghui Wang
Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE 68588, USA
{kpradip,revesz,wangyh}@cse.unl.edu

Abstract: This paper presents MLPQ/GIS, a GIS database system that allows powerful and easy querying via a graphical user interface. The internal data representation of MLPQ/GIS is based on linear constraint databases, which can describe geo-spatio-temporal data, allow efficient conjunctive query evaluation, and facilitate data integration and database interoperability.

Keywords: Geographic information systems, graphical user interface, constraint databases, spatial databases, spatio-temporal databases, linear programming, linear constraints.

1 Introduction

In the Millennium 2000, the World Wide Web will provide increased access to various types of databases all around the world. However, the incompatibility of different data models and formats used at different sites may hamper us in taking full advantage of the World Wide Web. It is, for example, difficult to integrate geographic data stored in an ARC/INFO GIS system [12] with temporal data stored in a TSQL temporal database system. Recently, Chomicki and Revesz [5] proposed to use *constraint databases* [10] as a common basis for various spatial, temporal and GIS data. The main motivation behind that proposal is the recognition that most already proposed spatial, temporal and GIS data models can be viewed as subcases of constraint databases restricted to an appropriate type of constraints.

In this paper we describe a GIS system built on top of the MLPQ constraint database system [14]. MLPQ (short for *Management of Linear Programming Queries*) allows constraint relations with any number of attributes, which may be spatial, temporal etc., to be constrained using linear constraints over the rational numbers. MLPQ was not built for geographic data and had no graphical user interface. The extension MLPQ/GIS is intended for easy, convenient geo-spatio-temporal data querying using a graphical user interface.

The potential of using linear constraint databases for GIS was recognized by several other researchers, (eg., Brodsky et al. [3], Frank and Wallace [7], Paredaens et al. [13], Grumbach et al. [8]). The MLPQ/GIS system has a unique implementation strategy that is based on the MLPQ system and goes beyond other proposals in integrating aggregate functions into the system that were not considered by other systems in current development (Section 3 will give examples).

The goal of our system was to build a graphical user interface GUI for MLPQ that allows users to do more and easier GIS queries than in other systems. We distinguish between beginning users

*This work was supported by NSF grant IRI-9625055.

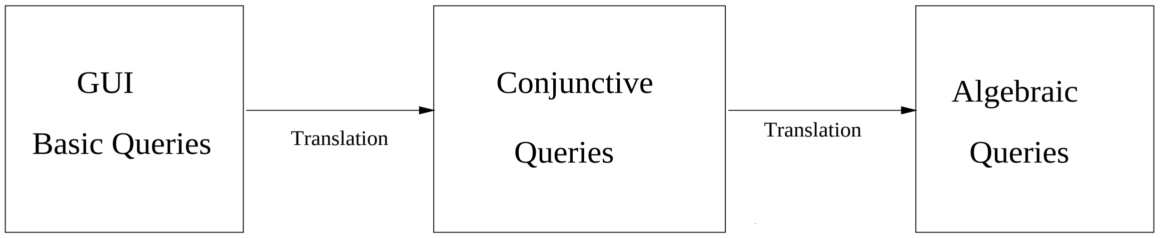


Figure 1: MLPQ/GIS Query Evaluation

and advanced users. For beginning users, a purely icon-based querying is provided. We call this basic queries. For advanced users a more general querying using conjunctive queries [16] is also allowed. As shown in Figure 1 in MLPQ/GIS the basic queries are translated into conjunctive queries. Conjunctive queries are translated into a procedural algebraic language, which is optimized using algebraic optimization and then evaluated based on efficient implementations of constraint database versions of selection, projection, join, union, and intersection operators.

The rest of the paper is organized as follows. Section 2 deals with how GIS data is input to the system. Section 3.1 describes basic queries, and Section 3.2 describes general queries. Section 4 compares this system with some other GIS systems currently being used. Section 5 discusses further implementations in the future and how data interoperability facilities will be added to MLPQ/GIS.

2 Entering Data

In MLPQ/GIS the data is stored in a constraint database format [10], which facilitates efficient data manipulation using linear constraint solving, including linear programming [2, 4, 6, 9, 11, 15].

It is however important to mention that the user need not be concerned with the constraint data storage, but only with its visual display. This is because the user has many options of entering into MLPQ/GIS a linear constraint database indirectly. Below we list some of these options.

Drawing: The system allows the end user to draw pieces of land in the form of lines, rectangles or general polygons. The first four icons in Figure 2 shows the drawing tools that the system provides.

Buying: Much of the data of the GIS could come from a data provider. For example, the data provider could provide the blue print of a house or data about land use in the required form of a constraint database.

Image Scanning: Certain image data can be scanned in and automatically translated into a linear constraint database.

Translation: By calling built-in translation routines (which are currently being implemented) data provided in other data formats, such as an ARC/INFO GIS database, will be converted into to a consraint database.

Constraint Database: The user would be able to specify directly linear constraints on the data. This method can be used if the other methods fail.

Any way the data is entered, the system will store a constraint database representation of the data entered. We give a few examples of constraint databases used in the internal representation of MLPQ/GIS.

Example 2.1 A piece of land is sprayed with pesticide and this information is provided. The data comes in the form of a two relations, *Pesticide_A* and *Pesticide_B* with the following attributes: *id*, x and y. For example, the 37th piece of land sprayed with pesticide A can be represented in the linear constraint database as:

$$\begin{aligned} pesticide\_A(37,x,y) \quad :- \quad & x - y \geq -18, \\ & 2x + 3y \leq 104, \\ & 4x - 7y \leq -78, \\ & x + 2y \leq 48. \end{aligned}$$

Each data item is provided with an id. This id serves as an index in the relational database in which the non-spatial attributes are stored.

Example 2.2 A data provider has information about ownership of a piece of land. This ownership information could be stored in a linear constraint database as a relation *Land* with the following attributes: *id, x, y, t* and *owner*. For example, the 5th piece of land as well as the information that it belongs to Adam between 1980 and 1990 can be represented by the following constraint database tuple:

$$\begin{aligned} Land(5,x,y,t,\text{"}Adam\text{"}) \quad :- \quad & x \geq 0, \\ & y \geq 0, \\ & y \leq 5, \\ & 5x - 3y \leq 10, \\ & 1.67x + 2y \leq 13.34, \\ & t \leq 1990, \\ & t > 1980. \end{aligned}$$

MLPQ/GIS uses linear constraint databases as the input representation allowing expression of any linear relationships between the spatial and the temporal attributes. In addition, linear constraint databases also have an advantage over traditional relational databases in being able to represent incomplete information with ease. Example 2.3 illustrates how incomplete information could be stored in a linear constraint database.

Example 2.3 Let us assume a plane departing from $(0,0)$ would go east between 490 to 580 miles, and go north between 400 to 440 miles in one hour. Another plane's departure location is not certain, but it is known that it could take off from some location between a 1000 to 1120 miles of west and between 690 and 750 north of the first plane. After one hour, it will go west at least 590 miles and at most 650 miles. It will also go south between 300 miles to 340 miles. Three relations are used to represent this incomplete information in constraint format. Note that there is no way this kind of information could be stored and used in a traditional relational database in a format that is convenient to query. The relation *Begin* has three attributes, namely *id*, x and y to store information about the beginning location of the plane. For example the initial location of the second plane could be represented as:

$$\begin{aligned} Begin(2,x,y) \quad :- \quad & x \geq 1000, x \leq 1120, \\ & y \geq 690, y \leq 750. \end{aligned}$$

Relation *Plane2* represents the possible final destination of the second plane using attributes *id*, x and y. For example, expressing the location of the second plane in constraint format would look like:

$$\begin{aligned} Plane2(2,x,y) \quad :- \quad & 590 \leq x - x0, x - x0 \leq 650, \\ & 300 \leq y - y0, y - y0 \leq 340, \\ & Begin(2, x0, y0). \end{aligned}$$

3 Querying in MLPQ/GIS

The queries that the system supports can be broadly classified into two types, namely *basic queries* for beginning users and *general queries* for intermediate to advanced users.

3.1 Basic Queries

Basic queries are those queries that can be specified using only the icons available in the GUI. Some of these queries may need some parameters to be specified, and these parameters are usually specified through an input dialog box that prompts the user with simple questions when necessary. The GUI contains the following icons for querying purposes:

Intersection Dialog box prompts for the name of the new relation generated that will contain the intersection of the relations selected using the mouse.

Union. Similar to "Intersection".

Area Dialog box prompts user for the name of the new relation generated, the left and the right bound and the step sizes. For example, for a 100×100 area picture, a specification of 0 for the the left and 100 for the right bound and 50 for the step size finds the total area falling within the left and the total area falling within the right half of the picture for the selected relation. A finer step size is useful for getting various aggregate information for different bands of land.

Buffer Dialog box prompts user for the name of the new relation generated, which contains the surrounding region of a selected spatial object. The dialog box also prompts for the distance d. Any point in the buffer region will be $\leq d$ distance from at least one point of the spatial object.

Max Dialog box prompts user for the name of the new relation generated, the objective function to be maximized, and a constraint relation.

Min Similar to "Max".

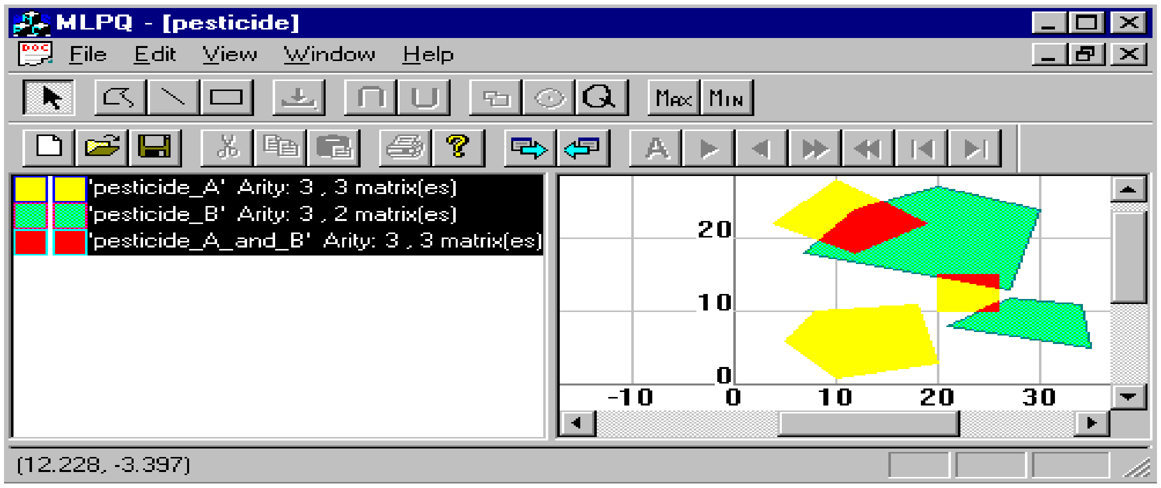
We illustrate the use of these icons for some basic querying in the following examples.

3.1.1 Pesticide Example:

Suppose we have a database as in Example 2.1, with information about a farmland in which two types of pesticides, say A and B have been applied.

Query: Find the regions in which both pesticides were applied and the area of that region.

To implement the query using the GUI interface, we select the relations *pesticide_A* and *pesticide_B* and then click on the intersection icon. A dialog box would appear in which the user could enter the name of the new relation that is to be generated. By typing in "*pesticide_A_and_B*", a new relation with that name is created. The area of the region could be computed by selecting the relation *pesticide_A_and_B* and then clicking the area icon. Like before, a dialog box pops up in which the name of the new relation could be entered. The computed value could be seen by inspecting the new relation generated.



🟨 🟨 'pesticide_A' Arity: 3 , 3 matrix(es)
🟩 🟩 'pesticide_B' Arity: 3 , 2 matrix(es)
🟥 🟥 'pesticide_A_and_B' Arity: 3 , 3 matrix(es)

(12.228, -3.397)

Figure 2: Pesticide Example

3.1.2 Tracking Problem

A person travelling through the US mid-west finds his current locations using an up-link to a satellite that beams to him his current X and Y coordinates. His current location is then put in a linear constraint database. The database also has information about the mid-western part of the United States.

Query: Which state is he currently in?

Suppose that the map of each state is stored in a relation with its name (eg. Colorado, Nebraska, etc.) while the current location of the user is stored in the relation *Current_Location*. For answering this query simply click on the state names and then on *Current_Location*. Whatever *Current_Location* contains will be overlayed on the already displayed states. Hence we can conclude by looking at Figure 3 that the person is in Colorado State.

3.1.3 Incomplete Information Query

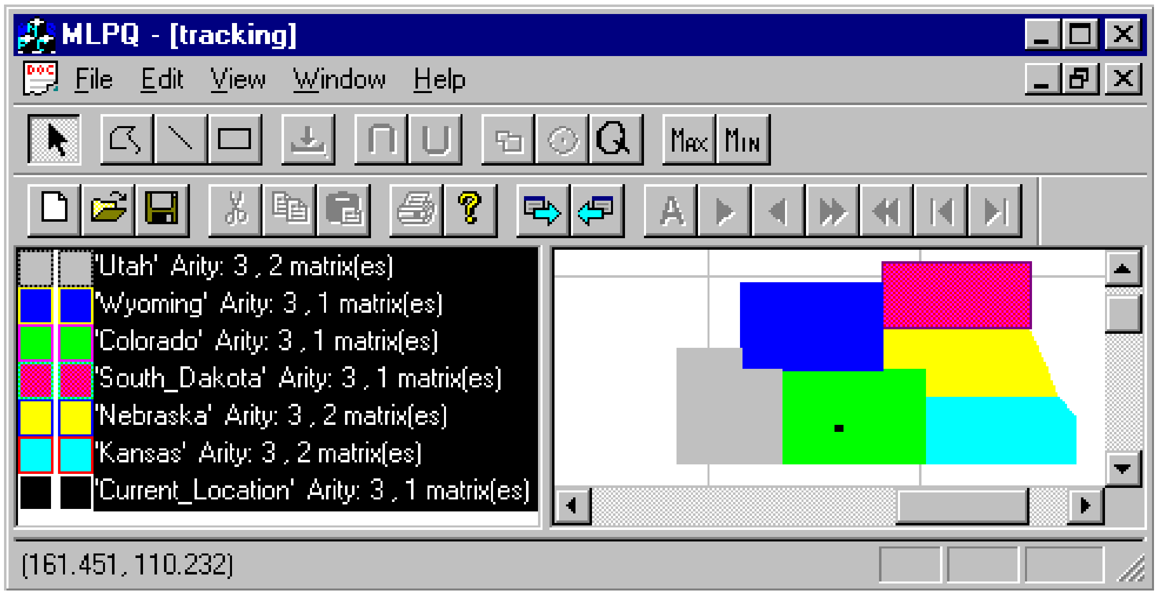
We are given incomplete information about the starting point of two planes and incomplete information about their final position after one hour of flight as given in Example 2.3.

Query: Is it possible that the two plane meet each other after one hour?

The query can be answered by testing whether there could be an intersection between the final positions of the two planes. Since relations *Plane1* and *Plane2* store the information about the possible final locations of the two planes respectively, the query can be answered by selecting these relations and then visually checking whether they intersect. From Figure 4, it is obvious that the two regions intersect, and hence we could say that the planes could meet each other after an hour.

3.1.4 Buffer Query

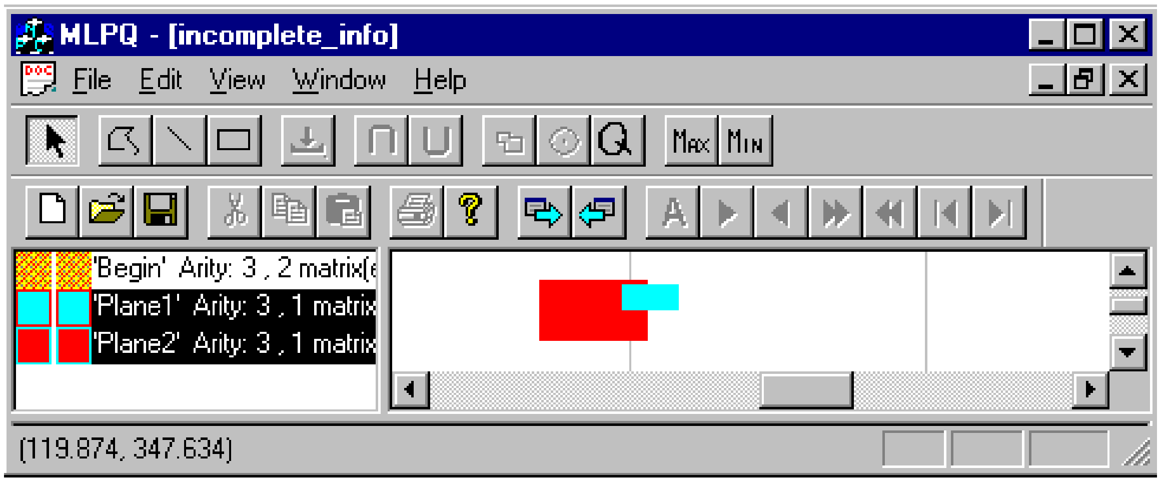
A person is headed westward on a national highway and is using a GPS (Global Point System) to keep track of his direction and surrounding. We assume that the GPS system would provide the Linear Constraint Database the relation *Current_Pos* with attributes x and y. This relation gives



'Utah' Arity: 3 , 2 matrix(es)
'Wyoming' Arity: 3 , 1 matrix(es)
'Colorado' Arity: 3 , 1 matrix(es)
'South_Dakota' Arity: 3 , 1 matrix(es)
'Nebraska' Arity: 3 , 2 matrix(es)
'Kansas' Arity: 3 , 2 matrix(es)
'Current_Location' Arity: 3 , 1 matrix(es)

(161.451, 110.232)

Figure 3: Tracking Example



'Begin' Arity: 3 , 2 matrix(e
'Plane1' Arity: 3 , 1 matrix
'Plane2' Arity: 3 , 1 matrix

(119.874, 347.634)

Figure 4: Incomplete Information query

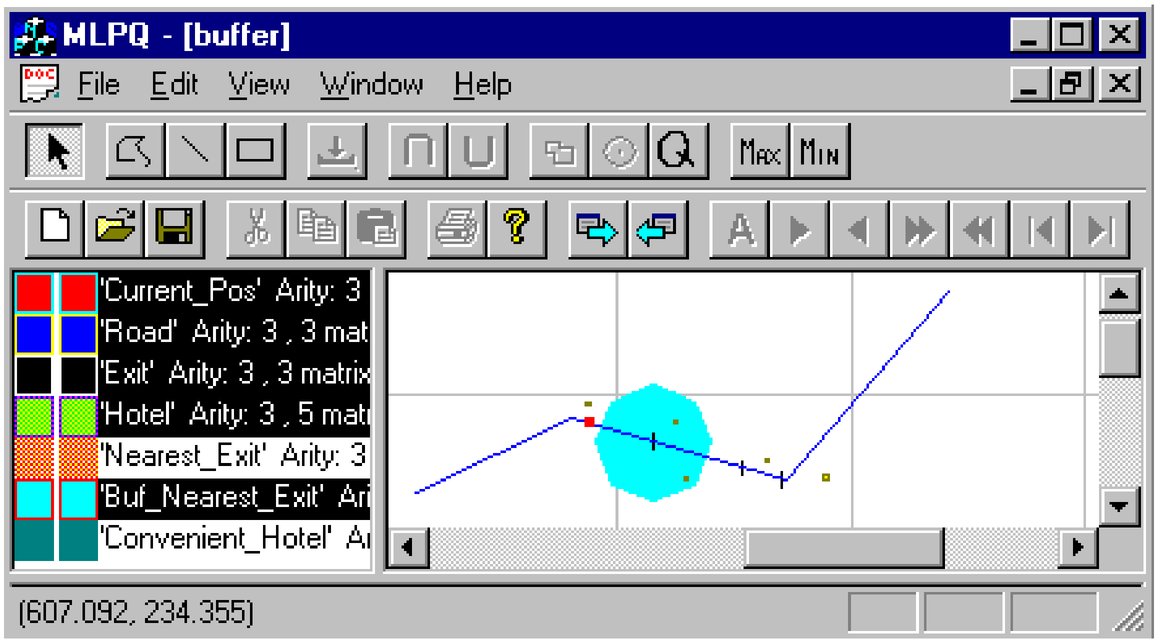


Figure 5: Buffer Example

the current location of the person. Relation *Road* with attributes *highwayId,x* and *y* has information of the highway the person is on while relation *Exit* has information of all the exits on the highways. The relation *Exit* has attributes *exitId,x* and *y*. The relation *Hotel* with attributes *HotelName, x* and *y* stores information of all hotels in the region.

Query: Which is the closest exit on the highway in his direction?

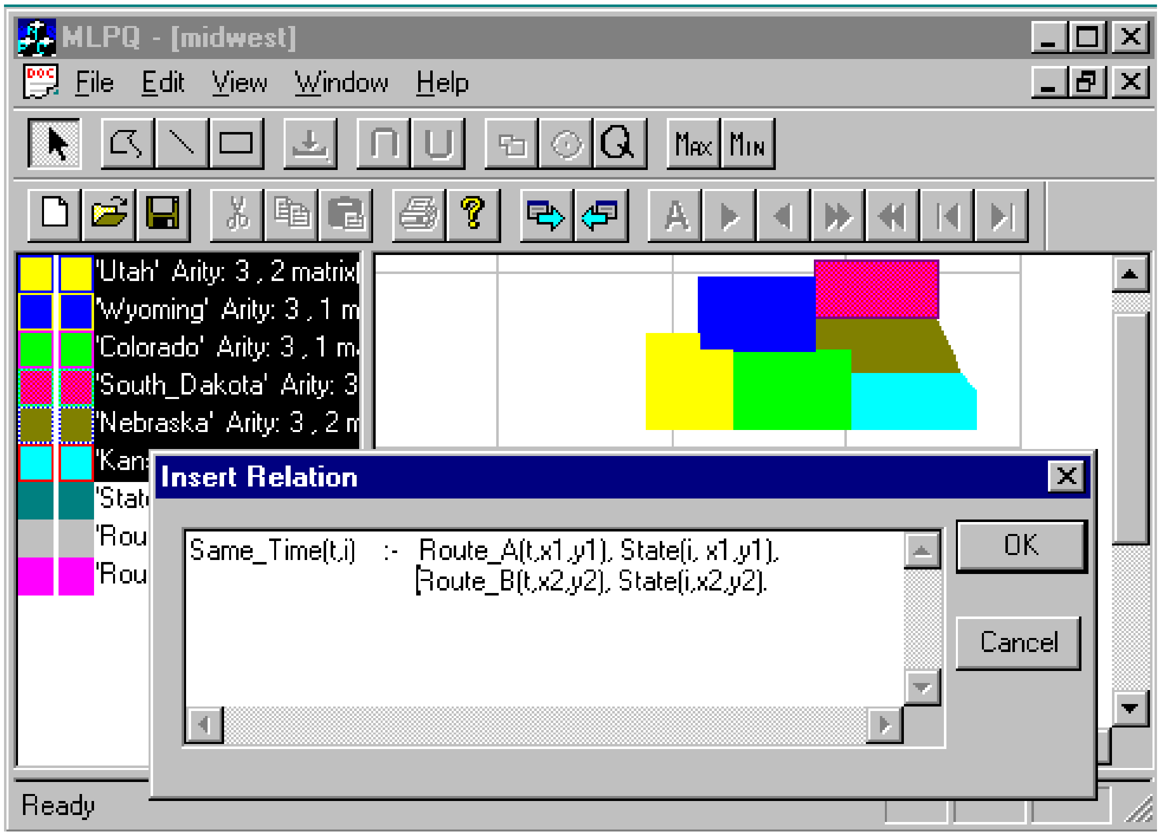
From Figure 5 we notice that the relation road is a piece-wise linear function of x. To answer the first query, click the Min icon and create a new relation *Nearest_Exit* with attributes x and y. We find the minimum value of x in the relation represented in Datalog as $Current\_Pos(x0, y0)$, $Road(highId, x, y)$, $Exit(exitId, x, y)$, $x > x0$ where $x0$ is the current position of person.

Query: The name of the hotels that are 500 meters from the nearest exit.

For the second query, the buffer icon is used to create a relation, say *Buf_Nearest_Exit*, that creates a 500 meter buffer around the x and y coordinates of the *Nearest_Exit* relation. Using the intersection icon, find the intersection of relation *Hotel* and *Buf_Nearest_Exit* and lets call this relation *Convenient_Hotel*. The *Hotel_Name* attribute of this relation gives the name of the hotels we were looking for in query 2.

3.2 General Queries

A linear constraint database representation of GIS information brings with it the advantage of using simple Datalog rules to implement queries that are not possible to implement using standard SQL. To enable the user to be able to tap this querying power and to define new relations (maps) the system provides a mechanism by which the user can enter queries at run-time by entering Datalog rules in a query dialog box. We illustrate the general query capabilities of MLPQ/GIS system in



```
Same_Time(t,i)   :-  Route_A(t,x1,y1), State(i, x1,y1),
                     Route_B(t,x2,y2), State(i,x2,y2).
```

Figure 6: Mid-West Problem

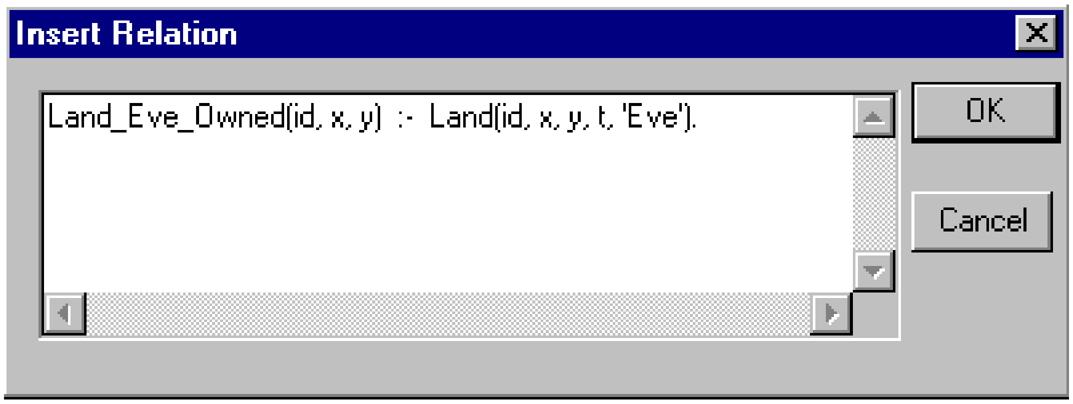
the following examples.

3.2.1 Mid-West Problem

We are given the map of the mid-western United States represented using a linear constraint database. The database also contains information about the route and itinerary of two persons, A and B who are travelling across the mid-west.

Query: When are the two people in the same state at the same time?

To answer this query, we read in the mid-western states one by one, and then create a relation called *State* from their union using the union icon. Next, we call a query dialog box and enter a Datalog query as shown in Figure 6. This query says to return all pair of time and state $pairs(t, i)$ such that at time t person A is at a location $(x1, y1)$ and person B is at location $(x2, y2)$ and both the locations are in the same state i.



```
Land_Eve_Owned(id, x, y)  :-  Land(id, x, y, t, 'Eve').
```

Figure 7: Land Owner Problem

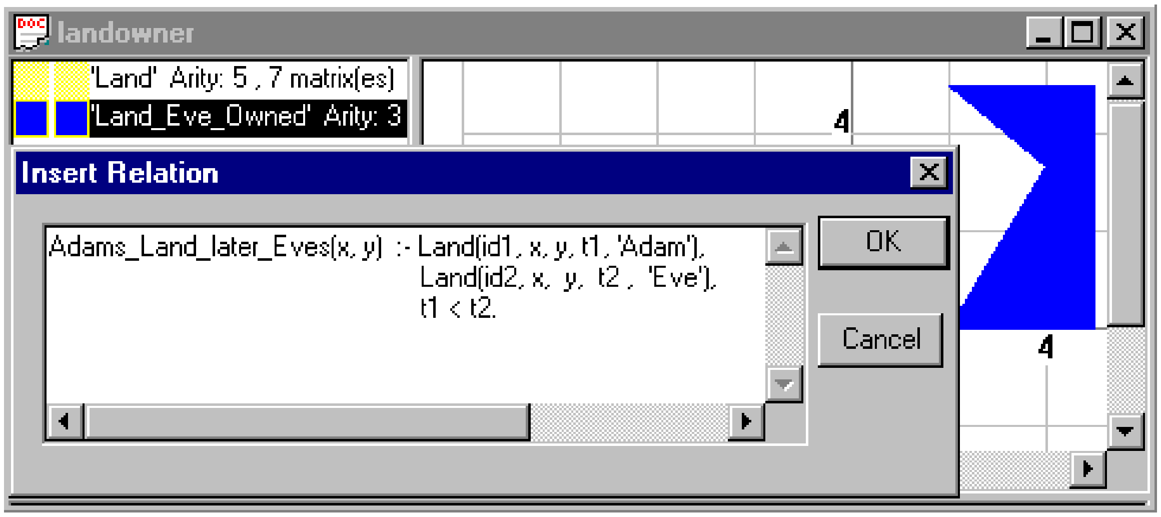
3.2.2 Land Owner

Suppose that information about ownership of a plot of land is stored in the relation *Land*, which has attributes *id, x, y, time,* and *owner* in that order. An example of such a tuple can be found in Example 2.2.

Query: What parts of the land belonged to Eve at some point in time?

To solve the first query, we open a query dialog box by clicking on the query icon. We write a simple Datalog query as shown in Figure 7. The query selects all those tuples from relation *Land* that have owner attribute equal to Eve.

Query: Which parts of the land belonged once to Adam and later to Eve?



```
Adams_Land_later_Eves(x, y)  :- Land(id1, x, y, t1, 'Adam'),
                                Land(id2, x,  y,  t2 ,  'Eve'),
                                t1 < t2.
```

Figure 8: Land Owner Problem: Adam and Eve

For the second query, we have to find the intersection of all parts of the land that both Adam and Eve owned while ensuring that that only those tuples in the intersection be in the result, such that the land was first owned by Adam and later by Eve. The Datalog query that implements this query is shown in Figure 8.

4 Comparison with ARC/INFO

ARC/INFO is one of today's most successful GIS systems that is based on the geo-relational model [17]. In systems following this model, the spatial and the non-spatial data are kept separate. In ARC/INFO, the spatial data handling is done in a package called ARC, while the non-spatial data is handled using the INFO package. There is an automatic set-up of a one-to-one correspondence between the spatial and the non-spatial tables using internally generated ids.

ARC can represent only spatial information. If temporal data is used, it must be handled separately by the INFO part. Because of this separate handling of spatial and temporal data, ARC/INFO cannot represent (linear) dependencies between spatial and temporal attributes.

In contrast, MLPQ/GIS is a constraint database system, which allows any number of attributes to be used within a relation and each MLPQ tuple can represent a complex relationship among all the attributes. For example, in a single MLPQ relation the three spatial dimensions, the time, the precipitation, and the price could all be attributes that are inter-constrained by each other using complex linear relationships.

For some specialized queries, ARC/INFO provides a good optimized performace. It also allows ready-built software packages to be added to the system, providing one way of extending its basic query language. However, it is difficult to improvise new queries and express them in ARC/INFO.

MLPQ/GIS does not provide the optimization on two dimensional spatial data that ARC/INFO provides. However, it has a much more flexible query language, integrated into the graphical user interface. Hence it is easier to improvise queries and express them in MLPQ/GIS. The advantage of higher expressive power is highlighted by the summary provided in Table 1 below.

Query	ARC/INFO	MLPQ/GIS
Pesticide	3 ARC/INFO commands	Intersection and area icons
Tracking	1 ARC/INFO command	Overlay diplays
Incomplete Info.	No valid input database	Overlay displays
Nearest Exit (Buffer 1.)	Not expressible	Min icon
Hotels Close (Buffer 2.)	2 ARC/INFO commands	Buffer icon
Midwest	Not expressible	1 Conjunctive Query
Land Owner 1	2 ARC/INFO commands	1 Conjunctive Query
Land Owner 2	Not expressible	1 Conjunctive Query

Table 1: Comparison of ARC/INFO and MLPQ/GIS Queries

5 Future Work

5.1 Interoperability of MLPQ/GIS with other Systems

We plan to make MLPQ/GIS useful in the real world applications. So it is important to get real world data for MLPQ/GIS. We are currently doing research on database interoperability [5] between ARC/INFO and MLPQ/GIS. Now, the importing of ARC/INFO's polygon and temporal information into MLPQ/GIS 's constraint data has already been implemented. In fact, the data for the land owner example is obtained from ARC/INFO. By doing queries on the constraint data, we were able to obtain answers to queries that were not expressible in the basic langauge of ARC/INFO. The next step is translating the query result back into ARC/INFO data. So we can also use MLPQ/GIS as an intermediate layer that enhance the query ability for ARC/INFO by constraint query language. Based on MLPQ/GIS's constraint data model, we will extend our research on database interoperability to other commercial GIS systems.

Because MLPQ/GIS can deal with spatio-temporal information, we will also do research on the animation of dynamic spatial information that change with time. Using MLPQ/GIS, we can store maps of different times(land owner example) into one constraint relation without any data redundancy. It's easy to get the map of any time using the projection query provided by MLPQ/GIS.

5.2 New Applications

Now, we are also using MLPQ/GIS to solve the crops planting arrangement problem and harvesting problem in University of Nebraska-Lincoln's Agricultural Research and Development Center. By using the linear programming algorithms embedded in MLPQ/GIS, we have already got satisfying result.

References

[1] M. Berkelaar. *The Lp_Solve Linear Programming Software Package.*, At: `ftp://ftp.es.ele.tue.nl/pub/lp_solve/`.

[2] R.G. Bland, D. Goldfarb and M.J. Todd, The Ellipsoid Method: A Survey, *Operations Research* 29:1039-1091, 1981

[3] A. Brodsky, Y. Kornatzky. The *Lyric* Language: Querying Constraint Objects. In: *Proc. ACM SIGMOD Conf on Management of Data.*, San Jose, CA, p. 35–46, 1995.

[4] V. Chandru. Variable Elimination in Linear Constraints, *The Computer Journal* 36: 463-472, 1993.

[5] J. Chomicki and P.Z.Revesz, Constraint-Based Interoperability of Spatiotemporal Databases, In: *Proc. 5th International Symposium on Spatial Databases*, Berlin, Germany, 1997.

[6] V. Chvatal, *Linear programming.* W.H. Freeman, New York, 1983.

[7] A. U. Frank and M. Wallace, Constraint based modeling in a GIS: Road design as a case study, In: *Proc. Twelfth International Symposium on Computer- Assisted Cartography*, 4:177-186, 1995.

[8] S. Grumbach, P. Rigaux, L. Segoufin. The DEDALE System for Complex Spatial Queries. In: *Proc. ACM SIGMOD International Conference on Management of Data*, Seattle, USA, p. 213–224, 1998.

[9] J.-L. Imbert, *Linear Constraint Solving in CLP-Languages*, Lecture Notes in Computer Science, vol. 910, 1995.

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

[11] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, **4**, p. 373–395, 1984.

[12] S. Morehouse. The Architecture of ARC/INFO. In: *Proc. International Symposium on Computer-Assisted Cartography*, p. 266-277, 1989.

[13] J. Paredaens, J.V.D. Bussche, D.V. Gucht. Towards A Theory of Spatial Database Queries. In: *Proc. 13h ACM Symp. on Principles of Database Systems*, 279–288, 1994.

[14] P. Z. Revesz and Yiming Li, MLPQ: A Linear Constraint Database System with Aggregate Operations. In: *Proc. International Database Engineering and Applications Symposium*, Montreal, Canada, 1997.

[15] D. Srivastava, Subsumption and Indexing in Constraint Query Languages with Linear Arithmetic Constraints, *Annals of Mathematics and Artificial Intelligence*, 8:315-343, 1993.

[16] J.D. Ullman. *Databases and Knowledge Base Systems.* Computer Science Press, 1988.

[17] T.C. Waugh and R. Healey, The GEOVIEW design. A relational database approach to geographic data handling, *International Journal of GIS*, 1, 101-118, 1987.