A Comparison of Abstract Data Type and Constraint Database Approaches to GIS Query Languages

Peter Revesz
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588, USA
revesz@cse.unl.edu

ABSTRACT

Designing query languages for geographic information systems using the traditional approach of constantly adding new data types and operations on the new data types has reached a limit beyond which the query language is no longer easy to understand or convenient to use. We advocate instead the design of query languages based on constraint databases. We show that many formerly difficult-looking queries, such as the shortest path query, can be expressed using simple SQL-like queries of constraint databases.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

Keywords

abstract data types, constraint databases, query language

1. INTRODUCTION

Most current work on designing query languages for GISs tries to extend relational database query languages with new operators [5, 17, 18]. At first sight the additional operators look attractive, but the problem is that they are inflexible. The additional operators often provide simple solutions to toy problems presented in many research articles and textbooks. However, when a more complex real-life problem arises, then the problem is often inconvenient or impossible to express.

The typical solution to the inherent inflexibility is to introduce more operators. Unfortunately, carrying this process too far results in rather difficult-to-understand and unwieldy query languages. For example, the International Committee for Information Technology Standards described a 166 pages long standard for spatial data operators (ISO # 19107) and 49 pages for a limited set of moving objects (ISO #19141).

A fundamental problem is that current GIS users are not given direct access to the defined spatial and spatio-temporal data types as a *set of points* $\{(x, y)\}$ in the real plane $\mathcal{R}^2$ or $\{(x, y, t)\}$ in real space $\mathcal{R}^3$. In contrast, *constraint query languages* (Revesz [13], Rigaux et al. [15] Chap. 4, and Güting and Schneider [8] Chap. 6), although implemented only in prototype systems, such as MLPQ [14], allow the users access to the x and y coordinates of the objects. As we describe below, this enables expressing shortest path and other complex queries in a much simpler, SQL-like way without the need for additional abstract data types and operators.

2. GEOGRAPHIC DATA MODELS

We review some geographic data models from [2, 3, 8, 10, 13, 15] without trying to be comprehensive.

Vector Data Model [1, 4, 16], used in the ARC/INFO system [11, 12], can describe streets and towns as follows.

Street

Id	Type	List
Bear	polyline	[(2, 11), (17, 16)]
Hare	polyline	[(4.5, 20), (9, 11)]
Maple	polyline	[(7, 5), (20, 5)]
Oak	polyline	[(11, 2), (11, 12)]
Vine	polyline	[(5, 2), (5, 14)]
Willow	polyline	[(4, 19), (20, 19)]

Town

Id	Type	List
Lincoln	polygon	[(2, 18), (6, 18), (8, 16), (12, 18), (14, 18), (14, 8), (8, 8), (6, 14), (2, 14)]

Worboys' Data Model [19] can describe a park [13].

Park

Id	Ax	Ay	Bx	By	Cx	Cy	From	To
Fount	10	4	10	4	10	4	1980	1986
Road	5	10	9	6	9	6	1995	1996
Road	9	6	9	3	9	3	1995	1996
Tulip	2	3	2	7	6	3	1975	1990
Park	1	2	1	11	12	11	1974	1996
Park	12	11	12	2	1	2	1974	1996
Pond	3	5	3	8	4	9	1991	1996
Pond	4	9	7	6	3	5	1991	1996
Pond	3	5	7	6	6	5	1991	1996

Topological Data Models, for example [6, 7], describe relations of spatial objects, such as a distance graph [13]:

ACM GIS'09, November 4-6, 2009. Seattle, WA USA

Edge

City1	City2	Miles
Chicago	Des_Moines	600
Chicago	Kansas_City	800
Chicago	Minneapolis	370
Des_Moines	Chicago	600
Des_Moines	Kansas_City	180
Des_Moines	Omaha	400
Kansas_City	Chicago	800
Kansas_City	Des_Moines	180
Kansas_City	Lincoln	150
Lincoln	Kansas_City	150
Lincoln	Omaha	60
Minneapolis	Chicago	370
Minneapolis	Omaha	650
Omaha	Des_Moines	400
Omaha	Minneapolis	650
Omaha	Lincoln	60

Constraint Data Model [9] finitely represents infinite geographic relations. Below *Travel* contains tuples such that if we reach City1 when the odometer shows Mile1, then we can reach also City2 when the odometer shows Mile2 [13].

Travel

City1	Mile1	City2	Mile2	
Chicago	d_1	Des_Moines	d_2	$d_2 - d_1 \geq 600$
Chicago	d_1	Kansas_City	d_2	$d_2 - d_1 \geq 800$
Chicago	d_1	Minneapolis	d_2	$d_2 - d_1 \geq 370$
Des_Moines	d_1	Chicago	d_2	$d_2 - d_1 \geq 600$
Des_Moines	d_1	Kansas_City	d_2	$d_2 - d_1 \geq 180$
Des_Moines	d_1	Omaha	d_2	$d_2 - d_1 \geq 400$
Kansas_City	d_1	Chicago	d_2	$d_2 - d_1 \geq 800$
Kansas_City	d_1	Des_Moines	d_2	$d_2 - d_1 \geq 180$
Kansas_City	d_1	Lincoln	d_2	$d_2 - d_1 \geq 150$
Lincoln	d_1	Kansas_City	d_2	$d_2 - d_1 \geq 150$
Lincoln	d_1	Omaha	d_2	$d_2 - d_1 \geq 60$
Minneapolis	d_1	Chicago	d_2	$d_2 - d_1 \geq 370$
Minneapolis	d_1	Omaha	d_2	$d_2 - d_1 \geq 650$
Omaha	d_1	Des_Moines	d_2	$d_2 - d_1 \geq 400$
Omaha	d_1	Minneapolis	d_2	$d_2 - d_1 \geq 650$
Omaha	d_1	Lincoln	d_2	$d_2 - d_1 \geq 60$

3. QUERIES

Example 3.1 Find areas where three park objects intersect.

SQL and ADT: Add the operators:

intersect_ΔΔ(triangle, triangle) $\rightarrow 2^{triangle}$*:* This operator returns the intersection of two triangle ADTs of the Worboys' data model. Note that the intersection of two triangles is in general not a single triangle but a set of triangles.

intersect_*$\Delta 2^{\Delta}$(triangle,* $2^{triangle}$*)* $\rightarrow 2^{triangle}$*:* This operator returns the intersection of a triangle ADT of the Worboys' data model with a set of triangle ADTs of the Worboys' data model.

Let $\overline{Pi} = Pi.Ax, Pi.Ay, Pi.Bx, Pi.By, Pi.Cx, Pi.Cy$ for $1 \leq i \leq 3$. Then the query is:

SELECT intersect_$\Delta 2^{\Delta}$($\overline{P1}$,intersect_ΔΔ($\overline{P2}$,$\overline{P3}$))
FROM Park AS P1, Park AS P2, Park AS P3

SQL and Constraint Database: Allows a simpler query:

SELECT P1.X, P1.Y
FROM Park AS P1, Park AS P2, Park AS P3
WHERE P1.X = P2.X AND P2.X = P3.X AND
P1.Y = P2.Y AND P2.Y = P3.Y

Example 3.2 Find intersections of the streets and the town.

SQL and ADT: Since the streets and the town maps have different units and origins, we need the following operators:

inter_|◇(polyline, region) $\rightarrow 2^{polyline}$*:* This operator returns the intersection of a vector data model polyline and region ADT.

scale_|(polyline,c_1*,*c_2*)* $\rightarrow$ *polyline:* This operator scales a polyline ADT of the vector data model by a factor of c_1 units in the x and c_2 units in the y direction.

shift_|(polyline,b_1*,*b_2*)* $\rightarrow$ *polyline:* This operator shifts a polyline ADT of the vector data model by b_1 units in the x and b_2 units in the y direction.

Using these operators with the proper values of c_1, c_2, b_1, and b_2, the query can be now correctly expressed as follows:

SELECT inter_|◇(shift(scale(S.List,c_1,c_2),b_1,b_2), T.List)
FROM Street AS S, Town AS T

SQL and Constraint Database: Assume (x, y) in *Street* corresponds to (x_2, y_2) in *Town* where $x_2 = c_1 x + b_1$ and $y_2 = c_2 y + b_2$. Then the query can be expressed as follows:

SELECT T.X, T.Y
FROM Street AS S, Town AS T
WHERE S.X = c_1 T.X + b_1 AND
S.Y = c_2 T.Y + b_2

Example 3.3 Projected on a table, the shadow of a book is enlarged by 10 centimeters on all sides when the book is raised up towards the ceiling. Find the enlarged shadow.

SQL and ADT: We introduce the following operator.

buffer_□(rectangle,rational) $\rightarrow$ *rectangle:* This operator takes in a rectangle and a rational number d and finds a rectangle that contains the points (x, y) such that $|x' - x| \leq d$ or $|y' - y| \leq d$ from some point (x', y') of the rectangle.

SELECT buffer_□(x_1, y_1, x_2, y_2, 10)
FROM Book

SQL and Constraint Database: Let *Plane* be the relation that contains the entire Eucledian plane. Then the shadow of the book can be found as follows.

SELECT P.X, P.Y, B.T
FROM Book AS B, Plane AS P
WHERE ((B.X - P.X $\leq$ 10) OR (P.X - B.X $\leq$ 10)) AND
((B.Y - P.Y $\leq$ 10) OR (P.Y - B.Y $\leq$ 10))

Example 3.4 Find (the length of) the shortest path from Lincoln to Chicago using the *Edge* relation.

SQL and ADT: We introduce the following operator.

shortest_distance(graph,vertex,vertex) $\rightarrow$ *number:* This operator takes in a distance graph with positive distances and two vertices and returns the shortest distance from the first vertex to the second vertex.

shortest_path(graph,vertex,vertex) $\rightarrow$ *number:* This operator takes in a distance graph with positive distances and two vertices and returns the shortest path from the first vertex to the second vertex.

The respective queries can be expressed similarly to the shadow query replacing buffer with the above operators.

SQL and Constraint Database: First we calculate the possible distances from Lincoln to the other cities.

```
CREATE VIEW    Possible(City,Mile)
SELECT         City2, Mile2
FROM           Travel
WHERE          City1 = "Lincoln" AND
               Mile1 = 0
RECURSIVE
SELECT         City2, Mile2
FROM           Possible, Travel
WHERE          City = City1 AND
               Mile = Mile1
```

We can find now the length of the shortest path from Lincoln to any other city as follows.

```
CREATE VIEW    Distance(City,Mile)
SELECT         City, Min(Mile)
FROM           Possible
GROUP BY       City
```

(If we only wanted the distance to Chicago we could add a WHERE clause which selects the city to be Chicago.) We can use the above *Distance* relation and the *Edge* relation to find the actual shortest path. Note that an edge from City1 to City2 is on (one of) the shortest path(s) if City1 is Mile1 from Lincoln, City2 is Mile2 from Lincoln, and the length of the edge is exactly Mile2-Mile1 miles long. We can express this observation and find all the edges that are on any shortest path from Lincoln to any other city as follows.

```
CREATE VIEW    Shortest_DAG(City1,City2,Mile)
SELECT         E.City1, E.City2, L2.Mile
FROM           Distance D1, Distance D2, Edge E
WHERE          D1.City = E.City1 AND
               D2.City = E.City2 AND
               D2.Mile > D1.Mile AND
               D2.Mile - D1.Mile = E.Mile
```

Let us view the *Shortest_DAG* relation as a directed acyclic graph (DAG) where the edges are directed from City1 to City2 because a shortest path always goes from City1 to City2. Clearly, *Shortest_DAG* is acyclic because no shortest path can contain a cycle. There can be several incoming edges to a City2 only if all of the incoming edges are on equally short paths. Hence if $City1, City2, Mile)$ and $(City1', City2, Mile')$, then $Mile = Mile'$. The DAG can be made a tree by selecting for each City2 the alphabetically smallest parent node City1 in the DAG.

```
CREATE VIEW    Shortest_Tree(City1,City2,Mile)
SELECT         Min(City1), City2, Mile
FROM           Shortest_DAG
GROUP BY       City2, Mile
```

Now we can go from Chicago back to Lincoln, which is the root of this tree.

```
CREATE VIEW    Shortest_Path(City1,City2,Mile)
SELECT         City1, City2, Mile
FROM           Shortest_Tree
WHERE          City2 = "Chicago"
RECURSIVE
SELECT         ST.City1, ST.City2, ST.Mile
FROM           Shortest_Tree ST, Shortest_Path SP
WHERE          ST.City2 = SP.City1
```

Finally, we can sort on the Miles attribute the rows in the *Shortest_Path* relation to find the shortest path.

Example 3.5 Find whether all the streets are connected.

SQL and ADT: We introduce the following operator.

connected(street map) $\rightarrow$ *Boolean:* This operator takes in a street map and tests whether each pair of streets is connected by a path.

The query can be expressed similarly to the shadow query replacing buffer with the above operators.

SQL and Constraint Database: Consider the *Street* relation and assume that it is represented as a constraint relation. If the streets are connected, then we can go from any arbitrary street point, for example $(5, 2)$, to any street. At first we find each pair of streets that intersect, making it possible to cross from one to the other:

```
CREATE VIEW    Cross(From,To)
SELECT         S1.Name, S2.Name
FROM           Street AS S1, Street AS S2
WHERE          S1.X = S2.X AND
               S1.Y = S2.Y
```

Second, we find all possible ways to turn as follows:

```
CREATE VIEW    Reach(To)
SELECT         Name
FROM           Street
WHERE          X = 5  AND  Y = 2,
RECURSIVE
SELECT         C.To
FROM           Reach AS R, Cross AS C
WHERE          R.To = C.From
```

Next, we find the number of different reachable streets.

```
SELECT    Count(DISTINCT To)
FROM      Reach
```

The streets are connected if and only if the count is six, the total number of streets.

Example 3.6 While making a furniture delivery, a truck cannot use wet streets with a slope greater than 27^0. The furniture store is at $(5, 2)$ on Vine street, just opposite the hospital. The delivery location is at $(19, 19)$ on Willow street. Suppose the entire town map is located on a hillside with the elevation $z = 0.5(x + y)$. If it rains now, can the truck make the delivery today?

SQL and ADT: There is no standard ADT for this query.

SQL and Constraint Database: First, find the slope of the streets. Consider any street with points (x_1, y_1) and (x_2, y_2) on it. Between these two points, the length of the street projected onto the (x, y) plane is:

$$length = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The rise in elevation is:

$$rise = 0.5(x_2+y_2) - 0.5(x_1+y_1) = 0.5((x_2-x_1)+(y_2-y_1))$$

Hence we have the constraint: We also have:

$$\frac{rise}{length} \leq \tan(27^0) \approx 0.5$$

or

$$rise \leq 0.5\ length$$

Since the truck has to go both up and down the street, we can assume that the rise is positive (otherwise interchange points (x_1, y_1) and (x_2, y_2)). Hence we can square both sides:

$$rise^2 \leq 0.25\ length^2$$

Substituting, we get:

$$((x_2 - x_1) + (y_2 - y_1))^2 \leq (x_2 - x_1)^2 + (y_2 - y_1)^2$$

Simplifying, we get:

$$(x_2 - x_1)(y_2 - y_1) \leq 0$$

We can use the above observation to find the safe streets by:

```
CREATE VIEW   Safe(Name, X, Y)
SELECT        S1.Name, S1.X, S1.Y
FROM          Street AS S1, Street AS S2,
WHERE         S1.Name = S2.Name AND
              (S2.X - S1.X) (S2.Y - S1.Y) ≤ 0
```

Finally, we can proceed as in Example 3.5 but using relation *Safe* instead of *Street*.

4. CONCLUSION

Our argument that "fewer is better" regarding ADTs and operators may be as popular among members of the International Committee for Information Technology Standards as the notion of "basic English is enough" among the editors of the *Oxford English Dictionary*, which lists over 500,000 English words. Our aim was not to please experts who love complexity but to appeal to users who seek simplicity. While dictionaries in libraries grow larger, language in life is simplified. Yet expressiveness is preserved because simpler grammar is compensated by stricter word order and smaller vocabulary by words that can be both nouns and verbs.

As New English simplifies Old English, constraint query languages simplify GIS query languages. Yet expressiveness is again preserved because fewer ADTs and operators are compensated by allowing attributes like x, y and t to refer to both finite and infinite sets of points in space and time.

In human languages, flexibility springs from simplicity. Likewise, for GIS query languages to be widely used by millions, simplicity is not an option but a must.

5. REFERENCES

[1] N. R. Adam and A. Gangopadhyay. *Database Issues in Geographic Information Systems.* Kluwer, 1997.

[2] S. Anderson and P. Revesz. Efficient maxcount and threshold operators of moving objects. *Geoinformatica*, 13(4):355–396, 2009.

[3] J. Chomicki and P. Revesz. Constraint-based interoperability of spatiotemporal databases. *Geoinformatica*, 3(3):211–43, 1999.

[4] B. Dent. *Cartography Thematic Map Design.* McGraw-Hill, 1999.

[5] M. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, 1994.

[6] M. Egenhofer and R. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–74, 1991.

[7] F. Geerts, P. Revesz, and J. Van den Bussche. On-line maintenance of simplified weighted graphs for efficient distance queries. In *Proc. 14th ACM International Symposium on Advances in Geographic Information Systems*, pages 203–10. IEEE Press, 2006.

[8] R. Güting and M. Schneider. *Moving Objects Databases.* Morgan Kaufmann, 2005.

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

[10] H. J. Miller and E. A. Wentz. Geographic representation in geographic information systems and spatial analysis. *Annals of the Association of American Geographers*, 93(3):574–94, 2003.

[11] S. Morehouse. The architecture of ARC/INFO. In *Proc. 9th Auto Carto Conference*, pages 266–77, 1989.

[12] D. J. Peuquet and D. F. Marble. ARC/INFO: An example of a contemporary geographic information system. In D. J. Peuquet and D. F. Marble, editors, *Introductory Readings in Geographic Information Systems.* Taylor & Francis, 1990.

[13] P. Revesz. *Introduction to Constraint Databases.* Springer-Verlag, 2002.

[14] P. 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.

[15] P. Rigaux, M. Scholl, and V. Agnés. *Introduction to Spatial Databases: Applications to GIS.* Morgan Kaufmann, 2002.

[16] J. Star and J. Estes. *Geographic Information Systems: An Introduction.* Prentice-Hall, 1989.

[17] P. Svensson and H. Zhexue. Geo-sal: a query language for spatial data analysis. In *Proc. Advances in Spatial Databases*, volume 525 of *Lecture Notes in Computer Science*, pages 119–40. Springer-Verlag, 1991.

[18] T. C. Waugh and R. G. Healey. The GEOVIEW design: A relational database approach to geographical data handling. *International Journal of Geographic Information Systems*, 1(2):101–18, 1987.

[19] M. F. Worboys. A unified model for spatial and temporal information. *Computer Journal*, 37(1):26–34, 1994.