

Algebras for Moving Objects and Their Implementation*

Ralf Hartmut Güting

LG Datenbanksysteme für neue Anwendungen

FB Informatik, Fernuniversität Hagen, D-58084 Hagen, Germany

rhg@fernuni-hagen.de, <http://www.fernuni-hagen.de/inf/pi4/>

1 An Algebra for Moving Objects in the 2D Plane

The general goal of the work described in the sequel is to provide support for time-dependent geometries, or moving objects, in databases. Hence we would like to find extensions to DBMS data models and query languages that allows us to model moving objects and to formulate all kinds of queries about such movements. This needs to be supported by corresponding extensions of the DBMS implementation such as new data structures and algorithms, indexes, optimization rules, cost functions, etc.

Systems of spatial data types and operations, or *spatial algebras*, are a key concept in the modeling and implementation of spatial database systems. The idea can be extended to spatio-temporal databases by introducing spatio-temporal types. A basic assumption is that geometries can change continuously (whereas the earlier work generally supported only discrete changes). To emphasize the capability of continuous change, we speak of *moving objects*. Hence we have data types such as *moving point*, describing an entity for which only the time-dependent position in space is of relevance, or *moving region*, a time-dependent area for which also the extent is relevant, hence, which can not only move, but also grow or shrink. A value of type *moving point* might represent, for example, the trip of a person or vehicle (embedded in space and time); a value of type *moving region* might represent the growth of a forest, a forest fire, or an oil spill in the sea. In the sequel we first briefly illustrate this approach.

At a conceptual, or *abstract* level, a value of type *moving point* (type *mpoint*, for short) is a function from time into point values, and a *moving region* (*mregion*) value is a function from time into region values. Such values exist in a three-dimensional (2D + time) space, as illustrated in [Figure 1](#).

Data types may be embedded in the role of attribute types into a DBMS data model. For example, in a relational setting, we may have relations to represent the movements of air planes or storms:

```
flight (id: string, from: string, to: string, route: mpoint)  
weather (id: string, kind: string, area: mregion)
```

The data types include suitable operations such as:

(*) This work was partially supported by a grant Gu 293/8-1 from the Deutsche Forschungsgemeinschaft (DFG), project "Datenbanken für bewegte Objekte" (Databases for Moving Objects).

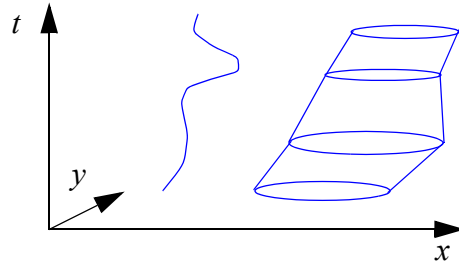


Figure 1: A *moving point* value and a *moving region* value

intersection:	<i>mpoint</i> × <i>mregion</i>	→ <i>mpoint</i>
trajectory:	<i>mpoint</i>	→ <i>line</i>
deftime:	<i>mpoint</i>	→ <i>periods</i>
length:	<i>line</i>	→ <i>real</i>

One discovers quickly that in addition to the main types of interest, *mpoint* and *mregion*, related spatial and temporal as well as other time-dependent types are needed. The operations above have the following meaning: **Intersection** returns the part of a moving point whenever it lies inside a moving region, which is a moving point (*mpoint*) again. **Trajectory** projects a moving point into the plane, yielding a *line* value (a curve in the 2D space). **Deftime** returns the set of time intervals when a moving point is defined, of a data type called *periods*. **Length** returns the length of a *line* value.

Given such operations, we may formulate queries:

“Find all flights from Düsseldorf that are longer than 5000 kms.”

```
select id
from flights
where from = 'DUS' and length(trajectory(route)) > 5000
```

“At what times was flight BA488 within the snow storm with id S16?”

```
select deftime(intersection(f.route, w.area))
from flights as f, weather as w
where f.id = 'BA488' and w.id = 'S16'
```

The approach described so far has been worked out in a number of papers. The first paper of this series [EGSV99] develops the basic idea, discusses the distinction between *abstract models* (defining values of data types as infinite point sets, and describing e.g. a moving region as a function from time into region values) and *discrete models* (selecting a suitable finite representation, e.g. describing a moving region as a polyhedron in the 3D space), and clarifying several fundamental questions related to this approach. The second paper [GBE+00] carefully defines a system of related data types and operations, that is, *an algebra for moving objects*, emphasizing genericity, closure, and consistency. The semantics of these types is defined at the abstract level. The paper [FGNS00] introduces a corresponding discrete model, which means, it provides finite representations for all the data types which can be mapped into data structures. Finally, the paper [CFG+03] refines the data structures from [FGNS00] and then studies systematically algorithms implementing the operations defined in [GBE+00].

2 An Algebra for Objects Moving in Networks

Whereas free movement in space is the most general case, in many applications entities move along fixed paths or *networks*. In particular, vehicles move on road networks, public transport networks, and so forth. Such movement can be described by the data model of [Section 1](#), but it makes a lot of sense to exploit the properties of the special case. First of all, the data model should have an explicit concept of a spatially embedded network. Second, positions of static or moving objects should be described relative to the network, rather than the Euclidean space. Advantages are the following:

- Users can formulate relationships between objects and components of the network in queries.
- The system can create dedicated data structures to represent the network for efficient traversal.
- Descriptions of moving objects become much more compact, especially if a route-oriented network model is used (see below).
- Relationships between objects and the network can be evaluated much more efficiently, since no geometric checks are needed.

In [GAD04] an algebra for network-constrained moving objects is defined. A network is formally modeled as a set of routes and a set of junctions between routes, rather than as a directed graph (a route corresponds to a path over a directed graph). Two main advantages over the directed graph model are that (i) routes rather than edges are the conceptual entities in real life (e.g. streets or highways are paths over the respective networks), and (ii) descriptions of movement are much more compact. For example, a vehicle moving along a highway at constant speed can be represented by a single entry whereas in the directed graph model a new entry is needed whenever a node of the network (an exit or a junction) is passed. The network model allows one to distinguish simple and dual roads such as divided highways, and it captures possible transitions at junctions.

The model of [GAD04] then introduces data types *network*, *gpoint*, and *gline* to represent the network, a position on the network, or a region within the network (which geometrically corresponds to a set of curves, hence, a *line* value). For example, a facility such as a gas station could be represented as a *gpoint* value, and a construction or speed limit area as a *gline* value. Network information can be exported from a *network* value into relations and vice-versa. The data types *gpoint* and *gline* are integrated into the type system of the algebra of [Section 1](#) [GBE+00] which yields data types *moving(gpoint)* and *moving(gline)* (*mgpoint* and *mgline* for short) corresponding to time-dependent network positions and regions. Hence a moving vehicle can be represented as an *mgpoint* value and a traffic jam as a value of type *mgline*. A comprehensive set of operations is defined to arrive at a flexible query language for network-constrained moving objects. Many operations of the generic model [GBE+00] are still applicable.

3 Implementing an Algebra in the SECONDO Environment

Systems of data types and operations, i.e., algebras, can be implemented in suitable extensible DBMS environments as data blades, cartridges, extenders, etc. For a full support it is not only necessary to provide data structures for the types and algorithms for the operations, but also appropriate

kinds of indexes, extensions of the optimizer (rules, cost functions), and extensions of the graphical user interface to visualize and possibly animate values of the new data types.

Our group has worked for several years on an extensible DBMS environment called *SECONDO* suitable for research prototyping as well as teaching concepts of database systems. We give a brief overview. *SECONDO* does not implement a fixed DBMS data model but is open for implementing a wide variety of models. It consists of three major components, namely (i) the kernel system, (ii) the optimizer, and (iii) the graphical user interface (GUI).

- The *SECONDO* kernel, written in C++, implements specific data models, is extensible by *algebra modules* (explained below), and provides query processing over the implemented algebras. It is implemented on top of BerkeleyDB, a storage manager providing stable storage with transactions and recovery.
- The optimizer, written in PROLOG, provides as its core capability conjunctive query optimization (i.e., find a plan, given a set of relations and a set of selection and join predicates). It also implements the essential part of an SQL-like language in a notation adapted to PROLOG.
- The GUI, written in Java, is on the one hand an extensible interface for an extensible DBMS such as *SECONDO*. It is extensible by *viewers* for new data types or models. On the other hand, there is a specialized viewer available in the GUI for spatial types and moving objects, providing a generic spatial database interface, including animation of moving objects.

The three components can be used independently or together, running as three communicating processes as shown in [Figure 2](#). In this configuration, the GUI can send commands or queries to the ker-

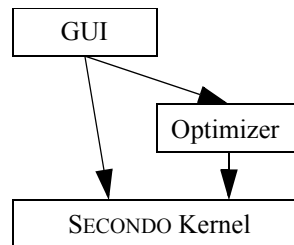


Figure 2: Cooperation of *SECONDO* Components

nel, where queries are terms of the algebras implemented in the kernel, hence in fact *query plans* (so-called *executable queries*). The kernel returns results to the GUI for display. The GUI can also send SQL-like queries to the optimizer which returns a query plan to the GUI. The GUI then sends this plan for execution to the kernel. In optimizing a query, the optimizer also sends executable queries to the kernel, to determine for example cardinalities of relations or selectivities of predicates.

The *SECONDO* kernel provides generic query processing over a set of implemented algebra modules. A query is a term built from names of objects in the database and operations of the implemented algebras. An *algebra module* offers a set of data types (type constructors, to be precise) and operations. For each type constructor, the module implements a data structure as well as some support functions. For example, there are functions to map a value from a nested list representation¹ to the data structure and vice-versa. For each operation, there are also some support functions. For exam-

ple, there are functions for type checking and mapping, resolution of overloading, and of course for *value mapping*, i.e., computing the result value from the argument values.

The concept of algebra modules is fairly general and covers the data model dependent part of implementing a specific data model in `SECONDO`. Some algebra modules currently available are:

- `StandardAlgebra`. Provides data types *int*, *real*, *string*, *bool*.
- `RelationAlgebra`. Provides relation and tuple types with all operations to implement an SQL-like relational language.
- `BTreeAlgebra`. Offers B-trees and construction and search operations.
- `RTreeAlgebra`. The same for R-trees.
- `SpatialAlgebra`. Offers spatial data types *point*, *points*, *line*, *region*.
- `DateTimeAlgebra`. Provides temporal data types *instant* and *duration*.
- `TemporalAlgebra`. Provides data types *periods* (sets of time intervals), *range(α)* (sets of disjoint intervals of other data types), *mpoint*, *mreal*, i.e., a subset of the types and a subset of the operations of the algebra of [Section 1](#) [GBE+00]. It also contains *mgpoint* and *mgline* from [Section 2](#) [GAD04].
- `NetworkAlgebra`. Contains data types *network*, *gpoint*, and *gline* from [Section 2](#) [GAD04].

All algebras offer appropriate operations with their data types. One can observe that the algebras of [GBE+00] and [GAD04] are distributed here over several algebra modules.

`SECONDO` does cost-based query optimization. To support query optimization for new algebras like those for moving objects, the optimizer needs to be extended by rules describing how selections or joins can be translated to the use of specialized index structures or join algorithms. Such rules can be formulated relatively easily in `PROLOG`. It is also necessary to provide cost functions for the algebra operations; these are also written in `PROLOG`.

The hardest problem in query optimization for such non-standard algebras is selectivity estimation. Traditional DBMS use mainly statistics such as histograms. However, traditional DBMS have very simple types and an extremely limited set of operations in comparison. Extending the traditional approach to moving objects algebras (with very large sets of types and operations) requires a very large amount of research and implementation work. Whereas this is a nice topic for many Ph.D. dissertations, for someone who wants to build a system now it takes too long to wait for the results.

`SECONDO` implements an “automatic” approach to selectivity estimation based on sampling. The (currently implemented relational) system maintains for each relation a small random sample of its tuples (say, 500 tuples). Query optimization starts by evaluating each individual selection and join predicate by a simple standard technique on the samples to determine its selectivity. Samples are kept in memory to make this fast. The time to evaluate a selection predicate is usually quite small and hardly noticeable. The time to evaluate a join predicate may be a bit larger (but is still fast on the small samples). Selectivities found in this way are stored. Whereas there are many possible selection predicates (involving different constants), in practice only a few join predicates are relevant. This

1. Nested lists, as in the LISP language, are widely used in `SECONDO`, for example, as an external data exchange format, as an internal format for queries, or as a notation for constants of non-standard data types.

effect helps to make the slightly higher times for evaluating join predicates on the samples tolerable since one has to wait only once when this predicate is used for the first time.

The great advantage of this technique is that selectivity estimation for new algebras works immediately, without any additional effort. Essentially, the sample is the statistics data structure for the relation and it works for all kinds of operations of the algebra directly as the implemented operations are used to evaluate the predicate. This is in contrast to reimplementing the operations on a specialized statistics data structure such as, for example, a multi-dimensional histogram.

SECONDO is described in more detail in [DG00, GBA+04]. Papers by the author can be accessed via the URL <http://www.informatik.fernuni-hagen.de/import/pi4/gueeting/home.html>.

4 Related Work

In this short paper we can mention only a very small sample of related work. Please see the papers cited above for detailed discussions .

The group around Ouri Wolfson has pioneered the study of modeling current movement in databases which allows one to formulate queries about current and expected near future positions of moving point objects [SWCD97]. The constraint database approach can also be used to arrive at data models and query languages for moving objects [RSSG03, SXI01]. For spatially embedded networks, query processing issues are addressed in [PZMT03], data modeling in [SJK03], and indexing in [Fr03]. The use of sampling techniques for query optimization is discussed in [LNS90].

Acknowledgements

The work of the co-authors of the papers cited in the first three sections is gratefully acknowledged. In particular, I thank Zhiming Ding and Victor Almeida, current researchers in the DFG project “Databases for Moving Objects”, as well as Dirk Ansorge, Thomas Behr, Frank Hoffmann, and Markus Spiekermann for their contributions in implementing the SECONDO system.

References

- [CFG+03] Cotelo Lema, J.A., L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider, Algorithms for Moving Object Databases. *The Computer Journal* 46:6 (2003), 680-712.
- [DG00] Dieker, S., and R.H. Güting, Efficient Handling of Tuples with Embedded Large Objects. *Data & Knowledge Engineering* 32 (2000), 247-269.
- [EGSV99] Erwig, M., R.H. Güting, M. Schneider, and M. Vazirgiannis, Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica* 3 (1999), 265-291.
- [FGNS00] Forlizzi, L., R.H. Güting, E. Nardelli, and M. Schneider, A Data Model and Data Structures for Moving Objects Databases. Proc. ACM SIGMOD Conf. (Dallas, Texas), 2000, 319-330.
- [Fr03] Frentzos, R., Indexing Moving Objects on Fixed Networks. Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases (SSTD), 2003, 289-305.

- [GAD04] Güting, R.H., V.T. de Almeida, and Z. Ding, Modeling and Querying Moving Objects in Networks. Fernuniversität Hagen, Informatik-Report 308, 2004.
- [GBA+04] Güting, R.H., T. Behr, V.T. de Almeida, Z. Ding, F. Hoffmann, and M. Spiekermann, SECONDO: An Extensible DBMS Architecture and Prototype. Fernuniversität Hagen, Informatik-Report 313, 2004.
- [GBE+00] Güting, R.H., M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, and M. Vazirgiannis, A Foundation for Representing and Querying Moving Objects in Databases. *ACM Transactions on Database Systems* 25 (2000), 1-42.
- [LNS90] Lipton, R.J., J.F. Naughton, and D.A. Schneider, Practical Selectivity Estimation through Adaptive Sampling, Proc. ACM SIGMOD, 1990, 1-11.
- [PZMT03] Papadias, D., J. Zhang, N. Mamoulis, and Y. Tao, Query Processing in Spatial Network Databases. In Proc. of the 29th Conf. on Very Large Databases (VLDB), 2003, 790-801.
- [RSSG03] Rigaux, P., M. Scholl, L. Segoufin, and S. Grumbach, Building a Constraint-Based Spatial Database System: Model, Languages, and Implementation. *Information Systems*, 28(6): 563-595, 2003.
- [SJK03] Speicys, L., C.S. Jensen, and A. Kligys, Computational Data Modeling for Network-Constrained Moving Objects. Proc. of the 11th ACM Symp. on Advances in Geographic Information Systems (ACM-GIS, New Orleans, Louisiana), 2003, 118-125.
- [SWCD97] Sistla, A.P., O. Wolfson, S. Chamberlain, and S. Dao, Modeling and Querying Moving Objects. Proc. Int. Conf. on Data Engineering (Birmingham, U.K.), 1997, 422-432.
- [SXI01] Su, J., H. Xu, and O. Ibarra, Moving Objects: Logical Relationships and Queries. Proc. 7th Int. Symp. on Spatial and Temporal Databases (SSTD, Redondo Beach, California), 2001, 3-19.