Data Management for Moving Objects

Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
{hdchon,agrawal,amr}@cs.ucsb.edu

Abstract

We model a moving object as a sizable physical entity equipped with GPS, wireless communication capability, and a computer such as a PDA. Furthermore, we have observed that a real trajectory of a moving object is the result of interactions among moving objects in the system yielding a polyline instead of a line segment. In this paper, we first choose a partitioning approach to efficiently manage such trajectory information and develop a system called RouteManager based on a space-time grid that manages moving objects in a one-dimensional space. We then extend this to two-dimensional space. The time-dependent shortest path problem is an interesting application where such a system can be used.

1 Introduction

Global Positioning System (GPS) has been widely accepted as the technology for locating objects such as vehicles on a highway or soldiers in a battlefield. One of the features GPS provides, other than locating, is speed tracking so that we can get speed information without integrating GPS to a speedometer in the vehicle. Wireless communications technology such as Cellular Digital Packet Data (CDPD) [6] and Mobile IP [12] has also gained popularity. With these technologies in place, finding and managing the current locations of moving objects have become possible. Since keeping track of the location of many, continuously moving objects is hard to achieve, there are significant data management problems that need to be addressed. The focus of much of the research on this issue has been on how to manage location information efficiently without updating the location information every time it moves. One basic and fundamental observation on moving objects is that if a moving object maintains a constant speed for a certain period, future locations of a moving object can be modeled as a linear function of time during that period. Then the location information can be managed by maintaining the parameters of the linear functions instead of constant updating [18, 8, 2]. There is no known alternative to this approach so far. We also base our research on this observation and make further observations: (1) Moving objects with a few exceptions follow certain paths such as a route on a highway system. (2) Such paths on which moving objects move have some physical limitations such as the number of lanes and/or route conditions. (3) Due to the limitations, a real trajectory of a moving object is a polyline instead of a line segment. In Section 3, we explain how to manage such polyline information.

Copyright 2002 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Among many possible applications that could benefit from this research is *Time-Dependent Shortest Path Problems*. In transportation applications, *Time-Dependent Shortest Path Problems*, where the cost (travel time) of edges between nodes varies with *time*, are of significant interest [11]. Traffic congestion is a huge problem, especially in metropolitan areas. According to a recent report from the Texas Transportation Institute [17], the total congestion cost for the 68 major U.S. urban areas in 1999 was $78 billion, and Los Angeles was ranked first ($12.5 billion) in the congestion cost as well as other congestion measures. In recent years, in order to avoid traffic congestion a growing number of vehicles are equipped with a navigation system which provides a shortest path to their destination as well as other features like roadside assistance [1]. The navigation system uses a digital map, which includes static information such as the distance of a path and the speed limits along the way. However, such information does not reflect the dynamically changing traffic information, hence could lead vehicles to incorrect shortest paths. In contrast to this static approach, if there is a way to obtain the cost in real time and then apply a time-dependent shortest path algorithm, it would result in a better solution for the shortest path queries. The system we develop in this paper could be used to gather this dynamically changing information in order to run a time-dependent shortest path algorithm.

In the following section, we discuss why we choose the partitioning approach over the indexing approach to manage moving object information. Also, more detailed assumptions and observations on moving objects are given in Section 2. In Section 3, we explain the system we have developed using the partitioning approach. The system is initially designed in the context of moving objects in a one-dimensional space. Later, in Section 4, we argue the importance of the time-dependent shortest path problems and describe how we can handle moving objects in a two-dimensional space using multiple one-dimensional components. We conclude the paper with a discussion of future direction of research in this area in Section 5.

2 Indexing vs. Partitioning

In general, there are two main paradigms for managing large data sets: the indexing approach and the partitioning approach. The indexing approach is a bottom up approach where data objects are clustered together and index structure is built to efficiently locate the individual data items. This approach is usually more appropriate for sparse unstructured data sets. Alternatively, the partitioning approach a priori partitions the data space and uses simple (often constant time) methods for locating data. This approach is usually more appropriate for uniformly distributed data with well defined structure [7]. In this section, we discuss why the partitioning approach is more appropriate for managing moving object information than the indexing approach.

We start by stating in detail some assumptions regarding our model of moving objects. The best example of moving objects is vehicles on a road although it is not the only example. The moving objects considered are not particles, meaning that they do not move irregularly such as in a Brownian motion. Instead, they follow a certain path such as a route on a highway system. This is true with few exceptions such as soldiers in a battlefield or persons in a large open field. In addition, they are sizable physical entities with the following characteristics: capable of locating their position, capable of wireless communication, and equipped with a computer such as a PDA. Although there are several other technologies to locate an object, GPS is by far the dominant technology and it is getting better and cheaper [4]. As a result, we expect that in the future many (if not all) moving objects, especially vehicles, will be equipped with a GPS. Wireless communication technologies such as CDPD or IEEE 802.11b can be used together with a PDA or laptop to connect to a control center, which manages data and answers queries to either moving objects or some other interested party such as advertisers.

Let us now consider how to model moving objects' current and future locations. Future locations of a moving object can be modeled as a linear function of time if the moving object maintains a constant speed for a certain period. Therefore, the trajectory of a moving object can be expressed in terms of parameters representing a linear function. In addition to this linearity, we make further observations on the paths (space) on which moving objects move: there are physical limitations of paths on which moving objects move such as the number of lanes, route conditions, etc. Based on our everyday experience, we can expect that as the number of vehicles in a section of a route increases, the average speed of the vehicles decreases. This is supported by a report from the Texas Transportation Institute [17]. The report observed that as the daily traffic volume (the average number of vehicles observed in a day at a location) increases, the average speed of vehicles decreases. Due to this physical limitation, the trajectory of a moving object is a result of interactions among all moving objects present in the system. Consequently, the resulting trajectory would be a polyline, a sequence of linear lines. For this reason, if we are to manage future location information about moving objects on a network of routes, the routes need to be broken into smaller sections, which then are associated with information such as the number of objects and the maximum velocity allowed in the section. Regardless of what approaches (indexing or partitioning) are used, we need at least to maintain such information in each smaller section. Moreover, as we will see in Section 3, there are times when an insertion of a moving object causes a series of updates of other moving objects in its way. Therefore, each section needs to be associated with the list of moving objects in the section as well. For these reasons, we take the partitioning approach in that we partition the domain space into a grid. Each cell in the grid corresponds to a certain period of time and a section. In addition, each cell is associated with a list of moving objects and the maximum velocity allowed in the cell. To our knowledge, this approach has not been previously explored for the management of dynamic objects. In the next section, we explain the space-time partitioning model and describe how insertion works.

3 Space-Time Grid

In the previous section, we argued that the partitioning approach is more appropriate for managing moving objects' trajectory information. In this section, we summarize the model we have developed [3] for maintaining dynamically changing information about moving objects assuming that a moving object moves on a one-dimensional path.

3.1 Data Structures, and Definitions

We assume that each section of a route is associated with a list of moving objects in the section and a predefined function of the maximum velocity at which moving objects can move. This maximum velocity function associated with a section will return the maximum velocity a moving object can move in the section depending on the number of moving objects currently in the section. We can derive the function from statistical data [5]. We partition the domain space $T \times Y$, where T corresponds to the time domain and Y corresponds to the space domain, into an array of cells such that $Y = [y_0, y_1) \cup ... \cup [y_{p-2}, y_{p-1}) \cup [y_{p-1}, y_{max}]$ and $T = [t_0, t_1) \cup [t_1, t_2) \cup ... \cup [t_{q-1}, t_q)$, where $t_q = t_0 + \Delta T$. A cell $g[i, j]$ is defined as a rectangle $[t_i, t_{i+1}) \times [y_j, y_{j+1})$, for some $t_i \in T, y_j \in Y$. Note that although time is increasing indefinitely, we only maintain the information about some predefined ΔT period of time. As time evolves, we shift the time domain forward. We also assume that on entry to the system, a moving object m is responsible to provide initial information such as the starting location (s), the starting time (t_s), the destination (e), and the intended velocity (v), which is the maximum velocity at which it intends to move. When a moving object m appears in the system providing its initial information, it is inserted into the system as a polyline. Shortly, we will explain how those information becomes a polyline.

Choosing appropriate data structures is critical to achieve efficient storage and retrieval of information. We use the following data structures in the implementation: A linked list with an extra pointer to the last element is used to represent the polyline corresponding to a moving object's trajectory. The entire grid is implemented as an array of cells, which efficiently supports random accesses to individual cells. An individual cell stores only the identifiers of intersected moving objects along with the velocity of the moving objects in the cell using a skip list [15]. Since we do not store the entire information about a moving object in each cell, we need to store it

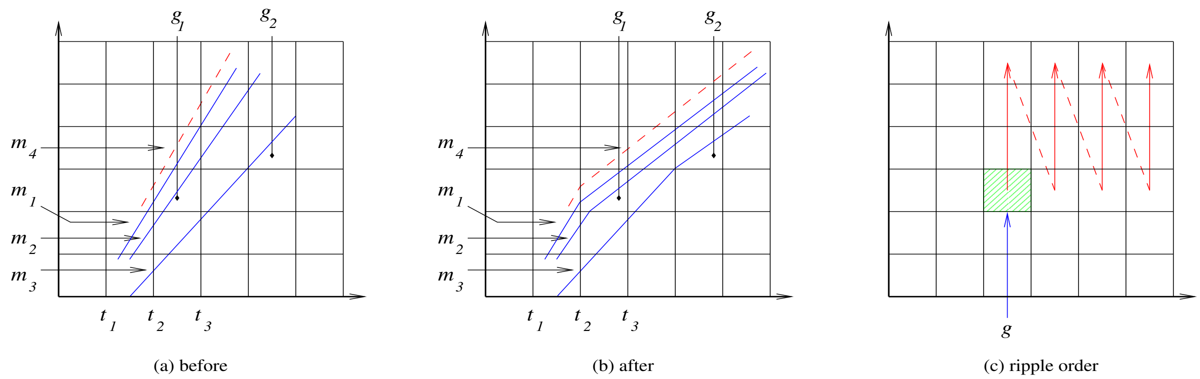

Figure 1: Ripple Effect

somewhere else, which we call an *Object Store*. A hashtable is used for the Object Store, which needs to support frequent random accesses.

Before we explain the insertion algorithm, we introduce a notation $\triangleright$ and a term *entry point* as follows: If the corresponding polyline of m intersects with a cell g, m is said to intersect with g and denoted by $m \triangleright g$. An entry point of m to a cell g is a point where m's corresponding polyline and g intersect for the first time.

3.2 Object Insertion

We now explain how to insert an object m into the system. With the initial information $m = (s, t_s, e, v)$, we find the first cell g (skip list) such that $m \triangleright g$ and repeat the following steps: we compute the maximum velocity allowed in g which depends on the number of moving objects in the cell, and adjust m's velocity accordingly. When there is a velocity change, we add the corresponding entry point to the linked list associated with m which represents a polyline. We then find the next cell to intersect by computing the entry point to the cell, and insert its identifier along with the adjusted velocity into g. We repeat these steps until we process the cell to which the destination e of m belongs. In that case, we complete the insertion process by inserting m into the Object Store, and return the polyline information to m.

There are, however, cases where one insertion may cause a series of updates. Consider the situation shown in Figure 1(a) where three moving objects, $\{m_1, m_2, m_3\}$, are in the system and a new one, m_4, is about to be inserted. Assuming that three moving objects is a threshold for moving objects to reduce their velocities to a certain degree, cell g_1 is about to have three objects so that existing objects as well as the new one will need to reduce their velocities based on the maximum velocity function associated with g_1. Furthermore, the updates of m_1 and m_2 cause the cell g_2 to have three moving objects. As a result, another update on m_3 has to be executed. Figure 1(b) shows the result of the insertion of m_4. A moving object m_4 in this case is said to cause a ripple effect. For this reason, we need to check if the number of objects in g reaches a threshold, before inserting a moving object m into g, by checking $f_g(|g|) > f_g(|g|+1)$, where $f_g(n)$ is the maximum velocity function associated with g. If it is true, an algorithm called *RippleInsert* is invoked with a (row, column) pair of g. Otherwise continue the insertion of m until it is finished.

The RippleInsert algorithm works as follows: suppose g is the first cell whose maximum velocity needs to be changed because of the insertion of a moving object. The information about all moving objects in g is then adjusted according to the new velocity allowed in g and continue this process with the cells whose time and location intervals are greater than or equal to those of g. Other cells need not to be processed since an insertion would not cause any update to the past and to previous locations. Figure 1 (c) shows the order in which cells are updated. The details of this algorithm can be found in [3].

With the data structures and this insertion algorithm in place, we can process range and k-nearest neighbor queries [5]. In the query processing algorithms we proposed, the target objects are moving as well as the query points. We do not consider the case where the target objects are stationary. If so, after getting the future or current location information of a moving object, the query becomes a traditional query (range or k-nearest neighbors), for which many algorithms have been proposed.

4 Time Dependent Shortest Paths

In the previous section, we developed a system that manages moving object information assuming that the moving objects are restricted to move in a one-dimensional space. In the real world, however, this would not be a realistic assumption. Rather, objects move in a two-dimensional space. More precisely, a moving object follows a certain path, which can be viewed as a path on a network of one-dimensional lines in a two-dimensional space [5]. Therefore, we need to extend the system to handle mobility in two-dimensional space. In this section, we explain the extension of the system and describe how it can be used for time-dependent shortest path problems.

Shortest path problems are among the most studied network flow problems and one of the most important application areas is transportation. In transportation applications, *Time-Dependent Shortest Path Problems*, where the cost (travel time) of edges between nodes varies with *time*, are of significant interest [11]. As in transportation applications, where the cost changes dynamically, the correctness of the shortest path very much depends upon the correctness of the cost model. For example, typical drivers choose a route based on static information that, in general, is heavily weighted on the length of the route. Unfortunately, as many drivers in metropolitan areas are aware, superiority of a route cannot entirely be based on the length of the route. In particular, current and future condition of routes such as the severity of congestion should play an important role in determining which route would incur minimum delay. In this case, using only the static information to find a shortest path is not enough. We need to take other variables into consideration.

In general, the time-dependent cost (normally, travel time) is given arbitrarily [10, 9, 11]. We noticed that this time-dependent shortest path problem would be an interesting application for the data management of moving objects. Once we extend the system to two-dimensional space, we will be able to provide the dynamically changing cost to a time-dependent shortest path algorithm [11] leading many vehicles to real shortest paths. This will result in better utilization of the road network and efficient traffic management.

We now extend the system to a two-dimensional space. Note first that any path in a network of one-dimensional lines in a two-dimensional space can be divided into subpaths, each of which belongs to a one-dimensional line. Therefore we can build the two-dimensional system as a network of one-dimensional subsystems. The one-dimensional system we developed in the previous section is associated with a route and we call it the RouteManager. Hence, each RouteManager maintains a local view of the corresponding route. A local view consists of the dynamically changing cost information for sections on the route. Based on these local views, a frontend interface to moving objects, PathServer, maintains a global view of the network as a directed graph. In this directed graph, each section on a route corresponds to an edge. When a moving object appears in the system, the PathServer finds a time-dependent shortest path on the graph, pushes it into a queue along with the path information, and returns it to the moving object. The queue is constantly checked by a thread, DistributionManager. If it is not empty, an element (a pair of an id of a moving object and the corresponding 2D path) is dequeued and broken into 1D subpaths. The DistributionManager then distributes the 1D subpaths to corresponding RouteManagers. For PathServer to keep up-to-date information about the routes, each RouteManager constantly sends an updated local view to the PathServer. Then a thread in the PathServer, PathManager, maintains up-to-date information in the graph structure. Hence, we are able to reuse the system (RouteManager) we developed in the previous section and take advantage of parallelism. Note that a ripple effect can occur in any RouteManager after DistributionManager distributes 1D subpaths to the corresponding RouteManagers. Once a ripple effect occurred, the moving objects affected will arrive at subsequent routes later than the routes antici-

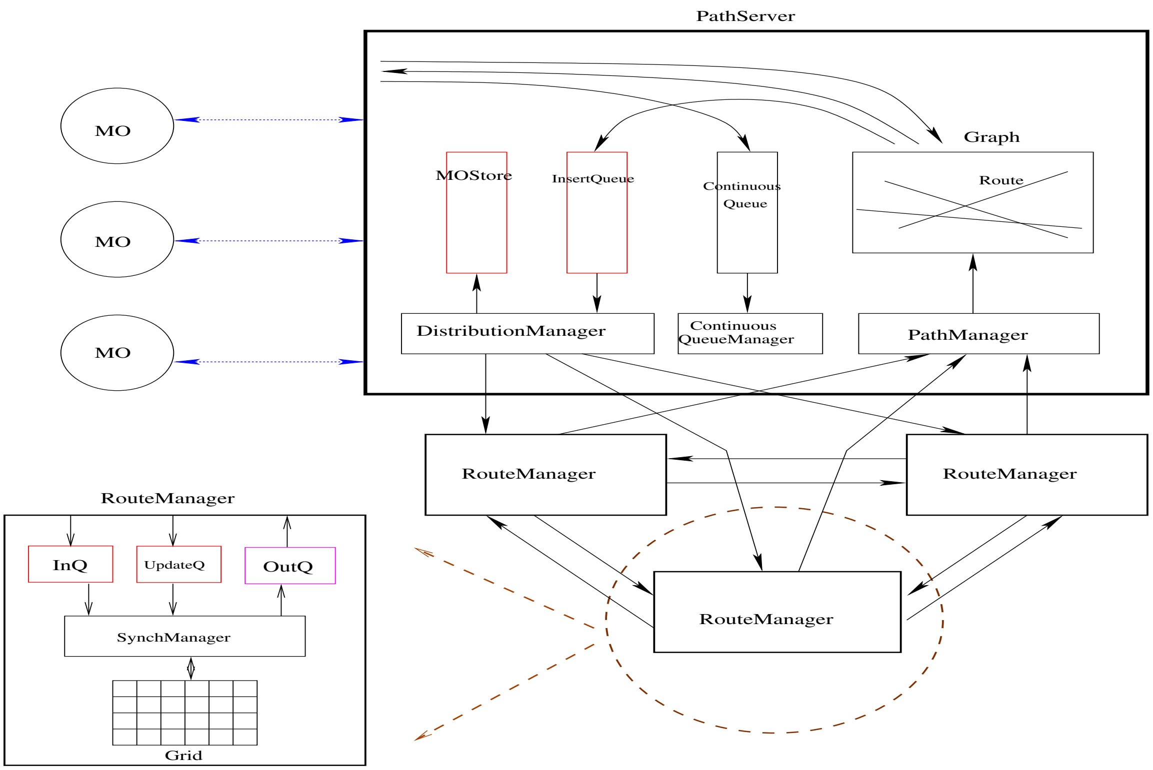

Figure 2: System Architecture

pate. Due to this, each RouteManager needs to communicate each other exchanging up-to-date information. We accomplish this by developing a SynchManager with an outgoing queue (OutQ) and an incoming queue (UpdateQ). The system is being developed using Java. PathServer and RouteManager are RMI Servers, possibly on different machines and any other Managers are threads. Figure 2 shows an overview of the system architecture.

5 Conclusion and Future Direction

The moving objects we have modeled in this paper are sizable physical entities following certain paths. Such paths have certain physical limitations and due to those limitations, the partitioning approach is better than the indexing approach to manage moving object information. In addition, we have observed that a real trajectory of a moving object is the result of interactions among moving objects in the system yielding a polyline instead of a line segment. We then built a system (RouteManager) based on a space-time grid that manages moving objects in a one-dimensional 1D space. Since in the real world moving objects would move on a two-dimensional network of one-dimensional lines, we extend the system to two-dimensional space as a network of one-dimensional components. The time-dependent shortest problem is an interesting application where such a system can be used. Also, it is of significant interest in transportation applications.

Since Sistla et al. [18] proposed a query model for moving objects, there has been much work in this field. Most of the early work [19, 21, 13, 8, 16, 2] did not take the following fact into account: the trajectory of a moving object is the result of the interactions among other moving objects yielding a polyline as a trajectory. Recently, however, Pfoser et al. [14], Chon et al. [3, 5], and Vazirgiannis et al. [20] worked on the assumption that the trajectory of a moving object is a polyline. One difference between [14] and [5, 20] is that in [14] the polyline is the result of recording of a moving object's past locations and only past information can be queried instead of the future. We believe that this assumption will be the basis for future research that will address issues of managing and querying future locations of moving objects. We also believe that the time-dependent shortest problem is a very interesting problem. Since several promising approaches have been proposed in the context of mobile data management, the research community needs to develop ways to evaluate different techniques. Also, research prototypes need to be build to determine the effectiveness of proposed applications.

Acknowledgements

This research was partially supported by the NSF under grant numbers EIA-0080134, EIA98-18320, IIS98-17432, and IIS99-70700 and the Caltrans-Testbed Center For Interoperability (TCFI).

References

[1] OnStar. http://www.onstar.com.

[2] H. Chon, D. Agrawal, and A. El Abbadi. Storage and Retrieval of Moving Objects. In *Proceedings of the Int. Conf. on Mobile Data Management*, pages 173–184, 2001.

[3] H. Chon, D. Agrawal, and A. El Abbadi. Using Space-Time Grid for Efficient Management of Moving Objects. In *MobiDE*, pages 59–65, May 2001.

[4] H. Chon, D. Agrawal, and A. El Abbadi. NAPA: Nearest Available Parking lot Application. In *Proceedings of the Int. Conf. on Data Engineering*, pages 496–497, 2002.

[5] H. Chon, D. Agrawal, and A. El Abbadi. Query Processing for Moving Objects with Space-Time Grid Storage Model. In *Proceedings of the Int. Conf. on Mobile Data Management*, 2002.

[6] C. Forum. Cellular Digital Packet Data System Specification, Release 1.1. Technical report, Jan. 1995.

[7] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 3(2):170–231, 1998.

[8] G. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Moving Objects. In *Proceedings of ACM Symp. on Principles of Database Systems*, pages 261–272, 1999.

[9] K. Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83:154–166, 1995.

[10] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

[11] S. Pallottino and M. Scutella. Shortest path algorithms in transportation models: classical and innovative aspects. In *In Equilibrium and Advanced Transportation Modelling, Kluwer*, pages 245–281, 1998.

[12] C. E. Perkins. Mobile IP. *IEEE Communications Magazine*, pages 84–99, May 1997.

[13] D. Pfoser and C. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *Proc. of the SSDBM Conf.*, pages 111–132, 1999.

[14] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 395–406, 2000.

[15] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of ACM*, 33(6):668–676, 1990.

[16] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 331–342, 2000.

[17] D. Schrank and T. Lomax. The 2001 Urban Mobility Report. Technical report, Texas Transportation Institute, 2001.

[18] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of the Int. Conf. on Data Engineering*, pages 422–432, 1997.

[19] J. Tayeb, O. Ulusoy, and O. Wolfson. A Quadtree Based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3):185–200, 1998.

[20] M. Vazirgiannis and O. Wolfson. A Spatiotemporal Model and Language for Moving Objects on Road Networks. In *7th International Symposium on Spatial and Temporal Databases*, pages 20–35, July 2001.

[21] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 111–122, 1998.