LOCUS: A Testbed for Dynamic Spatial Indexing

Jussi Myllymaki James Kaufman
IBM Almaden Research Center
{jussi,kaufman}@almaden.ibm.com

Abstract

We describe an extensible performance evaluation testbed for dynamic spatial indexing that is directly geared towards Location-Based Services (LBS). The testbed exercises a spatial index with several query types relevant for LBS: proximity queries (range queries), k-nearest neighbor queries, and sorted-distance queries. Performance metrics are defined to quantify the cost (elapsed time) of location updates, spatial queries, spatial index creation and maintenance.

The testbed is extensible with new spatial indexing methods, new query generators, and new index visualization methods. Synthetic but realistic, three-dimensional "moving object" data is generated with CitySimulator, a toolkit we have developed specifically for LBS research and made available on the IBM alphaWorks developer Web site.

1 Introduction

The DynaMark benchmark for dynamic spatial indexing is designed to evaluate the performance of middleware and database servers supporting Location-Based Services (LBS) [6]. Commonly referred to as "moving object databases," these systems manage continuously changing data representing the current, past, or future locations of mobile users. The high update load required of databases supporting Location-Based Services defines a new set of database requirements driving the need for new performance evaluation criteria. Queries typical of these systems include "find current position of user X," "find objects within distance Y from user X" and "find 10 nearest objects to user X."

In this paper, we describe the LOCUS Dynamic Spatial Indexing Testbed that provides a convenient way to conduct performance experiments on dynamic spatial indexing methods. The testbed was directly motivated by the need to evaluate the performance of systems supporting LBS applications. The testbed is designed to be extensible so as to easily include new spatial indexing methods, new location data sources (both simulated and real), new spatial query types, and new index visualization methods. It has been adapted to several commercial spatial data management systems as well as prototype systems, and includes a proximity query generator, a k-nearest neighbor query generator, and a sorted distance query generator for measuring spatial query performance.

The testbed is complemented by CitySimulator, a scalable data source that simulates a three-dimensional model city. CitySimulator generates realistic location data for large populations of mobile users. Combining the testbed with simulated location data, one can evaluate the performance of LBS middleware serving up to millions of moving objects. The system provides a powerful graphical user interface that allows the user to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

control several aspects of the simulation at runtime, including traffic flow, traffic congestion, and blocked streets and routes.

The paper is organized as follows. In Section 2, we review existing literature on location management, dynamic spatial indexing, dataset generation, and performance benchmarking. In Section 3, we briefly describe salient features of the DynaMark benchmark, and then introduce the LOCUS testbed in Section 4. Methods for analyzing the performance and scalability of spatial data management systems are discussed in Section 5.

2 Related Work

Location-based services build upon two well-established areas of information technology: wireless communication and spatial data management. While the primary role of wireless communication is to provide a connection between a mobile subscriber and others, its proper function – routing of data packets and voice calls – requires knowledge about the position of each mobile subscriber to some degree of accuracy.

New applications for real-time location information are now emerging. These applications are largely driven by the expanding use of mobile phones, but also the United States Federal Communications Commission's (FCC) mandate that the originating location of all emergency 911 calls from mobile phones in the United States be determined with some minimum accuracy [3]. Today, mobile phone operators have a choice of half a dozen technologies for location determination, each with distinct power consumption, handset compatibility, time-to-first-fix, and in-building coverage characteristics.

A spatial index for location-based services contains a very large number of simple spatial objects (e.g. points) that are frequently updated. These "moving object databases" pose new challenges to spatial data management [15]. The workload is characterized by high index update loads and relatively simple but frequent queries (e.g. range queries). A location update may merely contain the position of a user but it may also include the user's trajectory (direction and speed over time) [10, 7]. Supporting trajectories adds additional requirements to the index and query scheme [1]. A location update may also expire at a certain point in time [9] and is inherently imprecise due to inaccuracies in location determination technologies.

Several benchmarks exist for transaction processing in relational databases (TPC benchmarks), object-relational databases (BUCKY), and object-oriented databases (OO7), but to our knowledge no standard benchmarks exist for LBS applications and dynamic spatial indexing. The SEQUOIA 2000 benchmark [11] was motivated by Earth Sciences and focused on retrieval performance over large datasets of complex spatial objects. Several spatial access methods were compared in [5], but apart from an initial insertion cost, the benchmark focused on query performance. A qualitative comparison of spatio-temporal indexing methods appears in [13].

Use of realistic datasets is critical to gaining meaningful data from performance evaluation experiments [2]. Most LBS experiments to date have used spatial data generated by random walks or algorithms using simple distribution functions. For instance, [5] looked at six random distributions for spatial data: diagonal, sinus, bit, x-parallel, clusters, and uniform distributions. A few generators have been proposed for creating spatio-temporal data. The GSTD generator [14] creates data for randomly moving points and rectangles. In [12], GSTD was extended to account for buildings and other "infrastructural" obstructions, but it still allowed objects to move free of influence from other moving objects.

The Oporto toolkit [8] describes an elaborate scheme with moving point objects (fishing boats), moving extended objects (schools of fish), stationary extended objects that change size (areas with plenty of plankton that attract fish, and storms that boats try to avoid), and static extended objects (harbors). Another approach for creating semantic moving object datasets is described in [9]. The dataset was created by distributing a set of destinations uniformly in a coordinate space and connecting every pair of destinations with two one-way roads. Moving objects (cars) are placed on roads randomly and move with a randomly assigned speed until a destination is reached, at which point a new destination is picked.

3 DynaMark Benchmark for Dynamic Spatial Indexing

The DynaMark benchmark measures the performance and scalability of a spatial data management system [6]. The benchmark executes a set of standard spatial queries against a set of standard location trace files. Performance metrics consist of the cost of updating a user's location in the spatial index and the cost of spatial queries against the index. The spatial queries are motivated by LBS applications but are similar to those found in traditional Geographic Information Systems (GIS). The difference is that the data being queried is highly dynamic and the queries are issued extremely frequently and have a dynamic reference point (a user's location).

The size of an individual benchmark run is determined by the number of mobile users contained in the location trace file and ranges from 10,000 to one million or above. Our trace files are generated by CitySimulator, a toolkit we have developed specifically for LBS research (see Section 4.4 for details). Each record in the location trace file represents a location update that contains the following information: object ID (identifies mobile user), timestamp (indicates the time when the location was determined or reported by the user), and X, Y, and Z coordinates of the location. The X and Y coordinates represent the longitude and latitude values of the location, and Z indicates elevation in meters.

The benchmark defines three types of queries: proximity queries, k-nearest neighbor (kNN) queries, and sorted-distance queries. These queries are typically centered around the location of a user issuing the request. A *proximity query* finds all objects that are within a certain range. The range condition may vary along different axes hence the query forms a 2D ellipse or 3D ellipsoid depending upon the topological dimension of the space. Alternatively, the range can be expressed by forming a 2D rectangle or 3D ortho-rhombus.

A *kNN query* finds the k nearest objects. The query may search other mobile users or it may search stationary objects such as coffee shops, gas stations, hospitals, etc.

A *sorted-distance query* lists each object in increasing distance order relative to the reference point. This is like a k-nearest neighbor query where k is the total number of objects, but the difference is that during query evaluation the number of objects inspected by the application is unknown (otherwise the system could execute it efficiently as a kNN query). Also, the result must include distance information.

4 LOCUS Dynamic Spatial Indexing Testbed

The LOCUS testbed provides a convenient way to run performance experiments on spatial data management systems and yield performance data that conform to the DynaMark benchmark specification. The testbed is written in portable C and has been tested on several platforms, including Windows, AIX, and Solaris. We are currently in the process of making the testbed code available on the IBM alphaWorks developer Web site.

4.1 Extensibility

The testbed is extensible with new spatial indexing methods, new query generators, and new index visualization methods (Figure 1). The testbed defines a C API for each of the extension types, using function pointer arrays to achieve something analogous to the Java "interface" concept.

The task of an *indexing method extension* is to provide a spatial indexing capability and support API calls such as "create index," "delete index," "insert index entry," and "search index." In Section 4.3 we list several indexing method extensions that are built into the default testbed. We distinguish between two types of indexing method extensions: native extensions and adapter extensions. A native indexing method extension implements the required spatial indexing capability by itself, without relying on a database system. An adapter extension converts the C API calls to calls in another system, for example SQL statements that run against a database.

A *query generator* creates one or more spatial queries, typically using the location of an existing user as a reference. The testbed has three built-in query generators, corresponding to the query types defined by the benchmark: proximity queries, k-nearest neighbor queries, and sorted-distance queries. The parameters needed

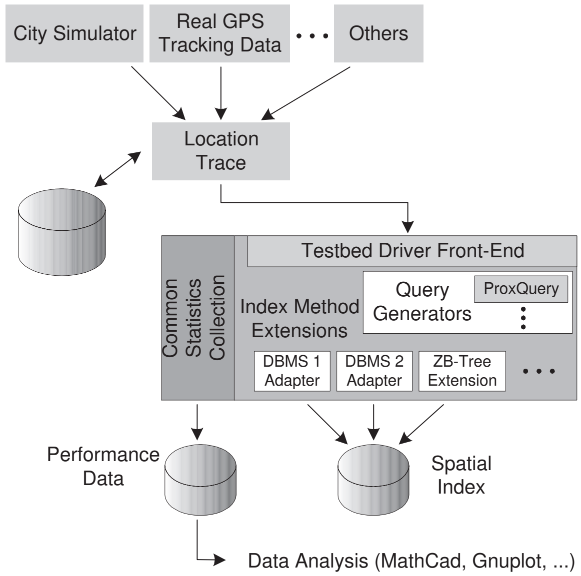


Figure 1: Architecture of LOCUS testbed.

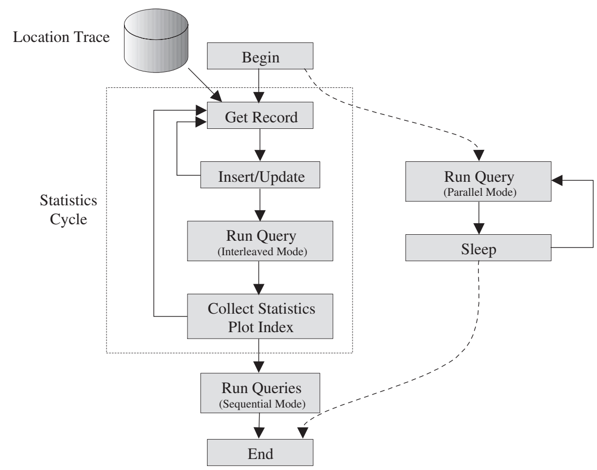


Figure 2: Update and query flow in the testbed.

by the query generators (e.g. the range value in proximity queries or the value of k in kNN queries) are defined in a configuration file. The same query generator code works in all query modes defined earlier.

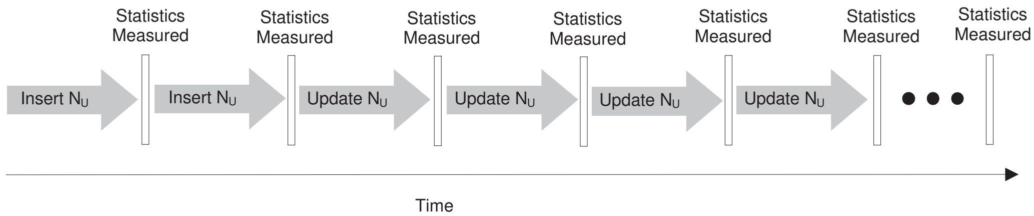
An *index visualization method* provides a simple way for the testbed to plot an index (e.g. minimum bounding rectangles of an R-tree index) using lines and rectangles, multiple colors or line types, and labels. Several index visualization methods are built into the testbed by default. The "gnuplot" visualization method produces files in GIF, Postscript, DXF, and other graphics formats. Using external tools, static GIF images can be converted to animations that show the evolution of the index as a function of time. An "OGS" visualization method outputs ST_Polygon geometries that can be loaded into any OpenGIS-compatible spatial database and analyzed with its visual data exploration tools. The testbed also provides a "generic" visualization method that outputs the index geometries as comma-separated values (CSV).

4.2 Update and Query Flow

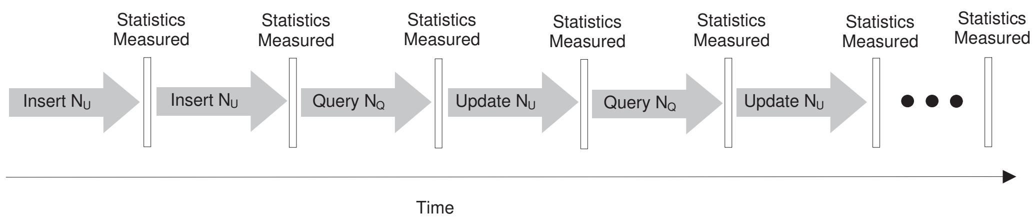
Figure 2 illustrates the general process flow of the testbed. The basic principle is that after every n updates to the index, the testbed runs m queries. The process repeats until the entire location trace file has been processed, or until a maximum experiment time has been reached.

The number of updates n and queries m executed in an iteration depend on the *query mode*. In the *immediate mode*, one spatial query ($m = 1$) is issued after every update ($n = 1$), with the position of the update being used as the reference point. In the *interleaved mode*, the values n and m are configured to some values N_U and N_Q, respectively. After processing N_U updates, the testbed runs N_Q queries where for each query a different, random user is picked as the reference point. In the *sequential mode*, the index is "loaded" by processing the first iteration (initial locations) of the location trace file and then queried by N_Q queries as in the interleaved mode. The *parallel mode* executes queries simultaneously with updates. A time parameter specifies the frequency of queries, and for each query a random user is picked as the reference point.

We generally prefer to run experiments in the interleaved mode because it provides a way to gather consistent, repeatable update and query cost measurements as the index grows and updates are performed. The execution plan in the interleaved mode is further illustrated in Figure 3. To minimize the impact of queries on updates, we conduct two experiments for each population size and indexing method. In an UPDATE experiment,



A) Flow of UPDATE Experiment



B) Flow of QUERY Experiment

Figure 3: Execution plan for updates, queries, and statistics collection in the testbed.

only updates are performed and no queries are executed. This gives us the minimum response time of individual index updates. In a QUERY experiment, we run updates and queries in the interleaved mode, but only inspect query performance data. Again, this is to minimize the interference between updates and queries. In Section 4.5 we describe how the independently-measured update and query cost metrics are combined into a total cost estimate and extrapolated to different population sizes to estimate maximum population sizes supported by the system under test.

4.3 Built-In Indexing Method Extensions

Most commercial database systems can be extended with a spatial data component, either provided by the vendor itself or by a third party. Some of the more popular spatial indexing methods these extensions employ include grid indices, R-trees, and Quadtrees (see [4] for a comprehensive survey). Some commercial solutions also make use of space-filling curves such as Z-order curves and Hilbert curves. It is fairly straightforward to use one of these existing spatial indexing methods for LBS applications. However, most of them were not designed to support the high update frequencies required by LBS applications. In fact, some of them perform poorly when faced with dynamically changing spatial data. Clearly, a controlled comparison between various spatial indexing schemes is required to assure satisfactory performance in LBS applications.

The testbed has been adapted to use most commercial database systems and their spatial extensions, including IBM DB2 with Spatial Extender, Informix Dynamic Server with Spatial DataBlade, Oracle 9i with Spatial Cartridge, and ESRI ArcSDE. We have also developed a generic Z-order indexing method that runs on top of any SQL database system.

The testbed has also been adapted to work with main-memory indexing methods. Two main-memory extensions are included by default: a naive array extension and a ZB-tree extension. The array extension implements a combination of a simple array and hash table which work together to record the position of moving objects. Updates into this data structure are extremely fast but queries obviously perform poorly. In contrast, the ZB-tree calculates Z-order values in the extension code and stores them in a binary tree. We used a readily available implementation of binary trees, namely the `libavl` library that implements several varieties of them. The

ZB-tree extension stores Z-order values in a right-threaded AVL-tree which performs very well for proximity queries (requiring only a left-to-right traversal of the leaf nodes of the tree).

4.4 Location Simulation

Location trace files for the testbed are produced by CitySimulator, a complementary tool available for download at the IBM alphaWorks developer Web site at http://alphaworks.ibm.com. CitySimulator is a scalable, three-dimensional model city that simulates an arbitrary number of mobile users moving about in a city, driving on streets or walking on sidewalks and entering buildings, where they move up and down the floors and stay on a floor some amount of time before returning to the streets and finding a new place to go to (or even leaving the city). We typically simulate population sizes ranging from 10,000 users up to 1 million users and use an average location report periodicity of 30 seconds.

Simulation control parameters allow easy creation of realistic events such as daily commutes. A graphical user interface allows observation of the simulation. Advanced settings allow exploration of efficiency of indexing algorithms over daily commute cycles. The output of the simulation can be viewed in real time; output is also produced as a comma-separated variable (CSV) text file, which contains a unique person ID, a time stamp, and spatial coordinates.

Custom city plans can be created as XML documents and imported into the simulator. The simulator also provides Java interfaces and abstract classes to facilitate extensions capable of generating city plans from, for example, real GIS data. A Java properties file allows easy configuration of a wide variety of program parameters. The GUI allows change of parameters affecting motion rules in real time. The simulator also includes an optional traffic flow model which causes the traffic flow to take on the characteristics of a compressible fluid. When this feature is enabled, obstructions to traffic flow can cause shock waves or traffic jams.

Simulated cities are constructed with collections of various Java objects. The high level objects in the simulation are “Place” and “Person.” Objects of type Building, Road, Intersection, and Floor all extend Place. Places have several attributes: coordinates, extents, altitude or floor number, and pointers to neighboring places. Places also contain enter and exit probabilities, up down probabilities, drift probabilities (on roads), scatter probabilities (on intersections), etc.

People move according to rules based on the place they are in. A person on a building floor does a random walk. At specific points (stairways) they may move up or down (if not at the top floor or ground floor respectively) or leave a building (if near a door). A person on a road moves with a linear combination of *Velocity = Random Walk Component + Drift Velocity*. Hence the magnitude of the drift velocity increases as a person moves closer to the center of a road. For two-way streets, the direction of the drift velocity changes sign at the center (so on one side of a road people move North or East, on the other side they move South or West). Road objects also have orientations that determine if the drift velocity is North/South or East West.

4.5 Methods for Performance Analysis

To estimate the maximum population size supported by a spatial data management system, we fitted a function $f(P) = a \cdot P + b$ to the measured update and query costs of the system. Parameter P is population size and $f(P)$ is the estimated cost of the database update or query for that population size. We then produced a closed-form equation that relates parameters a and b to P and i, where i is the average request interval in a real-time location tracking application. We assume that the workload consists of a sequence of database requests where a given request is an update with probability p and query with probability $1-p$. In a system where moving objects report their location every 5 minutes and every update is followed by a query, the values are $i = 300$ and $p = 0.5$.

In order for a location tracking application to function in real time, it holds that the total number of requests handled in time period i must have a lower overall cost than i itself. The closed-form equation representing this requirement is $P \leq \sqrt{b^2/4a^2 + ip/a} - b/2a$.

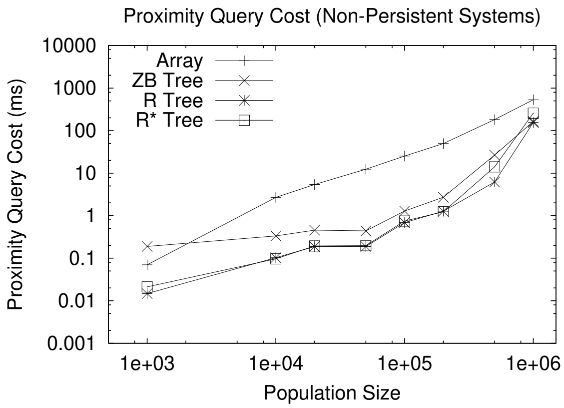


Figure 4: Actual cost of proximity queries for main-memory indexing methods.

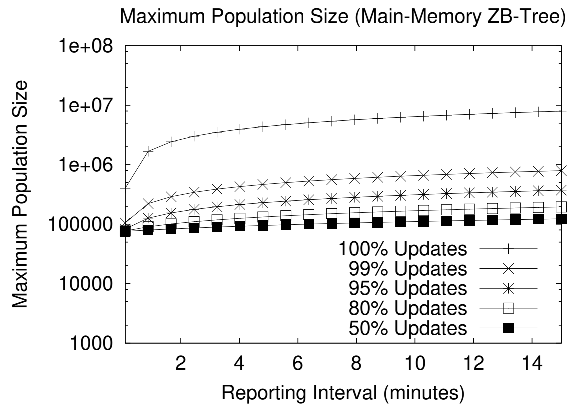


Figure 5: Maximum population size supported by a main-memory ZB-tree.

In Figure 4, we show performance results gathered by the testbed on several main-memory indexing methods running on a 1 GHz, 1 GB RAM server. The two variations of R-trees we tested performed best on all population sizes, although with the highest population size (1 million moving objects) the advantage appeared to all but evaporate. The ZB-tree performed almost equally well, except for the smallest population sizes. As expected, the naive array/hash table method was 1-2 orders of magnitude more expensive than the other methods.

Figure 5 shows the maximum population size supported by the main-memory ZB-tree method under various update/query workloads. When the workload consisted of updates only and each moving object reported its location every 5 minutes, the indexing method could handle roughly 5 million moving objects. The high capacity was made possible by the low update cost of the ZB-tree shown in our experiments, less than 50 microseconds per update. The maximum population size dropped quickly as the fraction of queries increased. With a 99% update ratio, maximum population size was 0.5 million, and with 95% updates it dropped to 0.3 million. With higher query loads the maximum population size settled at the 100,000 to 250,000 range.

5 Conclusion

We have described the LOCUS testbed for dynamic spatial indexing that is a generic platform for running controlled performance experiments on any spatial indexing method and query generator. The testbed is easy to extend with new spatial indexing algorithms and query types. The testbed currently supports three spatial query types common to LBS applications, and provides adapters for several commercial database systems and their spatial extensions. Native indexing methods that do not use a database system, e.g. memory-resident tree indices, can also be plugged into the testbed. The testbed also provides an index plotting capability for visualizing the internal structure of a spatial index (e.g. minimum bounding boxes of an R-tree) and producing portable GIF animations that are useful for analysis and development.

The testbed is complemented by CitySimulator, a scalable, three-dimensional model city that generates realistic location data for large populations of mobile users. It provides a powerful graphical user interface that allows the user to control several aspects of the simulation at runtime, including traffic flow, traffic congestion, and blocked streets and routes. A visualization display shows the progress of the simulation in real time. New city models can be imported as XML documents which describe the layout of the city, including roads, buildings, and vacant space.

References

[1] Pankaj K. Agarwal, Lars Arge, and Jeff Erickson. Indexing moving points. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 175–186, 2000.

[2] Thomas Brinkhoff. Generating network-based moving objects. In *Proceedings of the International Conference on Statistical and Scientific Database Management (SSDBM)*, pages 253–255, Berlin, Germany, July 2000.

[3] FCC Docket 94-102. FCC adopts rules to implement enhanced 911 for wireless services, June 1996. http://www.fcc.gov/e911.

[4] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[5] Hans-Peter Kriegel, Michael Schiwietz, Ralf Schneider, and Bernhard Seeger. Performance comparison of point and spatial access methods. In *Proceedings of the International Symposium on the Design and Implementation of Large Spatial Databases (SSD)*, pages 89–114, Santa Barbara, CA, 1989.

[6] Jussi Myllymaki and James Kaufman. DynaMark: A benchmark for dynamic spatial indexing, March 2002. Submitted for publication.

[7] Dieter Pfoser and Christian S. Jensen. Querying the trajectories of on-line mobile objects. In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*, Santa Barbara, CA, May 2001.

[8] Jean-Marc Saglio and Jose Moreira. Oporto: A realistic scenario generator for moving objects. In *Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA)*, pages 426–432, 1999.

[9] Simonas Saltenis and Christian S. Jensen. Indexing of moving objects for location-based services. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, San Jose, CA, February 2002.

[10] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and querying moving objects. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 422–432, Birmingham, UK, April 1997.

[11] Michael Stonebraker, James Frew, Kenn Gardels, and Jeff Meredith. The SEQUOIA 2000 storage benchmark. In *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, pages 2–11, Washington, DC, June 1993.

[12] Yannis Theodoridis. Generating semantics-based trajectories of moving objects. In *Proceedings of the International Workshop on Emerging Technologies for Geo-Based Applications*, Ascona, Switzerland, May 2000.

[13] Yannis Theodoridis, Timos K. Sellis, Apostolos Papadopoulos, and Yannis Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *Proceedings of the International Conference on Statistical and Scientific Database Management (SSDBM)*, pages 123–132, 1998.

[14] Yannis Theodoridis, Jefferson R. O. Silva, and Mario A. Nascimento. On the generation of spatiotemporal datasets. In *Proceedings of the International Symposium on Spatial Databases (SSD)*, pages 147–164, Hong Kong, China, July 1999.

[15] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving objects databases: Issues and solutions. In *Proceedings of the International Conference on Statistical and Scientific Database Management (SSDBM)*, pages 111–122, Capri, Italy, July 1998.