LNCS 3882, pp 125-140, 2006.
(URL: http://www.springerlink.com/link.asp?id=52j5u51548852766)

The COST Benchmark—Comparison and Evaluation of Spatio-temporal Indexes

Christian S. Jensen, Dalia Tiešytė, and Nerius Tradišauskas

Department of Computer Science, Aalborg University, Denmark
{csj, dalia, nerius}@cs.aau.dk

Abstract. An infrastructure is emerging that enables the positioning of populations of on-line, mobile service users. In step with this, research in the management of moving objects has attracted substantial attention. In particular, quite a few proposals now exist for the indexing of moving objects, and more are underway. As a result, there is an increasing need for an independent benchmark for spatio-temporal indexes.

This paper characterizes the spatio-temporal indexing problem and proposes a benchmark for the performance evaluation and comparison of spatio-temporal indexes. Notably, the benchmark takes into account that the available positions of the moving objects are inaccurate, an aspect largely ignored in previous indexing research. The concepts of data and query enlargement are introduced for addressing inaccuracy. As proof of concepts of the benchmark, the paper covers the application of the benchmark to three spatio-temporal indexes—the TPR-, TPR*-, and B^x-trees. Representative experimental results and consequent guidelines for the usage of these indexes are reported.

1 Introduction

With the availability of mobile computing technologies, geo-positioning, and wireless communication capabilities, it has become possible to accumulate the changing locations of populations of moving objects in real time. Consumer electronics are affordable, current Global Positioning System (GPS) [1] receivers are capable of geo-positioning with an accuracy of up to a few meters, the General Packet Radio Service (GPRS) [2] and similar technologies have become common and relatively cheap means of wireless data transfer. It is thus possible for an object to continually obtain and transmit its current position to a central server.

Applications are emerging that require or may benefit from the tracking of the locations of moving objects. These occur in areas such as logistics, traffic management, public transportation, and location-based services. Current applications usually track only relatively small numbers of objects, but as the underlying technologies continue to improve, applications that concern large numbers of objects are on the horizon.

The increasing interest in mobile location data has served as motivation for the development of spatio-temporal indexes for the current and near-future positions of moving objects. A number of spatio-temporal indexes have been proposed, such as R-tree-based indexes, e.g., the TPR-tree [3], the TPR*-tree [4], the STAR-tree [5], and the R^{EXP}-tree [6]; the quadtree-based index STRIPES [7], and the B^+-tree-based B^x-tree [8], to name but a few.

M.L. Lee, K.L. Tan, and V. Wuwongse (Eds.): DASFAA 2006, LNCS 3882, pp. 125–140, 2006.

This continuing proliferation of indexing techniques creates a need for a standard procedure for performance evaluation and comparison. Although mathematical complexity analysis is valuable, empirical evaluation [9] is indispensable for evaluation and comparison of spatio-temporal indexing techniques. The current state of affairs is that indexes being proposed are being evaluated empirically and are being compared to, typically, one other indexing technique. The empirical studies reported are rarely exhaustive and, not surprisingly, tend to focus on the favorable qualities of the index being proposed. The availability of an independent benchmark specification establishes an equal footing for obtaining experimental results and enables broader comparison.

This paper proposes a benchmark specification, termed COST, for the evaluation and comparison of spatio-temporal indexes. The benchmark is independent in the sense that it is proposed independently of a specific indexing technique. The benchmark aims to provide a unified procedure that covers an extensive variety of possible and realistic settings. In particular, the benchmark evaluates the index ability to accommodate uncertain object positions. Queries and updates are considered, as are both I/O and CPU performance.

The remainder of this paper is outlined as follows. Related work is covered in Sect. 2. The addressed indexing problem is detailed in Sect. 3. Sections 4 and 5 contain the benchmark specification. As proof of concepts, Sect. 6 reports on experimental results that were obtained using the benchmark. Section 7 concludes and offers directions of future work.

2 Related Work

We cover in turn existing benchmarks for spatio-temporal data, previous work on the indexing of uncertain data, and past empirical evaluations of spatio-temporal indexes.

A number of benchmarks exist that measure transaction performance in traditional database systems. For example, a set of benchmarks that evaluate system performance and price is provided by Gray [10]. However, these benchmarks are not applicable to spatio-temporal data.

Of relevance to moving objects, Theodoridis [11] provides a benchmark that includes a database description and 10 non-predictive queries for the static and moving spatial data. Myllymaki and Kaufman [12] also propose a benchmark, DynaMark, for moving objects. The query and update performance measure is CPU time, as a main-memory resident index is assumed. Future queries on anticipated future locations are not considered. Werstein [13] proposes a benchmark for 3-dimensional spatio-temporal data. The benchmark is oriented towards general operating system and database system performance comparison, including evaluation of the spatio-temporal and 3-dimensional capabilities. Tzouramanis et al. [14] perform an extensive, rigorous experimental comparison of four types of quadtree-based spatio-temporal indexes, using a benchmark specification when performing experiments with the four indexes. However, their proposal concerns raster data, generated with the G-TERD benchmark tool.

The concept of data uncertainty for moving object positions has previously been studied quite extensively (see, e.g., [15, 17, 18, 19]). While the bulk of this work has

been conducted independently of indexing, some works (see, e.g., [16, 19]) offer insights into the indexing of uncertain positions. The present paper goes further by proposing a simple and yet effective method for storing and retrieving position data with accuracy guarantees. Existing indexes can straightforwardly be extended to accommodate such data.

Many authors of spatio-temporal indexes have compared their indexes to usually one other competitive index (e.g., [3, 4, 7, 8]). However, these comparisons tend to focus on exploring the properties of the new index being proposed; and with the new index being the main topic, the experimental specifications are relatively limited and lack independence.

The benchmarks covered above consider neither uncertain data nor accuracy guarantees. DynaMark shares similarities with COST benchmark with respect to the generated traffic data, but it lacks aspects to do with future positions. To the best of our knowledge, no independent benchmark exists that has been designed specifically for the evaluation of disk-based indexes for the current and near-future uncertain positions of moving objects.

3 Spatio-temporal Indexing

This paper is concerned with the indexing of large amounts of current and near-future, 2-dimensional moving object positions, and predictive queries are of interest. In this setting, position data are received from continuously moving objects capable of reporting their position and velocity. Mobile applications—e.g., those that provide location-enabled services to mobile users—issue queries on this data.

3.1 Spatio-temporal Data and Queries

The objects, represented as 2-dimensional points, update their positions periodically. As the server is recording the positions of a large amount of objects, updates should occur as rarely as possible. The current and anticipated future positions of the objects can be queried at any time. Therefore, continuous function that approximates the actual object movements and enables predictive queries is derived from the position data received.

An appropriate approximation function should satisfy the following requirements: (1) the parameters of the function can be obtained from the moving object; (2) the function reduces the amount of updates; (3) predicted positions are helpful in answering predictive queries; and (4) the function is easy to compute and its representation is compact.

It is common to predict an object's near-future position using a linear function of time [3, 4, 7, 8]. An object's position at time t is denoted by a 2-dimensional vector $\overrightarrow{P}$, and its velocity is given by a 2-dimensional vector $\overrightarrow{V}$. The function takes time as an argument, and returns the object's position:

$$\overrightarrow{P}(t) = \overrightarrow{P}(t_{\mathrm{up}}) + \overrightarrow{V}(t_{\mathrm{up}})(t - t_{\mathrm{up}}) \tag{1}$$

Here t_{up} is the time of the last update, at which the object's position was $\overrightarrow{P}(t_{\mathrm{up}})$; $\overrightarrow{V}(t_{\mathrm{up}})$ is the velocity at time t_{up}, and $\overrightarrow{P}(t)$ is the predicted position at time t.

This function may be represented as a tuple $(\overrightarrow{P}(t_{\mathrm{ref}}), \overrightarrow{V}(t_{\mathrm{up}}))$, where time t_{ref} is an agreed upon, global reference time at which the object's position is stored. When an update of an object arrives at time t_{up}, its position $P(t_{\mathrm{ref}})$ at time t_{ref} is calculated using (1).

The linear function satisfies the four requirements for the approximation function. Velocity and position values are easy to obtain—they are output by GPS receivers [1], and the velocity can also be estimated based on previous positions (first requirement). The function's value is calculated in a constant time, and the representation is compact (fourth requirement). Studies show that using this function for vehicle positions, the average number of updates is reduced by more than a factor of two for accuracy thresholds below 200 meters, in comparison to the standard approach where the current position is assumed to be given by the most recently reported position [17] (second requirement). Finally, linear prediction offers better approximations of near-future positions than does constant prediction, yielding more reasonable answers to predictive queries (third requirement).

Three types of queries that a spatio-temporal index should support can be distinguished [3]. Let t, t_1, and t_2 be time points and let q_{r}, q_{r_1}, and q_{r_2} be 2-dimensional rectangles.

Q1. Timeslice query $Q1 = (q_{\mathrm{r}}, t)$ returns the objects that intersect with q_{r} at time t.

Q2. Window query $Q2 = (q_{\mathrm{r}}, t_1, t_2)$ returns objects that intersect with q_{r} at some time during time interval $[t_1, t_2]$. This query generalizes the timeslice query.

Q3. Moving window query $Q3 = (q_{\mathrm{r}_1}, q_{\mathrm{r}_2}, t_1, t_2)$ returns the objects that intersect, at some time during $[t_1, t_2]$, with the trapezoid obtained by connecting rectangles q_{r_1} and q_{r_2} at times t_1 and t_2, respectively. This query generalizes the window query.

Figure 1 offers an example encompassing four objects and three queries in 1-dimensional space. The *arrows* in the figure represent object movement.

The queries $q1$, $q2$, and $q3$ are timeslice, window, and moving window queries, respectively. Query $q3$ has spatial ranges $q3_{r_1} = [-20, -10]$, $q3_{r_2} = [-25, -10]$, and time range $[5, 6]$. The result of the query depends on when the query is issued. If issued before time $t = 3$, the result is $\{o1, o4\}$. Otherwise, the result is $\{o4\}$. Object $o1$ is updated at time 3 and its predicted trajectory changes. Its new trajectory does not intersect with the query.

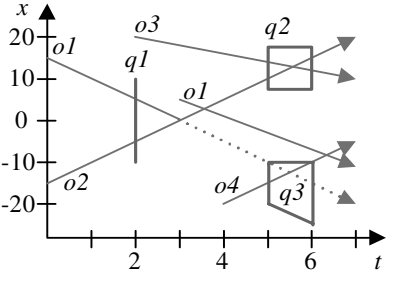

Fig. 1. Example of objects and queries in a 1-dimensional space

3.2 Update Policies

The inaccuracy of the moving object positions available at the server side stems from two sources. The positions measured by the moving objects (e.g., using GPS) are inaccurate, and the use of sampling introduces inaccuracy. Because the measurement

inaccuracy is much smaller than the sampling inaccuracy in a typical setting, we assume that the measurements are accurate and focus on the inaccuracy due to sampling.

In particular, we assume an approach where, at any point in time, the actual position of an object deviates from the position assumed on the sever side, the predicted position, by no more than a chosen distance threshold thr. An update policy should be adopted that satisfies the accuracy guarantee with as few updates as possible.

The so-called *point-based* update policy requires an object to issue an update when the distance between the object's current and its most recently reported positions reaches the threshold value. With this policy, the server assumes that an object remains where it was when it most recently reported its position. Frequent updates result.

To reduce the cost of updates a *vector-based* policy may be adopted [17], where each moving object shares a linear prediction, as given by (1), of its position with the server. When the distance between an object's actual and predicted positions exceeds the distance threshold thr, the object issues an update to the server. The point-based policy is the special case of the vector-based policy, where the linear prediction function is constant ($\overrightarrow{V} = \overrightarrow{0}$, where $\overrightarrow{0}$ is the zero vector).

The point-based update policy is shown in Fig. 2 (a). Here, the position $\overrightarrow{P}(t_i)$ is updated at time t_i, and the actual position remains in the circle with center $\overrightarrow{P}(t_i)$ and radius thr for some time, yielding a predicted position of $\overrightarrow{P}(t_i)$. At time t_{i+1}, the difference between the actual and predicted positions reaches thr, and an update is issued.

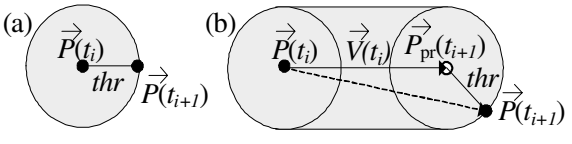

Fig. 2. Point-based (a) and vector-based (b) update policies with accuracy threshold thr

Next, the vector-based policy is illustrated in Fig. 2 (b). First, at time t_i, the object reports its actual position $\overrightarrow{P}(t_i)$ and velocity $\overrightarrow{V}(t_i)$ to the server. The server's prediction is illustrated by the *solid horizontal vector*. The object shares this prediction with the server. In addition, it repeatedly compares its actual position with the predicted position $\overrightarrow{P}_{\mathrm{pr}}$. When at time t_{i+1}, the object's position is $\overrightarrow{P}(t_{i+1})$, the distance between the two positions is thr, and an update is generated. Updates are sent only when needed in order to maintain the accuracy guarantees.

As discussed in Sect. 3.1, the vector-based policy yields fewer updates than the point-based policy for the same accuracy guarantees and therefore is preferable.

3.3 Query and Data Enlargement

The notions of *precision* (p) and *recall* (r) [20] are commonly used for measuring the accuracy of a query result. The precision is the fraction of the objects in the result that actually satisfy the query predicate, and the recall is the fraction of the objects that satisfy the query predicate that are in the query result. Ideally, $p = r = 1$, meaning that the query result contains exactly the objects that satisfy the query.

However, the data are inaccurate—the positions of the objects are only known with accuracy thr. It is thus not possible to achieve $p = r = 1$; however, perfect recall can be achieved[1] and is a desirable requirement for an index. Thus, the query result is guaranteed to contain all objects that may satisfy the query predicate.

To achieve prefect recall, it is necessary to take the inaccuracy of the predicted positions into account. This may be done by means of either data or query enlargement.

Query enlargement addresses position inaccuracy by expanding the query area by thr in all directions. If different objects have different thresholds, the maximum threshold must be used. Perfect recall is achieved as all the objects that are actually in the query area have predicted positions that are no further than thr away from their actual positions.

The "fattened" query rectangle may be obtained as the Minkowski sum [21] of the two sets. Each point p_{q} that belongs to the query rectangle q_{r} is added to each point p_{s} that belongs to the segment s of length thr:

$$q_{\mathrm{r}} \oplus s = \{p_{\mathrm{q}} + p_{\mathrm{s}} | p_{\mathrm{q}} \in q_{\mathrm{r}} \wedge p_{\mathrm{s}} \in s\}$$

Figure 3 (a) shows query enlargement in a 2-dimensional space.

Next, with data enlargement object positions are expanded into spatial regions with extent. In particular, an object's position becomes a circle with radius thr, instead of being a point. The center of the circle is the predicted position. The object's actual position is always inside the circle. If the circle intersects with the query area, the object must be included in the query result. Figure 3 (b) illustrates data enlargement. The shaded area denotes the movement of the object.

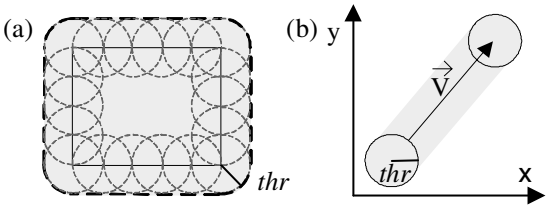

Fig. 3. Example of query (a) and data (b) enlargement

A spatio-temporal index should support either query or data enlargement. However, existing indexes tend to ignore position inaccuracy and simply assume that they know the exact position of each object, meaning that $thr = 0$. Such indexes must be adjusted to index positions with non-zero threshold values.

4 Benchmark Data and Settings

The workload for an index consists of a sequence of the updates and queries. The benchmark specification contains definitions of workloads and procedures of using them. The desired properties of the workloads and workload generation are discussed first. Definitions of benchmark procedures, termed *experiments*, then follow.

[1] We note that perfect recall for queries that concern future times is only possible when updates that occur between the time a query is issued and the future times specified in the query cannot affect the query result.

4.1 Workload Parameters

A set of update and query parameters defines the benchmark workloads. The workloads aim to simulate a wide range of situations in which an index may be used. The following parameters are of interest:

Number of Objects. The number of objects largely determines the size of the index and may be used to examine the scalability of the index.

Position and Velocity Skew. These parameters determine the distribution in space of the object positions and velocities. They are highly related, as velocity skew leads to position skew. An example of skew is the concentration of stationary vehicles in the suburbs at night and in business districts during working hours, and many moving vehicles during the morning and afternoon rush hours.

Update Arrival Pattern. The rate of updates depends on the chosen update policy as described in Sect. 3.2. With the vector-based policy, the durations in-between updates vary greatly. The update frequency depends on the movement trajectories and speeds of the objects. This parameter allows examination of how an index accommodates different frequencies of updates.

Position Accuracy Threshold. The distance threshold thr (defined in Sect. 3.2) affects the update arrival rate and the query or data extents. By varying this parameter, the index ability to support various update frequencies as well as data and query sizes can be studied.

Query Parameters. The required query types, their spatial and temporal extents and their time intervals are the query parameters of interest. The types of queries considered are described in Sect. 3.1.

Workload Duration. The workload duration is measured as a number of updates executed by the index. This parameter allows examination of how an amount of updates affects the performance of an index.

4.2 Workload Generator

The workloads in the COST benchmark are generated using a workload generator that extends the generator by Šaltenis et al. [22]. That generator was chosen as the starting point because it is capable of easily creating workloads according to many of the parameters discussed in Sect. 4.1 and because it is fast in comparison to such generators as CitySimulator [23, 24] and GSTD [25, 26], which use complex functions, e.g., functions that control the interactions among the objects. We proceed to explain the original generator, then describe the extensions implemented.

A workload intermixes queries and updates with a chosen proportion. An index is then subjected to these operations. In the generator, object movement is either random or network-based. To accommodate the latter, a number of "hubs" with random positions and links between these form a complete, bi-directional, spatial graph. Objects move between hubs until the end of a simulation. The maximum speed of an

object is chosen randomly from a set of maximum speeds. An object accelerates and decelerates when moving from one hub to another. Updates are generated in average intervals of $UpdateInterval$ time durations. For any kind of data, these parameters can be set:

$Objects$. Total number of moving objects.
$Space$. The extent of the space where the objects are moving.
$Speed_i, i = 1, ..., 50$. Possible maximum speeds of the objects. For each object, its maximum speed is chosen at random.
$TotalUpdates$. The number of update operations performed in the simulation.
$UpdateInterval$. The average duration between two successive updates of an object.
$Hubs$. The number of destinations between which the objects are moving. Value 0 implies uniform (random) distribution.
$QuerySize$. The maximum spatial extent of a query in percentages of the indexed space.
$QueryTypes$. The fractions of timeslice, window, and moving window queries (see Sect. 3.1). The sum of the three fractions must add up to 1.
$QueryTime$. The maximum temporal extents of window and moving window queries.
$QueryWindow$. The maximum duration of time that queries may reach into the future.
$QueryingInterval$. Querying frequency relative to update operations.
$QueryQuantity$. The number of queries generated at each query generation event.

The generator was extended, enabling it to choose between its original update policy and the vector-based policy (as described in Sect. 3.2). The original policy was extended so that it is able to randomly select a different update interval for each object. Specifically, the generator was extended to accommodate three parameters:

$UpdatePolicy$. Either the shared prediction based vector policy (0) or the original time-based (1) policy is used.
$Threshold_i, i = 1, ..., 50$. The threshold distance between the predicted and the actual positions, used in the vector policy ($UpdatePolicy = 0$). Up to 50 thresholds may coexist. For each object, its threshold is chosen at random.
$UpdateInterval_i, i = 1, ..., 50$. The average duration between two successive updates of an object (as in the original generator). Up to 50 update intervals are possible. For each object, its average update interval is chosen at random. This parameter is used only when $UpdatePolicy = 1$.

With the vector-based update policy, updates are generated when the distance between the actual position of an object and the predicted position reaches $Threshold_i$. An additional update is generated when an object reaches a hub.

4.3 Evaluation Metrics

The COST benchmark uses two types of performance metrics: the average number of I/O operations per index operation, and the average CPU time per index operation (update, query). One I/O operation is one read of a page from disk to main memory or one write of a page to disk. Reads and writes from and to the available main memory buffer are not counted. The CPU time for one operation is the time of CPU usage from

the moment when the operation is issued to the moment when the result of the operation is computed. I/O is typically considered to be the main cost factor in determining an index's performance, while the CPU time is a minor factor.

5 Definitions of Experiments

A benchmark experiment is defined by a set of workload parameters and disk page and main memory buffer size settings. In each experiment, one parameter, or a set of related parameters, as defined in Sect. 4.1, is varied. The set of experiments was chosen with the objective of varying the important workload parameters from Sect. 4.1. Parameter values are chosen so that the workloads cover a wide variety of situations. To ensure that the benchmark stress-tests the indexes under study, some experiments use extreme parameter values. The page and buffer size settings are kept constant for all experiments.

The default values for all workload parameters and settings are listed in Table 1. The chosen values are commonly used in existing evaluations of spatio-temporal indexes (e.g., [4, 8]). The default speeds are typical speeds of vehicles, and the number of hubs simulates a real-world road network with a substantial number of destinations. The page and buffer sizes are relatively small, the objective being to obtain the effects of large indexes with relatively small volumes of data. For each experiment, described shortly, only parameters with values that differ from the defaults are listed. Note that it is possible to use only a subset of parameters $Speed_i$, $Threshold_i$, and $UpdateInterval_i$, $i = 1, ..., 50$, e.g., it is possible to assign the same speed to all objects by setting $Speed_1$ and omitting parameters $Speed_i, i = 2, ..., 50$.

All experiments measure the average CPU time and number of I/O's per operation.

Table 1. Default workload parameters and settings used in experiments

Parameter	Value	Parameter	Value
$Page, Buffer$	1 KB, 50 KB (50 pages)	$QueryInterval$	400 updates
$Objects$	100 K	$QueryQuantity$	2 (in total 1000)
$Space$	$100,000 \times 100,000$ m^2	$QueryTime$	10 s
$Speed_i, i = 1, ..., 4$	12.5, 25, 37.5, 50 m/s	$QuerySize$	0.25% of $Space$
$TotalUpdates$	200 K	$QueryWindow$	50 s
$Hubs$	500	$QueryTypes$	0.6:0.2:0.2
$UpdatePolicy$	0	$Threshold_1$	100 m

Experiment 1. Number of Objects *Objective:* Examine index scalability.
Parameter values: $Points = 100, 200, ..., 1000$ K.
Number of workloads: 10.

Experiment 2. Position and Velocity Skew *Objective:* Examine the effects of position and velocity skew.
Parameter values: Part 1 (very high skew): $Hubs = 2, 4, ..., 20$. Part 2 (average skew): $Hubs = 20, 40, ..., 200$. Part 3 (low skew): $Hubs = 500, 1000, ..., 5000$, and 0 hubs (uniform distribution).
Number of workloads: 10 for parts 1 and 2, 11 for part 3.

Experiment 3. Maximum Speeds of Objects *Objective:* Examine the effects of varying maximum speeds as well as varying distributions of speeds among the objects. As fast objects are more likely to be updated than slow ones, the update frequency increases with increasing speeds.
Parameter values: Part 1 (distribution of speeds): All objects are assigned either speed 25 m/s or 200 m/s, and workloads are generated so that the fractions of objects with speed 200 m/s are: 0.02; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 0.98. Thus, all $Speed_i$ are assigned either 25 m/s or 200 m/s, and for each workload, the smallest i is chosen that allows us to obtain the needed fraction of fast objects. Part 2 (low maximum speeds): $Speed_1 = 0.05; 2; 4; 6; 8; 10; 12; 14; 16; 18$. Part 3 (high maximum speeds): $Speed_1 = 30, 60, ..., 300$ m/s.
Number of workloads: 11 for part 1, 10 for the parts 2 and 3.

Experiment 4. Position Accuracy Threshold *Objective:* Examine the influence of varying thresholds as well as the distribution of varying thresholds among the objects. Note that the update rate depends on the threshold and that the simulation time increases as updates become infrequent.
Parameter values: Part 1 (distribution of thresholds): All objects are assigned either a threshold of 100 m or a 1000 m, and workloads are generated so that the fractions of objects with speed 1000 m are : 0.02; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 0.98. Thus all $Threshold_i$ are assigned either 100 m or 1000 m, and for each workload the minimum i is chosen that allows us to obtain the needed fraction of objects with large (and small) threshold. Part 2 (equal thresholds for all objects): $Threshold_1 = 100, 200, ..., 1000$ m.
Number of workloads: 11 for part 1, 10 for part 2.

Experiment 5. Update Arrival Interval *Objective:* Examine the influence of varying update intervals as well as distribution of update intervals. The update frequency affects the time duration of a workload.
Parameter values: $UpdatePolicy = 1$. Part 1 (distribution of update intervals): Similarly to the two previous experiments, two values of a parameter, here $UpdateInterval_i$, are used—60 s (frequent) and 600 s (rare). The value of i is chosen so that workloads are obtained where the fractions of objects with an interval of 600 s are: 0.02; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 0.98. Part 2 (frequent updates): $UpdateInterval_1 = 20, 40, ..., 200$ s. Part 3 (rare updates): $UpdateInterval_1 = 120, 240, ..., 1200$ s.
Number of workloads: 11 for part 1, 10 for parts 2 and 3.

Experiment 6. Index Lifetime *Objective:* Examine the effect of varying index lifetime (in numbers of updates).
Parameter values: $TotalUpdates = 100, 200, ..., 1000$ K.
Number of workloads: 10.

Experiment 7. Query Types *Objective:* Examine the differences in performance for different types of queries: timeslice, window, and moving window queries.
Parameter values: $QueryTypes = 1:0:0, 0:1:0, 0:0:1$.
Number of workloads: 3.

Experiment 8. Query Parameters *Objective:* Examine the effects of varying spatial extents, temporal extents, and time windows of queries.

Parameter values: Part 1 (spatial extents): $QueryTypes = 0 : 1 : 0$, $QuerySize = 0.05, 0.15, ..., 0.95\%$. Part 2 (temporal extents): $QueryTypes = 0 : 1 : 0$, $QueryTime = 0, 20, ..., 120$ s. Part 3 (time windows): $QueryTypes = 1 : 0 : 0$, $QueryWindow = 0, 20, ..., 120$ s.
Number of workloads: 10 for part 1 and 7 for parts 2 and 3.

6 Application of the COST Benchmark

In order to ensure that the benchmark was well specified and yields useful results, it was applied for evaluating and comparing three existing indexes, namely the TPR-, TPR*-, and B^x-trees [3, 4, 8]. The TPR*- and B^x-trees were chosen because they are recent and represent the state of the art, and the TPR-tree is the predecessor of a dozen proposals for spatio-temporal indexes.

6.1 Introduction to the TPR-, TPR*-, and B^x-Trees

The TPR-tree (Time Parametrized R-tree) [3] and its descendant, the TPR*-tree [4], are based on the R*-tree [27]. These indexes are adapted for time-parametrized data and queries. Data objects are assigned to minimum bounding rectangles (MBRs) as in the R*-tree. Additionally, the TPR- and TPR*-trees use linear functions of time to represent the movements of the objects and MBRs.

Operations in the TPR-tree are handled similarly to the operations in the R*-tree, except that the penalty metrics of the R*-tree (e.g., MBR enlargement) are generalized to being integrals over a time period ranging from the current time and H time units into the future (calculated based on the update rate).

The authors of the TPR*-tree have modified the TPR-tree by introducing new insertion and deletion algorithms. An additional heap structure is used during insertions with the objective of achieving better insertions. Instead of the integral used in the TPR-tree, the TPR*-tree calculates penalty metrics based on sweeping regions (the area covered by a moving MBR from the current time and H time units into the future).

The B^x-tree uses the B$^+$-tree structure and algorithms to store and retrieve data. Spatial data are transformed into 1-dimensional data using space-filling curves.

The B^x-tree partitions the time axis into intervals with a duration equal to the maximum duration in-between two updates of any object. Each such interval is further partitioned into n phases. For each phase, an index partition is created, at most $n + 1$ partitions existing at a time. The partition in which to insert an object is chosen according to the object's insertion time. As time passes, partitions expire, and new partitions are created. Objects in an expiring partition are reinserted into the newest partition.

The index key of an object is calculated using the update time and position of the object that is stored at the reference time of object's partition. To achieve perfect recall (with the assumption that the accuracy threshold is 0), queries are expanded according to the maximum velocity of all objects and the query time. The objects that qualify for the query according to their velocities are selected; all other objects are filtered out.

For the experimental evaluation, the TPR- and TPR*-trees were extended to support data enlargement, and the B^x-tree was extended to support query enlargement. Enlarged data and query objects are approximated to squares and rectangles, respectively.

6.2 Experimental Evaluation Using the COST Benchmark

Implementations of the three indexes were obtained from their authors and modified where needed in order to perform the benchmark experiments. The indexes require a number of parameters to be set. For the B^x-tree, the maximum update interval is

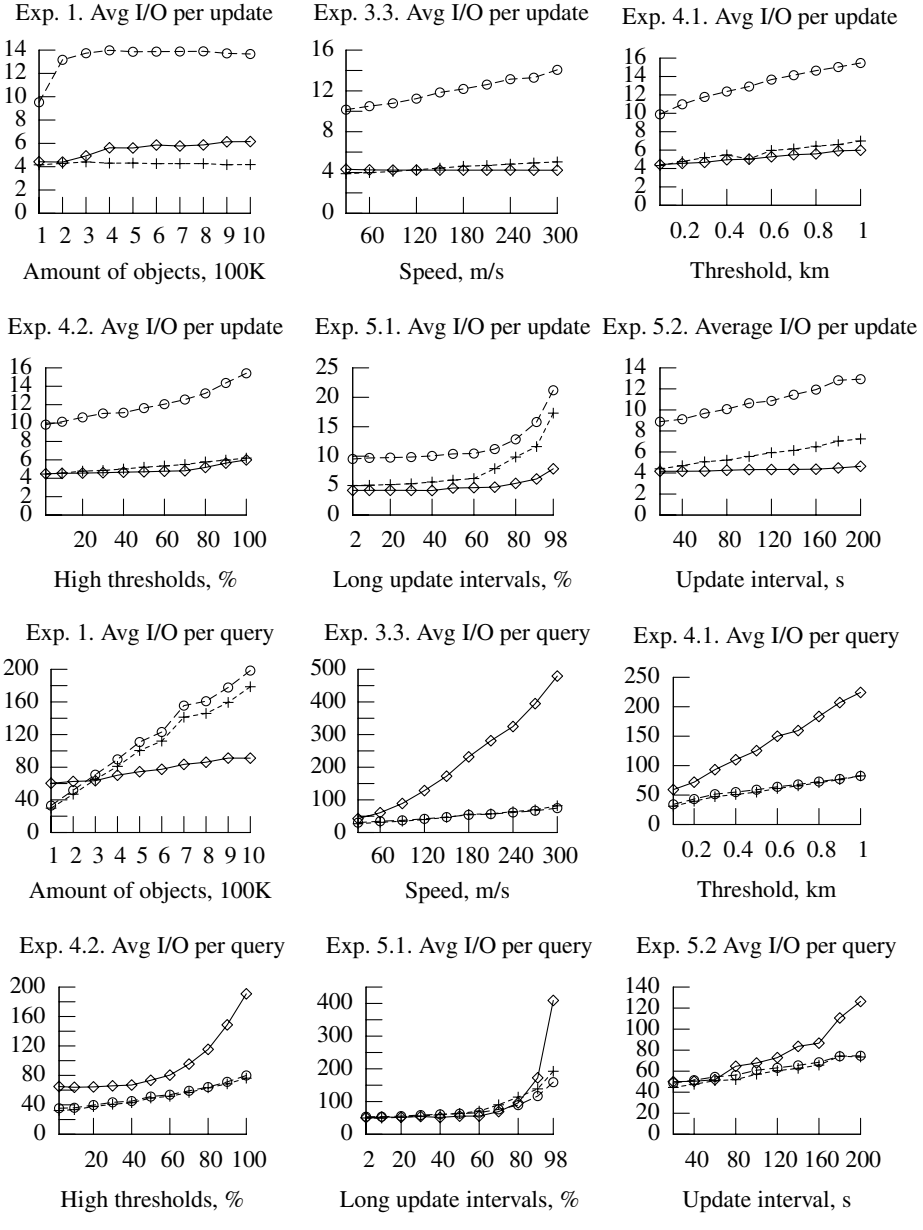

Fig. 4. Example experimental results obtained using the COST benchmark

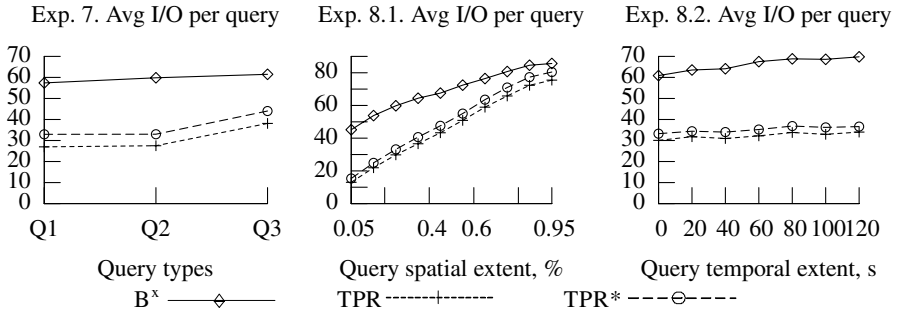

Fig. 5. An example of experimental results using the COST benchmark

120 s, there are 2 phases, the cell size is 100×100 m^2. For the TPR and TPR*-trees, $H = 120$ s.

Representative experimental results are provided in Fig. 4 and Fig. 5. The remaining results are omitted due to space limitations. The experimental results in Fig. 4 show that the indexes are sensitive to changing workloads. For example, high object speeds (more than 75 m/s, Exp. 3.3) or rare updates (less than once in 160 s on average, Exp. 5.1 and 5.2) significantly degrade the query I/O performance of the B^x-tree. However, the update performance of the B^x-tree tends to be more stable than for the TPR- and TPR*-trees (Exp. 3.3, 5.1, and 5.2). When the threshold increases, the query and update performances of the indexes degrade gradually (Exp. 4.1). However, when there is a high percentage of objects with large thresholds, the query performance of B^x-tree degrades significantly (Exp. 4.2). This is due to the resulting long update intervals and large query expansions (Exp. 5.1 and 5.2).

The TPR- and TPR*-trees exhibit inadequate query performance when the index size is large (in the benchmark experiments, above 600 K objects, Exp. 1). The B^x-tree scales well for both query and update performance (Exp. 1).

Experiments that concern query types are shown in Fig. 5. The indexes are largely insensitive to changing query types (Exp. 7). The B^x-tree has a higher overhead compared to the other indexes when query spatial extent is small, but query performance becomes similar for all indexes with larger queries (Exp. 8.1). Varying the temporal extent from 0 to 120 s has only a small effect on query performance (Exp. 8.2).

The experiments demonstrate that the benchmark fulfills its purpose: it has uncovered strengths and weaknesses of the indexes (only some of which were covered by the papers that introduced the indexes). For example, the experimental results identify situations in which the B^x-tree has lower query performance than the TPR-tree and that were not covered by the paper presenting the B^x-tree [8]. As another example, the benchmark shows that situations (not covered by the paper presenting the TPR*-tree [3]) exist where the TPR-tree outperforms the TPR*-tree for updates.

Summarizing the experimental results, the TPR-, TPR*-, and B^x-trees appear each to be the best choice in different situations, characterized by different workloads. The B^x-tree seems to be a good choice in situations with large numbers of objects, which degrade the performance of the TPR- and TPR*-trees. The B^x-tree also performs well when the maximum interval in-between the updates is known; the maximum position

accuracy threshold is low; and the speeds of objects do not exceed the usual speeds of vehicles. In other cases, the TPR- or TPR*-trees, which most often exhibit very similar query performance, should be chosen. With extremely long update intervals, the TPR*-tree might be preferable over the TPR-tree.

The TPR- and TPR*-trees appear to be the most versatile indexes; however, the B^x-tree is based on the B^+-tree; which is already available in many DBMSs. Therefore, a creation of a more robust version of the B^x-tree may be promising direction for research.

7 Conclusions and Future Work

A number of indexes for the current and near-future positions of moving objects exist, and more are underway. This state of affairs creates an increasing need for a neutral and well-articulated experimental setting for evaluating and comparing these indexes.

This paper proposes a benchmark, termed COST, that is targeted specifically toward the evaluation of such indexes. The benchmark aims to make realistic assumptions about the experimental settings—data is inherently inaccurate, predictive queries that reach into the future are covered, the indexes are assumed to be stored persistently on disk. More specifically, an update technique is assumed where positions are guaranteed to be accurate within agreed-upon thresholds and where updates occur only when necessary in order to satisfy the guarantees. The indexes may use either query or data enlargement to account for the inaccurate data. The benchmark includes a workload generator, definitions of experiments, and evaluation metrics. It considers a wide range of workload parameters that cover many real-world situations.

As proof of concept and to evaluate the benchmark, it was applied to the TPR-, TPR*-, and B^x-trees. The experiment demonstrates that the benchmark is well-specified and is capable of covering a wide range of situations. Weaknesses and strengths of the indexes were detected by examining the sensitivity of the indexes to workloads with varying parameter values, including workloads with extreme settings. The experimental results cover situations that were not covered in the papers that introduced the indexes, due to more extensive experiments. The obtained results provide guidance as to when each of the indexes should and should not be used.

The benchmark may be extended by inclusion of such aspects as index size in disk pages, CPU time and numbers of I/O for bulkloading and bulk operations, and evaluation of concurrent accesses. Further studies of existing spatio-temporal indexes are also warranted, possibly including detailed studies of special cases and aspects specific to individual indexes. Examples include detailed studies of overlaps among MBRs, growth rates of MBRs, and the grouping of objects into MBRs in R-tree-based indexes. For the B^x-tree, such studies may cover query enlargement aspects and migration loads. For all indexes, it is of interest to investigate aspects such as tree depths and node fanouts. Studies such as these have the potential to offer insights that may guide the development of improved indexes.

Acknowledgments. This research was conducted within the project Telematics Applications Based on Ubiquitous Sensor Networks, funded by the Electronics and Telecommunications Research Institute, South Korea. C. S. Jensen is also an adjunct professor in Department of Technology, Agder University College, Norway.

References

1. Blewitt, G.: Basics of the GPS technique: observation equations. Geodetic Applications of GPS (1997) 10–54
2. Wikipedia: GPRS (2001–2005) http://en.wikipedia.org/wiki/GPRS.
3. Šaltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: Proc. ACM SIGMOD. (2000) 331–342
4. Tao, Y., Papadias, D., Sun, J.: The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In: Proc. VLDB. (2003) 790–801
5. Procopiuc, C.M., Agarwal, P.K., Har-Peled, S.: STAR-tree: an efficient self-adjusting index for moving objects. In: Revised Papers from the 4th International Workshop on Algorithm Engineering and Experiments. (2002) 178–193
6. Šaltenis, S., Jensen, C.S.: Indexing of Moving Objects for Location-Based Services. In: Proc. ICDE. (2002) 463–472
7. Patel, J.M., Arbor, A., Chen, Y., Chakka, V.P.: STRIPES: an efficient index for predicted trajectories. In: Proc. ACM SIGMOD. (2004) 635–646
8. Jensen, C.S., Lin, D., Ooi, B.C.: Query and update efficient B+-tree based indexing of moving objects. In: Proc. VLDB. (2004) 768–779
9. Zobel, J., Moffat, A., Ramamohanarao, K.: Guidelines for presentation and comparison of indexing techniques. SIGMOD Rec. **25** (1996) 10–15
10. Gray, J., ed.: The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann Publishers, Inc. (1993)
11. Theodoridis, Y.: Ten benchmark database queries for location-based services. The Computer Journal **46** (2003) 713–725
12. Myllymaki, J., Kaufman, J.: DynaMark: A Benchmark for Dynamic Spatial Indexing. In: Proc. MDM. (2003) 92–105
13. Werstein, P.F.: A performance benchmark for spatiotemporal databases. In: Proc. of the 10th Annual Colloquium of the Spatial Information Research Centre. (1998) 365–373
14. Tzouramanis, T., Vassilakopoulos, M., Manolopoulos, Y.: Benchmarking access methods for time-evolving regional data. Data Knowl. Eng. **49** (2004) 243–286
15. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Querying imprecise data in moving object environments. IEEE Trans. on Knowl. and Data Eng. **16** (2004) 1112–1127
16. Tao, Y., Cheng, R., Xiao, X., Ngai, W.K., Kao, B., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: Proc. VLDB. (2005) 922–933
17. Čivilis, A., Jensen, C.S., J. Nenortaitė, J., Pakalnis, S.: Efficient tracking of moving objects with precision guarantees. In: Proc. MobiQuitous. (2004) 164–173
18. Wolfson, O., Sistla, A.P., Chamberlain, S., Yesha, Y.: Updating and querying databases that track mobile units. Distrib. Parallel Databases **7** (1999) 257–387
19. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. In: Proc. SSD. (1999) 111–132
20. Lazaridis, I., Mehrotra, S.: Approximate selection queries over imprecise data. In: Proc. ICDE. (2004) 140–152
21. Weisstein, E.W.: Minkowski sum. From MathWorld—A Wolfram web resource (1999–2005) http://mathworld.wolfram.com/MinkowskiSum.html.
22. Šaltenis, S., Jensen, C.S., Leutenegger, S., Lopez, M.: Indexing the positions of continuously moving objects. Technical report, Aalborg University (November 1999)
23. Kaufman, J., Myllymaki, J., Jackson, J.: CitySimulator (2001) https://secure.alphaworks.ibm.com/aw.nsf/techs/citysimulator.
24. Myllymaki, J., Kaufman, J.: LOCUS: A testbed for dynamic spatial indexing. IEEE Data Eng. Bull. (Special Issue on Indexing of Moving Objects). **25** (2002) 48–55

25. Theodoridis, Y., Nascimento, M.A.: Generating spatiotemporal datasets on the WWW. SIGMOD Rec. **29** (2000) 39–43
26. Theodoridis, Y., Silva, J.R.O., Nascimento, M.A.: On the generation of spatiotemporal datasets. In: Proc. of the 6th International Symposium on Advances in Spatial Databases. (1999) 147–164
27. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. In: Proc. SIGMOD. (1990) 322–331