An Efficient Spatiotemporal Indexing Method for Moving Objects in Mobile Communication Environments

Hyun Kyoo Park[1], Jin Hyun Son[2], and Myoung Ho Kim[1]

[1] Div. of Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Guseong-dong, Yuseong-gu, Daejeon, South Korea, 305-701
{hkpark, mhkim}@dbserver.kaist.ac.kr

[2] Dept. of Computer Science and Engineering, Hanyang University, 1271 Sa-1 dong Ansan, Kyunggi-Do, South Korea, 425-791
jhson@cse.hanyang.ac.kr

Abstract. The spatiotemporal databases concern about the time-varying spatial attributes. And one of the important research areas is tracking and managing moving objects for the location-based services. Many location-aware applications have arisen in various areas including mobile communications, traffic control and military command and control (C2) systems. However, managing exact geometric location information is difficult to be achieved due to continual change of moving objects' locations.

In this paper we propose the B^{st}-tree that utilizes the concept of multiversion B-trees. It provides an indexing method for future location queries based on the dual transformation. This approach can be applied for the range query on moving object's trajectories specifically in the mobile communication systems. Also we present a dynamic management algorithm that determines the appropriate update interval probabilistically induced by various mobility patterns to guarantee the query performance.

1 Introduction

The recent advances in mobile computing and sensing technology such as *GPS* have made it possible to perform location-based services via wireless links. One of the important problems for location-based services is to provide fast answers about range queries that retrieve the moving objects in a certain area. The spatiotemporal indexing method for the location management is one of the important issues for this purpose.

However, in the conventional database systems, the data remain unchanged unless it is explicitly modified. It means that the accuracy of the location information is difficult to be maintained in the database systems, since a large number of continual updates may not be properly processed. To reduce the number of updates required while keeping the reasonable data accuracy, functions of time that express the object's positions may be used. Then updates are necessary only when the parameters of the functions change "significantly" [6, 10].

In this paper we propose an indexing approach that can answer the queries about current or anticipated future positions of moving objects. We use one linear function per object's trajectory that is composed of the initial position and a function of velocity for a certain period of time. Then we index the location information by the ideas of *Space Filling Curve* and duality. Also we provide a dynamic management algorithm of the index structure to maintain the performance of range queries about future position.

2 Problem Statement and Motivation of Research

The database management system (DBMS) technology can provide a foundation for spatiotemporal applications in spite of some problems such as frequent updates. So, the spatiotemporal capabilities need to be integrated, adapted, and built on top of existing database management systems.

Previously most related work focused on historical queries for the spatiotemporal databases. However, the location-based services require future queries as well as the conventional historical queries. In this paper we concentrate on the range queries of moving objects in the future. An example future query may be "retrieve the patrol cars that are or will be in a given region within next 30 minutes". More generally, the location-based range queries considered in this paper have the following forms:

Type-1(Time-sliced) Query: "Find all objects that are(or will be) in a certain area at time *t*" □

Type-2(Time-window) Query: "Find all objects that are(or will be) in a certain area during time [*ts, te*]" □

In the location-based services, the positional information of moving objects is obtained via wireless communication. Then the real trajectories can be approximated as linear functions through the *Dead-Reckoning* policy that can predict the future position of the object. Then the trajectories can be reduced to lines starting from an initial location during a certain period of time. Since several mobile communication protocols have been developed as a solution of location management for next-generation communication system (e.g., *IS-41, GSM*), this assumption is practical [10].

One of the indexing methods for the lines is a *Spatial Access Method* (SAM) such as the R-tree [9]. However, there are several problems when handling the trajectories for the future position. Some of them are: (i) The *Minimum Bounding Rectangle* (MBR) assigned to a trajectory often has too large area, and (ii) implicitly all lines may extend to '*infinity*' until being updated. It may incur many overlaps of MBRs.

Hence we need an appropriate methodology to support indexing of trajectories rather than previous spatial access methods. In addition, the index structure should afford the frequent updates after storing the trajectory information.

3 Preliminaries

3.1 Related Work

Wolfson et al. [6] proposed a data model to represent moving objects. They proposed a framework called *Moving Object Spatio-Temporal* (MOST) for tracking objects and a temporal query language called *Future Temporal Logic* (FTL). Several spatiotemporal indexing methods were proposed for the concept of the paper [6].

In the past, research in spatial and temporal database has mostly been made separately. And for the theoretical basis, the community of computational geometry developed early work on moving points focused on bounding the number of changes in various geometric structures as the points move [6]. One of the result is the *Kinetic Data Structure* (KDS) by Basch et al. proposed in [2].

A survey of previous work on spatiotemporal databases can be found in the papers [4, 7, 9]. An important result of these is the TPR-tree by Saltenis et al. [9]. They proposed the R-tree based approach but we think that the R-tree based approach has some restrictions as a remedy of our problems. Another result by Kollios et al. [4] proposed a sophisticated approach for the future queries that can outperform the kd-B-tree and the R-tree. In their work, they used the *Hough-X* transformation for indexing moving points, however they assumed that the space is linear or quadratic and the velocities are fixed.

To support the future queries those we are concerning, we use multi-version structure concept [3, 11] and extend the study of [4]. Also an appropriate prediction of mobility patterns can reduce the difficulties in studying the continuous updates in spatiotemporal indexing and the performance degradation in the multi-version structure. Yet much of the literature to date has not considered it. So we refer the information theory and the stochastic process [1] to predict update intervals for our dynamic update algorithm in the mobile communication environment [10].

In the practical environment, the location information is acquired from mobile communication systems. And various location management scheme were suggested [5, 10] and the dynamic location management scheme in the next generation communication network [10] is used as a framework of our method.

3.2 Duality and Space Filling Curves

In the mobile communication environment, moving objects send their location information and we can translate them as a set of tuples $\{\ t, x, y, Fx(t), Fy(t)\}$ where t is the time when the update message is sent. x, y denotes the coordinates of the location. $Fx(t)$ and $Fy(t)$ that are functions of time denote velocity vectors

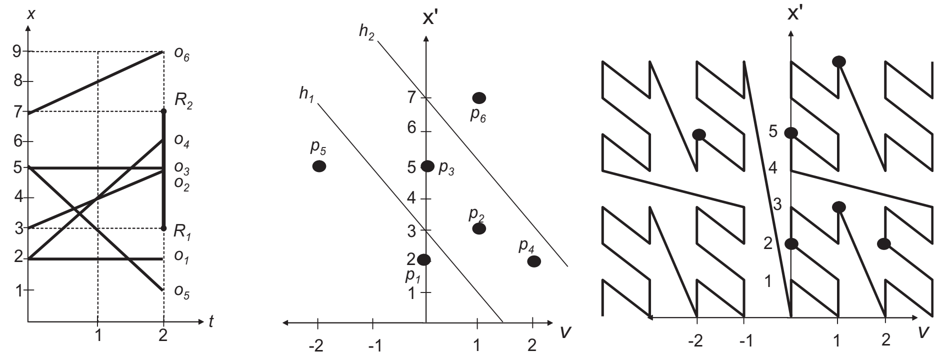

Fig. 1. Trajectories of moving points in the primal and the dual space

at x and y, respectively. This location information is transformed to a point in the dual space as follows:

In d-dimensional space R^d, the hyperplane represented as a function of $x_d = \nu_1 x_1 + \ldots + \nu_{d-1} x_{d-1} + \nu_d$ in the primal space is transformed to a point $(\nu_1, \ldots, \nu_d)$ in the dual space and the point $(\nu_1, \ldots, \nu_d)$ in the primal space can be transformed to a line of $x_d = -\nu_1 x_1 - \ldots - \nu_{d-1} x_{d-1} - \nu_d$ in the dual space [7].

Figure 1 shows the geometric representation of moving objects in the primal and the dual spaces. Generally, a trajectory in the primal xt plane induces a static point p in the dual $x'\nu$ plane. Similarly a query range R induces a query strip σ that is bounded by two hyperplanes. For example, the trajectories of objects $(o_1, \ldots, o_6)$ in the primal space can be transformed to points $(p_1, \ldots, p_6)$ in the dual space. Also extreme points R_1 and R_2 of range R in the primal space can be represented as a strip between the hyperplanes h_1 and h_2 in the dual space. Hence the query of retrieving trajectories intersecting a range R parallel to the x axis can be rewritten to a query of retrieving set of points in the query strip σ.

In this paper we consider *Space Filling Curve* for an indexing in the dual space. In Figure 1, the *Peano* curve provides a linear ordering in multidimensional spaces starting from a lower left corner. Let $SFC(x, v)$ be the function that assigns a linear address for two-dimensional point (x, v). Then the dots in the dual space have unique addresses that can be mapped as key values in one-dimensional index structure.

4 The Proposed Index Structure for Moving Objects

In this section, we describe the B^{st}-tree; namely the Spatiotemporal B-tree. The B^{st}-tree incorporates features of B-tree and the multi-version tree [3, 11]. Since the B^{st}-tree stores the whole trajectory of a moving object as pieces, we need a multi-version access method that stores the records with their own life-time to handle the updated records.

Although moving object traces out a certain trajectory in three-dimensional spatiotemporal space, we describe the two-dimensional case for simplicity. Then the three dimensional queries can be achieved by conjunctions of each trees.

In Figure 2(a), object O_2 changes its trajectory at time t_2 and t_4, and O_3 changes at every timestamp while O_1 keeps its movement function all the time. The moving object checks its location periodically. We define unit time Δt and normalize to 1 for convenience. Since it is impractical to monitor the location at every unit time, the object monitors its location every time interval $\tau = n\Delta t$. For example, Figure 2(b) shows the state of object O_2. Then there are two updates at t_2 and t_4. Here we define two different updates.

Definition 1. ***Adaptive Update***
The Adaptive Update(AU) is an update operation to make an index of current location information for all moving objects.

Definition 2. ***Instantaneous Update***
The Instantaneous Update(IU) is a demanded update by individual moving object when a movement function of an object is changed.

The time interval between two consecutive *AU* is represented as T_u in Figure 2(b) and determined by a dynamic method described in Section 5. *IU* is occurred when the deviation from the expected trajectory exceeds the distance threshold that is predefined by the *Distance-based update* protocol [10].

In our scheme, we discard the notion of *Location Area* borders and communication cells in most *PCS* or *Cellular* systems. Dynamic mobility management in wireless communication services is known to provide better cost-effectiveness and our method can be dynamically adapted to this scheme [5, 10].

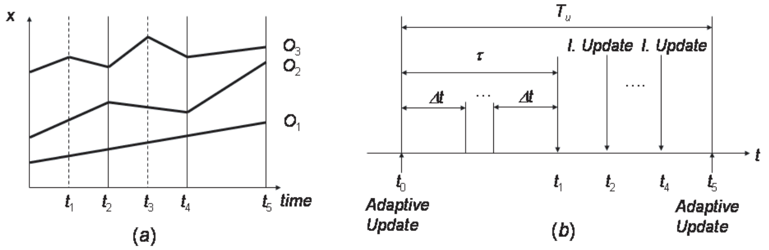

Fig. 2. Representation of trajectories and the update parameters

4.1 Overview of the B^{st}-tree

Although a multi-version structure keeps both the current and old data, it may experience a significant overhead when a large number of updates. Thus we may need an appropriate management strategy to handle this case. The B^{st}-tree has two types of nodes, i.e., *Branch nodes* and *Version nodes*, as shown in Figure 3. When AU occurs, a new *Branch* is made with the update timestamp. Then the *Branch node* has a timestamp as a key value and a link to the *Version node.* For example, an AU occurred at time 0 and 5. Then we have two *Branches* of $b_1(0)$, $b_2(5)$ as shown in Figure 3.

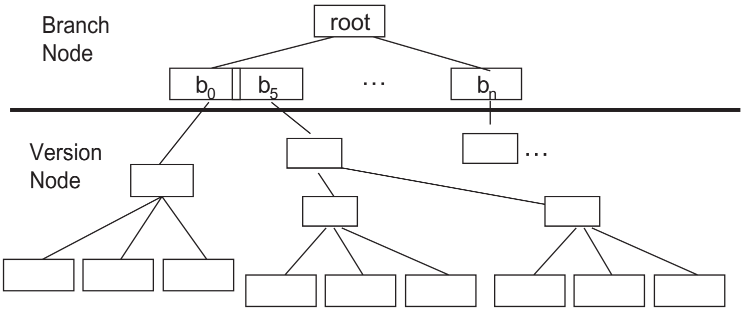

Fig. 3. The illustration of the B^{st}-tree structure

Definition 3. ***B^{st}-tree Record Structure***
The record in a Version node of the B^{st}-tree has the form (Key, TS, Pointer) where Key, TS and Pointer denote a key, life-time of a record and a link to a node at the next level (intermediary) or of the actual record (leaf) respectively.

In the *Version node*, the index key value is acquired from the method described in Section 3.2. The life-time TS is set of (t_s, t_e): the start and end time of the record respectively. The currently valid record has the TS value of $(t_s, *)$ and the updated old record has (t_s, t_e) value.

4.2 Fundamental Operations

In this section we describe the insert and update operations. The B^{st}-tree is partially persistent; therefore insertions and updates only are applied to current data pages. To insert a new record or update an existing record, the B^{st}-tree starts searching the current *Branch* node until finding the appropriate leaf node which the record must be added.

To differentiate from the inserted record, we call the updated record a *variant.* In the update operation, we do not perform the delete operation but invalidate the currently existing record. The invalidation changes the t_e value of the record to current timestamp. Hence the update operation inserts a *variant* into a proper node that is logically the same as the insert operation followed by invalidation. This process is the same as a conventional temporal database. However, if there is modification of the structure by the split or merge operation, the operations are not trivial.

Definition 4. ***Node Splitting***
The P_{max} and P_{min} are node split parameters to control the node occupancy. The values of parameters are $P_{min} \leq P_{max} \leq B$, where B is the node capacity.

When an update operation triggers the modification of structure, the valid records of overflowed node are copied to a new assigned node with the current timestamps. Then we invalidate all the records in the previous node and insert the *variant* into the new node. The node split is divided into two cases where N_v is the number of valid records in a node.

Case 1. $Nv > P_{max}$; Create two nodes. And copy all the valid records in the overflowed node to the new nodes by their key values. □

Case 2. $P_{min} \leq Nv \leq P_{max}$; Create one node. And copy all the valid records in the overflowed node to the new node. □

In Figure 4, the upper illustration describes the Case 1 and the lower illustration describes the Case 2 updates. In the Case 1, the update operation copies the valid records in overflowed node A (e.g., the records have key value 5, 6, 8, 21) into nodes B and C at time 5. Then we change the life-time value '*' of the TS in node A to 5. After all we insert a *variant* of key value 25 in C. In the Case 2, there are two valid records (e.g., the records have key values 5, 7) in overflowed node. Then we assign one node and perform the similar update procedure in the Case 1. In the case of the number of valid records being less than the value of P_{min}, the underflow condition is occurred. When the underflow condition is occurred, we merge valid records in underflowed node with sibling node as the same way of the update procedure. Therefore, when a node is created, it is ensured that it contains less than the value of P_{max} valid records.

4.3 Range Searching in the Dual Space

The range searching in the primal space induces the infinite range problem in the dual space as shown in Figure 1. For example, query range of [R1, R2] in the primal space in Figure 1 derives infinate query strip in the dual space. So, we need a "*Pruning*" method to limit the query strip to avoid the infinite range problem in the dual space.

The pruning phase is processed as follows. Since the possible positive and negative maximum velocity of objects can bound the query strip in the dual space, we can set the query strip to two rectangular regions as shown in Figure 5. The maximum velocity trajectories T_1 and T_3 in the primal space are represented by points T_1^* and T_3^* in the dual space. The staying objects' trajectories that may intersect the query range are depicted points T_2^* and T_4^* in the dual space in the same way.

This restriction of range query is intuitive since the moving objects have maximum velocities in real world. Then those four points make a super set of

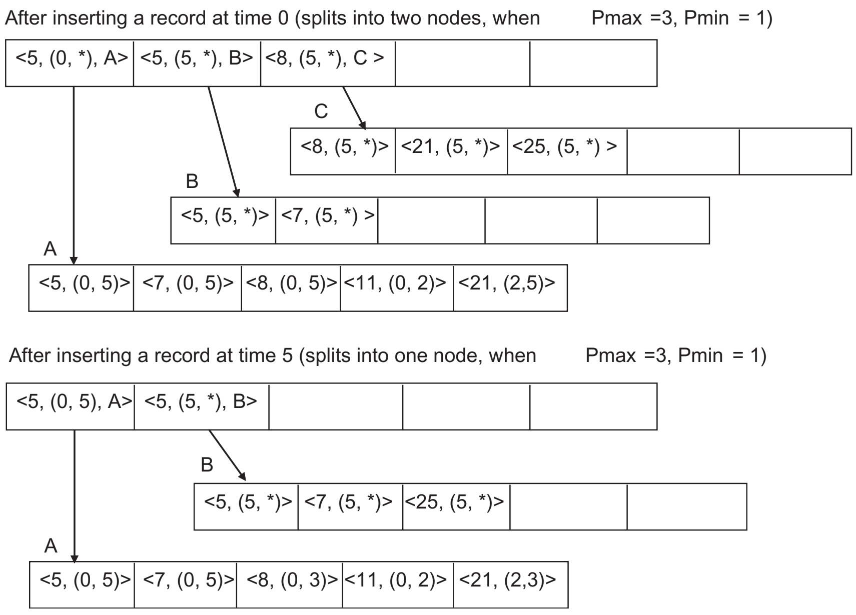

Fig. 4. Illustration of updates with structure modification

range query result. For time-window range queries, we only consider the range at time t_e during the pruning phase since it contains all possible trajectories between t_s and t_e. Then the query strip in the dual space is bounded by rectangles the same as the previous time-slice queries. Based on this observation, we can define corresponding $Lower\_Left(LL)$ for the lowest key value and analogously $Upper\_Right(UR)$ value in a query range region. The result set is composed of several data pages based on the *Peano* value in an ascending order. However all the pages in the key ranges $[LL, UR]$ may not be the result set. The result set corresponds to a set of leaves that are composed of several node region that is represented by I_n of the *Version node* in Figure 5. Each node has its key range $LL \leq l_i \leq u_i \leq UR$ where l_i and u_i are the lower and upper key values of each node i respectively.

To retrieve the exact pages rather than the all pages in range $[LL, UR]$, we first search for the leaf node that contains the lowest value larger than LL value. Then follow pointers to find subsequent leaves until the node that has the larger l_i value than UR value. The algorithm of excluding the unnecessary pages shown in Figure 5 was introduced in several works [8], so that we do not mention the details here.

5 Dynamic Management of the B^{st}-tree

In this paper we propose a dynamic algorithm to select the appropriate update interval (the value of T_u in Figure 2) in the mobile environments. The determi-

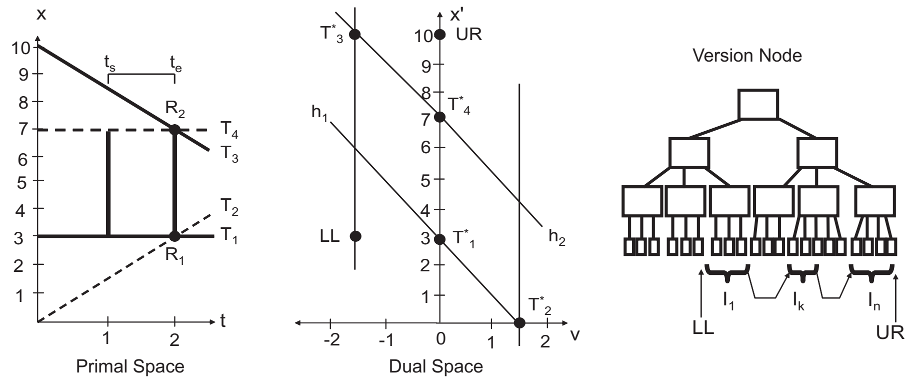

Fig. 5. Primal-Dual representation for range query

nation of the interval between two consecutive AU depends on the duration that objects keep their movement linear and the performance of *Version node*. Various mobility models found in the literature [10], however most index structures assume linear movement with constant speed [2, 12], that does not reflect reality. In our approach, we adopt the multi-version structure to reflect updates. So we model a moving object as a dynamic linear function and use the cost model that is suggested by Tao et al. [11] for the evaluation of the performance and suggest the parameter determination methodology.

5.1 Cost Model for Dynamic Management

The performance of B^{st}-tree depends on the number of node accesses that are affected by the mobility pattern that defines how random the movement is. The movement pattern is described as *Randomness* as;

Definition 5. *Randomness*
The Randomness defines how random the mobility pattern is. If on average, v new variants by IU will be occurred in a Branch, then the Randomness is v/N_v where N_v is the valid number of records at a certain time.

At a timestamp, the expected number of N_v is $ln2(P_{max}/B)$ since the B^{st}-tree follows the average utilization of B-tree. Hence after a certain number of updates, the performance of *Version node* approaches the degradation point [3, 11]. At the degradation point, the *Version node* produces worse performance than a single ephemeral B-tree.

The value of *Affordable Randomness* is determined mostly by the value of P_{max} that is selected as a tuning parameter. Then we need the value of *Affordable Randomness* for query parameters (q_k, q_t). In Eq. (1), the value of *Affordable Randomness* for query parameters is shown.

$$AffordableRandomness(q_k, q_t) \cong \frac{\frac{N_v}{N_n} q_t - 1}{\frac{ln2(P_{max}/B)}{1 - ln2(P_{max}/B)}(q_t - 1)} \tag{1}$$

where q_k and q_t denote query range of key and time space respectively. We can allow *IU* in the interval of *AU* until *Affordable Randomness* by queries and the value of *Affordable Randomness* can determine the optimal *Branch* duration T_u in the B^{st}-tree.

5.2 Probability-Based Dynamic Update Algorithm

In continuous time, the movement of objects can be described as a *Brownian* motion and the location distribution of an object in the future is a stochastic process as time progresses [1]. Furthermore, the velocity of object is a stationary *Gauss-Markov* process since the initial location and the ensuing location estimation process have the same probability law. Hence, the predictive location of moving object in the future is correlated in time and modeled by some finite states with its velocity. In the linear motion assumption, we discretize the continuous movement into intervals to check the location information as shown in Figure 2.

Let define $\alpha = e^{-\lambda_p \tau}$ $(0 \leq \alpha \leq 1$, where $0 \leq \lambda_p)$, the rate of *IU*. Then the velocity of moving object in discrete time is;

$$v_n = \alpha v_{n-1} + (1-\alpha)\mu + \sqrt{1-\alpha^n}x_{n-1} \tag{2}$$

In the sense of *Gauss-Markov* process, the general representation of velocity in terms of the initial velocity v_0 is

$$v_n = \alpha^n v_0 + (1-\alpha^n)\mu + \sqrt{1-\alpha^n}\sum_{i=0}^{n-1}\alpha^{n-i-1}\zeta_i \tag{3}$$

In the equation, ζ_n is an independent, uncorrelated, and stationary *Gaussian* process, with mean $\mu_\zeta = 0$ and $\sigma_\zeta = \sigma$, where μ and σ are the asymptotic mean and standard deviation of v_n when n approaches infinity [1, 5].

To expect the update interval, the moving object checks its location periodically. Let τ be the location information inspection period as shown in Figure 2, and $s_{k\tau-1}$ be the displacement from an initial state. Then each moving object triggers *IU* at the k^{th} location inspection where the condition of $|s_{k\tau-1} - \mu_{s_k}|$ is greater than a distance threshold. Probabilistically the expected location information at k^{th} location inspection is x_k that we can derive from the Eq. (3)

$$x_k = \sum_{i=k\tau}^{i=k\tau+\tau-1} v_i \quad \text{and} \quad s_{k\tau-1} = \sum_{i=0}^{k-1} x_i \tag{4}$$

$s_{k\tau-1}$ is a *Gaussian* random variable with mean $\mu_{s_k} = \sum_{i=0}^{k-1}\mu_{x_i}$. The expected value of location x_k and displacement $s_{k\tau-1}$ is $hatx_k = x_k - \mu_{x_k}$ and $hats_k = s_{k\tau-1} - \mu_{s_k}$. Then the following recursive iteration predicts the update interval probabilistically.

$$\hat{s}_k = \hat{s}_{k-1} + \hat{x}_{k-1} \tag{5}$$

Table 1. Definition of fuctions and parameters for the update interval

Functions	*Contents*
$\overline{p}_k(\hat{s},\hat{x})$	Probability there is no update up to time $k-1$ where $\hat{s}_k=\hat{s}$, $\hat{x}_{k-1}=\hat{x}$
$p_{k-1}(s,\hat{x})$	Probability there is no update up to time k where $s_k=s$, $x_{k-1}=x$
$p_k(s,x)$	Probability there is no update up to time k where $s_k=s$, $x_k=x$
$\overline{F}(k)$	Probability there is no update up to time k
$f(k)$	PDF of time k between two consecutive updates

From the previous analysis, our deterministic algorithm provides the predictive k^{th} interval of each *Branch*. In Table 1, we define some random variables and probability density functions used in our algorithm. Probabilistically the AU interval guarantees the query performance. So, the location distributions at time $k>0$ can be found from $k=1$ until $f(k)=\overline{F}(k-1)-\overline{F}(k)$ is sufficiently small, with initial condition of $\overline{p}_k(\hat{s},\hat{k})=f_{\hat{x}_0}(\hat{x})\delta(\hat{s}-\hat{x})$ where $k=0$, distance threshold δ and $f_{\hat{x}}$ the PDF of $\hat{x}_k$ for the x direction. where . The $\overline{F}(k)$ is acquired from Eq. (6).

$$\overline{F}(k)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}p_{k-1}(\hat{s},\hat{x})d\hat{x}d\hat{s} \tag{6}$$

In the algorithm, the iteration will stop when the value of k exceeds the *Affordable Randomness* iteratively for each object. Finally, we set the *AU* interval for the new *Branch* from the current time to the value of k.

```
Algorithm determine_interval_of_Adaptive_Update
   step 1: calculate k of all objects and enqueue in queue_k
   step 2: dequeue minimum k in queue_k
   step 3: while randomness > affordable_randomness do loop
     increase the number of variants and calculate randomness;
     dequeue the current k in queue_k;
     calculate new k of the current object;
     enqueue the new k in queue_k;
```

6 Experimental Results

We evaluated the B^{st}-tree in the *Sun Ultra-60* with 256MB main memory system. The experimental data set was generated with $GSTD$[1] software that simulates the trajectories of moving objects. The B^{st}-tree allows *variants* to be inserted until it reaches until the *Affordable Randomness.* In Figure 6, we investigated

[1] Synthetic moving object dataset generation tool, SIGMOD Record, 29(3).

the *Affordable Randomness* for the time-sliced and time-window queries where the moving objects are uniformly distributed. This is the expected result that is suggested in [11]. We investigated the performance of the B^{st}-tree with the uniformly distributed datasets here. As a result, for the time-slice query, it is preferable not to allow *IU*. Also for the time-window query, the practically *Affordable Randomness* converges almost 30%. In the *Affordable Randomness*, our method is efficient for the construction and reflection of updates.

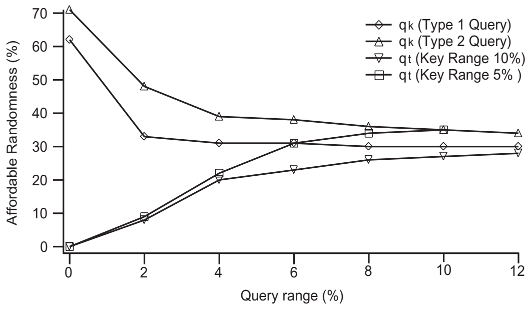

Fig. 6. *Affordable Randomness* for the various queries

And the R*-trees consume more construction time than the B^{st}-tree. We do not reflect the TPR-tree algorithm since it is not available to get the end time of the movement functions when we construct the trees. Hence we construct the R*-tree after one iteration of *GSTD* and use the trajectories. To evaluate *IU* performance, we use 500,000 objects to construct trees and update 1,000 objects up to 20,000 objects. The results of comparisons are shown in Figure 7.

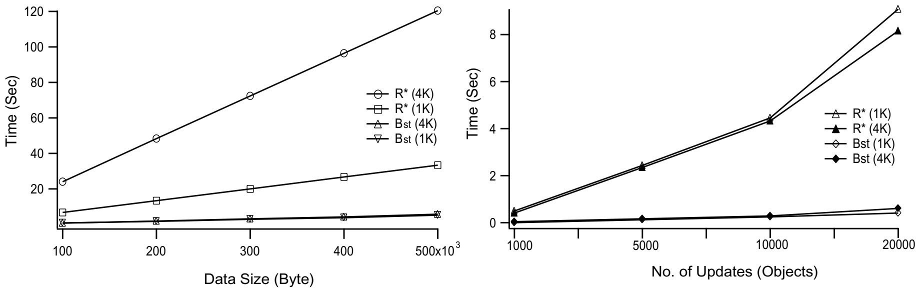

Fig. 7. Performance comparison of construction and update time

The size comparison of the R*-tree and the B^{st}-tree is shown in Figure 8. For future queries in our environment, the size of the R*-tree is not affected by the variance of the *Randomness* significantly but the B^{st}-tree is increased exponentially. Although in the 30% *Randomness*, the size of the B^{st}-tree is less than half of the R*-tree. Furthermore, the node accesses in answering the time-slice query of 5% key range in the 20,000 objects are shown in Figure 8. Since the future queries are affected by the *Randomness* directly, the 0% *Randomness* matches the optimal performance and the B^{st}-tree increases the number of node accesses almost linearly less than 25% *Randomness* range.

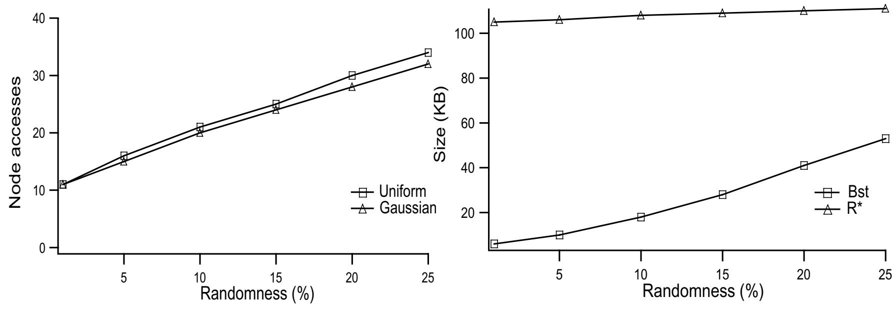

Fig. 8. Node access and size comparison by the *Randomness*

7 Conclusions and Future Work

The location-based service is expected being used widely in the near future, and it is an important application area of the spatiotemporal databases. In this paper we have proposed an index structure, named the B^{st}-tree as a spatiotemporal indexing method for the future location queries in mobile communication environments. The contribution of our work can be summarized as follows.

(i) An efficient indexing approach for moving objects to support the historical and future queries: While the R-tree based indexing is very time consuming for construction or updating, our index structure requires less time consuming and supports $O(log_B N + t)$range query performance.

(ii) A comprehensible probability-based determination of update interval: The proposed algorithm can predict the behavior of an index structure for guaranteeing the query performance. Our update policy uses the mobile communication framework that can be very useful for the next generation communication environments, specifically the *Distance-based* management protocol.

For further work, we are working on the effects of the mobility patterns that can be generated in various environments. We use the discrete time interval for the prediction, and we will analyze our method in continuous time.

Acknowledgement: This work was supported in part by grant No. R01-1999-00244 from the interdisciplinary Research program of the KOSEF.

References

1. Ross, S.: Stochastic Processes 2nd Ed., John Wiley & Sons (1996)
2. Basch, J., Guibas, L. Hershberger, J.: Data Structures for Mobile Data. Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms.(1997) 747–756
3. Bertimas, D., Tsitsiklis, J.: Introduction to Linear Optimization. Athena Scientific.(1997)
4. Varman, P., Verma, R.: An Efficient Multiversion Access Structure. IEEE TKDE Vol.9(3) (1997) 391–409
5. Liang, B., Haas, Z.: Predictive Distance-Based Mobility Management for PCS Networks. Proc. of IEEE INFOCOM.(1999) 1377–1384
6. Wolfson, O., Sistla, P., Chamberlain, S., Yesha, Y.: Updating and Querying Databases that Track Mobile Units. J. of Distributed and Parallel Databases. (7). (1999) 257–287
7. Agarwal, P., et al.: Efficient Searching with Linear Constraints. J. of Computer and System Sciences. 61. (1999) 194–216
8. Ramsak, F., et al.: Integrating the UB-Tree into a Database System Kernel. Proc. of VLDB. (2000) 263–272
9. Saltenis, S., et al.: Indexing the Positions of Continuously Moving Objects. Proc. of SIGMOD. (2000) 331–342
10. Wong, V., Leung. V.: Location Management for Next-Generation Personal Communication Networks. IEEE Network 14(5). (2000) 18–24
11. Tao, Y., Papadias, D., Zhang, J.: Cost Models for Overlapping and Multi-Version Structures. Proc. of ICDE. (2002) 191–200
12. Kollios, G., Gunopulos, D., Tsotras, V., "On Indexing Mobile Objects", Proc. of PODS, (1999) 262–272