An Efficient Location Encoding Method Based on Hierarchical Administrative District [⋆]

SangYoon Lee[1], Sanghyun Park[1], Woo-Cheol Kim[1], and Dongwon Lee[2]

[1] Department of Computer Science
Yonsei University, Korea
{sylee, sanghyun, twelvepp}@cs.yonsei.ac.kr
[2] School of Information Sciences and Technology
Penn State University, USA
dongwon@psu.edu

Abstract. Due to the rapid development in mobile communication technologies, the usage of mobile devices such as cell phone or PDA becomes increasingly popular. As different devices require different applications, various new services are being developed to satisfy the needs. One of the popular services under heavy demand is the Location-based Service (LBS) that exploits the spatial information of moving objects per temporal changes. In order to support LBS efficiently, it is necessary to be able to index and query well a large amount of spatio-temporal information of moving objects. Therefore, in this paper, we investigate how such location information of moving objects can be efficiently stored and indexed. In particular, we propose a novel location encoding method based on hierarchical administrative district information. Our proposal is different from conventional approaches where moving objects are often expressed as geometric points in two dimensional space, (x, y). Instead, in ours, moving objects are encoded as one dimensional points by both administrative district as well as road information. Our method is especially useful for monitoring traffic situation or tracing location of moving objects through approximate spatial queries.

Keywords: Location-Based Service, Road network, Moving object, Indexing

1 Introduction

Due to the recent development in mobile communication technologies, the usage of mobile devices such as cell phone or PDA becomes increasingly popular, and novel services are being developed to serve various needs. One of the popular services for mobile devices is the Location-based Service (LBS) that exploits the location information of moving objects (i.e., mobile devices). For instance, the following queries are utilizing the "location" of moving objects: "Find the location of a person with a phone number X.", "What is the nearest Thai restaurant

[⋆] This work was partially supported by Korea Research Foundation Grant (KRF-2004-003-D00302 and KRF-2005-206-D00015) to a hotel Y?", or "Where is the delivery truck, shipping the TV that I purchased over the Internet?", etc.

The LBS is the service that keeps track of the location information of moving objects per time unit, stores them into databases, and handles users' queries based on the stored location information. The queries used in the LBS can be categorized as spatial (i.e., finding moving objects within some spatial constraints), trajectory (i.e., finding moving paths of objects per some time units), and hybrid (i.e., both spatial and trajectory) queries [9].

In particular, moving objects in the context of LBS have the following challenges: (1) they have high update cost since databases have to update location information as time passes; (2) they have high storage cost since location information is typically multi-dimensional (i.e., object, time, location, etc.); (3) data to handle are large-scale since databases need to maintain temporal data (i.e., past and present); (4) they have high retrieval cost due to the large amount of data. Therefore, it is important to devise an indexing and query processing technique that can handle such a large-scale multi-dimensional spatio-temporal data efficiently.

In this paper, we investigate a data encoding method to enable effective indexing and query processing for such a setting. In conventional approaches (e.g., 3DR-tree [11], HR-tree [5], STR-tree [8], TB-tree [9], and MV3R-tree [10]), the location information of moving objects were expressed as a geometric coordinate (x,y) in two dimensional space. However, instead, we propose to express location information using both *hierarchical administrative district* and *road network* [3][7] in one dimensional space that, we believe, fits better the real world. For instance, if a moving object is in a building with a coordinate of latitude=125.58 and longitude=-37.34, then it can be expressed as a set of fields according to an administrative district such as city, road-name, road-block (e.g., Seoul, Main road, 165). Furthermore, by converting the fields into a binary string that has efficient ways to process queries, we overcome the aforementioned challenges of the LBS.

Our proposed scheme has at least three advantages: (1) it reduces the storage cost and dimensions of index by expressing location information in one dimensional space, instead of two dimensional space. This results in the improved query processing. (2) In real world, moving objects can only follow along the "roads". However, if one expresses location information as geometric coordinates, then one may include spaces where moving objects can never move into, so called *dead space*, incurring storage waste. (3) Since location information is based on the information of the administrative district, the results can be easily converted into address formats that are easier, as answers, for human users to interpret.

2 Proposed Encoding Method of Location Information

In many countries, addresses are often represented as a set of fields such as district name, road name, and location on the road. For example, the address of the City Hall of Seoul, Korea is represented as a triplet of (Seoul, Eulji road, 31).

Similarly, the address of the Natural History Museum of England is (London, Cornwell Road, -), where the third field is null. Exploiting this addressing scheme, one can easily encode the location of a moving object as a one-dimensional binary string. By adding more fields, it is trivial to extend the scheme to be able to support more general addresses. From here forward, to keep the presentation simple, we only focus on the triplet scheme, (district, road, location on road), to represent addresses within a specific country.

The procedure to encode the location of a moving object consists of three steps as follows: (1) obtain the address of the place at which the moving object is located and express it as a triplet, (2) transform each field of the triplet into a binary string, and (3) concatenate the three binary strings into a single binary string. The first and third steps are trivial, and thus we elaborate on the second step.

We first discuss the way to encode districts. For easier illustration, let us consider an imaginary country with 4 counties (A, B, C, D) as a whole and 8 cities (a, b, ..., g) in each county - a total of 32 districts to encode. The simplest encoding method is to use their lexicographical orders. That is, by using 2 bits for county names and 3 bits for city names, one can encode a district as a 5-bit string whose first two bits represent the lexicographical order of its county name and the remaining three bits represent the lexicographical order of its city name. For example, one can express the district "A county a city" as "00 000", the district "A county b city" as "00 001" and "B county a city" as "01 000".

Although this encoding scheme is simple to implement, it does not provide the information about the relative position of districts. For example, let us consider two moving objects, one located at the district "00 000" and the other at the district "00 001". Comparing these two binary strings, one can deduce that the two objects be in the same county but in different city. These two binary strings, however, do not provide any clue as to the relative positions of the two objects.

To overcome these limitation, we propose to use a mapping technique based on space-filling curves such as Z-ordering [6], R-ordering [1], and H-ordering [2]. A space-filling curve is a one-dimensional curve which visits every point within a multi-dimensional space. In order to represent the relative locations of districts more efficiently, we choose Z-ordering among various space-filling curves and modify it to start from the upper left corner rather than the lower left corner as in the original Z-ordering. The detailed algorithm to encode the districts contained in a region is given in Algorithm 1, and an illustrative example is shown in Fig.1.

Compared to the encoding method based on lexicographical orders, the proposed encoding method produces more informative binary strings. Let us consider the two moving objects again, one located at the district "00 000" and the other at the district "00 001". In addition to the facts that the two objects are in the same county but are in different city, we can infer more facts: (1) since the first two bits for cities are all "00", the cities are located at northwest area of the county, and (2) since the last bits for cities are different, the city where the first object is located is north of the city where the second object is located.

Algorithm 1: Mapping administrative districts into binary strings.

```
1 Compute the central point of each district.
2 Divide the region into two sub-regions, north and south, so that the numbers
  of central points in both north and south are similar.
3 If region north has more than one central point, divide it into two sub-regions,
  north_east and north_west, so that the numbers of central points in both
  north_east and north_west are similar.
4 Do the same for region south symmetrically.
5 For each sub-region obtained from Steps 3 and 4, if it contains more than one
  central point, repeat Steps 2 - 4.
6 Considering the division process undergone, map the central point of each
  district onto a two-dimensional space.
7 Using a modified Z-ordering, assign a binary string to each district.
```

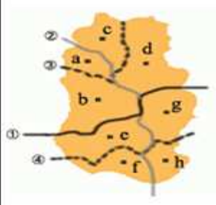
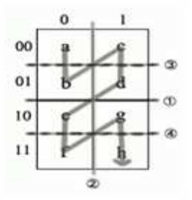
			a: 000 b: 001 c: 010 d: 011 e: 100 f: 101 g: 110 h: 111
(a) compute the central point of each district in A county	(b) divide the region into two sub-regions repeatedly	(c) map the central points onto a two dimensional space	(d) assign a binary string to each district

Fig. 1. An example which illustrates how Algorithm 1 works on *A* county.

The algorithm to encode the roads within a district is not much different from Algorithm 1. The changes needed to be made on Algorithm 1 are as follows: (1) every instance of word "district" is to be replaced with word "road", and (2) every instance of word "region" is to be replaced with word "district".

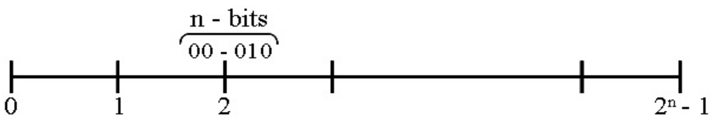


Fig. 2. A road which is partitioned into $2^n - 1$ units of the same size.

Now let us consider the way to encode the location on road. We first partition the road into $2^n - 1$ units of the same size, and then represent each boundary as an n-bit binary string as shown in Fig.2. Lastly, we choose the boundary nearest from an object and use its binary string as the location of the object on road.

The proposed encoding scheme has the following characteristics: (1) one can find out the *lowest* common administrative district by extracting the longest common prefix of a given set of binary strings, and (2) a district containing a set of lower districts can be represented by the range of binary strings; for example, county "A" in Fig.1 is represented by the range $[00000, 00111]$.

3 System Organization

As shown in Fig.3, our LBS implementation consists of two sub-systems for *population* and *query processing*. The population sub-system is responsible for collecting the information of moving objects and storing it into databases, and the query processing sub-system is in charge of answering to the queries about the moving objects. To support the proposed encoding scheme, in addition, the

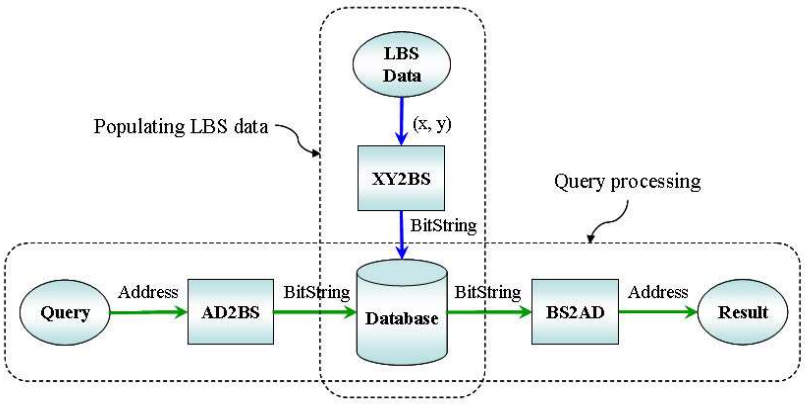


Fig. 3. LBS system which uses the proposed location encoding scheme.

Module XY2BS converts a two dimensional coordinate denoting the location of a moving object into the equivalent binary string. To expedite the conversion process, XY2BS maintains an R-tree built from the roads in administrative districts. For a given road R, let bitstring(R) and rectangle(R) denote the binary string of R and the rectangle for the two end points of R, respectively. For each road R in districts, then, the R-tree stores rectangle(R) and bitstring(R) in one of its leaf nodes. Algorithm 2 describes how XY2BS makes use of the R-tree to quickly convert two dimensional points to corresponding binary strings.

It is much more intuitive for users to ask queries using real-life address such as "Seoul, Main road, 100" than using coordinates such as "longitude=-65, latitude=+45". Similarly, it is also preferable to use such real-life address in the query results. Therefore, in our prototype, we assume that both users' queries and query results are in the real address format. Module AD2BS converts this real-life addresses into equivalent binary string representations, and

Algorithm 2: Utilizing an R-tree to quickly convert a two dimensional point, (x, y), into the equivalent binary string.

```
1 Generate the rectangle $uMBR$ by expanding $x$ to its left and right by $uR$, and
  expanding $y$ up and down by $uR$. $uMBR$ is then expressed as ($[x - uR, x +
  uR], [y - uR, y + uR]$). Here, $uR$ is a system parameter used for determining
  the nearness of roads from a two dimensional point.
2 Search the R-tree for the roads whose MBRs overlap $uMBR$.
3 From the roads obtained in Step 2, select the road $R$ whose Euclidean distance
  to (x, y) is the smallest.
4 Project (x, y) onto the road $R$. Let (x', y') denote the coordinate of (x, y) after
  the projection.
5 Using the relative position of (x', y') on the road $R$, calculate the binary string
  for (x', y').
6 Concatenate bitstring($R$) and the binary string for (x', y').
```

module BS2AD converts binary strings back to equivalent real-life addresses. For rapid conversion to binary strings, AD2BS maintains a B-tree where district and road names are used as a key and binary strings are stored at leaf nodes. For fast conversion to real-life addresses, BS2AD also maintains a B-tree where binary strings are used as a key, and district and road names are stored at leaf nodes.

4 Query processing

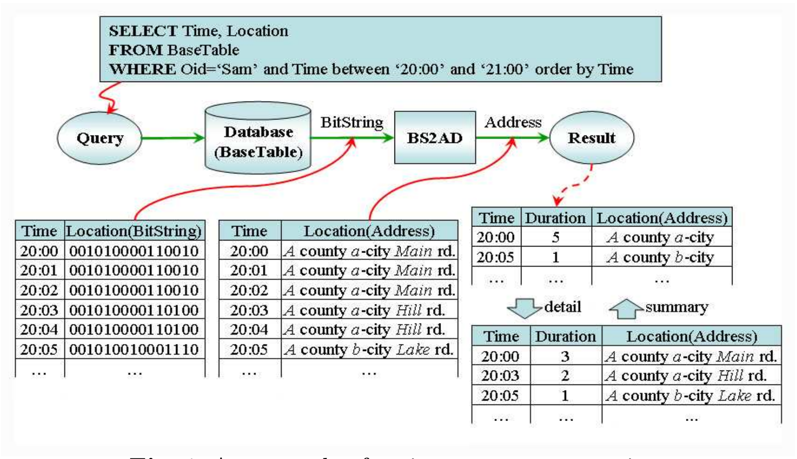
This section describes how our LBS implementation processes typical LBS range and trajectory queries.

4.1 Range query processing

Range queries are to find the moving objects within a specific region during a given time interval or to find a set of time intervals during which a specific moving object was within a given region. Let us consider an example query: "Find all cell phone users who have been in *b* city of *A* county during the time interval [10 pm, 11 pm]". To answer this query, the system first calls module AD2BS to convert the district name (i.e., "*A* county, *b* city") to the corresponding binary string. Since there are likely to be more than a single road in the given district, the district name is expressed as a range of binary strings. The system then searches the database using the range of binary strings and the time interval (i.e. [10 pm, 11 pm]) as a query predicate.

4.2 Trajectory query processing

Trajectory queries are to retrieve the path on which a moving object has traversed during a given time interval. Let us consider an example query: "Between the time interval [10 pm, 11 pm], where has Sam been moving around?". If the system represents the locations of moving objects as two dimensional geometric points, the answer to such a query consists of a set of line segments and thus can be meaningfully displayed only on electronic maps. However the answers from the proposed LBS system can be easily converted to real-life addresses and thus can be delivered to users in text or voice format (in addition to being useful on electronic maps as well).



```
SELECT Time, Location
FROM BaseTable
WHERE Oid='Sam' and Time between '20:00' and '21:00' order by Time
```

Time	Location(BitString)
20:00	001010000110010
20:01	001010000110010
20:02	001010000110010
20:03	001010000110100
20:04	001010000110100
20:05	001010010001110
...	...

Time	Location(Address)
20:00	*A* county *a*-city *Main* rd.
20:01	*A* county *a*-city *Main* rd.
20:02	*A* county *a*-city *Main* rd.
20:03	*A* county *a*-city *Hill* rd.
20:04	*A* county *a*-city *Hill* rd.
20:05	*A* county *b*-city *Lake* rd.
...	...

Time	Duration	Location(Address)
20:00	5	*A* county *a*-city
20:05	1	*A* county *b*-city
...	...	...

Time	Duration	Location(Address)
20:00	3	*A* county *a*-city *Main* rd.
20:03	2	*A* county *a*-city *Hill* rd.
20:05	1	*A* county *b*-city *Lake* rd.
...	...	...

Fig. 4. An example of trajectory query procesing.

To process trajectory queries, the system first searches the database using the object and time interval information, and then sort the result in ascending order of time as shown in Fig.4. The system then calls module BS2AD to convert the binary strings in the result into the corresponding administrative district addresses, and finally sends out the result in text or voice format to users. When showing the result to users, the system may represent a set of adjacent rows as a single row by extracting their common prefixes. That is, it is feasible that the result is displayed in the unit of 'county' first and, whenever necessary, in the unit of 'city' (similar to the drill-down of OLAP applications)

5 Experiment

To evaluate the effectiveness of the proposed location encoding scheme, we performed experiments with real district and road data of a specific region in Seoul, Korea. The region we used for experiments consists of 2 counties (actually 'gu' in Korea), 46 cities (actually 'dong' in Korea), and 387 roads. We created synthetic moving objects within this region and let them follow the roads in a random fashion for 500 minutes. We then observed their locations every 1 minute. At first the collected data were stored in 3DR-tree as a triplet of (timestamp, x-coordinate, y-coordinate) and then stored in 2DR-tree as a pair of (timestamp, binary string representation of location). Identifiers of moving objects were used as a key and thus stored in leaf nodes of 3DR-tree or 2DR-tree.

We evaluate the effectiveness of the proposed encoding scheme by comparing the 3DR-tree with the 2DR-tree in terms of index size and query processing time. The machine for the experiments was a personal computer with a Pentium-IV 2.6 GHz CPU, the main memory of 512 MB, and the operating system of Linux Fedora core 3.

5.1 Index size

While increasing the number of moving objects from 400 to 2,000, we measured the sizes of the 2DR-tree and the 3DR-tree. Since 500 location data were collected from each object, the total number of records stored in the indexes was 200,000 when there were 400 objects and 1 million when there were 2,000 objects. As shown in Table 1, the 2DR-tree which stores the locations in binary string representation consumed about 58% of the storage space spent by the 3DR-tree. Therefore, the reduction ratio of the index size was approximately 42% and this reduction ratio increased slightly when the number of moving objects became 2,000.

Table 1. Sizes of 2DR-tree and 3DR-tree.

# of moving objects (Tuples)	Size of 3DR-Tree (KB)	Size of 2DR-Tree (KB)	Reduction ratio (%)
400 (200,000)	10,973	6,393	41.7
800 (400,000)	22,467	12,779	43.1
1200 (600,000)	34,342	19,329	43.7
1600 (800,000)	46,221	25,906	44.0
2000 (1,000,000)	58,218	32,565	44.1

5.2 Query processing time

While increasing the number of moving objects from 400 to 2,000, we observed how long it takes for the 2DR-tree and the 3DR-tree to process range queries and trajectory queries. We generated 1,000 queries for each query type and measured the time elapsed to process all the 1,000 queries.

We first performed the two types of range queries: "Find a set of time intervals during which a specific object was in a given *city*" (type 1) and "Find a set of time intervals during which a specific object was in a given *county*" (type 2). As shown in Fig.5, the query processing times of both the 2DR-tree and the 3DR-tree increase linearly as the number of moving objects grows, but the increase ratio of the 2DR-tree is smaller than that of the 3DR-tree. Compared to the 3DR-tree, the 2DR-tree achieved about 98% performance improvement for the queries in type 1 and 67% to 69% improvement for the queries in type 2. Such an improvement seems to be achievable because the proposed scheme reduces the index size significantly and makes search regions become one-dimensional ranges rather than two-dimensional rectangles.

We then performed another two types of range queries: "Find the moving objects which were within a given *city* at any time in the first 250 minutes" (type

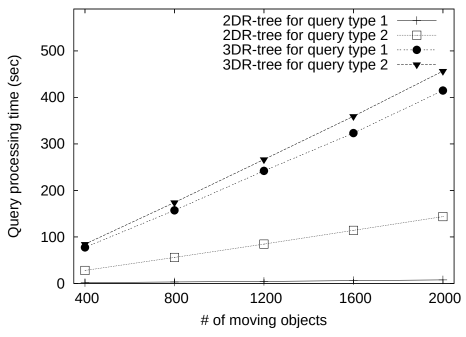


Fig. 5. Elapsed time to process type 1 and type 2 queries.

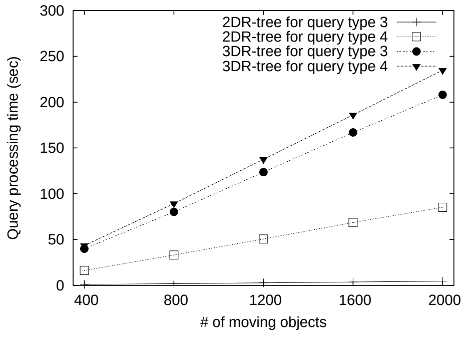


Fig. 6. Elapsed time to process type 3 and type 4 queries.

3) and "Find the moving objects which were within a given *county* at any time in the first 250 minutes" (type 4). To process these types of range queries, we have to search the index using the rectangles representing the time and location constraints. Remember that the 2DR-tree and the 3DR-tree express locations as one-dimensional binary strings and two-dimensional geometric points, respectively. Therefore, the search regions for the 2DR-tree become two dimensional while the search regions for the 3DR-tree become three dimensional. As shown in Fig.6, the performance improvement of the 2DR-tree becomes larger as the number of moving objects increases. As a result, the 2DR-tree achieved improvement up to 98% for the queries in type 3, and up to 64% for the queries in type 4.

We lastly performed a trajectory query: "Where has a specific object been moving around for the first 250 minutes?" (type 5). To process such a query, we have to traverse down the index using the time constraint. Since traversing the 2DR-tree is more effective than traversing the 3DR-tree in terms of CPU and I/O cost, the 2DR-tree achieved about 44% performance improvement when there were 400 objects and about 41% when there were 2,000 objects.

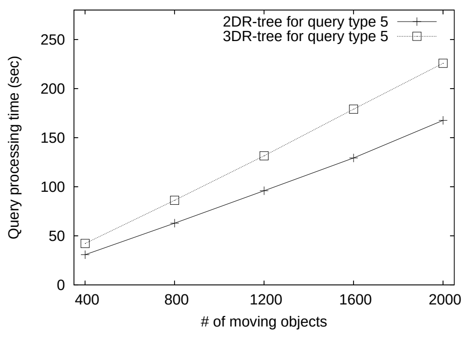


Fig. 7. Elapsed time to process type 5 queries.

6 Conclusion

In this paper, we have proposed an effective location encoding method that uses the information in the hierarchical administrative district and the road network of real world. Our method captures moving objects as binary strings in one dimensional space instead of conventional (x, y) coordinates in two dimensional space, and thus can reduce storage cost by upto 44% while improving query processing by 64% to 98%. The benefits of our proposal include: (1) it improves upon previous indexing and query processing algorithms by exploiting binary strings; (2) it is easy to drill-down or roll-up query results in a hierarchical administrative district; (3) since it uses the ontologies of administrative district that are intuitive to human users, it is suitable to display query results as text or voice even without electronic maps.

One of the obstacles of the Location-based Service (LBS) is how to reduce the rapidly increasing spatio-temporal data without sacrificing query performance. To address this problem, we plan to exploit the property that when two binary strings of location information share the same prefix, two corresponding moving objects on the road network must be located in the same administrative district. That is, the location information of moving objects can be further compressed per administrative district by using common prefixes.

References

1. C. Faloutsos, “Gray Codes for Partial Match and Range Queries”, IEEE Trans. on Software Engineering, 14(10), pp. 1381-1393, 1988.
2. C. Faloutsos and S. Roseman, “Fractals for Secondary Key Retrieval”, In Proc. ACM PODS, pp. 247-252, 1989.
3. S. Gupta, S. Kopparty, and C. Ravishankar, “Roads, Codes, and Spatiotemporal Queries”, In Proc. ACM PODS, pp. 115-124, 2004.
4. A. Guttman, “R-trees: A Dynamic Index Structure for Spatial Searching”, In Proc. ACM SIGMOD, pp. 47-54, 1984.
5. M. A. Nascimento and J. R. O. Silva, “Towards Historical R-trees”, In Proc. ACM Symposium on Applied Computing, pp. 235-240, 1998.
6. J. A. Orenstein and T. H. Merrett, “A Class of Data Structures for Associative Searching”, In Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, pp.181-190, 1984.
7. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, “Query Processing in Spatial Network Databases”, In Proc. VLDB Conference, pp. 802-813, 2003.
8. D. Pfoser, Y. Theodoridis, and C. S. Jensen, “Indexing Trajectories in Query Processing for Moving Objects”, Chorochronos Technical Report, CH-99-3, 1999.
9. D. Pfoser, C. S. Jensen, and Y. Theodoridis, “Novel Approaches in Query Processing for Moving Objects”, In Proc. VLDB Conference, pp. 395-406, 2000.
10. Y. Tao and D. Papadias, “MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries”, In Proc. VLDB Conference, pp. 431-440, 2001.
11. Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis, “Spatio-Temporal Indexing for Large Multimedia Applications”, In Proc. IEEE International Conference on Multimedia Computing and Systems, pp. 441-448, 1996.