March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

# Search Method for Moving Sensors on a Big Data-based Road Network

Jeongmin Park[1], Seunghwa Lee[2], Jeong-Joon Kim[*3]
[1]Professor, Department of Computer Engineering, Korea Polytechnic University, 237 Sangidaehak-ro Gyeonggi-Do Siheung, 15073, South Korea
[2]Professor, Division of Information and Communication, Baekseok University, 76, Munam-ro, Dongnam-gu, Cheonan-si, Chungcheongnam-do, South Korea
[*3]Professor, Department of Software, Anyang University, 22 Samdeok-ro, Manan-gu, Anyang, Gyeonggi-do, 14028, South Korea
jmpark@kpu.ac.kr[1], sh.lee@bu.ac.kr[2], jjkim@anyang.ac.kr[*3]

***Article Info***
***Volume 83***
***Page Number: 4638 - 4650***
***Publication Issue:***
***March - April 2020***

***Article History***
***Article Received****: 24 July 2019*
***Revised****: 12 September 2019*
***Accepted****: 15 February 2020*
***Publication****: 26 March 2020*

***Abstract***
KNN search, which is a representative location-based query, is used to search the K moving sensors with the closest network distance from the query point by setting the network peripheral object as the query point. Typical examples of the KNN search method in a road network include the IER and INE techniques. However, existing KNN search methods have the disadvantage that the storage space increases as the number of moving sensors increases, and the search time length increases due to the inefficient search process. Therefore, this paper proposes a Middle Point-based QR-tree (MPBQR) based on a QR-tree using the midpoint and an efficient KNN search method that uses MPBQR to solve the problem of existing KNN search methods in road network environments and to support the efficient processing of large capacity sensor data.

***Keywords:*** *Bigdata, Moving Sensor, Road Network, Quad-tree, R-tree.*

## 1. Introduction

This paper focuses on KNN [1,2] search and range retrieval, which are most commonly used in road network environments [3,4]. Typical examples related to searching moving sensor data in a road network include the IER and INE techniques. The IER technique [5] finds sensor data candidates using Euclidean distance and search sensor data using the actual network distance. Since the IER technique uses the actual network distance to repeat the Euclidean region query several times, the search performance drops significantly. The INE technique [5] retrieves the POI from the query point while sequentially expanding the line strings that constitute the network. Since the INE technique searches the R-tree storing the moving sensor several times, its search performance is poor. Therefore, this paper proposes the Middle Point-based QR-tree (MPBQR), which is a QR-tree [6] that uses the midpoint (MP, Middle Point) and the KNN and range search method based on MPBQR to solve the problem of existing search methods of moving sensors and supports more efficient processing of large capacity sensor data.

MPBQR proposed in this paper can be used in various GIS applications such as ITS, telematics, and LBS [7] services, which mainly use KNN search and range search because the search process is uncomplicated and its structure does not depend on a specific system. In addition, it can be extended to KNN search and range search

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

for moving sensors used in services such as when monitoring moving sensor data.

## 2. Related Works

### 2.1. Incremental Network Expansion (INE)

INE is a method of performing KNN searches by extending line strings individually [5]. In the KNN search, when a query point is given, it first searches for a line string that contains the query point. Thereafter, the adjacent line strings are expanded individually from the corresponding line string, and whether there is a movement sensor in the corresponding line string is retrieved using the R-tree [8,9] in which the movement sensor is stored. The above process is repeated until the user finds the K moving sensors requested by the user to expand the line string. Figure 1 shows an example KNN search using INE.

As shown in Figure 1, when there is a moving sensor in the road network, P represents the moving sensor point, q represents the query point, and n represents the node. When processing a KNN query through INE, the first step is to find the line string that contains the query point. Since a two-dimensional R-tree that consists of a line string is constructed, a line string that contains the query point is searched.

Both nodes of the searched line string are inserted into the priority queue in ascending order of distance. The existence of the moving sensor in the line string is determined using the position of the detected line string that is inquired to the R-tree in which the moving sensor is stored. In Figure 1, since no moving sensor is found, the nearest node is popped into the priority queue to search for another line string connected to that node.

INE manages the linking relationship between line strings as a linked list with nodes. Thus, it is possible to extend from node n1 to the line string n1n7. After the node n7 of the extended line string is inserted into the priority queue, the moving sensor' s existence is detected in the extended line string using the R-tree that stores the moving sensor.

Here, finding the moving sensor P5 in the line string n2n4 means that the network distance from the query point to P5 can be calculated. If this network distance is greater than the distance from the query point to the next node to be popped, P5 is the closest moving sensor, a result is returned, and the query is terminated. Otherwise, since there may be a moving sensor closer to P5, the query is continued by extending the line string. In Figure 1, the network distance from the query point to P5 is seven, and the network distance of node n4 to be popped next is eight, so P5 is returned as a result and the search is terminated.

INE has the advantage of saving the storage space because it stores the line string and moving sensor in each R-tree. However, the search performance deteriorates when expanding a line string because each time it searches, it used the R-tree in which the moving sensor is stored to check for the presence of the moving sensor on the line string.

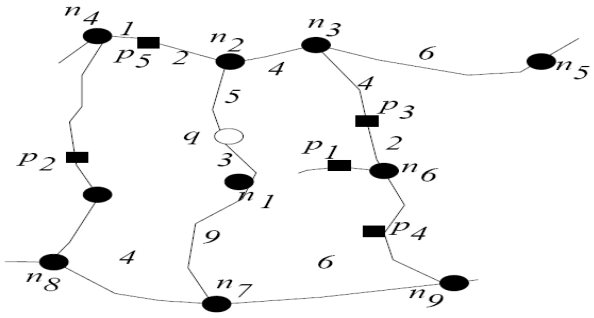



**Figure 1. Example of KNN search using INE.**

### 2.2. Incremental Euclidean Restriction (IER)

IER is a method that performs a KNN search using the property that the network distance between two points is always ⩾ the Euclidean distance [1,2]. First, K candidate moving sensors nearest the query point are searched

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

based on the Euclidean distance [10] using the R-tree that is storing the moving sensor. Then, the actual network distance from the candidate moving sensors to the query point is calculated and the region query on the Euclidean space is performed with the largest value as the radius.

In the presently retrieved moving sensor, the nearest moving sensor based on the Euclidean distance is searched to determine the network distance from the query point except for the candidate moving sensors in the previous step. If this distance is closer to the moving distance of the candidate moving sensor than the moving distance from the query point, the Kth moving sensor is dropped from the candidate and the newly detected moving sensor is inserted into K candidate moving sensors. The above process is repeated to search for K closest moving sensors at the query point. Figure 2 shows an example of KNN search processing through IER.

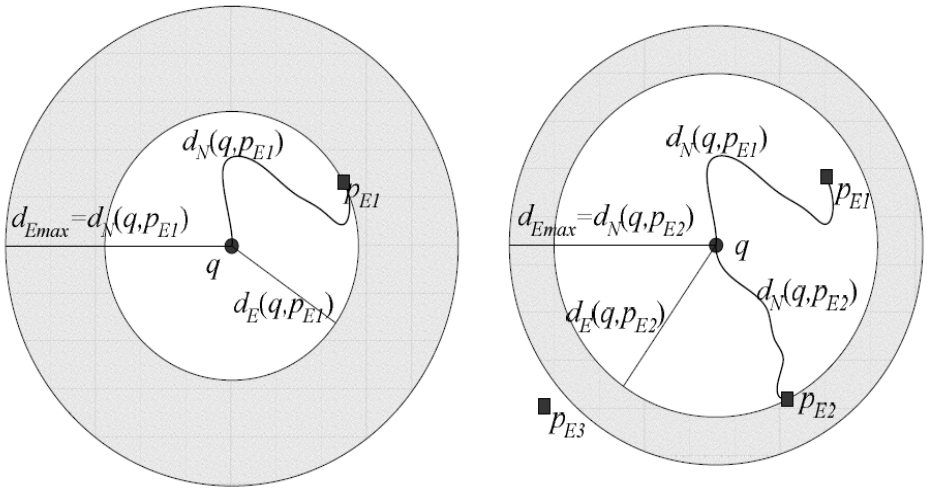



**Figure 2. Example of KNN search through IER.**

In Figure 2, q denotes the query point and P denotes the moving sensor coordinates. When the query point q is given, IER finds the closest moving sensor PE1 on the Euclidean distance. The IER searches the moving sensor using the R-tree that stores the moving sensor in the same manner as the INE.

After this, the network distance of PE1 is calculated from the query point. The network distance to PE1 is defined as dEmax and the Euclidean distance is defined as dE. From the query point, the moving sensor is searched between the dE and dEmax through the area query in Euclidean space. The network distances of the retrieved moving sensors are all calculated to determine whether the network distance is smaller than dEmax.

In the case of Figure 2, since the network distance of PE2 is less than dEmax from the query point, PE1 is excluded from the result set and the values of dEmax and dE are changed to PE2 after inserting PE2. In the case of PE3, the Euclidean distance is larger than the changed dEmax, so PE2 is returned as a result and the search is terminated.

However, since both the calculation of the actual network distance from the candidate movement sensors to the query point and the area query regarding the Euclidean space that has the largest value as the radius are repeated many times, the search performance is greatly degraded.

## 3. Middle Point-based QR-tree (MPBQR)

### 3.1. Overview of MPBQR

The proposed system is shown in Figure 3. The sensors located around the road provide sensor

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

data for road conditions such as temperature, humidity, weather, etc. and messages about abnormal conditions such as earthquakes, fires, car accidents, the speed of surrounding cars, pedestrian count, pedestrian crossing waiting times, traffic light information, etc. are collected from the road network such as bridges and tunnels and stored in the Hadoop Distribute File System (HDFS) of this system, which uses HDFS and Map Reduce. HDFS stores various collected sensor data, and Map Reduce plays the role of storing, updating, and searching through the MPBQR presented in this paper. Finally, the complexity of the traffic environment will be eliminated, and rapid disaster propagation, pedestrian safety enhancement and facility risk detection are possible.

MPBQR consists of a Quad-tree, a cell component, an LN component, an MP component, a cell connection component, a cell R-tree, and an MP R-tree. Figure 3 shows an example of generated MPBQR when cell segmentation is performed for a road network and moving sensor.

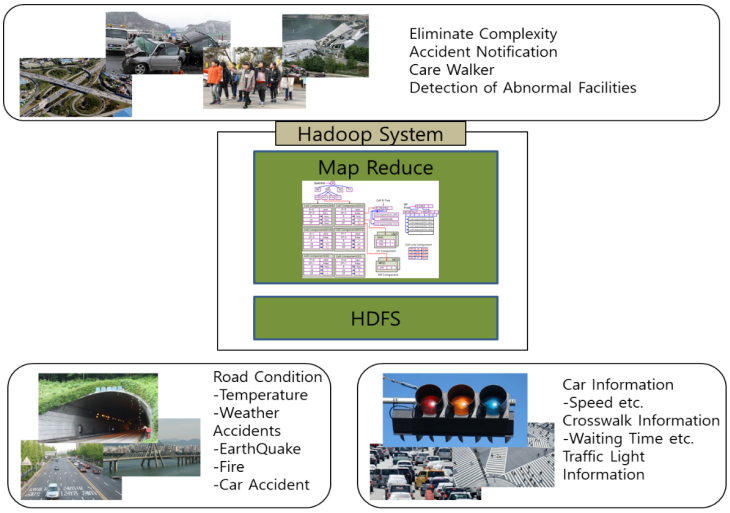
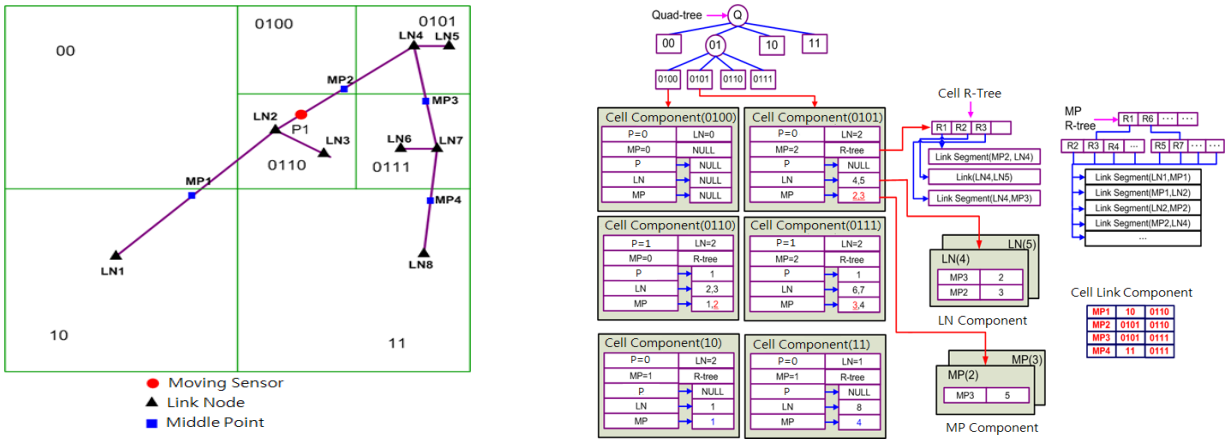



**Figure 3. MPBQR Architecture.**




(a) Cells created by moving sensors and links (b) Storage structure example

**Figure 4. MPBQR Architecture.**

In Figure 4, the circle represents the movement sensor and is represented by Pi, the triangle represents the link node and is denoted by LNi, while the rectangle represents the middle point and is denoted by MPi. Since seven cells are generated as in the example, the seven terminal nodes of 00, 0100, 0101, 0110, 0111, 10, and 11 are generated in the quad-tree that constitutes the

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

MPBQR. There are zero mobile sensors, two LNs, and two MPs stored in the cell component indicated by the 0101 terminal node, and IDs 4 and 5 of the stored LNs and IDs 2 and 3 of the MPs are stored. The cell R-tree of cell 0101 stores the link in which the link node is included in the cell 0101 or link segments (MP2, LN4), (LN1, LN3), and (LN4, MP3).

The MP component of cell 0101 stores the network distance between MP2 and MP3 stored in the cell component of cell 0101. The LN component of cell 0101 stores the network distance from the two LNs stored in the cell component of cell 0101 to MP2 and MP3. The cell connection component stores the cell ID that stores the MP. That is, MP1 stores 10 and 0110, MP2 stores 0101 and 0110, MP3 stores 0101 and 0111, and MP4 stores 11 and 0111. In KNN search and range search, this information is used to expand the cell and perform the search. Before describing MPBQR generation, we define Inner_MP and Outer_MP, which are used for cell division and merging. Inner_MPS is a set of MPs stored only in the cell components of a given cell and sibling cells, and Outer_MP is the set of MPs excluding Inner_MP in all MPs stored in the cell components of a given cell and sibling cells.

When MP1, MP2, MP3, and MP4 are generated as shown in Figure 3, the Inner_MP of cells 0100, 0101, 0110, and 0111 is {MP2, MP3} and Outer_MP is {MP1, MP4} because MP1 and MP4 are also stored in cells 10 and 11; although the MPs included in cells 0100, 0101, 0110, and 0111 are respectively MP1, MP2, MP3, and MP4, Inner_MP becomes MP2 and MP3, and Outer_MP becomes MP1 and MP4.

### 3.2. MPBQR Creation

When MP1, MP2, MP3, and MP4 are generated as shown in Figure 3, the Inner_MP of cells 0100, 0101, 0110, and 0111 is {MP2, MP3} and Outer_MP is {MP1, MP4} because MP1 and MP4 are also stored in cells 10 and 11; although the MPs included in cells 0100, 0101, 0110, and 0111 are respectively MP1, MP2, MP3, and MP4, Inner_MP becomes MP2 and MP3, and Outer_MP becomes MP1 and MP4.

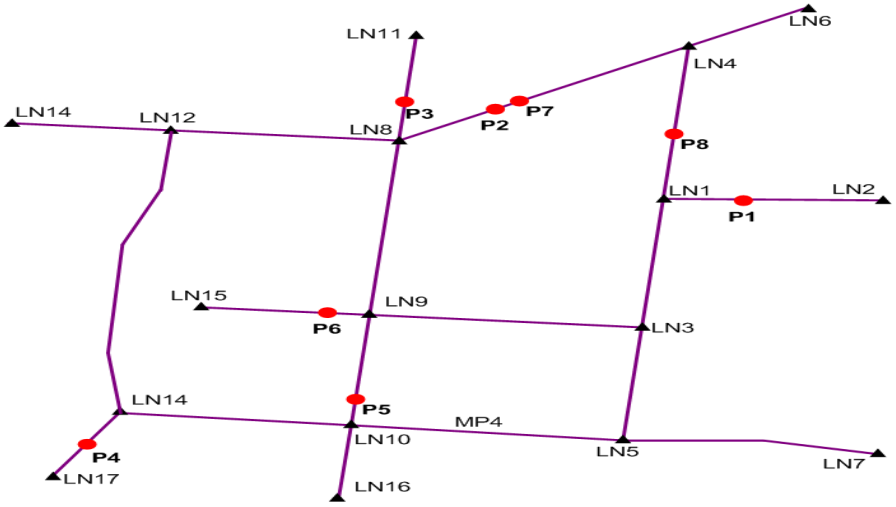
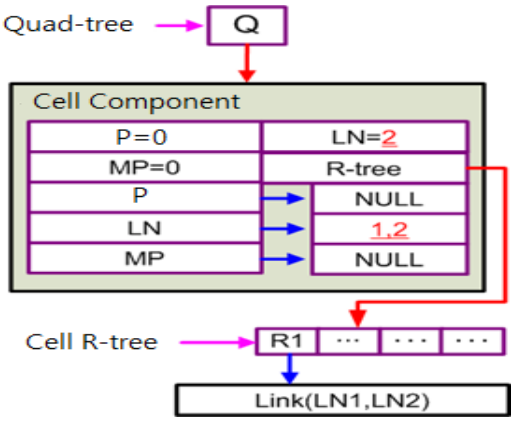



**Figure 5. Input data.**

Before inputting the link data, the initial region is set using the values x_min, x_max, y_min, and y_max of the link data to be input, and the root node and the cell component of the MPBQR are generated; the link is then inserted. Figure 6 shows an example of inserting the first link.

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650


(a) Insertion of cell link

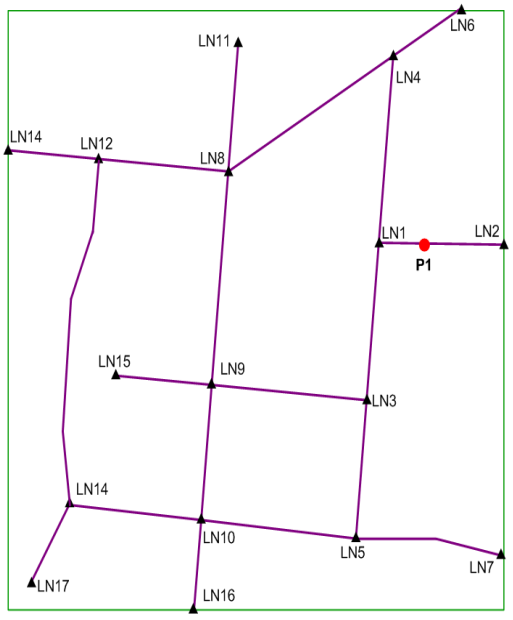



(b) Storage structure example

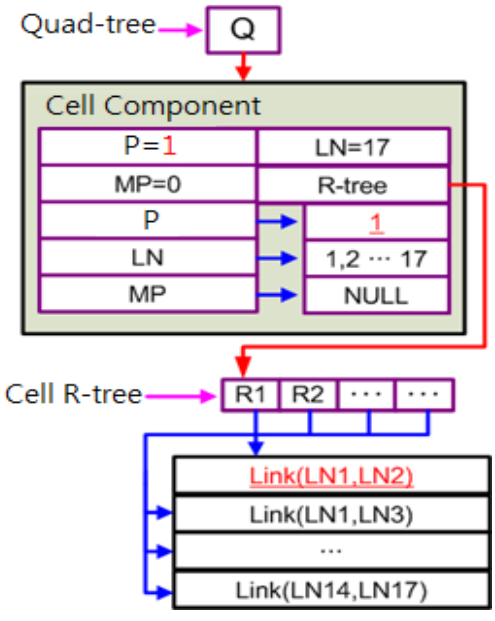
**Figure 6. Example of first link insert.**

As shown in Figure 6, by inserting the first link (LN1, LN2), two link nodes LN1 and LN2 included in the link are stored in the cell component and the link (LN1, LN2) is stored in the cell R-tree. Insertion is performed for all links in the same manner as the first link insertion process.

After all links have been inserted, the moving sensor is inserted, during which cell division occurs. In this example, the cell division criterion is set to six. Insertion is performed for the first moving sensor; it is inserted into the cell component of the cell in which the link or link segment that contains the moving sensor is located. Figure 7 shows the insertion example for the moving sensor coordinate 1.




(a) Insertion of the moving sensor into cell




(b) Storage structure

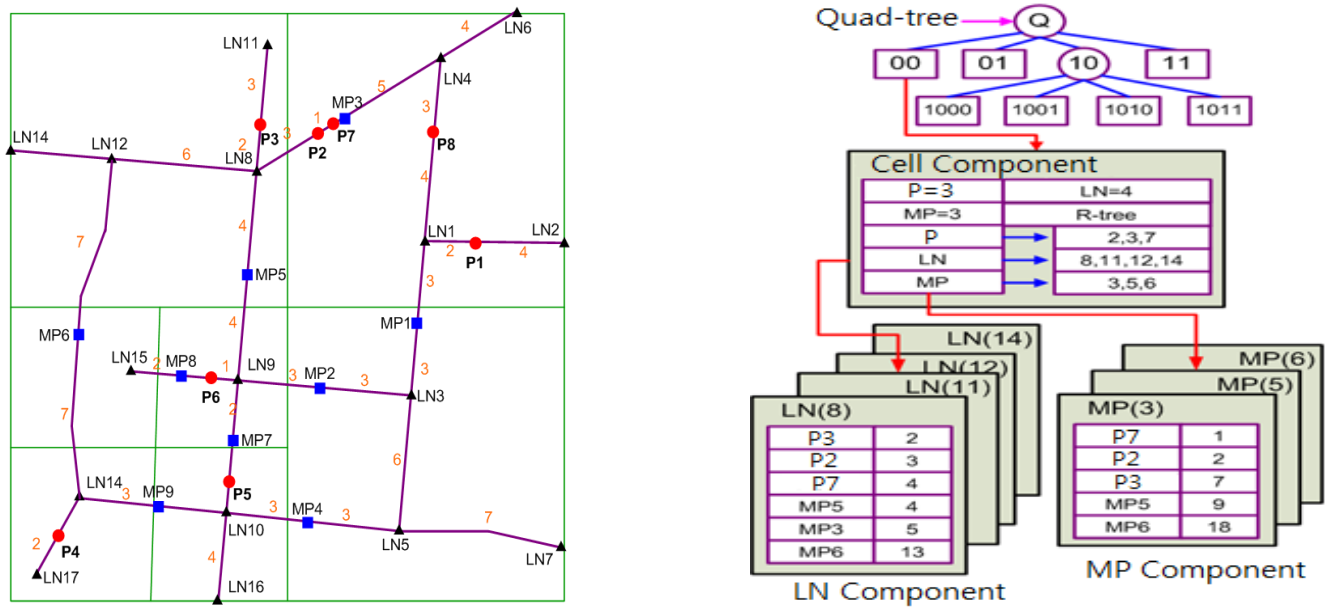
**Figure 7. Insertion example of the moving sensor coordinate.**

As shown in Figure 7, the terminal node area where the moving sensor is located is first searched. Here, the moving sensor is located in the root terminal node area. Then, whether the link or link segment stored in the cell R-tree of the next root terminal node includes the moving

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

sensor is checked. Since moving sensor 1 is included in the links (LN1, LN2), it is inserted into the root cell component. The insertion of the remaining moving sensor from 2 to 7 is performed using the same method as for moving sensor 1.

After all moving sensors stored in the cell component of the root cell have been inserted, the root cell component and root cell R-tree are deleted. Finally, an LN component and MP component are generated using the link or link segment stored in each cell of the cell component and the cell R-tree. The network distance calculation uses the Daikstra algorithm; Figure 8 shows an example of creating an LN component and MP component.



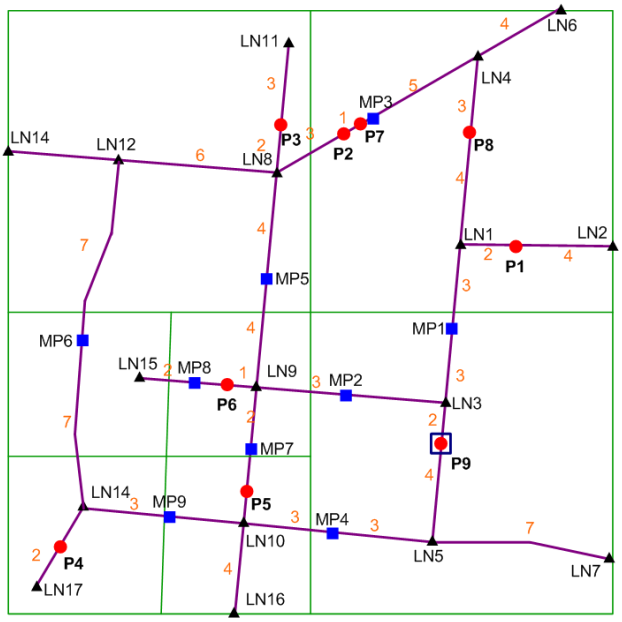

(a) Completion of link and POI insertion into the cell

(b) Storage structure

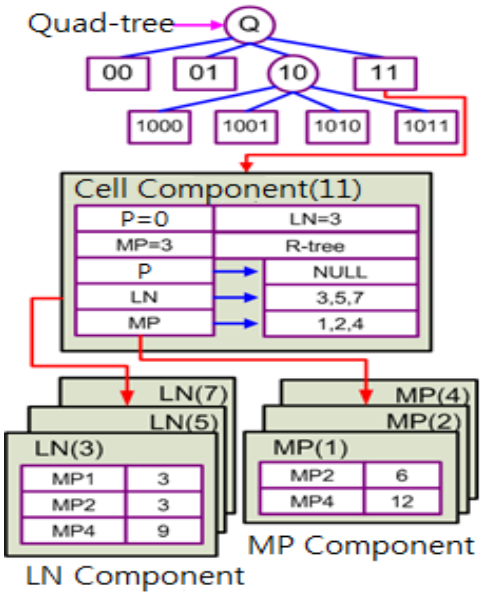
**Figure 8. Example of creating LN component and MP component.**

As shown in Figure 8, an LN component and MP component are created for each cell. The LN component of cell 00 stores the network distances from each LN of cell 00 to the moving sensor and the MP in ascending order. In the MP component of cell 00, the network distances from each MP of cell 00 to the moving sensor and MP are stored in ascending order. Moving sensor insertion is the process of inserting a moving sensor in addition to the MPBQR already created. Therefore, the process of inserting the moving sensor is the same as the process of inserting the moving sensor in the MPBQR generation process. However, in the MPBQR generation process, after completing the insertion of all moving sensors, the LN and MP components are generated by collectively calculating the network distance. However, when the mobile sensor is inserted, the LN and MP components of the divided cell are newly created when the cell is divided by inserting the moving sensor. When the cell is undivided, the corresponding moving sensor is inserted into the LN and MP components. Figure 9 shows an example insertion of moving sensor 9.

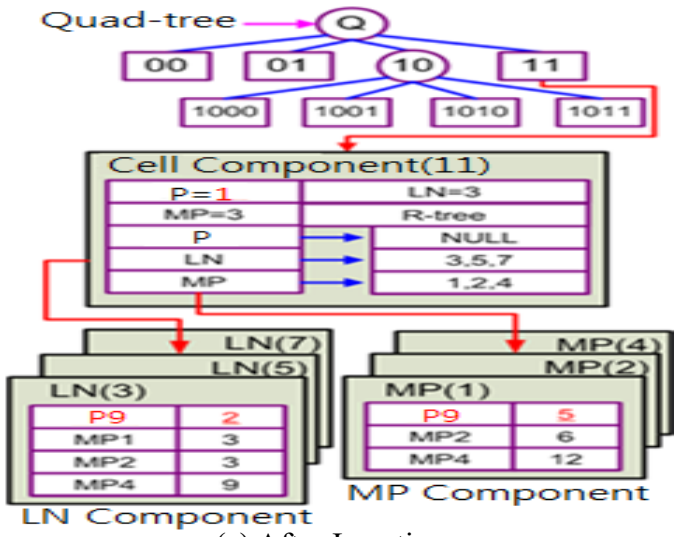
March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650




(a) Completion of link and POI insertion into the cell




(b) Storage structure



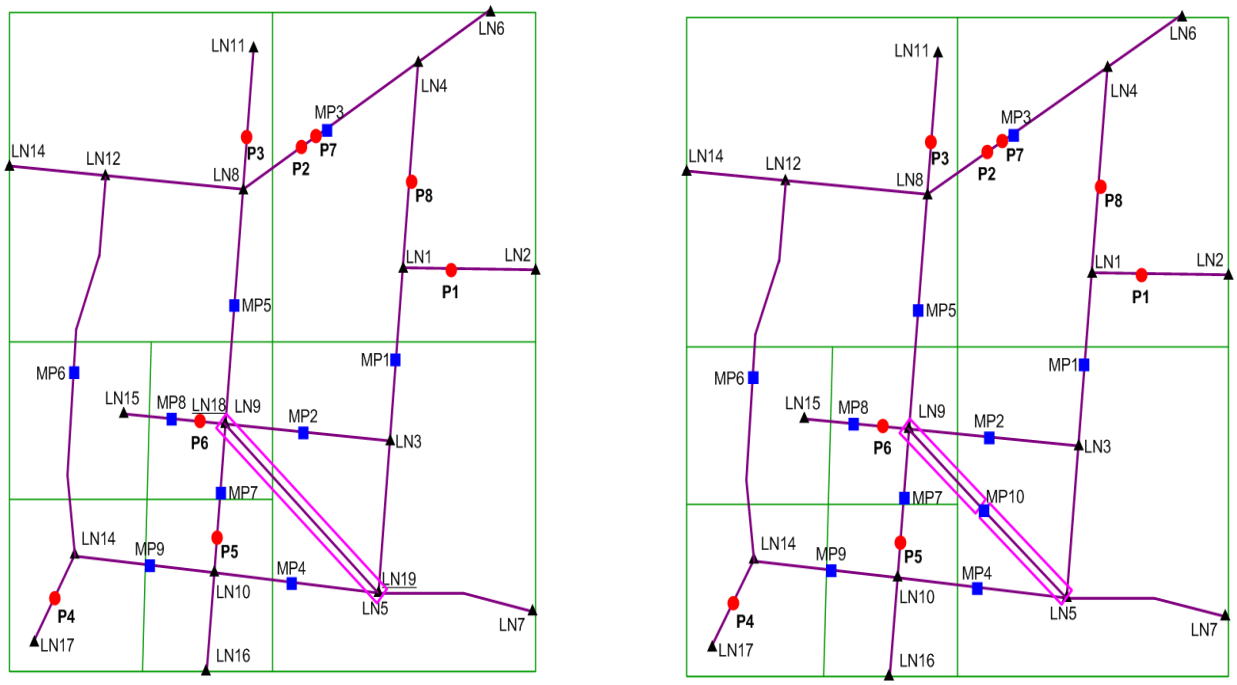

(c) After Insertion

**Figure 9. Insertion example for moving sensor 9.**

As shown in Figure 9, when moving sensor 9 is inserted, it is located on the links (LN3, LN5) of cell 11 so that moving sensor 9 is inserted into the cell component of cell 11. Since cell 11 into which moving sensor 9 is inserted does not exceed the division criterion, moving sensor 9 is inserted into the LN and MP components of cell 11.

The insertion of a link is the process of inserting a new link into the already-created MPBQR. When inserting a link, first check the overlap between the link node of the link to be inserted and that stored in the MPBR, and then insert the link after confirming the link node of the link to be inserted. If the cell in which the link has been inserted exceeds the division criterion, the LN and MP components are generated using the network of divided cells after performing the cell division operation. If the division criterion has not been exceeded, an LN component and MP component are newly created specifically for the cell in which the link has been inserted. Figure 10 shows an link insertion example (LN18, LN19).

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650




(a) Link insertion (b) Link split

**Figure 10. Example of link(LN18, LN19) insertion.**

As shown in Figure 10, first check to see whether there are link nodes that overlap the two link nodes of the link to be inserted. In the example, LN18 overlaps with LN9 and LN19 overlaps with LN5, so the (LN18, LN19) link is replaced with (LN9, LN5) and inserted into MPBQR. The link insertion process is the same as that in the index creation process. First, whether the link exceeds the division criterion with respect to the inserted cell is checked. In this example, no partitioning is performed because the partitioning criterion is not exceeded. Finally, an LN component and MP component are newly created for the cell in which the link is inserted.

### 3.3. MPBQR based Search

For the KNN search, the search count K is received as an input value. An example KNN search process is explained. First, Search_Count is set by comparing the number K of inputted searches and the total number of moving sensors stored in the current MPBQR. In this example, the number of searches is set to three and the total number of movement sensors stored is eight, so Search_Count is set to three. The link or link segment where the next query point q is located is searched using cell R-tree and MP R-tree. Figure 11 shows an example of the result of processing for a moving sensor.

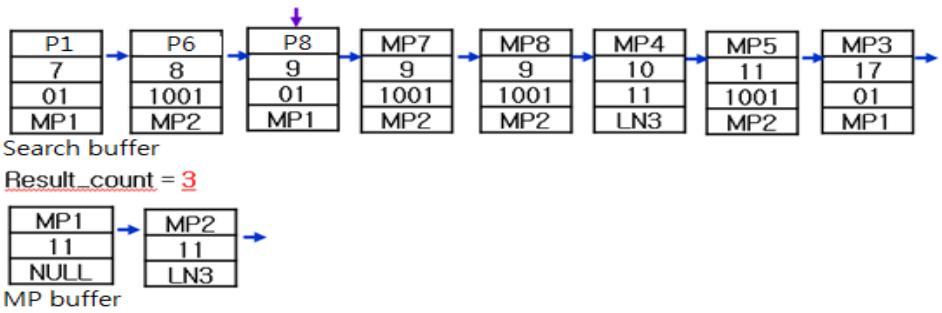



**Figure 11. Example of processing result for the moving sensor 8.**

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

As shown in Figure 11, after the process is completed for movement sensor 8, since the Result_Count is three, the search is terminated and the result is returned. The returned result returns the value from the first value of the search buffer to the cursor position and the network distance and path from the query point q to the movement sensor using the search buffer and the MP buffer.

The three results in this example are P7 = 7, P6 = 8, P8 = 9. The search buffer, the MP buffer, and the shortest path for each cell. Therefore, the final result is P1 = 7, path q→LN1→P1; P6 = 8, path q →LN9→P6; and P8 = 9, path q→LN1→P8. Figure 12 shows an example 3NN search result.

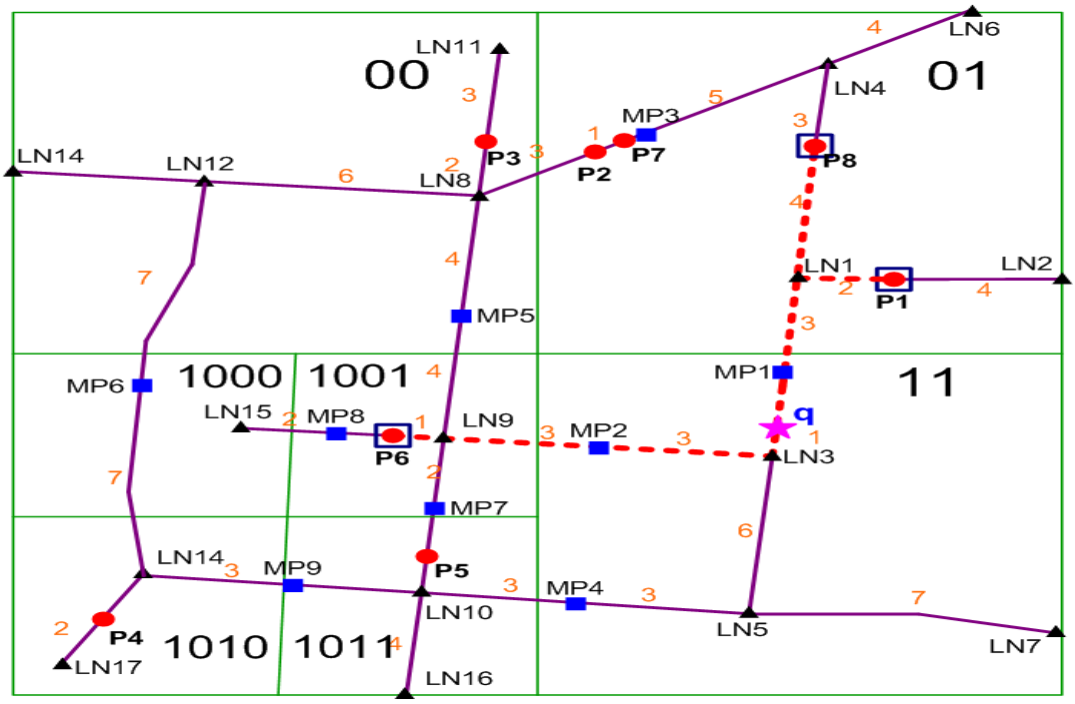



**Figure 12. 3NN search result example.**

The range search performs a search based on the MPBQR, and the search method is similar to the KNN search method.

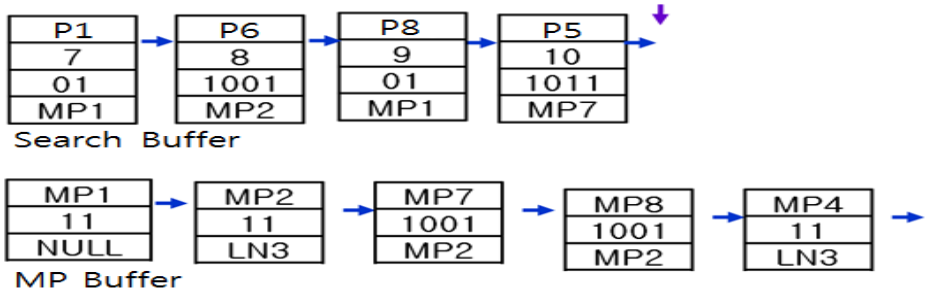



**Figure 13. Example of performing a search until a cursor points to null.**

As shown in Figure 13, if the cursor points to null, the search is terminated and the result is returned. The returned result returns the value from the first value of the search buffer to the previous position of the cursor, and returns the network distance and path from the query point q to the movement

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

sensor using the search buffer and deleted MP storage buffer. That is, the path is ultimately guided using the search buffer, the MP buffer, and the shortest path for each cell.

The result in this example is P1 = 7, path q→LN1 →P1; P6 = 8, path q→LN3→LN9→P6; P8 = 9, path q→LN1→P8; and P5 = 10, path q→LN3→ LN9→P5. Figure 14 shows an example 10 range result.

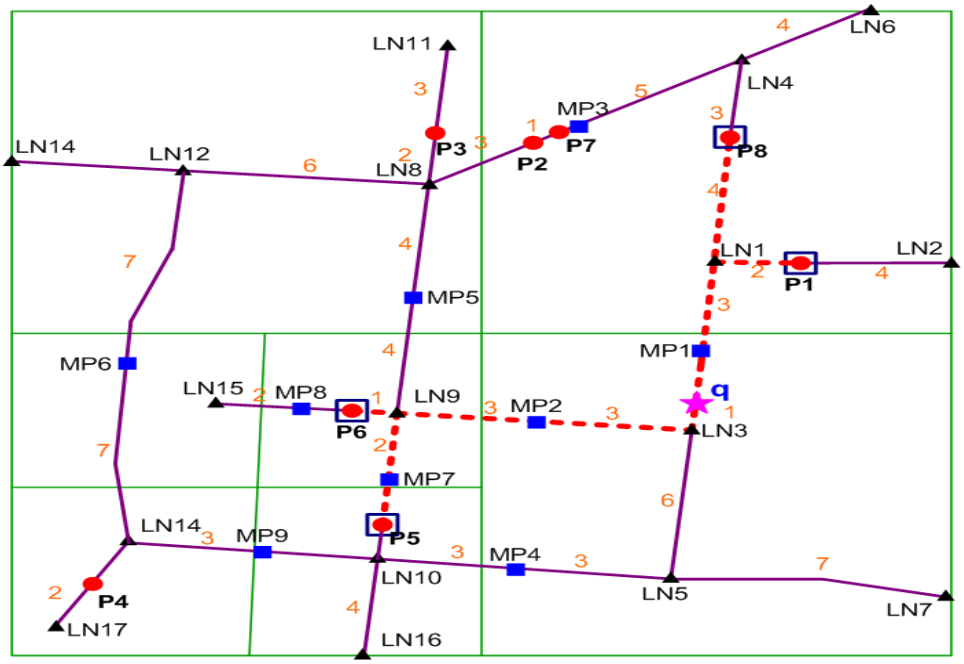



**Figure 14. A 10 range result example.**

## 4. Performance evaluation

In this paper, the system used for performance evaluation was a distributed structure that used five PCs. The Hadoop system consists of one main node and four slave nodes. INE and IER were implemented and compared to evaluate the performance of KNN search and range search using MPBQR.

In this paper, 30,000 arbitrary sensors were distributed over 100,000 line strings and 40,000 line segments, which were virtually implemented to justify the experimental results.

The performance of the moving sensor insertion time and the line string insertion time were evaluated in the insertion performance evaluation. The insertion rate of the moving sensor was set to a value obtained by dividing the number of inserted moving sensors by the amount of moving sensor data generated in Seoul and then multiplying this by 100% and setting the line string insertion ratio value to a value obtained by dividing the number of inserted line strings by the number of generated line strings and then multiplying this by 100%. Figure 15 shows the results of the insertion time performance evaluation according to the moving sensor's insertion ratio.

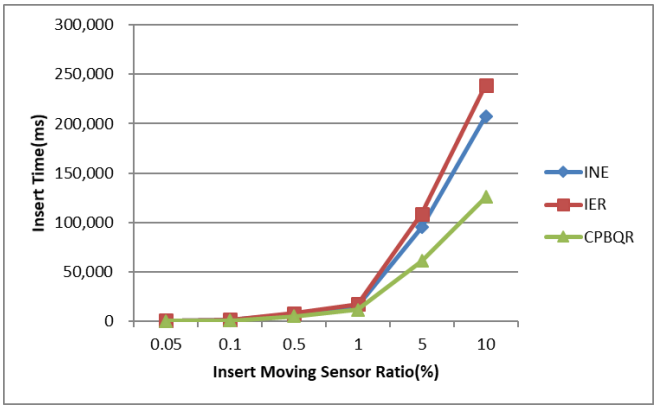



**Figure 15. Insertion time according to the moving sensor insertion ratio.**

March - April 2020
ISSN: 0193-4120 Page No. 4638 - 4650

In Figure 15, as a result of comparing the insertion time performance evaluation according to the moving sensor insertion ratio, the performance of MPBQR is improved by 65% on average and 90% for IER compared to INE.

Figure 16 shows the insertion time performance evaluation result according to the line string insertion ratio.

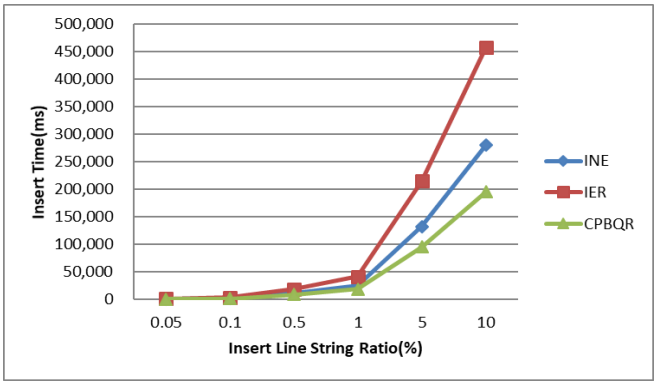



**Figure 16. Insertion time according to line string insertion ratio.**

In Figure 16, as a result of comparing the insertion time performance according to the line string insertion ratio, MPBQR was improved 43% on average and 23.3% when comparing IER to INE.

The performance of the KNN search time and range search time were evaluated in the search performance evaluation. In the performance evaluation of KNN search, the search time performance was evaluated according to the K value. Figure 17 shows the performance evaluation result for the KNN search time according to the K value.

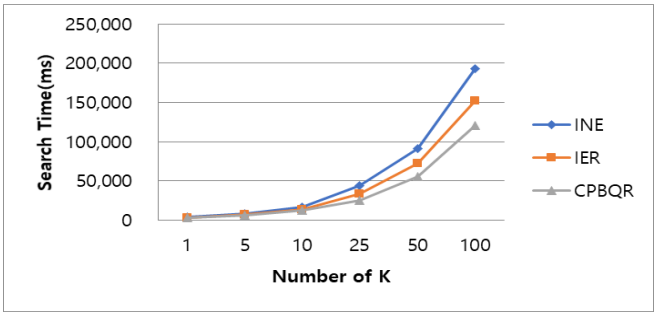



**Figure 17. KNN search time according to the K value.**

In Figure 17, as a result of comparing the KNN search time performance with the K value, MPBQR improved by 60% on average and 26% for IER compared to INE.

Figure 18 shows the performance evaluation result of the range search time according to the R value. The R value used in the range search was set to a value that was obtained by dividing the query range value by the length of the large side of the whole area and then multiplying this by 100%.

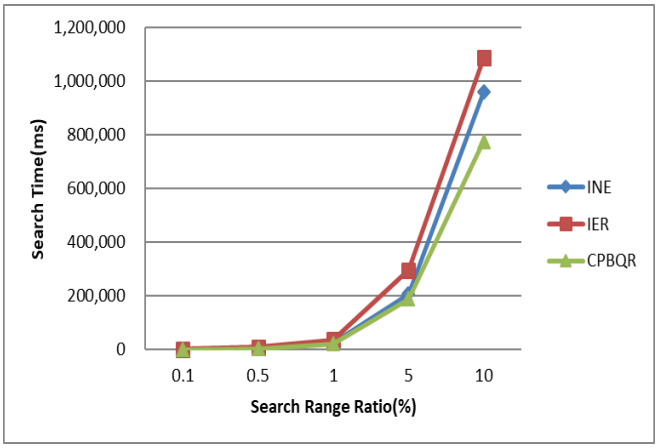



**Figure 18. Range search time according to the range search ratio.**

In Figure 18, in the results of comparing the range search time performance evaluation according to the search range value, the MPBQR was improved by 23% on average and 39% on average when compared to IER.

## 5. Conclusion

This paper proposed MPBQR to solve the problems in existing moving sensor search methods and to support more efficient processing of large capacity spatial data, and proposed KNN search and range search methods that use MPBQR.

MPBQR creates a quad-tree by dynamically dividing the cells based on the number of moving sensors and number of midpoints to reduce the storage space, stores the network distance between LN, MP, and the moving sensors in the cell in the LN and MP components. The search performance

is improved by managing the neighboring information of the cells using the midpoint so that the moving sensor stored in the adjacent cell can be quickly searched. The network distance is calculated from the LN and MP to the moving sensor in advance for the LN component and MP component, and these are stored in ascending order. The network distance calculation process from the query point to the MP and the movement sensor and search buffer storage process of the MP and movement sensor are simplified. Therefore, since MPBQR divides the network area into cells using a quad-tree and manages adjacent information between cells using the midpoint, the storage information is low and the storage efficiency and search performance are excellent.

## References

[1] Zhao G, Xuan K, Taniar D, Srinivasan B. Incremental K-Nearest-Neighbor Search on Road Networks. Journal of Interconnection Networks 2008;09(04):455-70.

[2] Liao W, Wu X, Yan C, Zhong Z. Processing of Continuous k Nearest Neighbor Queries in Road Networks. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 2009:31-42.

[3] Lin H. Efficient and compact indexing structure for processing of spatial queries in line-based databases. Data & Knowledge Engineering 2008;64(1):365-80.

[4] Vazirgiannis M, Wolfson O. A Spatiotemporal Model and Language for Moving Objects on Road Networks. Advances in Spatial and Temporal Databases 2001:20-35.

[5] Papadias D, Zhang J, Mamoulis N, Tao Y. Query Processing in Spatial Network Databases. Proceedings 2003 VLDB Conference 2003:802-13.

[6] Manolopoulos Y, Nardelli E, Papadopoulos A, Proietti G. QR-tree: A Hybrid Spatial Data Structure. Proceedings of the International Conference on Geographic Information Systems in Urban, Regional and Environmental Planning 1996:3-7.

[7] Wu S, Wu K. Effective Location Based Services with Dynamic Data Management in Mobile Environments. Wireless Networks 2006;12(3):369-81.

[8] Proietti G, Faloutsos C. I/O complexity for range queries on region data stored using an R-tree. Proceedings 15th International Conference on Data Engineering (Cat No99CB36337) 1999:628-35.

[9] Guttman A. R-Tree: A Dynamic Index Structure for Spatial Searching. Proceeding of the International Conference on Association for Computing Machinery Special Interest Group on Management of Data 1984;14(2):47-57.

[10] Demiryurek U, Banaei-Kashani F, Shahabi C. Efficient Continuous Nearest Neighbor Query in Spatial Networks Using Euclidean Restriction. Advances in Spatial and Temporal Databases 2009:25-43.