An Adjustable-Tree Method for Processing Reverse Nearest Neighbor Moving Queries

Ye-In Chang[1], Jun-Hong Shen[2,3(✉)], and Che-Min Chu[1]

[1] Department of Computer Science and Engineering,
National Sun Yat-Sen University, Kaohsiung City 804, Taiwan
changyi@cse.nsysu.edu.tw, roger8988@gmail.com
[2] Department of Information Communication,
Asia University, Taichung City 413, Taiwan
shenjh@asia.edu.tw
[3] Department of Medical Research, China Medical University Hospital,
China Medical University, Taichung City 404, Taiwan

Abstract. For a reverse nearest neighbor (RNN) query, the query object will find the data objects regard it as their nearest neighbor. Over the past few years, the RNN query on the road network database has attracted much attention. In the previous research, a multi-way tree efficiently solves the issue about moving data objects for the RNN query. However, in the scenario that the query object reaches a new location, i.e., the moving query, the multi-way tree needs to be reconstructed, which takes long time. Therefore, in this paper, we propose an adjustable-tree method for solving the above problem and improving the performance efficiency for processing moving queries. Via the performance evaluation, our proposed method performs better than the original multi-way tree method.

Keywords: Reverse nearest neighbor queries · Road network · Spatial database

1 Introduction

In recent years, the wireless communication technologies, mobile systems, and Global Positioning Systems (GPS) have been well developed [1]. Nowadays, mobile services are very popular so that the spatial database management for providing such services has attracted much attention. One of the fundamental research issues in the spatial database system is the reverse nearest neighbor (RNN) query. For an RNN query, the query object will find the data objects regard it as their nearest neighbor [4, 5, 9, 11]. For example, a convenience store owner wants to evaluate the possible location to open a new store. This can be regarded as an RNN query that evaluates how many buildings consider this possible location as their nearest neighbor.

Processing RNN queries over moving objects in road networks is a hot research topic over the past few years. An RNN query on moving objects is to search for the objects which take query point q as their nearest neighbor. The shortest network path distance $dist(Q, p)$, between query point Q and object p, is calculated based on the

J. C. Hung et al. (Eds.): FC 2019, LNEE 551, pp. 362–371, 2020.
https://doi.org/10.1007/978-981-15-3250-4_43

traffic situation in a road network [7]. The searching region of object p is defined as the circle centered at p with radius $dist(Q, p)$, and adjacent edges within it are verified to make sure that there is no other query points on these edges [7]. RNN queries can be classified into monochromatic and bichromatic queries [2, 10]. For a monochromatic RNN query, the query object and the data objects are of the same type. For a bichromatic RNN query, they are of two different types. In this paper, we investigate the bichromatic RNN queries.

In the literature, there are some research efforts to process RNN queries in road networks. In [6, 8], they used Voronoi diagram to solve RNN problems in road networks. However, the Voronoi diagram-based methods did not consider changing of the traffic situation so that they can be applied to only the RNN query with static data. In [2], the authors proposed a DLM tree, containing a range tree and the corresponding verifying trees, for processing the monochromatic reverse k nearest neighbor queries. In [7], the OpInitialBN-RNN method applied the multi-way tree to process bichromatic RNN queries on moving objects in road networks. However, when the query object moves to a new location in the road network, this method has to reconstruct the multi-way tree. As a result, it will take long time to find the results of the RNN query.

Therefore, in this paper, we propose an adjustable-tree method, which is also a multi-way tree, for processing RNN queries on the moving query in the road network. The proposed method rotates the original multi-way tree when the query object moves. After that, the proposed method shrinks/extends the multi-way tree from the leaf nodes according to the new location of the query object. The proposed method does not reconstruct the multi-way tree. In this way, the proposed method can improve the performance efficiency on finding the results of the RNN query. The contribution of this work is that the proposed method supports continuous RNN queries for moving query and data objects without the reconstruction of the tree structure.

The rest of this paper is organized as follows. Section 2 presents the proposed adjust-tree method for RNN queries of dynamic datasets in road networks. Section 3 compares the proposed method with the existing method. Section 4 evaluates the performance efficiency. Section 5 concludes this paper.

2 The Adjustable-Tree Method

In this section, we present our proposed adjustable-tree method for processing RNN moving queries. The proposed method contains two parts, the construction of the adjustable-tree and the corresponding incremental maintenance. In the construction of the adjustable-tree, the proposed method uses the PMR quadtree [3] to organize the road network and constructs the multi-way tree from the road network to answer the RNN query. This part of the proposed method is the same as that of the OpInitialBN-RNN method [7].

There are three moving cases in the maintenance of the multi-way tree. In Case 1, the query object is static, and the data objects move to a new location. In Case 2, the query object moves to a new location, and the data objects are static. In Case 3, both the query object and data objects move to new locations. The OpInitialBN-RNN method can deal with only the RNN queries in Case 1. By contrast, the proposed method can

deal with the RNN queries in all the cases. Instead of reconstructing the multi-way tree, the proposed method adjusts the multi-way tree when the query object and data objects move.

2.1 The Construction of the Adjustable-Tree

The proposed method uses a PMR quadtree [3] to store a road network, which is composed of nodes and edges. Edges are inserted into blocks of a PMR quadtree one by one. When the number of edges in a block exceeds the designated splitting threshold, the block will be partitioned into four equal-sized grid cells to store these edges. Consider a road network in Fig. 1 for example. The corresponding PMR quadtree is shown in Fig. 2, where the splitting threshold is set to 5.

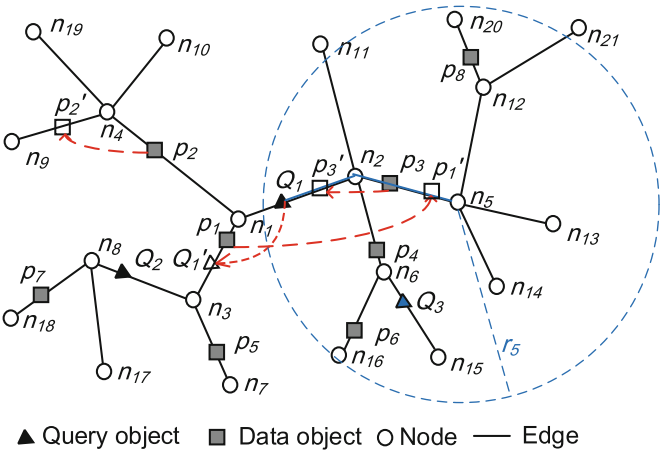

Fig. 1. A road network

The real index structure of the PMR quadtree in Fig. 2 is shown in Fig. 3. In this index structure, the block of the PMR quad-tree is linked to the edge list, and the edge list is linked to the node list and the object list. For example, edge $\overline{n_5 n_{12}}$ is in the block of the PMR quadtree. When we follow this link to the edge list from the block, we can know that there is no object on edge $\overline{n_5 n_{12}}$, and this edge is between node n_5 and node n_{12}.

The adjustable-tree, which is a multi-way tree, for answering an RNN query Q is constructed based on extending the road network stored in the PMR quadtree and verifying nodes. The extending of the adjustable-tree to find the next node is based on the Dijkstra strategy. Data objects on the corresponding edge, and the distance between the new extended node and it parent node are recorded during extending of the new node. Take RNN query Q_1 in Fig. 1 for example. Query object Q_1 is the root of its adjustable-tree shown in Fig. 4a. Based on the Dijkstra strategy, node n_1, the nearest node from Q_1 in Fig. 1, is inserted into the adjustable-tree, and is verified as the result of the RNN query. The verifying step for a node or data object is described later.

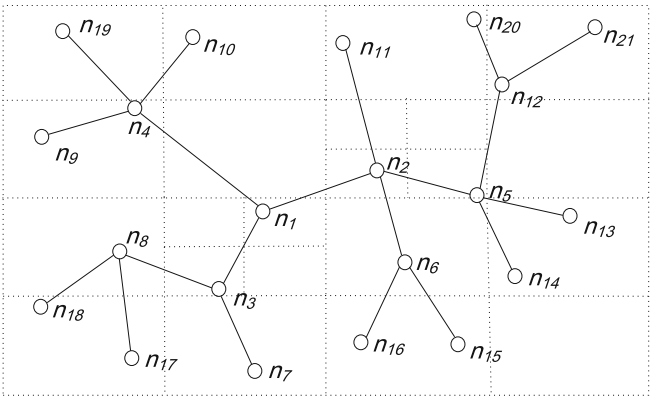

Fig. 2. The PMR quadtree

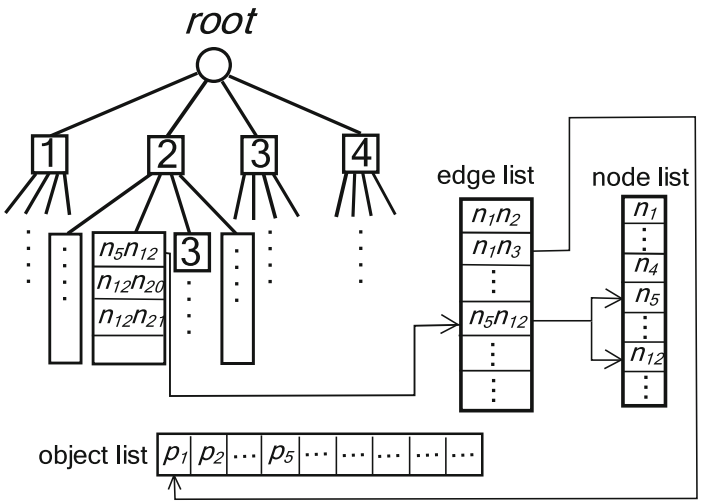

Fig. 3. The index structures for the PMR quadtree

During the extending of the road network, if the node is the result of the RNN query, it is inserted into the adjustable-tree. If the node is not the result of the RNN query, this node is marked as a bound node and inserted into the adjustable-tree. Since the bound node is not the result of the RNN query, it will not be further extended. After node n_1 is visited, node n_3 is extended and verified as a bound node so that it will not be further extended. The extending of the road network is iteratively spanned until no adjacent node can be extended. The node with no adjacent node to extend is called an unbound node. Figure 4a shows the adjustable-tree for answering RNN query Q_1. In Fig. 4a, nodes n_9, n_{10}, n_{11}, n_{13}, n_{14}, n_{19}, n_{20} and n_{21} are unbound nodes.

During the extending of the road network, the verifying step for an extended node n is to verify whether the node is the result of RNN query Q or not. In this step,

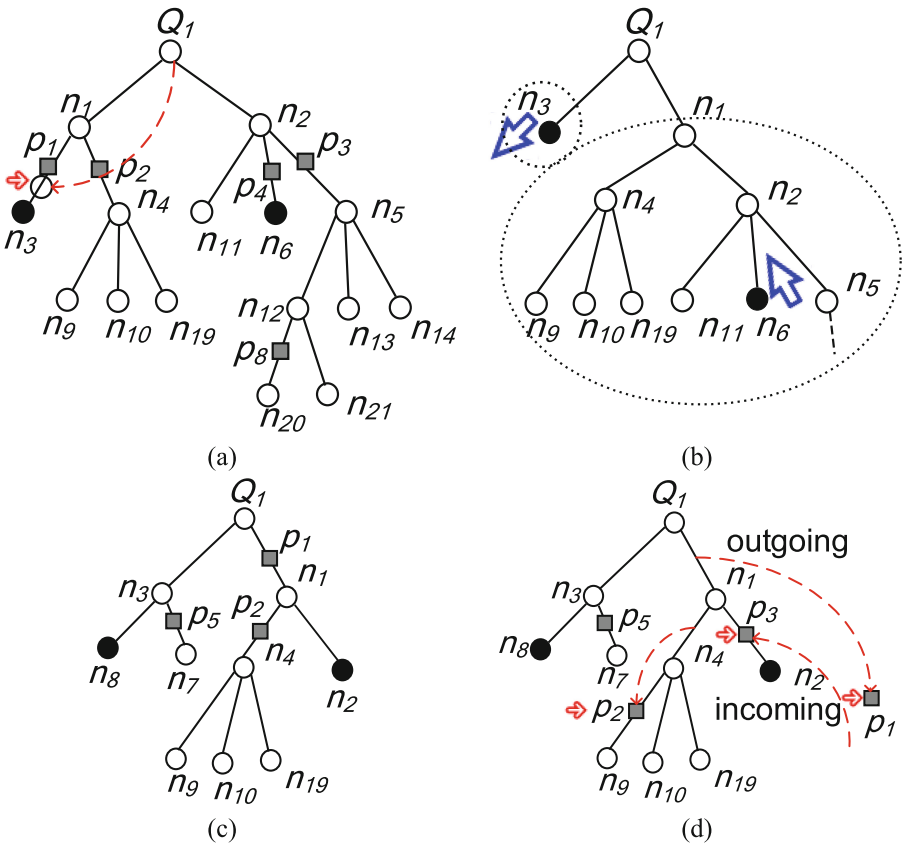

Fig. 4. The adjustable-tree: (a) the original multi-way tree; (b) the tree after rotating; (c) the final adjustable-tree; (d) the tree after the data objects moved.

we should examine whether it exists another query object on the adjacent edges within the search region for node n, which is the circle centered at node n with radius $dist(Q, n)$. Note that $dist(Q, n)$ is the shortest path distance between query object Q and node n. For example, in Fig. 1 node n_5 is extended and should be verified. The search region for node n_5 is the circle centered at node n_5 with radius r_5, which is equal to $dist(Q_1, n_2) + dist(n_2, n_5)$. In this search region, there is another query object Q_3, besides Q_1. Therefore, the shortest distance between Q_3 and node n_5, $dist(Q_3, n_5)$, should be compared with that between Q_1 and node n_5, $dist(Q_1, n_5)$. Since $dist(Q_3, n_5) > dist(Q_1, n_5)$, node n_5 is the result of RNN query Q_1. In addition, the verifying step could also be used to verify whether the data objects are the RNN result of the query.

In Fig. 4a, data objects p_2, p_3 and p_8 are the result of RNN query Q_1. Data objects p_1 and p_4 are on the edges whose end node is the bound node so that they should be further verified to examine whether they are the result of Q_1. In this case, p_1 is the result, but p_4 is not. As a result, the answer of RNN query Q_1 is $\{p_1, p_2, p_3, p_8\}$.

2.2 The Incremental Maintenance of the Adjustable-Tree

When the query object moves to a new location, the proposed method will use the query object at the new location as the new root by adjusting the original adjustable-tree with shrinking/extending the leaf nodes. The detail of the incremental maintenance of the adjustable-tree can be found in [1]. In Figs. 1, 4a, query object Q_1 moves to edge $\overline{n_1n_3}$ from edge $\overline{n_1n_2}$. The original adjustable-tree is rotated to make query Q_1 of the new location as the root node, dividing the tree into two parts, the subtree rooted by node n_3 and that rooted by node n_1, as shown in Fig. 4b. After that, the leaf nodes need to be updated. If the distance between the new root and leaf node x is longer than that between the old root node and leaf node x, node x should be evaluated to shrink; otherwise, node x should be evaluated to extend.

In Figs. 1, 4b, since the distances between the new root node and nodes n_9, n_{10} and n_{19} are longer than those between the old root node and them, they should be evaluated to shrink. After the verification, they are still the result of RNN query Q_1 so that they do not need to be shrunk. When node n_6 is verified, it is a bound node. Its ancestor nodes should be upward verified until a node y that is not the result of Q_1 and its parent node is the result of Q_1 is found. In this case, the adjustable-tree is shrunk, and a new bound node n_2 is found, as shown in Fig. 4c.

In Fig. 4b, since the distance between the new root node and node n_3 is shorter than that between the old root node and node n_3, it should be evaluated to extend. After the verification, node n_3 is the result of Q_1 and no longer the bound node. Therefore, it should be downward extended. In the process of extending, if the leaf node is a bound node, the node needs to be verified whether the node is still a bound node after the rotation of the tree. If the node is not a bound node, the adjustable-tree is further extended until the new bound node that is not the result of the RNN query or no adjacent node can be extended. If a leaf node is an unbound node, there is no extending from this node due to no adjacent node to be extended. In this example, the adjustable-tree is extended from node n_3 such that unbound node n_7 and bound node n_8 are found, as shown in Fig. 4c. After extending/shrinking the adjustable-tree, the final adjustable-tree of RNN query Q_1 is shown in Fig. 4c, where data objects p_1, p_2 and p_5 are on the corresponding edges. The result of Q_1 is $\{p_1, p_2, p_5\}$ after query object Q_1 moved to the new location.

To deal with the moving of a data object, there are three cases, incoming, outgoing, and moving within the tree. For the same example mentioned above, after query object Q_1 moved, data object p_3 moves to edge $\overline{n_1n_2}$ from edge $\overline{n_2n_5}$ shown in Fig. 1. It is an incoming object for the adjustable-tree, as shown in Fig. 4d. Next, data object p_1 moves to edge $\overline{n_2n_5}$ from edge $\overline{n_1n_3}$ shown in Fig. 1, and it is an outgoing object for the adjustable-tree shown in Fig. 4d. The outgoing object is not the result of the RNN query. Finally, data object p_2 moves to edge $\overline{n_4n_9}$ from edge $\overline{n_1n_4}$ shown in Fig. 1, which is the last case of the moving of a data object. The adjustable-tree of Q_1 after moving of the data objects is shown in Fig. 4d. If a data object moves to an edge whose end node is a bound node, the data object should be verified whether it is the result of the RNN query. Since $dist(Q_1, p_3) < dist(Q_3, p_3)$, data object p_3 is the result of Q_1. The final result of Q_1 after moving of the data objects is $\{p_2, p_3, p_5\}$.

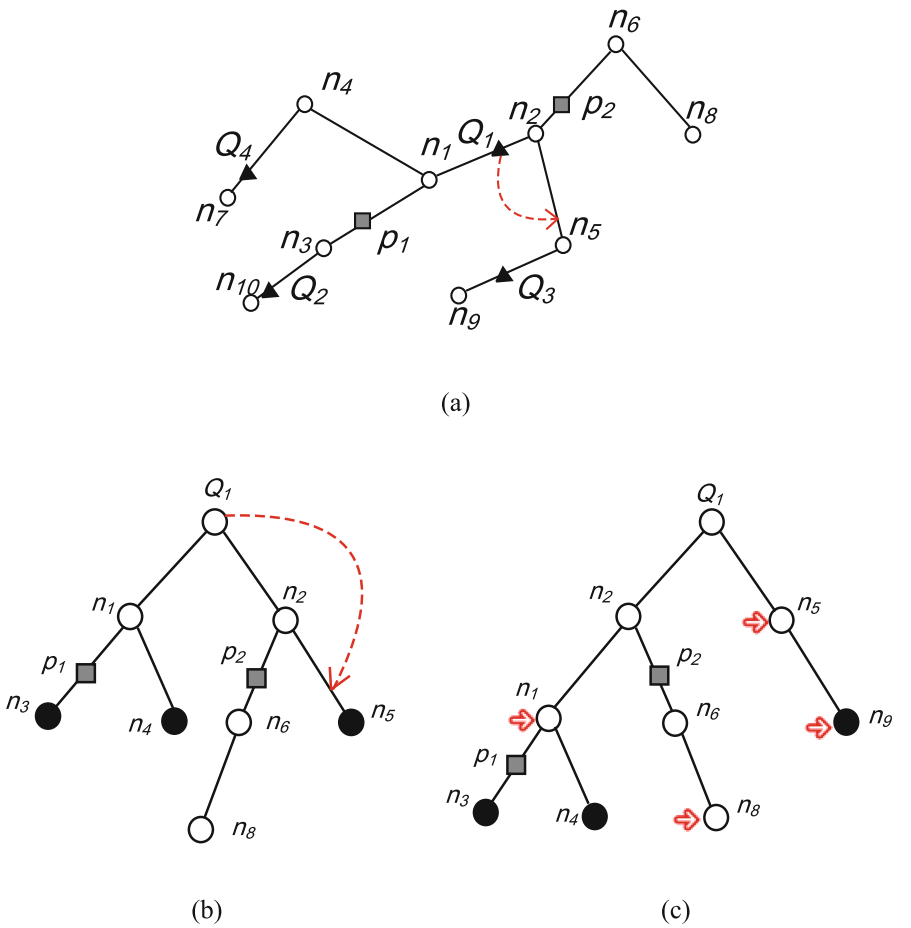

Fig. 5. A comparison: (a) query object Q_1 moves to a new location; (b) the initial multi-way tree; (c) the new multi-way tree after adjusting.

3 Comparison

In this section, we compare the proposed method with the OpInitialBN-RNN method [7] via an example shown in Fig. 5a. We use the OpInitialBN-RNN method to construct the corresponding multi-way tree shown in Fig. 5b. In Fig. 5a, a query object Q_1 on edge $\overline{n_1n_2}$ moves to edge $\overline{n_2n_5}$. In this case, the OpInitialBN-RNN method has to construct a new multi-way tree, whereas the proposed method adjusts the old multi-way tree to get the new multi-way tree. Both methods have the same new multi-way tree shown in Fig. 5c. In this example, the OpInitialBN-RNN method has to extend and verify all of eight nodes which are inserted into the multi-way tree in this road network. However, the proposed method needs to verify only four nodes n_1, n_5, n_8 and n_9 in the initial multi-way tree to extend or shrink the initial tree. Nodes n_3, n_4 and n_8 are farther

away from query Q_1, and we need to verify only node n_8 and node n_1, which is the parent node of n_3 and n_4. After moving of query Q_1, node n_5 is close to Q_1. We should then verify node n_5 and its adjacent node n_9. This comparison has shown that the proposed method performs better than the OpInitialBN-RNN method when the query object moves to a new location.

4 Performance Evaluation

In this section, we evaluate the performance efficiency on the average CPU time and the number of verified nodes of the proposed method with the OpInitialBN-RNN method [7]. The experiments were conducted in the real road network dataset, containing 18,263 nodes and 23,874 edges. Query objects and data objects were generated in the road network dataset at random.

In the first experiment, the number of data objects is set to 30,000, and the number of query objects is set to 40. The number of moving edges for a query object is varied from 2 to 10 with an interval of 2. Figure 6 shows that the processing (CPU) time of the proposed method is faster than that of the OpInitialBN-RNN method. The OpInitialBN-RNN method needs to verify all the nodes in the multi-way tree when a query object moves, but the proposed method does not. Because the OpInitialBN-RNN method will reconstruct the multi-way tree when the query object moves, the number of moving edges for a query object will not affect the processing time. However, the processing time of the proposed method will increase when the number of moving edges for a query object increases. This is because the number of nodes which need to be verified increases.

In the second experiment, we evaluate the corresponding number of verified nodes in both methods. The number of data objects and the number of query objects are set to the same values as those in the first experiment. The number of moving edges for a

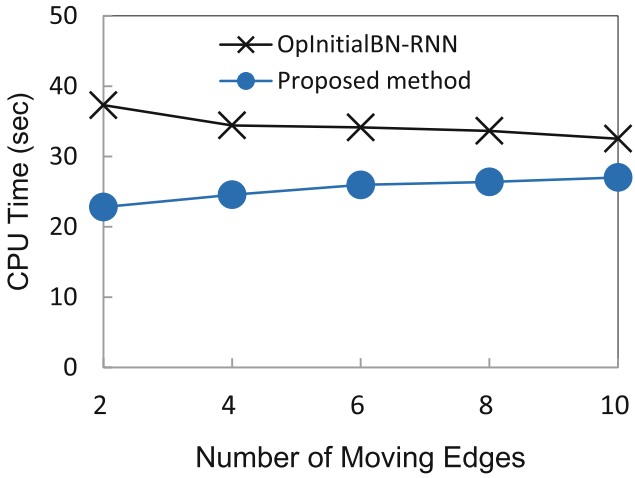

Fig. 6. A comparison of the processing time

query object is also varied from 2 to 10 with an interval of 2. Figure 7 shows that the number of verified nodes of the proposed method is less than that of the OpInitialBN-RNN method. The OpInitialBN-RNN method needs to verify all the nodes in multi-way tree when a query object moves, but the proposed method does not. The number of verified nodes will increase when the number of moving edges for a query object increases. The number of verified nodes is affected by the random distribution of the query objects in both the proposed method and the OpInitialBN-RNN method.

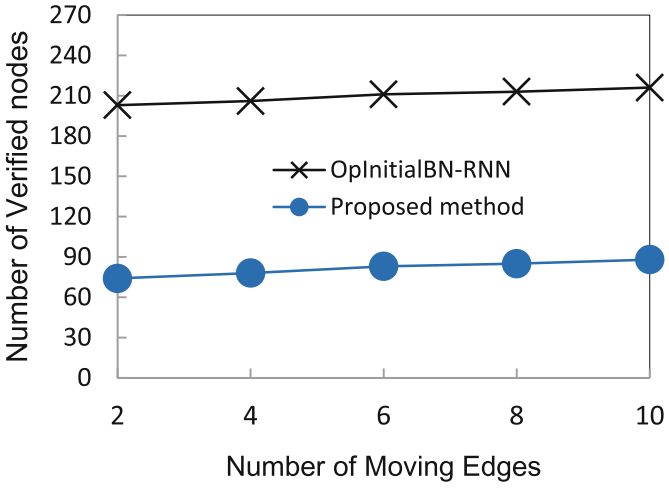

Fig. 7. A comparison of the number of verified nodes

5 Conclusion

In this paper, we propose an adjustable-tree method for improving the performance of reverse nearest neighbor (RNN) queries to deal with the situation in which the query object is moving in road networks. The proposed method can deal with the case that even though the query object and data objects are moving at the same time. From the performance evaluation, the proposed method performs better than the existing method.

Acknowledgments. This research was supported by grant MOST 104-2221-E-110-077 from the Ministry of Science and Technology, Taiwan.

References

1. Chiu, C.M.: An incremental approach to reverse nearest neighbor queries on the moving query in road networks. Master's thesis. National Sun Yat-Sen University, Taiwan (2014)

2. Guohui, L., Yanhong, L., Jianjun, L., Shu, L., Fumin, Y.: Continuous reverse k nearest neighbor monitoring on moving objects in road networks. Inf. Syst. **35**(8), 860–883 (2010)
3. Hoel, E.G., Samet, H.: Efficient processing of spatial queries in line segment databases. In: Proceedings of the Second International Symposium on Advances in Spatial Databases, pp. 237–256 (1991)
4. Kang, J.M., Mokbel, M.F., Shekhar, S., Xia, T., Zhang, D.: Incremental and general evaluation of reverse nearest neighbors. IEEE Trans. Knowl. Data Eng. **22**(7), 983–999 (2010)
5. Li, J., Li, Y., Shen, P., Xia, X., Zong, C., Xia, C.: Reverse k nearest neighbor queries in time-dependent road networks. In: Proceedings of IEEE 20th International Conference on High Performance Computing and Communications, pp. 1064–1069 (2018)
6. Safar, M., Ibrahimi, D., Taniar, D.: Voronoi-based reverse nearest neighbor query processing on spatial networks. Multimedia Syst. **15**(5), 295–308 (2009)
7. Sun, H.L., Jiang, C., Liu, J.L., Sun, L.: Continuous reverse nearest neighbor queries on moving objects in road networks. In: Proceedings of the 9th International Conference on Web-Age Information Management, pp. 238–245 (2008)
8. Taniar, D., Safar, M., Tran, Q.T., Rahayu, W., Park, J.H.: Spatial network RNN queries in GIS. Comput. J. **54**(4), 617–627 (2011)
9. Tao, Y., Yiu, M.L., Mamoulis, N.: Reverse nearest neighbor search in metric spaces. IEEE Trans. Knowl. Data Eng. **18**(9), 1239–1252 (2006)
10. Tran, Q.T., Taniar, D., Safar, M.: Bichromatic reverse nearest-neighbor search in mobile systems. Syst. J. **4**(2), 230–242 (2010)
11. Wu, W., Yang, F., Chan, C.Y., Tan, K.L.: Continuous reverse k-nearest-neighbor monitoring. In: Proceedings of the 9th International Conference on Mobile Data Management, pp. 132–139 (2008)