SR-KNN: An Real-time Approach of Processing *k*-NN Queries over Moving Objects

Ziqiang Yu, Yuehui Chen, Kun Ma*

Abstract Central to many location-based service applications is the task of processing *k*-nearest neighbor (*k*-NN) queries over moving objects. Many existing approaches adapt different index structures and design various search algorithms to deal with this problem. In these works, tree-based indexes and grid index are mainly utilized to maintain a large volume of moving objects and improve the performance of search algorithms. In fact, tree-based indexes and grid index have their own flaws for supporting processing *k*-NN queries over an ocean of moving objects. A tree-based index (such as R-tree) needs to constantly maintain the relationship between nodes with objects continuously moving, which usually causes a high maintenance cost. Grid index is although widely used to support *k*-NN queries over moving objects, but the approaches based on grid index almost require an uncertain number of iterative calculations, which makes the performance of these approaches be not predictable.

To address this problem, we present a dynamic Strip-Rectangle Index (SRI), which can reach a good balance of the maintenance cost and the performance of supporting *k*-NN queries over moving objects. SRI supplies two different index granularities that makes it better adapt to handle different data distributions than existing index structures. Based on SRI, we propose a search algorithm called SR-KNN that can rapidly calculate a final region space with a filter-and-refine strategy to enhance the efficiency of process *k*-NN queries, rather than iteratively enlarging the search space like the approaches based on grid index. Finally, we conduct experiments to fully evaluate the performance of our proposal.

Ziqiang Yu
The university of Ji'nan, Shandong province, China, 250022, e-mail: ise_yuzq@ujn.edu.cn

Yuehui Chen
The university of Ji'nan, Shandong province, China, 250022, e-mail: yhchen@ujn.edu.cn

Kui Ma
The university of Ji'nan, Shandong province, China, 250022, e-mail: ise_mak@ujn.edu.cn

* Corresponding author

F. Xhafa et al. (eds.), *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, Lecture Notes on Data Engineering and Communications Technologies 1, DOI 10.1007/978-3-319-49109-7_18

1 Introduction

Processing k nearest neighbor (k-NN) queries over moving objects is a fundamental operation in many location-based service applications. For example, a location-based social networking service may help a user find k other users that are closest to him/her. Some taxi-hailing applications such as UBER need monitoring the nearby taxies and users for a user or a taxi with submitting requests to applications respectively. In location-based advertising, a store may want to broadcast promotion messages only to the potential customers that are currently closest to the store. Such needs can be formulated as k-NN queries, where each user, taxi, or customer can be regarded as a moving object.

Consider a set of N_p moving objects in a two dimensional region of interest. An object o can be represented by a quadruple $\{id_o, t, (o_x, o_y), (o'_x, o'_y)\}$, where id_o is the identifier of the object, and t is the current time; (o_x, o_y) and (o'_x, o'_y) represent the current and previous positions of o respectively. The old location (o'_x, o'_y) can help us remove the obsolete positions of moving objects. In this study, we adopt the snapshot semantics because we make no assumption on the motion of objects, which conforms to the situation of moving objects in reality. The snapshot semantics refer to the answer of query q at time t is only valid for the past snapshot of the objects and q at time $t - \Delta t$, where Δt is the delay due to query processing. Since this study aims to processing k-NN queries over moving objects in real-time, we thus need to reduce the value of Δt as small as possible. In order to achieve this purpose, we focus on main-memory-based solutions.

The problem of k-NN query processing over moving objects has attracted considerable attentions in recent years. The existing works about this problem can be broadly classified into tree-based approaches and grid-based approaches. Tree-based approaches refer to the works [1] that adopt tree index structures (such as R-tree, B^+-tree) to process the k-NN queries on spatial-temporal data. Here, the R-tree [2] is a data structure used for indexing multi-dimensional information such as geographical coordinates, rectangles or polygons and it has been adopted to answer k-NN queries in some works. The general idea of these works is first searching the nearest neighbor for a given query, and then determining k nearest neighbors of this query. R-tree, as a spatial data index, well adapts to index the static spatial data, but it is not suitable for the maintenance of continuous moving objects. This is because the nodes of R-tree has to be split or merged frequently with objects constantly moving, and even the R-tree requires to be organized repeatedly. Therefore, indexing a large scale of moving objects with the R-tree will cause a large maintenance cost .

Grid index is a typical spatial index that partitions the whole search region (2-D surface in this study) of interest into equal-sized cells, and indexes objects and/or queries (in the case of continuous query answering) in each cell respectively. Extensive existing works propose various search algorithms based on the grid index to handle k-NN queries over moving objects. Most existing grid-based approaches to k-NN search [3, 4, 5] iteratively enlarge the search region to identify the k-NNs. For example, given a new query q, the YPK-CNN algorithm [3] initially locates a rectangle R_0 centered at the cell covering q; it then iteratively enlarges R_0 until it

encloses at least k objects. Let p' be the farthest object to query q in R_0. The circle C_r centered at q with radius $\| q - p' \|$ is guaranteed to contain the k-NNs of q, where $\| \cdot \|$ is the Euclidean norm. The algorithms then computes the k-NNs using the objects in the cells intersected with C_r. The other existing grid-based approaches are based on similar ideas. These approaches have a common defect that they need to iteratively search k objects and the number of iterations is unpredictable. In some cases, these approaches probably cause extensive search iterations especially on the data in non-uniformly distribution, which will degrade their performance.

To address this challenge, we propose a dynamic Strip-Rectangle index called SRI to support the processing of k-NN queries over moving objects. SRI is a two level index structure . The first level index of SRI is the strips that partition the whole region of interest. All strips can cover the whole search region and no overlap exists between any two different ones. Further, each strip is divided into smaller rectangles that form the second level index. SRI can dynamically adjust the sizes of strips and rectangles based on the distribution of moving objects to make each strip and rectangle guarantee covering at least ξ objects. This characteristic makes SRI better adapt to support the processing of k-NN queries and handle various distributions of spatial data. Based on SRI, we design an algorithm called SR-KNN to handle k-NN queries over moving objects without iterations occurred in grid-based approaches. For a given query q, SR-KNN adopts a filter-and-refine strategy to rapidly calculate a small search region that covers k neighbors of q, and then obtain k-NNs from this search region. Our contributions can be summarized as follows.

- We propose SRI, a strip and rectangle combined index structure that can well support the processing of k-NN queries over a large scale of moving objects in different distributions.
- Based on SRI, we design the SR-KNN algorithm that can improve the efficiency of processing spatial k-NN queries by avoiding the unpredictable iterative calculations, which solves the major flaw of existing grid-based algorithms.
- Extensive experiments are conducted to sufficiently evaluate the performance of our proposal.

2 Related work

The problem of k-NN query processing over moving objects has attracted considerable attentions in recent years. In this section, we present a brief overview of the literature.

The R-tree has been adopted extensively (e.g., [1, 6, 7, 8]) to answer nearest neighbor queries. Ni et al. [9], Roussopoulos et al. [10], and Chaudhuri et al. [11] use the TPR-tree to index moving objects and propose filter-and-refine algorithms to find the k-NNs. Gedik et al. [12] describe a motion-adaptive indexing scheme based on the R-tree index to decrease the cost of update in processing k-NN queries. Yu et al. [13] first partition the spatial data and define a reference point in each partition,

and then index the distance of each object to the reference point employing the B^+-tree structure to support k-NN queries.

Grid index is widely used to process spatial queries [3, 14, 15, 16, 17, 18, 19]. Zheng et al. propose a grid-partition index for NN search in a wireless broadcast environment [14]. The Broadcast Grid Index (BGI) method proposed by [15] is suitable for both snapshot and continuous queries in a wireless broadcast environment. Šidlauskas et al. [18] propose PGrid, a main-memory index consisting of a grid index and a hash table to concurrently deal with updates and range queries. Wang et al. [19] present a dual-index, which utilizes an on-disk R-tree to store the network connectivities and an in-memory grid structure to maintain moving object position updates.

3 The SRI structure

In building SRI, the region of interest R in an Euclidean space (normalized to the $[0, 1)$ square) is first partitioned into non-overlapping strips. In this study, we make the partition be done along the x axe, thus the whole region is divided into multiple vertical strips. For each strip, SRI further divides it into smaller rectangles without overlap and this partition is done along the y axe. An example of SRI structure is shown in Fig. 1.

A strip S_i ($1 \leq i \leq N_v$, where N_v is the number of strips) in SRI takes the form of $\{id_i, lb_i, ub_i, p_i, \Lambda_i\}$, where id_i is the unique identifier of S_i, lb_i and ub_i are the lower and upper boundaries of the strip respectively, p_i is the number of moving objects in this strip, and Λ_i is a list of identifiers of rectangles contained in this strip. Similarly, a rectangle R_j covered by S_i can be represented as $\{rid_j, b_j, u_j, \Gamma_j\}$, where rid_j is the identifier of the rectangle, b_i and u_i are the lower and upper boundaries of R_j respectively, and Γ_j is a list of objects in R_j. Since the strips and rectangles are both non-overlapping and every object must fall in one strip and one rectangle, we can deduce $\sum_{i=1}^{n} p_i = Np$ and no object belongs to two different strips, where N_p is the total number of moving objects. For any strip S_k with m rectangles, we can infer $\sum_{j=1}^{m} |\Gamma_j| = p_k$, and $\nexists o_t \in \Gamma_s \cap \Gamma_t$ ($s \neq t$). Fig. 1 describes attributes of index elements in SRI.

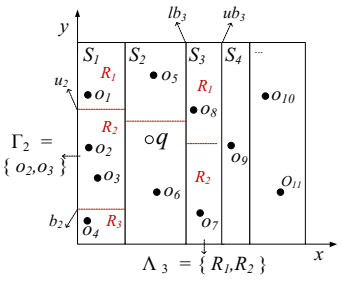

Fig. 1 The structure of SRI

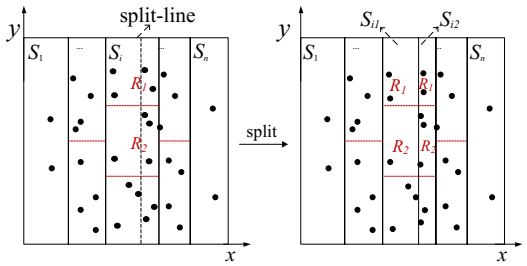

Fig. 2 The split of S_i

In SRI, we require every strip to contain at least ξ and at most θ objects, i.e., $\xi \leq p_i \leq \theta$ for all strips S_i. The strips are split or merged as needed to ensure this condition is met when object locations are updated. We call ξ and θ the *minimum occupancy* and *maximum capacity* of a strip respectively. Typically $\xi << \theta$. Be similar with the strip, every rectangle R_j is also required to contain at least ξ and at most β objects ($\xi < \Gamma_j < \beta$), where ξ and β are the *minimum occupancy* and *maximum capacity* of a single rectangle. Each rectangle also needs to be spit or merged with its adjacent rectangle within the same strip to ensure the volume of its moving objects belongs to $[\xi, \beta]$. In this study, a strip will be divided into multiple rectangles to form the index with a smaller granularity, thus the value of β is specified smaller than that of θ.

So far, the structure of SRI is more clear. Strips form the first level index and they are sorted in ascending order according to their boundaries. The rectangles of each strip construct the second level index and they are also maintained in order based on their boundaries within every strip. SRI has two level indexes, but it only needs to store one piece of all moving objects. This is because every strip does not store the locations of its moving objects but just record their quantity, which not only reduces the memory and maintenance costs but also is critical for designing k-NN algorithms. The benefits of SRI will be discussed later.

3.1 Insertion

When an object o_i sends the message $\{id_o, t, (o_x, o_y), (o'_x, o'_y)\}$ to the server, we need to insert the new position (o_x, o_y) of o_i and delete its old location (o'_x, o'_y).

Object o_i is inserted into SRI with two steps: (1) determining the strip S_i that o_i falls into its boundaries ($lb_i \leq o_x < ub_i$) and modifying the value of p_i ; (2) searching the rectangle R_j that satisfies $b_j \leq o_y < u_j$ from strip S_i and then inserting o_i into the rectangle R_j. The insertion is done by appending its id_o into the object list Γ_j. Initially, there is only one strip covering the whole region of interest and the strip itself is a rectangle.

When an object o_i is inserted into a rectangle R_j in the strip S_i, which probably causes two types of splits: rectangle spit and strip split. After o_i being inserted into rectangle R_j, it will be split if the volume of objects in it exceeds the maximum capacity, i.e., $|\Gamma_j| > \beta$. A split method that can adapt to the data distribution is to split R_j and generate two new ones that hold approximately the same number of objects. In this method, we first find an object o such that o_y is the median of the y coordinates of all objects in rectangle R_j, which implies that there are approximately $|\Gamma_i|/2$ objects whose y coordinates are less than or equal to o_y. This can be accomplished in $O(|\Gamma_i|)$ time using the binning algorithm. Next, we set the line $y = o_y$ as the split-line, according to which R_j can be split into two new strips R_{i1} and R_{i2}. Once R_j is split, the attributes of new strips can be determined as follows. The lower boundary of R_{i1} is the same as that of S_i, and its upper boundary is the split-line. For R_{i2}, it uses the split-line as the lower boundary, and the upper boundary of R_i as its upper boundary. The *id* of R_i is inherited by S_{i1}, and a new *id* is assigned to R_{i2}.

As to strip S_i, it will also be split if the number of moving objects covered by itself is greater than θ as an object o being inserted. The split method about strip is similar with that about the rectangle and Fig. 2 gives an example of strip split. When S_i is split into S_{i1} and S_{i2}, the attributes of new strips also can be easily deduced just like determining the attributes of new rectangles above. Meanwhile, every rectangle R_k in S_i also need to be split into R_{k1} and R_{k2} by the split-line. Then the objects in R_k will be assigned into R_{k1} and R_{k2} based on the split-line. That is, $\Gamma_{k1} \cup \Gamma_{k2} = \Gamma_k$ and $\Gamma_{k1} \cap \Gamma_{k2} = \emptyset$. Since R_{k1} and R_{k2} belong to two different strips, so both of them can use the identifier and the lower and upper boundaries of R_k.

3.2 Deletion

If object o disappears or moves out of a rectangle, it has to be deleted from the rectangle that currently holds it. To delete an object o, we need to determine which rectangle current holds it, which can be done using its previous position (o'_x, o'_y). After deleting an object, if the rectangle R_j in strip S_i has less than ξ objects (i.e., R_j has an underflow), it will be merged with an adjacent rectangle in S_i. Let this adjacent strip be R_h. R_j will be deleted from Strip S_i, and the merged strip will inherit the *id* of the R_h, and its lower and upper boundaries are set to be the lower and upper boundaries of R_j and R_h, respectively. The object lists Γ_j and Γ_h are merged.

When object o moves from strip S_i to another strip or disappears, the number of moving objects in strip S_i needs to minus 1, i.e., $p_i = p_i - 1$. At this time, if p_i is smaller than the *minimum occupancy* ξ, then S_i also needs to be merged with the adjacent strip S_j with fewer moving objects. Since $p_i < \xi$, S_i contains only one rectangle. In this case, we assign the objects in S_i into the corresponding rectangles in strip S_j, and then update the boundaries of S_j to make it cover the space of S_i. The boundaries of every rectangle in the new strip S_j remain the same.

4 The SR-KNN algorithm

The SR-KNN algorithm follows a filter-and-refine paradigm. For a given k-NN query q, the algorithm first prunes the search space by identifying *candidate strips* that are guaranteed to contain at least k neighbors of q. From candidate strips, it then identifies *candidate rectangles* that also covers at least k neighbors of q, which can further narrow down the search region. Next, it examines the objects contained in these candidate rectangles and identify the k-th nearest neighbor found so far. Using the position of this neighbor as a reference point, it calculates the final region that guarantees covering k-NNs of q and obtain the final result from it. We present the pseudocode of SR-KNN in Algorithm **??**. Now we present its details.

4.1 Calculating candidate strips

For a given query q, SR-KNN can directly identify the set of strips that are guaranteed to contain k neighbors of q, which we call the candidate strips.

Step 1: Calculating the number of candidate strips. Assume that the number of candidate strips is c. The idea is that from each strip we select χ ($1 \leq \chi \leq \xi$) objects that have the shortest Euclidean distances to q, such that $\chi * c \geq k$, where χ can be specified by users. This way, we have found at least k neighbors for q. Of course, these objects may not be the final k-NNs, but they can help us prune the search space and serve as the starting points for computing the final k-NNs. Hence, the number of candidate strips c is set to be $\lceil k/\chi \rceil$.

Step 2: Identifying the set of candidate strips. In this step, we identify the strips that are considered to be "closest" to q based on their boundaries. We use d_i^l and d_i^u to denote the distances from q to the lower and upper boundaries of S_i respectively. For the example shown in Fig. 3, the line l_i is perpendicular to lb_2 of S_2 and the distance from q to lb_2 is d_2^l. The distance between S_i and q is defined to be $dist(S_i, q) = \max\{d_i^l, d_i^u\}$.

If query q is located in S_i, then S_i is automatically a candidate strip and inserted into $\mathscr{C}^V$. Next, we decide whether its neighboring strips are candidate strips. Starting from the immediately adjacent strips, we expand the scope of search, adding to $\mathscr{C}^V$ the strip j that has the next least $dist(S_j, q)$. This procedure terminates when $|\mathscr{C}^V| = c$ or all strips have been processed. Fig. 3 gives an example, in which S_3 is determined to be a candidate strip first. Then by comparing d_2^l with d_4^u, we decide S_4 to be the next candidate strip. Next, we find S_2 also a candidate strip.

4.2 Calculating candidate rectangles

For a query q (x_q, y_q), candidate rectangles refer to the rectangles that are "closest" to q and at least cover k neighbors of q. We set the candidate rectangles to q must be covered by its candidate strips.

Step 1: Determining the number of candidate rectangles. In this step, we also suppose χ ($1 < \chi < \xi$) objects are chosen from each rectangle, then the number of candidate rectangles is $\lceil k/\chi \rceil$.

Step 2: Identifying the set of candidate rectangles. In SRI, there exists a center in every rectangle that has equal distance to the lower and upper boundaries. We adopt the distance between the center of a rectangle to q as the metric to identify the rectangles that are "closest" to q. Since the centers of all rectangles in a strip have the same x coordinate, for any two rectangles in the same strip, we can immediately infer which rectangle is closer to q only based on their y coordinates. Hence, we can rapidly identify the closest rectangle to q within every strip without computing distances between their centers and q, which will reduce extensive calculation costs.

To identify candidate rectangles, we use two sets $\mathscr{R}_c$ and $\mathscr{R}_t$ to separately store candidate rectangles and intermediate results. First, we find the closest rectangle to q in each candidate strip and put these rectangles into $\mathscr{R}_t$. Next, we choose the

closest rectangle R_f to q from $\mathscr{R}_t$ and add R_f into $\mathscr{R}_c$. Third, we put a rectangle R_s into $\mathscr{R}_t$, where R_s and R_f belong to the same strip S_i and R_s is closer to q than other rectangles in S_i except for R_f. We execute the second and third steps repeatedly until $\mathscr{R}_c$ contains $\lceil k/\chi \rceil$ rectangles or $\mathscr{R}_t$ is empty. Finally, $\lceil k/\chi \rceil$ candidate rectangles can be contained by $\mathscr{R}_c$. Fig. 3 shows an instance of identifying candidate rectangles and blue points are centers of rectangles.

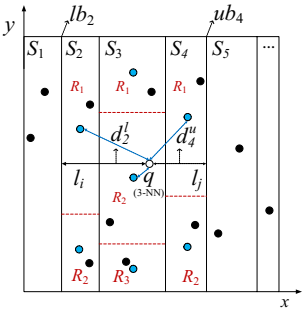

Fig. 3 Identifying candidate strips and rectangles

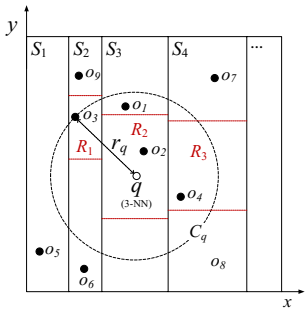

Fig. 4 Procedure of processing query q

4.3 Determining the final search region

After candidate rectangles being determined, we form the set of *supporting objects* Υ by selecting from each candidate rectangle χ objects that are closest to q. We then identify the supporting object $o \in \Upsilon$ that is the k-th closest to q. Let the distance between o and q be r_q. The circle with (q_x, q_y) as the center and r_q as the radius is thus guaranteed to cover k-NNs of q. Next, we identify the set of rectangles $\mathscr{F}$ that intersect with this circle, and search for the final k-NNs within the objects in $\mathscr{F}$.

Fig. 4 shows an example, where the query q is a 3-NN query and $\chi = 1$. We first identify the candidate strips $\{S_2, S_3, S_4\}$ as well as the candidate rectangles $\{R_1, R_2, R_3\}$, and then find three closest supporting objects (o_3, o_2, o_4) from candidate rectangles. Next, we set the radius r_q to be the distance between q and o_3 and the circle C_q is guaranteed to contain the 3-NNs of q. After scanning all objects that are located within C_q (by examining all rectangles intersecting C_q), we find that the 3-NNs are o_1, o_2, and o_4.

4.3.1 Advantages of SR-KNN

- **Powerful pruning strategy**: According to SRI index, SR-KNN can quickly narrow down the search space that covers the final results of queries by two pruning steps. It first identifies candidate strips to locate a much smaller search region that covers k neighbors of the query and further prunes the search region by calculating candidate rectangles, which significantly enhance the search performance.

- **Low costs of calculating candidate rectangles**: In one strip, SR-KNN can rapidly infer which rectangle is closest to q without computing the distances from the center of each rectangle to q, which effectively reduces expenses of calculating candidate rectangles and improves the whole efficiency of SR-KNN algorithm.
- **Avoiding multiple iterations**: Be different with grid-based algorithms, SR-KNN utilizes a cascading pruning strategy to narrow down the search space instead of iteratively enlarging the search space. Regardless of uniform or non-uniform data distribution, SR-KNN always can rapidly locate a small final search region with only three steps, which makes it be superior to grid-based algorithms.

5 Experiments

The experiments are conducted on a server with a 2.4GHz Intel processor and 8GB of RAM. We use the German road network to simulate three different datasets for our experiments. In these datasets, all objects appear on the roads only. In the first dataset (UD), the objects follow a uniform distribution. In the second dataset (GD1), 70% of the objects follow the Gaussian distribution, and the other objects are uniformly distributed. The third dataset (GD2) also has 70% of the objects following the Gaussian distribution, but they are more concentrated. In all four datasets, the whole area is normalized to a unit square, and the objects move along the road network, with the velocity uniformly distributed in [0, 0.002] unless otherwise specified.

5.1 Performance of index construction and maintenance

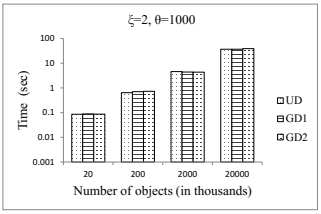

Fig. 5 Computation time for building SRI

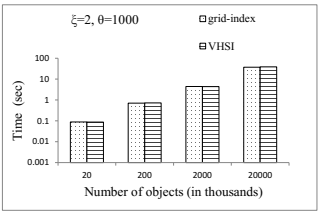

Fig. 6 Comparison of SRI and grid-index w.r.t building time

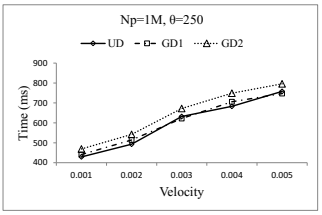

Fig. 7 Maintenance cost w.r.t velocity

Time of building SRI. We first test the time of building SRI from scratch to index different numbers of objects. Fig. 5 shows the time of building SRI as we vary the number of objects. In our study, the size of each object is approximately 50B, so we at most handle 1GB of data in this experiment. The time it takes to build the index increases almost linearly with the increasing number of objects.

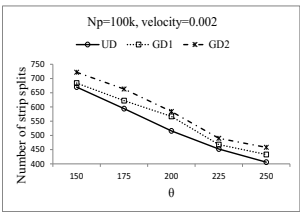

Fig. 8 Number of strip splits w.r.t. θ

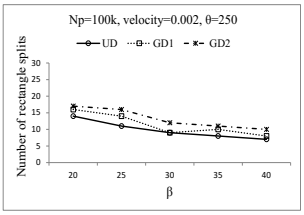

Fig. 9 Number of rectangle splits w.r.t. θ

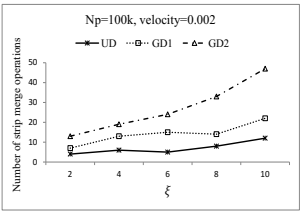

Fig. 10 Number of strip merge operations w.r.t. ξ

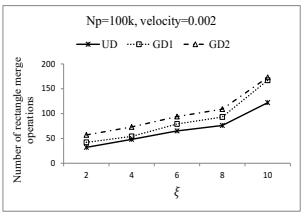

Fig. 11 Number of rectangle merge operations w.r.t. ξ

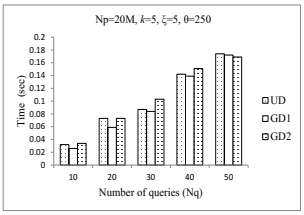

Fig. 12 Performance of RS-KNN w.r.t. number of queries

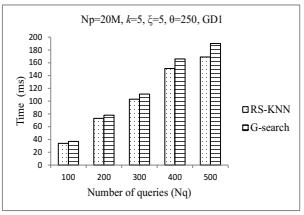

Fig. 13 Comparison of RS-KNN and G-search based on GD1

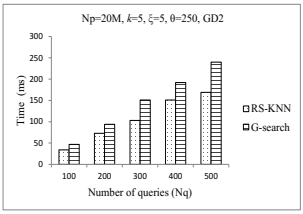

Fig. 14 Comparison of RS-KNN and G-search based on GD2

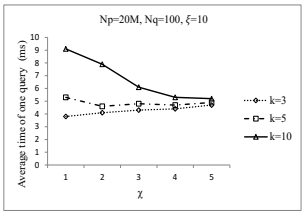

Fig. 15 Performance of RS-KNN w.r.t. ξ

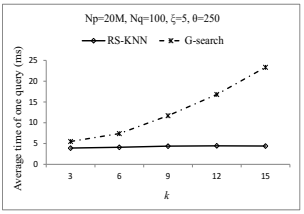

Fig. 16 Query evaluation time w.r.t. k

Comparison of building times. We then t test the time of building SRI and grid-index to index different numbers of objects based on the GD2 dataset. Fig. 6 demonstrates that SRI and grid-index need almost equal building time to handle the moving objects, which certifies the good performance of SRI with respect to indexing objects.

Effect of the velocity of objects. Fig. 7 demonstrates the effect of the velocity of objects on the computation time for maintaining SRI based on three datasets. In this set of experiments, we first build the SRI for 1M objects, and then 100K object are chosen to move continuously with varying velocities. As expected, the faster the objects move, the more split and merge operations happen, leading to an increase in maintenance time.

Effect of θ on SRI. Fig. 8 shows the effect of the maximum capacity of strip, θ, on the frequency of strip split. The number of moving objects indexed is 100K. As can be observed from Fig. 8, the strip split frequency is approximately reversely

proportional to the value of θ; a greater θ value would result in a reduction in the number of splits. Of course, θ cannot be overly large, because that will increase the time for processing queries.

Effect of β on SRI. Fig. 9 demonstrates the effect of the maximum capacity of rectangle on the frequency of rectangle split. This group of experiments also index 100k moving objects and test the average number of rectangle splits in every strip. The results show that the rectangle split frequency is also approximately reversely proportional to the value of θ; a greater β value would reduce the number of splits. Similarly, the large value of β will increase the time of processing queries, so β cannot be set overly large.

Effect of ξ on RSI. Fig. 10 and Fig. 11 show the influence of the minimum occupancy, ξ, on the frequency of strip and rectangle merge operations. A larger value of ξ means that underflow will occur more often and thus cause more strip and rectangle merge operations. Additionally, the number of rectangle merge operations is greater than that of strip merge operations.

5.2 Performance of query processing

We now perform experiments to evaluate the performance of SR-KNN, and make a comparison of SR-KNN and G-search algorithms.

Processing time. We feed a batch of queries into our system in one shot, and measure the time between the first query entering the system and the *k*-NN results of all queries having been obtained. As can be observed from Fig. 12, SR-KNN achieves similar performance for different distributions. This is because every strip and rectangle in SRI separately contains at most θ and β objects and typically each query only involves a few strips and rectangles. Therefore, the data distribution only has a slight impact on the query processing time.

In Fig. 13 and Fig. 14 , we make SR-KNN and G-search algorithms process the same batch of queries and measure their processing times with varying number of queries based on the GD1 and GD2 datasets. The results show that SR-KNN outperforms G-search on both datasets but more significantly on GD2, the reason is that G-search needs more iterative calculations to process *k*-NN queries on the non-uniformly dataset.

Effect of χ on SR-KNN. In SR-KNN, we select χ objects from each candidate strip to form the set of supporting objects. Fig. 15 shows the influence of χ on the cost of processing queries. In this set of experiments, we feed 100 queries into the system and record the average processing time of a query. The results show that χ has little effect on the processing time when *k* takes smaller values (3 and 5). But *k* increases, the influence becomes more obvious.

Effect of *k*. Finally, we study the influence of *k* on the two algorithms. As shown in Fig. 16, the processing time of SR-KNN almost remains unchanged as *k* increases, the reason being that we can adjust the value of χ accordingly to accommodate the increase in processing time. When *k* increases, G-search needs more iterations to compute the results, thus its processing time increases more rapidly than RS-KNN.

6 Conclusion

The problem of processing k-NN queries over moving objects is fundamental in many applications. In this study, we propose SRI, a novel index can better support the processing of spatial k-NN queries than other indexes. Based on SRI, we design the SR-KNN algorithm that answers k-NN queries over moving objects without numerous iterative calculations occurred in grid-based approaches and achieves a good performance.

References

1. K. L. Cheung and A. W.-C. Fu, "Enhanced nearest neighbour search on the r-tree," *ACM SIGMOD Record*, vol. 27, no. 3, pp. 16–21, 1998.
2. A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.
3. X. Yu, K. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *ICDE*, 2005, pp. 631–642.
4. K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *SIGMOD*, 2005, pp. 634–645.
5. M. Cheema, "CircularTrip and arctrip: Effective grid access methods for continuous spatial queries," in *DASFAA*, 2007, pp. 863–869.
6. Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *VLDB*, 2002, pp. 287–298.
7. K. Mouratidis and D. Papadias, "Continuous nearest neighbor queries over sliding windows," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 6, pp. 789–803, 2007.
8. M. S. H. A.-K. Sultan Alamri, David Taniar, "Tracking moving objects using topographical indexing," *Concurrency and Computation: Practice and Experience*, 27(8): 1951-1965, 2015.
9. K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos, "Fast nearest-neighbor query processing in moving-object databases," *GeoInformatica*, vol. 7, no. 2, pp. 113–137, 2003.
10. T. Seidl and H. Kriegel, "Optimal multi-step k-nearest neighbor search," in *SIGMOD*, 1998, pp. 154–165.
11. S. Chaudhuri and L. Gravano, "Evaluating top-k selection queries," in *VLDB*, 1999, pp. 399–410.
12. B. Gedik, K. Wu, P. Yu, and L. Liu, "Processing moving queries over moving objects using motion-adaptive indexes," *Knowledge and Data Engineering*, vol. 18, no. 5, pp. 651–668, 2006.
13. C. Yu, B. Ooi, K. Tan, and H. Jagadish, "Indexing the distance: An efficient method to knn processing," in *VLDB*, 2001, pp. 421–430.
14. B. Zheng, J. Xu, W.-C. Lee, and L. Lee, "Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services," *The VLDB Journal*, vol. 15, no. 1, pp. 21–39, 2006.
15. K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of spatial queries in wireless broadcast environments," *IEEE Transactions on Mobile Computing*, vol. 8, no. 10, pp. 1297–1311, 2009.
16. M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: scalable incremental processing of continuous queries in spatio-temporal databases," in *SIGMOD*, 2004, pp. 623–634.
17. X. Xiong, M. F. Mokbel, and W. G. Aref, "SEA-CNN: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *ICDE*, 2005, pp. 643–654.
18. D. Šidlauskas, S. Šaltenis, and C. S. Jensen, "Parallel main-memory indexing for moving-object query and update workloads," in *SIGMOD*, 2012, pp. 37–48.
19. H. Wang and R. Zimmermann, "Snapshot location-based query processing on moving objects in road networks," in *SIGSPATIAL GIS*, 2008, pp. 50:1–50:4.