Efficient k-Closest-Pair Range-Queries in Spatial Databases

Shaojie Qiao
School of Computer Science
Sichuan University
qiaoshaojie@cs.scu.edu.cn

Changjie Tang
School of Computer Science
Sichuan University
tangchangjie@cs.scu.edu.cn

Jing Peng
School of Computer Science
Sichuan University
pj@pku.edu.cn

Hongjun Li
School of Computer Science
Sichuan University
lihongjun@cs.scu.edu.cn

Shengqiao Ni
School of Computer Science
Sichuan University
nishengqiao@cs.scu.edu.cn

Abstract

In order to efficiently retrieve the k closest pairs between two spatial data sets in a specified space, such as in GIS and CAD applications, we propose a novel algorithm to handle the k-closest-pair range-query problem by progressively augmenting the query window instead of finding all objects in the whole space. We first describe a specific range estimation method to compute the circle query range which helps eliminate the unnecessary distance calculations among spatial objects and improve performance. Then, we use R-tree to store closest pairs and give algorithms for maintaining this structure. Extensive experiments performed with synthetic as well as with real data sets show that the new algorithm outperforms the existing approaches in most cases. In particular, this technique works well when two spatial data sets are identical.*

1. Introduction

Spatial databases have recently received more attention. The term "Spatial Database" refers to such database that stores data on, above or below the earth's surface [1], or a database that is optimized to store and query data related to objects in space, e.g., points, lines, polygons, volumes and other kinds of geometric objects [2]. Spatial databases include specialized systems such as geographical spatial databases, computer aided design datasets etc [2]. It has been applied to several applications, including mapping, urban planning, transportation management, resource management, etc [3].

The typical spatial queries are shown as follows [3].

- A "point location query" seeks for the objects that fall on a given point, e.g., the city where a specific university resides.
- A "range query" seeks for the objects that are included within a given area, usually depicted as a rectangle, e.g., the rivers that cross a forest.
- A "join query" has many forms. It involves two or more spatial data sets and discovers pairs of objects that satisfy a given spatial predicate, e.g., the pairs of hospitals and schools, ordered by distance 30km between them.
- The "nearest neighbor query" seeks for the objects locating more closely to a given object. It discovers k such objects (k nearest neighbors), e.g., there are k ambulances closer to a fire incident with k injured persons.

One important spatial database query is called "k Closest Pairs Query" (k-CPQ), which combines join and nearest neighbor queries in order to find the closest k pairs of spatial objects from two distinct data sets that have the k ($k \geq 1$) smallest distances [2]. Like a join query, all pairs of objects are candidates for the result. When $k = 1$, it degenerates to find the closest pair of spatial objects. For instance, in a criminal spatial database, investigators may be interested in finding the closest apartments and airports, the closest streets and subways, etc. The problem of k-CPQ has recently received more interest in the research of spatial databases (e.g. GIS, CAD databases) [2, 3, 4, 5]. It is a difficult problem to solve if there is much overlap between two distinct data sets. When two data sets are identical, there exists a large volume of overlap, traditional algorithms for closest pair queries, namely the Naive, the Exhaustive, the Simple, the Sorted Distances, and the Heap algorithm [2] cannot handle this problem appropriately. A better attempt is

978-0-7695-3185-4/08 $25.00 © 2008 IEEE
DOI 10.1109/WAIM.2008.12

IEEE computer society

detailedly illustrated in [5], Shan et al. proposed an extension to the CP (closest pair) problem to involve a query range, i.e., finding the k closest pairs of objects inside a given range (Range-CP problem). The CP problem is a specific case of the Range-CP problem (RCP) when the query range is the whole space [5]. The k Range-CP query (k-RCPQ) problem is of great practical value when applied to spatial databases. For CP problem, the pair of closest objects is predefined. But, people often concern the query results within a given region (workspace) instead of the whole space in practice. The key problem in RCP query is that the query range can be arbitrary, and to maintain the k closest pairs of spatial objects for every possible range is costly [5]. This problem motivates us to transform a k-RCPQ into one or multiple window queries.

The contributions of this paper are given as follows.

1) We propose an extension to the k-closest-pair range-query problem by employing the window query technique instead of querying all pairs of objects in the whole space.
2) We borrow the specific density-based method to estimate the range of a circle query.
3) We use a specific R*-tree to store the indices of spatial objects and give a node update algorithm for maintaining this structure.
4) Performance studies are presented to demonstrate the efficiency of our proposed approaches by comparing with other k-RCPQ algorithms.

2. Problem definition

2.1. Definition of k-RCPQ problem

The k-RCPQ problem is to find k pairs of spatial objects from a subset of objects in two data sets, where the subsets of objects are those whose locations are in a given spatial range. The definition is similar to the k-CPQ problem [2]. The difference lies in that we add a query range that intersects the given two data sets, and find the k-CPQ from this specified query range.

2.2. Examples of k-RCPQs

The commonly-used distance metrics are shown in Fig. 1(a) [5], and the minimum bounding rectangle (MBR) covering each data set is depicted by a solid-line rectangle. Given two MBRs A and B. Following [2], we can define some metrics. MinMinDist(A, B) represents the smallest distance between A and B boundaries, and shown as follows.

$$MinMinDist(A,B) = \min\{MinDist(p_i, q_j)\} \quad (1)$$

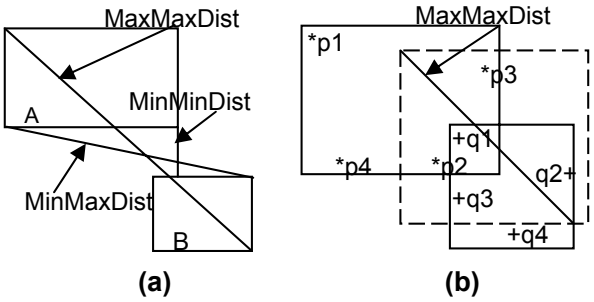


Figure 1. Two MBRs and metrics

When two MBRs intersect, MinMinDist(A, B) equals 0. MaxMaxDist(A, B) [5] represents the maximum distance between A and B boundaries and it is an upper bound of the distance between a spatial object enclosed in A and the other spatial object included in B. MinMaxDist(A, B) [5] is the minimum distance which guarantees that there is one pair of objects, one in A and the other one in B.

In Fig. 1(b), the first data set is represented by stars, while the second set by crosses. We extend these metrics to the case when there is a query range involved (depicted by a dashed rectangle). We can observe that the 1-RCPQ is (p_2, q_1), and the 2-RCPQs are (p_2, q_1) and (p_2, q_3). In the same manner, it is easy to find the 3-RCPQs, 4-RCPQs, etc. As shown in Fig. 1(b), if a query range is given, the value of MinMinDist keeps the same, while the value of MaxMaxDist changes. It is easy for us to prove that MaxMaxDist(A, B, R) = MaxMaxDist($A \cap R$, $B \cap R$) [5]. In real-world cases, the MBRs of two data sets always partially overlap. It is evident that the higher the percentage of overlapping between two MBRs, the higher is the probability that more pairs with smaller distances appear and the algorithm for discovering the closest pairs needs to examine more candidate pairs with distances that differ slightly [2]. In this paper, an efficient k-RCPQ algorithm is presented to handle the higher overlap case between two spatial data sets.

3. Related work

The closest pair query problem has recently been extended to spatial databases [2, 3, 4, 5]. The earlier work mainly focuses on k nearest neighbor (kNN) queries. Hjaltason et al. [6] proposed an incremental algorithm for computing the distance join and distance semi-join between two R-tree indices. The demerit of this approach is that it has to store any pair of index entries that can cause several disk accesses.

One improvement approach was proposed by Zhang et al. [7], i.e., the virtual height optimization. The merit of this optimization is that it can avoid pushing a pair

of objects into a queue. Another modification is that they set a threshold value T as the distance of the closest pair. If their distance is smaller than T, the pair of objects can be eliminated that helps prune tree nodes as well as save I/O cost.

However, the above algorithms are not efficient when two spatial data sets have much overlap, especially when two data sets are identical.

In order to solve the k-RCPQ problem, Shan et al. [5] developed two SRCP-trees. However, the SRCP-tree only support the closest pair query for one type of objects. In practice, the data sets may contain multi-type objects, i.e., airport, bar, market, bank, etc. The query may ask for k closest pairs of any two types of objects.

This study extends the solutions proposed by Shan et al. [5] to solve the k-RCPQ problem in two distinct data sets. Our solution is inspired by two intuitions.

1) The query range is always a large search space. As well as we know, the distribution of the index entries is always clustered, thus we do not need to find the k-CPs in the whole query space. Instead, we retrieve exactly k closest pairs within a circle with one point from one data set as center and the distance between this point and the kth nearest neighbor in the other data set as radius.
2) If we apply the incremental join algorithm to the k-RCPQ problem, every pair of nodes intersecting the query range must be pushed into a queue. But, if the shortest distance of objects is stored in the sub-tree, then this value can be used instead of zero as the priority [5]. This increases the chance that a pair can be pruned.

4. Efficient k-RCP query algorithm

This section introduces a novel k-RCP query algorithm. It employs a specific range estimation method [8] and a new R*-tree that integrates the idea of SRCP-tree [5] to store closest pairs. The tree structure proposed in this paper is named RCP-tree based on Beckmann's R*-tree [9], which outperforms the existing R-tree variants, such as Guttman's quadratic R-tree [10].

The query rang can be depicted by a window, i.e., $w = [(x_l, y_l); (x_u, y_u)]$, where (x_l, y_l) and (x_u, y_u) are its lower-left and upper-right corners of the query range. Here, the coordinate of the center of the query window is set as $((x_l + x_u)/2, (y_l + y_u)/2)$. In this paper, we use the circle query method introduced in [8] to approximate the range of the window query that helps reduce the query workspace.

Given a set of spatial objects denoted as S and a window w, a k-CP window query refers to finding the k closest pairs of objects in S located in w.

Before outlining the k-RCPQ algorithm, here we define some notations. Let A and B be two spatial data sets, i be an index object pointing to some node in the R*-tree, $Node(i)$ be the node that i points to, and $Sub(i)$ be the sub-tree rooted by $Node(i)$.

We follow the commonly-used R*-tree insertion algorithm to insert the objects. Here we use the same node splitting technique as R*-tree. The different is that we have to maintain two R*-tree and update the information of a node in one R*-tree while the pairwise node in the other R*-tree needs to be split.

4.1. Update

If a node is changed, its index record must be deleted, updated, and then re-inserted, so that it will find its way to the right place in the R*-tree. The node update algorithm is shown in Algorithm 1.

Algorithm 1: Update(Node e, R*-tree T, Stack S, Path_set P)

```
1: if (e is a leaf) //e is from the other R*-tree;
2:    for (each entry o pointing to e)
3:       {find a node t in T that has the smallest distance
         from e within a circle with e.dist as radius;}
         //e.dist is the distance from the root node to e
4:       if (t exists and e.dist > MinMinDist(t, o))
5:          {e.dist := MinMinDist(t, o) and update e;}
6:          for (each node s ∈ S)
7:             if (s.dist > MinMinDist(t, s))
8:                {s.dist := MinMinDist(t, s) and update s;}
9: else //e is an internal node
10:   {put e into S and e.dist to P;}
11:   for (each child node se of e)
12:      {update(se, T, S, P);} //iterations
13:   pop a node from S;
```

The basic idea of the update algorithm is to find the shortest distance from $Sub(e)$ to the nodes in the other R*-tree within a circle with a given radius (lines 2-8). In lines 11-12, we update the internal node (none-leaf node) information via iterations. For RCP-tree, we have to update two trees, that is, we have to call the function update($A.root$, B, S, P) and update(B.root, A, S', P') simultaneously, where $A.root$ and $B.root$ represent the root nodes of tree A and B, respectively. The time complexity of this function is $O(m*n)$, m is number of entries stored in each node of one R*-tree, and n is the size of the stack that is used to store the nodes in the other R*-tree.

4.2. Query

We employ the density-based range estimation approach [8] and SRCPQuery algorithm [5], and propose a new range query algorithm (WRCPQ) shown in Algorithm 2 for the RCP-tree.

Algorithm 2: *k*-RCP Query Based on Window

Input: R*-tree *A*, *B*
query range *R*
required number of closest pairs *k*
Output: *k* closest pairs of objects *queue*

```
1: rang := RangeEst1(k);
2: window := [(x_q-rg, y_q-rg); (x_q+rg, y_q+rg)];
3: list := WRCPQuery(A, B, R, window);
4: queue := PriorityQueue(k);
5: count := 0;
6: for (each pair of objects cp in list)
7:   if (there are k elements in queue)
8:     if (the distance of the top element > cp.dist)
9:     {remove the top element and push cp into queue;}
10:  else
11:    push cp into queue;
12:    count := count + 1;
13: while (count < k)
14:   range := RangeEst2(k, count);
15:   window := [(x_q-rg, y_q-range); (x_q+rg, y_q+rg)];
16:   list := WRCPQuery(A, B, R, window);
17:   count := 0;
18:   for (each pair of objects cp' in list)
19:     if (there are k elements in queue)
20:       if (the distance of the top element > cp'.dist)
21:     {remove the top element and push cp' into queue;}
22:       else
23:         push cp' into queue;
24:         count := count + 1;
25: return queue;
```

In line 1, the RangeEst1 function [8] is used to estimate the approximate range of the circle query.

The algorithm derives the query window from the estimated range by calling the *WRCPQuery* function. In line 3, the window query results are inserted into a temporary queue *list*. In line 4, we create an empty priority query *queue* to maintain the *k* closest pairs (in the form of a triple). Lines 6-12 are used to find the *k* closest pairs of objects. In line 13, if *count* (the number of closest pairs found so far) is larger than or equal to *k*, terminate this algorithm and all the results are stored in *queue*. Otherwise, other closest pairs have to be obtained by augmenting the query window. The augmented range is computed from the current window and *count* using the RangeEsti2 function [8] (line 14). By RangeEst2, we obtain a range value r_n that is larger than the input range r_{n-1} and less than or equal to $r_{\max} = \sqrt{(x_u - x_l)^2 + (y_u - y_l)^2}$ /2. The query radius r_n is defined as [8].

$$r_n = \begin{cases} 2r_{n-1} & \text{if count} = 0; \\ 2\sqrt{\dfrac{k}{\pi \times count}}\, r_{n-1} & \text{if } 0 < \text{count} \le \text{k -1}. \end{cases} \quad (2)$$

Lines 18-24 use the similar manner as lines 6-12 to find the *k*-CPs within the query window. Finally, the results are popped from *queue* (line 25).

This range estimation approach has the following benefits [8]. First, the space used to store the density information is only some bytes. Second, each time a new range estimate is calculated, it is derived from the previous queries, which helps eliminate the repeated distance calculations from the previous queries.

WRCPQuery function works as follows. First, it checks to see if both objects (from two data sets) in the closest pair are inside the query window *w* across the query range. If yes, output the closest pair to a priority queue *Q*. Each element in *Q* contains a pair of index entries and a distance value (a triple). For a pair of two different entries, the value is their *MinMinDist*. Second, pop from *Q* one triple at a time and process it. This step is the main task of WRCPQuery. In this paper, we use the similar query algorithm as *SRCPQuery* proposed in [5] to discover the closest pairs. For space limitations, we do not introduce this algorithm in this paper.

5. Experiments and discussions

In order to evaluate the performance of WRCPQ, we compare it with the popular *k*-CP query algorithm, i.e., the Heap algorithm [2] (denoted by HEAP). For solving the self range CP problem [5], we also implemented the SRCP-Tree (version two) [5], since SRCP-tree (version two) is measured to have better query performance than SRCP-tree (version one).

The experimentations consist of evaluating the effect of two important factors included: (1) the portion of overlapping between two data sets and (2) the number of closest pairs (the *k* value). We use the following data sets.

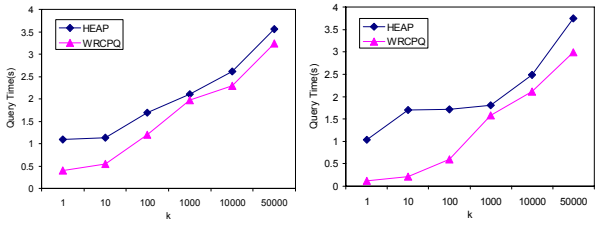
- real data sets from the city database [11] consisting of more than 380,000 cities around the world depicted by longitude and latitude,
- a collection of synthetic data sets of cardinality 20K, 40K, 60K, and 80K points with *x* and *y* coordinates following a uniform distribution.

The algorithms were implemented in Java based on spatial index library [12]. Our experiments were run on a PC of 1.4GHz AMD Sempron processor with 512 MB of RAM. Note that, the two data sets used for each set of experiments were either real or synthetic.

5.1. Performance comparison on k-RCPQs

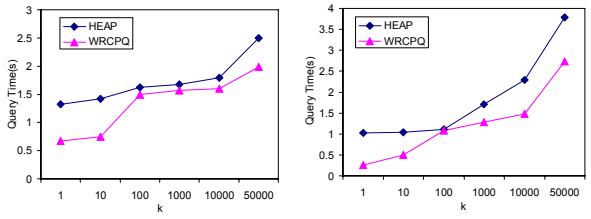
This section estimates the performance of WRCPQ with HEAP for *k*-RCPQ problem. Since the closest pair queries are very sensitive to the relative location of the two data sets involved, especially the percentage of overlapping [5], we evaluate the sensitivity of both algorithms on the portion of overlapping.

For this set of experiments, we run *k*-RCPQs in the real and in the synthetic data sets, with *k* varying from 1 up to 50,000. Fig. 2-3 illustrate the performance of each algorithm assuming (a) 0% and (b) 100% overlapping workspaces with the real and the synthetic data sets, respectively.



(a) (b)

Figure 2. Comparison of k-RCPQ algorithms: real data in (a) 0% and (b) 100% overlapping workspaces



(a) (b)

Figure 3. Comparison of k-RCPQ algorithms: synthetic data in (a) 0% and (b) 100% overlapping workspaces

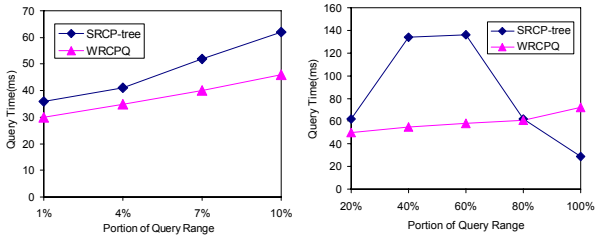
By Fig. 2-3, the query time of both algorithms gets higher as the *k* value grows in the real as well as in the synthetic data sets. Since both algorithms need more time to find the increased CPs as *k* grows. We can also observe that, the portion of overlapping plays an important role in the ranking of these two algorithms. For overlapping workspaces (Fig. 2(b) and Fig. 3(b)), WRCPQ achieves the best improvements of 87.9% (for real data) and 75.8% (for synthetic data) faster than HEAP. However, for none-overlapping workspaces (Fig. 2(a) and Fig. 3(a)), the best improvement are only 62.9% (for real data) and 49.4% (synthetic data). This further illustrates that WRCPQ is more adaptive to handle the cases of higher overlapping.

5.2. Comparison on k-RCPQs within the same data set

The *k*-RCPQ problem in the same data set is another practical problem. The SRCP-tree is an alternative to solve this problem and outperforms other approaches. We test the performance of WRCPQ by comparing the query time cost with that of SRCP-tree (version two).

5.2.1. The effect of query range. In order to keep consistent with the parameter settings in SRCP-tree, for this set of experiments, we first compare these two algorithms with a 40K real data set (the results agree with that from the synthetic data set as well) when the query range is small, between 1% and 10% of the workspace as shown in Fig. 4(a). Then, we extend the query range to measure the performance of WRCPQ and SRCP-tree. The results are presented in Fig. 4(b).

By Fig. 4(a), WRCPQ outperforms HEAP. As the query range increases, more objects will intersect the query range, so the query time increases. For WRCPQ, since the range of query window is expanded gradually, more nodes can be pruned before any distance calculation when it finds the *k* closest pairs of objects.

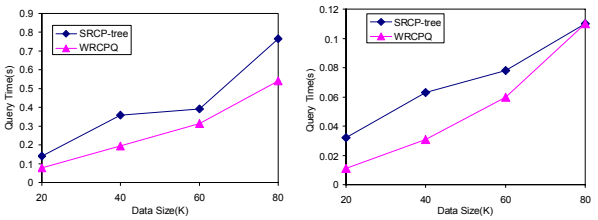


(a) (b)

Figure 4. Query time comparison for different query range percentage

In case of a large query range, as shown in Fig. 4(b), the query time of SRCP-tree decreases dramatically when the portion of query rang is higher than 60%. The reason is that when the query range gets larger, the probability of a CP pair in the query range increases [5]. However, WRCPQ is not sensitive to the query range when it finds the *k*-CPs. In this case, when the query range percentage reaches to 20%, WRCPQ finds the *k* closest pairs of objects, and the query time changes slightly even in case of a large query range.

5.2.2 Query time comparison in varying data size. We also compare the same query area ratio (50% of space) with the real as well as with the synthetic data sets. As shown in Fig. 5, when the size of the data set increases, the query time goes up as well. This is due to the increased spatial objects in the query range. In addition, we observe that WRCPQ performs better than SRCP-tree in the real as well as in the synthetic data sets. This is because the SRCP-tree cannot avoid searching the whole workspace to find the k closest pairs. As shown in Fig. 5, the query time of WRCPQ increased linearly with the data size. This is due to the progressive augment of the query window.



(a) (b)

Figure 5. Query time comparison for varying data size: (a) real and (b) synthetic data

6. Conclusions and future work

The closest-pair range-query for multiple types of objects in spatial databases is a challenging problem. In this study, we propose an efficient k-RCP query algorithm based on query window, and use a density-based range estimation method to approximate the circle query range. When discovering the k-RCPQs, there is no need to expand the window that helps eliminate the unnecessary distance calculations. An R*-tree structure is used to store the indices of the spatial objects. In particular, it is a better solution when two data sets are identical. Experiments demonstrate that WRCPQ outperforms the existing approaches w.r.t. the portion of overlapping and the k value between the workspace of two data sets.

The ongoing research directions include: (1) developing new range estimation methods for the window queries, such as the bucket-based approach [8]; (2) extending WRCPQ on multiway spatial joins [13]; (3) extending our proposed algorithm for the approximate k-closest-pair range-query [14].

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 60773169, and the Youth Software Innovation Project of Sichuan Province under Grant No. 2007AA0032.

References

[1] R. Laurini, and D. Thomson, *Fundamentals of Spatial Information Systems*, London: Academic Press, 1992.

[2] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases", in *Proc. ACM SIGMOD'00*, pp. 189-200, 2000.

[3] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Algorithms for processing K-closest-pair queries in spatial databases", *Data and Knowledge Engineering*, vol. 49, pp. 67-104, 2004.

[4] X. Liu, Y. Liu, and Y. Xiao, "Processing Constrained K Closest Pairs Query in Spatial Databases", *Wuhan University Journal of Natural Sciences*, vol. 11, no. 3, pp. 543-546, 2006.

[5] J. Shan, D. Zhang, and B. Salzberg, "On spatial-range closest-pair query", in *SSTD'03*, LNCS 2750, pp. 252-270, 2003.

[6] G. R. Hjaltason, and H. Samet, "Incremental distance join algorithms for spatial databases", in *Proc. ACM SIGMOD'98*, pp. 237-248, 1998.

[7] D. Zhang, V. J. Tsotras, and B. Seeger. "Efficient temporal join processing using indices", in *Proc. ICDE'02*, pp. 103-113, 2002.

[8] D. Liu, E. Lim, and W. Ng, "Efficient k nearest neighbor queries on remote spatial databases using range estimation", in *Proc. SSDBM'02*, pp. 121-130, 2002.

[9] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles", in *Proc. ACM SIGMOD'90*, pp. 322-331, 1990.

[10] A. Guttman, "R-trees a dynamic index structure for spatial searching", in *Proc. ACM SIGMOD'84*, pp. 47-57, 1984.

[11]http://geonetwork.unocha.org/mapsondemand/srv/en/main.home

[12] M. Hadjieleftheriou. (2002) [online]. Available: http://research.att.com/~marioh/spatialindex/index.html

[13] N. Mamoulis, and D. Papadias, "Multiway spatial joins", *ACM Transactions on Database Systems*, vol. 26, no. 4, pp. 424-475, Jan. 2002.

[14] N. Koudas, B. Ooi, K. Tan, and R. Zhang, "Approximate NN queries on streams with guaranteed error/performance bounds", in *Proc. VLDB'04*, pp. 804-815, 2004.