Reverse k-Nearest Neighbor Monitoring on Mobile Objects

Tobias Emrich, Hans-Peter Kriegel, Peer Kröger, Matthias Renz, Naixin Xu, Andreas Züfle
Institute for Informatics, Ludwig-Maximilians-Universität München, Germany
{emrich,kriegel,kroeger,renz,Naixin,zuefle}@dbs.ifi.lmu.de

ABSTRACT

In this paper we focus on the problem of continuously monitoring the set of Reverse k-Nearest Neighbors (RkNNs) of a query object in a moving object database using a client server architecture. The RkNN monitoring query computes for a given query object q, the set RkNN(q) of objects having q as one of their k-nearest neighbors for each point in time. In our setting the central server can poll the exact positions of the clients if needed. However in contrast to most existing approaches for this problem we argue that in various applications, the limiting factor is not the computational time needed but the amount of traffic sent via the network. We propose an approach that minimizes the amount of communication between clients and central server by an intelligent approximation of the position of the clients. Additionally we propose several poll heuristics in order to further decrease the communication costs. In the experimental section we show the significant impact of our proposed improvements to our basic algorithm.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval—*Query Processing*

General Terms

Algorithms, Performance

Keywords

RkNN, Mobile Objects, Spatial Pruning, Query Processing

1. INTRODUCTION

The prevalence of inexpensive Global-Positioning-Devices (GPS) enables to track and coordinate large numbers of continuously moving objects, and store their positions in databases. This results in new challenges for query optimization in such databases because it is no longer sufficient to have a query processing strategy that is tailored to meet disk specific problems only. Rather, query processing in mobile databases needs to consider also problems specific to the hardware of mobile devices such as power source, network traffic, etc. Particularly, a main goal in many applications where mobile objects are tracked is to minimize the network traffic and, as a consequence, minimize the drain of the power source of the devices.

In this paper, we consider the monitoring of the RkNNs of a query object (that may itself move or not) in a set of mobile clients at a central server. The set of reverse k-nearest neighbors (RkNN) of a query object q contains all objects in a database that have q among their corresponding k-nearest neighbors (kNN). Recent approaches for supporting RkNN queries focus on traditional database settings using spatial or metric index structures. Considerably less work has been done so far to support RkNN queries on mobile objects that may not be indexed by a point access method. In fact, many applications that deal with mobile objects require to continuously monitor the RkNNs of query objects in the database in order to know which objects the queries have an influence on. Our solution can handle mono-chromatic (queries and possible answers are drawn from the same set of clients) as well as bi-chromatic (queries and possible answers are drawn from two different sets of clients) scenarios, though we will focus on the mono-chromatic case for clarity reasons. To track and observe large numbers of continuously moving objects we use a standard model. The latest submitted positions of clients are stored in a database at the server. After the time slot of this transmission, the position of the client is conservatively approximated at the server by a minimal bounding box that contains all possible positions the client may have reached since its last location update and that usually grows over time.

2. CONTINUOUS RKNN MONITORING

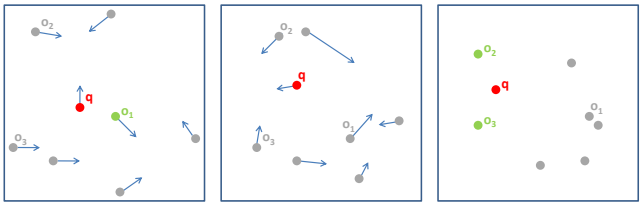
In the following, we assume that $\mathcal{D}$ is a database of n objects (clients) moving continuously within a 2D Euclidean space and $dist$ is the Euclidean distance[1]. In addition, we assume that the objects (clients) are connected with a central server via a wireless network and can send their exact positions when requested from the server. We focus on the mono-chromatic case here but all concepts can easily be extended for bi-chromatic scenarios.

At server side the position of each object $o \in \mathcal{D}$ is approximated by a 2-dimensional axis aligned rectangle O that minimally bounds the possible positions of o. The object sends its exact position $o.pos$ to the server only if necessary. During query processing at the server, the exact position of o will not be updated at server side as long as the exact position $o.pos$ is within the rectangle O and the query can be answered based on the information of the object positions that is currently available at the server.

The set of *reverse k-nearest neighbors* (RkNNs) of an object q contains all objects $pin\mathcal{D}$ such that q is one of the k-nearest neigh-

[1]The concepts described here can also be extended to any dimensionality $d > 2$ and any L_p-norm.

ACM GIS '10, November 2–5, 2010, San Jose, California, USA.



(a) Snapshot 1. (b) Snapshot 2. (c) Snapshot 3.

Figure 1: RkNN monitoring in a database of moving objects.

bors (kNN) of p, i.e., the distance $dist(p, q)$ is smaller or equal to the kNN-distance of p. The problem of monitoring the RkNNs of a query q can be illustrated by considering the sample database of mobile objects depicted in Figure 1 for $k = 1$. Three snapshots at consecutive time slots of the spatio-temporal data objects are shown. At each of these snapshots, the current position of each client is given and the current direction of movement as well as the velocity of each object is indicated by a vector. At the first snapshot (cf. Figure 1(a)), the query object q has object o_1 as its only R1NN. At next time slot (cf. Figure 1(b)), object o_1 moves away from q and is no longer a R1NN of q. At this time $R1NN(q) = \emptyset$, since no object in the database has q as its 1NN. At the next time slot (cf. Figure 1(c)) objects o_2 and o_3 are R1NNs of q.

2.1 General Idea

The aim of monitoring the RkNNs of a query q is to keep track of these changes of $RkNN(q)$, i.e., to continuously update the set of RkNNs of q (or multiple query objects $\{q_1, \ldots, q_m\}$) such that the set $RkNN(q)$ (or $RkNN(q_i)$) is valid at each point of time. During the monitoring, we want to keep the communication cost between the clients and the server as low as possible. For this reason, we try to avoid unnecessary position updates at the server side. For a straightforward approach to monitoring $RkNN(q)$, we require the exact positions of each mobile object in $\mathcal{D}$ at each point of time. Therefore, we propose that each time a mobile object sends its position to the server, the server sends to this mobile object a rectangular region (approximation) containing this position which we call *safe region* in the remainder. The mobile object automatically sends a position update to the server once it leaves its safe region and is assigned a new safe region by the server. The server can use the safe region as an approximation of the moving object's true position. The safe regions of the moving objects can be used to conservatively prune candidates, and progressively detect results without any communication cost.

In order to implement this idea, the following issues need to be solved. First, we need a basic filter-refinement framework. To obtain an actual algorithm from the framework, we need to explore possibilities for correctly and sufficiently performing conservative pruning and progressive true hit detection based on rectangular approximations (safe regions). This will be discussed in Section 3. Second, for the remaining candidates, we need to decide, which objects should be refined next, i.e., for which objects should the server request the exact positions. This is not trivial because the exact position of a candidate might be sufficient to decide about the status of other candidates without further refinement. We will solve this question in Section 4. Finally, the size and position of the safe regions obviously have an impact on the number of necessary position polls. Obviously, an object that is at the edge of being an RkNN of q should have a rather small safe region to allow more accurate decisions based on these region. Contrarily, an object that is far from being an RkNN of q should have a large safe region in order to reduce the number of updates caused by the object leaving its safe region. We will deal with this problem in Section 5.

2.2 Initialization

Initially, we need to create the safe regions of all clients in the database and to generate the initial set of RkNNs. We assume that each object $o_i \in \mathcal{D} \cup \{q\}$ is assigned a safe region O_i conservatively approximating its current location, i.e., O_i covers the exact location of o_i [2]. $\mathcal{D}'$ denote the set of all save regions of objects in $\mathcal{D} \cup \{q\}$. At the beginning, the server is assumed to be aware of the save regions in $\mathcal{D}'$ but not the exact object locations. Based on this setting, the server can initialize $RkNN(q)$ by performing the snapshot RkNN query. We propose a filter/refinement solution for this initial query where the query object and all database objects are approximated using the general pruning concepts described in [6]. For the selection of the object whose position is polled next, we use heuristics detailed in Section 4. Each object which has submitted its exact location is stored in a list called update_list. For all these objects we have to update the safe regions. The filter iteratively refines objects until there are no candidates anymore and $RkNN(q)$ is determined. Finally, we have to compute new safe regions for all objects in update_list. Heuristics for the determination of appropriate safe regions are given in Section 5. Subsequently, the safe regions are sent back to the objects (clients).

2.3 Monitoring

After performing the query initialization, the first query result is stored in $rknns$. It is obvious that if the objects now change their locations over time then the query result may change as well. However, the safe regions are designed such that the query result remains valid as long as the objects stay within their safe regions. However, when an object $o_i \in \mathcal{D} \cup \{q\}$ leaves its safe region it has to inform the server about its new position. Then, the current query result could become invalid and has to be updated. The safe region of o_i is repositioned and possibly changes its extension. Heuristics for this method are presented in Section 4. As long as the filter step yields candidates, the exact position of an object has to be polled and the safe region has to be updated. Similarly, new safe regions of refined objects are computed and returned to the clients. Thus, during monitoring, communication is only necessary when performing a location update.

3. SPATIAL PRUNING STRATEGIES

In the filtering step of the initialization and monitoring procedures, we need to decide if the safe region O_1 of object o_1 is closer to the safe region O_2 of an object o_2 than the safe region Q of the query q. For this purpose, we use two existing techniques for spatial pruning, namely the Minimal-Maximal-Distance (short: MinMaxDist) based spatial pruning and the optimal spatial pruning Criterion [6]. Probably the most widely used decision criterion for spatial pruning among rectangles is based on two well known metrics defined on rectangles [10]. The minimum distance $MinDist(A, B)$ between two rectangles A and B is the largest distance that always underestimates the distance of point pairs $(a, b) \in A \times B$. The maximum distance $MaxDist(A, B)$ between two rectangles A and B is the smallest distance that always overestimates the distances of all point pairs $(a, b) \in A \times B$. The Min-/MaxDist criterion

[2]Note that save regions are also allowed to have zero extension, in this case they represent the exact locations.

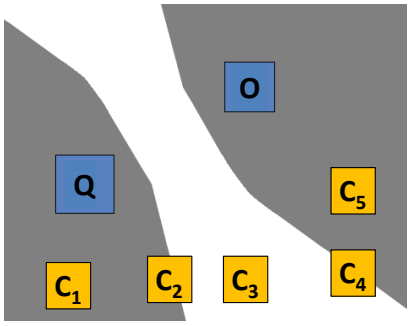


Figure 2: Illustration of the maximum pruning potential heuristics.

has certain limitations that are discussed in [6]. The second spatial pruning technique [6] that we propose to plug into our algorithm overcomes these limitation and is shown to be optimal.

4. REFINEMENT HEURISTICS

In our initialization and monitoring procedures, we need to choose an object that sends its current position to the server in order to compute the RkNNs of q. The Mindist heuristics propose to refine the object with the smallest minimal distance to the query object q. In the following we propose a novel heuristics called *Maximum Pruning Potential Heuristics* that utilizes the optimal spatial pruning technique. The main idea of this heuristics is to choose the object o, for which refinement has the highest expected candidate size reduction. Intuitively, the pruning potential of an object o is composed of the number of objects in the database that do not prune the safe region O of o (if o is a candidate) but may prune the exact representation of o (self pruning) and the number of candidates c, for which the refinement of o may prune c (mutual pruning).

For the self pruning potential of a given candidate object c the task is to determine the number of objects o in the database that may prune c if c is refined. Consider the example in Figure 2. Here, the pruning region of the safe region O of a database object o is depicted. In particular, the shaded region containing Q (analogously: O) depicts the half-space for which any point in this half-space is definitely closer to q (analogously: o) than to o (analogously: q). Note that we do not have to compute these pruning half-spaces. We just use them for illustration purpose here. The shaded region containing O is called the *conservative pruning region* of O, whereas the region containing Q is called *progressive pruning region* of O. The unshaded region contains the points p for which it cannot be decided (based only on the approximations Q and O) if p is closer to q or to o. Considering the safe region of the candidate c_1 (analogously: c_5), it is clear that refining the region C_1 (analogously: C_5) will not give us any new information with respect to o, since c_1 (analogously: c_5) are definitively closer to (analogously: farther from) q than o when considering the approximations only. In contrast, refining the region C_2 of candidate c_2 may result in c_2 being definitely closer to q than o which may allow c_2 to be reported as a true hit. Analogously, the exact position of c_4 may be in the conservative pruning region, so that refining its safe region C_4 may allow c_4 to be pruned. Therefore, the self pruning potential of c_2 (analogously: c_4) is increased by 1, since refinement of the safe regions of these objects may yield new information about the topology between q, o and c_2 (analogously: c_4). However, the self pruning potential for c_3 cannot be increased through the refinement of its safe region $C - 3$, since regardless of the exact position of c_3, it cannot be decided if it is closer to q or to o.

We can see that for the safe region Q of a given query q and the safe region O of a database object o, the self pruning potential is incremented for each candidate which is located on the edge of either the conservative or the progressive pruning region. Although computing the curvilinear pruning regions is expensive in the case of rectangular approximations [5], we can use the technique of *partial domination* ([6]) to detect if a candidate is located on the edge of the conservative (analogously: progressive) pruning region between the safe regions Q and O.

For the mutual pruning potential of a given database object o, the task is to determine the number of candidates c in the database that may be pruned if the safe region C of c is refined. Again, consider the example in Figure 2. The task is to determine the mutual pruning potential of o. It is clear that no more information is gained regarding candidates c_1 and c_5 since the location of these object w.r.t. q and o can be decided based on the safe regions Q, O, C_1, and C_5, only. Refining the safe region O of object o may allow a decision to be made for objects c_2 and c_5, since refining O will generate new pruning regions that contain the old pruning regions. In addition, the change of the pruning region may also allow a decision for object c_3 based on C_3. Since the refinement of O may allow to decide the spatial topology of the regions C_2, C_3 and C_4, the Mutual-Pruning Potential of o is three in this example.

We increase the mutual pruning potential of an object o by 1 for each candidate c for which refining the safe region O of o may allow a new decision. For identifying those objects, we can again use the optimal pruning criterion in [5]. If c is not yet pruned, the refinement of O may allow c to be pruned if and only if the safe region C of c does not intersect both the conservative and the progressive pruning region of O.

Our maximum pruning potential heuristics refines the objects with the highest total self- and mutual pruning potential.

5. SAFE REGIONS

In our solution, the server is required to determine new safe regions for moving objects that have been refined in the refinement step of the algorithm.

Our first approach is to use a safe region of constant size for each moving object, depending on the velocity of the moving object as well as possibly other, domain specific properties of the moving object. The choice of an optimal size is not trivial since too small safe regions will result in a large number of required position updates due to objects moving out of their safe region. On the other hand, if the safe regions are chosen too large, it is more likely that the object has to be refined. Our approach tries for each object o_i that are currently refined to re-compute the safe region to an object specific size. If the new safe region O_i is valid, i.e., if the result of the RkNN query is still deterministic based on O_i, then the server sends a message to o_i informing o_i to use its new safe region. Any object for which the new safe region is not possible has to continuously stream its position updates to the server until the server is able to set a new valid safe region for o_i.

We note that objects that are very far from the query point may have a large safe region. The reason is that for an object o that is far from the query object, it is likely that there exist objects that can prune o, even if the safe region of o is large. In contrast, objects that are close to the query object are likely necessary to be refined if their safe region is too large. Thus objects far from the query should have a smaller safe region. Therefore, we propose a second strategy where we change the size of the safe region dynamically. In particular, we propose to adjust the size of the safe region of an object o each time it is required to communicate with the server. We differentiate between two cases. First, if o sends its position update

to the server due to moving out of its safe region we try to enlarge the safe region of o. Second, if the exact position of o is polled by the server in the refinement step, we try to successively reduce the size of the safe region of o, until either no more reduction is necessary to answer the RkNN query, or until the size of the safe region of o becomes too small. In the later case, o is given no safe region and has to stream its position updates to the server until it becomes assigned a new safe region.

6. EXPERIMENTAL RESULTS

For the evaluation of the different approaches we used two different data set generators, an artificial data generator that generates mobile objects at random positions in a d dimensional space that move into a uniformly distributed random direction at a constant velocity and the traffic generator *Viewnet* [9] using the road network of Oldenburg, Germany, containing 6105 Nodes and 7035 Edges. The default parameter values are $k = 1$, $|\mathcal{D}| = 1000$. If not stated otherwise, we monitored the RkNNs of 100 query objects simultaneously and report the average results in terms of network traffic, i.e., the number of required exact positions that have to be sent to the server.

In a first experiment, we increased the number of moving objects in the database from 50 moving objects to 1000 moving objects and measured the number of required position updates that have to be sent to the server in a time frame of 30 minutes in order to simultaneously monitor the RkNNs of 100 query objects. The query objects in this experiment are static query points randomly located in the object space. The results of this scalability experiment showed that using the optimal decision criterion and the maximum pruning potential heuristics, the number of required position updates scales sub-linear in the number of moving objects.

In general, the number of position updates increases in our experiments as k increases, since more objects may have to be refined to answer the RkNN query at a given time. The absolute improvement of using the optimal decision criterion and the maximum pruning potential heuristic was constant in k.

We compared the strategies of using rectangles of fixed size with the strategy of dynamically adjusting the size of the approximations. It can be observed that the total number of position updates using dynamic safe regions is only a fraction of the number of position updates using static safe regions. However, in the case of static safe regions, moving objects only need to send their current position and do not need to receive the new safe region. It can be argued that the overhead of receiving the new safe region is negligible, since the communication channel has already been established for sending the current object position, and receiving data is always less expensive than sending.

Finally, we evaluated the impact of the proposed refinement heuristics using a random refinement heuristic as a baseline. The results confirm that using the MinDist heuristic reduces the number of required position updates by a significant factor. However, using the maximum pruning potential heuristic additional reduces the performance by another factor of similar size.

7. RELATED WORK AND CONCLUSIONS

A number of methods have been proposed for supporting RkNN queries for stationary points [7, 15, 11, 1, 14, 2, 8] focussing on specialized pruning techniques based on spatial or metric index structures. However, none of these pruning concepts can be applied to our problem of dynamic data.

An approach for index-based RkNN search on mobile objects has been proposed in [3] using a TPR-Tree [4]. This approach requires knowledge about the future trajectory of the moving objects, i.e., knowledge about the direction and velocity of moving objects. However, the movement of objects in real applications is usually unknown and the motion patterns are constantly changing [12].

In our paper, we do not make any assumption on the movement of objects. Instead, the objects may send updates of their locations to the central server at arbitrary time intervals. For this general scenario, we propose an approach for continuous reverse k-nearest neighbor monitoring in a client-server architecture. We aim at reducing the communication cost between the clients and the server by means of the concept of safe regions. Therefore, we propose a filter and refinement strategy using an effective spatial pruning filter and proposed effective heuristics for the refinement step. In addition we proposed appropriate heuristics for the safe region generation. Our proposed pruning strategies and refinement heuristics empirically outperform state-of-the-art solutions in terms of effectiveness and efficiency.

8. REFERENCES

[1] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proc. SIGMOD*, 2006.

[2] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. 2009.

[3] R. Benetis, C. S. Jensen, G. Karčiauskas, and S. Šaltenis. Nearest and reverse nearest neighbor queries for moving objects. 15(3):229–249, 2006.

[4] S. Šaltenis, S. Jensen, S. Leutenegger, and M. Lopez. Indexing the positions of continuously moving objects. In *Proc. SIGMOD*, 2000.

[5] T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Incremental reverse nearest neighbor ranking in vector spaces. In *Proc. SSTD*, 2009.

[6] T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Boosting spatial pruning: On optimal pruning of mbrs. 2010.

[7] F. Korn and S. Muthukrishnan. Influenced sets based on reverse nearest neighbor queries. In *Proc. SIGMOD*, 2000.

[8] H.-P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler. Reverse k-nearest neighbor search based on aggregate point access methods. 2009.

[9] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. ViEWNet: visual exploration of region-wide traffic networks. In *Proc. ICDE*, 2006.

[10] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. SIGMOD*, 1995.

[11] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun. High dimensional reverse nearest neighbor queries. In *Proc. CIKM*, 2003.

[12] P. Sun and S. Chawla. On local spatial outliers. In *Proc. ICDM*, 2004.

[13] Y. Tao, D. Papadias, and X. Lian. Reverse kNN search in arbitrary dimensionality. In *Proc. VLDB*, 2004.

[14] Y. Tao, M. L. Yiu, and N. Mamoulis. Reverse nearest neighbor search in metric spaces. *IEEE TKDE*, 18(9):1239–1252, 2006.

[15] C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *Proc. ICDE*, 2001.