Constrained Reverse Nearest Neighbor Search on Mobile Objects

Tobias Emrich, Hans-Peter Kriegel, Peer Kröger, Matthias Renz, Andreas Züfle
Institute for Informatics, Ludwig-Maximilians-Universität München
Oettingenstr. 67
80538 München, Germany
{emrich,kriegel,kroeger,renz,zuefle}@dbs.ifi.lmu.de

ABSTRACT

In this paper, we formalize the novel concept of Constrained Reverse k-Nearest Neighbor (CRkNN) search on mobile objects (clients) performed at a central server. The CRkNN query computes for a given query object q the set RkNN(q) of objects having q as one of their k-nearest neighbors, iff the result set exceeds a specific threshold m, i.e. $Card(RkNN(q)) \geq m$. Otherwise, the query reports an empty result. In our setting, the positions of the query object and database objects are approximated by minimal bounding rectangles that depend on the last reported location of the object, as well as on the time that has been passed since the object reported its recent exact location. We propose an approach that minimizes the amount of communication between clients and central server by using the approximation of the positions to identify true hits and true drops. We present a multi-step filter/refinement framework that uses a novel refinement heuristic to minimize the number of objects that are required to provide their exact location. Our solution does not assume any preprocessing steps which makes it applicable for dynamic environments where updates of the database frequently occur. Experiments show that our approach considerably reduces the communication load compared to existing approaches designed for traditional reverse nearest neighbor search in static data.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

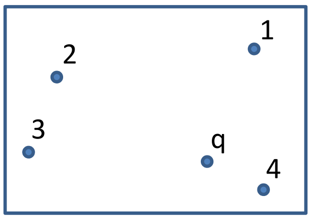
RkNN, Mobile Objects, Spatial Pruning, Query Processing

1. INTRODUCTION

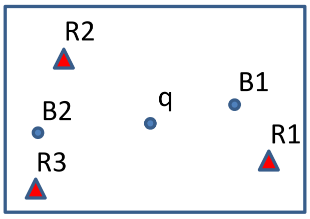
While the reverse k-nearest neighbor (RkNN) search problem, i.e. finding all objects in a database that have a given query q among their corresponding k-nearest neighbors, has been studied extensively in the past years, considerably less work has been done so far to support RkNN queries on mobile objects that may not be indexed by a point access method. The prevalence of inexpensive and very small Global-Positioning-Devices (GPS) gives rise to new spatio-temporal database applications. With these advances it is e.g. possible to track pupils by equipping them with such a GPS-device or even animals by attaching a GPS-device to their ear or under their skin. However, such very small GPS-devices posses only a very limited power supply, and the replacement of a power supply of such a GPS-device can be very expensive. As an example, think of a wildlife sanctuary, where many animals are equipped with small GPS-devices to observe and protect them. The GPS signal of the animals can be used to alert the ranger, when a very rare species is in immediate danger of being attacked by a predator, such as a tiger. The ranger may then intervene by chasing off the tiger. However, the act of attaching a GPS-device to an animal or replacing its power source is very stressful and dangerous for both the animal and the rangers. Usually, the animal has to be tranquilized in order to allow the veterinary a safe approach. Thus, the power of such miniature GPS-devices is a very precious resource. In order to preserve this source, the miniature GPS-device should only submit its exact position if necessary. As a consequence, in such an application scenario, query processing must account for the fact that any position poll necessary to answer the query is very costly, or in other words, each single position poll that is saved is valuable.

Many applications also only require to know if the number of RkNNs of a query object q exceeds a given threshold m. If q has less RkNNs than m, no results need to be reported. Only if the number of RkNNs of the query is at least m, all RkNNs need to be known. This type of query is called Constrained Reverse k-Nearest Neighbor (CRkNN) query. As an example application for a CRkNN query, consider lions in a wildlife sanctuary. Since lions generally hunt in packs, it is safe to assume that the query animal (or a tourist) q is not in danger if less than $m = 3$ lions have q as their nearest prey. If however the number of reverse nearest lions of the query exceeds the threshold $m = 3$, then the lions need to be chased off (or the tourist has to be warned).

To track and observe large numbers of continuously moving objects, their last submitted positions are stored in a database. The conventional assumption, that data remains constant unless it is explicitly modified, no longer holds when considering mobile objects. In our examples above, the position of an animal may be given by an exact location at the time slot the animal submitted its current position. After that, its position is conservatively approximated by a minimal bounding box that contains all possible positions the tiger may have reached since its last location update and that usually grows over time.



(a) Mono-chromatic case.



(b) Bi-chromatic case.

Figure 1: R1NN examples.

We consider the computation of CRkNN queries in this setting for both the mono-chromatic as well as the bi-chromatic case. The difference of both cases is that in the mono-chromatic case we have only one set of objects from which the query and the results are drawn, whereas in the bi-chromatic case, we have two sets, a set of potential query objects (that are not considered as results when a query is processed) and a set from which the results are drawn. Figure 1 illustrates the concept of CRkNN queries in both cases. The mono-chromatic CR1NN query (cf. Figure 1(a)) for point q returns point 1 and 4 if $m \in \{1, 2\}$ or nothing if $m \geq 3$. Points 2 and 3 are not returned for any choice of m, because they do not find q as their 1-nearest neighbor. In the bi-chromatic case, two object sets $\mathcal{D}_{red}$ (red objects) and $\mathcal{D}_{blue}$ (blue objects) are considered. The bi-chromatic CRkNN query returns all elements of $\mathcal{D}_{red}$ that have the query point as on of their k-nearest neighbors if all other red objects are ignored. Figure 1(b) shows the bi-chromatic CR1NN query for a set of lions (red objects $R1$, $R2$, and $R3$) and a set of potential prey (blue objects $B1$, $B2$, and q). The query object q (from the blue object set) could be a young elephant that is not yet able to defend itself. In this example, the bi-chromatic CR1NN query yields no results for any value of m, because each lion observes another animal as nearest neighbor.

A straightforward solution for computing CRkNN queries is to check for each point, whether is has a given query point as one of its k-nearest neighbors. If more than m objects do, the set is outputted as result, otherwise the result is empty. However, this approach lacks on a computational point of view when the number of points is large. Even more important in this setting is that each client is required to send its exact location at least once and, thus, the drain of the clients' power sources is very large.

In this paper we present a novel framework for CRkNN search on mobile objects that minimizes the amount of network traffic. Our framework is applicable to both *mono-chromatic* and *bi-chromatic* CRkNN queries. It extends the idea of using Voronoi hyperplanes [10] to prune the search space of possible results and (most importantly) allows pruning based on object approximations in order to save position polls. In particular, in the case of extended query objects and database objects, the pruning regions are no longer defined by simple Voronoi hyperplanes, but become rather complex regions. We propose new refinement heuristics that vastly reduces the required amount of network traffic compared to traditional refinement heuristics used for RkNN search. In general, the contributions of this paper can be summarized as follows:

- We formalize the problem of Constrained Reverse k-Nearest Neighbor (CRkNN) queries on mobile objects in a client/server scenario which is prevalent in several important applications.
- We propose a multi-step query processing algorithm for mono-chromatic and bi-chromatic CRkNN queries that allows the pruning of candidates based on approximations of locations of objects in the filter step and, thus, considerably decreases the communication between clients and server.
- We prove the correctness of our filter step, i.e., that it returns a superset of the true hits and does not produce false drops.
- We propose a heuristic for deciding which objects are required to submit their exact location in the refinement step, that performs significantly better in this scenario than state-of-the-art refinement heuristics used for traditional RkNN search in terms of communication costs.
- We give analytical and experimental evidence that our framework significantly reduces the amount of network traffic required for CRkNN-queries on moving objects which is important to minimize waste of the most precious resource, the battery of the miniature clients.

The remainder is organized as follows. We review related work in Section 2. The problem of CRkNN query processing in spatio-temporal data implementing a client-server setting is formalized in Section 3. Section 4 provides the details of our novel multi-step CRkNN query processor and Section 5 describes the complete query algorithm. A comparative experimental evaluation is presented in Section 6. Section 7 concludes the paper.

2. RELATED WORK

Existing approaches for RkNN search on static data can be classified according to the pruning approaches used.

Self-pruning approaches are usually designed on top of a hierarchical index structure. They usually try to estimate the kNN distance of each index entry E, i.e. E can be a database object or an intermediate index node. If the kNN distance of E is smaller than the distance of E to the query q then E can be pruned. Several approaches use an exact estimation by simply pre-computing kNN distances. The RNN-Tree [6] is an R-Tree-based index that pre-computes for each object p the distance to its 1NN, i.e. $nndist_1(p)$. The objects are not stored in the index itself. Rather, for each object p, the RNN-Tree manages a sphere with radius $nndist_1(p)$, i.e. the data nodes of the tree contain spheres around objects. The RdNN-Tree [12] extends the RNN-Tree by storing the objects of the database itself in an R-Tree rather than circles around them. For each object p, the distance to p's 1NN, i.e. $nndist_1(p)$, is aggregated. For each leaf entry E, the maximum of the 1NN distances of all objects in E is aggregated. An inner node of the RdNN-Tree again aggregates the maximum 1NN distances of all its child nodes. In addition, the RdNN-Tree, can be extended to metric spaces (e.g. by applying an M-Tree instead of an R-Tree). However, both approaches are limited to a fixed value of k. To overcome this problem, the MRkNNCoP-Tree [2] has been proposed which is conceptually similar to the RdNN-Tree but stores a conservative and progressive approximation for all kNN distances of any data object rather than the exact kNN distance for one fixed k. The only limitation is that k is constrained by a parameter k_{max} specifying the maximal value of k that can be supported. For RkNN queries with $k > k_{max}$, the MRkNNCoP-Tree cannot be applied. The conservative and progressive approximations of any index node are propagated to the parent nodes. Using these approximations, the MRkNNCoP-Tree can identify a candidate set, true hits, and true drops. For each object in the candidate set, a kNN query is launched for refinement. A variant of the MRkNNCoP-Tree is proposed in [1, 3] that achieves a further runtime improvement and gets rid of the k_{max} bound for the value of k to the cost of generating only approximative results. A different approach is proposed in [11] where a method for RkNN search in metric spaces that is tailored to the M-Tree is presented. The authors derive several rules from the M-Tree structure that can be used to estimate the kNN distance of an index entry. For each database object o that is contained in a non-pruned leaf node of the M-Tree a kNN query is required for refinement. The above self-pruning approaches are not applicable for a database containing mobile objects, because RkNN-distances (or their approximations) have to be materialized and frequently updated due the constant change of position information of objects.

Mutual-pruning approaches are usually designed for the Euclidean space only and use other objects to prune a given index entry E. For that purpose, they use special geometric properties of the Euclidean space, typically the concept of Voronoi cells. The basic idea is that given the Voronoi-cell around the query object q, each object or index node E can be pruned if E is beyond a Voronoi plane (for $k = 1$). In [8] a two-way filter approach for supporting R1NN queries based on this idea is proposed that provides approximate solutions. In [10] the first approach for RkNN search was presented, that can handle arbitrary values of k. The method uses any hierarchical index structure to compute a nearest neighbor ranking of the query object q. From this ranking, a Voronoi cell around q is iteratively constructed. Objects that are beyond k Voronoi planes w.r.t. q can be pruned and need not to be considered for Voronoi construction. The remaining objects must be refined. In [9], a different approach for R1NN search in a 2D data set is presented. It is based on a partition of the data space into six equi-sized units where the border lines of the units are cut at the query object q. The 1NN of q in each unit is determined and all these neighbors are merged together to generate a candidate set. This considerably reduces the cost for the nearest-neighbor queries. Each candidate is refined by computing its nearest neighbor.

Hybrid approaches combine the strengths of both pruning concepts to get the "best of both worlds". In [4], a framework is presented to obtain conservative and progressive distance approximations between a query point and arbitrarily approximated regions, such as MBRs of an R*-Tree in the Euclidian case or "circles" of an M-Tree in the general metric case. An approximated object E may prune itself, if the minimal distance to the query object is already greater than the maximal distance to E itself. Similarly, one approximated object E may prune another approximated object E', if the minimal distance of E' to q is already greater than the maximal distance of E to q. A specialization of this approach to Euclidean data is proposed in [7] exploiting geometric properties to achieve a higher pruning power. The hybrid approaches have shown to yield a better pruning but all of these approaches assume that the query is given as a single point. To the best of our knowledge, no work has been done to tackle the problem where the query object is given by an approximated region only.

3. PROBLEM FORMALIZATION

3.1 Client-Server Scenario

In the following, we assume that $\mathcal{D}$ is a database of n objects (clients) moving continuously within a 2-dimensional Euclidean space and $dist$ is the Euclidean distance[1] on the objects in $\mathcal{D}$. 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.

[1]The concepts described here can also be extended to any L_p-norm.

At server side the position of each object o is approximated by a two-dimensional axis aligned rectangle $o.mbr$ that minimally bounds the possible positions of o. The objects send their exact positions to the server only if necessary. The exact position of an object o will not be updated at server side as long as

- the exact object position $o.pos$ is within the region $o.mbr$.
- the query can be answered based on the information of the object positions that is currently available at the server.

Here, we will be interested in performing queries among the clients at the server.

3.2 CRkNN Query

We will first review the two different types of traditional reverse k-nearest neighbor queries, the mono-chromatic and the bi-chromatic variant. Then, we will derive the novel constrained reverse k-nearest neighbor query.

3.2.1 Mono-Chromatic RkNN Query

In a mono-chromatic setting, we have only one set of objects $\mathcal{D}$. The set of *k-nearest neighbors* (kNNs) of an object q is the smallest set $kNN(q) \subseteq \mathcal{D}$ that contains at least k objects such that $\forall o \in kNN(q), \forall \hat{o} \in \mathcal{D} - kNN(q) : dist(q, o) < dist(q, \hat{o})$. The object $p \in kNN(q)$ with the highest distance to q is called the *k-nearest neighbor* (kNN) of q. The distance $dist(q, p)$ is called kNN distance of q.

The set of *reverse k-nearest neighbors* (RkNNs) of an object q is defined as

$$RkNN(q) = \{p \in \mathcal{D} \mid q \in kNN(p)\}.$$

3.2.2 Bi-Chromatic RkNN Query

For the bi-chromatic RkNN query we assume that the database is divided into two disjunctive sets of objects, $\mathcal{D}_{red} \subseteq \mathcal{D}$ and $\mathcal{D}_{blue} \subseteq \mathcal{D}$, where $\mathcal{D}_{red} \cup \mathcal{D}_{blue} = \mathcal{D}$. Here, the set of *$k$-nearest neighbors* of an object $q \in \mathcal{D}_{blue}$ is the smallest set $kNN(q) \subseteq \mathcal{D}_{red}$ that contains at least k objects such that $\forall o \in kNN(q), \forall \hat{o} \in \mathcal{D}_{red} - kNN(q) : dist(q, o) < dist(q, \hat{o})$.

Let us note that the mono-chromatic query is a special case of the bi-chromatic query. The mono-chromatic query is equal to the bi-chromatic query if we assume that $\mathcal{D}_{red} = \mathcal{D}_{blue}$. In addition, the two distinct sets can be exchanged with each other in the above definition.

The set of RkNNs of an object q is defined as above taking the bi-chromatic version of the kNNs into account.

3.2.3 Constrained RkNN Query

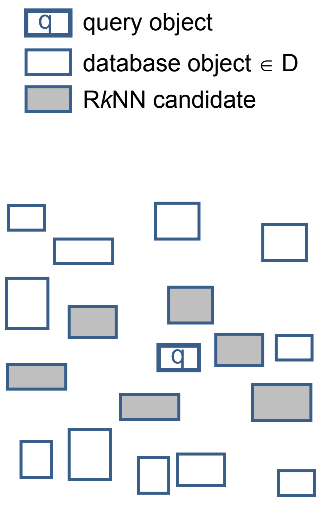
The problem of a *constrained RkNN* (CRkNN) query is to report for a given query object q the result of a RkNN query if and only if the result size exceeds a specific threshold m, formally

$$CRkNN(q) = \begin{cases} RkNN(q) & \text{, if } Card(RkNN(q)) \geq m \\ \emptyset & \text{, else.} \end{cases}$$

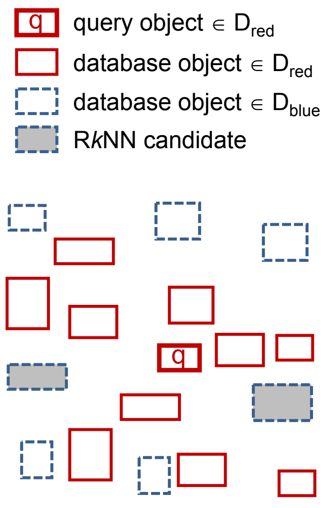
Note that the case where $m = 1$ corresponds to a traditional, non-constrained RkNN query.

3.3 General Idea for Answering CRkNN Queries

The main objective of our approach is to keep the communication cost between the clients and the sever as low as possible while answering CRkNN queries for a given query object q, and specified values for k and m that may vary from query to query. For this



(a) $Card(RkNN(q)) = 6$ (monochromatic).



(b) $Card(RkNN(q)) = 2$ (bichromatic).

Figure 2: R1NN selectivity estimation based on position approximations.

reason, we try to avoid unnecessary position updates at the server side. The communication cost of each position update is assumed to be very expensive, since it composes the cost required to build up the communication channel between the server and the client, and the cost for transferring the position information from the client to the server. The problem is that we might update object positions when issuing CRkNN queries in order to get exact results. However, the CRkNN query offers a potential for a lazy update strategy. Whenever a CRkNN query is issued, first, we estimate conservatively the selectivity of the query only based on the information available at the server, i.e. in consideration of the object approximations. An example illustrating this estimation task for the mono-chromatic and the bi-chromatic case is depicted in Figure 2. Detailed strategies for estimating the selectivity, i.e. finding the RkNN candidates, will be discussed in the next section. Only if the estimated size of the result set exceeds a specified threshold m, then the server requests exact position updates from the clients. In our example, m must be smaller than 6 in the mono-chromatic case (cf. Figure 2(a)) and smaller than 2 in the bi-chromatic case (cf. Figure 2(b)). Thereby, only those position updates are requested that are necessary to compute the exact result. The determination of those objects (including the query object) that definitely have to transmit their exact positions to the server is challenging. In summary, there are two problems to be solved:

- First, we have to compute a conservative but accurate estimation of the query selectivity.
- Secondly, we have to find a possibly small set of objects whose refined positions suffice to compute the correct query result.

Obviously, the higher the accuracy of the position information of the query object and the database objects at sever are, the better is the query selectivity estimation. However, since our main focus is to keep the position updates as low as possible, we first estimate the query selectivity without any position update. For the query selectivity estimation we have to compute RkNN candidates in consideration of the minimal bounding rectangles associated with the query object and the database objects.

4. CRkNN QUERY PROCESSING

In the following sections, we present our framework for answering CRkNN queries on objects initially approximated by minimal bounding rectangles. For a clear presentation, we focus on the mono-chromatic case but all concepts can be directly applied to the bi-chromatic case.

4.1 Pruning Strategies

First, we will introduce pruning strategies which are applicable for extended query objects and are used in a filter step to identify potential candidates and, thus, to estimate the selectivity of the query. Thereby, the main focus is to cut down the number of CRkNN candidates to achieve a good conservative estimation of the selectivity of an RkNN query and to identify a possibly small number of objects for which the server has to request the exact positions.

4.1.1 Basic Pruning

Let us first construct the simple case where we consider the basic pruning strategy based on exact object representations. Let q be an exactly represented query object, e' be an exactly represented database object, and E be a rectangular approximated object as illustrated in Figure 3. Now, we want to identify whether E can be pruned according to a R1NN(q) query. E can be pruned (according to e'), if there exists no point $e \in E$ which could be R1NN of q. A straight forward approach would be to use the pruning criterion based on distance approximations called *Min-/MaxDist pruning* as proposed in [4], where E can be pruned if the minimal distance $MinDist(q, E)$ between q and E is larger than the maximal distance $MaxDist(e', E)$ between e' and E. However this strategy would not work in our case, since $MinDist(q, E) < MaxDist(e', E)$. Rather, we adopt the geometrical pruning approach as proposed in the TPL approach [10]. The main idea is to compute the Voronoi plane (Voronoi line in the two-dimensional

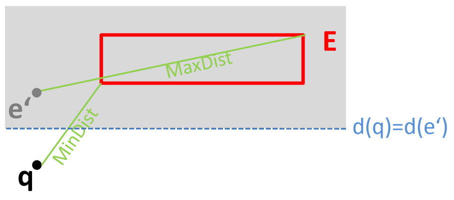


Figure 3: Pruning according to TPL.

space) between q and e', containing all points having equal distance to q and to e'. This line which is depicted as dashed line in our example divides the Euclidean space into two half-spaces. All points in the half-space containing e' can be pruned because they are closer to e' than to q. This means, that if an object or an object approximation, e.g. object E in our example, lies completely within this area, it can be pruned. However this approach requires (at least two) exact object representations in order to build such a Voronoi line. Thus, it is not applicable to prune objects based on approximative object representations. Rather, we need more advanced concepts which can be applied to rectangles.

4.1.2 *Pruning Based on Extended Regions*

Now we consider our initial setting assuming that all objects (including the query object) are approximated by axis-aligned minimal bounding rectangles. Again, the *Min-/MaxDist pruning* as proposed in [4] can be easily applied to the rectangles, but would not be a good choice due to similar reasons as for the simple case described above. A geometrical pruning approach would be more selective, but the question at issue is, how can this be done in case of rectangles, e.g. for the rectangles Q.mbr and E'.mbr as illustrated in Figure 4. A naive solution would be to materialize all possible Voronoi lines defined by all pairs $(q, e') : q \in Q, e' \in E'$. In order to guarantee no false dismissals, the corresponding pruning area have to be built by intersecting all half-spaces opposite to Q. Obviously, the materialization of the Voronoi lines of all possible pairs of points of Q and E' is not applicable. Rather, we only need to focus on the pairs of points which are responsible for the border of the pruning area.

In fact, the pruning space defined by a Voronoi line between a query point q and a database point e' has the property that all points p in this space are closer to e' than to q, i.e. $dist(p, q) \geq dist(p, e')$ (cf. Figure 2). Now, let us again consider the object approximations Q and E'. The pruning space according to Q and E' is represented by all points p that are definitely closer to each point in E' than to each point in Q, i.e.

$$\forall q \in Q : \forall e' \in E' : dist(p, q) \geq dist(p, e'). \tag{1}$$

In order get a conservative pruning space we can replace the distance p and Q by the minimum distance $MinDist(p, Q)$ and replace the distance between p and E' by the maximum distance $MaxDist(p, E')$. Consequently, we can define the pruning area by means of the following lemma:

Lemma 1. *Let Q be a rectangle containing the query object q and E' be a rectangle containing a database object. Then, all points p for which the following equation holds cannot be the R1NN(q) and, thus, can be pruned:*

$$MinDist(p, Q) \geq MaxDist(p, E'). \tag{2}$$

PROOF. Though the pruning spaces defined by the two equations 1 and 2 are in fact identical, for the proof of the above lemma it suffices to show that each point p in the pruning space defined by Equation 2 is within the pruning space defined by Equation 1. Let p be any point, for which Equation 2 holds. Then, each point q in Q has a larger distance to p than $MinDist(p, Q)$, i.e.

$$\forall q \in Q : dist(p, q) \geq MinDist(p, Q).$$

Furthermore, each point e' in E' has a smaller distance to p than $MaxDist(p, E')$, i.e.

$$\forall e' \in E : MaxDist(p, E) \geq dist(p, e').$$

Consequently, with Equation 2 we get the following equation:

$$\begin{aligned}\forall q \in Q : dist(p, q) &\geq MinDist(p, Q)\\ &\geq MaxDist(p, E') \geq dist(p, e'),\end{aligned}$$

which is equal to Equation 1.

Let us note, that the pruning strategy based on Lemma 1 can be easily applied for RkNN queries with $k > 1$ by considering k pruning areas.

Based on Lemma 1 we can define the border of the half space representing the pruning area associated with the rectangles Q and E'. This area composes all points that are definitely closer to each point in E' than each point in Q. The border of this half-space is defined by all point p for which the following equation holds:

$$MinDist(p, Q) = MaxDist(p, E'). \tag{3}$$

In the remainder we will show how the pruning area can be constructed in consideration of the topology of the corresponding rectangles. The computation of the border of the pruning area defined by two rectangles in a two-dimensional space can be partially reduced to the basic case. In fact, we have to consider 326 cases, i.e. the space can be decomposed into $4 \cdot 9 = 36$ partitions having certain topological properties w.r.t. to both rectangles, as depicted in Figure 4. The query rectangle Q decomposes the space into 9 partitions along the rectangle margins, 3 partitions per dimension. For each partition P we can determine a point or margin of the rectangle Q that represents or contains the nearest point of Q to any point in P. For example, consider the north-east partition P^Q_{NE} built by Q. The upper-right corner of Q is the nearest point in Q for all points in P^Q_{NE}. Furthermore, the lower-right corner of Q is the nearest point in Q for all points in the south-east partition P^Q_{SE}. Similar representatives can be found for the north-west and south-west partitions. The nearest point in Q to any point in the east partition P^Q_E is not a unique point and corresponds to the nearest point on the right rectangle margin. This also holds for the south, west and north partition P^Q_S, P^Q_W and P^Q_N, respectively. Obviously, the partition which is defined by the rectangle Q itself composes all points having a distance of zero to Q.

The database rectangle E' decomposes the space into 4 additional partitions, 2 partitions per dimension. Here, the center of E' is used to define the partitioning which leads to the partitions $P^{E'}_{NE}$, $P^{E'}_{SE}$, $P^{E'}_{SW}$ and $P^{E'}_{NW}$. In contrast to the partitions defined by Q, here we are interested in partitions having the farthest point in E' in common. For example, all points in the south-west partition $P^{E'}_{SW}$ built by E' have the upper right point of E', i.e. the opposite corner of E' w.r.t. $P^{E'}_{SW}$, as their farthest point in E'. Analogous considerations hold for the other four partitions.

Let us now consider the combined partitioning defined by the two rectangles Q and E' in order to determine the margin of the pruning area. The margin of the pruning area is composed of parts defined for each partition. In our example illustrated in Figure 4, the margin of the pruning area within partition $P^Q_{N,E} \cup P^{E'}_{S,W}$ is defined by the upper-right corner of Q and the upper-right corner

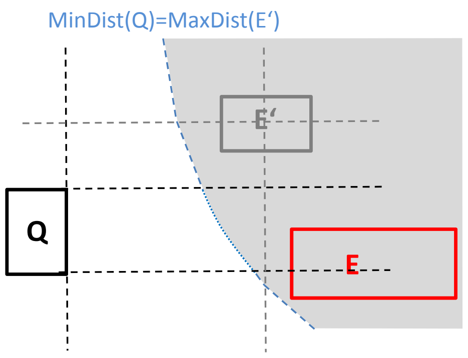


Figure 4: Geometrical pruning on MBRs

of E'. In contrast the partition $P^Q_{N,E} \cup P^{E'}_{N,W}$ defines the margin of the pruning area by means of the upper-right corner of Q and the lower-right corner of E'. Each of the above margin parts can be represented by a single line segment. We achieve a more complex structure when considering the margin part within partition $P^Q_E \cup P^{E'}_{S,W}$. Since, this partition has not a common nearest point in Q, i.e. the nearest point in Q depends on the location of the margin point, the geometry of the margin is not linear anymore. The resulting structure of the margin of the pruning area is now rather complex, persisting of several line segments and even non linear parts (see the solid blue line in Figure 4). The materialization of the pruning area in order to test whether an object E lies completely within the pruning area is not trivial and could be computationally expensive.

4.1.3 Advanced Geometrical Pruning

Since the geometrical pruning based on the pruning area as described above is effective but the materialization of the pruning area is expensive, we have to find a method to check whether a point or rectangle is completely within the pruning area without requiring to materialize the pruning area. Note that in figure 4 it is sufficient to only test whether the corners of E can be pruned, in order to prune E as a whole. In the following, we will show that this approach is sufficient in general.

First, we utilize the fact that the pruning area always have a convex structure which is shown by the following two lemmas:

Lemma 2. *Let $P_1, \ldots, P_n \subset \mathbb{R}^2$ be convex areas. Then the intersection $P = P_1 \bigcap \ldots \bigcap P_n$ is also convex.*

PROOF. Let $p_1, p_2 \in P$, then p_1 and p_2 are within every set P_j. Since the sets P_j are convex, the whole segment $[p_1, p_2]$ lies within each set P_j. Since this segment lies within each set P_j, it also is part of the intersection of these sets, i.e. $[p_1, p_2] \subset P$. Thus the intersection P of finite number of convex sets is also convex.

Lemma 3. *The pruning area defined by a query rectangle Q and a database rectangle E' is convex.*

PROOF. The pruning area defined by any two points $q \in Q$, $e' \in E'$ is a half-plane, which is a convex set. The resulting area when intersecting the pruning areas of all pairs (p, e') is again convex as a consequence of lemma 2.

Now, we can use the convexity property of a pruning area in order to check whether a point or rectangle is completely within the pruning area in an efficient way using the following lemma.

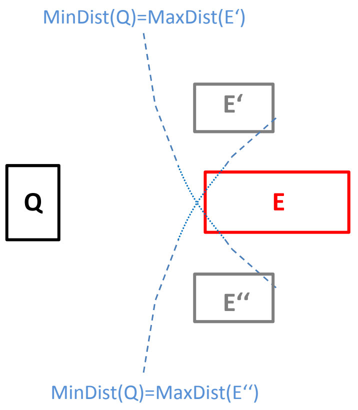


Figure 5: Partial pruning

Lemma 4. *In order to prune an Object E according to a query region Q and a database object E' it suffices to show that all corners v of E can be pruned according to the following criterion:*

$$MaxDist(v, E') \leq MinDist(v, Q).$$

PROOF. According to lemma 3 Q and E' form a convex pruning area. Therefore if E is not completely within the pruning area, at least one corner v of E must be outside the pruning area. Thus the following inequality $MinDist(v_i, Q) < MaxDist(v_i, E')$ holds.

As a consequence of Lemma 4 the test if an rectangular object E lies within the R1NN pruning area formed by a rectangular query region Q and a rectangular database object E' can be reduced to the Min-/MaxDist pruning test of the corners of E'. This pruning criterion is optimal as it is based on the exact pruning area defined by Q and E'. In particular, it is different from the *Min-/MaxDist pruning* as proposed in [4] which applies the pruning test only for the complete rectangles rather than to the corner points and, thus, is significantly less selective. This technique is extendible to the RkNN-problem where an object E can be pruned if it lies completely within k pruning areas (defined by Q and k database objects).

4.1.4 Partial Pruning

The objective of the partial pruning is to increase the pruning power of the geometrical pruning strategy. The idea of this approach is that objects can be pruned by means of pruning regions, even if the objects are only partially covered by the pruning regions [5]. This can be applied, if some pruning regions complement one another. An example of an R1NN-based pruning is illustrated in Figure 5. Here, two objects E' and E'' build two pruning regions together with the query object Q. The pruning regions partially intersect the rectangle of another object E. Though, the rectangle of E does not completely lie within any pruning region, each point in E does. Consequently, each point in E can be pruned by at least one pruning area.

4.1.5 Progressive Pruning

Up to now, we presented conservative pruning techniques in order to prune candidates that can be excluded from the query result. Now, we present additional pruning strategies called *progressive geometrical pruning* that find true hits without any refinement of candidates. Thereby, we apply similar geometrical properties as used for the conservative geometrical pruning. For a given query

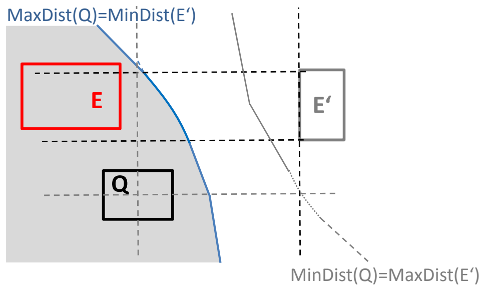


Figure 6: Progressive pruning area.

rectangle Q and a database rectangle E' the progressive pruning area defines all points that are definitely closer to all points in Q than to all points in E'. An example is illustrated in Figure 6. In contrast to the margin of the conservative pruning area, the margin of the progressive pruning area is defined by all points p for which the following equation holds:

$$MaxDist(p, Q) = MinDist(p, E'). \tag{4}$$

Hence, if a point or rectangle completely lies within this area, it can be concluded that it is closer to any point in E' than to Q. For a R1NN query, if a point or rectangle completely lies within the progressive pruning area of each database object (except itself), it can be immediately reported as R1NN(q) result.

Like the conservative pruning area the progressive pruning area is convex which can be used to identify true hits efficiently.

Lemma 5. *An object E can be identified as true hit according to a query object Q if all corners v of E lie within the progressive pruning areas of all database objects according to the following criterion:*

$$\forall E' \in \mathcal{D}, E \neq E' : MaxDist(v, Q) \leq MinDist(v, E').$$

The above lemma can easily be proved analogously to Lemma 4.

Again, this technique is extendible to the RkNN-problem where an object E can be pruned if it lies completely within $Card(\mathcal{D}) - k - 1$ progressive pruning areas (defined by Q and $Card(\mathcal{D}) - k - 1$ database objects).

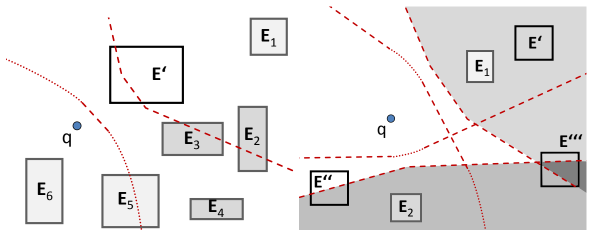
4.2 Refinement Strategies

After applying our pruning strategy, there may still candidates left. For these candidates, the decision whether they are part of the result, cannot be made by using only information stored on the server. In this case, one of the objects of the database has to be refined, i.e. one of the objects is required to transmit its exact location to the server. In this section we propose strategies that aim to minimize the total number of refinements. This is important since we follow our overall goal in minimizing the number of messages sent between clients and server.

4.2.1 *Minimal Mindist Heuristics*

The idea of refining the object of the database, that has the smallest Mindist to the query object q has been proposed in [10]. This heuristic has also been adapted in [4]. The general idea is, that objects closest to q are likely to be true hits and also generate pruning areas close to q that potentially prune other objects that are more far apart. Thus, for each object approximation E, we compute the minimal distance $minDist(E, q)$ to q, and refine object

$$argMin_{E \in DB}(minDist(E, q)).$$



(a) Mutual Pruning. (b) Self Pruning.

Figure 7: Different pruning potentials.

4.2.2 *Maximum Pruning Potential Heuristics*

The key of this idea is to pick the object that has the most influence on all other candidates as next refinement. In particular, that object for which the likelihood, that due to its refinement, other candidates can be pruned, is maximized should be chosen.

To evaluate this influence, each object E is associated a pruning potential. The pruning potential of an object is composed of two components, the Mutual-Pruning Potential, $MPP(E)$, and the Self-Pruning Potential, $SPP(E)$. The Mutual-Pruning Potential of an object E estimates the number of candidates that can possibly be pruned by the exact position e of E, but cannot be pruned by means of the approximation of E only. An example of this estimation is given in Figure 7(a). In this example, we want to determine the Mutual-Pruning Potential of E', $MPP(E')$. Object E_1 does not affect $MPP(E')$, because E_1 can already be pruned based on the approximation of E'. E_6 does not contribute to $MPP(E')$ either, because regardless of the exact position of $e' \in E'$, E_6 cannot be pruned by e'. The reason for this is that E_6 cannot be pruned by any line that is contained in the region between the conservative and the progressive approximation of the Voronoi lines between q and any $e' \in E'$. The same applies for E_5. Even in the best case, where $e' \in E'$ is very close to q, E_5 cannot be completely pruned by e'. Let us note that it is possible that the Voronoi line between q and e' intersects E_5, and by means of partial pruning, the prune count of E_5 can still be increased in combination with Voronoi lines of other objects. Thus E_5 increments $MPP(E)$ by one. For objects E_2, E_3 and E_4 there exist possible positions $e' \in E'$, for which they can be completely pruned. Thus, the Mutual-Pruning Potential of E' is incremented by one for each of these objects and finally we get $MPP(E') = 4$. In the bi-chromatic case, the following additional constraints apply:

- The object E for which the Mutual-Pruning Potential is computed must be a member of set $\mathcal{D}_{blue}$ (the *blue* objects), because objects of set $\mathcal{D}_{red}$ (the *red* objects) do not interfere with other objects, and thus cannot prune other candidates.

- The objects (E_1,...,E_6 in the example) that increase the Mutual-Pruning Potential of E must be *red*, since the result of a bi-chromatic CRkNN query may only contain *red* objects. Therefore only *red* objects can be candidates.

Thus, *red* objects do not have a Mutual-Pruning Potential and *blue* objects ignore other *blue* objects in the computation of their Mutual-Pruning Potential.

The Self-Pruning Potential of an object E estimates the number of objects that can possibly prune the exact position of $e \in E$, but cannot prune the approximation E. An example for this situation is depicted in Figure 7(b). Here, we want to determine the Self-Pruning Potential $SPP(E')$ of E'. Object E_1 does not increase the Self-Pruning Potential of E', because the approximation of E' can already be pruned completely by E_1. E_2 does not increase the Self-Pruning Potential of E' either, because regardless of the exact position of $e' \in E'$, e' cannot be pruned. Thus, $SPP(E') = 0$. Now, consider the Self-Pruning Potential of E''. Although E_2 cannot prune the approximation E'', E_2 may be able to prune $e'' \in E''$ if it falls into the pruning region of E_2, denoted by the shaded area. Therefore, E_2 increases the $SPP(E'')$ by one. The situation of E''' is even more interesting, because here, both E_1 and E_2 contribute to the Self-Pruning Potential of E'''. Again, in the bichromatic case additional constraints apply including:

- The object E for which the Self-Pruning Potential is computed must be a member of set $\mathcal{D}_{red}$ (the set of red objects), because the result of a bichromatic (C)RkNN query may only contain red objects, and thus, only red objects may be candidates.
- The objects (E_1 and E_2 in the example) that increase the Self-Pruning Potential of candidates must be $blue$, because only members of the blue set can prune other objects.

Thus, $blue$ objects do not have a Self-Pruning Potential, and red objects ignore other red objects in the computation of their Self-Pruning Potential.

The object with the highest sum of Mutual-Pruning Potential and Self-Pruning Potential

$$argMax_{E \in DB}(MPP(E) + SPP(E))$$

is chosen to be refined next.

As seen in the examples, in order to compute $MPP(E)$ and $SPP(E)$ for an object E, we need to find objects that are not completely contained in either the pruning region or in the true-hit region of other objects. Due to the convexity of both the true-hit and the pruning region of an object E, we can do this efficiently by testing corners of MBRs only (see above).

5. ALGORITHM

In this section we present our algorithm for processing CRkNN queries on approximated objects. An abstract outline of the algorithm is shown in Algorithm 1. In the first step, the algorithm applies pruning based on extended regions utilizing the information that is available on the server. If the number of candidates returned in this step is less than m the algorithm terminates with the empty set as result, since the number of results can not possibly exceed m. Otherwise, if the number of candidates is at least m, then the query object q is refined, and thus required to send its exact position to the server. Then the candidate and result lists are updated according to the new information about the exact position of q using the various pruning strategies described above. This is repeated while there are candidates left to be refined. If at any time, the possible number of results, i.e. the number of current hits ($result$) plus the number of remaining candidates $cand$ becomes less than m, the algorithm terminates again with the empty set as result. Once no more candidates are left, the results are returned if their number exceeds m.

Algorithm 1 CRcNN query processing.

```
CRkNN(query $q$, $k$, $m$)
  LIST $cand$ = filter(query, k, m);
  LIST $result$ = $\emptyset$;
  **if** ($Card(cand) < m$)
    **return** $result$;
  **else**
    Refine $q$;
    Update $cand$;
    Update $result$;
    **while**($Card(cand) + Card(result) > m$
          && $Card(cand) > 0$)
      Apply Refinement Strategy;
      Update $cand$;
      Update $result$;
    **if** ($Card(result) > m$)
      **return** $result$;
    **else**
      **return** $\emptyset$;
```

6. EXPERIMENTAL EVALUATION

In the following, we outline the experimental evaluation. Since all proposed techniques aim at reducing the number of refined objects, the number of refinements is the main variable to be measured. In this regard we will show:

- A comparison of the two proposed filter approaches *Min-/MaxDist-Pruning (MMP)* (cf. Section 4.1.1) and geometrical pruning called *Corner-Based-Pruning (CBP)* (cf. Section 4.1.3).
- A comparison of the two proposed refinement strategies *Minimal Mindist-Heuristics (MMH) and Maximum Pruning Potential Heuristics (MPPH).*
- The impact of the constraint parameter m.

The experiments will be performed under various different parameter settings as well as on different datasets and in both monochromatic and bichromatic environment. Default values for parameters are $k = 5$, $m = 1$, $Card(\mathcal{D}) = 10k$, $Card(\mathcal{D}_{red}) = 10k$ and $Card(\mathcal{D}_{blue}) = 10k$.

6.1 Datasets and Parameters

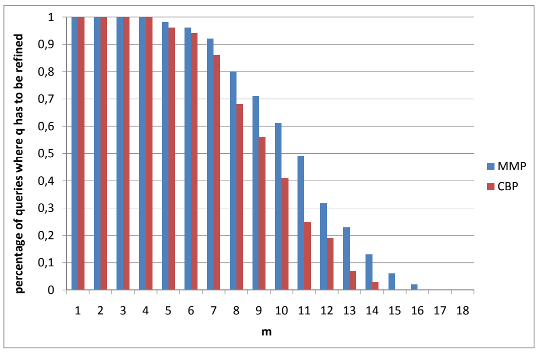
For the evaluation of the different approaches we used two different 2D point datasets:

- Synthetic dataset with uniform distribution (2D-Uniform)
- The Twin Astrographic Catalog Version 2 [13]

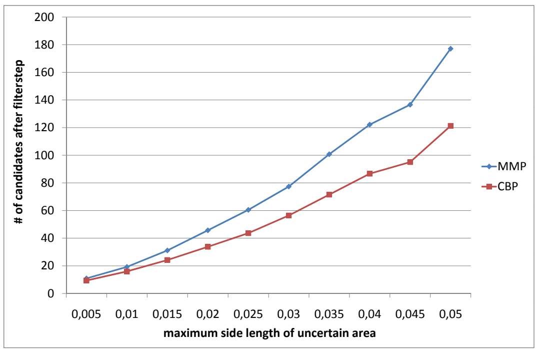
Both datasets were normalized to have a feature value in [0,1]. Each point is equivalent to the exact position of one object. The uncertain area of each object o is constructed by choosing a random rectangle that covers o with a random side length in $[0, maxsidelength]$. Queries as well as database objects were picked from one of the datasets for each experiment.

6.2 Filter Step

In our first experiment we evaluate the pruning power of our pruning strategy CBP in comparison to the basic pruning strategy MMP. First, we investigate the cost required to refine the query object for varying threshold parameter m in terms of the average number of query object refinements. Note, that in the case where refining the query object is not required, we have no refinement costs at all, i.e. no database object has to be refined. The results are depicted in Figure 8(a). For very small m ($m < 5$) our advanced approach CBP has no improvement in comparison to the simple

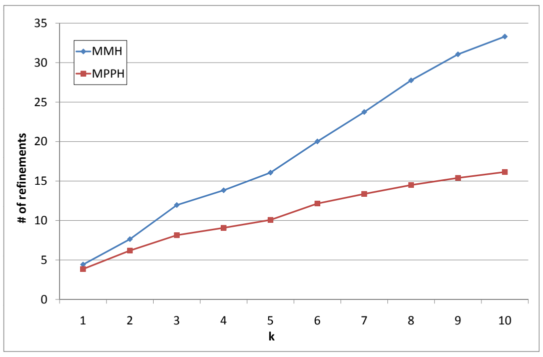


(a) Relative number of refinements of q

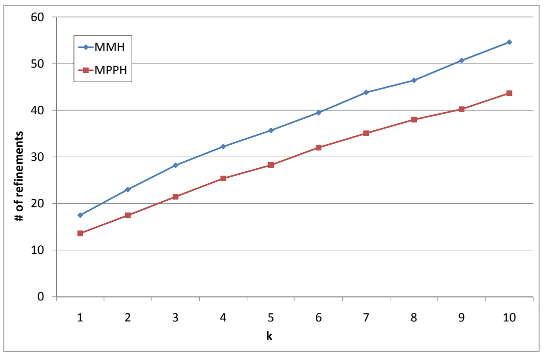


(b) Number of candidates after the first filter step

Figure 8: MMP vs. CBP on uniform dataset



(a) MMH vs. MPPH on uniform data



(b) MMH vs. MPPH on tac dataset

Figure 9: MMH vs. MPPH on uniform and tac dataset

MMP approach because for a small m almost in all cases the result set $Card(RkNN(q))$ exceeds m and, thus, the query object has to be refined. For larger settings of the parameter m, indeed we can save refinements of the query object and, thus, can save the computation of the CRkNN(q) result. With increasing the parameter m, we can observe that our geometrical pruning strategy increasingly outperforms MMP. In particular, for $m = 11$ the CBP is two times more selective than the MMP approach.

Furthermore, we investigated the pruning power of both pruning strategies in consideration of the number of result candidates reported by the first filter (without any refinement). Figure 8(b) shows the number of candidates generated in the first filter for varying sizes of the rectangles used to approximate the object positions in terms of the side length of the rectangles. It is obvious that larger rectangles lead to a lower pruning power with both competing approaches. However, the CBP pruning produces about $1/3^r d$ less candidates compared to MMP.

6.3 Refinement Strategy

The next set of experiments concerns the effectivity of the proposed refinement strategies. As a baseline, we use the *Minimal Mindist Heuristics* (MMH) (c.f. 4.2.1) and compare it to our proposed *Maximum Pruning Potential Heuristics* (c.f. 4.2.2). We measure the total number of refinements, that is, the number of objects that are required to submit their exact position. Figure 9(a) shows the result of our evaluation with respect to k on uniform dataset. It can be observed that the $MPPH$ shows only a small improvement over the the MMH for small values of k, but becomes significant outperforms MMH as k increases. In contrast, the same evaluation is shown in Figure 9(b) for the tac dataset. It can be observed that the total number of refinements is higher on the tac dataset. This can be explained by the clustered nature of the tac dataset. Objects are closer to each other than in the uniform dataset, while the max side length of an uncertain area is the same, resulting in more overlap of the uncertain regions. It can also be observed that the improvement of $MPPH$ is relatively large for small values of k, increases rather slowly as k increases.

6.4 Constraint Parameter m

As the parameter m constrains the minimum number of RkNN results, which are of interest to the user, it can be used to stop the refinement process at an early stage. Thus, increasing m results in a smaller number of refinements. Figure 10 again shows the number of necessary refinements (using $MPPH$) with respect to k, but with different values of m. It can be seen, that for the same number of k, the number of refinements can be enormously reduced by a bigger m. Keep in mind that the case where $m = 1$ corresponds to a traditional, non-constrained RkNN query.

6.5 Bichromatic Case

In the bichromatic case the interesting variable is the propotion of $Card(\mathcal{D}_{red})$ to $Card(\mathcal{D}_{blue})$. Figure 11(a) shows the number of refinements dependent on the size of $\mathcal{D}_{red}$, where the size $\mathcal{D}_{blue}$ is fixed to 10000. Figure 11(b) shows the opposite case, where

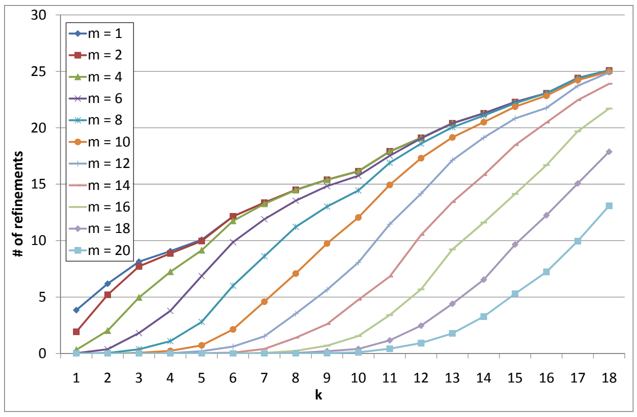
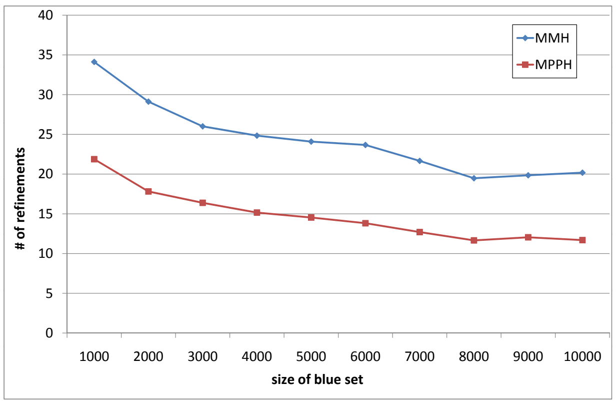
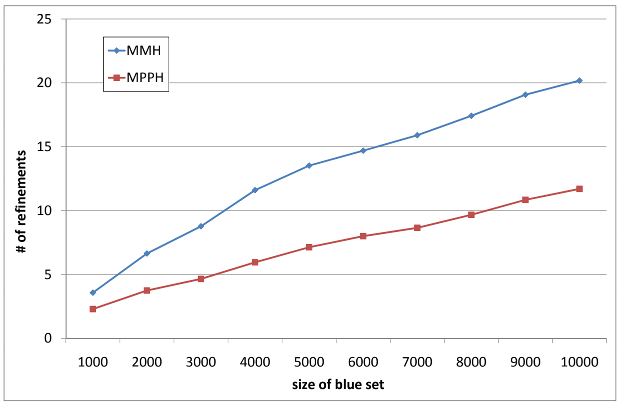


Figure 10: Impact of parameter m on the number of refinements



(a) Varying $Card(\mathcal{D}_{red})$



(b) Varying $Card(\mathcal{D}_{blue})$

Figure 11: MMH vs. MPPH with varying size of the two sets

Card($\mathcal{D}_{red}$) is set to 10000 and Card($\mathcal{D}_{blue}$) varies from 1000 to 10000. In both cases MPPH outperforms MMH, using up to 50% fewer refinements.

7. CONCLUSIONS

In this paper, we formalize a novel server-side reverse k-nearest neighbor problem for moving clients, the constrained reverse k-nearest neighbor (CRkNN) query. In a client/server scenario, the bottleneck for the resources is typically not the query execution time like I/O or CPU costs at clients or at the server. Rather, the communication load needs to be minimized because the most precious resource is the power supply of the clients. Especially in applications, where miniature GPS-devices with low power resources are used and the exchange of single devices is very complex, the goal of query processing is to save even single messages between clients and the server. While existing methods for the traditional RkNN problem are not directly designed to optimize the communication load but rather the query execution time, we propose an original solution for CRkNN queries in such a client/server application. In fact, we propose several new pruning strategies in order to keep the number of candidates as low as possible using only approximative information on the location of the clients. Our experimental evaluation confirms that our novel solution is superior to existing approaches adapted to the CRkNN problem in terms of communication costs.

8. REFERENCES

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

[2] 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.

[3] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor estimation. In *Proc. BTW*, 2007.

[4] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *EDBT*, pages 886–897, 2009.

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

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

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

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

[9] I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse nearest neighbor queries for dynamic databases. In *Proc. DMKD*, 2000.

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

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

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

[13] N. Zacharias and M. I. Zacharias. The twin astrographic catalog on the hipparcos system. *The Astronomical Journal*, 118(5):2503–2510, 1999.