Aggregate Nearest Neighbor Queries in Spatial Databases

DIMITRIS PAPADIAS
Hong Kong University of Science and Technology, Hong Kong, China

YUFEI TAO
City University of Hong Kong, Hong Kong, China

KYRIAKOS MOURATIDIS
Hong Kong University of Science and Technology, Hong Kong, China

CHUN KIT HUI
Hong Kong University of Science and Technology, Hong Kong, China

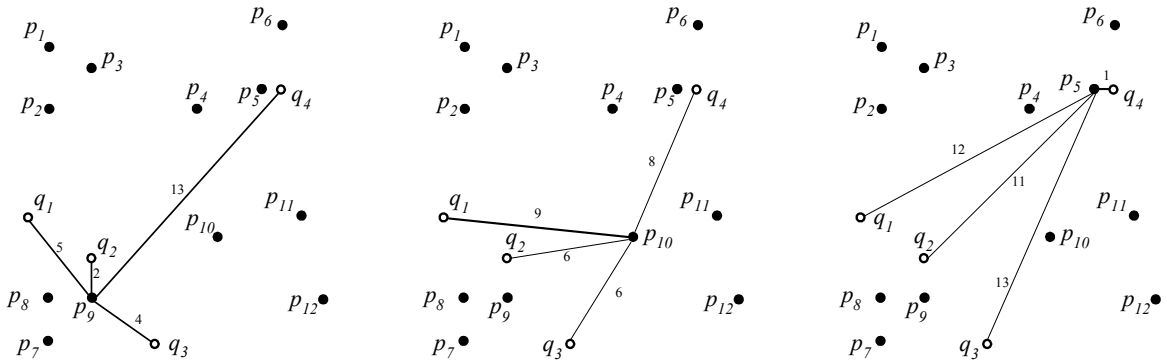
Given two spatial datasets P (e.g., facilities) and Q (queries), an *aggregate nearest neighbor* (ANN) query retrieves the point(s) of P with the smallest aggregate distance(s) to points in Q. Assuming, for example, n users at locations q_1, ... q_n, an ANN query outputs the facility $p \in P$ that minimizes the *sum* of distances $|pq_i|$ for $1 \leq i \leq n$ that the users have to travel in order to meet there. Similarly, another ANN query may report the point $p \in P$ that minimizes the *maximum* distance that any user has to travel, or the *minimum* distance from some user to his/her closest facility. If Q fits in memory and P is indexed by an R-tree, we develop algorithms for aggregate nearest neighbors that capture several versions of the problem, including weighted queries and incremental reporting of results. Then, we analyze their performance and propose cost models for query optimization. Finally, we extend our techniques for disk-resident queries and approximate ANN retrieval. The efficiency of the algorithms and the accuracy of the cost models are evaluated through extensive experiments with real and synthetic datasets.

Categories and Subject Descriptors: H.2 [**Database Management**]; H3.3 [**Information Storage and Retrieval]**
General Terms: Algorithms, Experimentation
Additional Key Words and Phrases: Spatial Database, Nearest Neighbor Queries, Aggregation

This research was supported by the grant HKUST 6180/03E and CityU 1163/04E from Hong Kong RGC.
Authors' addresses: Dimitris Papadias, Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong; email: dimitris@cs.ust.hk; Yufei Tao, Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong; email: taoyf@cs.cityu.edu.hk; Kyriakos Mouratidis, Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong; email: kyriakos@cs.ust.hk; Chun Kit Hui, Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong; email: michaelh@cs.ust.hk.

1. INTRODUCTION

This paper proposes and solves *aggregate nearest neighbor* (ANN) queries in spatial databases. Let f be a monotonically increasing function[1] and $Q=\{q_1,\ldots,q_n\}$ be a set of query points. We define the *aggregate distance* between a data point p and Q as $adist(p,Q) = f(|pq_1|, \ldots, |pq_n|)$, where $|pq_i|$ is the Euclidean distance of p and q_i. Given a set $P=\{p_1,\ldots,p_N\}$ of static data points, an ANN query returns the data point p with the minimum aggregate distance. Similarly, a k-ANN query outputs the k ($\geq$1) data points with the smallest aggregate distances. As an example consider Figure 1, where the dataset P contains a set of facilities $p_1,\ldots,p_{12}$ and Q is a set of user locations $q_1,\ldots,q_4$. If f is the *sum* function (Figure 1a), the corresponding 1-ANN query reports the facility (p_9) that minimizes the total distance that the users have to travel in order to meet there, i.e., $adist(p_9,Q) = \sum_{i=1}^{4}|p_9q_i| = 24 \leq adist(p,Q)\ \forall\ p \in P$.



(a) f=*sum* (b) f=*max* (c) f=*min*

Fig. 1. Examples of ANN queries

On the other hand, if f is the *max* function (Figure 1b), the ANN query will report the facility (p_{10}) that minimizes the maximum distance that any user has to travel ($adist(p_{10},Q)= max_{i=1}^{4}|p_{10}q_i| = |p_{10}q_1| = 9$), which, in turn, leads to the earliest time that all users will arrive at the meeting point (assuming that they move with the same speed). Finally, if f = *min* (Figure 1c), the result is the facility (p_5) which is closest to any user, i.e., $adist(p_5,Q) = min_{i=1}^{4}|p_5q_i| = |p_5q_4| = 1 \leq adist(p,Q)\ \forall\ p \in P$. Another interesting instance of ANN queries occurs when each point q_i in Q is associated with a positive weight w_i. Returning to our example, consider that in Figure 1a each q_i is the position of some airplane carrying w_i passengers and the goal is to find the airport p that minimizes the total distance traveled by all passengers (as opposed to airplanes), i.e., $adist(p,Q) = \sum_{i=1}^{4}w_i\cdot|pq_i|$. Similarly, in Figure 1b, if each airplane has average travel speed v_i, then the airport p that leads to the shortest meeting time is the one that minimizes $max_{i=1}^{4}|pq_i|/v_i$, i.e., $w_i = 1/v_i$.

Aggregate NN queries constitute a generalized form of nearest neighbor search, where there are multiple (instead of one) query points and the optimization goal depends on the function f. Applications that will benefit from the efficient processing of ANN queries include resource allocation, virtual battlefield and meteorology. As an example of the first case, consider a franchise that wants to open a restaurant in order to attract the maximum number of

[1] A function f is monotonically increasing iff: $\forall i\ x_i \geq x'_i$ implies that $f(x_1, \ldots,x_n) \geq f(x'_1,\ldots,x'_n)$.

customers from a set of urban blocks. A (weighted) *sum* ANN query reports the location that minimizes the sum of distances to all urban blocks of interest. Such examples can be constructed for several similar problems (e.g., a supermarket chain that wants to determine the warehouse location minimizing the sum of distances to all stores served by the new warehouse). For the second case, assume that a military unit wants to identify a collection point for troops under distress. A *max* ANN query outputs the point that leads to the earliest pick-up time. As a meteorology example, consider a system that monitors severe weather phenomena (e.g., typhoons). A *min* ANN reports the urban areas under the highest potential danger based on their proximity to any phenomenon.

In addition to their usefulness as standalone methods, ANN query processing techniques can be integrated as modules for solving numerous related problems. As an example, assume that a franchise plans to open k branches in a city, so that the average distance from each residential block to the closest branch is minimized. This is an instance of *k-medoids* application, where residential blocks constitute the input dataset and the k branch locations correspond to the medoids. Despite an avalanche of methods for small and moderate-size datasets [Ng and Han 1994], currently there exists no technique for very large databases. The proposed ANN algorithms may potentially lead to scalable solutions. Furthermore, in *clustering* [Jain et al. 1999] and *outlier detection* [Aggrawal and Yu 2001], the quality of a solution can be evaluated by the sum of distances (or the maximum distance) between the points and their nearest cluster center. Finally, the operability and speed of very large circuits depends on the relative distance between their various components. ANN queries can be applied to detect abnormalities and guide relocation of components [Nakano and Olariu 1997].

For the following discussion we consider Euclidean distance and 2D point datasets indexed by R-trees [Guttman 1984, Beckmann et al. 1990], but the proposed techniques are applicable to higher dimensions and alternative data-partition access methods (e.g., A-trees [Sakurai et al. 2000]). For ease of presentation, the examples focus on the *sum*, *max* and *min* functions due to their significance in practice. In cases where the extension to the weighted versions of these problems is non-trivial, we comment on the necessary changes. The rest of the paper is structured as follows. Section 2 outlines related work on spatial nearest neighbor search and top-k queries. Section 3 develops three algorithms for processing memory-resident ANN queries, which are independent of the function, i.e., f is also an input parameter. Section 4 describes cost models that provide significant insight into the behavior of the algorithms and can be used for query optimization. Section 5 deals with disk-resident query sets and Section 6 discusses approximate ANN retrieval. Section 7 experimentally evaluates the effectiveness of the proposed techniques and Section 8 concludes the paper with directions for future work.

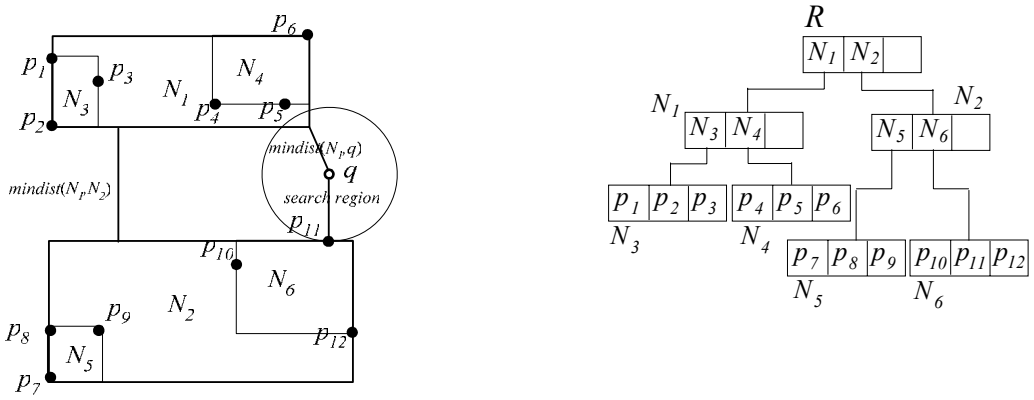
2. RELATED WORK

Nearest neighbor (NN) search is one of the oldest problems in computer science. Several algorithms and theoretical performance bounds have been devised for exact and approximate processing in main memory [Sproull 1991, Arya et al. 1998]. Furthermore, the application of NN search to content-based and similarity retrieval has led to the development of numerous cost models [Papadopoulos and Manolopoulos 1997, Weber et al. 1998, Beyer et al. 1999, Bohm 2000] and indexing techniques [Sakurai et al. 2000, Yu et al. 2001] for high-dimensional versions of the

problem. In spatial databases most of the work has focused on the *point NN query* that retrieves the k ($\geq$1) objects from a dataset P that are closest (usually according to Euclidean distance) to a point q. Algorithms for conventional (i.e., point) NN queries are discussed in Section 2.1. Section 2.2 overviews some related work in the literature of top-k (or *ranked*) queries.

2.1 Conventional NN Queries

The existing algorithms assume that P is indexed by a spatial access method (most often an R-tree) and utilize some pruning bounds to restrict the search space. Figure 2 shows an R-tree for point set $P=\{p_1,p_2,\ldots,p_{12}\}$ with a capacity of three entries per node (typically, the capacity is in the order of hundreds). Points that are close in space (e.g., p_1, p_2, p_3) are clustered in the same leaf node (N_3). Nodes are then recursively grouped together with the same principle until the top level, which consists of a single root. Given a node N and a query point q, the *mindist*(N,q) corresponds to the closest possible distance between q and any point in the subtree of node N. Similarly, *mindist*(N_1,N_2) is the minimum possible distance between any two points that reside in the sub-trees of nodes N_1 and N_2. Figure 2a shows the *mindist* between point q and node N_1, and between N_1 and N_2.



(a) Points and node extents (b) The corresponding R-tree

Fig. 2. Example of an R-tree and a point NN query

The first NN algorithm for R-trees [Roussopoulos et al. 1995] searches the tree in a depth-first (DF) manner, recursively visiting the node with the minimum *mindist* from q, e.g., in Figure 2 DF accesses the root, followed by N_1 and N_4, where the first potential nearest neighbor is found (p_5). During backtracking to the upper level (node N_1), the algorithm prunes entries whose *mindist* is equal to or larger than the distance (*best_dist*) of the nearest neighbor already retrieved. In the example of Figure 2, after discovering p_5, DF will backtrack to the root level (without visiting N_3), and then follow the path N_2, N_6 where the actual NN p_{11} is found.

Figure 3 illustrates the pseudocode for DF NN queries. This is a simplified version because the original algorithm [Roussopoulos et al. 1995] includes some additional pruning heuristics (e.g., *minmaxdist*), which are redundant (as shown in Cheung and Fu [1998]). The first call of the algorithm uses the root of the data R-tree after setting the initial value of *best_dist* to ∞. DF can be easily extended for the retrieval of k>1 nearest neighbors: the k points discovered so far with the minimum overall distances are maintained in an ordered list of k pairs <p, $|pq|$> (sorted on

|*pq*|) and *best_dist* equals the distance of the *k*-th NN. Whenever a better neighbor is found, it is inserted in the list and the last element is removed.

```
Algorithm DF NN (Node: R-tree node, q: query point)
1.  if Node is an intermediate node
2.      sort entries N_j in Node according to mindist(N_j,q) in list
3.      repeat
4.          get next entry N_j from list
5.          if mindist(N_j,q) < best_dist
6.              DF NN(N_j,q);  //Recursion
7.      until mindist(N_j,q) ≥ best_dist or end of list
8.  else // leaf node
9.      for each point p_j in Node
10.         if |p_j q| < best_dist
11.             best_NN = p_j;   best_dist = |p_j q|  //Update current NN
```

Fig. 3. DF NN algorithm

The DF algorithm is sub-optimal, i.e., it accesses more nodes than necessary. In particular, as proven in Berchtold et al. [1997], Papadopoulos and Manolopoulos [1997], Bohm [2000], an optimal algorithm should visit only nodes intersecting the *search region*, i.e., a circle centered at the query point q with radius equal to the distance between q and its nearest neighbor. In Figure 2a, for instance, an optimal algorithm should visit only the root, N_1, N_2, and N_6 (whereas DF also visits N_4). The best-first (BF) algorithm of Henrich [1994] and Hjaltason and Samet [1999] achieves the optimal I/O performance by maintaining a heap H with the entries visited so far, sorted by their *mindist*. As with DF, BF starts from the root, and inserts all the entries into H (together with their *mindist*), e.g., in Figure 2a, H={<N_1, *mindist*(N_1,q)>, <N_2, *mindist*(N_2,q)>}. Then, at each step, BF visits the node in H with the smallest *mindist*. Continuing the example, the algorithm retrieves the content of N_1 and inserts all its entries in H, after which H={<N_2, *mindist*(N_2,q)>, <N_4, *mindist*(N_4,q)>, <N_3, *mindist*(N_3,q)>}. The next two nodes accessed are N_2 and N_6 (inserted in H after visiting N_2), in which p_{11} is discovered as the current NN. At this time, the algorithm terminates (with p_{11} as the final result) since the next entry (N_4) in H is farther (from q) than p_{11}.

Similarly to DF, BF can be easily extended to kNN queries (k>1). In addition, BF is *incremental*, i.e., it can output the nearest neighbors in ascending order of their distance to the query without a pre-defined termination condition. Consider, for example, a query that asks for the nearest city of Hong Kong with population more than 2M. An incremental algorithm finds the nearest city c_1, and examines whether it qualifies the population condition. If the answer is negative, the algorithm retrieves the next nearest city c_2 and repeats this process until a city satisfying the population condition is found. The implication is that the number of nearest neighbors to be retrieved is not known in advance. Figure 4 shows the incremental version of BF, where data points are reported according to the order that they are de-heaped. To simplify the pseudocode, in line 6 we assume that the *mindist* of a data point corresponds to its actual distance.

Algorithm *incremental* BF NN (q: query point)

1. insert <*R-tree root*, *mindist*(*root*,*q*)> into heap H
2. repeat
3. deheap next entry <*e*, *mindist*(*e*,*q*)> from H
4. if e is a data point report <*e*, *mindist*(*e*,*q*)>
5. else // e is a node
6. for each entry e_j in e insert < e_j, *mindist*(e_j,*q*)> into heap H
7. until termination condition

Fig. 4. Incremental BF NN algorithm

A similar framework also applies to *closest pair queries* that find the pair of objects (from two datasets) with the minimum distance. Hjaltason and Samet [1998], Corral et al. [2000], and Shan et al. [2003] propose various algorithms based on DF and BF traversal. The difference from NN is that the algorithms access two index structures (one for each data set) simultaneously. If the *mindist* of two intermediate nodes N_i and N_j (one from each R-tree) is already greater than the distance of the closest pair of objects found so far, the sub-trees of N_i and N_j cannot contain a closest pair (thus, the pair is pruned). Furthermore, variations of nearest neighbor queries have been studied in the context of (i) road networks, where the distance between two points is defined as the length of the shortest path connecting them in the network [Papadias et al. 2003, Kolahdouzan and Shahabi 2004], (ii) spatio-temporal databases, where the query and/or the data objects move [Kollios et al. 1999, Song and Roussopoulos 2001, Benetis et al. 2002, Tao and Papadias 2003], (iii) data stream environments, which necessitate techniques for the efficient handling of continuous queries and intensive updates [Liu and Ferhatosmanoglu 2003, Yu et al. 2005, Xiong et al. 2005].

2.2 Top-*k* Queries

A top-k query returns the k tuples with the highest scores according to a monotone preference function. Assuming that the data reside in the same repository, such queries can be processed by *Onion* [Chang et al. 2000] (based on convex hulls) or *Prefer* [Hristidis and Papakonstantinou 2004] (based on view materialization). If the data reside in multiple databases, several algorithms (see Fagin et al. [2001], Bruno et al. [2002], Fagin [2002]) compute the final output by integrating partial search results in each repository. As an example, consider that a user wants to find the k images that are most similar to a query image Q, where similarity is defined according to n features q_1,.., q_n, e.g., color histogram, object arrangement, texture, shape etc. The query is submitted to n retrieval engines that return the best matches for particular features together with their similarity scores. The problem is to combine the multiple outputs in order to determine the top-k results in terms of their overall similarity.

Assume that we want to retrieve the single best match in terms of the overall similarity, defined as the sum of scores for individual features. The *threshold algorithm* [Fagin et al. 2001], shown in Figure 5, works as follows. The first search engine returns the data object p_1 with the highest score on the first feature q_1. The global similarity between p_1 and Q (with respect to all features) is computed. Then, the second search engine returns the best match p_2 according to q_2. The overall similarity of p_2 is also computed, and the best of p_1 and p_2 becomes the current result (*best_result*). The process is repeated in a round-robin fashion, i.e., after the last search engine is queried, the second match is retrieved with respect to q_1 and so on. The algorithm terminates when the global similarity of the

current result is higher than the similarity that can be achieved by any subsequent solution. In order to compute this value, the algorithm maintains, for each search engine, a local threshold t_i that corresponds to the partial score of the last retrieved object (with respect to feature q_i). The best possible similarity that can be achieved by any not-yet discovered point is $T = \sum_{i=1}^{n} t_i$. In the next section we adapt this approach to ANN processing.

Algorithm *threshold* (Q: set of query features)
/ T: global threshold; best_sim: similarity of the best current solution*/*
1. for each query feature: t_i=0
2. T=0; *best_sim* =∞; *best_result*=null *//Initialization*
3. while (T < *best_sim*)
4. select the next query feature q_i
5. get the next match p_i of q_i *// probe search engine i*
6. t_i = score of p_i on feature q_i; update T *// the global threshold increases*
7. compute the global *similarity*(p_i,Q) of p_i
8. if *similarity*(p_i,Q)<*best_sim*
9. *best_result*=p_i; *best_sim* = *similarity*(p_i,Q) *//Update current result*
10. end of while

Fig. 5. The *threshold* algorithm

3. ANN ALGORITHMS FOR MEMORY-RESIDENT QUERY SETS

Sections 3.1, 3.2 and 3.3 present three methods for processing memory-resident ANN queries. For simplicity, we illustrate all algorithms for single (k=1) ANN retrieval in 2D space using depth-first traversal. Section 3.4 discusses modifications for arbitrary values of k and dimensionality, as well as, best-first traversal including incremental output of the results and handling of non-spatial preferences. Table I contains the primary symbols used in our description.

Table I. Frequently used symbols

Symbol	Description
Q (Q_i)	set of query points (that fits in memory)
n (n_i)	number of query points in Q (Q_i)
M (M_i)	MBR of Q (Q_i)
$\|pq_i\|$	Euclidean distance between p and query point q_i
$adist(p,Q)= f(w_1 \cdot \|pq_1\|, \ldots, w_n \cdot \|pq_n\|)$	aggregate distance between point p and query points in Q
q	centroid of Q
$mindist(N,q)$	minimum distance between MBR of node N and point q
$mindist(p,M)$	minimum distance between data point p and query MBR M
$amindist(N,Q)=f(mindist(N,q_1),\ldots, mindist(N,q_n))$	*aggregate mindist* of node N from points in Q
$amindist(N,M)=f(mindist(N,M),\ldots, mindist(N,M))$	*aggregate mindist* of node N with respect to M
$amindist(p,M)=f(mindist(p,M),\ldots, mindist(p,M))$	*aggregate mindist* of point p with respect to M

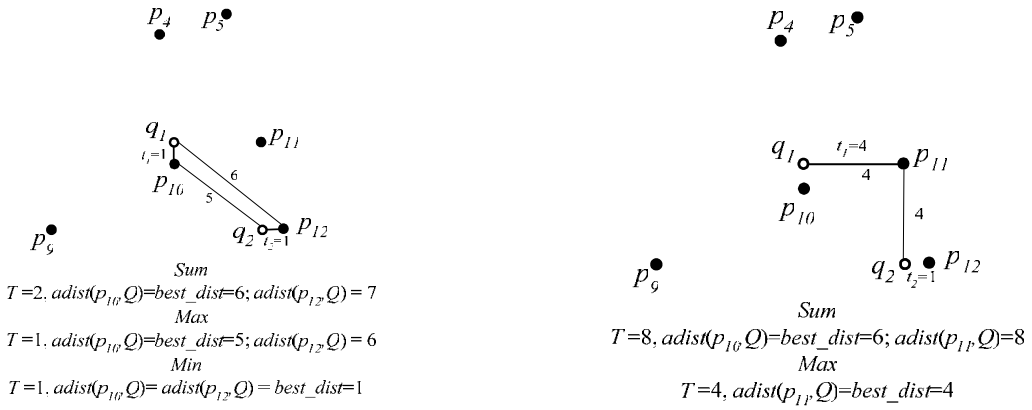
3.1 Multiple Query Method

The *multiple query method* (MQM) utilizes the main idea of the *threshold algorithm*, i.e., it performs incremental NN queries for each point in Q and combines their results.

- **Algorithm**

Assume the set of data points shown in Figure 6 and a query with $Q = \{q_1,q_2\}$, $f = sum$. MQM retrieves the first NN of q_1 (p_{10} with $|p_{10}q_1|$=1) and computes $|p_{10}q_2|$=5, $adist(p_{10},Q)= |p_{10}q_1| + |p_{10}q_2| = 6$. Similarly, the method finds the

first NN of q_2 (point p_{12} with $|p_{12}q_2|$=1) and computes $|p_{12}q_1|$ = 6, $adist(p_{12},Q)$=7. Since $adist(p_{10},Q) < adist(p_{12},Q)$, p_{10} becomes the current ANN (*best_NN*) of Q. *Best_dist* (=6) denotes the aggregate distance of *best_NN* (p_{10}). For each query point q_i, MQM stores a threshold t_i, which is the distance of its current NN, i.e., $t_1=|p_{10}q_1|=1$ and $t_2=|p_{12}q_2|=1$. The total threshold T is defined as $T = f(t_1, ..., t_n)$, which in this case is the sum of the two thresholds (=2). Continuing the example, since $T <$ *best_dist*, it is possible that there exists some point in P whose distance to Q is smaller than *best_dist*. Therefore, MQM retrieves the second NN of q_1 (p_{11}) and computes the aggregate distance $adist(p_{11},Q) = |p_{11}q_1| + |p_{11}q_2| = 8$. The *best_NN* remains p_{10} because *best_dist* < $adist(p_{11},Q)$. The threshold values are set to t_1 (=$|p_{11}q_1|$) = 4, T=5 and MQM continues since *best_dist* (=6) > T. The next query finds again p_{11} (as the second NN of q_2) and sets t_2 = 4 and T=8. MQM now terminates (*best_dist* < T) with p_{10} as the result. In other words, every non-encountered point has distance greater than or equal to T, and therefore it cannot be closer to Q than p_{10} in terms of aggregate distance.



(a) Retrieval of 1st NN (b) Retrieval of 2nd NN

Fig. 6. Example of MQM

The only difference for the *max* function concerns the computation of the aggregate distances and the value of the threshold T. Returning to the example of Figure 6, the aggregate distance of the first NN (p_{10}) of q_1 is $adist(p_{10},Q) = max(|p_{10}q_1|,|p_{10}q_2|) = 5$. Similarly, for the first NN of q_2, $adist(p_{12},Q) = max(|p_{12}q_1|,|p_{12}q_2|) = 6$ and p_{10} becomes the *best_NN* with *best_dist*= 5. The threshold T is set to $max(t_1,t_2)$ =1. Since *best_dist* > T, MQM continues with the second NN of q_1 (p_{11}), which has $adist(p_{11},Q) = max(|p_{11}q_1|,|p_{11}q_2|) = 4$ and replaces p_{10} as the *best_NN*. The value of T is set to 4 (i.e., the new t_1) and MQM terminates with p_{11} as the final result. The processing of the *min* function is simpler since it suffices to find the first NN of each query point; the final result is the NN (in this case, p_{10} or p_{12}) with the minimum distance.

Figure 7 shows the pseudocode for MQM (single NN retrieval) capturing any monotonically increasing function. The only lines specific to individual functions are 7 (update of threshold T) and 8 (computation of *adist*). The algorithm for computing nearest neighbors of query points should be incremental (e.g., best-first search as shown in Figure 4) because the termination condition is not known in advance. The handling of weights is straightforward; in particular, the aggregate distance and global threshold computations must also take into account the weights (e.g., in the case of the *sum* function $T = \sum_{i=1}^{n} w_i \cdot t_i$). Furthermore, instead of retrieving NNs for each query point in a round-

robin fashion, we choose the next query point q_i with a probability that is proportional to its weight w_i. This minimizes the extent of individual queries since, intuitively, the final result is more likely to be near the "most important" query points. The weighted *min* function is an exception to this policy because, unless there is a data point that coincides with some query point (i.e., $adist(p,Q)=0$), the ANN algorithm can only terminate if there is one NN found for each query point.

Algorithm MQM (Q: set of query points, f: monotonic function)
/ T: global threshold; best_dist: aggregate distance of the current NN */*
1. for each query point: $t_i=0$
2. $T=0$; *best_dist*$=\infty$; *best_NN*=null *//Initialization*
3. while ($T <$ *best_dist*)
4. select the next query point q_i
5. get the next nearest neighbor p_j of q_i *//Using incremental BF NN*
6. $t_i = |p_j q_i|$
7. update T; *//*$T = f(t_1, t_2, ..., t_n)$
8. compute $adist(p_j,Q)$
9. if $adist(p_j,Q)<$*best_dist*
10. *best_NN* $=p_j$; *best_dist* = $adist(p_j,Q)$ *//Update current ANN of Q*
11. end of while

Fig. 7. The MQM algorithm

- **Proof of correctness**

Lemma 3.1: MQM reports the point of P with the minimum aggregate distance from Q.

Proof: Let p_{best} be the output of MQM and $adist_{best}$ its aggregate distance. Clearly, p_{best} has the minimum *adist* among all the points encountered by the algorithm before its termination. In order to prove correctness, it suffices to show that each non-encountered point $p \in P$ has $adist(p,Q) \geq adist_{best}$. MQM terminates when $T = f(t_1,\ldots,t_n) \geq adist_{best}$. It holds that $|pq_i| \geq t_i$ for each q_i because, otherwise, p would have been discovered by the incremental NN search for some query point. Due to the monotonicity of f: $adist(p,Q) = f(|pq_1|,\ldots,|pq_n|) \geq f(t_1,\ldots,t_n) = T \geq adist_{best}$. Hence, the result of MQM is correct. The proof also captures the existence of weights since the inequality $f(w_1 \cdot |pq_1|,\ldots,w_n \cdot |pq_n|) \geq f(w_1 \cdot t_1,\ldots,w_n \cdot t_n) = T$ still holds.

3.2 Single Point Method

MQM may incur multiple accesses to the same nodes and retrieve the same data point through different queries. To avoid this problem, the *single point method* (SPM) processes ANN queries by a single index traversal. First, SPM computes the *aggregate centroid* q of Q, which is a point in space that minimizes (exactly or approximately) the value of $adist(q,Q)$. The intuition behind this approach is that the aggregate nearest neighbor is a point of P "near" q. The centroid depends on the function f and its exact computation is not always possible. Nevertheless, SPM returns correct results for any possible selection of q; a good approximation (or, ideally, an exact aggregate centroid) leads to fewer node accesses. It remains to derive (i) the computation of q, and (ii) the range around q in which we should search for points of P, before we conclude that no better NN can be found.

- **Centroid computation**

The centroid computation is performed at the initialization phase of SPM and does not incur any I/O operations since Q is memory-resident. In the case of *sum*, q minimizes the function $adist(q,Q) = \sum_{i=1}^{n}|qq_i|$, and is also known as the Fermat-Weber point [Wesolowsky 1993]. Let (x,y) be the coordinates of q and (x_i,y_i) be the coordinates of query point q_i. Since the partial derivatives of function $adist(q,Q)$ with respect to its independent variables x and y are zero at the centroid q, we have the following equations:

$$\frac{\partial\, adist(q,Q)}{\partial\, x} = \sum_{i=1}^{n} \frac{(x-x_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2}} = 0, \quad \frac{\partial\, adist(q,Q)}{\partial\, y} = \sum_{i=1}^{n} \frac{(y-y_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2}} = 0$$

Unfortunately, the above equations cannot be solved into closed form for n>2, or in other words, they must be evaluated numerically, which implies that the centroid is approximate. In our implementation, we use the *gradient descent* [Hochreiter et al. 2001] method to quickly obtain a good approximation. Specifically, the method starts with the *geometric centroid*, i.e., $x=(1/n)\cdot \sum_{i=1}^{n}x_i$, and $y=(1/n)\cdot\sum_{i=1}^{n}y_i$, and modifies its coordinates as follows:

$$x = x - \eta\frac{\partial\, adist(q,Q)}{\partial\, x} \text{ and } y = y - \eta\frac{\partial\, adist(q,Q)}{\partial\, y},$$

where η is a step size. The process is repeated until the distance function $adist(q,Q)$ converges to a minimum value. The same technique, but with initial coordinates (for the geometric centroid) $x=(1/\sum_{i=1}^{n}w_i)\cdot\sum_{i=1}^{n} w_i\cdot x_i$ and, $y=(1/\sum_{i=1}^{n} w_i)\cdot\sum_{i=1}^{n}w_i\cdot y_i$, captures the existence of weights.

For the *max* function, the centroid (minimizing the function $adist(q,Q) = max_{i=1}^{n}|qq_i|$) corresponds to the center of the smallest disk that contains all points in Q. This is also known as the *minimum enclosing circle problem*, for which a variety of algorithms derive exact answers. In our implementation, we use the *randomized incremental algorithm* of Welzl [1991] with expected linear time (to the number of query points). Unfortunately, to the best of our knowledge there is no method for deriving the exact centroid in the presence of weights. Thus, for weighted *max*, we can use the same centroid as in the un-weighted case.

Considering the *min* aggregate function, any of the query points can be selected as the centroid since it leads to $adist(q,Q) = min_{i=1}^{n}|qq_i| = 0$. Among the n alternatives, we choose the one that minimizes $max_{i=1}^{n}|q_iq_j|$ (i.e., the query point with the lowest maximum distance from any point in Q). The reason for this choice will become apparent after the description of the pruning strategy of SPM. In the weighted *min* scenario, among the n possible choices for q, we select the query point q_i with the maximum weight w_i.

- **Algorithm**

Having computed the aggregate centroid, we now proceed with the main algorithm. The centroid q can be used to prune the search space based on the following lemma.

Lemma 3.2: Let $Q=\{q_1,...,q_n\}$ be a set of query points and q an arbitrary point in space. The following inequality holds for any data point p: $adist(p,Q) \geq f(|pq|-|q_1q|,\ldots, |pq|-|q_nq|)$, where $|pq|$ denotes the Euclidean distance between p and q.

Proof: By the triangular inequality, for each query point q_i we have that: $|pq_i|+|q_iq|\geq|pq| \Rightarrow |pq_i|\geq|pq|-|q_iq|$. Due to the monotonicity of f: $adist(p,Q) = f(|pq_1|,\ldots, |pq_n|) \geq f(|pq|-|q_1q|,\ldots, |pq|-|q_nq|)$.

As shown in the proof of correctness, Lemma 3.2 ensures that the following heuristic for pruning intermediate nodes is safe, i.e., it only discards nodes that cannot contain qualifying ANNs.

Heuristic 1: Let q be the centroid of Q and *best_dist* be the distance of the best ANN found so far. Node N can be pruned if:

$$f(mindist(N,q)-|q_1q|,\ldots, mindist(N,q)-|q_nq|)\geq best\_dist$$

where $mindist(N,q)$ is the minimum distance between the MBR of N and the centroid q, and *best_dist* is the aggregate distance of the current ANN. For *sum*, the inequality of Heuristic 1 becomes:

$$\sum_{i=1}^{n}(mindist(N,q)-|q_iq|)\geq best\_dist \Leftrightarrow n\cdot mindist(N,q) \geq best\_dist +\sum_{i=1}^{n}|q_iq|.$$

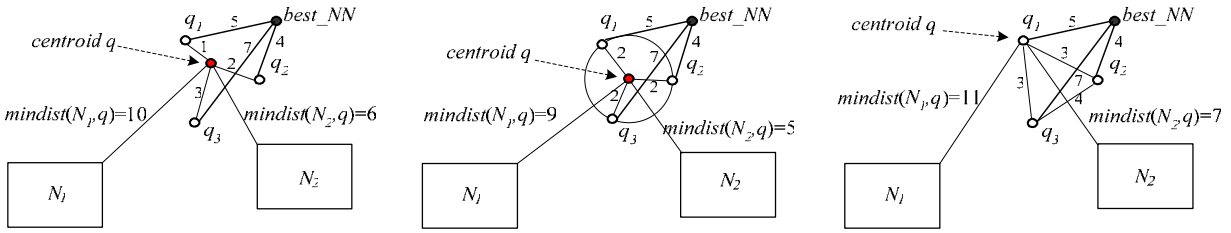
Figure 8a shows an example, where $\sum_{i=1}^{n}|q_iq|=6$ and *best_dist* = 16. Since $3\cdot mindist(N_1,q) = 30 \geq best\_dist + \sum_{i=1}^{n}|q_iq| = 22$, node N_1 is pruned. On the other hand, N_2 has to be visited because it passes Heuristic 1 (i.e., $3\cdot mindist(N_2,q) =18< 22$). For *max*, Heuristic 1 takes the form:

$$max_{i=1}^{n}(mindist(N,q)-|q_iq|)\geq best\_dist \Leftrightarrow mindist(N,q) \geq best\_dist + min_{i=1}^{n}|q_iq|.$$

As an example, consider Figure 8b, where $min_{i=1}^{n}|q_iq|=2$ and $best\_dist = max(4,5,7) = 7$ (note that the centroids are different for each function). Node N_1 is pruned since $mindist(N_1,q) = 9 \geq best\_dist + min_{i=1}^{n}|q_iq| = 9$, but node N_2 is visited because $mindist(N_2,q) = 5 < 9$. For *min*, the pruning condition translates to:

$$min_{i=1}^{n}(mindist(N,q)-|q_iq|)\geq best\_dist \Leftrightarrow mindist(N,q) \geq best\_dist + max_{i=1}^{n}|q_iq|,$$

which explains (in the un-weighted case) the choice of q as the query point that minimizes $max_{i=1}^{n}|q_iq|$. In the example of Figure 8c, q_1 is selected because it leads to the lowest maximum distance (=3), and therefore to the highest pruning power of the heuristic. Assuming that the ANN found so far has $best\_dist = min(4,5,7)=4$, both N_1 and N_2 are pruned because $mindist(N_1,q) = 11 \geq best\_dist + max_{i=1}^{n}|q_iq| = 7$ and $mindist(N_2,q) = 7 \geq 7$.



(a) *f=sum* (b) *f=max* (c) *f=min*

Fig. 8. Pruning of nodes in SPM

Based on the above observations, it is straightforward to implement SPM using the depth-first or best-first paradigms. Figure 9 shows the pseudocode of DF SPM. Starting from the root of the R-tree (for P), entries are sorted in a list according to their *mindist* from the query centroid q and are visited (recursively) in this order. Once the first entry N_j with $f(mindist(N_j,q)-|q_1q|, \ldots, mindist(N_j,q)-|q_nq|) \geq best\_dist$ is found, the subsequent ones in the

list are pruned. Note that the order according to $mindist(N_j,q)$ is the same as the order according to $f(mindist(N_j,q)-|q_1q|, \ldots, mindist(N_j,q)-|q_nq|)$, since $|q_iq|$ are constants and f is monotonic. Thus, SPM is similar to a conventional algorithm that retrieves the NNs of the centroid q; the difference is in the termination condition which is now based on aggregate, instead of conventional, distance.

Algorithm SPM (*Node*: R-tree node, Q: set of query points, f: monotonic function)
/* *q: the aggregate centroid of Q* */
1. if *Node* is an intermediate node
2. sort entries N_j in *Node* according to $mindist(N_j,q)$ in *list*
3. repeat
4. get next entry N_j from *list*
5. if $f(mindist(N_j,q)-|q_1q|,\ldots, mindist(N_j,q)-|q_nq|) < best\_dist$ //passes Heuristic 1
6. **SPM**(N_j,Q) //Recursion
7. until $f(mindist(N_j,q)-|q_1q|,\ldots, mindist(N_j,q)-|q_nq|) \geq best\_dist$ or end of *list*
8. else if *Node* is a leaf node
9. for each point p_j in *Node*
10. if $adist(p_j,Q) < best\_dist$
11. $best\_NN = p_j$; $best\_dist = adist(p_j,Q)$ //Update current ANN

Fig. 9. The SPM algorithm

- **Proof of correctness**

Lemma 3.3: SPM correctly reports the point of P with the smallest aggregate distance from Q.

Proof: It suffices to prove that Heuristic 1 is safe, or, equivalently, that SPM does not prune any nodes that may contain a better NN (than the one already found). For all points p in a node N it holds that $|pq| \geq mindist(N,q) \Rightarrow |pq|-|q_iq| \geq mindist(N,q)-|q_iq| \ \forall i$. Thus, due to the monotonicity of f: $f(|pq|-|q_1q|,\ldots, |pq|-|q_nq|) \geq f(mindist(N,q)-|q_1q|,\ldots, mindist(N,q)-|q_nq|)$. A node is pruned by Heuristic 1 if and only if $f(mindist(N,q)-|q_1q|,\ldots, mindist(N,q)-|q_nq|) \geq best\_dist$. By combining the two inequalities, for each p in N: $f(|pq|-|q_1q|, \ldots, |pq|-|q_nq|) \geq best\_dist$. By Lemma 3.2, this implies that $adist(p,Q) \geq best\_dist$ for all points p in N, and, therefore, N can be safely pruned. All the above inequalities (including Lemma 3.2) hold for positive weights; thus, the proof of correctness also captures weighted functions.

3.3 Minimum Bounding Method

SPM visits the nodes and points around the aggregate centroid using the triangular inequality to prune the search space. Depending on the distance of the centroid and the query points, Heuristic 1 may lead to unnecessary node visits. The *minimum bounding method* (MBM) avoids this problem by using directly the aggregate distance between each node N and all query points: $amindist(N,Q) = f(mindist(N,q_1), \ldots, mindist(N,q_n))$.

- **Algorithm**

Starting from the root of the R-tree for dataset P, MBM visits only nodes that may contain candidate points. We first discuss heuristics for pruning nodes that cannot contain any results.

Heuristic 2: Let *best_dist* be the distance of the best ANN found so far. A node N can be safely pruned if:

$$amindist(N,Q) \geq best\_dist$$

For *sum*, the inequality of Heuristic 2 becomes $\sum_{i=1}^{n} mindist(N,q_i) \geq best\_dist$. Figure 10 shows a set of query points $Q=\{q_1,q_2\}$ and its current *best_NN* for *sum* (*best_dist*=5). N_2 is pruned because $mindist(N_2,q_1) + mindist(N_2,q_2) = 6 \geq best\_dist = 5$. For *max*, the pruning condition becomes $max_{i=1}^{n} mindist(N,q_i) \geq best\_dist$, i.e., N_2 is pruned because $mindist(N_2,q_1)=5 \geq best\_dist = 3$. For *min*, Heuristic 2 transforms to $min_{i=1}^{n} mindist(N,q_i) \geq best\_dist$, i.e., N_2 must be visited because $mindist(N,q_2)=1 < best\_dist = 2$. Heuristic 2 requires multiple distance computations (one for each query point). In order to save unnecessary computations, it is applied only for nodes that pass Heuristic 3.

Heuristic 3: Let M be the MBR of Q, and *best_dist* be the aggregate distance of the best ANN found so far. A node N cannot contain qualifying points if:

$$amindist(N,M) \geq best\_dist$$

where $amindist(N,M)=f(mindist(N,M),\ldots, mindist(N,M))$, i.e., $mindist(N,M)$ appears n times as parameter of f.

In Figure 10, since $amindist(N_1,M) = 2 \cdot mindist(N_1,M) = 6 \geq best\_dist = 5$, N_1 can be pruned. In other words, even if there is a data point p at the upper-right corner of N_1 and all the query points were lying at the lower right corner of Q, it would still be the case that $adist(p,Q) > best\_dist$. In the cases of (un-weighted) *max* and *min*, Heuristic 3 takes the same form: $mindist(N,M) \geq best\_dist$, but the values of *best_dist* differ. For instance, in Figure 10, *best_dist* =3 for *max*, whereas *best_dist* =2 for *min*. In both cases N_1 can be pruned because $mindist(N_1,M) = 3 \geq best\_dist$. The concept of Heuristic 3 also applies to the leaf entries; when a point p is encountered, we first compute $amindist(p,M)$ from p to the MBR of Q. If $amindist(p,M) \geq best\_dist$ (where $amindist(p,M) = f(mindist(p,M),\ldots, mindist(p,M))$, p is discarded since it cannot be closer than the *best_NN*.

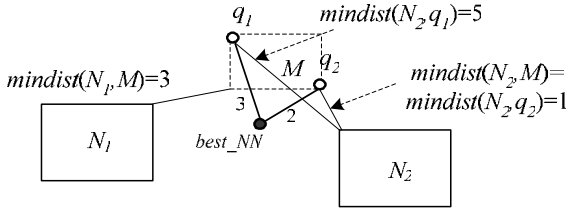


Fig. 10. Example of Heuristic 3

Because Heuristic 3 requires a single distance computation for each node under consideration, it is used as a fast filter step before the application of Heuristic 2. Note that node N_2 (in Figure 10) passes Heuristic 3 (assuming *sum*), although it cannot contain qualifying points. Heuristics 2 and 3 can be used with both the depth-first and best-first traversal paradigms. Figure 11 shows DF MBM, where nodes are visited recursively according to $amindist(N_j,Q)$.

```
Algorithm MBM (Node: R-tree node, Q: set of query points, f: monotonic function)
/* M: the MBR of points in Q */
1.  if Node is an intermediate node
2.      for each entry N_j of Node
3.          if amindist(N_j,M) < best_dist  //N_j passes Heuristic 3
4.              if amindist(N_j,Q) < best_dist  //N_j passes Heuristic 2
5.                  insert N_j in a list sorted on amindist(N_j,Q)
6.      repeat
7.          get next entry N_j from list
8.          if amindist(N_j,Q) < best_dist  then MBM(N_j,Q,f)  //Recursion
9.      until amindist(N_j,Q) ≥ best_dist or list is empty
10. else if Node is a leaf node
11.      for each data point p_j in Node
12.          if amindist(p_j,M) < best_dist  //p_j passes Heuristic 3 for points
13.              if adist(p_j,Q) < best_dist
14.                  best_NN = p_j; best_dist = adist(p_j,Q)  //Update current ANN
```

Fig. 11. The MBM algorithm

- **Proof of correctness**

Lemma 3.4: MBM correctly reports the point of P with the minimum aggregate distance from Q.

Proof: In order to prove the correctness of MBM it suffices to show that pruning is safe. For every point p in N it holds that $|pq_i| \geq mindist(N,M)$ for each q_i. Due to the monotonicity of f: $adist(p,Q) = f(|pq_1|,\ldots, |pq_n|) \geq f(mindist(N,M),\ldots, mindist(N,M)) = amindist(N,M)$. Therefore, for all points in a node N pruned by Heuristic 3 it holds that $adist(p,Q) \geq amindist(N,M) \geq best\_dist$. Similarly, for Heuristic 2, since $mindist(N,q_i)$ is the minimum distance between N and query point $q_i \in Q$: $adist(p,Q) \geq amindist(N,Q) \geq best\_dist$ for all points p in N. Hence, Heuristic 2 is also safe because N cannot contain a better ANN than the current one. It is easy to verify that the proof of correctness captures the existence of weights.

3.4 Discussion

The extension of all methods to k (>1) ANNs is similar to that of conventional NN algorithms (see Section 2.1). In particular, the k current (candidate) neighbors are maintained in a list of k pairs $<p, adist(p,Q)>$ (sorted on $adist(p,Q)$) and $best\_dist$ equals the distance of the k-th NN. Whenever a better neighbor is found, it is inserted in the list and the last element is removed. All algorithms can be used in any dimensionality. However, similar to other variations of nearest neighbor search, they are expected to suffer from the *dimensionality curse* [Korn et al. 2001]. Furthermore, since the performance of R-trees deteriorates fast with the number of dimensions and we focus on spatial data, in the sequel we assume 2D spaces. In addition, each method can be extended to other distance metrics: (i) MQM simply requires the incremental retrieval of NNs according to the new metric, (ii) SPM requires that the metric satisfies the triangular inequality, whereas (iii) MBM requires simple modifications of the *amindist* definitions. Zero weights are trivially captured by removing the corresponding query points from the query set. On the other hand, none of the proposed methods can handle negative weights because in this case the aggregate distance of an object may decrease as its Euclidean distance increases invalidating the termination conditions (e.g., in correspondence with shortest path computation, Dijkstra's algorithm can only be applied for non-negative edge

costs).

Both SPM and MBM can be implemented using best-first traversal by following the framework of conventional NN search (see Figure 4). Furthermore, all the proposed algorithms (including MQM and the ones presented in subsequent sections) can output ANNs incrementally by using a *result heap RH* to store the discovered points on ascending aggregate distance. Each point is reported when de-heaped from *RH*. For instance, incremental MQM can output all points p with $adist(p,Q)$ smaller than or equal to the current threshold T since they are guaranteed to be better than any not-yet discovered point. Similarly, incremental SPM reports all points p from *RH* such that $adist(p,Q) \leq f(|p_jq|-|q_1q|,\ldots, |p_jq|-|q_nq|)$, where p_j is the last discovered point. Lemma 3.2 guarantees that for each subsequent point p_k: $adist(p_k,Q) \geq f(|p_kq|-|q_1q|,\ldots, |p_kq|-|q_nq|)$, and therefore (due to the monotonicity of f): $adist(p_k,Q) \geq f(|p_jq|-|q_1q|,\ldots, |p_jq|-|q_nq|) \geq adist(p,Q)$. Figure 12 shows the pseudo code for incremental SPM (the other algorithms are similar and omitted).

Algorithm *incremental* BF SPM (Q: set of query points, f: monotonic function)

```
/* q: the aggregate centroid of Q */
1.  insert <R-tree root, mindist(root,q)> into heap H
2.  repeat
3.      deheap next entry <e, mindist(e,q)> from H
4.      if e is a data point p_j
5.          insert <p_j, adist(p_j,Q)> into result heap RH
6.          while the first entry <p, adist(p,Q)> of RH has adist(p,Q) ≤ f(|p_jq|-|q_1q|,…, |p_jq|-|q_nq|)
7.              deheap <p, adist(p,Q)>; output <p, adist(p,Q)>
8.      else // e is a node
9.          for each entry e_j in e insert < e_j, mindist(e_j,q)> into heap H
10. until termination condition
```

Fig. 12. Incremental BF SPM

The incremental output of the results is important for a memory-resident ANN algorithm because, as shown in Section 5.1, it permits its incorporation as a module for processing disk-resident queries. Furthermore, the incremental property allows more general queries involving additional preferences. Assume that each data point p (i.e., facility) has some non-spatial attribute(s) p^A and the goal is to find the facility that minimizes $F(adist(p,Q), p^A)$, where F is a monotone function (possibly including non-negative weights for the relative importance of $adist(p,Q)$ and p^A). For example, if p^A is the fee for using p, we want the facility that leads to the minimum overall cost (i.e., combined aggregate distance and fee). Figure 13 shows a generalized algorithm that reports the data points in ascending order of overall cost assuming that the minimum possible value of p^A is v_{min} (v_{min} can be negative). The points discovered by an incremental ANN method are inserted into a heap. Every time a point p_j is inserted, all points p such that $F(adist(p,Q), p^A) \leq F(adist(p_j,Q), v_{min})$ are de-heaped and reported, because they are guaranteed to have lower cost than any not-yet discovered point (given that the weights are non-negative). Alternatively, if the data points can be efficiently retrieved in ascending order of the p^A values (e.g., by *Prefer* [Hristidis and Papakonstantinou 2004]), we can integrate the partial result with incremental MQM using the *threshold*, or some other related, algorithm [Fagin et al. 2001].

Algorithm *generalized* ANN (Q: set of query points, F: monotonic function)

1. Repeat
2. get the next ANN p_j of Q // *using any incremental ANN algorithm*
3. compute $F(adist(p_j,Q), p_j^A)$
4. insert $<p_j, F(adist(p_j,Q), p_j^A)>$ into heap H
5. while the first entry p of H has $F(adist(p,Q), p^A) \leq F(adist(p_j,Q), v_{min})$
6. deheap $<F(adist(p,Q), p^A)>$; output $<F(adist(p,Q), p^A)>$
7. end of while
8. until termination condition

Fig. 13. Generalized ANN including non-spatial preferences

Finally, note that a simple way to process memory-resident queries is to (i) scan the data file, (ii) compute the aggregate distance of each point, and (iii) return the k points with the minimum distance. This *file scan method* (FSM) is general since it can deal with negative weights, any function, distance metric and dimensionality. Furthermore, it performs sequential page accesses, whereas the proposed methods incur random node visits. Nevertheless, it is non-incremental and, as we show in the experimental evaluation, it is usually very expensive due to the CPU overhead (for computing the aggregate distance of all data points).

4. PERFORMANCE ANALYSIS

This section presents a performance study on ANN retrieval, which (i) provides insight into the characteristics of the proposed algorithms, and thus promotes our understanding about their behavior; (ii) leads to models that can be utilized by query optimizers. To facilitate analysis, we present the main results with two assumptions: (i) data points distribute uniformly in the data space (without loss of generality, we assume each axis has range [0, 1]); (ii) all queries have the same weight (i.e., they are equally important). We later remove these constraints and deal with general settings. Our derivation utilizes some previous formulae describing several properties of R-trees.

Property 1 Theodoridis et al. [2000], Bohm [2000]: For uniform data, the node MBRs at the same level of the tree have approximately equal sizes. Specifically, at the j-th level, each MBR is a square with side length s_j equal to:

$$s_j = \left(max\left(|P| / f^{j+1}, 1 \right) \right)^{-1/d} \qquad (4\text{-}1)$$

where $|P|$ is the data cardinality, f the node fan-out (i.e., average number of entries in a node, typically 69% of the node capacity [Theodoridis et al. 2000]), and d (=2 in the following discussion) is the dimensionality. We assume that leaves are at level 0.

Property 2 Tao et al. [2004]: The number $n_{spInt}(s_j,r)$ of level-j nodes whose MBRs intersect a circle with radius r can be computed as:

$$n_{spInt}\left(s_j, r\right) = \left(|P| / f^{j+1} \right) \left(s_j + r\sqrt{\pi} \right)^2 \qquad (4\text{-}2)$$

where s_j is given in Equation 4-1.

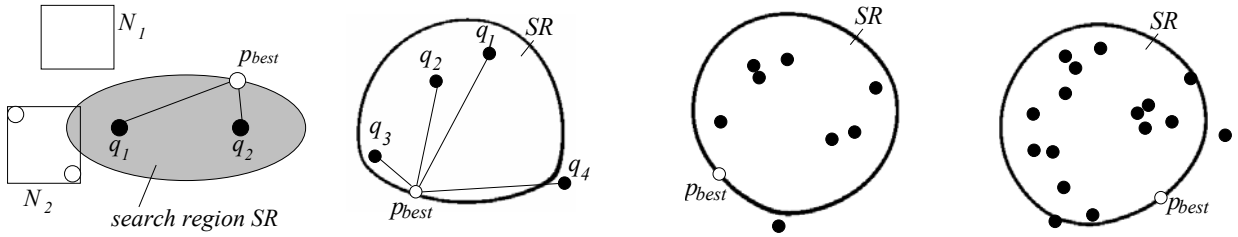
Property 3 Berchtold et al. [1997], Papadopoulos and Manolopoulos [1997], Bohm [2000]: Any exact algorithm for processing a conventional k NN query q, must access all nodes whose MBRs intersect a *search region* (*SR*) defined

by a circle centered at q with radius $|p_{best}q|$, where p_{best} is the k-th NN of q. In other words, the query cost equals $\sum_{j=0}^{h-1} n_{spInt}(s_j,|p_{best}q|)$, where h is the height of the tree, and $n_{spInt}(s_j,|p_{best}q|)$ is given in Equation 4-2.

Section 4.1 elaborates the search regions for ANN retrieval and derives $adist_{best}$, i.e., the expected aggregate distance of the k-th ANN (p_{best}) of Q. Sections 4.2, 4.3 develop cost models for MQM, SPM and MBM. Finally, Section 4.4 discusses the application of the models to arbitrary data and queries.

4.1 Optimal Search Regions

Similar to previous work on conventional NN queries [Berchtold et al. 1997, Papadopoulos and Manolopoulos 1997, Bohm 2000], we first derive the *search region SR*, i.e., the area of the data space that may contain candidate results. In particular, the *SR* is defined by its shape and extent, which depends on the expected $adist_{best}$ of the best[2] ANN p_{best}. Consider the *sum* ANN query with $Q = \{q_1,q_2\}$ of Figure 14a and k=1. Then, $adist_{best} = adist(p_{best},Q) = |p_{best}q_1|+|p_{best}q_2|$. The shaded area corresponds to the *SR* containing all possible points p_x satisfying $adist(p_x,Q) \leq adist_{best}$. For n=2, the boundary of *SR* is the ellipse $|p_xq_1|+|p_xq_2| = adist_{best}$. For n>2, the search region is no longer bounded by an ellipse, but has an irregular shape that depends on the query cardinality and distribution. Figures 14b-14d illustrate the *SR* for (*sum*) ANN queries with n=4, 8, 16 query points, respectively.



(a) *SR* for 2 query points (b) *SR* for 4 points (c) *SR* for 8 points (d) *SR* for 16 points

Fig. 14. The search regions for *sum* ANN

Any ANN algorithm for processing Q must access *all nodes* that intersect the *SR* of Q. To understand this, assume that p_{best} has been found; before it can be returned as the best ANN, however, we must ascertain that there does not exist another point with even smaller *adist*. For this purpose, a node (e.g., N_1 in Figure 14a) whose MBR is disjoint with the *SR*, can be safely pruned from consideration (all points p_x in N_1 lie outside the *SR*, and hence, $adist(p_x,Q) > adist(p_{best},Q)$). On the other hand, a node that intersects the *SR* must be visited because it may contain potential candidates. For example, although both points in N_2 fall outside *SR* (i.e., they have larger aggregate distance than p_{best}), this cannot be verified without retrieving the content of N_2.

The concept of *SR* extends to other aggregation types (e.g., *min*, *max*). In general, *SR* is the locus of points p_x satisfying the inequality $adist(p_x, Q) \leq adist_{best}$, based on the corresponding aggregate function. For k-ANN, the *SR* contains exactly the k-ANNs and its boundary crosses the k-th best ANN. Figure 15a demonstrates the *SR* for *max* (assuming n=2, k=1), which is the intersection of two circles with radii $adist_{best}$ ($=max_{i=1}^{2}|p_{best}q_i|=|p_{best}q_2|$) centered at q_1, q_2, respectively. Node N, for example, should not be visited (even though it intersects the circle of q_2) because,

[2] Note that p_{best} is the final ANN, whereas *best_NN* (used in the description of the algorithms) is the best aggregate neighbor discovered so far, i.e., p_{best} is the last *best_NN* before termination. The same relation holds between $adist_{best}$ and *best_dist*.

for any point p_x in its MBR, $adist(p_x,Q) = |p_xq_1| > adist_{best}$. Figure 15b illustrates the *SR* for *min*, which is the union of two circles (centered at q_1, q_2) with radii $adist_{best} = min_{i=1}^{2}|p_{best}q_i| = p_{best}q_1$.

(a) *SR* for *max* (b) *SR* for *min*

Fig. 15. The search regions for *max* and *min* ANN

Let C_Q be the minimum enclosing circle that contains all points of Q, and q be the center of C_Q. The following lemma provides the shape of the *SR*.

Lemma 4.1: When Q contains an infinite number of points uniformly distributed in C_Q, the *SR* is a circle centered at q for *sum* and *max*.

Proof: Since $n\rightarrow\infty$, every point in C_Q is a query. To prove that the *SR* is a circle, it suffices to show that the *adist* of any two points p_1 and p'_1, which are equally distant to the center q, is identical. Let q_1 be a query point. Since there are infinite query points in C_Q we can find a "mirror" q'_1 (see Figure 16) such that (i) $|p_1q_1| = |p'_1q'_1|$, (ii) angle $\angle p_1qq_1$ (i.e., bounded by segments p_1q and qq_1) equals $\angle p'_1qq'_1$, and (iii) $\angle p_1qp'_1 = \angle q_1qq'_1$. Following the above methodology, we can derive for each $q_i \in Q$ the corresponding mirror $q'_i \in Q$ such that (i) $|p_1q_i|=|p'_1q'_i|$ and (ii) if $q_i \neq q_j$, then $q'_i \neq q'_j$. Hence, $adist(p_1,Q) = adist(p'_1,Q)$, thus completing the proof. ■

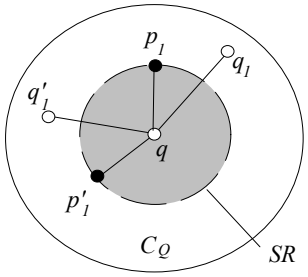
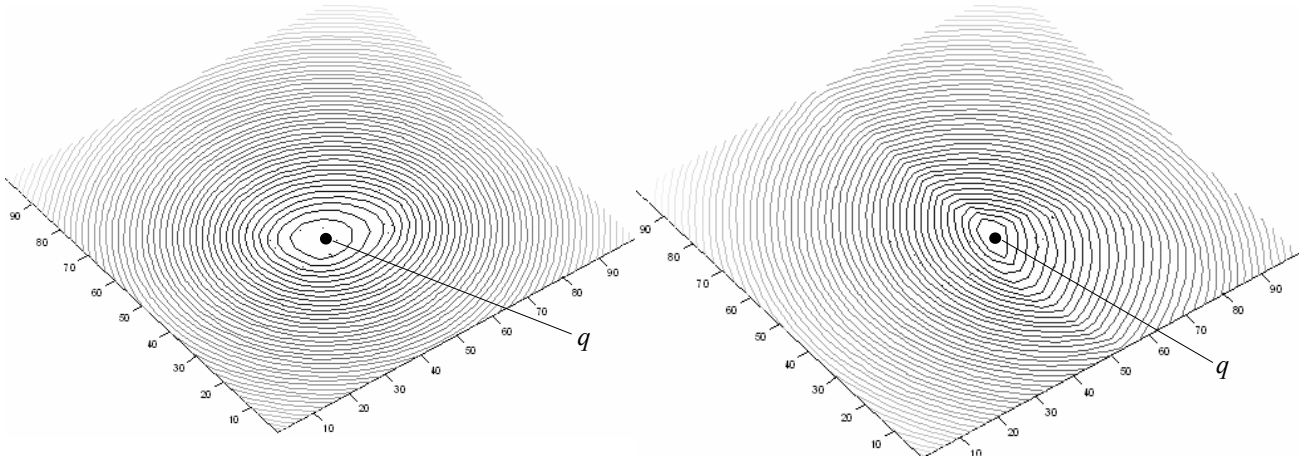


Fig. 16. Proof of Lemma 4.1

Although Lemma 4.1 is quite restrictive since it requires that $n\rightarrow\infty$, for the sake of cost analysis, the *SR* can be well estimated (for *sum* and *max* but not for *min*) as a circle, even for relatively small n. For instance, in Figure 14 the *SR* (for *sum*) becomes more circular as n increases and almost forms a perfect circle for n=8. In order to verify that this estimation is still valid even if the query points are non-uniform and have weights, we generated 16 query points following skewed distribution in a rectangle located at the center and covering 10% of the data space. Each query is associated with a weight uniformly distributed in the range [1, 100]. The "concentric" shapes in Figures 17a (*sum*) and 17b (*max*) correspond to points that have the same aggregate distance, which can be approximated as circles centered at the aggregate centroid q.



(a) *Sum* aggregate function (a) *Max* aggregate function

Fig. 17. Shape of the *SR* for finite queries

Now it remains to clarify the computation of $adist_{best}$, i.e., the estimated aggregate distance of p_{best}. In the case of *sum* and *max*, let r_{SR} be the radius of the (circular) *SR* that contains exactly k data points. Similar to the analysis of conventional NN queries, for uniform data in a unit space, the area covered by the *SR* is proportional to k; thus: $\pi r_{SR}^2 = k/|P| \Rightarrow r_{SR} = \sqrt{k/(\pi \cdot |P|)}$. Therefore, the expected $adist_{best}$ can be obtained by generating an arbitrary point with distance $\sqrt{k/(\pi \cdot |P|)}$ from q and computing its aggregate distance. Another interesting property (elaborated further in Section 6) is that the aggregate distance of a point p is expected to increase monotonically with the (Euclidean) distance $|pq|$.

On the other hand, for *min*, the *SR* is the union of n circles centered at the query points. Assuming that the n circles that constitute the *SR* contain k data points and do not intersect each other, the area covered by each one equals $k/(|P| \cdot n)$, i.e., the radius of the circle (also the value of $adist_{best}$) equals $\sqrt{k/(\pi \cdot |P| \cdot n)}$. For instance, if k=1, $|P|=10^5$ and n=64, the *SR* consists of 64 tiny circles (with radius 2.23×10^{-4}) distributed inside C_Q. Note that given the small size of the individual circles, the assumption that they do not intersect is valid for most practical settings. Furthermore, Lemma 4.1 still holds for *min* and $n \to \infty$ because the n circles essentially cover the entire *SR*.

Based on the above discussion, the pseudocode of Figure 18 returns the estimated $adist_{best}$. Unlike conventional NN queries where the *SR* provides a tight lower bound for the number of node accesses (achieved by BF traversal), as discussed in the following sections, the cost for each ANN algorithm is higher and depends on the characteristics of the method.

Algorithm *estimate_adist*$_{best}$(k: number ANNs, n: cardinality of Q, $|P|$: cardinality of P, f: monotonic function)
1. if f = *sum* or f = *max*
2. generate an arbitrary point p with distance $\sqrt{k/(\pi \cdot |P|)}$ from the centroid q
3. return $adist(p,Q)$
4. if f = *min*
5. return $\sqrt{k/(\pi \cdot |P| \cdot n)}$

End *estimate_ adist*$_{best}$

Fig. 18. Algorithm for predicting $adist_{best}$

4.2 Cost Models for MQM and SPM

MQM performs an incremental NN search for each query point q_i ($1 \le i \le n$), and continuously updates its threshold t_i, which equals the distance between q_i and its last retrieved NN. The algorithm terminates when $T=f(t_1, t_2, \ldots, t_n)$ reaches (or exceeds) $adist_{best}$. For uniform data, each query is expected to retrieve the same number of NNs (before termination) and the distance between a query point and its NN is roughly the same regardless of the query's location [Berchtold et al. 1997]. As a result, the final thresholds of all queries are approximately equal. We utilize this fact to predict t_i based on the estimation of $adist_{best}$ (as shown in Figure 18). For *sum*, when MQM terminates, $\sum_{i=1}^{n} t_i \approx adist_{best}^{sum}$, and thus $t_1 \approx t_2 \approx \ldots \approx t_n \approx adist_{best}^{sum}/n$. For *max*, eventually, $t_i \approx f(t_1, t_2, \ldots, t_n) \approx adist_{best}^{max}$ for all $1 \le i \le n$. Similarly, for *min*, $t_i \approx f(t_1, t_2, \ldots, t_n) \approx adist_{best}^{min}$.

The overhead of MQM equals the cost of n NN queries. By Property 3, the number of nodes accessed by each query is $\sum_{j=0}^{h-1} n_{spInt}(s_j,t)$, where t is the expected threshold value computed as discussed above. Therefore, the total cost of MQM is:

$$cost_{MQM} = n \cdot \sum_{j=0}^{h-1} n_{spInt}(s_j, t) \qquad (4\text{-}3)$$

Before termination SPM must visit all data points in a circle centered at the centroid q with radius r_{SPM}, determined as follows (recall the form of Heuristic 1 for individual functions):

$$r_{SPM} = \begin{cases} (adist_{best}^{sum} + \sum_{i=1}^{n} |qq_i|)/n & for\ sum \\ adist_{best}^{max} + min_{i=1}^{n} |qq_i| & for\ max \\ adist_{best}^{min} + max_{i=1}^{n} |qq_i| & for\ min \end{cases} \qquad (4\text{-}4)$$

Thus, its cost can be estimated as $cost_{SPM} = \sum_{j=0}^{h-1} n_{spInt}(s_j, r_{SPM})$. For instance, to predict the cost of a *sum* ANN query (the case of *min*/*max* is similar), the optimizer can compute r_{SPM} by using the estimate of $adist_{best}$ and calculating $\sum_{i=1}^{n} |qq_i|$. SPM is expected to outperform MQM in most cases, especially for large n due to the linear increase of MQM's cost with n. The only exception occurs for *min* retrieval and sparse query points. In this case, MQM is better because each of the (n) NN queries searches a very small area (with radius $adist_{best}^{min} \approx 0$), while SPM must access all the nodes intersecting a large circle (with radius $adist_{best}^{min} + max_{i=1}^{n} |qq_i|$).

Finally, note that the cost of SPM is sensitive to the selection of the query centroid. Consider, for instance, Figure 19 which shows a *sum* query $Q=\{q_1, q_2\}$ and two MBRs N_1, N_2 (p_{best} lies in N_2). Assuming that the centroid is q, the first point discovered is indeed p_{best}. All the node MBRs (including N_1) outside the dashed circle (centered at q with radius $adist(p_{best}, Q) + \sum_{i=1}^{n} |qq_i|$) can be pruned. On the other hand, if we select q_1 as the centroid of SPM, the first potential result is p_1, which leads to the pruning region corresponding to the solid circle with radius $adist(p_1, Q) + \sum_{i=1}^{n} |q_1 q_i|$. SPM still needs to access N_2 in order to discover p_{best}. The cost difference between q and q_1 cannot be reflected by Equation 4-4 since they both result in the same $\sum_{i=1}^{n} |qq_i| = |q_1 q_2|$. The implication is that a good aggregate centroid should, in addition to minimizing $\sum_{i=1}^{n} |qq_i|$, also be close to p_{best}.

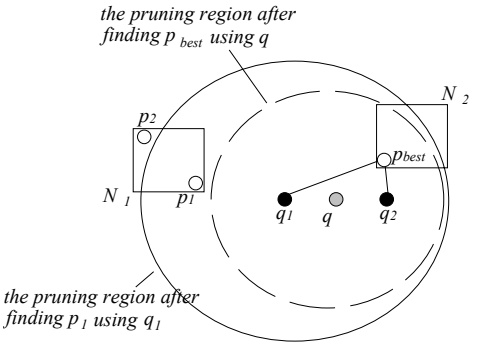


Fig. 19. The search region for SPM

4.3 Cost Model for MBM

Given an aggregate function f, MBM processes nodes N in ascending order of their $amindist(N,Q)$ and terminates when the inequality $amindist(N,Q)>adist_{best}$ holds for all non-processed nodes. In the sequel, we present the analysis for *sum*, *max*, *min* aggregates separately, due to their different characteristics.

- **Analysis of *sum***

We first deal with a related problem: given a MBR N with side length s_j (i.e., the average node extent at tree level j computed by Equation 4-1) that distributes uniformly in the data space, what is the expected $amindist(N,Q)$ as a function of $mindist(N,q)$? Figure 20 shows a quarter of the MBRs having the same $mindist(N,q)=x$. The locus of the lower-left corners of these MBRs constitutes arc AB (on the circle centered at q with radius x), plus a horizontal segment BC with length s_j. We use θ to denote the angle bounded by the x-axis and the segment connecting q with the lower-left corner of N. Obviously, (for the demonstrated locus) θ uniformly distributes in range $[0,\angle CqA]$, where $\angle CqA$ (denoted as ξ in the sequel) is the angle between segments Cq and qA and its value equals $\pi/2+tg^{-1}(s_j/x)$.

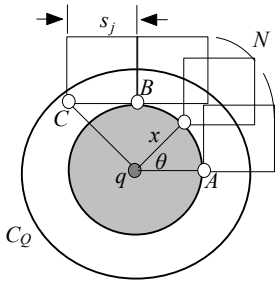


Fig. 20. Search region of MBM

Let $mindist(\theta,x,q_i)$ be the *mindist* of a node N, whose location is decided by θ and x, with respect to a query point q_i in C_Q. Similarly, $amindist(\theta,x)$ is the sum of *mindist* (with respect to all query points) as a function of θ and x. Assuming that the query points distribute uniformly in C_Q (with radius r_Q), the average $mindist(\theta,x,q_i)$ is $1/(\pi \cdot r_Q^2)\int_{q_i \in C_Q}(mindist(\theta,x,q_i))dq_i$. Consequently, $amindist(\theta,x) = n/(\pi \cdot r_Q^2) \int_{q_i \in C_Q}(mindist(\theta,x,q_i))dq_i$, leading to the following formula for the expected $amindist(x)$ (i.e., the *amindist* of a node N, whose *mindist* from q is x):

$$amindist(x) = \frac{1}{\xi}\int_0^{\xi} amindist(\theta, x)\mathrm{d}\theta = \frac{n}{\pi \cdot \xi \cdot r_Q^2}\int_0^{\xi}\int_{q_i \in C_Q} \left(mindist(\theta, x, q_i)\right)\mathrm{d}q_i\mathrm{d}\theta \qquad (4\text{-}5)$$

where $\xi=\pi/2+tg^{-1}(s_j/x)$. Although the above equation is obtained by considering a quarter of all MBRs satisfying $mindist(N,q)=x$, it also represents the overall expected *amindist* since the derivation for the other quarters is exactly the same (due to symmetry). Using Equation 4-5, we could formulate the search region of MBM at the j-th level as the circle whose radius[3] $r_{MBM\text{-}j}$ satisfies $amindist(r_{MBM\text{-}j}) = adist_{best}$. Deriving $r_{MBM\text{-}j}$ from Equation 4-5, however, is non-trivial due to the fact that the solution cannot be written in closed form, but must be solved by numerical approaches that incur high cost.

To remedy this problem, we propose an alternative method that efficiently obtains a rough estimate $r_{MBM\text{-}j}$ based on the *half-narrowing* technique [Press et al. 2002]. Specifically, an initial solution range $[x_{-}, x_{+}] = [0, c \cdot r_Q]$ is selected, where c is a constant (in our experiments, c=3). Then, we set $x_h = (x_{-}+x_{+})/2$, and formulate a MBR N with side length s_j such that (i) $mindist(N,q) = x_h$, and (ii) the segment connecting q with the lower-left corner of N is horizontal (i.e., θ=0 in Figure 20). Next we compute $amindist(N,Q)$ (i.e., using the actual query points in Q), and compare it with $adist_{best}$. If $amindist(N,Q) > adist_{best}$, the solution range is updated to $[x_{-}, x_{+}] = [x_{-}, x_h]$; otherwise, $[x_{-}, x_{+}] = [x_h, x_{+}]$. The new $x_h = (x_{-}+x_{+})/2$ is calculated and the above process is repeated until $amindist(N,Q)$ and $adist_{best}$ differ by less than a certain threshold. The final x_h constitutes the value of $r_{MBM\text{-}j}$ for computing the cost of MBM: $cost_{MBM} = \sum_{j=0}^{h-1} n_{spInt}(s_j, r_{MBM\text{-}j})$. It is worth mentioning that the above half-narrowing method does not require uniform distribution of the query points in C_Q. However, unlike the derivation based on Equation 4-5, it is not rigorous, namely, it aims at quickly finding an approximation for $r_{MBM\text{-}j}$ (instead of providing a theoretically-sound estimate).

- **Analysis of *max***

Similar to the *sum* case, we need to represent $amindist(N,Q)$ as a function of $mindist(N,q)$. Towards this, let us first fix the location of N such that (i) $mindist(N,q)=x$ and (ii) the angle between the horizontal axis and the segment connecting q with the lower-left corner of N equals θ. As shown in Figure 21, we obtain a region $ext(N,y)$ by extending N towards all directions by distance y ($ext(N,y)$ is called the *Minkowski* region of N). Let q_i be a query point uniformly distributed inside C_Q. Then, $mindist(N,q_i) \leq y$ if and only if q_i falls in $ext(N,y)$, and hence, the probability $p(x,y,\theta)$ that $mindist(N,q_i) \leq y$ equals $area(ext(N,y) \cap C_Q)/area(C_Q)$. Since $amindist(N,Q) \leq y$ if and only if *every* query point q_i satisfies $mindist(N,q_i) \leq y$, the probability for $amindist(N,Q)$ to be no more than y is $p(x,y,\theta)^n$.

[3] Note that the radius $r_{MBM\text{-}j}$ differs at various levels of the tree because, as shown in Figure 20, it depends on the MBR extent s_j at each level.

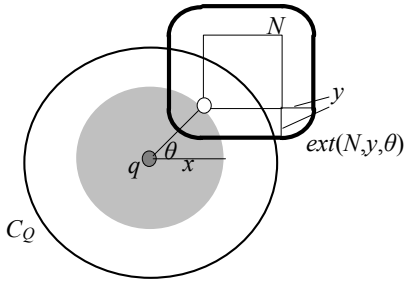


Fig. 21. Search region of MBM

Recall that so far we have fixed N at angle θ; thus, the probability that $amindist(x) \le y$ (provided that $mindist(N,q)=x$) can be obtained by integrating $p(x,y,\theta)^n$ over all θ, or formally: $(1/\xi)\int_0^{\xi} p(x,y,\theta)^n d\theta$, where $\xi=\pi/2+tg^{-1}(s/x)$, as derived earlier using Figure 20. Thus:

$$amindist(x) = \int_0^{\infty} y \cdot \frac{d\left[\frac{1}{\xi}\int_0^{\xi}\left(\frac{area\left(ext\left(N,y,\theta\right)\cap C_Q\right)}{area\left(C_Q\right)}\right)^n d\theta\right]}{dy} dy \qquad (4\text{-}6)$$

The level-j search region of MBM under *max* is a circle centered at q with radius $r_{MBM\text{-}j}$ that satisfies $amindist(r_{MBM\text{-}j}) = adist_{best}$, where *amindist* is represented in Equation 4-6 and the average node extents are provided by Equation 4-1. The cost of MBM is $cost_{MBM} = \sum_{j=0}^{h-1} n_{spInt}(s_j, r_{MBM\text{-}j})$. Given $adist_{best}$, a quick estimate of $r_{MBM\text{-}j}$ can be obtained using the half-narrowing technique discussed earlier for the *sum* case.

- **Analysis of *min***

In case of *min*, the probability $P_{acs\text{-}j}$ that a level-j node N is accessed equals the probability that $ext(N,adist_{best})$ covers any of the n query points, where $ext(N,adist_{best})$ is the region extended from N (see Figure 21) after replacing y with $adist_{best}=\sqrt{k/(\pi\cdot|P|\cdot n)}$. In order to derive $P_{acs\text{-}j}$, we first consider $P_{acs\text{-}j}(p)$, which equals the probability that $ext(N(p), adist_{best})$ covers any query point, *provided that* the center of N locates at point p. As derived in the analysis for the *max* case, the probability for *one* query point to fall into $ext(N(p), adist_{best})$ is $area(ext(N(p),adist_{best})\cap C_Q)/area(C_Q)$. As a result, the probability that a query point is not covered by $ext(N(p), adist_{best})$ equals $[1-area(ext(N(p), adist_{best})\cap C_Q)/area(C_Q)]$. Therefore, $P_{acs\text{-}j}(p)$ can be represented as: $P_{acs\text{-}j}(p) = 1-[1-area(ext(N(p),adist_{best})\cap C_Q)/area(C_Q)]^n$. Having obtained $P_{acs\text{-}j}(p)$, $P_{acs\text{-}j}$ can be computed by integrating $P_{acs\text{-}j}(p)$ over all possible positions p for the center of N, which constitute a square with the same center as the data space, but side length $1-s_j$ (where s_j is the size of a node at the j-th tree level). Therefore: $P_{acs\text{-}j}=1/(1-s_j)^2 \int_{p\in DS} P_{acs\text{-}j}(p)\, dp$.

The above formula of $P_{acs\text{-}j}$ can be simplified using the following observation. For most locations p (i.e., the center of N), $P_{acs\text{-}j}(p)=0$ because $ext(N(p), adist_{best})$ simply does not intersect C_Q at all. As a result, the integral range "$p\in DS$" can be shrunk to the area in which $P_{acs\text{-}j}(p)\neq 0$. To illustrate this, Figure 22 shows the centers A, B, C of 3 MBRs whose extended regions $ext(N(A),adist_{best})$, $ext(N(B),adist_{best})$, $ext(N(C),adist_{best})$ are tangent with C_Q. Arc ABC (the dashed curve) shows a quarter of the boundary for the actual shape of the shrunk integral region where $P_{acs\text{-}j}(p)\neq 0$. Since such an actual shape is irregular, in our implementation, we approximate it using the circle

centered at q with radius $r_Q + adist_{best}$ (we denote the circle as $ext(C_Q, r_Q + adist_{best})$).

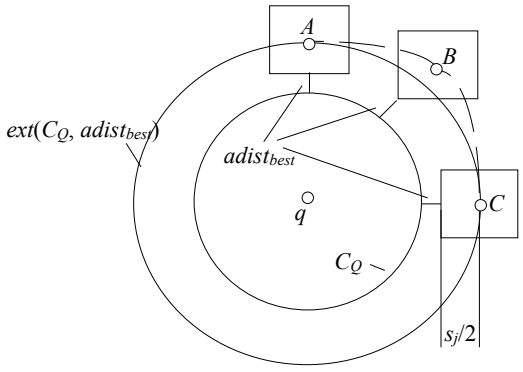


Fig. 22. Approximate integral region

To verify that this approximation does not cause significant error, observe that, for any point p in the region not captured by $ext(C_Q, adist_{best})$ (the area between the boundary of $ext(C_Q, adist_{best})$ and arc ABC), $P_{acs\text{-}j}(p)$ has very low value, since the intersection of $ext(N(B), adist_{best})$ and C_Q is very small. Therefore, $P_{acs\text{-}j}$ can be accurately represented using Equation 4-7.

$$P_{acs-j} = \frac{1}{\left(1-s_j\right)^2} \int_{p \in ext(C_Q,\ adist_{best})} 1 - \left[1 - \frac{area\left(ext\left(N(p),\ adist_{best}\right) \cap C_Q\right)}{area\left(C_Q\right)}\right]^n \mathrm{d}p \qquad (4\text{-}7)$$

The cost of MBM is $cost_{MBM} = \sum_{j=0}^{h-1} (|P|/f^{j+1} \cdot P_{acs\text{-}j})$, where $|P|/f^{j+1}$ is the number of level j nodes (f is the average node fan-out). Finally, although the above analysis is based on the assumption that queries distribute uniformly inside C_Q, it can be easily adapted to general query distribution with the following changes: $P_{acs\text{-}j}(p) = 1$ if $N(p)$ (i.e., the center of N lies at location p) covers any of the actual query points in Q, and 0 otherwise.

4.4 Capturing Weighted Queries and Arbitrary Data Distribution

As demonstrated in Figure 17, if the queries are associated with weights, the *SR* region can still be approximated as a circle for *sum* and *max* queries, permitting the direct application of algorithm *estimate_* $adist_{best}$ (Figure 18). Estimating $adist_{best}$ for *min*, however, is more complicated. In this case, *SR* is still the union of n circles, but with different sizes. Specifically, the radius r_i of each circle is such that $r_i \approx adist_{best}/w_i$ based on the intuition that search around each query point terminates when $adist_{best}$ is reached. Given the fact that the sum of the areas of all circles equals $k/|P|$, we have $\sum_{i=1}^{n} \pi \cdot (adist_{best}/w_i)^2$. Therefore, $adist_{best} = \sqrt{k/(|P| \cdot \sum_{i=1}^{n} (\pi/w_i)^2)}$ (i.e., line 5 in Figure 18 should be replaced with this estimate), and as a corollary, $r_i = \sqrt{k/(|P| \cdot \sum_{i=1}^{n} (\pi/w_i)^2)}\ /w_i$.

Similarly, the cost of MQM consists of the overhead of n NN queries such that $adist_{best} \approx w_1 \cdot t_1 \approx w_2 \cdot t_2 \approx \ldots \approx w_n \cdot t_n$. For *sum*, $w_i \cdot t_i = adist_{best}/n$, while for *max* and *min*, $w_i \cdot t_i = adist_{best}$. Hence, $cost^{sum}_{MQM} = \sum_{i=1}^{n} \sum_{j=0}^{h-1} n_{spInt}(s_j, adist_{best}/(w_i \cdot n))$ and $cost^{max}_{MQM} = cost^{min}_{MQM} = \sum_{i=1}^{n} \sum_{j=0}^{h-1} n_{spInt}(s_j, adist_{best}/w_i)$. The analysis of SPM follows exactly that in Section 4.2, i.e., $cost_{SPM} = \sum_{i=0}^{h-1} n_{spInt}(s_i, r_{SPM})$, where r_{SPM} is given in Equation 4-4. Assuming that all weights are in the range $[0, W]$, for MBM (*sum*) we first modify Equation 4-5 as follows:

$$amindist(x) = \frac{n}{\pi \cdot W \cdot \xi \cdot r_Q^2} \int_0^W \int_0^\xi \int_{q_i \in C_Q} \left(mindist(\theta, x, q_i) \right) dq_i d\theta dw \qquad (4\text{-}8)$$

For *max*, the only change is that the probability for *amindist*(N,Q), as a function of x and θ, to be no more than y is $\prod_{i=1}^{n} p(x,y/w_i,\theta)$. For *min*, the MBM formulae are the same as in the un-weighted case.

Non-uniform data can be processed using multidimensional histograms that divide the data space into a set of (disjoint) rectangular buckets. Each bucket b stores the number of points n_b in its extent a_b. Given an ANN query q, we obtain the n_b and a_b of those buckets that intersect its bounding circle C_Q. Then, the query cost is computed using directly the uniform model, by setting the dataset cardinality $\sum n_b/\sum a_b$. The intuition is that the "local" distribution inside a small region is almost uniform. Furthermore, the nodes accessed in processing q are mostly around C_Q, or equivalently, for the purpose of query estimation, the data outside C_Q can be *ignored* without causing significant error.

5. ANN ALGORITHMS FOR DISK-RESIDENT QUERY SETS

For disk-resident query sets, we consider that, in addition to monotone, the aggregate function f is also *decomposable*. Formally, f is decomposable, if it can be computed by another function g as follows: $f(x_1, x_2, ..., x_n) = g(f(x_1,...,x_\kappa), f(x_{\kappa+1},..., x_n))$ where κ can be any integer between 1 and n. Most practical aggregate functions, including *sum*, *max* and *min*, satisfy this property. For example, in case of *sum*, $g=f=sum$, i.e., $adist(p,Q) = \sum_{qi \in Q1} |pq_i| + \sum_{qj \in Q2} |pq_j|$ for any partition of Q into two sets Q_1 and Q_2. Similarly, (i) for *max*: $adist(p,Q) = max(max_{qi \in Q1} |pq_i|, max_{qj \in Q2} |pq_j|)$ (i.e., $g=f=max$) and (ii) for *min*: $adist(p,Q) = min(min_{qi \in Q1} |pq_i|, min_{qj \in Q2} |pq_j|)$ (i.e., $g=f=min$) [4]. It can be easily verified that decomposability also holds for weighted versions of these functions, e.g., $\sum_{qi \in Q} w_i \cdot |pq_i| = \sum_{qi \in Q1} w_i \cdot |pq_i| + \sum_{qj \in Q2} w_j \cdot |pq_j|$. Sections 5.1 and 5.2 present two algorithms for ANN queries based on MQM and MBM, respectively. Section 5.3 concludes with a discussion about alternative methods.

5.1 File - Multiple Query Method

MQM can be applied directly for disk-resident Q with, however, very high cost due to the large number of individual queries that must be performed (as discussed in Section 4.2, its cost increases linearly with the cardinality of Q). In order to overcome this problem, we propose F-MQM (*file-multiple query method*). F-MQM initially splits Q into blocks $\{Q_1, Q_2,..., Q_m\}$ such that each Q_i fits in memory. For every block, it computes the ANN (according to the f function) using one of the main memory algorithms, and combines their results using MQM (this time, according to the g function).

- **Algorithm**

The complication of F-MQM is that once the ANN of a block has been retrieved, we cannot directly compute its global aggregate distance (i.e., with respect to all query points). Instead, we follow a *lazy* approach: first, we find

[4] Although for *sum*, *max* and *min*, it holds that $f=g$, this is not necessarily true for other decomposable functions, e.g., $count(x_1, x_2, ..., x_n) = sum(count(x_1,...,x_\kappa), count(x_{\kappa+1},..., x_n))=n$. Furthermore, note that if f is monotonic, g must also be monotonic.

the ANN p_1 of the first block Q_1; then, we load in memory the second block Q_2 and retrieve its ANN p_2. At the same time, we also compute the distance between p_1 and Q_2, whose current aggregate distance becomes $curr\_dist(p_1) = g(adist(p_1,Q_1), adist(p_1,Q_2))$. Similarly, when we load Q_3, we update the current distances of p_1 and p_2 taking into account the objects of the third block. After the end of the first round, we only have one data point (p_1) whose *global* distance with respect to all query points has been computed. This point is the first *best_NN*.

The process is repeated in a round robin fashion and at each step a new global distance is derived. For instance, when we read again the first block (to retrieve its second ANN), the distance of p_2 (first ANN of Q_2) is completed with respect to all blocks. Between p_1 and p_2, the point with the minimum global distance becomes the *best_NN*. The threshold t_j for each block Q_j equals $adist(p_j,Q_j)$, where p_j is the last aggregate neighbor of Q_j. The global threshold T is the value of function g on the individual thresholds, i.e., $T = g(t_1, t_2,\ldots, t_m)$. F-MQM terminates when T becomes equal or larger than the global distance of the best ANN found so far.

Figure 23 illustrates the pseudocode for F-MQM. The method for retrieving the ANN of individual blocks should be incremental because the termination condition is not known in advance. In our implementation, we apply BF MBM due to its superior performance (see experimental evaluation). Weighted queries simply require the corresponding *adist* and *g* functions for each case. For the *min* function and k>1, instead of retrieving ANNs for each query block Q_i in a round-robin fashion, we compute in each step the next ANN of the block Q_i that corresponds to the minimum threshold t_i because it leads to the earliest termination (the same optimization is used for MQM on memory-resident queries).

Algorithm F-MQM ($\{Q_1, .., Q_m\}$: blocks of query points that fit in memory, f: monotonic decomposable function)
1. *best_NN = NULL; best_dist* = ∞; T=0; /* *initialization*
2. while (T < *best_dist*)
3. read next block Q_j
4. get the next aggregate nearest neighbor p_j of block Q_j // by incremental MBM
5. $curr\_dist(p_j) = adist(p_j,Q_j)$
6. $t_j = adist(p_j,Q_j)$; update T; // $T=g(t_1, t_2, \ldots, t_m)$
7. if this is one of the first m iterations of the loop
8. for each current neighbor p_i of Q_i ($1 \le i < j$) /* *update distance of other NNs*
9. $curr\_dist(p_i) = g(curr\_dist(p_i), adist(p_i,Q_j))$
10. else /**local NNs have been computed for all m blocks*
11. for each current neighbor p_i of Q_i ($1 \le i \le m, i \ne j$) /* *update distance of other NNs*
12. $curr\_dist(p_i) = g(curr\_dist(p_i), adist(p_i,Q_j))$) ;
13. $next=(j+1)$ modulo m; /* *block whose ANN global dist. is complete*
14. if $curr\_dist(p_{next}) < best\_dist$
15. $best\_NN = p_{next}$; $best\_dist = curr\_dist(p_{next})$ /* *update the global ANN of Q*
16. $next=(j+1)$ modulo m; /**next block to process*
17. end while

Fig. 23. The F-MQM algorithm

- **Proof of correctness**

Lemma 5.1: F-MQM correctly reports the point of P with the minimum aggregate distance from Q.

Proof: The proof is similar to that of MQM. In particular, F-MQM terminates when $T = f(t_1,\ldots,t_n) \ge adist_{best}$, where $adist_{best}$ is the minimum aggregate distance of all the points discovered by the algorithm. For all non-encountered

points $p \in P$ it holds that $adist(p,Q_j) \geq t_j$ for each $1 \leq j \leq m$ because, otherwise, p would have been discovered by the incremental ANN search for some query block. Due to the monotonicity and decomposability properties: $adist(p,Q) = g(f(p,Q_1), \ldots, f(p,Q_m)) \geq g(t_1, \ldots, t_m) = T \geq adist_{best}$. Thus, $adist(p,Q) \geq adist_{best}$ and p cannot belong to the result. The proof can be easily applied in the presence of weights.

5.2 File - Minimum Bounding Method

Next we describe F-MBM, an adaptation of the minimum bounding method. The basic differences with respect to MBM are: (i) there exist multiple (instead of one) query MBRs and (ii) each aggregate distance computation (for data points) incurs I/O overhead since Q is disk-resident. The first fact necessitates an alternative pruning strategy, while the second one motivates heuristics for eliminating data points without having to compute their complete distances.

- **Algorithm**

First, the points of Q are sorted by their Hilbert value and are inserted in pages according to this order. Each block Q_i consists of a number of consecutive pages that fit in memory. Due to the proximity preservation of Hilbert sorting (or any other space filling curve), each block MBR M_i is expected to be relatively small compared to the MBR of the entire Q. We keep in memory the MBRs M_i (but not their contents) and, depending on f, some additional aggregate information for the points in each block, e.g., for *sum* we also maintain in memory the number n_i of points in each Q_i. Then, F-MBM descends the R-tree of P (using DF or BF traversal), only visiting nodes that may contain qualifying points. Given that we have the values of M_i for each query block in memory, we identify qualifying nodes as follows.

Heuristic 4: Let *best_dist* be the aggregate distance of the best ANN found so far and M_i be the MBR of block Q_i. A node N can be safely pruned if:

$$g(amindist(N,M_1), \ldots, amindist(N,M_m)) \geq best\_dist$$

where $amindist(N,M_i) = f(mindist(N,M_i), \ldots, mindist(N,M_i))$, i.e., $mindist(N,M_i)$ appears n_i times as parameter of f, where n_i is the cardinality of Q_i. Figure 24 shows an example for *sum*, where 5 query points are split into two blocks with MBRs M_1, M_2 and *best_dist* = 20. According to Heuristic 4, N is pruned because: $sum(2 \cdot mindist(N,M_1) + 3 \cdot mindist(N,M_2)) = 20 \geq best\_dist$, and it cannot contain a better ANN. On the other hand, for *max*: $max(mindist(N,M_1), mindist(N,M_2)) = 4 < best\_dist = 8$ and N has to be visited. For *min*: $min(mindist(N,M_1), mindist(N,M_2)) = 4 \geq best\_dist = 1$ and N is pruned.

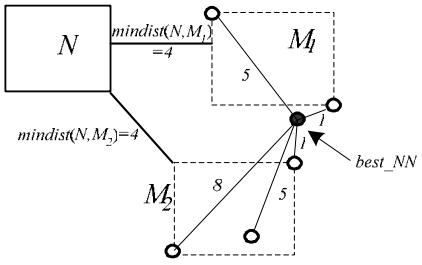


Fig. 24. Example of Heuristic 4

When a leaf node N is reached, we have to compute the global aggregate distance of its data points with all blocks. Heuristic 4 also applies for data points; i.e., $p \in N$ can be pruned if: $g(amindist(p,M_1), \ldots, amindist(p,M_m)) \geq best\_dist$, thus, avoiding the distance computations with respect to all query points. For points that pass this heuristic, we initially set the current distance $curr\_dist(p)$ to 0. Then, for each new block Q_i ($1 \leq i \leq m$) that is loaded in memory, $curr\_dist(p)$ is updated to $g(curr\_dist(p), adist(p,Q_i))$. We reduce the CPU overhead of the distance computations based on the following heuristic.

Heuristic 5: Let $curr\_dist(p)$ be the accumulated distance of data point p with respect to blocks $Q_1,\ldots,Q_{i-1}$. Then, p can be safely excluded from further consideration if:

$$g(curr\_dist(p), amindist(p,M_i), \ldots, amindist(p,M_m)) \geq best\_dist,$$

where $amindist(p,M_i)=f(mindist(p,M_i),\ldots, mindist(p,M_i))$. Figure 25 shows an example of Heuristic 5 in the *sum* case, where the first block Q_1 has been processed and $curr\_dist(p) = adist(p,Q_1) = 5+3$. Point p is not compared with the query points of Q_2, since $8+3 \cdot mindist(p_j,M_2)=20 \geq best\_dist=20$; i.e., p cannot lead to closer aggregate distance than the current result. For *max*: $max(curr\_dist(p), amindist(p,M_2)) = max(5,4) < best\_dist = 8$ and $adist(p,Q_2)$ must be computed. For *min*: $min(curr\_dist(p), amindist(p,M_2)) = min(3,4) \geq best\_dist = 1$ and p is pruned.

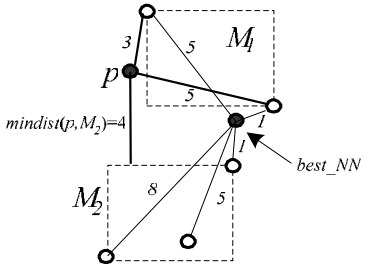


Fig. 25. Example of Heuristic 5

The handling of weights is straightforward. The only difference occurs in the case of weighted *sum*, where for each block Q_i, instead of the cardinality n_i, we store the *sum* of weights of points in it: $W_i = \sum_{qj \in Qi} w_j$. Then $amindist(N,M_i) = W_i \cdot mindist(N,M_i)$ and the inequality of Heuristic 4 takes the form of $\sum_{i=1}^{m} W_i \cdot mindist(N,M_i) \geq best\_dist$. Similarly, the pruning criterion of Heuristic 5 becomes $curr\_dist(p) + \sum_{\lambda=i}^{m} W_\lambda \cdot mindist(N,M_\lambda) \geq best\_dist$. Figure 26 shows the pseudocode of DF F-MBM. Starting from the root of the R-tree of P, entries are sorted in a list by $g(amindist(N,M_1), \ldots, amindist(N,M_m))$, and visited (recursively) in this order. The intuition is that a small value of $g(amindist(N,M_1), \ldots, amindist(N,M_m))$ is likely to lead to neighbors with small global distance, increasing the effectiveness of Heuristic 4. Once the first node that fails Heuristic 4 is found, all subsequent nodes in the sorted list can also be pruned. For leaf nodes, if a point violates Heuristic 5, it is removed from the list and is not compared with subsequent blocks.

Algorithm F-MBM(*Node*: R-tree node, $\{Q_1,..,\ Q_m\}$: blocks of query points that fit in memory, *f*: monotonic decomposable function)

```
1.  if Node is an intermediate node
2.      sort entries N_j in Node (in ascending order of g(amindist(N_j,M_1), ..., amindist(N_j,M_m)) in list;
3.      repeat
4.          get_next entry N_j from list;
5.          if g(amindist(N_j,M_1), ..., amindist(N_j,M_m)) < best_dist  /*N_j passes Heuristic 4
6.              F-MBM(N_j, {Q_1,.., Q_m}, f) ; /* Recursion
7.      until g(amindist(N_j,M_1), ..., amindist(N_j,M_m)) ≥ best_dist  or end of list;
8.  else // Node is a leaf
9.      for each point p_j in Node
10.         if g(amindist(p_j,M_i), ..., amindist(p_j,M_m))< best_dist /*p_j passes Heuristic 4 for points
11.             insert p_j in list and set curr_dist(p_j)=0; /* initialization
12.     while list is not empty and not end of blocks
13.         read next block Q_i (1≤i≤m)
14.         for each point p_j in list
15.             curr_dist(p_j)= g(curr_dist(p_j), adist(p_j,Q_i))
16.             if g(curr_dist(p_j), amindist(N,M_i+1), ..., amindist(N,M_m) ) ≥ best_dist
17.                     remove p_j from list; /* p_j fails Heuristic 5
18.     end while
19.     for each point p that remains in list  /*after termination of loops
20.         if  curr_dist(p)< best_dist
21.             best_NN = p; best_dist =  curr_dist(p)   //Update current ANN
```

Fig. 26. The F-MBM algorithm

- **Proof of correctness**

Lemma 5.2: F-MBM correctly reports the point of *P* with the minimum aggregate distance from *Q*.

Proof: First we show that for every point *p* in a pruned node *N* it holds that $adist(p,Q) \geq best\_dist$:

$adist(p,Q)=g(adist(p,Q_1),\ldots,adist(p,Q_m))$ — decomposability of *f*

$\geq g(amindist(p,M_1),\ldots, amindist(p,M_m))$ — $\forall 1\leq i\leq m\ adist(p,Q_i)\geq amindist(p,M_i)$ - monotonicity of *g*

$\geq g(amindist(N,M_1), \ldots, amindist(N,M_m))$ — $\forall 1\leq i\leq m\ amindist(p,M_i)\geq amindist(N,M_i)$ - monotonicity of *g*

$\geq best\_dist$ — definition of Heuristic 4

The version of Heuristic 4 for points (lines 9-11) is also safe since $adist(p,Q) \geq g(amindist(p,M_i), \ldots, amindist(p,M_m)) \geq best\_dist$. In addition, the proof applies to Heuristic 5 (based on the monotonicity of *f*, *g* and the fact that $adist(p,Q_i) \geq amindist(N,M_i)\ \forall 1\leq i\leq m$) and holds in the presence of weights.

5.3 Discussion

Several function-specific optimizations are possible for both F-MQM and F-MBM. In the case of *min* F-MQM, for instance, we can terminate the search for a block if the corresponding local threshold exceeds *best_dist*, even if the query has not retrieved any ANN. For F-MBM, the first leaf node visited during the processing of a *sum* query, requires reading the entire query set. Gradually, as *best_dist* decreases, so does the number of blocks that need to be loaded since numerous data points can be pruned just by using the block MBRs (instead of their contents). Because blocks that are far away from the node are likely to prune numerous data points (thus, saving the distance computations for these points with respect to other blocks), we can load each block Q_i in descending order of

amindist(N,M_i). Furthermore, we could compute the aggregate distances for points in multiple nodes by loading the query set only once. For generality, we do not focus on such function-specific optimizations.

In Papadias et al. [2004], before the application of F-MQM, we perform Hilbert sorting on the query file in order to obtain blocks with small MBRs (i.e., similar to F-MBM). Interestingly, this approach is worse than the one presented here (i.e., without Hilbert sorting). Figure 27 demonstrates this with an example where there exist 4 query blocks with disjoint MBRs, whose union covers the entire universe. For *sum* and *max*, the ANN p_i of each block Q_i lies somewhere near the centroid of M_i. On the other hand, the best ANN (p_{best}) lies near the center of the data space. Thus, it is likely that each incremental ANN query will retrieve a large number of points before p_{best} is discovered as an ANN of some block. On the other hand, if the query points in each block distribute in the entire data space, their individual ANNs are likely to be near the center, leading to the early discovery of p_{best}. Furthermore, as in the case of (main-memory) MQM, F-MQM may perform redundant computations, if it encounters the same data point as a nearest neighbor of different query blocks. A possible optimization is to keep the id of each point discovered so far, and ignore it if it is encountered later through another block (i.e., continue with the next ANN of the block). This however, may not be possible if the main memory size is limited.

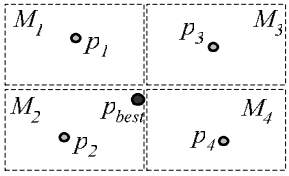


Fig. 27. Shortcomings of Hilbert sorting for F-MQM

Similar to MQM and MBM, SPM can also be applied to disk-resident queries. The difference in this case is that the computation of the aggregate centroid may require multiple passes of Q. For instance, the gradient descent method (used for *sum*) needs to compute the aggregate distance for each new centroid in order to evaluate if it constitutes an improvement over the previous one. Likewise, the derivation of the minimum enclosing circle (for *max*) must also read Q several times. In order to avoid these problems, we can use directly the geometric centroid for both *sum* and *max*. Nevertheless, F-SPM, similar to F-MBM, has to scan Q when the aggregate distance of a data point is computed. Heuristic 5 can be applied in this case, to eliminate points without computing their complete aggregate distance. Since the relative performance of F-SPM and F-MBM is expected to be similar as in the case of memory-resident queries, we omit it from the experimental evaluation.

Assuming that Q is also indexed by an R-tree, in Papadias et al. [2004] we propose GCP, an adaptation of the incremental *closest pairs* algorithm (see Section 2.1), for the *sum* function. GCP can be easily extended to any monotonic and decomposable function, but it is efficient only for *f*=*min*, since the ANN is the data point p_1 in the first output pair $<p_1, q_i>$. For other functions (such as *sum*), where the computation of *adist*(p,Q) requires all the individual distances $|pq_i| \ \forall \ q_i \in Q$, GCP incurs very high cost and extreme space requirements for managing the heap (usually in the same order as the cartesian product $P \times Q$). Therefore, we do not consider this algorithm further.

Finally, note that ANN for disk-resident queries is essentially a join problem that can be answered by computing the aggregate distance of each tuple in P (with respect to all points in Q) and then returning the k best tuples. Thus, it

can be processed by a straightforward application of the block nested loops (BNL) algorithm. Specifically, given an input buffer of B main memory pages, BNL allocates B-1 pages to P and 1 page to Q. Let b be the number of resulting blocks (with size B-1 pages) of P. For every block of P, BNL scans Q, computing the aggregate distance of all points in the block, i.e., it reads P only once and scans Q a total of b times. Compared to the previous algorithms, BNL may have better I/O performance for a small number of blocks[5] because (i) F-MQM may access the same R-tree node of P and load the same query block multiple times (through different queries or for each neighbor retrieved), while (ii) F-MBM has to scan Q every time a leaf node of the data R-tree is visited. On the other hand, the CPU cost of BNL is likely to be high because the algorithm involves $|P|\cdot n$ distance computations. BNL is experimentally compared with F-MQM and F-MBM in Section 7.2.

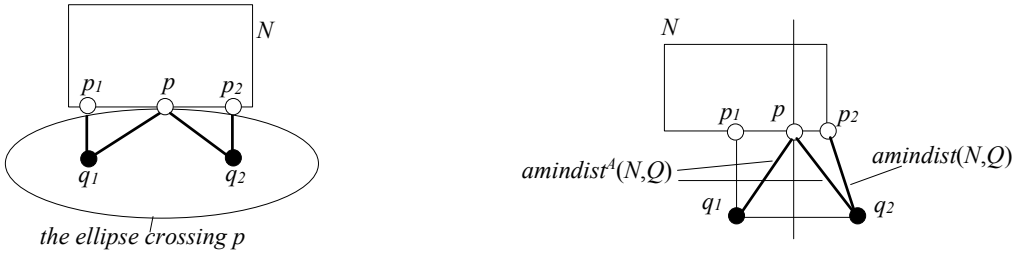
6. APPROXIMATE ANN ALGORITHMS

In general, if both the data and the query sets are large, the processing of ANN queries incurs high cost since in the worst case it may lead to $|P|\cdot n$ distance computations and multiple reads of the query file (or accesses to the same R-tree node in the case of F-MQM). In this section, we compromise accuracy for efficiency, by proposing methods for fast but approximate ANN retrieval.

The first method is based on coalescing the query points in a smaller set that fits in memory. Consider, for instance, that the available memory can accommodate $n' < n$ query points. The *coalescing approximate method* (CAM) reads the first n' query points in memory, which constitute the initial set $Q'=\{q'_1,.., q'_{n'}\}$ such that $q'_i= q_i$. Each subsequent point q_j, $n'<j\leq n$, is combined with the closest q'_i (in terms of the Euclidean distance). In particular, the combination replaces q'_i with a new point that lies on the line segment $q_jq'_i$ and its distance from the two end-points is inversely proportional to w_j and w'_i (i.e., biased towards the more important one). The weight of the new point becomes $w_j+w'_i$. After the process terminates, the ANNs are retrieved by a memory-resident algorithm using the revised set Q'. Note that CAM is sensitive to the order according to which the query points are considered. Techniques based on clustering or k-medoids could potentially lead to a Q' that better preserves the distribution of the original query set. However, such methods require reading Q several times and incur prohibitive cost (as shown in Section 7.3, CAM is already very expensive for large query sets).

Before we proceed with the second method, we define $amindist^A(N,Q)$ as the minimum *adist* of all possible points in a node N. For *sum* and *max*, $amindist^A(N,Q)\geq amindist(N,Q)$. As an example, consider the *sum* query $Q=\{q_1,q_2\}$ in Figure 28a, where $amindist(N,Q)= |p_1q_1|+|p_2q_2|$ (obtained using *two* points p_1, p_2 in N), while $amindist^A(N,Q) = |pq_1|+|pq_2|$, i.e., the point p achieving $amindist^A$ lies on the smallest ellipse that has q_1, q_2 as its foci, and intersects N. Similarly, for *max* (Figure 28b), $amindist^A(N,Q) = |pq_1| = |pq_2| > amindist(N,Q) = max(|p_1q_1|, |p_2q_2|) = |p_2q_2|$, i.e., point p lies on the perpendicular bisector of segment q_1q_2.

[5] If P fits in memory BNL reads Q exactly once, therefore, it is expected to have the best performance for memory-resident datasets and disk-resident queries.



(a) *Sum* aggregate function (b) *Max* aggregate function

Fig. 28. Metric $amindist^A$

Assuming that we can compute $amindist^A$, we can obtain a tight MBR-based heuristic a follows: if $amindist^A(N,Q)<best\_dist$, visit N; otherwise, reject it. The combination of best-first traversal with this heuristic would lead to an I/O optimal[6] method (for the same reasons that BF is optimal for conventional NN queries). However, for *sum* and *max*, the point p in N minimizing $amindist^A(N,Q)$ cannot be computed exactly. In the case of *sum*, finding p is equivalent to locating the Fermat-Weber point, but this time in a region constrained by the node MBR, which, as discussed in Section 3.2, can only be solved numerically (i.e., approximately). Similarly, for *max*, finding p is equivalent to obtaining the minimum enclosing circle whose center lies in the node MBR.

Motivated by the above observations we propose an algorithm for *sum* and *max* based on approximations of $amindist^A$. As discussed in Section 4.1, for both functions the *SR* can be regarded as a circle centered at the centroid q. Furthermore, the aggregate distance $adist(p,Q)$ of a point p increases monotonically with $|pq|$. Thus, we can obtain a rather accurate estimation of $amindist^A(N,Q)$ as follows. If N contains q we set $amindist^A(N,Q)= adist(q,Q)$ because the query centroid is assumed to minimize the aggregate distance. Otherwise, $amindist^A(N,Q)= adist(p,Q)$ where p is the point minimizing $mindist(N,q)$ (i.e., p is the point at the boundary of N that is closest to q). We call the resulting algorithm A-SPM (for *approximate* SPM) because the order of node traversal is the same as that of SPM (i.e., according to *mindist* from q), but the termination condition is different (A-SPM terminates after it retrieves the k conventional NNs of q, potentially missing some actual ANNs).

A-SPM, similar to CAM, first reads the query file in order to obtain the centroid q (or the geometric centroid, in order to avoid multiple reads of Q). Then, it simply retrieves the k Euclidean NNs of q. Finally, A-SPM and CAM have to read the query file again in order to compute the aggregate distance of the results. Assuming typical values of k (i.e., the k NNs fit in memory), all aggregate distance computations can be combined in a single scan of Q. Both algorithms are applicable to *sum* and *max* but not *min* because (i) a data point p retrieved by CAM, may not be close to any real query point (although p is close to some point of Q'), and (ii) A-SPM is based on the assumption of a single *SR* around q, whereas (as discussed in Section 4.1) the *SR* of *min* consists of n small circles. A trivial approximation method for *min*, would simply select a random subset $Q' \subseteq Q$ of query points that fit in memory, and return the ANN of Q'.

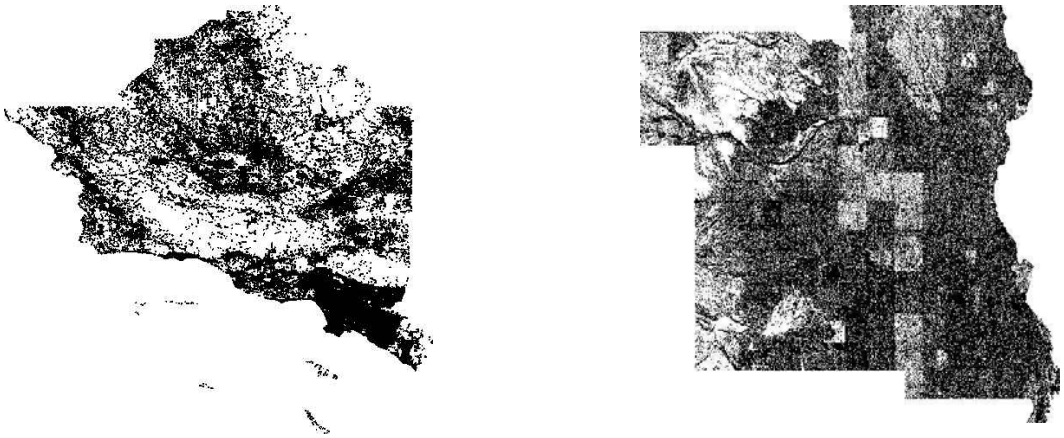
Naturally, both approximate algorithms may incur error because (i) for CAM, Q' is only a coarse estimation of Q; (ii) for A-SPM, the *SR* is not exactly a circle and q may not be the point with the minimum aggregate distance. However,

[6] MBM is optimal for *min* because $amindist^A(N,Q) = min_{i=1}^{n} mindist(N,q_i) = amindist(N,Q)$.

it is difficult to provide guarantees for the quality of the approximation since the error depends on factors, such as the distribution of the query points (and their weights), that cannot be quantified. In the next section, we experimentally evaluate the efficiency of all algorithms and the accuracy of the approximation.

7. EXPERIMENTAL EVALUATION

We evaluate the efficiency of the proposed algorithms with a Pentium 3 GHz CPU assuming a page size of 4KBytes. We use a uniform dataset (UNI) with 10^6 points, and the real datasets of Figure 29: LA and TS available at *www.rtreeportal.org*. The datasets are indexed by R*-trees [Beckmann et al. 1990] with a capacity of 204 entries per node. All algorithmic implementations are based on best-first traversal. Section 7.1 focuses on memory-resident queries, Section 7.2 on disk-resident queries, and Section 7.3 evaluates approximate algorithms.



(a) LA (1,314,620 points) (b) TS (194,971 points)

Fig. 29. Real datasets

7.1 Evaluation of Memory-Resident Queries

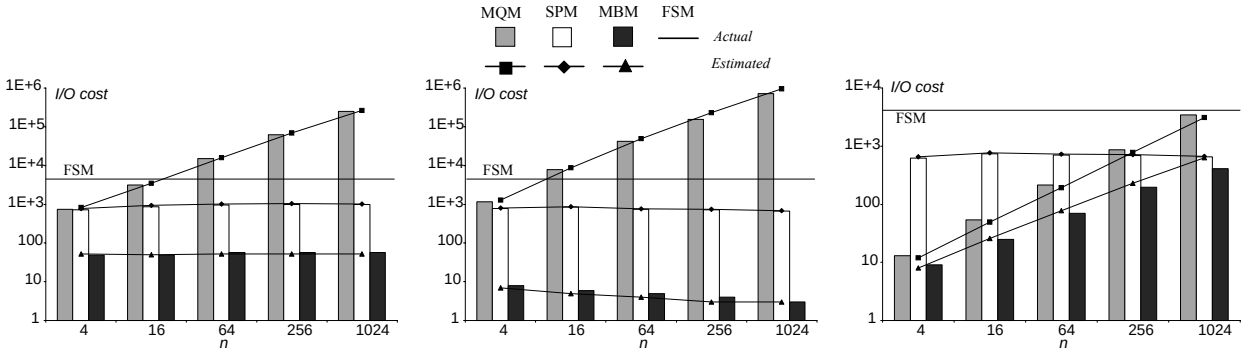
We compare MQM, SPM, MBM and the simple file scan method (FSM), discussed in Section 3.4, using workloads of 100 (un-weighted) queries on UNI and LA. For UNI, the query points follow zipf distribution (parameter a=0.8) in a circle C_Q. For LA, the query points are uniformly distributed in C_Q. Each workload has two parameters: the number n of query points and the area A_Q of C_Q. The only change between two queries in the same workload is the position of the center of C_Q, which distributes uniformly in the data space. The reported cost corresponds to the average of all queries in the workload. For the LA dataset, we apply a regular grid [Theodoridis et al. 2000] of size 100x100 to keep statistics[7] about the data distribution (to be used by the cost models as discussed in Section 4.4). For UNI, our models are applied directly to produce the estimated number of node accesses.

First we study the effect of the cardinality n of Q, by fixing A_Q to 8% of the data space and the number k of retrieved ANNs to 4. Figures 30 and 31 show the actual and estimated I/O cost (R-tree node accesses for MQM, SPM, MBM and page accesses for FSM), for UNI and LA, respectively. MQM is the worst method for *sum* and *max* due to the

[7] We use the grid approach because of its simplicity. More sophisticated multidimensional histograms (e.g., Acharya et al. [1999]) are expected to produce better results under limited memory.

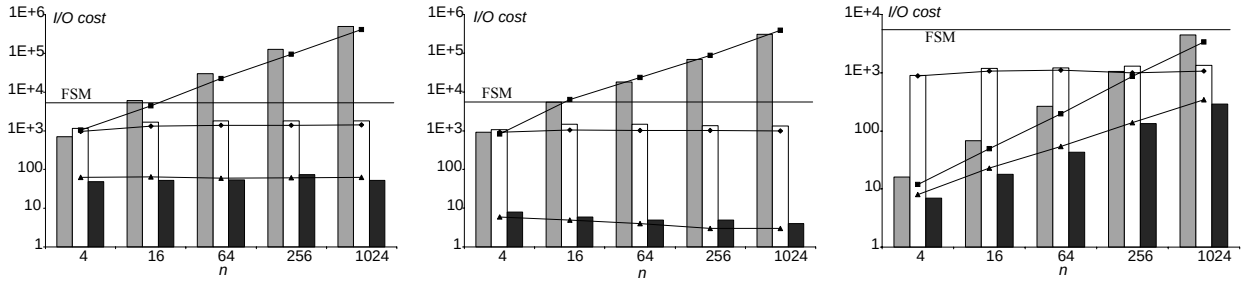
multiple NN queries, some of which access the same nodes and retrieve the same points. As predicted by the models of Section 4 its cost increases linearly with n, whereas the query cardinality has little or no effect on the other algorithms. On the other hand, for low values of n and the *min* function, MQM outperforms SPM (and FSM) because it searches a few, very small circles (around the query points) as opposed to a large circle defined by the pruning condition: $mindist(N,q) \geq best\_dist + max_{i=1}^{n}|q_i q|$ of SPM. Nevertheless, SPM is still insensitive to n and eventually surpasses MQM.

MBM is the best method in all cases. Its cost increases with n only for *min*, because in this case heuristic 2 becomes $min_{i=1}^{n} mindist(N,q_i) \geq best\_dist$, i.e., a node is visited if it is close to any query point. The results are similar for both Figures 30 and 31 despite the different data and query distributions, suggesting the generality of the above observations. The cost models capture the performance accurately; the relative error is usually around 5%-10% and it does not exceed 20% in any case. The inaccuracy is slightly larger for LA due to the error introduced by the grid histogram.



(a) *f=sum* (b) *f=max* (c) *f=min*

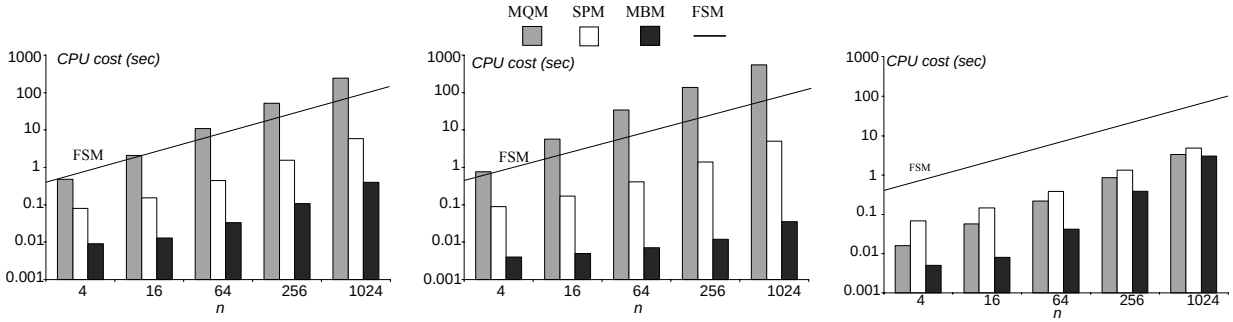
Fig. 30. Actual and estimated I/O cost vs. cardinality n of Q (UNI, A_Q=8%, k=4)



(a) *f=sum* (b) *f=max* (c) *f=min*

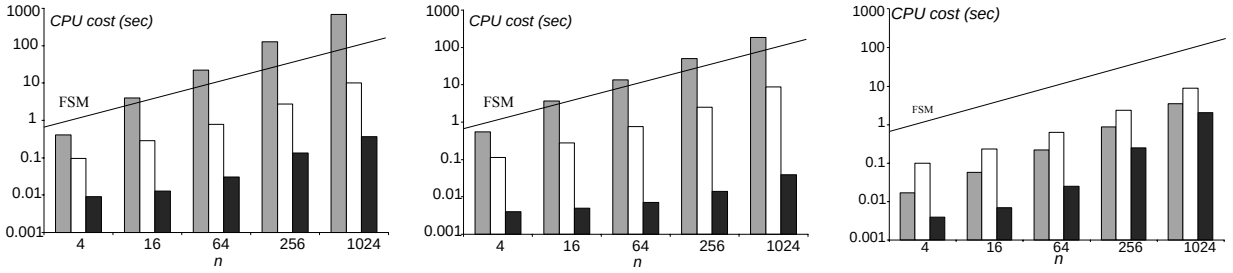
Fig. 31. Actual and estimated I/O cost vs. cardinality n of Q (LA, A_Q=8%, k=4)

Figures 32 (UNI) and 33 (LA) measure the CPU time of all algorithms under the same setting as Figures 30 and 31. The main difference is that now the overhead of FSM, SPM and MBM increases with n for all functions. This happens because, although the number of accesses may remain approximately the same for different values of n, the CPU cost for computing the aggregate distance of the data points is proportional to n. The relative performance of the algorithms is the same as the previous experiments. In the rest of the subsection, we omit the CPU diagrams since they are similar to the ones for I/O cost in all cases.



(a) *f=sum* (b) *f=max* (c) *f=min*

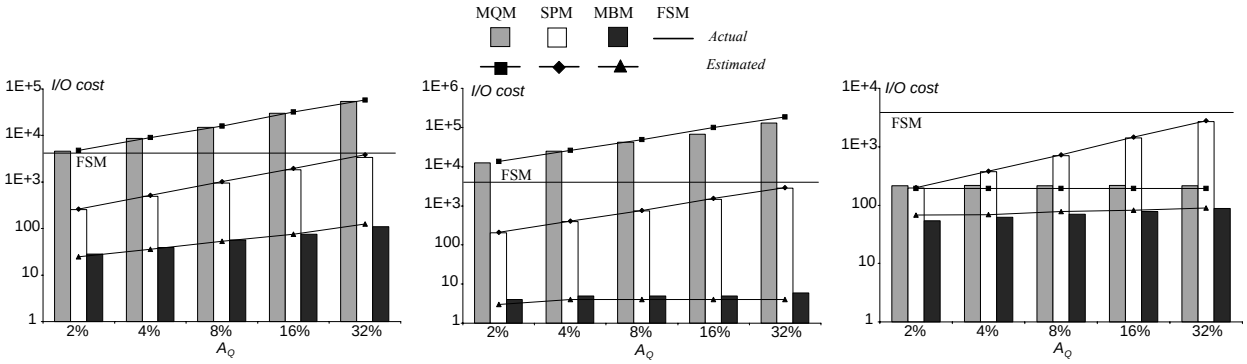
Fig. 32. CPU cost vs. cardinality *n* of *Q* (UNI, A_Q=8%, *k*=4)



(a) *f=sum* (b) *f=max* (c) *f=min*

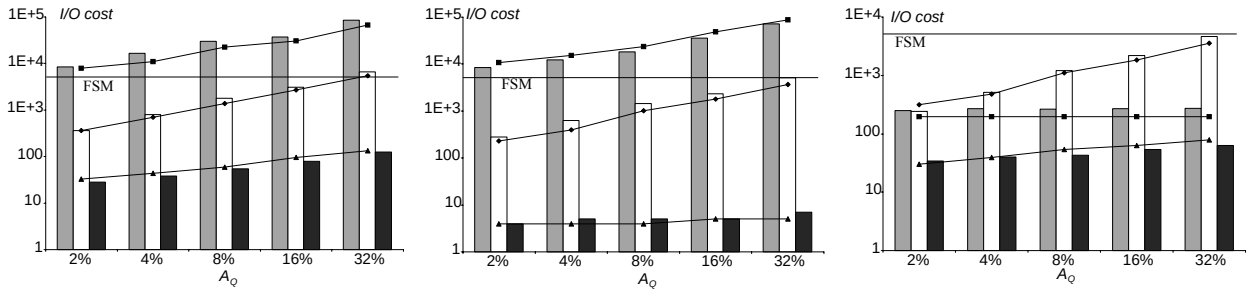
Fig. 33. CPU cost vs. cardinality *n* of *Q* (LA, A_Q=8%, *k*=4)

In order to measure the effect of the area covered by *Q*, we set *n*=64, *k*=4 and vary A_Q from 2% to 32% of the data space. As shown in Figures 34 (UNI) and 35 (LA), the I/O cost of MQM, SPM and MBM increases with A_Q for *sum* and *max*. For MQM, the termination condition is that the global threshold *T* (i.e., *sum*/*max* of thresholds for each query point) should exceed *best_dist*, which increases with the query area. For SPM and MBM, the reason is the degradation of the pruning power of the heuristics. In the *min* case, however, only SPM is affected significantly by A_Q because the factor $max_{i=1}^{n}|q_iq|$ involved in the pruning heuristic increases with the query size. The cost of FSM is obviously independent of A_Q.



(a) *f=sum* (b) *f=max* (c) *f=min*

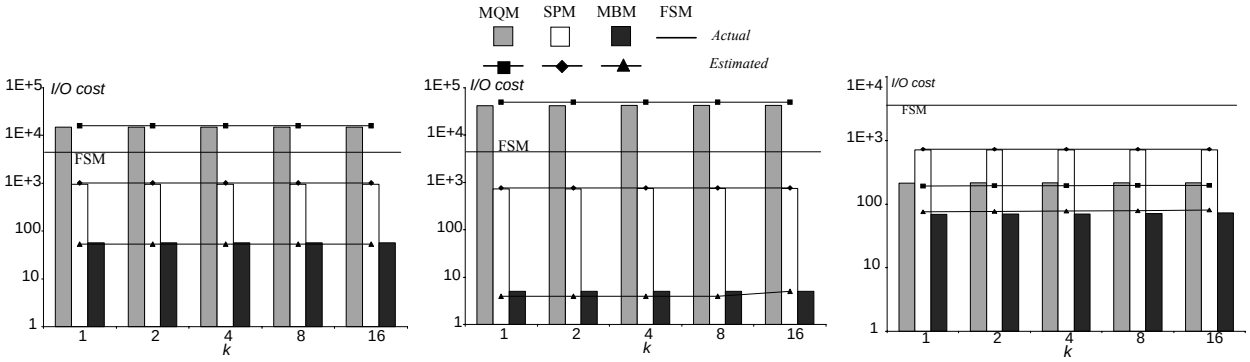
Fig. 34. Actual and estimated I/O cost vs. area A_Q of *Q* (UNI, *n*=64, *k*=4)



(a) *f=sum* (b) *f=max* (c) *f=min*

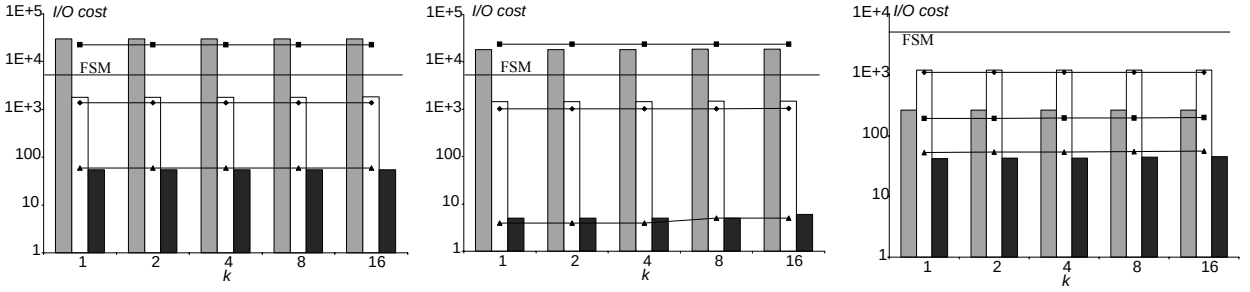
Fig. 35. Actual and estimated I/O cost vs. area A_Q of Q (LA, n=64, k=4)

In Figures 36 (UNI) and 37 (LA), we set n= 64, A_Q=8% and vary the number k of retrieved neighbors from 1 to 16. The value of k does not influence the cost of any method significantly, because in most cases all neighbors are found in a few leaf nodes.



(a) *f=sum* (b) *f=max* (c) *f=min*

Fig. 36. Actual and estimated I/O cost vs. number k of ANNs (UNI, n=64, A_Q=8%)

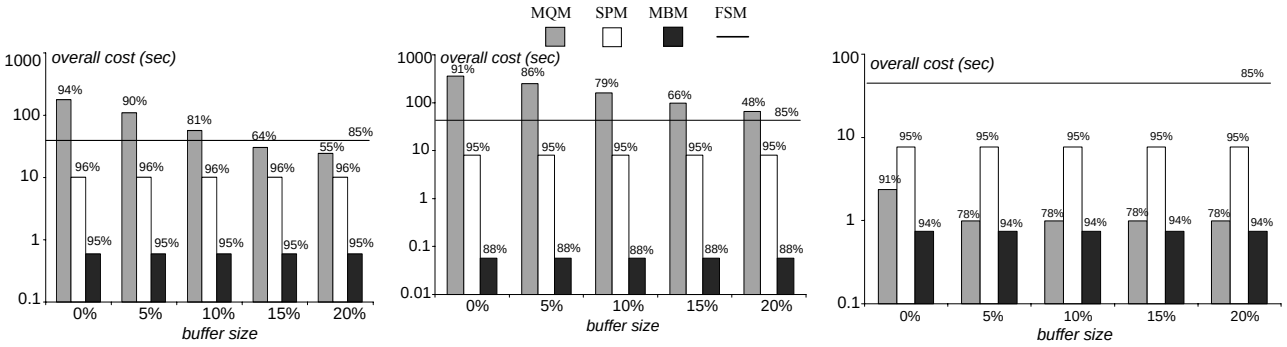


(a) *f=sum* (b) *f=max* (c) *f=min*

Fig. 37. Actual and estimated I/O cost vs. number k of ANNs (LA, n=64, A_Q=8%)

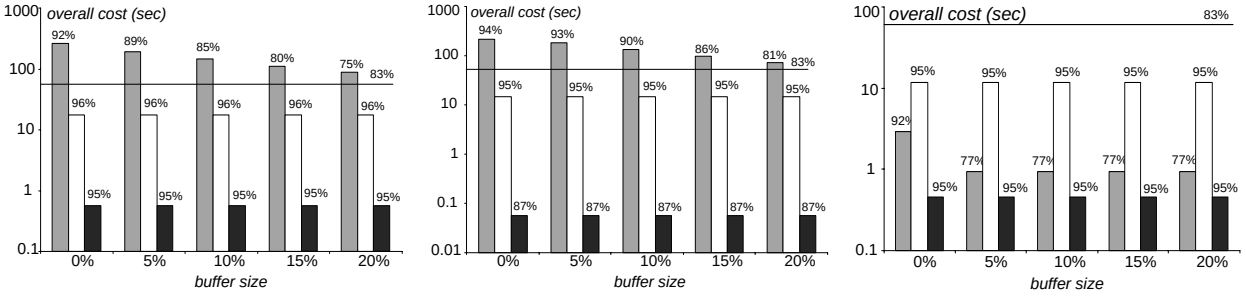
Figures 38 (UNI) and 39 (LA) study the total running time in the presence of an LRU buffer that can accommodate 0% to 20% of the cumulative number of pages in the R-tree of P. We set n=64, k=4, A_Q=8% and measure the sum of I/O and CPU costs, charging 10 milliseconds for each page fault. The percentage of the overhead corresponding to I/O appears above each column. SPM and MBM do not benefit from the existence of the buffer, since they process each node of the R-tree of P at most once. The cost of FSM is also constant because it reads each page of P exactly once and it incurs a fixed amount (i.e., $|P|\cdot n$) of aggregate distance computations. On the other hand, MQM improves significantly for larger buffer sizes because of the locality of node visits for conventional NN queries. The

experiment confirms again the superiority of MBM under all settings and functions. Thus, it is the method of choice for centralized databases indexed by R-trees (or other data-partition methods). MQM could be used for distributed systems, where each query point is processed locally and the individual results are combined to produce the final output. SPM does not have any obvious advantages over MBM, but as discussed in Section 6 and verified in 7.3, it motivates an effective approximate algorithm. Finally, FSM is applicable for non-incremental processing on non-indexed datasets.



(a) *f=sum* (b) *f=max* (c) *f=min*

Fig. 38. Overall cost vs. buffer size (UNI, n=64, k=4, A_Q=8%)



(a) *f=sum* (b) *f=max* (c) *f=min*

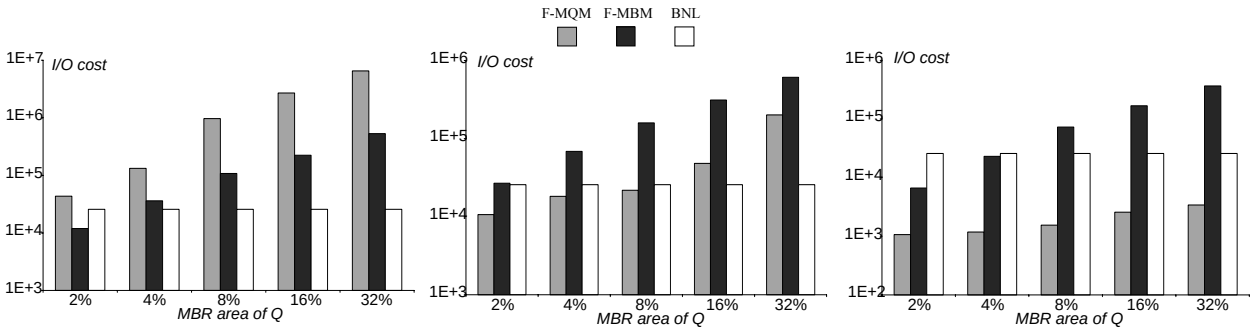
Fig. 39. Overall cost vs. buffer size (LA, n=64, k=4, A_Q=8%)

7.2 Evaluation of Disk-Resident Queries

For this set of experiments, we use TS and LA alternatively as query (Q) and data (P) sets. We compare F-MQM, F-MBM, and BNL assuming that the main memory can accommodate 50,000 data points (i.e., 196 pages). P is indexed by an R*-tree (except for the case of BNL) and Q is stored as a flat file (LA occupies 5136 and TS 762 pages). The I/O cost corresponds to the sum of R-tree node visits and page accesses for reading Q. When P=LA (1,314,620 points), there are 27 data blocks and the cost of BNL is 5136+27·762, whereas, if P=TS (194,971 points) the number of blocks is 4 and the cost of BNL becomes 762+4·5136 (these costs are independent of the query parameters). F-MQM applies MBM to compute the ANN of each block because, as shown in Section 7.1, it is the most efficient method for memory-resident queries. The I/O in the diagrams also includes the cost of sorting Q (according to Hilbert values) in F-MBM.

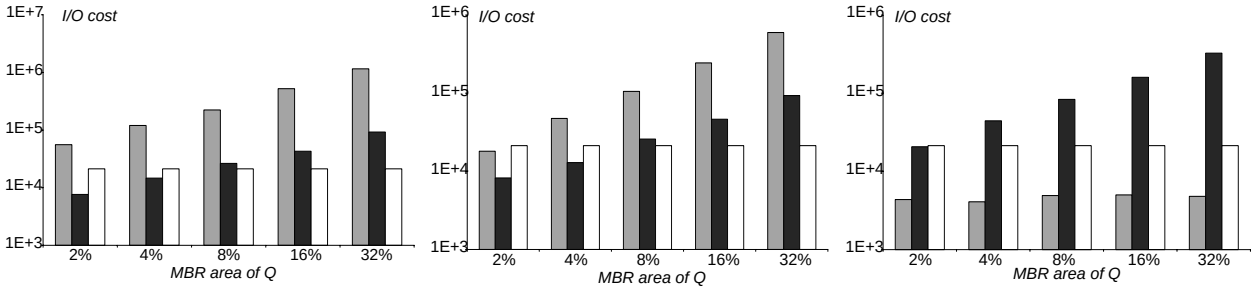
Since now the query cardinality n is fixed to that of the corresponding dataset, we perform experiments by varying the relative data spaces of P and Q. First, we assume that the spaces of P and Q have the same centroid, but the area

A_Q of Q varies between 2% and 32% of the data space of P (similar to the experiments of Figure 34 and 35, but now the query points are enclosed by an MBR instead of a circle). Figure 40 (41) shows the I/O cost as a function of query size assuming that the query dataset is TS (LA) and k=4. In general, BNL has the lowest cost for *sum* and *max* (except for very small A_Q) because it reads the data file only once. F-MQM is the best method for *min* since it retrieves the ANN of the block that corresponds to the minimum threshold, leading to the earliest termination of the algorithm. For *sum*, F-MBM outperforms F-MQM because it can prune nodes N "far" from the query centroid based on Heuristic 4: $sum(amindist(N,M_1), \ldots, amindist(N,M_m)) \geq best\_dist$. Especially, for the case that Q=LA (Figure 41), each of the 27 query blocks occupies a small part of the data space (due to Hilbert sorting), enhancing the pruning power of the heuristic. In case of *max*, the relative performance of the two methods depends on the problem: F-MBM is better for Q=LA (for the reason mentioned above), but worse than F-MQM for Q=TS (since there exist only 4 query blocks and the pruning power of the heuristic is lower).



(a) *f=sum* (b) *f=max* (c) *f=min*

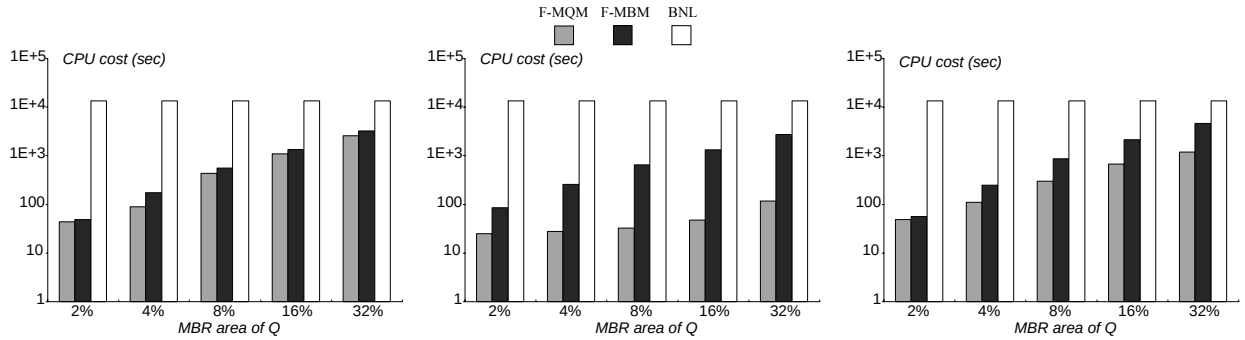
Fig. 40. I/O cost vs. area A_Q of Q (P=LA, Q=TS, k=4)



(a) *f=sum* (b) *f=max* (c) *f=min*

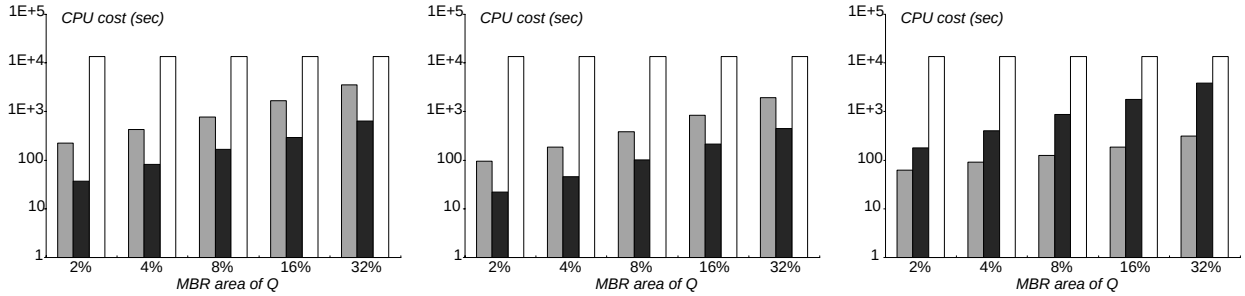
Fig. 41. I/O cost vs. area A_Q of Q (P=TS, Q=LA, k=4)

Figures 42 and 43 compare the CPU costs of the methods for the same settings as Figures 40 and 41, respectively. In this case, BNL is the worst algorithm due to the exhaustive distance computations for all data-query point pairs. Similar to I/O diagrams, F-MQM is again the best method for *min*. For *sum* and *max*, the winner depends on the cardinality of the query dataset: (i) when Q=TS, it is F-MQM since it efficiently combines the ANNs of only 4 blocks (ii) when Q=LA, the winner is F-MBM because of the enhanced pruning power of Heuristic 4 and the higher cost of F-MQM (for combining the results of 27 groups).



(a) *f*=*sum* (b) *f*=*max* (c) *f*=*min*

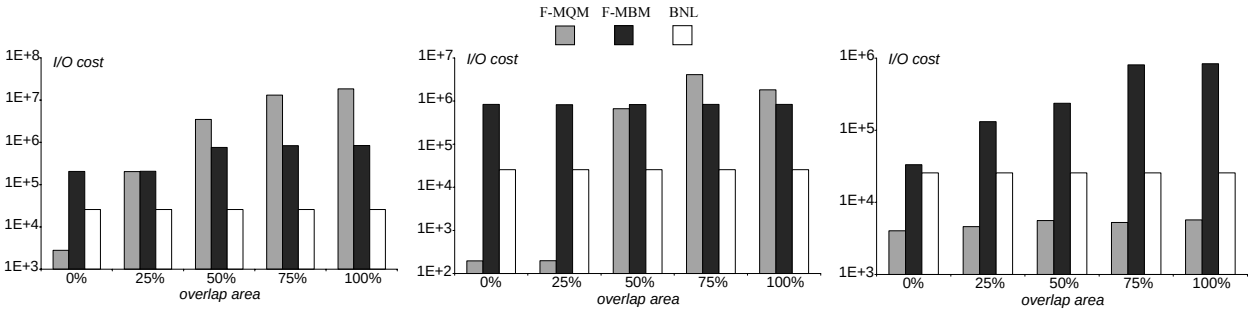
Fig. 42. CPU cost vs. area A_Q of Q (P=LA, Q=TS, k=4)



(a) *f*=*sum* (b) *f*=*max* (c) *f*=*min*

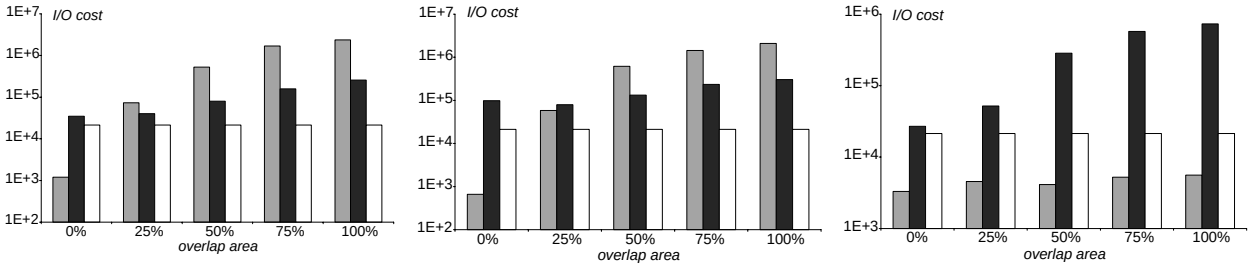
Fig. 43. CPU cost vs. area A_Q of Q (P=TS, Q=LA, k=4)

In order to further investigate the effect of the relative data space positions, for the next set of experiments we assume that both datasets lie in spaces of the same size, and vary their overlap area from 0% (i.e., P and Q are totally disjoint) to 100% (i.e., on top of each other). Intermediate values are obtained by starting from the 100% case and shifting the query dataset on both axes. Figure 44 (45) shows the cost of the algorithms assuming that Q=TS (LA). For *sum* and *max*, F-MQM is the best method for 0% overlap, but its cost grows fast with the overlap area. To explain this, let us consider the 0% overlap case assuming that the query space starts at the upper-right corner of the data space. The nearest neighbors of all query blocks must lie near this upper-right corner, since such points minimize the aggregate distance. Therefore, F-MQM can find the best ANN relatively fast, and terminate when all the points in or near the corner have been encountered. On the other hand, as the overlap area increases, so does the number of data points that need to be considered. F-MBM is not very sensitive to the overlap area since, intuitively, its performance depends more on the relative position of R-tree nodes with respect to the query blocks. BNL is the best algorithm for *sum* and *max* in the case of overlap areas exceeding 25%, whereas F-MQM is the winner in the remaining settings.



(a) *f=sum* (b) *f=max* (c) *f=min*

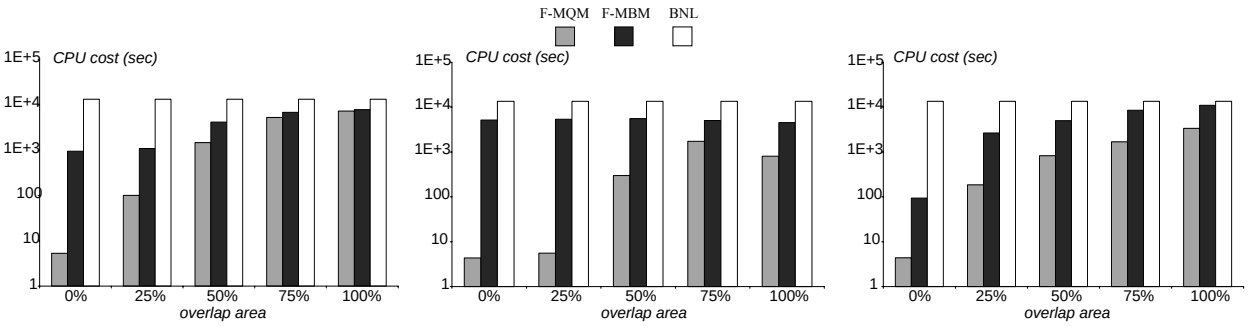
Fig. 44. I/O cost vs. overlap area (*P*=LA, *Q*=TS, *k*=4)



(a) *f=sum* (b) *f=max* (c) *f=min*

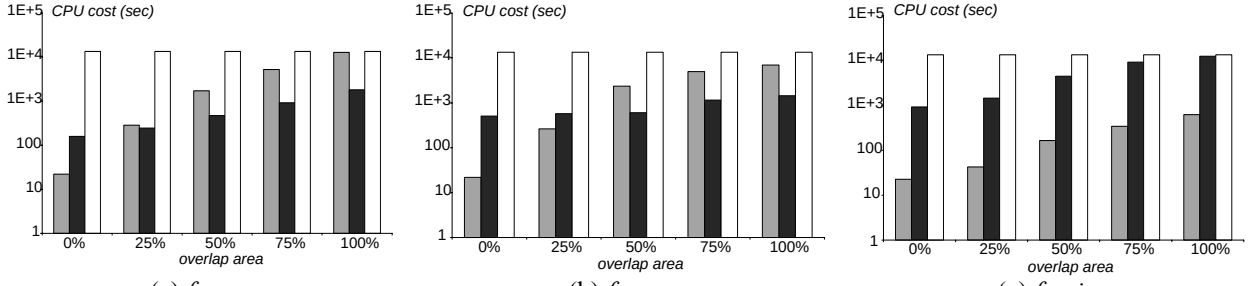
Fig. 45. I/O cost vs. overlap area (*P*=TS, *Q*=LA, *k*=4)

Figures 46 and 47 show the CPU cost vs. the overlap area. Similar to Figures 42 and 43, BNL has the highest CPU overhead and F-MQM the lowest cost for *min*. For *sum* and *max* the winner is F-MQM if *Q*=TS, and F-MBM when *Q*=LA. We also performed experiments by varying the number of neighbors retrieved, while keeping the other parameters fixed. As in the case of main-memory queries, *k* does not have a significant effect on the performance (and the diagrams are omitted).



(a) *f=sum* (b) *f=max* (c) *f=min*

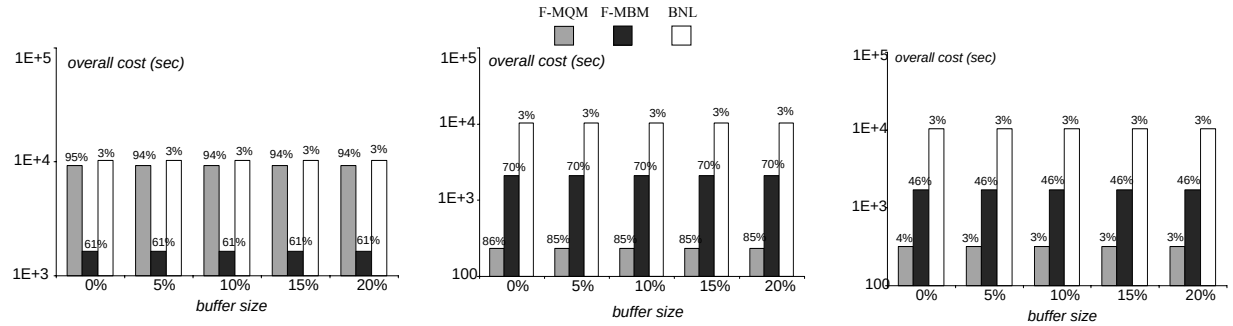
Fig. 46. CPU cost vs. overlap area (*P*=LA, *Q*=TS, *k*=4)



(a) *f=sum* (b) *f=max* (c) *f=min*

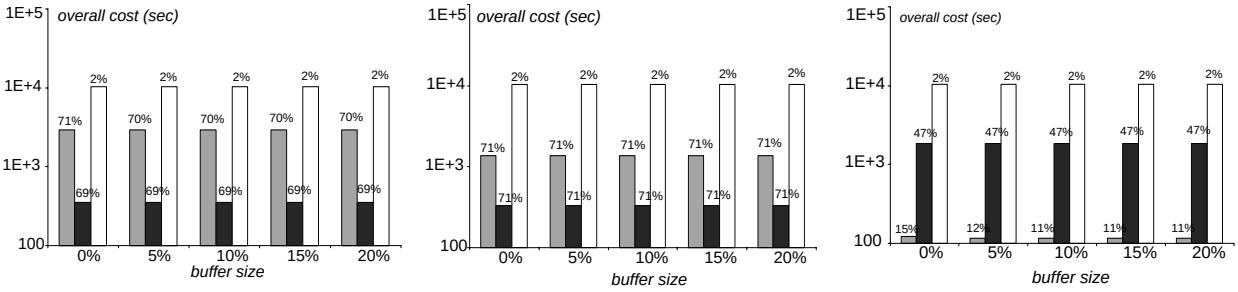
Fig. 47. CPU cost vs. overlap area (*P*=TS, *Q*=LA, *k*=4)

In Figure 48 (49) we measure the overall cost (including I/O and CPU time) in the presence of an LRU buffer, for P=LA (TS), Q=TS (LA) and k=4. The spaces of P and Q have the same centroid and A_Q is set to 8%. The buffer is dedicated to disk pages of P. Caching is pointless for Q, because it is always scanned linearly. Similar to Figures 38 and 39, the buffer size varies between 0% and 20% of the total number of pages in P, and each page fault corresponds to 10 milliseconds. The percentage above each column indicates the I/O overhead. The performance of F-MBM (BNL) is independent of the buffer size, because it accesses each page of P at most once (exactly once). On the other hand, F-MQM improves slightly when a buffer is available, but the enhancement is negligible because its I/O cost is largely dominated by reading the query set Q. As opposed to algorithms for memory-resident queries, which are I/O bound, for disk resident queries often the CPU-time corresponds to a large part of the total cost (i.e., 97-98% for BNL, 85-97% for F-MQM and the *min* function).



(a) *f=sum* (b) *f=max* (c) *f=min*

Fig. 48. Overall cost vs. buffer size (P=LA, Q=TS, k=4, A_Q=8%)



(a) *f=sum* (b) *f=max* (c) *f=min*

Fig. 49. Overall cost vs. buffer size (P=TS, Q=LA, k=4, A_Q=8%)

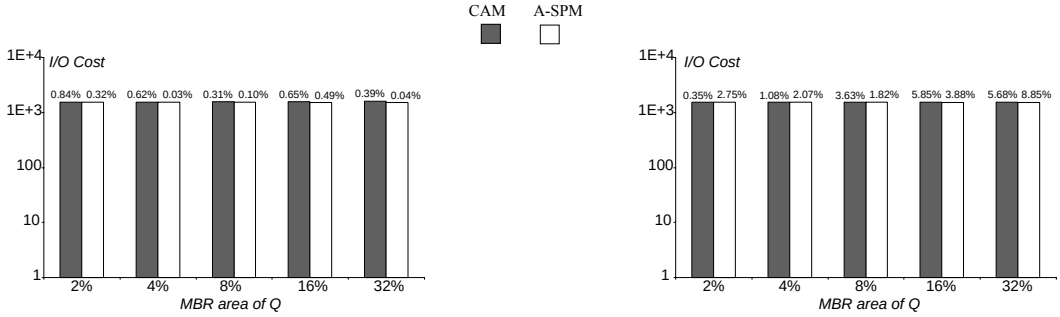
In summary, unlike the case of main-memory queries, where there is a clear winner (MBM), the best method for disk-resident queries depends on the problem characteristics. F-MQM is always preferable for *min*, and performs well for *max* given few query blocks. F-MBM is best for *sum* and to a lesser extend for *max*, especially for large query sets. BNL has balanced performance and could be the method of choice for *sum* and *max*, if the data and query sets cover the same space. An important observation from the above experiments refers to the high cost of the disk-resident queries, which motivates the approximate algorithms, evaluated in the next section.

7.3 Evaluation of Approximate Algorithms

For approximate algorithms, in addition to performance, we have to evaluate the quality of approximation. Let p'_{best} be the k^{th} ANN retrieved by an approximate algorithm and $adist'_{best}$ be its aggregate distance. We define the error as:

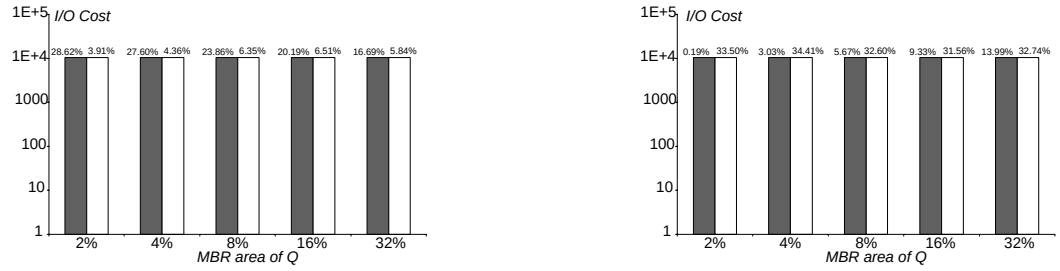
($adist'_{best}$ - $adist_{best}$)/$adist_{best}$, where $adist_{best}$ is the aggregate distance of the actual k^{th} ANN (note that $adist'_{best} \geq adist_{best}$). CAM and A-SPM are compared on disk-resident queries using the datasets LA and TS alternatively as query points. Both algorithms read Q twice: the first time for computing Q' (CAM) or the geometric centroid (A-SPM) and the second time for computing the actual aggregate distance of the retrieved ANNs.

Figures 50 and 51 illustrate the I/O cost as a function of the area A_Q of Q. Similar to Figures 40 and 41, we assume that the data spaces of P and Q have the same centroid, but the MBR of Q ranges between 2% and 32% of the data space of P. The two methods have almost the same overhead because their I/O cost is dominated by reading Q and the node accesses to the data R-tree have little effect on performance. The diagram also includes the error of the algorithms at the top of the corresponding column. A-SPM incurs small error for *sum* (maximum value 6.51%) because the *SR* can be well approximated by a circle around the geometric centroid. The quality of approximation is lower for Q=LA due to the higher skeweness of the dataset (compared to TS). On the other hand, CAM is better for *max* because the *SR* has a more irregular shape (see Figure 17).



(a) *f=sum* (b) *f=max*

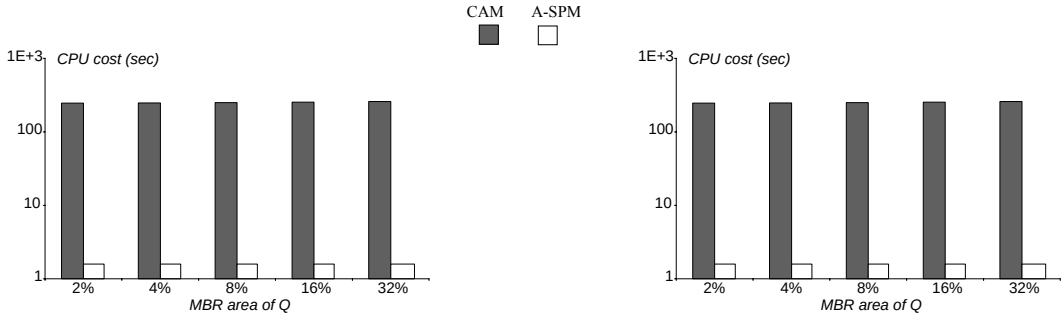
Fig. 50. I/O cost vs. area A_Q of Q (P=LA, Q=TS, k=4)



(a) *f=sum* (b) *f=max*

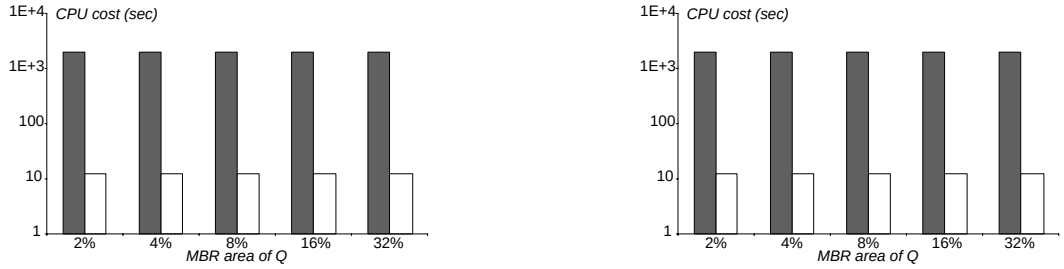
Fig. 51. I/O cost vs. area A_Q of Q (P=TS, Q=LA, k=4)

The last experiment measures the CPU time of the approximate algorithms. A-SPM is better than CAM because finding the Euclidean NNs of q involves processing just a few nodes, whereas coalescing query points is CPU-intensive. Especially for Q=LA (Figure 53), the CPU overhead of CAM exceeds that of the best exact algorithm (Figure 43) and the method is ineffective. In general, A-SPM is far more efficient (and, for *sum*, more accurate). CAM should be used only for *max*, provided that the query dataset is relatively small.



(a) *f=sum* (b) *f=max*

Fig. 52. CPU cost vs. area A_Q of Q (P=LA, Q=TS, k=4)



(a) *f=sum* (b) *f=max*

Fig. 53. CPU cost vs. area A_Q of Q (P=TS, Q=LA, k=4)

8. CONCLUSIONS

In this paper we propose the novel problem of aggregate nearest neighbor retrieval, a generalized form of NN search, where there are multiple query points and the optimization goal depends on an input function. ANN is important both as a standalone query type in spatial applications (e.g., GIS, VLSI), as well as a module for efficient clustering-related methods. We provide algorithms for memory-resident queries and cost models that accurately predict their performance in terms of node accesses. As a second step, we also develop methods for disk-resident query sets and approximate retrieval. We evaluate all the proposed techniques through extensive experiments.

In the future we intend to explore the application of related techniques to variations of ANN search. Consider, for instance, that Q represents a set of facilities and the goal is to assign each object of P to a single facility so that the sum of distances (of each object to its nearest facility) is minimized. Additional constraints (e.g., a facility may serve at most k users) may further complicate the solutions. Similar problems have been studied in the context of clustering and resource allocation, but the existing methods are different from the ones presented in this paper (as they do not rely on spatial indexing). Finally, we plan to study analytically ANN retrieval for disk-resident sets. In this case, the derivation of cost models is more complex than memory-resident queries because of the multiple reads of Q required by the processing techniques.

REFERENCES

ACHARYA, S., POOSALA, V., AND RAMASWAMY, S. 1999. Selectivity Estimation in Spatial Databases. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, Philadelphia, PA, June 1999, 13-24.

AGGRAWAL, C., AND YU, P. 2001. Outlier Detection for High Dimensional Data. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, Santa Barbara, CA, May 2001, 37-47.

ARYA, S., MOUNT, D., NETANYAHU, N., SILVERMAN, R., AND WU, A. 1998. An Optimal Algorithm for Approximate Nearest Neighbor Searching. *Journal of the ACM*, 45(6), 891-923.

BECKMANN, N., KRIEGEL, H., SCHNEIDER, R., AND SEEGER, B. 1990. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, Atlantic City, NJ, May 1990, 322-331.

BENETIS, R., JENSEN, C., KARCIAUSKAS, G., AND SALTENIS, S. 2002. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *Proceedings of International Database Engineering and Applications Symposium (IDEAS)*, Edmonton, Canada, July 2002, 44-53.

BERCHTOLD, S., BOHM, C., KEIM, D.A., AND KRIEGEL, H. 1997. A Cost Model for Nearest Neighbor Search in High-Dimensional Data Space. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Tucson, AZ, May 1997, 78-86.

BEYER, K., GOLDSTEIN, J., RAMAKRISHNAN, R., AND SHAFT, U. 1999. When Is Nearest Neighbor Meaningful? In *Proceedings of International Conference on Database Theory (ICDT)*, Jerusalem, Israel, January 1999, 217-235.

BOHM, C. 2000. A Cost Model for Query Processing in High Dimensional Data Spaces. *ACM Transactions on Database Systems*, 25(2), 129-178.

BRUNO, N., CHAUDHURI, S., AND GRAVANO, L. 2002. Top-k Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation. *ACM Transactions on Database Systems*, 27(2), 153-187.

CHANG, Y., BERGMAN, L., CASTELLI, V., LI, C., LO, M., AND SMITH, J. 2000. The Onion Technique: Indexing for Linear Optimization Queries. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, Dallas, TX, May 2000, 391-402.

CHEUNG, K., AND FU, A. 1998. Enhanced Nearest Neighbour Search on the R-tree. *SIGMOD Record* 27(3), 16-21.

CORRAL, A., MANOLOPOULOS, Y., THEODORIDIS, Y., AND VASSILAKOPOULOS, M. 2000. Closest Pair Queries in Spatial Databases. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, Dallas, TX, May 2000, 189-200.

FAGIN, R. 2002. Combining Fuzzy Information: an Overview. *SIGMOD Record*, 31(2), 109-118.

FAGIN, R., LOTEM, A., AND NAOR, M. 2001. Optimal Aggregation Algorithms for Middleware. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Santa Barbara, CA, May 2001, 102-113.

GUTTMAN, A. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, Boston, MA, June 1984, 47-57.

JAIN, A., MURTHY, M., AND FLYNN, P., 1999. Data Clustering: A Review. *ACM Comp. Surveys*, 31(3), 264-323.

HENRICH, A. 1994. A Distance Scan Algorithm for Spatial Access Structures. In *Proceedings of ACM Workshop on Geographic Information Systems (ACM GIS)*, Gaithersburg, MD, December 1994, 136-143.

HJALTASON, G., AND SAMET, H. 1998. Incremental Distance Join Algorithms for Spatial Databases. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, Seattle, WA, June 1998, 237-248.

HJALTASON, G., AND SAMET, H. 1999. Distance Browsing in Spatial Databases. *ACM Transactions on Database Systems*, 24(2), 265-318.

HOCHREITER, S., YOUNGER, A.S., AND CONWELL, P. 2001. Learning to Learn Using Gradient Descent. In *Proceedings of International Conference on Artificial Neural Networks (ICANN)*, Vienna, Austria, August 2001, 87-94.

HRISTIDIS, V., AND PAPAKONSTANTINOU, Y. 2004. Algorithms and Applications for Answering Ranked Queries using Ranked Views. *VLDB Journal*, 13(1), 49-70.

KOLAHDOUZAN, M., AND SHAHABI, C. 2004. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *Proceedings of Very Large Data Bases Conference (VLDB)*, Toronto, Canada, August 2004, 840-851.

KOLLIOS, G., GUNOPULOS, D., AND TSOTRAS, V. 1999. Nearest Neighbor Queries in Mobile Environment. In *International Workshop on Spatio-Temporal Database Management (STDBM)*, Edinburgh, Scotland, September 1999, 119-134.

KORN, F., PAGEL, B., AND FALOUTSOS, C. 2001. On the "Dimensionality Curse" and the "Self-Similarity Blessing". *IEEE Transactions on Knowledge and Data Engineering*, 13(1), 96-111.

LIU, X., AND FERHATOSMANOGLU, H. 2003. Efficient k-NN Search on Streaming Data Series. In *Proceedings of International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, Santorini Island, Greece, July 2003, 83-101.

NAKANO, K., AND OLARIU, S. 1997. An Optimal Algorithm for the Angle-Restricted All Nearest Neighbor Problem on the Reconfigurable Mesh, with Applications. *IEEE Transactions on Parallel and Distributed Systems*, 8(9), 983-990.

NG, R., AND HAN, J. 1994. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of Very Large Data Bases Conference (VLDB)*, San Francisco, CA, September 1994, 144-155.

PAPADIAS, D., SHEN, Q., TAO, Y., AND MOURATIDIS, K. 2004. Group Nearest Neighbor Queries. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, Boston, MA, March 2004, 301-312.

PAPADIAS, D., ZHANG, J., MAMOULIS, N., AND TAO, Y. 2003. Query Processing in Spatial Network Databases. In *Proceedings of Very Large Data Bases Conference (VLDB)*, Berlin, Germany, September 2003, 802-813.

PAPADOPOULOS, A., AND MANOLOPOULOS, Y. 1997. Performance of Nearest Neighbor Queries in R-trees. In *Proceedings of International Conference on Database Theory (ICDT)*, Delphi, Greece, January 1997, 394-408.

PRESS, W., FLANNERY, B., TEUKOLSKY, S., AND VETTERLING, W. 2002. Numerical Recipes in C++ (second edition). *Cambridge University Press*, ISBN 0-521-75034-2.

ROUSSOPOULOS, N., KELLY, S., AND VINCENT, F. 1995. Nearest Neighbor Queries. In *Proceedings of ACM Conference on the Management of Data (SIGMOD)*, San Jose, CA, May 1995, 71-79.

SAKURAI, Y., YOSHIKAWA, M., UEMURA, S., AND KOJIMA, H. 2000. The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. In *Proceedings of Very Large Data Bases Conference (VLDB)*, Cairo, Egypt, September 2000, 516-526.

SHAN, J., ZHANG, D., AND SALZBERG, B. 2003. On Spatial-Range Closest-Pair Query. In *Proceedings of International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, Santorini Island, Greece, July 2003, 252-269.

SONG, Z., AND ROUSSOPOULOS, N. 2001. K-Nearest Neighbor Search for Moving Query Point. In *Proceedings of International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, Redondo Beach, CA, July 2001, 79-96.

SPROULL, R. 1991. Refinements to Nearest Neighbor Searching in K-Dimensional Trees. *Algorithmica*, 6(4), 579-589.

TAO, Y., AND PAPADIAS, D. 2003. Spatial Queries in Dynamic Environments. *ACM Transactions on Database Systems*, 28(2), 101-139.

TAO, Y., ZHANG, J., PAPADIAS, D., AND MAMOULIS, N. 2004. An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. *IEEE Transactions on Knowledge and Data Engineering*, 16(10), 1169-1184.

THEODORIDIS, Y., STEFANAKIS, E., AND SELLIS, T. 2000. Efficient Cost Models for Spatial Queries Using R-trees. *IEEE Transactions on Knowledge and Data Engineering*, 12(1), 19-32.

WEBER, R., SCHEK, H.J., AND BLOTT, S. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of Very Large Data Bases Conference*

(VLDB), New York, NY, August 1998, 194-205.

WELZL, E. 1991. Smallest Enclosing Disks (Balls and Ellipsoids). In *New Results and New Trends in Computer Science*, Springer Verlag LNCS, 555, 359-370.

WESOLOWSKY, G. 1993. The Weber problem: History and perspectives. *Location Science*, 1(1), 5-23.

YU, C., OOI, B, TAN, K., AND JAGADISH, H. 2001. Indexing the Distance: An Efficient Method to KNN Processing. In *Proceedings of Very Large Data Bases Conference (VLDB)*, Rome, Italy, September 2001, 421-430.

YU, X., PU, K., AND KOUDAS, N. 2005. Monitoring K-Nearest Neighbor Queries Over Moving Objects. To appear in *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, Tokyo, Japan, April 2005.

XIONG, X., MOKBEL, M., AND AREF, W. 2005. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. To appear in *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, Tokyo, Japan, April 2005.

Received July 2004; revised December 2004; accepted December 2004.