Group Enclosing Queries

Feifei Li, Bin Yao, Piyush Kumar

Abstract—Given a set of points P and a query set Q, a *group enclosing query* (GEQ) fetches the point $p^* \in P$ such that the maximum distance of p^* to all points in Q is minimized. This problem is equivalent to the Min-Max case (minimizing the maximum distance) of aggregate nearest neighbor queries for spatial databases [27]. This work first designs a new exact solution by exploring new geometric insights, such as the minimum enclosing ball, the convex hull and the furthest voronoi diagram of the query group. To further reduce the query cost, especially when the dimensionality increases, we turn to approximation algorithms. Our main approximation algorithm has a worst case $\sqrt{2}$-approximation ratio if one can find the exact nearest neighbor of a point. In practice, its approximation ratio never exceeds 1.05 for a large number of data sets up to six dimension. We also discuss how to extend it to higher dimensions (up to 74 in our experiment) and show that it still maintains a very good approximation quality (still close to 1) and low query cost. In fixed dimensions, we extend the $\sqrt{2}$-approximation algorithm to get a $(1+\epsilon)$-approximate solution for the GEQ problem. Both approximation algorithms have $O(\log N + M)$ query cost in any fixed dimension, where N and M are the sizes of the data set P and query group Q. Extensive experiments on both synthetic and real data sets, up to 10 million points and 74 dimensions, confirm the efficiency, effectiveness and scalability of the proposed algorithms, especially their significant improvement over the state-of-the-art method.

Index Terms—Aggregate nearest neighbor, approximate nearest neighbor, minmax nearest neighbor, nearest neighbor

1 Introduction

Aggregate nearest neighbor queries represent a class of important query types that are defined by taking the minimum over the combination of some aggregate operators, such as the SUM, AVG, MAX, and the fundamental nearest neighbor search [27]. These queries are defined over two data sets, the data set P and a query data set Q. Such queries could be generally interpreted as a minimizing optimization problem defined on the nearest neighbor search. For example, when the aggregate operator is the SUM, the aggregate nearest neighbor query is equivalent to the group nearest neighbor query [26] where the goal is to find a point from P that has the minimum SUM distance to all query points in Q. Papadias et al. have given a complete, thorough treatment to a number of aggregate nearest neighbor queries in spatial databases [27]. While being general enough to cover different aggregate operators, its generality also means that important opportunities could be overlooked to optimize query algorithms for specific operators. For instance, the state of the art [27] is limited by heuristics that may yield very high query cost in certain cases, especially for data sets and queries in higher (more than two) dimensions. Motivated by this observation, this work focuses on one specific aggregate operator, namely the MAX, for the aggregate nearest neighbor queries in large databases and designs methods that significantly reduce the query cost compared to the MBM (Minimum Bounding Method) algorithm from [27]. Following the

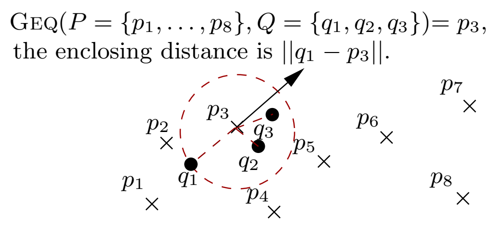

GEQ($P = \{p_1, \dots, p_8\}, Q = \{q_1, q_2, q_3\}$)= p_3,
the enclosing distance is $||q_1 - p_3||$.
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
q_1
q_2
q_3

Fig. 1. A group enclosing query example.

previous instance when studying a specific aggregate type for aggregate nearest neighbor queries (e.g., group nearest neighbor queries for the SUM operator [24], [26]), we designate a query name, the group enclosing query (GEQ), for an aggregate nearest neighbor query with the MAX operator. An example of the GEQ problem is illustrated in Figure 1.

Specifically, with the reference to Figure 1, suppose the cross points are the database (denoted as P) and the points in black-filled circles form the group of query points (denoted as Q), our goal in GEQ is to find one point $p* \in P$ such that $p*$ has the smallest distance to enclose all points from Q, where using a point $p \in P$ to enclose Q refers to finding a ball centered at p with the smallest possible radius to cover all points from Q. For example, the answer to the GEQ instance in Figure 1 is p_3 as indicated. In general, we assume that $|P| = N$ and $|Q| = M$. This query has many important and practical applications, for example:

Example 1 A group of people is trying to meet in one place. Given a large number of candidates, their goal is to minimize the longest distance traveled by anyone.

More applications could be listed to demonstrate the usefulness of this query and more examples are available from the prior study [27]. Intuitively, in contrast to many

• *All authors are with the Computer Science Department at Florida State University, Tallahassee, FL 32306-4530. E-mail:* `{lifeifei,yao,piyush}@cs.fsu.edu`. *Feifei Li and Bin* *Yao were partially supported by NSF Grant IIS-0916488. Piyush Kumar was partially supported by NSF through CAREER Grant CCF-0643593 and the Air Force Young Investigator Program.*

existing variants of the nearest neighbor queries that ask for the best answer of the average case [24], [26], [36], the GEQ problem searches for the best answer of the worst case. This problem becomes even more difficult if the query group is large as well.

The state of the art method for the GEQ problem is the Minimum Bounding Method (MBM) from [27]. The principal methodology adopted by the MBM is the triangle inequality. It is a heuristic method that has $O(N+M)$ query cost. In practice, its query cost has only been studied in the two-dimensional space. Our experiments over a large number of data sets in Section 7 suggests that the MBM algorithm may lead to high query cost for large data sets and more importantly, its performance degrades significantly with the increase of the dimensionality.

In fact, it is easy to see that any exact search method for the GEQ problem will inevitably suffer from the curse of the dimensionality, since the classical nearest neighbor search is a special instance of the GEQ problem (when Q has only one point). Hence, for data sets in high dimensions, similar to the motivation of doing approximate nearest neighbor search instead of retrieving the exact nearest neighbor in high dimensions [15], [20] (where almost all exact methods degrade to the expensive linear scan of the entire data set), finding efficient and effective approximation algorithms is the best alternative.

Our Contributions. This work presents new, efficient algorithms, including both exact and approximate versions, for the GEQ problem that significantly outperform the state of the art, the MBM algorithm. Specifically,

- We present a new exact search method for the GEQ problem in Section 4 that instantiates several new geometric insights, such as the minimum enclosing ball, the convex hull and the furthest voronoi diagram of the query group, to achieve higher pruning power than the MBM approach.
- We design a $\sqrt{2}$-approximation (worst case approximation ratio in *any dimensions*) algorithm in Section 5.1, if one can find the exact nearest neighbor of a point and the minimum enclosing ball of Q. Its asymptotic query cost is $O(\log N + M)$ in *any fixed dimensions*. Our idea is to reduce the GEQ problem to the classical nearest neighbor search by utilizing the center of the minimum enclosing ball for Q.
- We extend the above idea to a $(1+\epsilon)$-approximation algorithm in any fixed dimension in Section 5.2. This algorithm has a strong theoretical interest and it also achieves the optimal $O(\log N + M)$ query cost in any fixed dimension.
- We extend the same idea from the $\sqrt{2}$-approximate algorithm to much higher dimensions in Section 5.3, since it is impossible to find the exact nearest neighbor *efficiently* and the exact minimum enclosing ball in high dimensions in practice. We leverage on the latest, practical approximate nearest neighbor search method (the LSB-tree [31]) and the $(1+\epsilon)$-approximate minimum enclosing ball algorithm [22] in high dimensions. It shows that we can still obtain an efficient $(\sqrt{5}+\sqrt{2}\epsilon)$-approximation (with constant probability) in high dimensions that works extremely well in practice.
- We discuss the challenges when Q becomes large and disk-based in Section 6.1, and show how to adapt our algorithms to handle this case efficiently. We also present an interesting variation of the GEQ problem, the constrained GEQ, in Section 6.2.
- We demonstrate the efficiency, effectiveness and scalability of our algorithms with extensive experiments in Section 7. These results show that both our exact and approximate methods have significantly outperformed the MBM method up to 6 dimensions. Beyond 6 dimensions and up to very high dimensions ($d = 74$), our approximate algorithm is still efficient and effective, with an average approximation ratio that is close to 1 and very low IO cost.

Symbol	Description
$\|\|p-q\|\|$	Euclidean distance between p and q
$\|\cdot\|$	Size of a set
$\subset, \subseteq$	Both set and geometric containment
$\mathcal{B}(c,r)$	The ball centered at c with radius r
$\mathcal{C}_Q$	Q's convex hull as an ordered set of vertices and the corresponding convex polygon
fn(q,P)	The furthest neighbor of q in P
FC(q)	The furthest voronoi cell of q
GEQ (P,Q)	GEQ instance on P and Q
MEB(Q)	Minimum enclosing ball of Q
MEB(p,Q)	Minimum ball centers at p containing Q
maxEdist	MBR u's *max* enclosing distance to Q
minEdist	MBR u's *min* enclosing distance to Q
nn(q,P)	The nearest neighbor of q in P
N, M	$\|P\|$ and $\|Q\|$ respectively
p_1p_2	A line segment between p_1 and p_2
$p_1p_2\cdots p_tp_1$	The convex polygon by $\{p_1p_2,\ldots,p_tp_1\}$
p^*	The optimal answer to GEQ (P,Q)
r_p	The enclosing distance of p w.r.t Q
r^*	GEQ (P,Q)'s best enclosing distance (r_{p^*})

TABLE 1
Notation used.

We formalize the problem in Section 2, survey the related work in Section 3 and conclude in Section 8.

2 PROBLEM FORMULATION

Let P and Q denote the database and the query group respectively, d-dimensional Euclidean space. The Euclidean distance between two points p and q is denoted by $||p-q||$. Let $\mathcal{B}(c,r)$ denote the ball centered at c with radius r. We use MEB(Q) to denote the minimum enclosing ball for a set of points Q. MEB(Q) is the smallest (measured by r) ball $\mathcal{B}(c,r)$ such that $Q \subseteq \mathcal{B}(c,r)$. In this context, the operator $\subseteq$ refers to the geometric containment, i.e., every point in Q is fully contained by the ball defined by $\mathcal{B}(c,r)$. If the center is fixed at p, MEB(p,Q) denotes the smallest ball centered at p that encloses all points in Q. The enclosing distance of p w.r.t Q is $r_{p,Q} = \max_{q\in Q} ||p-q||$. Immediately, $r_{p,Q}$ is the radius of MEB(p,Q). In the sequel, the subscript Q in $r_{p,Q}$ will be omitted when the context is clear. An ordered list of points $p_1p_2\cdots p_tp_1$ represents a convex polygon defined by line segments $\{p_1p_2, p_2p_3, \ldots, p_tp_1\}$ with t vertices. The convex hull of Q is represented by a subset of its vertices as an ordered set of points $\mathcal{C}_Q = \{q_{c_1}, \ldots, q_{c_x}\}$ where $q_{c_i} \in Q$ for some $x \in [1, |Q|]$. The convex polygon associated with the convex hull of Q is simply $q_{c_1}q_{c_2}\cdots q_{c_x}q_{c_1}$ and it is also denoted by $\mathcal{C}_Q$. The nearest neighbor of q in the point set P is denoted as $\mathrm{nn}(q, P)$. Finally, let $|P| = N$ and $|Q| = M$ where $|\cdot|$ is the cardinality of a set. Table 1 provides a quick reference to our main notations.

Group Enclosing Query (GEQ). For a set of points $P = \{p_1, p_2, \ldots, p_N\}$ and a query set $Q = \{q_1, q_2, \ldots, q_M\}$, find the point $p^* \in P$ such that the radius r^* of the ball $\mathrm{MEB}(p^*, Q)$ is minimized, i.e., the enclosing distance of p^* w.r.t Q is minimal among all points in P. Formally, it amounts to the following optimization problem:

$$r^* := \min_{p \in P} \max_{q \in Q} ||p - q|| \tag{1}$$

Ties are broken arbitrarily in Equation 1. It follows that p^* and r^* are the center and radius of the minimum enclosing ball of Q centered at point $p^* \in P$. Henceforth, we use GEQ (P, Q) to denote an instance of this problem. Note that this problem can be trivially solved in $O(NM)$ cost. It can also be thought of as finding the minimum enclosing ball of Q subject to the constraint that the center can lie only on one of the points in P.

Given any value $\epsilon > 0$ (or a constant a), we say that $(p, r_p) \in \mathbb{R}^d \times \mathbb{R}$ is a $(1+\epsilon)$-approximate (or a-approximate) solution to the GEQ problem for the instance on (P, Q) if and only if

$$r^* \le r_p \le (1+\epsilon)r^* \text{ (or } r^* \le r_p \le ar^*) \tag{2}$$

In the rest of this paper, for the ease of illustration, some of our running examples in Figures will be based on two dimensions, but our discussion and algorithms apply to data sets and query sets in any fixed dimension.

3 Background and Related Work

R-tree [17] and its variants (e.g., R*-tree [8]) have been the most widely deployed indexing structure for the spatial database, or data in multi-dimensions in general. Intuitively, R-tree is an extension of the B-tree to higher dimensions. Points are grouped into minimum bounding rectangles (MBRs) which are recursively grouped into MBRs in higher levels of the tree. The grouping is based on data locality and bounded by the page size. An example of the R-tree is illustrated in Figure 2. Two important query types that we leverage on R-tree are nearest neighbor (NN) queries and range queries.

NN search has been extensively studied [5], [12], [15], [19]–[21], [30] and many related works therein. In particular R-tree demonstrates efficient algorithms using either the depth-first [30] and the best-first [19] approach. The main idea behind these algorithms is to utilize branch and bound pruning techniques based on the relative

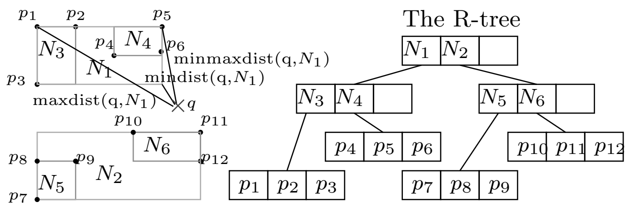


Fig. 2. The R-tree.

distances between a query point q to a given MBR N (e.g., minmaxDist, minDist, see Figure 2).

Range query is to return all points that are fully contained by the query rectangle (or the query ball). R-tree has good performance for answering range queries in practice [10], [28], [32], for data sets in relatively low dimensions. The basic idea is to search through all MBRs that intersect with the query rectangle (or ball).

Unfortunately, the worst case query costs are not logarithmic when R-tree is used for NN or range search (even for approximate versions of these queries). To design theoretically sound algorithms with logarithmic costs for our problem, we need a space partition tree with the following properties : (1) The tree has $O(N)$ nodes, $O(\log N)$ depth and each node of the tree is associated with at least one data point. (2) The cells have bounded aspect ratio. (3) The distance of a point to a cell of the tree can be computed in $O(1)$. Arya et.al [5] designed a modification of the standard kd-tree called the Balanced Box Decomposition (BBD) tree which satisfies all these properties and hence can answer $(1+\epsilon)$-approximate nearest neighbor queries in $O((1/\epsilon^d)\log N)$ and $(1+\epsilon)$-approximate range search queries in $O((1/\epsilon^d) + \log N)$. BBD-tree takes $O(N \log N)$ time to build. We use BBD trees in the design of the optimal $(1+\epsilon)$-approximation algorithm with the logarithmic query complexity for solving the GEQ problem.

For nearest neighbor search in high dimensions, all exact methods will eventually degrade to the expensive linear scan of the entire data set and one has to adopt efficient and effective approximate algorithms [15], [20]. The BBD-tree also becomes impractical for large data sets in high dimensions. In this case, we could use the iDistance [21], [34] index for exact nearest neighbor retrieval (in still relatively low dimensions), or Medrank [13] and LSH-based methods (locality sensitive hashing) [15], [20] (e.g., the latest development represented by the LSB-tree [31]) for the approximate versions in very high dimensions. Since our idea in designing the approximate algorithms for solving the GEQ problem is to reduce it to the basic nearest neighbor search problem, our approach could leverage on all these techniques for the nearest neighbor search and benefit by *any advancement* in this topic. This is a very appealing feature of our approximation algorithm and makes it extremely flexible and simple for adaptation.

The most related work to our study include the the group nearest neighbor (GNN) query [24], [26] and the aggregate nearest neighbor queries [27]. The goal there is to find a point that minimizes the sum distance to all query points. The aggregate nearest neighbor queries [27] expand the study for the GNN query to other aggregate operators, including the MAX that is equivalent to the GEQ problem and it represents the state of the art study on this problem.

The MBM method developed in [27] for $\max$-GNN is an R-tree based approach. It recursively visits nodes beginning from the root of the R-tree and prunes the nodes that cannot contain any results by *sequentially* using two pruning conditions. Let M be the MBR of the query group Q, N be the current node (and its MBR) from the R-tree, and *bestDist* be the enclosing distance of the best $\max$-GNN found so far. The two pruning conditions are minDist$(N, M) \geq$ bestDist, and $\max_{q_i \in Q}$(minDist$(N, q_i)) \geq$ bestDist. Either condition can safely prune the node N (and its subtree). The second condition is applied only for nodes that pass the first one in order to save unnecessary computations. For more details, please refer to Section 3.3 in [27]. However, as we have argued in Section 1, the MBM method only explores the triangle inequality and important pruning and optimization opportunities are missed, as we will show in this work. It is a heuristic method that has $O(N + M)$ query cost theoretically.

The insights in our algorithm development are completely disjoint from those for the MBM method. We first introduce a new exact method that has a much lower query cost than the MBM method, and scale well in 2 to 6 dimensions (in contrast to the subpar scalability of the MBM method w.r.t. the dimensionality). The key insights of our approach are to explore the pruning provided by the convex hull of Q, the minimum enclosing ball of Q, and the furthest voronoi cells of Q. We also explore high-quality approximate algorithms, that come with the theoretically bounded logarithmic query cost and close to 1 approximation ratios in any fixed dimension. The key insight of our approximate algorithm is the close relationship between the center of the minimum enclosing ball of Q and the optimal enclosing point for Q. Our practical approximation algorithm can be easily extended to high dimensions and still maintains its good approximation quality.

Our study is also related to the study in [25] where the goal is to compute medoids in large spatial datasets. Specifically, in the $\max$ and bi-chromatic case in this work, k points in P are selected (as medoids) so that the maximum distance between any point in Q and its closest medoid is (approximately) minimized. Hence, when $k = 1$, this is an approximate version of GEQ.

4 R-TREE BASED EXACT ALGORITHM

This section gives a new R-tree based exact algorithm for the GEQ problem.

4.1 Pruning with Q's convex hull

The convex hull $\mathcal{C}_Q$ of Q is the smallest convex set that contains Q. We represent $\mathcal{C}_Q$ as an *ordered set* of vertices that is a subset of Q, and also use $\mathcal{C}_Q$ to denote the convex polygon defined by these vertices. For example, in Figure 3, $\mathcal{C}_Q = \{q_1, q_2, q_3, q_4, q_5\}$ and $\mathcal{C}_Q$ also refers to the polygon $q_1q_2q_3q_4q_5q_1$ enclosing the shaded area. Yao et al. showed that the furthest point from Q to a point p is amongst the vertices in $\mathcal{C}_Q$ [33]. Formally,

Lemma 1 *[From [33]] Given a set of points Q and its convex hull $\mathcal{C}_Q$, for any point p, let q_f =fn(p, Q), then $q_f \in \mathcal{C}_Q$.*

With Lemma 1, the following claim becomes immediate.

Corollary 1 GEQ *(P, Q) is equivalent to* GEQ *(P, $\mathcal{C}_Q$).*

Proof: The key observation is that for any point $p \in P$, its enclosing distance r_p w.r.t Q is only determined by its furthest neighbor from Q, i.e., $r_p = ||p - \text{fn}(p, Q)||$. By Lemma 1, fn$(p, Q) \in \mathcal{C}_Q$, the result is immediate. $\square$

For example, in the example in Figure 3, p_1's enclosing distance is determined by q_1 and p_2's enclosing distance is determined by q_5. Both are vertices from C_Q. As a result, we can replace Q with $\mathcal{C}_Q$ to reduce the size of the query group. Of course, this may not always lead to removal of points from Q. It is possible that $Q = \mathcal{C}_Q$, e.g., Q has four corner points of a rectangle. Nevertheless, in most cases we could expect that $|\mathcal{C}_Q| < |Q|$, and most likely, $|\mathcal{C}_Q| \ll |Q|$ for a large query set.

4.2 Pruning with the MEB(Q)

Following the discussion that is immediate after equation 1 in the GEQ definition, we can see that the optimal answer p^* to GEQ(P, Q) is the center of the minimum enclosing ball of Q, but subject to the constraint of centering at p^*. This indicates that the optimal answer could not be too far away from the center of the minimum enclosing ball of Q. To use this intuition in our algorithm concretely, we need a simple property of MEB(Q):

Lemma 2 *(Same as the Lemma 2.2 in [7]) For a group of points Q, the surface of any half-sphere of the ball MEB(Q) contains at least one point $q \in Q$ for any dimension d.*

It helps understand this result by considering the case when $d = 2$. The above lemma says that *any half circle* resided on MEB(Q) (which is a circle in the two-dimensional space) contains a point from Q *on its perimeter*. When generalizing this result to d-dimension, it is important to notice that it guarantees the existence of a point from Q on the *surface* of any half-sphere (infinite number of them) defined on the ball MEB(Q).

Let $\mathcal{B}(c, r)$ =MEB(Q) and nn(c, P)= p, the next lemma bounds p^*'s distance from c compared to c's nearest neighbor $p \in P$.

Lemma 3 *Let MEB(Q)= $\mathcal{B}(c, r)$, nn$(c, P) = p$ and $||p - c|| = \lambda$. Then $p^* \subseteq \mathcal{B}(c, r + \lambda)$.*

Proof: Without loss of generality, we assume $c = o$ (the origin) and $p = (\lambda, \overrightarrow{0})$. For any other case, it is always possible to shift and rotate the coordinate

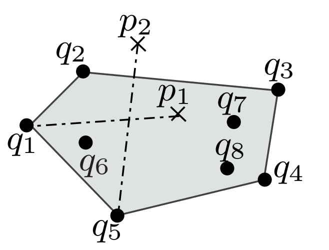


Fig. 3. The convex hull and the pruning with C_Q.

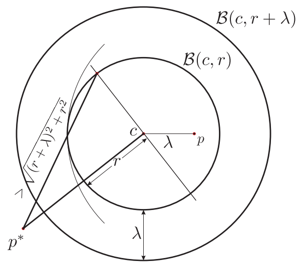
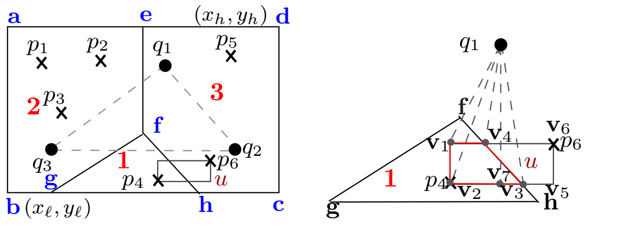


Fig. 4. Proof illustration for Lemma 3.



(a) Furthest voronoi cells. (b) minEdist and maxEdist.

Fig. 5. Furthest voronoi cells help calculate minEdist and maxEdist.

system to satisfy these conditions without affecting the GEQ result. If $p^* = p$, the lemma holds trivially since $||p-c|| = \lambda < r+\lambda$. Hence, consider $p^* \neq p$. The furthest point from p inside the ball $\mathcal{B}(c,r)$ is at distance $r+\lambda$ from p (see Figure 4). Hence, $MEB(Q) \subseteq \mathcal{B}(p, r+\lambda)$ and $r^* \leq r+\lambda$. For a contradiction, let $p^* \notin \mathcal{B}(c, r+\lambda)$. Then $||p^*|| > r+\lambda$. There must exist a point from Q on the other side of the halfspace perpendicular to p^*c by Lemma 2. This implies that the optimal radius $r^* \geq \sqrt{||p^*||^2+r^2}$ (using Pythagorean theorem), and hence $r^* > \sqrt{(r+\lambda)^2+r^2}$. But then we have deduced that $\sqrt{(r+\lambda)^2+r^2} < r^* \leq r+\lambda$ which can not be true since $r+\lambda \leq \sqrt{(r+\lambda)^2+r^2}$. Hence, the assumption $||p^*|| > r+\lambda$ was wrong and $p^* \subseteq \mathcal{B}(c, r+\lambda)$. □

This means that we only need to search the points or MBRs from the R-tree (indexing the data set P) that are contained or intersected by the ball $\mathcal{B}(c, r+\lambda)$.

4.3 Pruning with the furthest voronoi diagram of Q

Given an R-tree node's MBR u and a query group Q, the *tight minimal and maximal* enclosing distances for all possible points inside u clearly help us decide whether we can prune u or not. There are infinite number of possible points that u may contain (even if there are only finite number of actual points from P in u, their precise locations are unknown given only the MBR u). Hence, infinite number of possible enclosing distances exist. We are interested at efficiently finding out both the minimum and the maximum possible enclosing distances for an MBR node u w.r.t a query group Q, as these values tell us what the best and the worst possible scenarios are for any point inside u to enclose Q. Formally,

Definition 1 For an MBR u, the minimum and maximum enclosing distances of u to a query group Q are:

$$\text{minEdist}(u,Q) = \min_{p \subseteq u} r_{p,Q}, \text{maxEdist}(u,Q) = \max_{p \subseteq u} r_{p,Q}.$$

Note that in this definition, we consider all possible points p in the space that are contained by u. They are not necessarily points from P. Also, if u is a point, we let minEdist(u,Q)=maxEdist(u,Q)= $r_{u,Q}$. They could be similarly defined for a convex polygon.

In the sequel, we omit Q from the above definition when the context is clear. If one can efficiently compute minEdist(u) and maxEdist(u), a pruning method based on an R-tree with the branch and bound principle is immediately possible. The key observation is that given two MBRs u_1 and u_2, if u_2's minEdist is larger than u_1's maxEdist, we can safely prune the entire subtree rooted at u_2. To compute them (minEdist and maxEdist) efficiently, we need the help of the *furthest voronoi cells* (FVCs, same as the furthest voronoi diagram) [6], [33]. The FVCs for a set of points is similar to the well-known voronoi diagram except that the space is partitioned with the *furthest cell* instead of the *nearest cell*.

Definition 2 For a space S (a hyperplane) and a set of points Q, a convex polygon FC$(q)\subseteq \mathcal{S}$ is the *furthest cell* of a point $q \in Q$ iff: $\forall p \subseteq$FC(q), fn(p,Q)= q, i.e., any point contained by FC(q) has q as its furthest neighbor from the point set Q. The FVCs of Q is the collection of furthest cells for all points in Q.

An example of the FVCs on an input space $\mathcal{S} : (x_\ell, y_\ell) \times (x_h, y_h)$ and a point set Q is illustrated in Figure 5(a) where FC(q_i) is marked as region i. For example, FC(q_1) is the polygon fgh. Clearly, the furthest voronoi cells satisfy the following properties: 1) for any $q_i, q_j \in Q$, $i \neq j$, FC$(q_i)\cap$FC$(q_j) = \emptyset$; 2) FC$q_1 \cup \cdots \cup$FC$q_M = \mathcal{S}$, where the operator $\cup$ denotes the merge of two convex polygons into one larger convex polygon.

Computing FVCs could be done in a similar fashion as computing the voronoi diagram and it has been discussed in [33]. Yao et al. has shown that the FVCs for all points in Q could be computed efficiently by considering only points from $\mathcal{C}_Q$, specifically:

Lemma 4 *[From [33]] For a space $\mathcal{S}$, a set of points Q and its convex hull $\mathcal{C}_Q$, $\forall q \in Q$, if $q \notin \mathcal{C}_Q$, then q does not have a furthest cell, i.e., if $q \notin \mathcal{C}_Q$, FC$(q) = \emptyset$.*

The FVCs of Q cover the entire space (by its second property we have discussed). Hence, for any MBR u, there will be some q_i's from Q such that FC(q_i)'s intersect with (or only one FC(q$_i$) fully contains) u. By Lemma 4, these q_i's must be some vertices from the convex hull of Q. Furthermore, the union of these intersections will be equal to u. For example, in Figure 5(a) u is partitioned into two regions that correspond to intersections with FC(q_1) and FC(q_3) respectively. Taking a closer look of u, as shown in Figure 5(b), the furthest neighbor from Q for any point in the first region $(v_1v_2v_3v_4v_1)$ will be q_1 and similarly q_3 for any point in the second region $(v_3v_5v_6v_4v_3)$. This is simply because that they are the intersections of u with FC(q_1) and FC$(_3)$ respectively,

```
Algorithm 1: Edist(MBR $u$, FVCs $Q$, Convex Hull $\mathcal{C}_Q$)
1 minEdist= $\infty$; maxEdist= 0;
2 for $i = 1, \dots, |\mathcal{C}_Q|$ do
3     let $A_{u_i} = u \cap FC(q_i)$; // $q_i \in \mathcal{C}_Q$
4     if $A_{u_i} \neq \emptyset$ then
5         find $V_{u_i}$ and $W_{u_i}$ as in Lemma 6;
6         $d_1$=min($_{v \in W_{u_i}} ||q_i - v||$),
          $d_2$=max($_{v \in V_{u_i}} ||q_i - v||$);
7         if $d_1 \leq$ minEdist then minEdist=$d_1$;
8         if $d_2 \geq$ maxEdist then maxEdist=$d_2$;
9 output minEdist, maxEdits; return.
```

i.e., $v_1v_2v_3v_4v_1 \subset$FC(q_1) and $v_3v_5v_6v_4v_3 \subset$FC(q_3). By the definition of a furthest cell, any points inside $v_1v_2v_3v_4v_1$ will take q_1 as their furthest neighbors w.r.t Q and any points inside $v_3v_5v_6v_4v_3$ will take q_3 as their furthest neighbors w.r.t Q. Hence, we have:

Lemma 5 *For an MBR u and a point set Q, there will be some q_i's (in Q) whose FC(q_i)'s intersect with (or just one fully contains u). Denote these q_i's as $I(u) = \{q_{u_1}, \dots, q_{u_k}\}$, then $\cup_{q_{u_i} \in I(u)}(FC(q_{u_i}) \cap u) = u$; and $\forall q_{u_i} \in I(u)$, $q_{u_i} \in \mathcal{C}_Q$; and $\forall p \subseteq FC(q_{u_i}) \cap u$, fn$(p, Q)$= q_{u_i}.*

The main lemma is as follows.

Lemma 6 *For $\forall q_{u_i} \in I(u)$, Let $A_{u_i} = v_1 \cdots v_{s_i} v_1$ be the polygon for FC(q_{u_i})$\cap u$, let $V_{u_i} = \{v_1, \dots, v_{s_i}\}$ and initialize W_{u_i} =V_{u_i}. For $1 \leq j \leq s_i$, if the projection of q_{u_i} to the line segment v_jv_{j+1} (let $s_i + 1 = 1$) is a point v' contained by v_jv_{j+1}, insert v' to W_{u_i}:*

$$maxEdist(A_{u_i}, Q) = argmax_{v \in V_{u_i}} ||v - q_{u_i}|| \quad (3)$$

$$minEdist(A_{u_i}, Q) = argmin_{v \in W_{u_i}} ||v - q_{u_i}|| \quad (4)$$

Proof: By Lemma 5, given any point $p \subseteq A_{u_i}$, the enclosing distance of p to Q is $||p - q_{u_i}||$ as fn$(p, Q) = q_{u_i}$ if $p \subseteq A_{u_i}$. Hence, the minimum and maximum enclosing distances of the convex polygon A_{u_i} to Q equal to the minimum and maximum distances of the point q_{u_i} to A_{u_i}. Applying Lemma 1, we obtain equation 3, i.e., the furthest neighbor from a point q_{u_i} to a convex A_{u_i} is determined by the vertices of A_{u_i}; Next, in a similar way as arguing how we compute minDist for a point p to an MBR [30], it is easy to show that the minimum distance for a point q_{u_i} to a convex polygon A_{u_i} is achieved by either one of its perpendicular projection points *on* the boundaries, or one of the vertices of the convex polygon A_{u_i}, i.e., we only need to consider the candidates from the set Wu_i. Hence, equation 4 holds as well. □

Lemma 6 indicates that the minEdist and maxEdist for an MBR u are simply the minimum and maximum from the minEdist and maxEdist of the polygons corresponding to the intersections of u with FC(q_{u_i})'s, where $u_i \in I(u)$; and for each polygon A_{u_i} =FC$(q_{u_i}) \cap u$, its minEdist and maxEdist are solely determined by the distance between q_{u_i} and vertices of A_{u_i} (plus the perpendicular projection points of q_{u_i} on A_{u_i}), which are easy to compute as it boils down to computing distance between two points. Algorithm 1 outlines this idea.

Figure 5(b) shows an example. In this case, I(u)=$\{q_1, q_3\}$ and A_{q_1} = FC$(q_1) \cap u$ is the polygon $v_1v_2v_3v_4v_1$. Its V_{q_1} is $\{v_1, v_2, v_3, v_4\}$ and W_{q_1} is $\{v_1, v_2, v_3, v_4, v_7\}$. Note that among four perpendicular projections of q_1 to v_1v_2, v_2v_3, v_3v_4 and v_4v_1, only the projection on v_2v_3 produces a point v_7 to be inserted into W_{q_1} as other projection points are outside the corresponding line segments. Now, the maxEdist for FC$(q_1) \cap u$ is given by $||q_1 - v_2||$ and we only need to check $||q_1 - v_i||$ for $v_i \in V_{q_1}$; its minEdist is given by $||q_1 - v_4||$ and we only need to check additionally $||q_1 - v_7||$ (the one additional point in W_{q_1}). We can find the minEdist and maxEdist for A_{q_3} = FC$(q_3) \cap u$ in a similar fashion. The minEdist and maxEdist of u could be identified by taking the minimum and maximum from the minEdist's and maxEdist's of A_{q_1} and A_{q_3}.

Once we know how to compute the minEdist and maxEdist for an MBR u, we can use them to prune the search space from a R-tree in a typical branch and bound principle, as discussed at the beginning of this section.

4.4 Putting Everything Together: GEQS Algorithm

Our exact search algorithm GEQS is to simply combine the three pruning techniques introduced in sections 4.1, 4.2 and 4.3. Given Q and the R-tree on P, we first compute the convex hull $\mathcal{C}_Q$ to reduce the size of the query group. Then MEB($\mathcal{C}_Q$), the nearest neighbor of MEB($\mathcal{C}_Q$)'s center from P (with the help of the R-tree) and the furthest voronoi cells of $\mathcal{C}_Q$ are computed. Next, a priority queue $\mathcal{L}$ is maintained for all MBRs (points are treated as MBRs as well) that the algorithm has to search for. Entries in $\mathcal{L}$ are ordered by their minEdist distances. An R-tree node could be safely pruned without exploring its children nodes if either it does not intersect with $\mathcal{B}(c, r + \lambda)$ as argued from section 4.2, or *its minEdist is larger than the smallest maxEdist of any existing entries from the queue $\mathcal{L}$* as argued in section 4.3. Initializing the queue with the root node, we repeatedly pop the head of the queue and insert its children nodes back to the queue if they cannot be pruned. The algorithm terminates when the head of the queue becomes a point.

5 Approximate Query Algorithms

As we have pointed out in Section 1, the nearest neighbor search is a special case for the GEQ problem. Hence, any exact method for the GEQ problem will suffer from the curse of the dimensionality. The best hope for answering a GEQ query instance for data sets in high dimensions is to look for efficient and effective approximation algorithms, just as what people have done for the approximate nearest neighbor search in high dimensions [15], [20]. This section shows an efficient way to compute a $\sqrt{2}$-approximate solution to the GEQ problem in any fixed dimension using a nearest neighbor query. We then use it to obtain an optimal $(1+\epsilon)$-approximate solution to the GEQ problem in fixed dimensions for any $\epsilon > 0$, and extend it to high dimensions.

5.1 An $\sqrt{2}$-Approximation

Algorithm 2 presents our $\sqrt{2}$-approximation algorithm. As we can see, it is extremely simple and easy to implement in any dimension as long as a nearest neighbor data structure is available. We next prove the approximation factor for our algorithm.

Algorithm 2: GEQA(Query Q; Dataset P)

```
1 find MEB($Q$) and let its center be the point $c$;
2 find $c$'s nearest neighbor in $P$, i.e., $p$ =nn($c, P$);
3 return p.
```

Theorem 1 *Algorithm* GEQA *gives a $\sqrt{2}$-approximate solution to the* GEQ *problem on P and Q in any dimension d. In addition, this bound is tight.*

Proof: We first prove that it indeed gives an $\sqrt{2}$-approximation, then argue that the bound is also tight.

Let $\mathcal{B}(c,r)$ = MEB(Q), p =nn(c, P) and $||p-c|| = \lambda$. Assume that $p \neq p^*$ (recall that p^* is the exact, optimal answer for the GEQ problem), otherwise the lemma trivially holds. Since $||p-c|| = \lambda$ and $r_c = r$ (recall that r_c is c's enclosing distance w.r.t Q), the ball $\mathcal{B}(p, r+\lambda)$ clearly contains $\mathcal{B}(c,r)$ that is MEB(Q). Hence, we have $Q \subseteq \mathcal{B}(c,r) \subseteq \mathcal{B}(p, r+\lambda)$. Please refer to Figure 6(a) for a pictorial illustration where points in Q are highlighted by the solid circles.

Given that $Q \subseteq \mathcal{B}(p, r+\lambda)$, we can immediately infer that MEB(p, Q) $\subseteq \mathcal{B}(p, r+\lambda)$, i.e., the minimum enclosing ball for Q with p as the center will be contained by $\mathcal{B}(p, r+\lambda)$. This implies that $r_p \leq r+\lambda$, as r_p is p's enclosing distance that equals to the radius of MEB(p, Q). On the other hand, obviously, $r^* \leq r_p$ as r^* denotes the optimal enclosing distance w.r.t Q for all points in P. Hence,

$$r^* \leq r_p \leq r + \lambda. \tag{5}$$

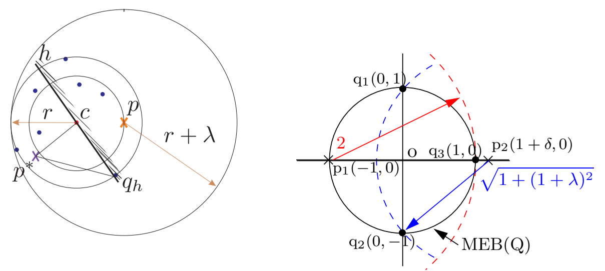
Since the algorithm GEQA returns p as the answer, we only need to show that the optimal enclosing distance r^* can not be too small compared to $r+\lambda$ (as it will be even closer to r_p according to equation 5).

By our construction, nn(c, P) $= p$. Hence, p^* is not closer to c compared to p, i.e.,

$$||p^* - c|| \geq \lambda. \tag{6}$$

Let h be the *halfspace* passing through c, perpendicular to p^*c and not containing p^*. Note that MEB(Q) $\cap\, h$ defines a halfsphere on the ball MEB(Q) (the $\mathcal{B}(c,r)$, see Figure 6(a)) as h is a halfspace. Using Lemma 2, there must exist a point $q_h \in Q$ on the surface of MEB(Q) $\cap\, h$ (note that this surface is on the side that does not contain p^*, by the construction of h). This means that:

$$||q_h - c|| = r. \tag{7}$$



(a) Theorem 1. (b) Tightness of Theorem 1.

Fig. 6. GEQA: p is the $\sqrt{2}$-approximate answer, $\mathcal{B}(c,r)$ =MEB(Q), p =nn(c, P), $||p-c|| = \lambda$.

Since the optimal solution of the GEQ problem centered at p^* must cover q_h (as $q_h \in Q$), this shows that:

$$r^* \geq ||p^* - q_h||. \tag{8}$$

On the other hand, consider the triangle defined by p^*, c and q_h, since p^*c is perpendicular to h and q_h is from the halfsphere defined by MEB(Q) $\cap\, h$ and on the side that does not contain p^*, the minimum distance between p^* and q_h is when p^*, c and q_h forms a right triangle. Hence, we have:

$$||p^* - q_h||^2 \geq ||q_h - c||^2 + ||p^* - c||^2.$$

By equations 6 and 7, this implies

$$||p^* - q_h||^2 \geq r^2 + \lambda^2. \tag{9}$$

Given equations 8 and 9, we arrive at

$$r^* \geq ||p^* - q_h|| \geq \sqrt{r^2 + \lambda^2}. \tag{10}$$

The approximation ratio of p is given by r_p/r^*, which satisfies, by applying equation 5 and equation 10 respectively:

$$\frac{r_p}{r^*} \leq \frac{r+\lambda}{r^*} \leq \frac{r+\lambda}{\sqrt{r^2+\lambda^2}} \leq \sqrt{2} \tag{11}$$

This completes the first part of our proof. Note that the above analysis is independent of the dimensionality.

To show that the above bound is tight, it is sufficient to construct an example where this algorithm indeed gives arbitrarily close to $\sqrt{2}$-approximate solutions. Consider $\mathbb{R}^2$ and let $Q = \{(0,1), (0,-1), (1,0)\}$ and $P = \{(-1,0), (1+\delta, 0)\}$, as illustrated in Figure 6(b). The center of MEB(Q) is the origin o, and nn(o, P) $= p_1 = (-1, 0)$. The enclosing distance r_{p_1} is 2 (the furthest point in Q to p_1 is $q_3 = (1,0)$). Hence, p_1 will be returned as the answer by algorithm GEQA. However, the optimal answer p^* should be $p_2 = (1+\delta, 0)$ with $r^* = r_{p_2} = \sqrt{1+(1+\delta)^2}$. This shows that the algorithm GEQA gives a $(2/\sqrt{2+\delta^2+2\delta})$-approximate solution for this example for arbitrarily small value of $\delta > 0$. This shows that the approximation bound $\sqrt{2}$ is tight. □

We would like to point out that even though Theorem 1 suggests that the $\sqrt{2}$-approximation bound is tight for algorithm GEQA, the worst case as illustrated in our proof is extremely rare in real data sets and queries. In fact, our extensive experiments (see Section 7) show that the approximation ratio of the GEQA algorithm is very close to 1 in practice.

In practice, it is possible that the *efficient* exact nearest neighbor and minimum enclosing ball algorithms are not available. We now show how the approximation factor of GEQA varies when approximate versions of nearest neighbor and minimum enclosing ball are used.

Theorem 2 *Given a data structure that can do α-approximate nearest neighbor queries (for example $\alpha = (1+\epsilon)$ using [5]), and another algorithm that can compute β-approximate minimum enclosing balls (for example $\beta = (1+\epsilon)$ using [22]), GEQA computes $\sqrt{\alpha^2+\beta^2}$-approximate solution. This matches Theorem 1 when $\alpha = \beta = 1$.*

Proof: This follows from replacing the numerator in Equation 11 for taking into account the approximations:

$$\begin{aligned}\frac{r_p}{r^*} &\le \frac{\alpha r + \beta\lambda}{r^*}\\ &\le \frac{\alpha r + \beta\lambda}{\sqrt{r^2+\lambda^2}} \le \frac{(\alpha+\beta\eta)}{\sqrt{1+\eta^2}},\end{aligned} \tag{12}$$

where $\eta = \lambda/r$, r is the radius of the exact minimum enclosing ball of Q and λ is the exact distance between the center of MEB(Q) to its exact nearest neighbor in P. Let $f(\eta) = \frac{(\alpha+\beta\eta)}{\sqrt{1+\eta^2}}$, which we want to upper bound. The only stationary point of f occurs at $\eta = \beta/\alpha$ (which we get from solving $\partial f/\partial\eta = 0$). We can now show that $f()$ is upper bounded by $f(\eta = \beta/\alpha) = \sqrt{\alpha^2+\beta^2}$:

$$\frac{\alpha+\beta\eta}{\sqrt{1+\eta^2}} \le \sqrt{\alpha^2+\beta^2} \tag{13}$$

which simplifies to $(\alpha\eta-\beta)^2 \ge 0$. Hence:

$$\frac{r_p}{r^*} \le \frac{\alpha r+\beta\lambda}{\sqrt{r^2+\lambda^2}} \le \sqrt{\alpha^2+\beta^2}$$

□

Finding the minimum enclosing ball in fixed dimensions, for a group of points has been studied extensively [9] and efficient algorithms exist for line 1 in Algorithm 2 using linear time w.r.t the size of Q, i.e., $O(M)$. For line 2 in Algorithm 2, one could use a BBD-tree with optimal approximate nearest neighbor search to find a $(1+\epsilon)$-approximate nearest neighbor from P for c in $O(\log N)$ query cost [5] as we have discussed in Section 3. Hence, the next corollary is immediate.

Corollary 2 *In any fixed dimension d and for a data set P, a $\sqrt{1+(1+\epsilon)^2} \le \sqrt{2}+\epsilon$-approximate solution for any GEQ instance defined by P and an arbitrary query group Q can be found by algorithm GEQA in $O(M+\log N)$ time [5]. The GEQA algorithm takes $O(N\log N)$ preprocessing cost to build the index and a linear space usage $O(N)$.*

Corollary 2 gives a theoretical bound for algorithm GEQA's query cost. However, BBD-tree is impractical for large data sets that are disk-resident in high dimensions. Hence, in practice, we could use an R-tree to find the nearest neighbor in algorithm GEQA, or any other technique for efficient, practical nearest neighbor retrieval as we have discussed in Section 3 (see also our discussion on this issue in high dimensions in Section 5.3).

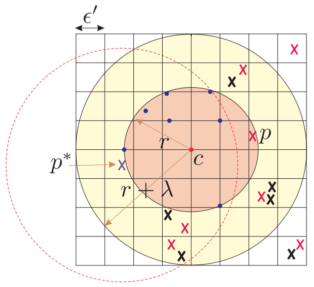


Fig. 7. An illustration of Theorem 3. p denotes the $\sqrt{2}$-approximate answer computed by the algorithm. Each grid cell elects one point for the algorithm.

Lastly, we would like to point out that GEQA is similar in principle to the approximate method, A-SPM, for the max-GNN query [26], [27], where the centroid of Q was returned as the approximate answer. However, the A-SPM method is focused on 2-d (speaking about the centroid or the minimum enclosing disks instead of balls) and more importantly, no theoretical analysis on its approximation quality was given. In contrast, we have generalized it to any dimensions and analyzed its approximation quality.

5.2 Optimal $(1+\epsilon)$-Approximation Algorithm

In the special case when $Q = \{q\}$, the GEQ problem becomes the nearest neighbor problem. If we restrict the space usage to linear, the best algorithm (in terms of both the approximation quality and the query cost) for the nearest neighbor problem gives $(1+\epsilon)$-approximate nearest neighbor in $O(\log N)$ cost with $O(N\log N)$ preprocessing [5] time to build the BBD-Tree. This suggests that the best approximation one can hope for the GEQ problem is an $(1+\epsilon)$-approximation, and such an approximation's best possible query cost is, when only allowed to use linear space usage, a preprocessing time of $O(N\log N)$ and query time of $O(\log N+M)$. We next show that indeed such an algorithm exists based on the optimal algorithm for approximate nearest neighbor search [5] and approximate range searching [4]. The basic idea of this algorithm is described in the following theorem and its proof.

Theorem 3 *In fixed dimensions, for a given $\epsilon > 0$, with $O(N\log N)$ preprocessing time and linear space usage, a $(1+\epsilon)$-approximate solution to the GEQ problem can be found in $O(\log N+M)$ time. In addition, these bounds are tight, given the linear space constraint, for an $(1+\epsilon)$-approximation.*

Proof: We first compute the MEB(Q) $= \mathcal{B}(c,r)$ in $O(M)$ time [9]. We then use Lemma 1 and Arya et al.'s [5] approximate nearest neighbor data structure (a BBD-Tree) to compute a $\sqrt{2}$-approximate answer (with the

GEQA algorithm shown in Section 5.1) in $O(\log N)$ time with $O(N \log N)$ preprocessing time.

As in Lemma 1, let $p = \text{nn}(c, P)$ and $||p - c|| = \lambda$ (see Figure 7). We can show that $p^* \in \mathcal{B}(c, r+\lambda)$ (see Lemma 3 in Section 4.2). We partition $\mathcal{B}(c, r+\lambda)$ using a grid whose cells have a side length $\epsilon' = \epsilon\sqrt{\frac{r^2+\lambda^2}{d}}$ (d is the dimensionality of the data set). This restricts the number of grid cells inside $\mathcal{B}(c, r+\lambda)$ to $O((\frac{r+\lambda}{\epsilon'})^d) = O(\left(\frac{\sqrt{2d}}{\epsilon}\right)^d) = O(1)$, since we consider fixed dimensions.

The number of cells of the BBD-tree that intersect with $\mathcal{B}(c, r+\lambda)$ and have minimum side length larger than ϵ' is $O(1)$ (cf. Lemma 3, [4]). We use the BBD-tree to compute a point in each grid cell that has a non-empty intersection with P (see red cross points, or darker bold cross points in a black-white printout, in Figure 7). We call this set of points P' whose size is at most $O(1)$ and which can be computed in $O(\log N)$ time using an approximate range query [4] (and terminating the search as soon as the boxes become smaller than side length ϵ'). For each of the points in P', we compute the furthest neighbor in Q which give us their enclosing distances. This takes at most $O(M)$ time. Finally, we just select the one with the minimum enclosing distance as the answer. Hence the total running time is $O(\log N + M)$. The total error in the computation is bounded by the error that we make in one cell of the grid. Since we choose randomly one representative point from the points a cell contains in P to be included in P', the error introduced is at most $\epsilon'\sqrt{d} = \epsilon\sqrt{r^2 + \lambda^2} \le \epsilon r^*$, by equation 10.

This completes our proof for the existence of an $(1+\epsilon)$-approximate algorithm for the GEQ problem with linear space usage, $O(N \log N)$ preprocessing time and $O(\log N + M)$ query cost. We finally show that these bounds are tight if using linear space.

In the special case when $Q = \{q\}$, the GEQ problem becomes the nearest neighbor problem. For such a nearest neighbor problem, with $O(N)$ space, $O(N \log N)$ preprocessing time and $O(\log N)$ time are shown to be required to find a $(1 + \epsilon)$-approximation [5]. For the general GEQ problem when $|Q| > 1$, since any of the query points in Q can change the answer, a query is lower bounded by $\Omega(M)$. This suggests that the best query time one can hope for the $(1+\epsilon)$-approximate GEQ problem, with linear space usage, is $O(M + \log N)$ with a preprocessing cost of $O(N \log N)$. This confirms the tightness of our algorithm. $\square$

There are hidden constant factors (depends on d) for claims in Corollary 2 and Theorem 3, since they both rely on the BBD-tree for the optimal approximate nearest neighbor search in any fixed dimensions [5]. These factors are the same as those in the BBD-tree, specifically, for N points from P and M points from Q in R^d, the preprocessing cost is $O(dN \log N)$, the space is $O(dN)$, and the query cost is $O(M + c_{d,\epsilon} \log N)$, where $c_{d,\epsilon} \le d\lceil 1 + 6d/\epsilon \rceil^d$.

This optimal approximation relies on the BBD-Tree, which is impractical for large data sets in high dimensions. Further more, even in relatively low dimensions, the treatment to the (many) small grid cells becomes complex and intricate to deal with. Hence, this result is of theoretical interest only and contributes to the completeness of the study on the GEQ problem.

5.3 High Dimensions

The $\sqrt{2}$-approximation by Algorithm 2 in Section 5.1 is independent of the dimensionality, if one can find the exact nearest neighbor of a given point and the exact minimum enclosing ball of a group of points. However, both tasks become hard (in terms of the efficiency) in high dimensions. For example, in high dimensions the exact nearest neighbor most likely requires a linear scan of the entire data set [15], [20], [31]. Hence, in practice, only approximate nearest neighbor can be found efficiently in high dimensions, so does the approximate minimum enclosing ball for a group of points.

We leverage on the state-of-the-art methods for efficient approximate nearest neighbor retrieval and approximate minimum enclosing ball calculation, specifically, the LSB-tree [31] and the $(1 + \epsilon)$-approximate minimum enclosing ball [22]. The LSB-tree can give a $(2 + \epsilon)$-approximate nearest neighbor with constant probability in high dimensions efficiently [31], where this constant probability depends on the configuration parameter of the LSB-tree (see [31] for details); the $(1+\epsilon)$-approximate minimum enclosing ball algorithm computes a minimum enclosing ball that has a radius that is at most $(1+\epsilon)$ times the radius of the optimal minimum enclosing ball in any dimension [22]. Based on our result in Theorem 2, the following result is immediate:

Corollary 3 *If steps 1 and 2 in* GEQA *(Algorithm 2) are replaced with the respective* $(2 + \epsilon)$*-approximation and* $(1 + \epsilon)$*-approximation algorithms,* GEQA *gives a* $\sqrt{(2 + \epsilon)^2 + (1 + \epsilon)^2} \le \sqrt{5} + \sqrt{2}\epsilon$*-approximation with constant probability in any dimensions.*

The constant probability in Corollary 3 is the same as the constant probability provided by the LSB-tree, which depends on its configuration parameters. Interested readers may refer to [31] for further details. One can always use a standard technique in randomized algorithms to improve the constant probability in Corollary 3 to at least $1-p$ for arbitrary probability p by repeating the LSB-trees $O(\log 1/p)$ number of times.

6 Other Issues

6.1 Handling disk-resident query groups

One limitation with our discussions so far is the problem of handling query groups that do not fit in internal memory. We could certainly use the convex hull of Q to reduce the number of points in the query group, as pointed by the pruning technique in section 4.1. There are I/O efficient algorithms for computing the convex hulls of disk-based data sets, with the help of sorting

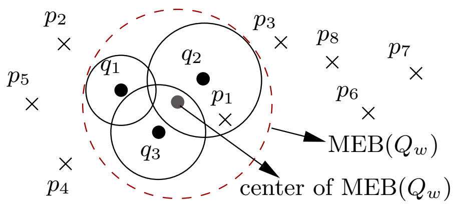


Fig. 8. Constrained GEQ, minimum enclosing ball of balls.

in external memory [16]. This means that we can find the convex hull of the disk-based query group Q with $O(sort(M))$ I/Os (here $sort(M)$ denotes the number I/Os when M fixed size elements of size $O(1)$ are sorted. We also use $scan(M)$ to denote the number of I/Os incurred for scanning M fixed size elements). For most cases, we expect that $|\mathcal{C}_Q| \ll |Q|$. However, special cases exist where $|\mathcal{C}_Q| = |Q|$, e.g., all points in Q are vertices of a convex polygon. To handle such special instances, we propose to obtain an approximate convex hull of Q using Dudley's approximation [35]. Dudley's construction generates an approximate convex hull of Q (denote it as $\mathcal{AC}_Q$) with $O(1/\epsilon^{(d-1)/2})$ vertices with maximum *Hausdorff distance* of ϵ to the convex hull of Q. The *Hausdorff distance* measures how far two convex polygons S_1 and S_2 are from each other. Formally, let X and Y be the vertices of S_1 and S_2 respectively, then:

$$d_H(S_1, S_2) = \max\big(\sup_{x\in X}\inf_{y\in Y}||x-y||, \sup_{y\in Y}\inf_{x\in X}||x-y||\big)$$

Roughly speaking, the edges of of $\mathcal{AC}_Q$ are within ϵ distances from edges of $\mathcal{C}_Q$. Clearly, there is a trade-off between the approximation quality and the size of $\mathcal{AC}_Q$. Henceforth, for a disk-based query group Q, we first compute its convex hull using the I/O efficient algorithm. If $\mathcal{C}_Q$ is still too large to fit in main memory, we replace $\mathcal{C}_Q$ with the Dudley's approximation and specify the size of $\mathcal{AC}_Q$ of our choice.

Dudley's approximation was originally proposed for main memory data sets [35]. For our purpose, it needs to work with external data sets. This has been recently addressed by Yao et al. [33], which presented a simple method that computes $\mathcal{AC}_Q$ in external memory efficiently by extending the technique from [35].

For GEQA the Dudley's approximation only creates an extra additive error ϵ for a given query. Intuitively, the center of the MEB($\mathcal{C}_Q$) is shifted by at most ϵ comparing to MEB($\mathcal{AC}_Q$). For general dimensions, the GEQA algorithm does $O(scan(M))$ I/Os to compute a $(1+\epsilon)$-approximate MEB [22] plus the I/Os needed for the nearest neighbor query. The preprocessing involves exactly the same number of I/Os as used by the nearest neighbor data structure. It is much more complicated to analyze the approximation quality of the GEQS algorithm and we leave it as an open problem. In practice, it gives very good approximations as well.

6.2 Constrained GEQ

An interesting variant of the GEQ problem is to add a "constrained area" to each query point. Each query object is represented by a center q_c and a radius q_r. The query point is allowed to be anywhere within the ball defined by $\mathcal{B}(q_c, q_r)$. We denote such a query set as Q_w (where each object is represented by a ball). The constrained GEQ is useful in many situations. For example, each person could be moving within a small range in our Example 1 from Section 1, or it could be the case that their precise locations are unavailable for privacy concerns and only approximate ranges are provided. Formally, the constrained GEQ is the following optimization problem:

$$r^*_{P,Q_w} := \min_{p\in P}\max_{\{q\subseteq\mathcal{B}(q_c,q_r)\wedge\mathcal{B}(q_c,q_r)\in Q_w\}}||p-q|| \qquad (14)$$

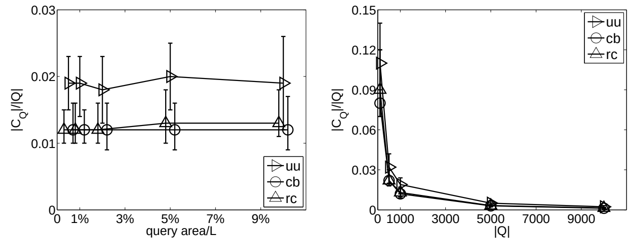
We would like to find the optimal enclosing distance r^*_{P,Q_w} as well as the point $p^*_w \in P$ that achieves this minimum enclosing distance. The enclosing distance of a point p to Q_w is defined as $r_{p,Q_w} = \max_{q\subseteq\mathcal{B}(q_c,q_r)\wedge\mathcal{B}(q_c,q_r)\in Q_w}||p-q||$. An example is shown in Figure 8 and p^*_w is p_1 is in this case.

Fortunately, our algorithms developed for the GEQ problem could be easily adapted to work with the constrained case. The key observation is that it is still possible to efficiently compute the minimum enclosing ball for a set of balls [14]. As an example, the minimum enclosing ball of Q_w in Figure 8 is the ball with the dotted edge. By applying a few technicalities, we can still obtain the $\sqrt{2}$-approximation as in Section 5, as well as the range pruning technique for the exact search in Section 4.2 using the center of this minimum enclosing ball and its nearest neighbor in P for the constrained GEQ problem. The furthest voronoi diagram pruning in Section 4.3 is, however, no longer applicable.

7 EXPERIMENTS

All proposed algorithms and the state-of-the-art method (the MBM) have been implemented into the widely used, disk-based R-tree index library [18] in low to relatively high dimensions. Standard geometric operations, such as computing MEB(Q), $\mathcal{C}_Q$, intersection of convex polygons and the furthest voronoi diagram, are provided by the CGAL and qhull libraries [1], [3]. Finally, the approximate convex hull algorithm is developed based on the library from [35]. Our implementation is compatible with the standard `gcc`. All experiments were executed on a Linux machine with an Intel Xeon 2GHz CPU and 2GB memory. For both R-tree and heapfiles the page size is set to 4KB. Finally, by default *1000* queries were generated for each experiment and we report the average.

Data sets. The first set of real data sets was obtained from the open street map project [2]. Each data set contains the road network for a state in the USA. Each point has its longitude and latitude coordinates. We normalized each data set into the space $L = (0, 0) \times (10^5, 10^5)$. For our experiments, we have used the Texas (*TX*) and California (*CA*) data sets. The *TX* data set has 14 million points and the *CA* data set has 12 million points.



(a) vary $\mathcal{A}$, $|Q| = 1000$. (b) vary $|Q|$, $\mathcal{A} = 3\%$.

Fig. 9. Pruning with the convex hull of Q.

These real data sets are in two dimensions. To test the performance of our algorithms on different dimensions and different types of distributions, we also use synthetic data sets in the space of $L = \overrightarrow{0}^d \times \overrightarrow{100,000}^d$ where d is the dimensionality, for low to relatively high dimensions ($d = 2$ to $d = 6$). In the *UN* data set, points are distributed uniformly in the space and in the *RC* data set, points are generated with random clusters in the space.

Finally, to study the performance of the GEQA algorithm in high dimensions, when only approximate nearest neighbor and minimum enclosing ball can be efficiently obtained in practice, we used the following three real data sets in very high dimensions. The *Color* data set [11] consists of $68,040$ points with 32 dimensions and the *MNIST* data set [23] has $60,000$ points with 50 dimensions. To investigate the performance of the GEQA with even higher dimensions and larger data size, we use the *Cortina* data set [29], which consists of $1,088,864$ points with 74 dimensions.

Query groups. For the GEQ problem, the cost of the query depends on several critical factors. They include the location for the center of Q, the covering range of Q (i.e., the area of MEB(Q)) and how points are distributed within the covering range. Lastly, the size of Q also plays a role. Given these observations, a random query is generated as follows. We specify $|Q|$ and its area $\mathcal{A}$ in terms of the percentage of the entire data space L (or volume in more than two dimensions). Next a random location in L is selected as the center of the query group. Three types of distributions were used to generate $|Q|$ number of points within the specified area. They include the uncorrelated uniform distribution (uu), the correlated bivariate distribution (cb) and the randomly clustered distribution (rc).

Default setup. In the following experiments, the default query size is $|Q| = 1000$, the default query area is $\mathcal{A} = 3\%$, the default dimensionality is $d = 2$, and the default query distribution type is rc. For all experiments, except the results in Section 7.3 for very high dimensions, the default size N of the data set P is 3 million. For the *TX* and *CA* data sets, we randomly sample 3 million points to create the default data sets. The default data sets for experiments in low ($d = 2$) to relatively high ($d = 6$) dimensions in Sections 7.1 and 7.2 are the *UN* and *CA* data sets. The results from the *TX* data set are similar to the results from the other real data set *CA*.

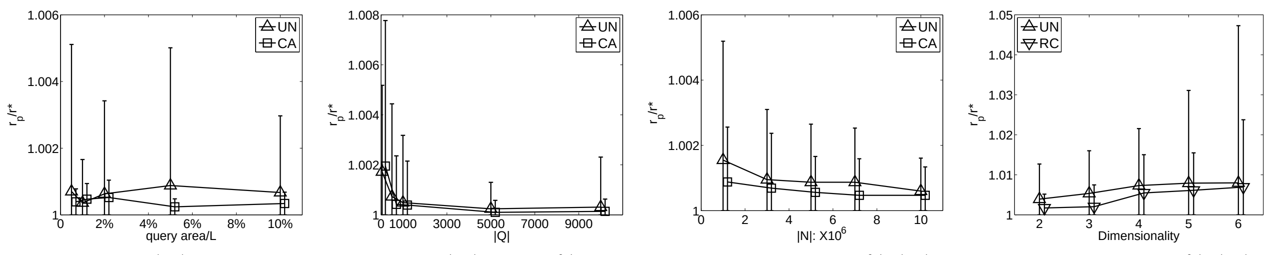
7.1 Performance of the GEQA and GEQS algorithms

Pruning power of $\mathcal{C}_Q$. Based on our findings in Section 4.1, any algorithm for the GEQ problem could first utilize the convex hull $\mathcal{C}_Q$ of the query group Q to reduce the query size. Hence, our first set of experiments explores the pruning power of $\mathcal{C}_Q$ for typical distributions that a query group may have. Figure 9 shows that for various query distributions, $\mathcal{C}_Q$ could significantly reduce the size of the query group. We plot the average ratios for $|\mathcal{C}_Q|/|Q|$ together with the $5\% - 95\%$ confidence interval. For $|Q| = 1000$, $|\mathcal{C}_Q|$ is only about 2% of $|Q|$ on average and never exceeds 3% in the worst case, regardless of the area of the query group, as shown in Figure 9(a). Figure 9(a) indicates that $\frac{|\mathcal{C}_Q|}{|Q|}$ is roughly a constant over the query area and *uu* distributed query groups have larger $|\mathcal{C}_Q|$ over correlated and clustered groups (however, $\mathcal{C}_Q$ is still just $2\%|Q|$). The pruning power of $\mathcal{C}_Q$ increases quickly (see Figure 9(b)) for larger query groups as $|\mathcal{C}_Q|$ grows at a much slower pace than the size of the query group does. In most of our experiments, the results from the uu, cb and rc query groups are quite similar. Hence, remaining experimental figures only report the results from the rc query groups.

The approximation quality of GEQA. Recall that the approximation ratio is defined as r_p/r^* if p is returned as the answer (r_p is p's enclosing distance with respect to Q and r^* is the optimal enclosing distance). Figure 10 plots the average approximation ratio of the GEQA algorithm (Algorithm 2) together with its $5\% - 95\%$ confidence interval. These results convincingly show that GEQA's approximation quality in practice is much better than its worst case theoretical bound, which is $\sqrt{2}$. In general, from Figure 10 we can tell that for all data sets we have tested, its approximation quality is very close to 1 with an average approximation ratio of 1.001 in two dimensions, and still below 1.01 in six dimensions. Not only it achieves an excellent average approximation ratio, but also it has a very small variance in its approximation quality in practice. When $d = 2$, its 95% interval is below 1.008 in the worst case. The overall worst case appears when $d = 6$, but its 95% interval is still below 1.05.

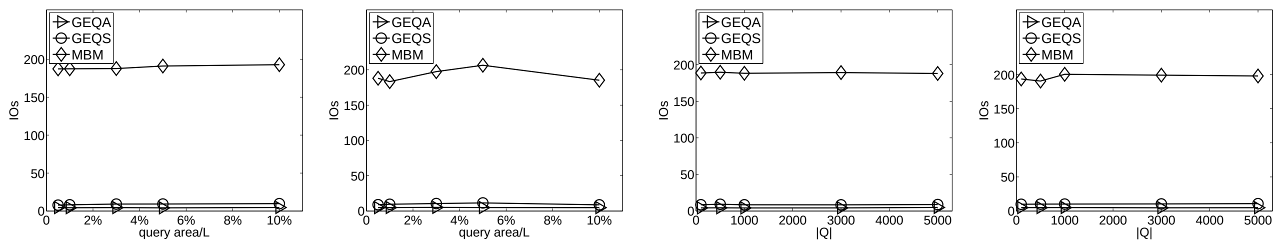
More specifically, for varying query covering areas $\mathcal{A}$ (Figure 10(a)), varying the query group size $|Q|$ (Figure 10(b)) and varying the data set size N (Figure 10(c)) in two dimensions, the approximation ratios for GEQA are almost a constant with an average that is around 1.001 and the 95% interval is below 1.006. Its approximation quality drops slowly as the dimensionality increases as shown in Figure 10(d). However, even in the worst case when d becomes six dimensions, its average approximation ratio is still below 1.01 with an 95% interval that is below 1.05. Finally, the distribution of the data set P affects the approximation ratios to some degree. But its effect is very minimal as evident by the close approximation ratios from the *UN* and *CA* data sets.

A very appealing property of GEQA is its simplicity,



(a) effect of $\mathcal{A}$: $|Q| = 1000$, $N = 3 \times 10^6$, $d = 2$. (b) effect of $|Q|$: $\mathcal{A} = 3\%$, $N = 3 \times 10^6$, $d = 2$. (c) effect of N: $\mathcal{A} = 3\%$, $|Q| = 1000$, $d = 2$. (d) effect of d: $\mathcal{A} = 3\%$, $|Q| = 1000$, $N = 3 \times 10^6$.

Fig. 10. Approximation quality of the GEQA algorithm.



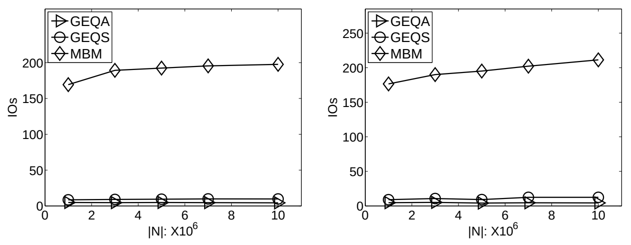
(a) IOs: effect of $\mathcal{A}$, $|Q| = 1000$, *UN* data set. (b) IOs: effect of $\mathcal{A}$, $|Q| = 1000$, *CA* data set. (c) IOs: effect of $|Q|$, $\mathcal{A} = 3\%$, *UN* data set. (d) IOs: effect of $|Q|$, $\mathcal{A} = 3\%$, *CA* data set.

Fig. 11. Query cost of different algorithms: IOs analysis, *UN* and *CA* data sets, $d = 2$, $N = 3 \times 10^6$.

especially in higher dimensions, since it mainly depends on the classical nearest neighbor search which has been extensively studied. It also features with the excellent query cost (we will see shortly experimental results on this issue), since only a nearest neighbor search is required after finding the MEB(Q) which is also a well-explored topic and can be done in linear time to the size of the query group. Now, given the fact that it has excellent approximation ratios (has almost identical enclosing distances to the optimal answer), the GEQA algorithm becomes an excellent candidate for answering a GEQ query in practice for large databases in the multi-dimensional space.

The query cost of various algorithms. We next study the query cost of our approximation algorithm, GEQA, and the new exact algorithm GEQS, comparing to the state-of-the-art method MBM [27]. Figure 11 summarizes the comparison in two dimensions, when we vary various parameters for the data sets and the query groups. In general, the average number of IOs for one query using the MBM method is around 200. In contrast, both the GEQA and GEQS only require around 10 IOs, and the GEQA algorithm consistently incurs the least amount of IOs. These results indicate that in two dimensions, the new algorithms proposed in this work is one order of magnitude better than the state-of-the-art approach.

Specifically, Figures 11(a) and 11(b) show the IOs of different algorithms on *UN* and *CA* data sets when we fix the query group size and vary the query group's covering area. Figures 11(c) and 11(d) show the same experiment, but we fix the query group's covering area and change the query group size. These results indicate that both the query group area and the query group size do not have a significant impact to the query cost of various algorithm. This is not that surprising, since our analysis suggests that the optimal query answer is very close to the center for the minimum enclosing ball of the query group. This indicates that the query cost is more affected by this center, which is hardly affected by the query group's area and size. In all cases, the query costs for the GEQA and GEQS algorithms are almost 20 times better than the MBM method.

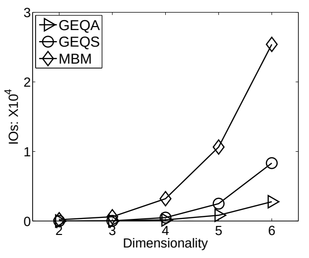


(a) IOs: vary N, *UN* data set. (b) IOs: vary N, *CA* data set.

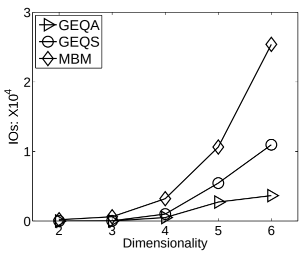
Fig. 12. Scalability w.r.t $|P|$: $\mathcal{A} = 3\%, |Q| = 1000$, $d = 2$.

We then test the scalability of these algorithms with respect to the size of the data set P. Using the *UN* and *CA* data sets in two dimensions, we vary the data set size from 1 million to 10 million. Figure 12 shows that our algorithms have an excellent scalability for larger data sets, e.g., they still just have less than 20 IOs for data sets that have 10 million points. Their query costs almost stay as a constant as N increases. On the hand, we can observe an an increasing query cost for the MBM method when the size of the data set increases.

Lastly, we study the performance of these algorithms under higher dimensions and the results are shown in Figures 13(a) and 13(b), for the *UN* and *RC* data sets respectively. As we can see, the query cost for all algorithms increases in higher dimensions. However, our algorithms have a much slower pace of increasing, especially the GEQA algorithm. The performance gap between the MBM method and our algorithms becomes

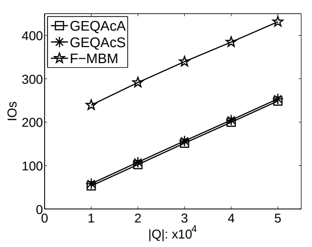


(a) IOs: vary d, *UN* data set.

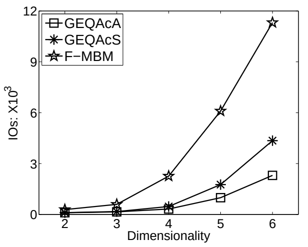


(b) IOs: vary d, *RC* data set.

Fig. 13. Query cost in higher dimensions: $\mathcal{A} = 3\%$, $|Q| = 1000$, $N = 3 \times 10^6$.



(a) IOs: vary $|Q|$, $d = 2$.



(b) IOs: vary d, $|Q| = 1000$.

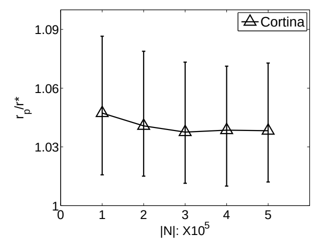
Fig. 14. Approximate convex hulls: $|\mathcal{AC}_Q| = 50$, $\mathcal{A} = 3\%$, $N = 3 \times 10^6$, *UN* data set.

larger in higher dimensions. In six dimensions, the MBM method takes almost $30,000$ IOs for one query, while the GEQS algorithm requires about $10,000$ IOs and the GEQA algorithm only needs less than 2000 IOs (on 3 million points). We can further reduce the query cost of the GEQA algorithm easily by using an approximate nearest neighbor search algorithm, instead of relying on the R-tree for the nearest neighbor search in high dimensions (see Section 7.3).

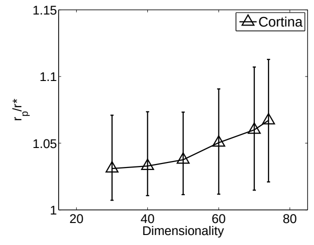
Given these results, we conclude that in relatively high dimensions, the best algorithm for the GEQ problem is to use the GEQA algorithm to obtain a high quality approximation using an efficient nearest neighbor search method. If an optimal answer must be obtained, then the GEQS algorithm could be used. However, there is a limitation when using the GEQS algorithm in high dimensions. Computing the furthest voronoi diagram becomes challenging in this case. Thus, when d goes beyond six dimensions, one should simply use the GEQA algorithm to get a high quality approximate answer (see further discussion on this issue in Section 7.3).

7.2 Query groups of large size

There are cases where not only $|Q|$ is large, but also $|\mathcal{C}_Q|$. These cases rarely happen in practice as query points in real applications hardly arrange themselves such that each of them is a vertex of a convex polygon with high probability. When this does happen, we can apply the idea of the approximate convex hull as discussed in Section 6.1 to remedy the problem. Figure 14 reports the performance of such an approach. We can apply both the GEQA and GEQS algorithms on the approximate convex hull $\mathcal{AC}_Q$ of the query group Q. We denote such algorithms as GEQAcA and GEQAcS respectively,



(a) effect of N : $\mathcal{A} = 3\%$, $|Q| = 1000$, $d = 30$.



(b) effect of d: $\mathcal{A} = 3\%$, $N = 3 \times 10^5$.

Fig. 15. Approximation quality of GEQA in very high dimensions: $\mathcal{A} = 3\%$, $|Q| = 1000$,*Cortina* data set.

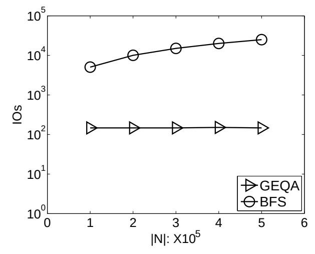
and comparing their query costs against the disk-based version of the MBM method (F-MBM) from [27]. In this experiment we fix the number of vertices in the approximate convex hull $\mathcal{AC}_Q$ to 50. In terms of the query cost, clearly our algorithms outperform the F-MBM algorithm and the GEQAcA algorithm has the best query cost. For all algorithms, clearly, their query costs increases almost linearly while $|Q|$ increases as shown in Figure 14(a). We also test these algorithms in higher dimensions and Figure 14(b) shows an increasing query cost for all algorithms. Our algorithms remain the lower query costs than the F-MBM method. Note that when dimension increases, for our algorithms the dominant query cost is contributed by querying the index structure, rather than computing the approximate convex hull $\mathcal{AC}_Q$.

There is a trade-off for the better efficiency achieved by our algorithms. In this case, both GEQAcA and GEQAcS algorithms only give approximate answers, while the F-MBM method gives an exact answer. However, from the discussion in Section 6.1, the error introduced by the approximation is independent of $|Q|$ and is only determined by $|\mathcal{AC}_Q|$. When $|\mathcal{AC}_Q| = 50$, ϵ is less than 0.001. Hence, both our algorithms achieve very good approximation qualities. On average, $r_p/r*$ for our algorithms is very close to 1 and the results are very similar to those reported in Figure 10. Hence, unless an exact answer must be obtained, one could use the the GEQAcA algorithm (since in this case, the GEQAcS also is an approximation algorithm and it has a higher query cost than the GEQAcA algorithm as shown in Figure 14) to find a good approximation when Q becomes too large.

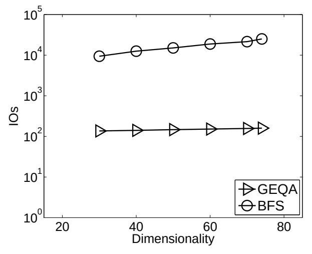
7.3 Performance of GEQA in Very High Dimensions

Next, we study the performance of GEQA in very high dimensions (d up to 74). Recall that in high dimensions, finding both the exact minimum enclosing ball and the nearest neighbor become too expensive to be practical. So we turn to use the approximation algorithms. For finding the approximate minimum enclosing ball, we adopt the $(1 + \epsilon)$-approximation algorithm in [22]. For finding the approximate nearest neighbor, we use the LSB-tree method proposed in [31] (with the default parameter values used in [31]).

In our study, we use the *Cortina* data set as the default data set, since it has the largest number of points

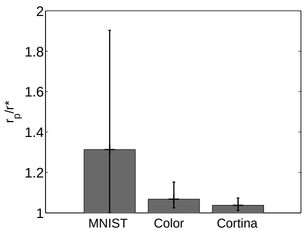


(a) IOs: vary N.

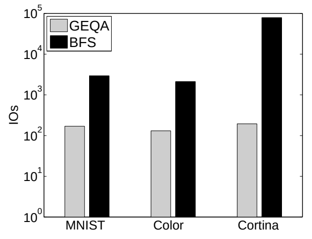


(b) IOs: vary d.

Fig. 16. Query cost in very high dimensions: $\mathcal{A} = 3\%$, $|Q| = 1000$, *Cortina* data set.



(a) approximation quality.



(b) query cost.

Fig. 17. Results on all datasets, $\mathcal{A} = 3\%$, $|Q|$ =1000, N=60,000, 68,040, 1,088,864 and d=50, 32, 74.

(approximately 1 million points) and the highest dimensionality (74). To test the scalability of our algorithm, we vary both N and d in the data set used for our experiments. When we vary N, we simply randomly sample N points from *Cortina*; when we vary d, we simply take the first d dimensions from any point in *Cortina*. In the default set up, $d = 30$ and N =300,000. Finally, we tested query groups of different sizes and different distributions, in similar ways as we did in low to relatively high dimensions in Section 7.1. The results are similar. Hence, we only show the results (average of 1000 queries) using the default query group with 1000 points and the rc distribution. The only available method for exact GEQ search in very high dimensions is to linearly scan the entire data set, and we refer to this method as the *BFS* method. It is also used to find the exact answer and the exact enclosing distance in order to calculate the approximation ratio of the GEQA algorithm.

The approximation quality of GEQA. Figure 15 plot the average approximation ratio of the GEQA algorithm using the *Cortina* data set, as well as the 5%-95% confidence interval. Although the approximation quality does degrade compared with the results in low dimensions (Figure 10), it is still much better than its worst case theoretical bound. In particular, the approximation quality of the GEQA on this data set stays within 1.12 in the worst case and the average approximation quality is below 1.08, when d becomes 74 and N =300,000 (Figure 15(b)). Figure 15(a) indicates that the number of points in the data set does not significantly affect the approximation quality, as N increases from 100,000 to 500,000. This is easy to understand since the points are randomly sampled from the original 1 millions points in *Cortina* data set when we vary N, meaning that these data sets have almost identical distributions. Figure 15(b) shows that the approximation quality of GEQA does degrade when the dimensionality of the data set increases, but at a fairly slow pace. In particular, its average approximation quality drops from 1.04 in 30 dimensions to 1.07 in 74 dimensions. The variance for the approximation quality of GEQA also increases with d, but even in 74 dimensions, this variance is still considerably small (ranging from 1.02 to 1.12 as seen in Figure 15(b)).

The query cost. We then compare the query cost of GEQA with the exact solution, the *BFS*. Figure 16 summarizes the comparison when we vary the data size, and the dimensionality, using the *Cortina* data set. In general, the average number of IOs for one query using the GEQA method is around 130 to 160 in all cases (even when d becomes 74 or N becomes 500,000), and the IOs of the BFS method is approximately two orders of magnitude larger or more. More importantly, Figures 16(a) and 16(b) show that GEQA has excellent scalability w.r.t. both N and d (its query cost increases very slowly when N or d increases). In contrast, the query cost of BFS increases linearly with both N and d.

More data sets. We also tested the performance of GEQA using other high dimensional, real data sets. Figure 17(a) compares the average approximation ratio (and the 5%-95% confidence interval) of GEQA using all three real data sets (all available points and dimensions in each data set respectively). Clearly, for all data sets we have tested, the average approximation ratio is below 1.3 in the worst case and the overall worst case approximate ratio is below 1.9 (both happen on the *MNIST* data set, it also yields the largest variance). The approximation variance is very small in *Color* and *Cortina*, and fairly small in *MNIST* as well.

Figure 17(b) compares the average IOs of one query for GEQA and BFS on all data sets. Clearly, in all data sets, GEQA outperforms the BFS by at least 1.5 to 3 orders of magnitude. Given these results, we conclude that the GEQA algorithm is an excellent solution for the GEQ problem in any dimension (from as low as 2 to as high as 74) because of its high approximation quality, low query cost, excellent scalability, and very simple implementation in practice.

8 CONCLUSION

This paper studies an important aggregate nearest neighbor query type (the GEQ problem) that has many real life applications. Our work presents novel exact and approximate methods that have significantly outperformed the existing method for this problem. Our approximate method is simple, efficient and effective. It has a worst-case $\sqrt{2}$-approximation ratio when exact nearest neighbor of a point can be found and it is close to 1 in practice. In high dimensions, this approximate algorithm could be adapted to work with approximate nearest neighbors and approximate minimum enclosing balls. It gives a (close to) constant approximations and retains its superior query performance and approximation quality in practice. Future work includes extending these algorithms to cope with moving objects and data in the road network databases.

References

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] Open street map. http://www.openstreetmap.org.

[3] Qhull, the Quickhull algorithm for convex hulls. http://www.qhull.org.

[4] S. Arya and D. M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17(3-4):135–152, 2000.

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of ACM*, 45(6):891–923, 1998.

[6] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Survey*, 23(3):345–405, 1991.

[7] M. Bādoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *STOC*, 2002.

[8] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.

[9] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications*. Springer, 1997.

[10] C. Böhm. A cost model for query processing in high dimensional data spaces. *ACM Transaction on Database Systems*, 25(2):129–178, 2000.

[11] K. Chakrabarti, K. Porkaew, and S. Mehrotra. The Color Data Set. http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.data.html.

[12] B. Cui, B. C. Ooi, J. Su, and K.-L. Tan. Contorting high dimensional data for efficient main memory kNN processing. In *SIGMOD*, 2003.

[13] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, 2003.

[14] K. Fischer and B. Gartner. The smallest enclosing ball of balls: combinatorial structure and algorithms. In *SoCG*, 2003.

[15] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[16] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *FOCS*, 1993.

[17] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[18] M. Hadjieleftheriou. The spatialindex library. http://www.research.att.com/∼marioh/spatialindex/index.html.

[19] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2), 1999.

[20] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

[21] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, 2005.

[22] P. Kumar, J. S. B. Mitchell, and E. A. Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *ACM Journal of Experimental Algorithmics*, 8, 2003.

[23] Y. LeCun and C. Cortes. The MNIST Data Set. http://yann.lecun.com/exdb/mnist/.

[24] H. Li, H. Lu, B. Huang, and Z. Huang. Two ellipse-based pruning methods for group nearest neighbor queries. In *GIS*, 2005.

[25] K. Mouratidis, D. Papadias, and S. Papadimitriou. Tree-based partition querying: a methodology for computing medoids in large spatial datasets. *VLDB J.*, 17(4):923–945, 2008.

[26] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, 2004.

[27] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst.*, 30(2):529–576, 2005.

[28] G. Proietti and C. Faloutsos. Analysis of range queries and self-spatial join queries on real region datasets stored using an R-tree. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):751–762, 2000.

[29] K. Rose and B. S. Manjunath. The CORTINA Data Set. http://www.scl.ece.ucsb.edu/datasets/index.htm.

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

[31] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, 2009.

[32] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *PODS*, 1996.

[33] B. Yao, F. Li, and P. Kumar. Reverse furthest neighbors in spatial databases. In *ICDE*, 2009.

[34] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the distance: an efficient method to kNN processing. In *VLDB*, 2001.

[35] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using core sets. In *SoCG*, 2004.

[36] D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *VLDB*, 2006.

PLACE PHOTO HERE

Feifei Li received the BS degree in computer engineering from Nanyang Technological University in 2002 and the PhD degree in computer science from Boston University in 2007. He has been an assistant professor in the Computer Science Department, Florida State University, since 2007. His research interests include data management, data structures, and databases, as well as security issues in data management.

PLACE PHOTO HERE

Bin Yao received both the BS degree and the MS degree in computer science from South China University of Technology in 2003 and 2007, respectively. He has been a PhD student in the Computer Science Department, Florida State University, since 2007. His research interests include databases and data management. In particular, query processing in spatial database, relational database, text and multimedia databases.

PLACE PHOTO HERE

Piyush Kumar received his B.Sc. degree in Mathematics and Computing from IIT Kharagpur in 1999. He was awarded a PhD in computer science from Stony Brook University in 2004. Prior to joining FSU as an Assistant Professor in 2004, he was also a visiting scientist at MPI-Saarbruecken, Germany. In 2010, he was promoted with tenure to the rank of Associate Professor. His research interests include algorithms, computational geometry, optimization, databases, computer graphics and scientific computing. He is a recipient of the NSF CAREER Award, the AFOSR Young Investigator Award and the FSU Innovator Award.