Flexible Aggregate Similarity Search

Yang Li[1] Feifei Li[2] Ke Yi[3] Bin Yao[2] Min Wang[4]
rainfallen@sjtu.edu.cn[1], {lifeifei, yao}@cs.fsu.edu[2], yike@cse.ust.hk[3], min.wang6@hp.com[4]
Shanghai JiaoTong University[1], Florida State University[2]
Hong Kong University of Science and Technology[3], HP Labs China[4]

ABSTRACT

Aggregate similarity search, a.k.a. aggregate nearest neighbor (Ann) query, finds many useful applications in spatial and multimedia databases. Given a group Q of M query objects, it retrieves the most (or top-k) similar object to Q from a database P, where the similarity is an aggregation (e.g., sum, max) of the distances between the retrieved object p and *all* the objects in Q. In this paper, we propose an added flexibility to the query definition, where the similarity is an aggregation over the distances between p and any subset of ϕM objects in Q for some *support* $0 < \phi \leq 1$. We call this new definition *flexible aggregate similarity* (Fann) search, which generalizes the Ann problem. Next, we present algorithms for answering Fann queries exactly and approximately. Our approximation algorithms are especially appealing, which are simple, highly efficient, and work well in both low and high dimensions. They also return near-optimal answers with guaranteed constant-factor approximations in any dimensions. Extensive experiments on large real and synthetic datasets from 2 to 74 dimensions have demonstrated their superior efficiency and high quality.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management—*Systems. Subject: Query processing*

General Terms

Algorithms

1. INTRODUCTION

Aggregate similarity search extends the classical similarity search problem with a group Q of query objects, and the goal is to retrieve the most (or top-k) similar object to the query group from the underlying database P, where similarity is defined by applying an aggregation function (usually sum or max) over the set of distances between the retrieved object and every query object [15–19, 23]. It is also commonly known as the aggregate nearest neighbor (Ann) or group nearest neighbor query. This generalizes the classical nearest neighbor (NN) search, while offering richer semantics with broader applications in spatial and multimedia databases, as pointed out by previous studies [15, 17–19, 23]. Due to its importance, this problem has already been studied in the Euclidean space [15,17,18], the road-network space [23], and the general metric space [19]. However, a major limitation of Ann search is that all objects in the query group must be involved in defining the optimal answer. As a result, any subset of points in the query group could affect the quality and the usefulness of the query answer. In other words, Ann requires that an object from P must be similar to all objects in Q in order to qualify as a good answer, which could be too restrictive in practice.

We observe that in many practical applications, it is often good enough, and in some cases actually desired, to find similar objects to a fraction of the objects from the query group. For example, suppose P is a collection of candidate locations, and Q is a set of potential customers. When trying to find a location to hold a marketing campaign from P, instead of trying to meet all customers where the meeting place should minimize the total (or maximum) traveled distance of all customers, it is often desired to find a place that is good for a certain fraction, say 50%, of the customers. In this case, the meeting place should be close (in terms of the total or maximum traveled distance) to 50% of the customers, regardless of which customers are in this 50% (i.e., meet 50% of potential customers). More precisely, a better and more general approach is to allow the user to specify a *support* $0 < \phi \leq 1$, and the goal is to find the best (or the top-k) object from the database that is the most similar to any $\phi|Q|$ objects from the query group. We call it *flexible aggregate similarity search*, or Fann in short. Clearly, Ann is a special instance of Fann when $\phi = 1$.

Fann also finds applications in similarity search in multimedia databases, which usually map objects to points in high dimensional feature spaces. When the query Q consists of a set of objects (e.g. images), the Ann will find an object in the database that is similar to all query objects, which might be too restrictive in many cases. While Fann returns an object of a certain support, namely being similar to $\phi|Q|$ of the query objects. This allows the user to be less careful (in other words, more flexible) when formulating his/her query group Q. When returning the top-k objects (called the k-Fann problem in Section 6.1), the diversity of the query answers also increases: the k objects might be similar to k different subsets of $\phi|Q|$ query objects each.

SIGMOD'11, June 12–16, 2011, Athens, Greece.

This added flexibility could make the problem a lot harder: a FANN query implicitly incorporates $\binom{|Q|}{\phi|Q|}$ ANN queries as each subset of $\phi|Q|$ objects of Q could be the best subset that an object in the database is similar to. Indeed, if one wants to answer a FANN query exactly, the cost is quite high. In Section 4 we present two algorithms based on standard techniques for answering FANN queries exactly, but the experiments show that they could be expensive; especially in high dimensions, they could be as bad as linear scans.

Therefore, we investigate approximation algorithms that greatly improve the efficiency while returning near-optimal answers. We present two such algorithms (for the sum and max versions of the problem respectively) in Section 5, which have the following appealing features:

- Guaranteed approximation ratios: We prove that the two algorithms return answers with guaranteed approximation ratios of 3 and $1+2\sqrt{2}$, for the sum and max versions of the problem, respectively, and more importantly, they hold in *any* dimensions. Note that since FANN degenerates into the ANN problem when $\phi = 1$, our results also imply a 3-approximation for the sum ANN problem, for which only heuristics are known and they work only in low dimensions [16–19].
- Excellent query answer quality in practice: The proved approximation ratios hold for the worst data. In practice, extensive experiments (in Section 7) on real and synthetic datasets show that the actual approximation ratios are much lower, usually below 1.3.
- Superior efficiency: The benefit of not returning the exact answer is superior efficiency. In low dimensions ($d = 2$), the algorithms answer a query Q with $|Q| = 300$ on a dataset of 2 million records in just about 1 millisecond; in high dimensions ($d = 30$), a query on a dataset of similar scale takes 0.01 to 0.1 second. Detailed experimental results are provided in Section 7.
- Simplicity: Our algorithms are actually very simple. They reduce the problem to a few instances of the standard nearest neighbor (NN) search, which is a well studied problem, and efficient solutions are known in both low and high dimensions. This is actually the main leverage on which these algorithms easily and nicely extend to high dimensions.

Below we first formally define the FANN problem in Section 2 and survey the related work in Section 3. Then we present two exact algorithms aiming at low and high dimensions respectively, in Section 4. We describe the two approximation algorithms, together with the proofs on their approximation ratios, in Section 5. We then talk about the extensions of these algorithms in Section 6, present the experimental results in Section 7, and conclude in Section 8.

2. PROBLEM FORMULATION

We use P to denote the set of points in the database, and Q as the set of query points, where $|P| = N$ and $|Q| = M$. Both P and Q are in a metric space with the distance function $\delta(p, q)$ defined for any two points. Let g be the aggregation function, either sum or max, and ϕ be a support value in $(0, 1]$. We further define $g(p, S)$, for any point p and a group of points S, as:

$$g(p, S) = g(\delta(p, q_1), \ldots, \delta(p, q_{|S|})), q_i \in S \text{ for } i = 1, \ldots, |S|,$$

i.e., it is the aggregate distance between p and all points in S aggregated by g. The flexible aggregate similarity search (FANN) problem is formally defined as follows.

Definition 1 (FANN query) Given P, Q, δ, g and ϕ, a FANN query returns:

$$(p^*, Q_\phi^*) = \operatorname*{argmin}_{p \in P, Q_\phi \subseteq Q} g(p, Q_\phi), \text{ where } |Q_\phi| = \lceil \phi M \rceil.$$

Let $r^* = g(p^*, Q_\phi^*)$ denote the optimal aggregate distance. Since finding the optimal p^* achieving r^* is expensive, we will mostly focus on finding approximately optimal answers. For any $\beta \geq 1$, we say that (p, Q_ϕ) is a *β-approximate* answer to the FANN query if $Q_\phi \subseteq Q$, $|Q_\phi| = \lceil \phi M \rceil$, and

$$r^* \leq g(p, Q_\phi) \leq \beta r^*.$$

For convenience, we will ignore the ceiling and assume that ϕM is an integer. A first observation is that, for any p, the Q_ϕ that minimizes $g(p, Q_\phi)$ consists of the ϕM points in Q closest to p. Thus, if we define Q_ϕ^p as such a set of points, the definition of a FANN query can be stated as finding

$$p^* = \operatorname*{argmin}_{p \in P} r_p, \text{ where } r_p = g(p, Q_\phi^p). \quad (1)$$

Similar to previous studies for the ANN problem, in most applications, P is large and disk-based, and Q is small and memory resident. We assume this setting by default, but will discuss the case when Q becomes disk-resident in Section 6. We assume the d-dimensional Euclidean space as the default metric space, and briefly discuss general metric spaces in Section 6. We summarize the main notations in Figure 1.

Symbol	Description
$\mathcal{B}(c, r)$	the ball centered at c with radius r
$\delta(p, q)$	distance between p and q
g	sum or max
$g(o, S)$	$g(\delta(o, s_1), \ldots, \delta(o, s_{\lvert S\rvert}))$ for all $s_i \in S$
MEB(S)	minimum enclosing ball of S
M, N	size of Q and P respectively
nn(o, S)	the nearest neighbor of o in S
Q_ϕ^p	ϕM nearest neighbors of p in Q
(p^*, Q_ϕ^*)	the optimal answer to FANN on P, Q, ϕ
r^*	optimal agg. similarity distance $g(p^*, Q_\phi^*)$

Figure 1: List of notations.

3. RELATED WORK

Study on aggregate similarity search was initialized by Papadias et al. [17] and Li et al. [16], where sum ANN queries have been examined in Euclidean spaces of low dimensions. The state-of-the-art exact algorithm appears in [18], which is an R-tree based MBM method. It adopts the typical branch-and-bound methodology using the R-tree and relies on the triangle inequality as the main principle for pruning the search space. Of course, the details will vary based on the aggregate function used. As such, the MBM method is a good heuristic algorithm that works well in low dimensions (2 or 3 dimensions). Razente et al. [19] used the same idea for other metric spaces with distance-based indexing structures, such as the M-tree [7]. The performance of these algorithms degrades quickly as dimensionality increases.

To get around the curse-of-dimensionality problem of the MBM method, approximation methods have been proposed, but only for max ANN queries in the Euclidean space [15].

The basic idea is to find the center of the minimum enclosing ball (MEB) of the Q, and then simply return the nearest neighbor of this center from P. Li et al. [15] showed that this simple method gives a $\sqrt{2}$-approximate answer to the max ANN query in any dimensions, and its query cost is essentially the same as one standard NN query. Alongside the MBM method, Papadias et al. [18] also proposed a few heuristic for approximating ANN queries, but with no provable approximation ratios.

All of the above works study the ANN problem. However, in the FANN problem, we are looking for the p^* that minimizes its aggregate distance to any subset of ϕM query points. If one were to adapt the existing ANN solutions, $\binom{M}{\phi M}$ subsets of the query points would have to be considered, an exponential blowup. Thus, none of the above results can be used to solve the FANN problem efficiently.

The standard NN search is also very relevant to our study. In low dimensions, the R-tree provides efficient exact algorithms using either the depth-first [21] or the best-first [11] search algorithms. They do not provide theoretical guarantees on the worst-case query cost, but are in general very efficient in answering exact NN queries in low dimensions. On the other hand, the BBD-tree [1] finds $(1+\epsilon)$-approximate nearest neighbors in worst-case $O((1/\epsilon^d)\log N)$ time where d is the dimensionality.

It is well known that the R-tree, and in general any space-partitioning scheme, gives poor performance beyond 6 dimensions [2, 4]. For exact NN search in high dimensions, iDistance [12] is the state of the art, but can still be quite expensive. As approximation can be often tolerated in high dimensional NN search, more efficient approximation algorithms have been designed. In particular, the techniques based on *locality sensitive hashing (LSH)* [10] have been shown to be highly efficient while returning near-optimal NN results. Currently, the most practical LSH based solution is the LSB-tree [22], which combines the LSH idea and space filling curves. By doing so, it is able to return 4-approximate NNs with high probability; on typical data sets, the approximation ratio is often much lower (usually close to 1). It also has a bounded query cost of $O(\sqrt{dN/B}\log_B N)$ IOs for an NN search, where B is the disk page size.

4. EXACT METHODS

A straightforward exact method for answering a FANN query is to do a linear scan of all points in P and find the optimal p^* by its definition. More precisely, for every point $p \in P$, we find the set Q_ϕ^p, namely, the ϕM nearest neighbors of p in Q and calculate $r_p = g(p, Q_\phi^p)$. Then, we find (p^*, Q_ϕ^*) with the smallest r_p. We denote this method as *BFS* (brute-force search).

Next, we present two imporved exact methods. The first method is based on the R-tree and can be seen as a generalization of the techniques in [18]. As it is based on the R-tree, it works only in low dimensions. The second method is based on the TA algorithm [8, 9] and works for any dimensions.

4.1 The R-tree algorithm

For any node in the R-tree, we can calculate the minimum possible distance from its MBR (minimum bounding rectangle) b to every query point q in Q, denoted as $\text{mindist}(q, b)$ [11, 21]. Let Q_ϕ^b be the subset of ϕM points from Q that have the ϕM smallest mindist values to b. Clearly, for any $p \in b$, we have

$$r_p \geq g(\text{mindist}(q_1, b), \ldots, \text{mindist}(q_{\phi M}, b)), q_i \in Q_\phi^b, \quad (2)$$

which yields a lower bound in the aggregate distance r_p for any point p inside b.

Let the MBR of the query group Q be b_Q. Another lower bound for r_p, which is cheaper to compute, but not as tight as (2), is as follows. For any MBR node b in an R-tree, we find the minimum possible distance between b_Q and b, denoted as $\text{mindist}(b_Q, b)$. Then for any $p \in b$, we have

$$r_p \geq \begin{cases} \phi M \cdot \text{mindist}(b_Q, b), & \text{if } g = \text{sum}; \\ \text{mindist}(b_Q, b), & \text{if } g = \max. \end{cases} \quad (3)$$

Based on (2) and (3), we can easily construct a search algorithm for FANN queries using an R-tree built on P. Specifically, when a leaf node of the R-tree is accessed, for each point p stored in the leaf, we find Q_ϕ^p and compute the aggregate distance $r_p = g(p, Q_\phi^p)$. When we encounter an internal R-tree node, we first compute (3) and then (2) and check if it is higher than the best candidate answer found so far. If so we skip the entire subtree rooted at this internal node; otherwise we add this node to a queue. The queue is sorted in the ascending order of their lower bounds on the aggregate distance, and we will visit the nodes from the queue in order. We denote this algorithm as the *R-tree* method.

We point out that when $\phi = 1$, the FANN problem reduces to the ANN problem, and this *R-tree* method described above also degenerates into the MBM method [18] for ANN.

4.2 The List algorithm

We conceptually build M lists, one for each query point q_i in Q. The list for $q_i \in Q$ sorts all points in P in the ascending order of their distances to q_i. In particular, we refer to the jth element in the ith list as a pair $(p_{i,j}, \delta_{i,j})$ where $\delta_{i,j} = \delta(p_{i,j}, q_i)$, $p_{i,j} \in P$ for $j = 1, \ldots, N$. By doing so, for any point $p \in P$, we can view p as an object with M attributes with its ith attribute taking value $\delta(p, q_i)$, and all points in P are given in M lists, sorted according to each of the M attributes, respectively. The aggregated "score" of p, $g(p, Q_\phi^p)$ is the sum or max of the ϕM smallest attribute values of p, which is monotone w.r.t. the M attributes. This is exactly the setting where the TA algorithm [9] applies. This allows us to design the *List* algorithm below.

Algorithm 1: List(P, Q, ϕ, g)

```
1  let ℓ_i = 1 and τ_i = δ_{i,ℓ_i} for i = 1, ..., M;
2  set τ = g(smallest φM values from τ_i s);
3  set p_o = null and δ_o = +∞;
4  while true do
5      let η = argmin_{i∈[1,M]} δ_{i,ℓ_i};
6      set p' = p_{η,ℓ_η} and compute δ' = g(p', Q_φ^{p'});
7      if δ' < δ_o then
8          set p_o = p' and δ_o = δ';
9      if ℓ_η < N then
10         set ℓ_η = ℓ_η + 1 and τ_η = δ_{η,ℓ_η};
11     else output (p_o, Q_φ^{p_o}); return;
12     update τ if smallest φM values in τ_i s have changed;
13     if δ_o < τ then
14         output (p_o, Q_φ^{p_o}); return;
```

The basic idea of the *List* algorithm is to perform sorted access to the M lists, while maintaining a lower bound for the best possible aggregate distance for any unseen point. We maintain one pointer per list (the ℓ_i's); initially they point to the first elements of the lists. We set the ith threshold value τ_i to be the attribute value of the point pointed by ℓ_i (line 1). In each of the subsequent steps, we pick the list whose current element has the smallest value, say the ηth list (line 5). We retrieve the ℓ_ηth element from the ηth list (line 6). This element gives a point p' and we compute its aggregate distance by applying g over p' and its ϕM nearest neighbors from Q (line 6). We keep the best candidate answer (the point and the achieved distance) so far in p_o and d_o (lines 3, 7–8). Then we move the ηth pointer (ℓ_η) down the list by one position, and update the ηth threshold value τ_η accordingly (lines 9–11). Clearly, for any unseen object from the ith list, its minimum possible ith attribute value will be at least τ_i, which indicates that applying g over the current ϕM smallest threshold values gives a lower bound on the best possible aggregate distance of any unseen point.

Implementation. Note that we do not have to materialize the M lists in order to run the algorithm above. The observation is that the jth element in the ith list, $(p_{i,j}, \delta_{i,j})$, is simply the jth nearest neighbor of q_i from P and the corresponding distance. Thus, lines 5 and 6 in Algorithm 1 can be easily done by finding the ℓ_ith nearest neighbor of q_i from P (similarly for line 1 and 10), as long as we have an index that can return the nearest neighbors for any given q_i in the ascending order of their distances to q_i. This is the standard k-NN problem and has been well studied.

In low dimensions, we can index P using an R-tree. We do not have to find the ℓ_ith nearest neighbor of q_i from scratch every time when we move down the ith list. Rather, with some simple bookkeeping, the R-tree's nearest neighbor search can be carried out incrementally, that is, to find the jth nearest neighbor of q_i, we can resume the search from the end of the search for the $(j-1)$th nearest neighbor. In higher dimensions, we can index P using the iDistance index [12] and also find the jth nearest neighbor of q_i incrementally. Alternatively, we can index P using a LSB-tree [22] for faster nearest neighbor retrieval. However, since the LSB-tree only returns approximate NNs, using a LSB-tree over P no longer guarantees that *List* will return an exact answer. Nevertheless, we can easily prove the following result (the proof is quite straightforward and omitted).

Proposition 1 *Given a β-approximate kNN algorithm,* List *gives a β-approximation for the* Fann *problem.*

5. APPROXIMATE METHODS

Our exact methods for the Fann problem outperform the *BFS* approach, but they are still quite expensive, especially on large datasets (as shown in our experimental study). Furthermore, it is well known that in high dimensions, even the standard NN search itself will require a linear scan of the dataset in most cases we are to find exact answers (see [22] and the references therein). Thus, it is not surprising that the exact methods become very expensive as dimensionality increases. In most applications of similarity search, however, approximate answers are often good enough, and past research has shown that allowing approximation can bring significant improvement on the query efficiency [1, 10, 22]. This motivates us to design approximate algorithms for the Fann problem with quality and efficiency guarantees.

5.1 Approximation algorithms for sum Fann

We first concentrate on the sum Fann problem. Our approximate method is given in Algorithm 2, which is denoted as the Asum method. The algorithm is actually very simple, provided that we have a method for standard NN search.

In line 4, recall that $Q_\phi^{p_i}$ simply consists of the ϕM nearest neighbors of p_i in Q and $r_{p_i} = \text{sum}(p_i, Q_\phi^{p_i})$. As Q is small and fits in memory, finding $Q_\phi^{p_i}$ is easy and cheap. That said, the algorithm just finds p_i, the NN in P for each of the M query point q_i, and returns the one with the smallest aggregate distance, in this case the sum of the distances from p_i to its ϕM closest points in Q.

Algorithm 2: Asum (P, Q, ϕ, sum)

```
1 set minr = +∞; α = −1;
2 for i = 1, ..., M do
3     let p_i = nn(q_i, P), where q_i is the ith point in Q;
4     find Q_φ^{p_i} and r_{p_i};
5     if r_{p_i} < minr then
6         set α = i, and minr = r_{p_i};
7 return (p_α, Q_φ^{p_α});
```

We can prove that the algorithm returns a 3-approximate answer to a sum Fann query in any dimensions.

Theorem 1 Asum *returns a* 3*-approximate answer to the* sum Fann *query in any dimensions.*

Proof. Let (p^*, Q_ϕ^*) be the optimal answer to the query group Q, and the optimal aggregate distance is

$$r^* = \sum_{x \in Q_\phi^*} \delta(p^*, x).$$

Let $q^* = \text{nn}(p^*, Q_\phi^*)$, and $p' = \text{nn}(q^*, P)$. Clearly, if $p' = p^*$, Asum will return the optimal answer $p' = p^*$, since $Q_\phi^* \subseteq Q$ and it iterates through all points in Q and finds their nearest neighbors in P as the set of candidate answers.

Consider the case $p' \neq p^*$. Given $p' = \text{nn}(q^*, P)$ we have:

$$\delta(p', q^*) \leq \delta(p^*, q^*). \tag{4}$$

Since $r_{p'} = \text{sum}(p', Q_\phi^{p'})$, where $Q_\phi^{p'}$ are the ϕM nearest neighbors of p' in Q. We have:

$$r_{p'} = \sum_{x \in Q_\phi^{p'}} \delta(p', x) \leq \sum_{x \in Q_\phi^*} \delta(p', x). \tag{5}$$

$$\begin{aligned}
& \sum_{x \in Q_\phi^*} \delta(p', x) \\
\leq\ & \sum_{x \in Q_\phi^*} (\delta(p', p^*) + \delta(p^*, x)) \qquad \text{(triangle inequality)} \\
=\ & \phi M \cdot \delta(p', p^*) + \sum_{x \in Q_\phi^*} \delta(p^*, x) \\
\leq\ & \phi M \cdot (\delta(p', q^*) + \delta(q^*, p^*)) + r^* \quad \text{(triangle inequality)} \\
\leq\ & 2\phi M \cdot \delta(q^*, p^*) + r^* \qquad \text{(by (4))} \\
\leq\ & 2r^* + r^* = 3r^*. \qquad \text{(by (7))}
\end{aligned} \tag{6}$$

The last '$\leq$' holds because $q^* = \text{nn}(p^*, Q_\phi^*)$, i.e., for any $x \in Q_\phi^*$, $\delta(q^*, p^*) \leq \delta(x, p^*)$. Therefore:

$$\phi M \cdot \delta(q^*, p^*) \leq \sum_{x \in Q_\phi^*} \delta(x, p^*) = r^*. \quad (7)$$

By (5) and (6), we have $r_{p'} \leq 3r^*$. Lines 2–6 in Algorithm 2 guarantee that $(p', Q_\phi^{p'})$ is one of the M candidates to be considered, which completes the proof. □

When exact NN search is expensive, we can replace the nn function in Algorithm 2 with approximate NN search. We can show that this still delivers a good approximation.

Theorem 2 *If the exact* nn *function in* Asum *is replaced with a β-approximate NN search, then the* Asum *algorithm gives a $(\beta + 2)$-approximation to the* sum Fann *query.*

Proof. Let p' and q^* be defined similarly as in the proof of Theorem 1. However, we can no longer guarantee to find p' precisely. Instead, we are guaranteed a point p'' that satisfies $\delta(p'', q^*) \leq \beta \cdot \delta(p', q^*)$. Going through the proof of Theorem 1, inequality (4) becomes

$$\delta(p'', q^*) \leq \beta \cdot \delta(p', q^*) \leq \beta \cdot \delta(p^*, q^*), \quad (8)$$

and the derivation in (6) becomes

$$\begin{aligned} \sum_{x \in Q_\phi^*} \delta(p'', x) &\leq \phi M \cdot (\delta(p'', q^*) + \delta(q^*, p^*)) + r^* \\ &\leq \phi M \cdot (\beta + 1)\delta(q^*, p^*) + r^* \quad \text{(by (8))} \\ &\leq (\beta + 1)r^* + r^* = (\beta + 2)r^*. \quad \text{(by (7))} \end{aligned}$$

Thus, the returned answer will be a $(\beta+2)$-approximation. □

5.1.1 Reducing the cost of Asum

The main cost of algorithm Asum is the M NN queries on the data set P, which are quite expensive when M is large, as each NN query involves accessing a disk-based NN index built on P. One idea to reduce this cost is to only run lines 3–6 of the Asum algorithm on a subset of points in Q. Interestingly enough, it turns out that doing so simply on a randomly chosen subset of Q suffices to (almost) preserve the approximation ratio, as shown in the next theorem.

Theorem 3 *For any $0 < \epsilon, \lambda < 1$, executing lines 3–6 of the* Asum *algorithm only on a random subset of $f(\phi, \epsilon, \lambda)$ points of Q returns a $(3+\epsilon)$-approximate answer to the* Fann *query in any dimensions with probability at least $1 - \lambda$, where*

$$f(\phi, \epsilon, \lambda) = \frac{\log \lambda}{\log(1 - \phi\epsilon/3)} = O(\log(1/\lambda)/\phi\epsilon). \quad (9)$$

Proof. Following the proof of Theorem 1, we note that the approximation ratio is guaranteed as long as q^* is one of the points in Q that have gone through lines 3–6 of the algorithm. Of course it is difficult to know which query point in Q is q^* since that depends on the optimal answer p^*, so the algorithm simply tries all possible $q \in Q$.

Now since we execute lines 3–6 of the algorithm only on a randomly chosen subset of Q, q^* may not be one of them. Nevertheless, if some other q' has been chosen that is among the $\epsilon\phi M/3$ closest points in Q_ϕ^* (thus also in Q) to p^*, i.e., $q' \in Q_{\epsilon\phi M/3}^{p^*}$, the proof can still go through except inequality (7), hence (6).

However, in this case given $q' \in Q_{\epsilon\phi M/3}^{p^*}$, we have:

$$\begin{aligned} (\phi M - \frac{\epsilon\phi M}{3})\delta(q', p^*) &\leq \sum_{x \in Q_\phi^{p^*} - Q_{\epsilon\phi M/3}^{p^*}} \delta(x, p^*) \\ &\leq \sum_{x \in Q_\phi^{p^*}} \delta(x, p^*) = \sum_{x \in Q_\phi^*} \delta(x, p^*). \end{aligned}$$

Thus, (7) becomes

$$\phi M \cdot \delta(q', p^*) \leq \frac{1}{1 - \epsilon/3} \sum_{x \in Q_\phi^*} \delta(x, p^*) = \frac{1}{1 - \epsilon/3} r^*,$$

where equality holds in the worst case when the $(\epsilon\phi M/3 - 1)$ closest points to p^* in Q all have distance 0 to p^*, q' is exactly the $(\epsilon\phi M/3)$-th closest point, and the next $(1 - \epsilon/3)\phi M$ closest points are all at the same distance to p^* as q'. Then (6) becomes

$$\frac{2}{1 - \epsilon/3} r^* + r^* \leq 2(1 + \epsilon/2)r^* + r^* = (3 + \epsilon)r^*.$$

Thus it suffices to ensure that at least one of the $\epsilon\phi M/3$ closest points in Q to p^* is chosen. In a random subset of $f(\phi, \epsilon, \lambda)$ points in Q, the probability that none of these $\epsilon\phi M/3$ points is chosen is at most $(1 - \epsilon\phi/3)^{f(\phi,\epsilon,\lambda)}$. Setting $f(\phi, \epsilon, \lambda)$ as (9) makes this probability at most λ. □

Note that by this optimization the number of NN searches we need to issue is independent of the size of Q and dimensionality, which makes the result especially appealing for a large Q and data in high dimensions.

5.1.2 A simpler algorithm for $\phi = 1$

When $\phi = 1$, the sum Fann problem reduces to the sum Ann problem [17, 18]. A simple heuristic approximate algorithm was proposed in [18], denoted as Asum1 (Algorithm 3), which simply returns the nearest neighbor of the geometric median of Q. However, no approximation ratio was proved in [18]. Here we show that this algorithm also gives a 3-approximation for sum Ann and the bound is tight.

Algorithm 3: Asum1 (P, Q, sum)

1 let q_m be the geometric median of Q;
2 **return** $p_m = \text{nn}(q_m, P)$;

The geometric median of a set of points Q is the point q_m minimizing the sum of distances from q_m to the points of Q, where q_m is not required to be one of the points in Q.

Theorem 4 *The* Asum1 *algorithm finds a* 3*-approximation for the* sum Ann *problem using only one nearest neighbor search, and the bound is tight.*

Proof. The query cost is obvious. To show the approximation bound, consider:

$$\begin{aligned} r_{p_m} &= \sum_{q \in Q} \delta(p_m, q) \leq \sum_{q \in Q} (\delta(p, q_m) + \delta(q_m, q)) \\ &\leq \sum_{q \in Q} (\delta(p^*, q_m) + \delta(q_m, q)) \quad (\text{since } p_m = \text{nn}(q_m, P)) \\ &\leq \sum_{q \in Q} (\delta(p^*, q) + \delta(q, q_m) + \delta(q_m, q)) \\ &= \sum_{q \in Q} \delta(p^*, q) + 2 \sum_{q \in Q} \delta(q_m, q) \leq 3r^*. \end{aligned}$$

The last '$\leq$' holds due to the definition of p_m, which is the point in the whole space minimizing the sum of distances to Q, where as p^* is the point from P minimizing the sum of distances to Q and it cannot yield a strictly better result than p_m. Hence, $\sum_{q\in Q}\delta(q_m,q)\leq\sum_{q\in Q}\delta(p^*,q)=r^*$.

To see that this bound is tight, consider the example in Figure 2, where $P=\{p_1,p_2\}$ and $Q=\{q_1,q_2\}$. Clearly, any point on the line segment q_1q_2 (inclusive) is a geometric median for Q. Suppose q_2 is returned as q_m, which means that $p_m=\text{nn}(q_2,P)=p_2$. However, $r_{p_2}=3r-\epsilon$, and in this case $p^*=p_1$ and $r_{p_1}=r$. We can construct this example in any dimension and make ϵ arbitrarily small, which shows that the bound is tight. □

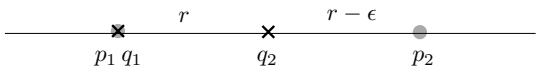


Figure 2: ASUM1's approximation bound is tight.

Remark: Computing the geometric median poses a challenge, as no algorithm can compute it exactly. However, one can use the Weiszfeld algorithm, a form of iteratively re-weighted least squares, to iteratively compute the coordinates of p_m to an arbitrary precision efficiently.

5.2 Approximation algorithms for max FANN

We next present our approximation algorithm for the max FANN problem. Recall that here we aim at finding the point $p\in P$ that minimizes the maximum distance from p to Q_ϕ^p, where Q_ϕ^p consists of the ϕM closest points to p in Q.

For a point $q\in Q$, we also use Q_ϕ^q to denote the set of the ϕM closest points to q in Q, including q itself. We use MEB(S) to denote the *minimum enclosing ball* of a set of points S, namely the smallest ball that fully contains S. Our algorithm, AMAX, is presented in Algorithm 4. This algorithm is actually almost identical to ASUM, except for each $q_i\in Q$, we find the NN in P for c_i, the center of the minimum enclosing ball of $Q_\phi^{q_i}$, instead of q_i itself.

Algorithm 4: AMAX (P, Q, ϕ, max)

```
set minr = +∞; α = −1;
for i = 1, ..., M do
    find Q_φ^{q_i}, and its minimum enclosing ball
    b_i = MEB(Q_φ^{q_i});
    let c_i be the center of b_i;
    let p_i = nn(c_i, P), find Q_φ^{p_i} and calculate r_{p_i};
    if r_{p_i} < minr then
        set α = i, and minr = r_{p_i};
return (p_α, Q_φ^{p_α});
```

Below we show that AMAX returns a $(1+2\sqrt{2})$-approximate answer to the max FANN query, which is slightly worse than our approximation ratio for the sum FANN problem.

We need a few technical lemmas first in order to prove this. Let $\mathcal{B}(c,r)$ be the ball centered at c with radius r. For a point o, a value γ, let $S_{o,\gamma}$ be any set of points such that

$$o\in S_{o,\gamma}\text{ and }S_{o,\gamma}\subseteq\mathcal{B}(o,2\gamma). \tag{10}$$

Lemma 1 *For any $S_{o,\gamma}$, let $\mathcal{B}(s,r_s)=\text{MEB}(S_{o,\gamma})$, then $\delta(o,s)\leq r_s\leq 2\gamma$.*

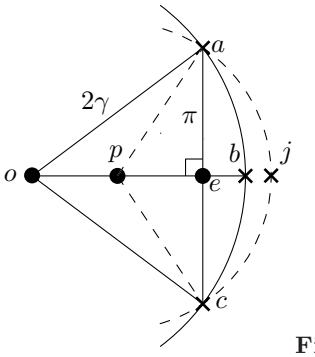


Figure 3: Lemma 2.

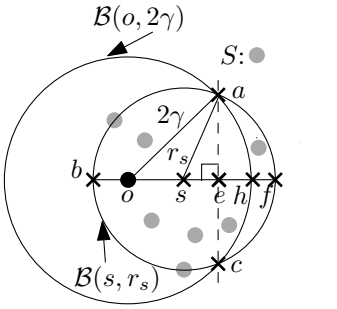


Figure 4: e cannot be outside the line interval os.

PROOF. Given $S_{o,\gamma}\subseteq\mathcal{B}(o,2\gamma)$, $r_s\leq 2\gamma$ is immediate by the definition of the minimum enclosing ball. Next, $o\in S_{o,\gamma}$ and $\mathcal{B}(s,r_s)=\text{MEB}(S_{o,\gamma})$ ensures that $\delta(o,s)\leq r_s$. □

Pick any point e inside $\mathcal{B}(o,2\gamma)$. Extend the segment $\overline{oe}$ (from the e side) and hit $\partial\mathcal{B}(o,2\gamma)$, the boundary of $\mathcal{B}(o,2\gamma)$, at b. Consider the hyperplane $\pi(o,e)$ passing e and orthogonal to $\overline{oe}$. Please see Figure 3 for an illustration in two dimensions. In 2D, $\pi(o,e)$ is a line, whose intersection with $\mathcal{B}(o,2\gamma)$ is a segment $\overline{ac}$. In d dimensions, the intersection of $\pi(o,e)$ with $\mathcal{B}(o,2\gamma)$ is a ball in $d-1$ dimensions; we let a be any point on the boundary of this ball in this case. The hyperplane $\pi(o,e)$ divides $\mathcal{B}(o,2\gamma)$ into two portions, and we denote the one containing b as a *cap* $C(o,e,b)$. Next, let p be any point on the segment $\overline{oe}$, and consider the ball $\mathcal{B}(p,\delta(p,a))$. Extend $\overline{oe}$ and hit $\partial\mathcal{B}(p,\delta(p,a))$ at j. Similarly, let $C(p,e,j)$ be the cap of $\mathcal{B}(p,\delta(p,a))$ separated out by $\pi(p,e)=\pi(o,e)$. We have the following:

Lemma 2 *For any $e\in\mathcal{B}(o,2\gamma)$ and any p on the segment $\overline{oe}$, $C(o,e,b)\subseteq C(p,e,j)$.*

PROOF. Since the two caps $C(o,e,b)$ and $C(p,e,j)$ share the same base, which is the intersection of $\pi(o,e)$ with $\mathcal{B}(o,2\gamma)$, we only need to show that $b\in C(p,e,j)$. As p belongs to the segment $\overline{oe}$, in $\triangle opa$, $\delta(o,p)+\delta(p,a)>\delta(o,a)=\delta(o,b)=\delta(o,p)+\delta(p,b)$. Thus, $\delta(p,j)=\delta(p,a)>\delta(p,b)$. □

Lemma 3 *For any point set $S_{o,\gamma}$ satisfying (10), let $\mathcal{B}(s,r_s)=\text{MEB}(S_{o,\gamma})$, and $\delta(o,s)=z$, then $r_s\leq\sqrt{(2\gamma)^2-z^2}$.*

PROOF. Note that by Lemma 1, $z\leq 2\gamma$, so $\sqrt{(2\gamma)^2-z^2}$ is always a real number. Suppose for contradiction that $r_s>\sqrt{(2\gamma)^2-z^2}$.

First, when this happens, we show that $\partial\mathcal{B}(s,r_s)$ and $\partial\mathcal{B}(o,2\gamma)$ must intersect. Consider the line passing through o and s. It intersects $\partial\mathcal{B}(s,r_s)$ at two points, say b and f, and let the one closer to o of the two be b (see an illustration in Figure 4). Now,

$$\begin{aligned}\delta(o,f) &= \delta(o,s)+\delta(s,f)\\ &= z+r_s\\ &> z+\sqrt{(2\gamma)^2-z^2} \qquad \text{(by the hypothesis)}\\ &\geq \sqrt{z^2+((2\gamma)^2-z^2)}=2\gamma,\end{aligned}$$

which means that f is outside $\mathcal{B}(o,2\gamma)$. Note that the last inequality is due to the fact that for any $x,y\geq 0$, $(x+y)^2\geq x^2+y^2$, hence $x+y\geq\sqrt{x^2+y^2}$.

Since $\mathcal{B}(s,r_s)$ contains both o and f, one inside $\mathcal{B}(o,2\gamma)$ and one outside, and has a radius r_s smaller than 2γ (by

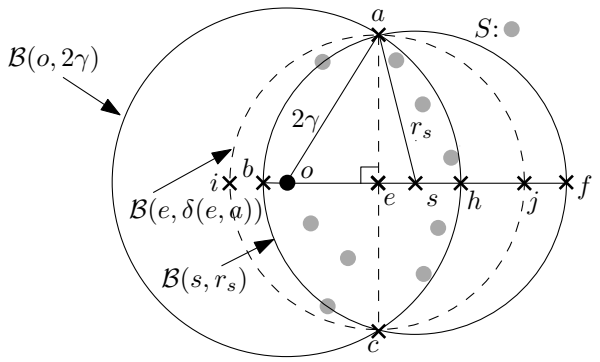


Figure 5: Proof of Lemma 3

Lemma 1), $\partial\mathcal{B}(s, r_s)$ must intersect $\partial\mathcal{B}(o, 2\gamma)$. The intersection in 2D is two points, and a $(d-2)$-sphere in d dimensions. Let a be any point on this $(d-2)$-sphere. Now consider the situation on the plane defined by o, s, and a (Figure 4). On this plane, the $(d-2)$-sphere becomes two points a and c. Suppose $\overline{ac}$ intersects $\overline{bf}$ at e. We first show that e must be inside the line segment $\overline{os}$. Suppose not, i.e., it is to the right of s. In the right triangle $\triangle oae$,

$$\begin{aligned}\delta(a,e)^2 &= \delta(o,a)^2 - \delta(o,e)^2 \\ &= (2\gamma)^2 - (\delta(o,s) + \delta(s,e))^2 \\ &= (2\gamma)^2 - z^2 - \delta(s,e)^2 - 2z\delta(s,e). \quad (11)\end{aligned}$$

While in the right triangle $\triangle sae$,

$$\begin{aligned}&\delta(s,a) \\ =\ &\sqrt{\delta(s,e)^2 + \delta(a,e)^2} \\ =\ &\sqrt{\delta(s,e)^2 + (2\gamma)^2 - z^2 - \delta(s,e)^2 - 2z\delta(s,e)} \quad \text{(by (11))} \\ =\ &\sqrt{(2\gamma)^2 - z^2 - 2z\delta(s,e)},\end{aligned}$$

which contradicts with our assumption that $\delta(s, a) = r_s > \sqrt{(2\gamma)^2 - z^2}$. This means that e cannot lie outside $\overline{os}$.

Given this, we must end up at a case in Figure 5. Clearly,

$$r_s = \delta(s, a) > \delta(e, a), \quad (12)$$

and since $S_{o,\gamma} \subseteq \mathcal{B}(o, 2\gamma)$ and $S_{o,\gamma} \subseteq \mathcal{B}(s, r_s)$, we have:

$$S_{o,\gamma} \subseteq \mathcal{B}(o, 2\gamma) \cap \mathcal{B}(s, r_s). \quad (13)$$

Now, consider $\mathcal{B}(o, 2\gamma) \cap \mathcal{B}(s, r_s)$. It is formed by two caps $C(o, e, h)$ from $\mathcal{B}(o, 2\gamma)$ and $C(s, e, b)$ from $\mathcal{B}(s, r_s)$. Consider the ball $\mathcal{B}(e, \delta(e, a))$ and suppose its boundary intersects with the line through b, f at i and j. The ball $\mathcal{B}(e, \delta(e, a))$ can be divided into two half-balls $C(e, e, i)$ and $C(e, e, j)$, which are special caps where the separating hyperplane passes its center. By Lemma 2, we know that $C(o, e, h) \subseteq C(e, e, j)$ and $C(s, e, b) \subseteq C(e, e, i)$. Therefore,

$$\mathcal{B}(o, 2\gamma) \cap \mathcal{B}(s, r_s) \subseteq \mathcal{B}(e, \delta(e, a)).$$

This means that there is a ball with radius $\delta(e, a) < r_s$ that contains $S_{o,\gamma}$, which contradicts with the fact that $\mathcal{B}(s, r_s)$ is the MEB of $S_{o,\gamma}$. □

Lemma 4 *For any point set $S_{o,\gamma}$ satisfying (10), let $\mathcal{B}(s, r_s) = \text{MEB}(S_{o,\gamma})$, and $\delta(o, s) = z$, then $z + r_s \le 2\sqrt{2}\gamma$.*

Proof.

$$\begin{aligned}z + r_s &\le z + \sqrt{(2\gamma)^2 - z^2} \quad \text{(by Lemma 3)} \\ &\le \sqrt{2(z^2 + (2\gamma)^2 - z^2)} = 2\sqrt{2}\gamma. \quad (14)\end{aligned}$$

Note that the second inequality is due to the fact that for any $x, y \ge 0$, $x^2 + y^2 \ge 2xy$. Thus, $(x + y)^2 \le 2(x^2 + y^2)$, and $x + y \le \sqrt{2(x^2 + y^2)}$. □

We are now ready to present the main theorem.

Theorem 5 Amax *gives a $(1 + 2\sqrt{2})$-approximate answer to the* max Fann *query in any dimensions, and it is tight.*

Proof. Let (p^*, Q_ϕ^*) be the optimal answer to the max Fann query with query group Q on P. Let r^* be the optimal aggregate distance, i.e.,

$$r^* = \max(p^*, Q_\phi^*) = \max_{q \in Q_\phi^*} \delta(p^*, q).$$

Let $\mathcal{B}(x, r_x) = \text{MEB}(Q_\phi^*)$. Since $\mathcal{B}(x, r_x)$ is the minimum enclosing ball of Q_ϕ^* and $Q_\phi^* \subseteq \mathcal{B}(p^*, r^*)$, we have

$$r_x \le r^*. \quad (15)$$

Consider any $q \in Q_\phi^*$. Clearly q is contained in $\mathcal{B}(x, r_x)$. This indicates that the maximum distance of q to any point in Q_ϕ^* is bounded by the diameter of $\mathcal{B}(x, r_x)$, i.e.,

$$\max(q, Q_\phi^*) \le 2r_x. \quad (16)$$

Note that Q_ϕ^q found by line 3 of the algorithm Amax consists of the ϕM nearest neighbors of q in Q (including q itself), and $Q_\phi^* \subseteq Q$. Thus,

$$\max(q, Q_\phi^q) \le \max(q, Q_\phi^*) \le 2r_x, \quad (17)$$

If we view q as o and r_x as γ, clearly $S_{o,\gamma} = Q_\phi^q$ satisfies (10). Line 3 in Amax also finds $b = \mathcal{B}(c, r_q) = \text{MEB}(Q_\phi^q)$, by Lemma 4, we have:

$$\delta(q, c) + r_q \le 2\sqrt{2}r_x. \quad (18)$$

Now, $p = nn(c, P)$, and Q_ϕ^p and r_p are found in line 5 of Amax. Recall that Q_ϕ^p is the ϕM nearest neighbors of p in Q and $r_p = \max(p, Q_\phi^p)$. We have:

$$\begin{aligned}r_p &= \max_{y \in Q_\phi^p} \delta(p, y) \\ &\le \max_{y \in Q_\phi^q} \delta(p, y) \qquad (Q_\phi^p \text{ is the } \phi M \text{ NNs of } p \text{ in } Q) \\ &\le \max_{y \in Q_\phi^q} (\delta(p, c) + \delta(c, y)) \\ &\le \max_{y \in Q_\phi^q} (\delta(p^*, c) + \delta(c, y)) \qquad (p = nn(c, P)) \\ &= \delta(p^*, c) + r_q \qquad (\mathcal{B}(c, r_q) = \text{MEB}(Q_\phi^q)) \\ &\le \delta(p^*, q) + \delta(q, c) + r_q \\ &\le r^* + 2\sqrt{2}r_x \qquad (\text{due to } q \in Q_\phi^* \text{ and (18)}) \\ &\le (1 + 2\sqrt{2})r^*. \qquad (\text{by (15)}) \quad (19)\end{aligned}$$

Finally, note that some q from Q_ϕ^* must have been iterated through by the Amax algorithm. Thus, the point p define above must have been checked as a candidate answer, which completes the proof. We show it is tight in Appendix A. □

Remark. When $\phi = 1$, the max Fann problem reduces to the max Ann problem, which is also referred to as the group enclosing query (Geq) in [15]. In this case, since all the $Q_\phi^{q_i}$'s are the same, which is the entire Q, the Amax algorithm degenerates to finding the nearest neighbor of the center of MEB(Q). This is exactly the algorithm proposed in [15] for the Geq problem. However, for this special case, a better approximation ratio of $\sqrt{2}$ can be proved [15].

Computational issues. Computing the minimum enclosing ball is well studied. For any point set S, MEB(S) can be computed efficiently in linear time in any constant dimensions [3]. In high dimensions, one can find a $(1+\epsilon)$-approximation of the minimum enclosing ball efficiently [13].

However, as we have pointed out in Section 5.1, exact NN search is expensive in high dimensions, and we can replace the exact NN search in line 5 of AMAX with a β-approximate NN search. When doing so, the approximation ratio of AMAX gets an extra β factor correspondingly.

Theorem 6 *Replacing the exact* nn *function in* AMAX *with a β-approximate NN search,* AMAX *gives a $((1+2\sqrt{2})\beta)$-approximate answer to the* max FANN *query.*

PROOF. Suppose the final answer returned now is $(p', Q_\phi^{p'})$ and the answer returned by AMAX with an exact nn method is (p, Q_ϕ^p). Following the derivation in (19), we have:

$$\begin{aligned}
r_{p'} &= \max_{y \in Q_\phi^{p'}} \delta(p', y) \\
&\le \max_{y \in Q_\phi^q} \delta(p', y) \\
&\le \max_{y \in Q_\phi^q} (\delta(p', c) + \delta(c, y)) \\
&\le \max_{y \in Q_\phi^q} (\beta\delta(p, c) + \delta(c, y)) \qquad (p' \text{ is } \beta\text{-approx. of } p) \\
&\le \beta \max_{y \in Q_\phi^q} (\delta(p, c) + \delta(c, y)) \\
&\le \beta(1+2\sqrt{2})r^*, \text{ (by the same derivation in (19))}
\end{aligned}$$

which shows that p' is a $((1+2\sqrt{2})\beta)$-approximate answer. □

As in Section 5.1.1, we can reduce the cost of AMAX by executing lines 3–7 of the algorithm on a random subset of points in Q, except that the analysis is simpler in this case.

Theorem 7 *For any $0 < \lambda < 10$, executing lines 3–7 of the* AMAX *algorithm only on a random subset of $f(\phi, \lambda)$ points of Q returns a $(1+2\sqrt{2})$-approximate answer to the* FANN *query with probability at least $1-\lambda$ in any dimensions, where*

$$f(\phi, \lambda) = \frac{\log \lambda}{\log(1-\phi)} = O(\log(1/\lambda)/\phi).$$

PROOF. We note that the proof of Theorem 5 only relies on at least one of the points in Q_ϕ^* being considered by the algorithm. If we run lines 3–7 on a random subset of $f(\phi, \lambda)$ points, the probability that none of ϕM points in Q_ϕ^* is considered is at most $(1-\phi)^{f(\phi,\lambda)}$. Setting $f(\phi, \lambda)$ as in the theorem makes this probability at most λ. □

Again, the theorem shows that the number of NN searches we need to issue is independent of $|Q|$ and dimensionality.

6. EXTENSIONS

6.1 The k-FANN problem

All of our algorithms can be extended to return the top-k FANN of a query group Q, that is, the k points p in P with the smallest aggregate distance $g(p, Q_\phi^p)$, where g is either sum or max. We briefly discuss these extensions next.

The R-tree method. The calculation of the pruning condition is intact, and the only difference is that a node should be compared against the kth best candidate answer found so far to decide whether it should be pruned or not.

The List algorithm. We calculate the threshold for the best possible aggregate similarity distance of any unseen object in the same way as before, but comparing this threshold value to the kth best candidate answer found so far to decide whether the algorithm should terminate or continue.

The ASUM algorithm. In line 3 of Algorithm 2, for any $q \in Q$, we find its top-k nearest neighbors from P. For each such point p, we carry out line 4 to find its Q_ϕ^p and r_p. Among the kM such candidate points generated after iterating through all points in Q, we return the k candidates with the k smallest r_p values. The approximation stays the same and the optimization in Section 5.1.1 can still be applied.

The ASUM1 algorithm. After finding the geometric median of the query group, we find its top-k nearest neighbors from P and return them as the answer. The approximation bound stays the same.

The AMAX algorithm. In line 5 of Algorithm 4, for any point q in Q, after finding its $\mathcal{B}(c, r) = \text{MEB}(Q_\phi^q)$, we find the k nearest neighbors of c in P. For each such point p, we find Q_ϕ^p and r_p. Among the kM such candidate points generated after iterating through all points in Q, we return the k candidates with the k smallest r_p values. The approximation stays the same and the optimization in Theorem 7 can still be applied.

6.2 General metric spaces

Our discussion so far focuses on the Euclidean space. Nevertheless, most of our algorithms generalize to other metric spaces without any changes, given the basic NN search algorithms in those spaces.

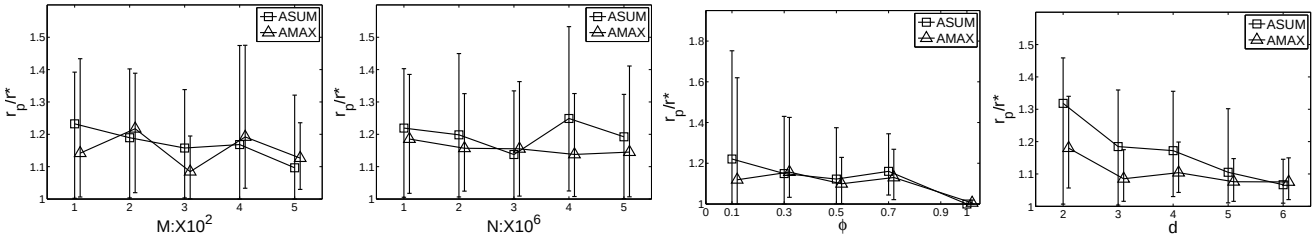
The *R-tree* method in principle works in any metric space, but we will need to replace the R-tree by a metric space indexing structure, such as the M-tree [7]. The *List* algorithm clearly works for any metric space, since it only relies on the basic NN search algorithm. For a similar reason, the ASUM algorithm works for any metric space as well and its approximation ratio remains the same in any metric space. The ASUM1 algorithm work in any metric space as well with the same approximation bound, as long as the geometric median is well defined and can be computed in that space. Finally, the AMAX algorithm works for the L_p^d space for any d and any $2 \le p < +\infty$, since it leverages on the minimum enclosing balls of point sets. How to design efficient approximation algorithms for the max FANN problem that work well in any metric space is an interesting open problem.

6.3 When Q is large

When Q becomes large and disk-resident, the exact methods work poorly and we should only consider the approximation methods. Our main approximation algorithms can be adapted to work with such large query groups easily.

First, consider the ASUM algorithm. The only new challenge is to support the efficient search of ϕM nearest neighbors in Q for a given point p. For this purpose, we can simply index Q using an index structure that supports k nearest neighbor search, such as R-tree in low dimensions, or LSB-tree in high dimensions.

Next, consider the AMAX algorithm. One of the challenges is the same as above, i.e., to support the efficient search



(a) vary M. (b) vary N. (c) vary ϕ. (d) vary d.

Figure 6: Approximation quality of ASUM and AMAX methods in low dimensions.

of the ϕM nearest neighbors in Q for a given point p (or q), which can be addressed similarly. The other challenge now is to efficiently find the minimum enclosing ball of ϕM points from Q. Since M is large, ϕM points could be a large set too. This can be addressed as follows. For any large point set S, note that its minimum enclosing ball is only determined by the convex hull of S. In other words, if we denote the set of vertices in the convex hull of S as C_S, then, $\text{MEB}(S) = \text{MEB}(C_S)$. There are I/O efficient algorithms for computing the convex hulls of disk-based data sets in multi-dimensions [5]. In most cases, $|C_Q| \ll |Q|$ which enables us find $\text{MEB}(Q)$ efficiently. However, in some rare cases $|C_Q|$ could be close to $|Q|$. When it does happen, we can obtain an approximate convex hull of Q efficiently using Dudley's approximation based on the coreset idea [24].

7. EXPERIMENTS

We implemented all algorithms in C++, and executed our experiments on a machine with an Intel Xeon 2GHz CPU and 4GB memory. For all index structures and data files, the page size is set to 4KB. For both of our approximation algorithms, namely ASUM and AMAX, we used the optimized versions with $1/\phi$ sampled query points from Q as guided by Theorems 3 and 7. The sample sizes indicated in Theorems 3 and 7 were necessary for the upper bound analysis, which are loose. In practice, we observed that simply taking a random sample of size $1/\phi$ is sufficient for both algorithms.

Datasets. In 2 dimensions, we obtained the Texas (TX) points of interest and road-network dataset from the Open Street map project, where each point is represented by its longitude and latitude. The TX dataset has 14 million points. We also generated synthetic datasets for 2 to 6 dimensions. To capture the clustering nature of real-world data, we generated random cluster (RC) datasets where clusters of different sizes and radius have been generated with random center locations in the space.

In high dimensions ($d \geq 10$), we used the *Color* dataset [6] consisting of $68,040$ points in 32 dimensions, the *MNIST* dataset [14] with $60,000$ points in 50 dimensions, and the *Cortina* dataset [20] with $1,088,864$ points in 74 dimensions.

Query groups. For the FANN problem, the cost of the query depends on several critical factors, including the location of the center (either the geometric median or the center of the minimum enclosing ball) of Q, the covering range of Q (i.e., the space enclosed by $\text{MEB}(Q)$), how points are distributed within the covering range, the size of Q and the value of ϕ. Thus, we generated queries as follows. For a certain query group size M, we set $\mathcal{A} = 5\%$ as the default volume of its covering range in terms of the percentage of the entire data space. Next a random location in the data space is selected as the center of the query group. Then M random points within a ball of volume $\mathcal{A}$, centered at this center location are generated. Two types of distributions were used to generate query points: uniform distribution (uu) and random cluster (rc). The relative performance of all algorithms were similar for these two distributions, so we only report the results using the rc query distribution. For each test, 40 random queries were generated. The efficiency (running time and the number of IOs) is very stable so we just report the average; the quality (approximation ratio) has some variation, so we report both the average as well as the 5%–95% interval.

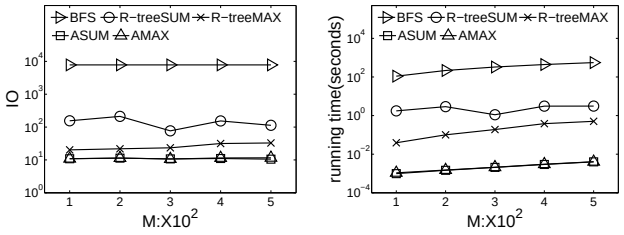
7.1 Low dimensions

Setup. For the low-dimensional experiments, we used the following default values: $M = 200$, $N = 2,000,000$, $\phi = 0.5$ and $d = 2$. We then varied each of them while keeping the others fixed at their default values. Specifically, we conducted 4 sets of experiments, where we respectively varied M from 100 to 500, N from 1 to 5 million, ϕ from 0.1 to 1 and d from 2 to 6. For the first three sets of experiments, we used the 2-dimensional real dataset TX where we picked N points randomly out of its 14 millions points. For the last set of experiments varying d, the synthetic RC datasets were used instead. In low dimensions, ASUM and AMAX utilize an R-tree which indexes P to answer NN queries.

Quality of approximation. The approximation ratios of ASUM and AMAX for the 4 sets of experiments are shown in Figure 6. Clearly, in all these cases, both methods achieved excellent approximation quality for the sum and max FANN problems, respectively. The average approximation ratio is between 1.1 to 1.3 in all these experiments. More importantly, both algorithms behave quite stably, with the 95% percentile at 1.5 for most scenarios. Figures 6(a) and 6(b) show that their approximation qualities are not affected by the size of the query group or the dataset. Figure 6(c) indicates that a larger ϕ value leads to better approximation quality (we used the ASUM1 algorithm for the sum FANN when $\phi = 1$). This is due to the fact that when ϕM is small, the probability that our $1/\phi$ sampled points do not cover at least one point from Q^*_ϕ is higher. Finally, Figure 6(d) shows that their approximation qualities actually improve slightly as d increases. This supports our theoretical analysis that the approximation ratios of ASUM and AMAX do not depend on dimensionality. The slight improvement of the approximation ratio may be attributed to the fact that the optimal distance, r^*, increases faster as d increases than the distance returned by the algorithm.

Efficiency. We next focus on the efficiency of different algorithms. Since some algorithms incur both considerable disk IOs and CPU costs, we reported both the number of IOs and

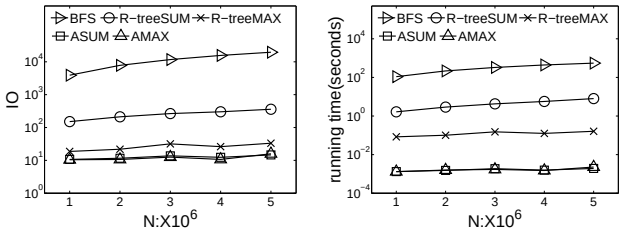
the end-to-end running time. For the exact methods in low dimensions, we observe that the *List* algorithm is strictly worse than the *R-tree* method. Hence, we only report the *R-tree* method as the representative of our exact methods.



(a) IO. (b) running time.

Figure 7: Low dimensions: vary M.

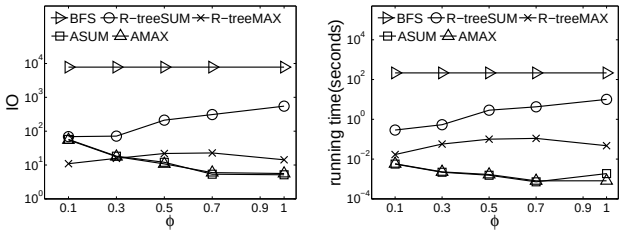
Figure 7 shows the results when we vary M. Clearly, the *R-tree* method outperforms the *BFS* method by 2-3 orders of magnitude in terms of IOs and running time, for the sum and max FANN problems respectively. Our approximation algorithms, ASUM and AMAX, are even more efficient. They further outperform the *R-tree* method by 0.5 order of magnitude in terms IOs and 2–3 orders of magnitude in terms of the running time. This is because for each MBR node, the R-tree method has to compute its mindist to every query point from Q. Both ASUM and AMAX methods are able to answer a query Q with 300 query points over 2 million points in P in just about 1 millisecond and 10 IOs! This well justifies the use of approximation instead of exactly solving the problem. The performance curves of ASUM and AMAX are almost identical. This is not surprising, as both of them issue $1/\phi$ NN queries, which is the main cost of these two algorithms (computing MEBs for a group of points in AMAX in low dimensions is very cheap).



(a) IO. (b) running time.

Figure 8: Low dimensions: vary N.

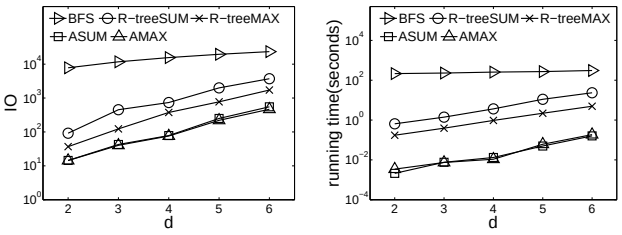
We next study the effect of dataset size, and Figure 8 shows a similar trend, where ASUM and AMAX have outperformed the *R-tree* and *BFS* methods by orders of magnitude. The results show that ASUM and AMAX have excellent scalability w.r.t. the dataset size. For example, they still only require around 10 IOs and 1 millisecond per query for 200 query points on 5 million P points.



(a) IO. (b) running time.

Figure 9: Low dimensions: vary ϕ.

Our next experiment investigates the effect of ϕ. Since ASUM and AMAX both issue $1/\phi$ NN queries on an R-tree indexing the P points, when ϕ is small, their query costs would be higher. This trend was observed in Figures 9(a) and 9(b). In particular, Figure 9(a) shows that ASUM has similar IO cost as the *R-tree* method when $\phi = 0.1$, but has much lower IO costs for larger ϕ. AMAX actually has a higher IO cost than the *R-tree* method when $\phi = 0.1$, but then achieved lower IO costs for all other ϕ values. In terms of the running time, ASUM and AMAX are both still much lower than the *R-tree* method for all ϕ values as shown in Figure 9(b).



(a) IO. (b) running time.

Figure 10: Low dimensions: vary d.

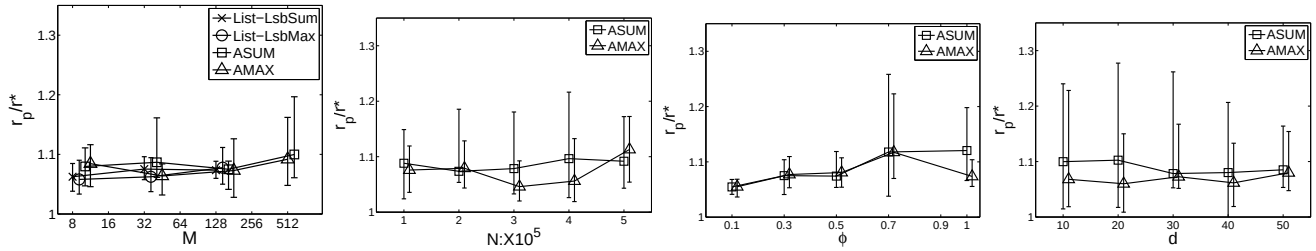
Next experiment studies the effect of dimensionality, where we tested all algorithms on the *RC* datasets from 2 to 6 dimensions as shown in Figure 10. Not surprisingly, the costs for all algorithms increase as d gets higher, as all of them rely on the underlying R-tree (except *BFS*), which gets less effective in higher dimensions. Nevertheless, ASUM and AMAX are clearly the winner in all dimensions.

7.2 High dimensions

Setup. R-tree does not scale to high dimensions ($d > 10$), so we used *BFS* and *List* as the exact methods. For *List*, we also changed the underlying NN index from the R-tree to *iDistance* [12], the state of the art for exact NN search in high dimensions. For ASUM and AMAX, we changed the underlying NN index to the LSB-tree [22], the state of the art for answering approximate NN queries in high dimensions. We also tested another natural approximate solution by plugging the LSB-tree into the *List* method.

The main dataset we used is the *Cortina* dataset, from which we extracted smaller ones for various experiments. To get a dataset of N points in d dimensions, we randomly sample N points from *Cortina* and take the first d coordinates for every such point. The default values for all parameters are $M = 200$, $N = 200,000$, $\phi = 0.5$ and $d = 30$. Similar to the low-dimensional experiments, we performed 4 sets of experiments, varying one of these 4 parameters respectively while keeping the rest fixed at their default values. Specifically, we varied M from 8 to 512, N from $100,000$ to $500,000$, ϕ from 0.1 to 1 and d from 10 to 50. Finally, we also tested on the three datasets, *MNIST*, *Color* and *Cortina* in their entirety in their original dimensions.

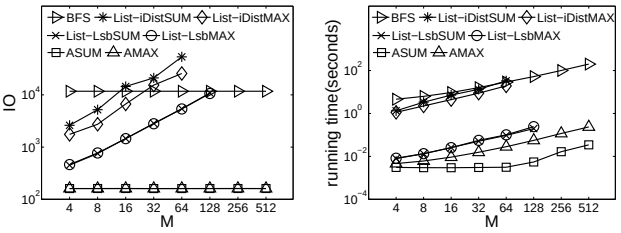
Quality of approximation. We first study the approximation quality of ASUM and AMAX, as well as *List* with the LSB-tree. Results from Figure 11 show that they retain their high quality approximations in high dimensions. The average approximation ratio for all of them is around 1.1, and its 95% percentile is below 1.3 in all cases. This backs up our analysis that the approximation quality is not affected by dimensionality, and at the same time demonstrates that the approximation ratio could be much better in practice than the worst-case bounds of 3 and $1 + 2\sqrt{2}$. Note that for



(a) vary M. (b) vary N. (c) vary ϕ. (d) vary d.

Figure 11: Approximation quality of ASUM and AMAX in high dimensions.

List, we only obtained its approximation ratios for the small M's, as it is too slow for $M = 200$ (to be seen later).

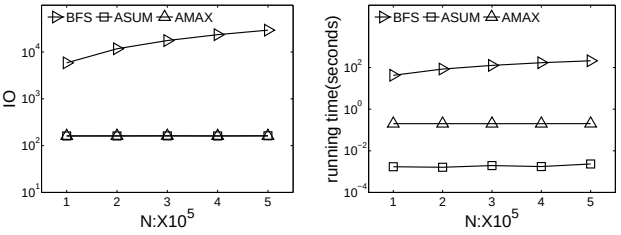


(a) IO. (b) running time.

Figure 12: High dimensions: vary M.

Efficiency. Figure 12 shows the efficiency of all methods when varying M. It is clear that in all cases, ASUM and AMAX maintain their superiority over other methods by 2–4 orders of magnitude in terms of both IOs and running time. In 30 dimensions, for 256 points in a query group over 200,000 data points, ASUM takes only 0.01 second per query; AMAX is more expensive, due to the higher computation cost of the MEBs in high dimensions, but it still only takes about 0.1–0.2 second per query. Their performance is also very stable even when M increases. In particular, the IO cost remains almost constant. This is because they incur IO costs only when performing NN queries on P, while they always issue $1/\phi$ NN queries irrespective of M.

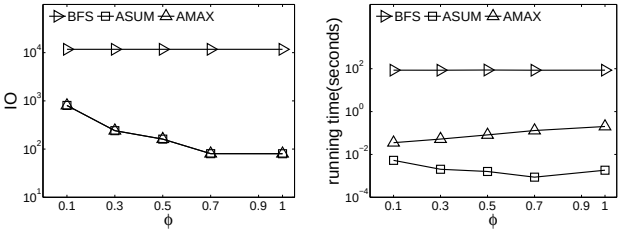
When M is small, the two *List* methods outperform *BFS*. However, as M increases, they start to deteriorate rapidly and eventually become as bad as or even worse than *BFS*. This is because *List* has to do at least M NN searches at the very beginning, followed by potentially more kNN searches. These NN and kNN searches become very expensive in high dimensions. Using the LSB-tree instead of iDistance does help a lot in terms of efficiency, but it no longer returns exact answers, and its approximation quality is not significantly better than ASUM and AMAX. Since the two *List* methods are highly expensive for our default query group size $M = 200$, we excluded them in the rest of the experiments.



(a) IO. (b) running time.

Figure 13: High dimensions: vary N.

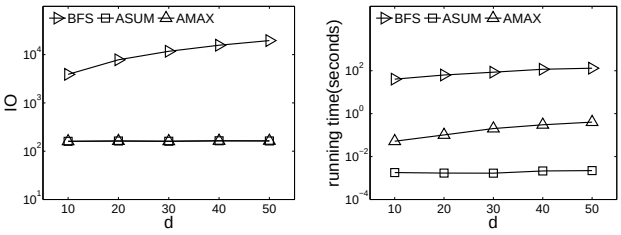
We next study the efficiency of the methods by varying the size of the dataset. The results are shown in Figure 13. We observe that ASUM and AMAX have excellent scalability w.r.t. the size of the dataset, with only slight increases in their IO and running times as N gets larger.



(a) IO. (b) running time.

Figure 14: High dimensions: vary ϕ.

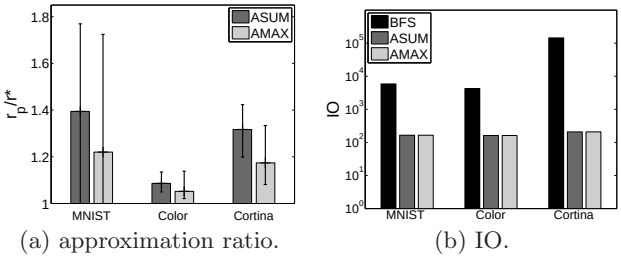
Figure 14 shows the experimental results when we vary ϕ. Similar to the results in low dimensions, smaller ϕ values lead to higher costs for both ASUM and AMAX, due to the $1/\phi$ sample size. Nevertheless, they are still much more efficient than *BFS*.



(a) IO. (b) running time.

Figure 15: High dimensions: vary d.

We also tested their performances in different dimensions as shown in Figure 15. Again, ASUM and AMAX scale very well with dimensionality, which is primarily attributed to the LSB-tree being able to handle high-dimensional data very well, and the dominating cost is that of the $1/\phi$ NN queries. While with almost identical IO cost, the running time of AMAX is longer due to the higher cost of computing MEBs in high dimensions.



(a) approximation ratio. (b) IO.

Figure 16: All datasets in high dimensions.

Lastly, we tested *BFS*, ASUM and AMAX on the three real datasets in high dimensions using all available points in their original dimensions, respectively. The results are shown in

Figure 16. Similar to the previous experiments, ASUM and AMAX have an 2–3 orders of magnitude of improvement in terms of efficiency, while returning query answers of high quality. We note that the approximation ratios are higher on the MNIST dataset than the other two, due to some special properties of the dataset, but they are still much lower than the guaranteed approximation ratios.

8. CONCLUSION

Flexible aggregate similarity search (FANN) extends the aggregate similarity search (ANN) with added flexibility that are useful in many applications. In this paper, we presented a comprehensive study on the FANN problem, by designing exact and approximation methods that work well in low to high dimensions. Our approximation methods are especially appealing, which come with constant approximation ratios in theory and perform extremely well in practice, in terms of both approximation quality and query efficiency, as evident from our extensive experimental study. Future work includes the design of an efficient approximation method for max FANN that works well in any metric space.

9. ACKNOWLEDGMENT

Feifei Li and Bin Yao were supported by NSF Grant IIS-0916488. The work was conducted when Yang Li was an intern student at HP Labs China. Part of the work was done when Feifei Li and Ke Yi were visiting HP Labs China.

10. REFERENCES

[1] 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.

[2] S. Berchtold, C. Böhm, D. A. Keim, and H. P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *PODS*, 1997.

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

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

[5] C. Böhm and H.-P. Kriegel. Determining the convex hull in large multidimensional databases. In *International Conference on Data Warehousing and Knowledge Discovery*, 2001.

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

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.

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

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

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

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

[12] 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.

[13] 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.

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

[15] F. Li, B. Yao, and P. Kumar. Group enclosing queries. *IEEE TKDE*, To Appear, 2010.

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

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

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

[19] H. L. Razente, M. C. N. Barioni, A. J. M. Traina, C. Faloutsos, and C. Traina, Jr. A novel optimization approach to efficiently process aggregate similarity queries in metric access methods. In *CIKM*, 2008.

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

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

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

[23] M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate nearest neighbor queries in road networks. *IEEE TKDE*, 17(6):820–833, 2005.

[24] 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.

APPENDIX

A. TIGHTNESS OF AMAX

Here we show that the $(1 + 2\sqrt{2})$ approximation ratio of AMAX is tight, by giving a concrete example. Consider the case in Figure 17 where ϵ is an arbitrarily small positive.

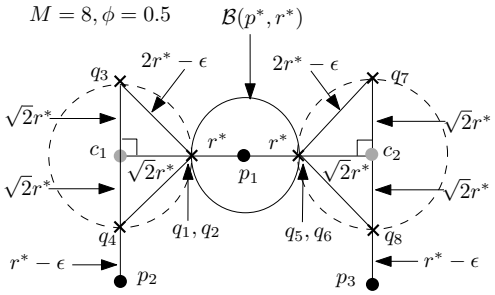


Figure 17: AMAX's approximation bound is tight.

In this case, $M = 8$, $\phi = 0.5$, hence $\phi M = 4$ and $\phi M - 1 = 3$. Consider q_1, its 3-nearest neighbors in Q are $\{q_2, q_3, q_4\}$, hence $Q_\phi^{q_1} = \{q_1, q_2, q_3, q_4\}$. Note that $\text{MEB}(\{q_1, q_2, q_3, q_4\}) = \mathcal{B}(c_1, \sqrt{2}r^*)$, and $\text{nn}(c_1, P) = p_2$. Now, p_2's 4-nearest neighbors in Q are $\{q_4, q_3, q_2, q_1\}$. Hence, $Q_\phi^{p_2} = \{q_4, q_3, q_2, q_1\}$, $r_{p_2} = \max(p_2, \{q_4, q_3, q_2, q_1\}) = (1 + 2\sqrt{2})r^* - \epsilon$.

It's easy to verify that the results from q_2, q_3 and q_4 are the same as q_1, since $Q_\phi^{q_2}$, $Q_\phi^{q_3}$ and $Q_\phi^{q_4}$ are the same as $Q_\phi^{q_1} = \{q_1, q_2, q_3, q_4\}$. Furthermore, q_5, q_6, q_7 and q_8 are symmetric to q_1, q_2, q_3 and q_4, and p_3 is symmetric to p_2. Thus they yield $(p_3, Q_\phi^{p_3})$ as the answer, and $r_{p_3} = \max(p_3, \{q_5, q_6, q_7, q_8\}) = (1 + 2\sqrt{2})r^* - \epsilon$.

As a result, AMAX will return either $(p_2, Q_\phi^{p_2})$ or $(p_3, Q_\phi^{p_3})$ as the answer, with $r_2 = r_3 = (1 + 2\sqrt{2})r^* - \epsilon$. But in this case $p^* = p_1$, $Q_\phi^* = \{q_1, q_2, q_3, q_4\}$, and $\max(p^*, Q_\phi^*) = r^*$.