Noname manuscript No.
(will be inserted by the editor)

Exact and Approximate Flexible Aggregate Similarity Search

Feifei Li[1], Ke Yi[2], Yufei Tao[3], Bin Yao[4*], Yang Li[4], Dong Xie[4], Min Wang[5]
[1] ***University of Utah, USA***
[2] ***Hong Kong University of Science and Technology, Hong Kong, China***
[3] ***Chinese University of Hong Kong, Hong Kong, China***
[4] ***Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai, China***
[5] ***Visa Research, Visa Inc., USA***
[1] ***lifeifei@cs.utah.edu,*** [2] ***yike@cse.ust.hk,***
[3] ***taoyf@cse.cuhk.edu.hk,*** [5] ***minwang@visa.com***
[4] ***yaobin@cs.sjtu.edu.cn, rainfallen@sjtu.edu.cn, skyprophet@sjtu.edu.cn,***
[*] ***corresponding author.***

the date of receipt and acceptance should be inserted later

Abstract Aggregate similarity search, also known as aggregate nearest neighbor (Ann) query, finds many useful applications in spatial and multimedia databases. Given a group Q of M query objects, it retrieves from a database the objects most similar to Q, where the similarity is an aggregation (e.g., sum, max) of the distances between each 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 search*, and accordingly refer to a query as a *flexible aggregate nearest neighbor (*Fann*) query*. 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.

1 Introduction

Aggregate similarity search extends the classical similarity search problem with a group Q of query objects, and the goal is to retrieve from the underlying database P the objects most similar to Q, where similarity is defined by applying an aggregate function (usually sum or max) over the set of distances between each retrieved object and every query object [15, 16, 18–20, 25]. 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, 18–20, 25]. Due to its importance, this problem has already been studied in the Euclidean space [15, 18, 19], the road-network space [25], and the general metric space [20]. 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 can be too restrictive in practice.

We observe that in many applications, it is often good enough, and in some cases even desired, to find objects similar to a fraction of the objects in a query group Q. For example, suppose that 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

Address(es) of author(s) should be given

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 object from the database that is the most similar to any $\phi|Q|$ objects from Q. We call it *flexible aggregate similarity search*, and accordingly, refer to a query as a *flexible aggregate nearest neighbor* (FANN) query. Clearly, the classic aggregate nearest neighbor query 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. Given a set Q of objects (e.g. images), the ANN will find a data object that is similar to all query objects, which can be too restrictive in many cases. Instead, 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, having more flexibility) when formulating his/her query group Q. When returning the top-k objects (called the k-FANN problem in Section 7.1), the diversity of the query answers also increases: the k objects might be similar to k different subsets of $\phi|Q|$ query objects each.

Henceforth, by default, we assume that each data/query object is a multi-dimensional point, and that the distance of two points is measured by L_2 norm (i.e., Euclidean distance). Intuitively, the added flexibility of FANN queries creates significant challenges in designing efficient algorithms because a FANN query implicitly incorporates $\binom{|Q|}{\phi|Q|}$ ANN queries, as each subset of $\phi|Q|$ objects in Q can be the best subset that a data object is similar to. In Section 4 we present two algorithms based on standard techniques for answering FANN queries exactly. As shown in the experiments, these algorithms are rather expensive, especially in high dimensions, where they can be as bad as linear scans.

Therefore, we investigate approximation algorithms that greatly improve the efficiency while returning near-optimal answers. These algorithms, presented in Section 5 and 6 for the sum and max versions of the problem, respectively, have the following appealing features:

- *Guaranteed approximation ratios:* Our first algorithms return answers with guaranteed approximation ratios of 3 and $1 + 2\sqrt{2}$, for the sum and max versions of the problem, respectively, regardless of the dimensionality. 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, 18–20].

  As a second step, we show that in 2d space (which enjoys an especially important standing of all dimensionalities), one can extend our first algorithms with an elegant new idea to improve the approximation ratio to 2 (for both sum and max). Finally, in low dimensional spaces, we explain how to guarantee an approximation ratio of $(1 + \varepsilon)$, where ε can be an arbitrarily small constant, with further extensions to the algorithms.
- *Excellent query answer quality in practice:* The approximation ratios hold even for the worst data. In practice, extensive experiments 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 a similar scale takes 0.01 to 0.1 second. Detailed experimental results are provided in Section 8.
- *Simplicity:* Our algorithms are actually very simple. Except the ones with approximation ratio $(1 + \epsilon)$ which bear significant theoretical interest, they reduce the problem to a few instances of standard nearest neighbor (NN) search, which is a well studied problem, and efficient solutions are known in both low and high dimensions. The $(1 + \epsilon)$ approximate algorithms, on the other hand, demand only the extra functionality of range reporting, which is also well studied in low dimensional space (recall that the $(1 + \epsilon)$-approximate algorithms are only designed for low dimensionality).

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 present approximation algorithms and analyze their theoretical guarantees in Sections 5 and 6 for the sum and max aggregate functions, respectively. We discuss several extensional issues in Section 7, present the experimental results in Section 8, and finally conclude in Section 9.

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 $d(p, q)$ defined for any two points. By default, $d(p, q)$ represents the Euclidean distance of p and q; we will discuss how our methods can be adapted

to support other distance metrics in Section 7. 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:

$$g(p, S) = g(d(p, q_1), \ldots, d(p, q_{|S|})),$$

where $q_i \in S$ 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, d, 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. 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 point 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

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

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 d-dimensional Euclidean space as the default metric space, and briefly discuss general metric spaces in Section 7. We summarize the main notations in Figure 1.

Symbol	Description
$\mathcal{B}(c, r)$	the ball centered at c with radius r
$d(p, q)$	distance between p and q
g	sum or max
$g(o, S)$	$g(d(o, s_1), \ldots, d(o, s_{\|S\|}))$ 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 aggregate similarity distance $g(p^*, Q_\phi^*)$

Fig. 1 List of notations.

3 Related Work

Research on aggregate similarity search was initialized by Papadias et al. [18] 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 [19], 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. [20] used the same idea for other metric spaces with distance-based indexing structures, such as the M-tree [6]. 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 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. [19] also proposed a few heuristics 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, namely, 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 [22] 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 [24], 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.

New contributions. The problem of flexible aggregate similarity search was formalized in [17], and a number of state-of-the-art results were also proposed in the same work, for both the sum and max aggregate functions. Specifically, they designed R-tree based and TA-based (the threshold algorithm from [8]) exact algorithms for answering Fann queries in low and high dimensions respectively, which we will review in details in Section 4. They also designed an $(1+2\sqrt{2})$-approximation algorithm, and an 3-approximation algorithm for max Fann and sum Fann queries respectively that work well in all dimensions. We denote these approximation algorithms as Asum and Amax respectively (including the efficiency-improved versions based on a subset of random samples from Q), and review them in details in Sections 5.1 and 6.

This work makes significant new contributions compared to the earlier work [17]. The first new contribution is the design of an unified algorithmic framework that gives an 2-approximation for both sum and max Fann queries in 2d, which enjoys particular importance since most practical applications in spatial databases are indeed in a 2-dimension space. Not only our new design enjoys better approximation bound in theory (and in practice), but also they are almost equally efficient compared to Asum and Amax. We denote the new approximation methods as Asum2 and Amax2 and discuss them in Sections 5.2 and 6.2 respectively.

The next contribution of this work is the design of an $(1+\varepsilon)$-approximation algorithm for both sum and max Fann queries, which is less practical but bears high theoretical interest. This discussion appears in Sections 5.3 and 6.3 for sum and max case respectively. Note that having an $(1+\varepsilon)$-approximation algorithm is extremely useful in theory since this shows us how to nicely adjust the tradeoff between efficiency and accuracy, and allows us to reduce the approximation error to some arbitrarily small values if we would like to. This is the same motivation behind the design of the $(1+\varepsilon)$-approximation algorithm for the classic nearest neighbor queries [1].

Lastly, our work presents extensive new experimental results in Section 8 to empirically verify the effectiveness and efficiency of all proposed methods using large real data sets.

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 improved exact methods. The first method is based on the R-tree and can be seen as a generalization of the techniques in [19]. As it is based on the R-tree, it works only in low dimensions. The second method is based on the TA algorithm [7,8] 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 $\text{mindist}(q, b)$ [11,22]. 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

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

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

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

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 [19] 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 $(p_{i,j}, d_{i,j})$ where $d_{i,j} = d(p_{i,j}, q_i)$, $p_{i,j} \in P$ 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 $d(p, q_i)$, 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 [8] 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 = d_{i,ℓ_i} for i = 1, ..., M;
2  set τ = g(smallest φM values from τ_i s);
3  set p_o = null and d_o = +∞;
4  while true do
5      let η = argmin_{i∈[1,M]} d_{i,ℓ_i};
6      set p' = p_{η,ℓ_η} and compute d' = g(p', Q_φ^{p'});
7      if d' < d_o then
8          set p_o = p' and d_o = d';
9      if ℓ_η < N then
10         set ℓ_η = ℓ_η + 1 and τ_η = d_{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 d_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, $(p_{i,j}, d_{i,j})$, 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 [24] 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 Approximation Algorithms for sum FANN

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 [24] 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, 24]. This motivates us to design approximation algorithms for the FANN problem with quality and efficiency guarantees. In this section, we will do so for sum FANN, while the next section is devoted to max FANN.

5.1 A 3-approximate algorithm

Our first approximate method, denoted as ASUM, is given in Algorithm 2, which is 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 now prove a quality guarantee for the above algorithm:

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

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

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

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:

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

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

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

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

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

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

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 $d(p'', q^*) \leq \beta \cdot d(p', q^*)$. Going through the proof of Theorem 1, inequality (4) becomes

$$d(p'', q^*) \leq \beta \cdot d(p', q^*) \leq \beta \cdot d(p^*, q^*), \tag{8}$$

and the derivation in (6) becomes

$$\sum_{x\in Q_\phi^*} d(p'',x) \le \phi M\cdot(d(p'',q^*)+d(q^*,p^*))+r^*$$
$$\le \phi M\cdot(\beta+1)d(q^*,p^*)+r^* \quad \text{(by (8))}$$
$$\le (\beta+1)r^*+r^* = (\beta+2)r^*. \quad \text{(by (7))}$$

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 sum* 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., $q'\in Q_{\epsilon\phi/3}^{p^*}$, the proof can still go through except inequality (7), hence (6).

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

$$(\phi M-\frac{\epsilon\phi M}{3})d(q',p^*)\le \sum_{x\in Q_\phi^{p^*}-Q_{\epsilon\phi M/3}^{p^*}} d(x,p^*)$$
$$\le \sum_{x\in Q_\phi^{p^*}} d(x,p^*)=\sum_{x\in Q_\phi^*} d(x,p^*).$$

Thus, (7) becomes

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

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

$$\frac{2}{1-\epsilon/3}r^*+r^*\le 2(1+\epsilon/2)r^*+r^*=(3+\epsilon)r^*. \quad (10)$$

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 [18, 19]. A simple heuristic approximate algorithm was proposed in [19], denoted as Asum1 (Algorithm 3), which simply returns the nearest neighbor of the geometric centroid of Q. However, no approximation ratio was proved in [19]. 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 centroid of Q;
2 **return** $p_m=\text{nn}(q_m,P)$;

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. Next, we focus on the approximation bound:

$$r_{p_m}=\sum_{q\in Q} d(p_m,q)\le\sum_{q\in Q}(d(p,q_m)+d(q_m,q))$$
$$\le\sum_{q\in Q}(d(p^*,q_m)+d(q_m,q)) \quad (\text{since } p_m=\text{nn}(q_m,P))$$
$$\le\sum_{q\in Q}(d(p^*,q)+d(q,q_m)+d(q_m,q))$$
$$=\sum_{q\in Q} d(p^*,q)+2\sum_{q\in Q} d(q_m,q)\le 3r^*.$$

The last '$\le$' holds because the geometric centroid q_m of Q has the property that it is the point q (among all

the points in the data space) minimizing the sum of the Euclidean distances from q to the points of Q. Hence, $\sum_{q\in Q} d(q_m, q) \leq \sum_{q\in Q} d(p^*, q) = r^*$.

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 centroid 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.□

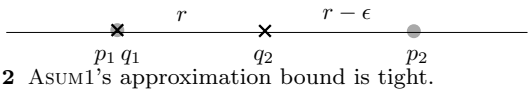


Fig. 2 ASUM1's approximation bound is tight.

5.2 A two-dimensional 2-approximate algorithm

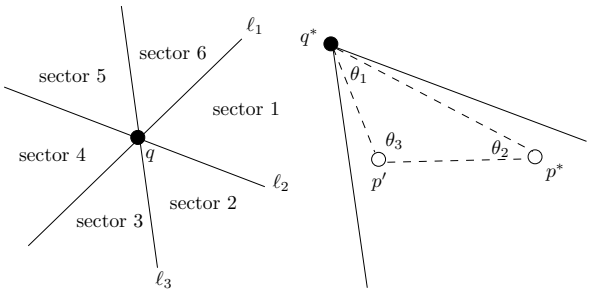
We now show that the approximation ratio can be improved to 2 with a slight extension to the ASUM algorithm. We will see that this extension works best in 2d space, and thus, lends itself nicely to applications dealing with entities in a longitude-latitude world.

Let us first introduce a useful concept called *sector nearest neighbors* due to [23]. Given a point q in two-dimensional space, let ℓ_1, ℓ_2, and ℓ_3 be lines such that (i) all of them cross q, and (ii) each pair of lines makes an angle of 60 degrees. See Figure 3a for an illustration. Note that the group of ℓ_1, ℓ_2, ℓ_3 satisfying the two conditions is not unique; and any group suffices for our purposes. These lines cut the data space into 6 sectors around q, each of which as shown in Figure 3a is an infinite cone-like area. The sector nearest neighbors of q with respect to a set P of points are $p_1, ..., p_6$ such that p_i $(1 \leq i \leq 6)$ has the smallest distance to q among all the points of P in sector i. Note that p_i does not exist if P has no point in sector i (hence, q may have less than 6 sector nearest neighbors).

Algorithm 4: ASUM2 (P, Q, ϕ, sum)

```
1 set C = ∅;
2 for i = 1, ..., M do
3     add to C the sector nearest neighbors of q_i with
      respect to P;
4 return (p, Q_φ^p) where p is the point in C with the
  smallest r_p;
```

Algorithm 4 presents our new algorithm named ASUM2. We now prove:



(a) Sectors around a point (b) Illustration of a property in the proof of Theorem 5

Fig. 3 Ideas behind ASUM2

Theorem 5 ASUM2 *returns a 2-approximate answer to any sum* FANN *query.*

Proof Let (p^*, Q^*_ϕ) be an optimal answer to the query. Let $q^* = \text{nn}(p^*, Q^*_\phi)$. Let p' be the point closest to q^*, among all the points of P in the same sector of q^* as p^*. If $p' = p^*$, then Algorithm ASUM2 returns p^*; and hence, we are done. Next, we consider $p' \neq p^*$; see Figure 3b for an illustration of the relative positions of q^*, p^*, and p' in this scenario.

Let us focus on the triangle q^*p^*p', and its angles θ_1, θ_2, and θ_3 as defined in Figure 3b. Since $d(p', q^*) \leq d(p^*, q^*)$ by definition of p', we know that $\theta_3 \geq \theta_2$. On the other hand, θ_1 cannot exceed 60 degrees (which is the angle between the two solid lines, by definition of a sector). These facts, together with the property that $\theta_1 + \theta_2 + \theta_3 = 180$ (degrees), imply $\theta_3 \geq 60$. Hence, $\theta_3 \geq \theta_1$, indicating that $d(p', p^*) \leq d(p^*, q^*)$.

We now complete the proof with the following argument:

$$
\begin{aligned}
r_{p'} &= \sum_{x\in Q^{p'}_\phi} d(p', x) \;\leq\; \sum_{x\in Q^*_\phi} d(p', x) \\
&\leq \sum_{x\in Q^*_\phi} (d(p', p^*) + d(p^*, x)) \\
&= (\phi M) \cdot d(p', p^*) + r_{p^*} \\
&\leq (\phi M) \cdot d(p^*, q^*) + r_{p^*} \\
&\leq r_{p^*} + r_{p^*} \qquad \text{(by (7))} \\
&= 2r_{p^*}.\square
\end{aligned}
$$

Finding the sector nearest neighbors of a point q can be done by running 6 *constrained nearest neighbor queries* [9] concurrently, which can be efficiently supported using an R-tree on P. Note that the concurrent execution of the 6 queries avoids accessing the same node of the R-tree twice. Compared to the ASUM algorithm, the overhead of ASUM2 comes from: (i) for each query point $q \in Q$, ASUM2 pays higher cost because the 6 constrained nearest neighbor queries together are

more expensive than a single nearest neighbor query; (ii) ASUM2 needs to calculate the aggregate distance of up to $6M$ points, as opposed only M in ASUM. In return, however, ASUM2 guarantees that the quality of its answer is *never worse* than that of ASUM. This is because the set C of candidate points considered by ASUM2 includes *all* the candidate points considered by ASUM (in Algorithm 2, the candidates are fetched at Line 3)—noticing that the nearest neighbor of a point q must be a nearest neighbor in a sector of q.

As mentioned before, ASUM2 is best suited for 2d space, but what happens in higher dimensional space? The underlying idea of ASUM2 can still be generalized, except that a sector of q is no longer bounded by 2 lines, but instead, is bounded by d lines $\ell_1, ..., \ell_d$ (where d is the dimensionality) such that *each* pair of these lines forms an angle of 60 degrees. It can be shown that we can cover the whole d-dimensional space by using c^d such sectors, for some constant c. To solve a sum FANN query, all that remains is to find, for each $q_i \in Q$, the constrained nearest neighbor of q_i in each sector of q_i. An argument similar to the proof of Theorem 4 shows that at least one of these $c^d M$ constrained nearest neighbors gives a 2-approximate answer. Unfortunately, this idea is mainly of theoretical interests because it is non-trivial to solve a constrained nearest neighbor of this sort even in 3d space.

5.2.1 Improve the efficiency of ASUM2

Similar to the sampling method we have designed to reduce the cost of ASUM in Section 5.1.1, a sampling based approach can also be used here to dramatically improve the efficiency of the ASUM2 method, without sacrificing much of its approximation accuracy. Specifically, we have:

Theorem 6 *For any $0 < \epsilon, \lambda < 1$, executing line 3 of the* ASUM2 *algorithm only on a random subset of $f(\phi, \epsilon, \lambda)$ points of Q returns a $(2+\epsilon)$-approximate answer to the sum* FANN *query in any dimensions with probability at least $1 - \lambda$, where $f(\phi, \epsilon, \lambda)$ is set in the same way as that in* (9)*, i.e.,*

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

Proof Following the proof of Theorem 5, we note that the approximation ratio is guaranteed as long as q^* is one of the points in Q that have gone through line 3 of the algorithm. Hence, we can leverage the same intuition as that followed by Theorem 3 to optimize this with a randomly chosen subset of Q.

Now since we execute line 3 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., $q' \in Q^{p^*}_{\epsilon\phi/3}$, the proof can still go through except that now we have q' instead of q^* in Figure 3(b). But still, p' is the nearest neighbor of q', among all points of P in the same sector of q' as p^*. If $p' = p^*$, then ASUM2 returns p^*; and hence we are done. When $p' \neq p^*$, we still have $d(p', q') \leq d(p^*, q')$ by definition of p', which implies that it is still the case that $\theta_3 \geq \theta_2$. Using the same argument as before, we can show that $d(p', p^*) \leq d(p^*, q')$.

On the other hand, using the same argument from the proof of Theorem 3, in this case given $q' \in Q^{p^*}_{\epsilon\phi/3}$, we still have equation (10), i.e.,

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

Combining the above observations, we have:

$$\begin{aligned} r_{p'} = \sum_{x \in Q^{p'}_\phi} d(p', x) \ &\leq \sum_{x \in Q^*_\phi} d(p', x) \\ &\leq \sum_{x \in Q^*_\phi} (d(p', p^*) + d(p^*, x)) \\ &= (\phi M) \cdot d(p', p^*) + r_{p^*} \\ &\leq (\phi M) \cdot d(p^*, q') + r_{p^*} \\ &\leq \frac{1}{1 - \epsilon/3} r^* + r_{p^*} \\ &\leq (2 + \epsilon) r^*. \end{aligned}$$

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.□

5.3 A $(1+\epsilon)$-approximate algorithm in low dimensional space

The approximation algorithms we have proposed so far are highly efficient—they perform $O(1)$ instances of nearest neighbor search for each query point in Q. They, however, are not suitable if one demands an approximate answer whose quality is extremely close to the optimal. In this section, we explain how to bring down the approximation ratio to $1+\epsilon$, where ϵ can be any arbitrarily small constant. Our algorithm is designed for low dimensional space where the dimensionality d is a small constant.

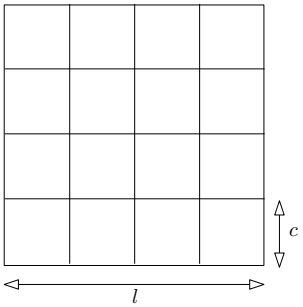


Fig. 4 An (l, c)-regular grid in 2d space

Let us review an operation called *find-any*. Given an axis-parallel rectangle ρ, such an operation reports whether the dataset P has any point in ρ; and if the answer is yes, the operation also returns such a point (if the answer is no, then no more output is required of the operation). We will need a slightly more sophisticated version of *find-any*, which we call *grid-find-any*(G, P). Here, as before, P is a set of points. G is an (l, c)-*regular grid* defined as follows: (i) the boundary of G is an axis-parallel (hyper) square in the underlying d-dimensional space with length l on each dimension, and (ii) G is partitioned into $(l/c)^d$ axis-parallel (hyper) squares—each of which is called a *cell*—such that each cell has length c on each dimension. See Figure 4 for a 2d example. The operation *grid-find-any*(G, P) returns, for each cell of G, an arbitrary point of P covered by the cell (if no such a point exists, then no output is needed for the cell). It is worth nothing that *grid-find-any*(G, P) is essentially the union of several *find-any* operations, one for each cell.

Now we are ready to present our $(1+\epsilon)$-approximate algorithm, called Asum3. First, we apply Asum to obtain a point p that serves as a 3-approximate answer. Let $r = r_p/(\phi M)$; recall that r_p is the aggregate distance of p. Next, for each query $q_i \in Q$, we perform an operation *grid-find-any*(G_i, P), where G_i is the $(2r, \epsilon r/(3\sqrt{d}))$-regular grid centered at q_i. Collect into a set C all the points returned by the M grid-find-any operations. The points in C are the only candidates to be considered. We return the one $p' \in C$ with the smallest $r_{p'}$. Algorithm 5 summarizes the above steps in pseudo code.

Algorithm 5: Asum3 (P, Q, ϕ, sum)

```
(p, Q_φ^p) = the output of Asum (P, Q, φ, sum);
r = r_p/(φM);
set C = ∅;
for i = 1, ..., M do
    G_i = the (2r, εr/(3√d))-regular grid centered at q_i;
    add to C the points returned by
    grid-find-any(G_i, P);
return (p', Q_φ^{p'}) where p' is the point in C with the
smallest r_{p'};
```

Theorem 7 Asum3 *returns a* $(1+\epsilon)$*-approximate answer for any sum* Fann *query.*

Proof Let $(p^*, Q_\phi^{p^*})$ be an optimal answer to the query. Since (p, Q_ϕ^p) is 3-approximate, we know that $r_{p^*} \leq r_p \leq 3r_{p^*}$. Hence:

$$r_{p^*}/(\phi M) \leq r \leq 3r_{p^*}/(\phi M). \tag{11}$$

The fact $r_{p^*}/(\phi M) \leq r$ implies that p^* is within distance r from at least one query point in $Q_\phi^{p^*}$; let us denote this query point by q^*.

Let G^* be the $(2r, \epsilon r/(3\sqrt{d}))$-regular grid centered at q^*. Since $d(p^*, q^*) \leq r$, we know that p^* is covered by G^*, and hence, falls in a certain cell in G^*. We denote this cell by c.

Asum3 performs a *grid-find-any*(G^*, P) operation. Let $\hat{p}$ be the point this operation returns for cell c; note that the operation must return a point for c because c covers at least one point of P, namely, p^*. If $\hat{p} = p^*$, then Asum3 returns an optimal answer; and we are done. Next, we focus on the scenario where $\hat{p} \neq p^*$. Since c is a (hyper) square with length $\epsilon r/(3\sqrt{d})$ on each dimension, any two points in c can have distance at most $\epsilon r/3$. It thus follows that

$$d(\hat{p}, p^*) \leq \epsilon r/3. \tag{12}$$

We complete the proof with the following argument:

$$\begin{aligned}
r_{\hat{p}} &= \sum_{x \in Q_\phi^{\hat{p}}} d(\hat{p}, x) \leq \sum_{x \in Q_\phi^{p^*}} d(\hat{p}, x) \\
&\leq \sum_{x \in Q_\phi^{p^*}} (d(\hat{p}, p^*) + d(p^*, x)) \\
&= \phi M \cdot d(\hat{p}, p^*) + r_{p^*} \\
&\leq \phi M \cdot \epsilon r/3 + r_{p^*} \qquad \text{(by (12))} \\
&\leq \epsilon \cdot r_{p^*} + r_{p^*} \qquad \text{(by (11))} \\
&= (1+\epsilon) r_{p^*}. \square
\end{aligned}$$

Having established the quality guarantee of Asum3, next we discuss the efficiency of Asum3. Recall that, after invoking Asum, the algorithm performs M find-any-grid operations. A crucial observation is that each grid-find-any operation involves only a small number of find-any operations. Specifically, each grid-find-any operation works with a $(2r, \epsilon r/(3\sqrt{d}))$-regular grid, which has $(\frac{2r}{\epsilon r/(3\sqrt{d})})^d = (\frac{6\sqrt{d}}{\epsilon})^d = O(1)$ cells (both ϵ and d are constants). A grid-find-any operation is the union of the same number of find-any operations.

As mentioned earlier, Asum3 aims at a low dimensionality d, in which case a find-any operation can be

efficiently processed with an R-tree. Given a rectangle ρ, a find-any can proceed as if it was doing range reporting with ρ on the R-tree, except that it terminates as soon as the first point in ρ is found. Following this idea, a grid-find-any operation can be implemented by running all the $O(1)$ corresponding find-any operations concurrently, in order to avoid accessing the same node of the R-tree more than once.

So in summary, ASUM3 does M iterations of grid-find-any operations. Each grid-find-any opertaion does $(\frac{6\sqrt{d}}{\epsilon})^d$ number of find-any operation in parallel, where each find-any operation does a range-search-like operation on an R-tree over P, using a d-dimension rectangle ρ with length $\frac{2r}{\epsilon r/(3\sqrt{d})}$ in each extent, such that it terminates and returns the first point found from the R-tree.

6 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.

6.1 A $(1+2\sqrt{2})$-approximate algorithm

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 $\text{MEB}(S)$ to denote the *minimum enclosing ball* of a set of points S, namely the smallest ball that fully contains S. Our first algorithm, AMAX, is presented in Algorithm 6. 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 6: AMAX (P, Q, ϕ, max)

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

Even though the proof for its approximation bound is rather involved, the algorithm AMAX itself is actually easy to understand, as shown in Algorithm 6. Basically, it iterates through each $q_i \in Q$, and finds the center of c_i of the minimum enclosing ball for q_i and its ϕM nearest neighbors in Q. It then finds an approximation candidate p_i that's the nearest neighbor of c_i in P. The final approximation answer is the p_j that has the minimum max distance to p_j's ϕM nearest points in Q, among all M candidates for $j = 1, \ldots, M$. The intuition behind AMAX is that the minimum enclosing ball around a query point and its ϕM nearest neighbors in Q naturally forms a good candidate for lower bounding the max distance from a point in space to ϕM points in Q. So the nearest neighbor for the center of this ball from P must be a good candidate for minimizing the max distance to any ϕM points in Q.

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. If $\mathcal{B}(c,r)$ covers all points (geometrically) in S, we say $S \subseteq \mathcal{B}(c,r)$. For a point o, a value γ, let $S_{o,\gamma}$ be any set of points such that

$$o \in S_{o,\gamma} \text{ and } S_{o,\gamma} \subseteq \mathcal{B}(o, 2\gamma). \tag{13}$$

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

Proof Given $S_{o,\gamma} \subseteq \mathcal{B}(o, 2\gamma)$, $r_s \le 2\gamma$ 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 $d(o,s) \le r_s$. □

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 5 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 $\mathcal{B}(p, d(p,a))$. Extend $\overline{oe}$ and hit $\partial\mathcal{B}(p, d(p,a))$ at j. Similarly, let $C(p,e,j)$ be the cap of $\mathcal{B}(p, d(p,a))$ 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

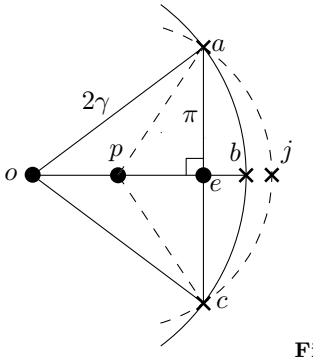


Fig. 5 Lemma 2.

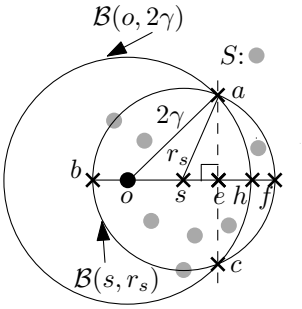


Fig. 6 e cannot be outside the line interval os.

belongs to the segment $\overline{oe}$, in $\triangle opa$, $d(o,p)+d(p,a) > d(o,a) = d(o,b) = d(o,p)+d(p,b)$. Thus, $d(p,j) = d(p,a) > d(p,b)$. □

Lemma 3 *For any point set $S_{o,\gamma}$ satisfying* (13)*, let $\mathcal{B}(s,r_s) = \text{MEB}(S_{o,\gamma})$, and $d(o,s) = z$, 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 6). Now,

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

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 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 6). 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 $\triangle oae$,

$$\begin{aligned} d(a,e)^2 &= d(o,a)^2 - d(o,e)^2 \\ &= (2\gamma)^2 - (d(o,s) + d(s,e))^2 \\ &= (2\gamma)^2 - z^2 - d(s,e)^2 - 2z\,d(s,e). \end{aligned} \tag{14}$$

While in the right triangle $\triangle sae$,

$$\begin{aligned} d(s,a) &= \sqrt{d(s,e)^2 + d(a,e)^2} \\ &= \sqrt{d(s,e)^2 + (2\gamma)^2 - z^2 - d(s,e)^2 - 2z\,d(s,e)} \quad \text{(by (14))} \\ &= \sqrt{(2\gamma)^2 - z^2 - 2z\,d(s,e)}, \end{aligned}$$

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

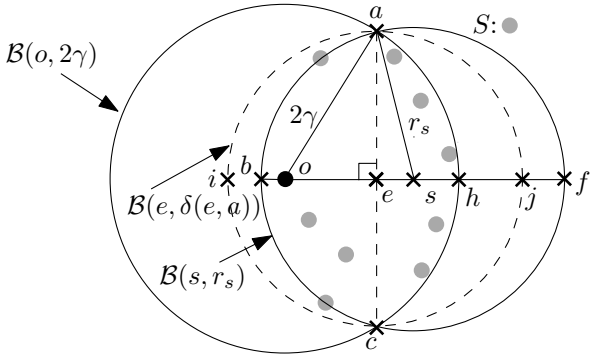


Fig. 7 Proof of Lemma 3

Given this, we must end up at a case in Figure 7. Clearly,

$$r_s = d(s,a) > d(e,a), \tag{15}$$

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

$$S_{o,\gamma} \subseteq \mathcal{B}(o,2\gamma) \cap \mathcal{B}(s,r_s). \tag{16}$$

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 $\mathcal{B}(e,d(e,a))$ and suppose its boundary intersects with the line through b, f at i and j. The ball $\mathcal{B}(e,d(e,a))$ 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,

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

This means that there is a ball with radius $d(e,a) < r_s$ 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* (13)*, let $\mathcal{B}(s,r_s) = \text{MEB}(S_{o,\gamma})$, and $d(o,s) = z$, then $z + r_s \leq 2\sqrt{2}\gamma$.*

Proof

$$\begin{aligned} z + r_s &\leq z + \sqrt{(2\gamma)^2 - z^2} \qquad \text{(by Lemma 3)} \\ &\leq \sqrt{2(z^2 + (2\gamma)^2 - z^2)} = 2\sqrt{2}\gamma. \end{aligned} \tag{17}$$

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

We are now ready to present the main theorem.

Theorem 8 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.,

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

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

$$r_x \le r^*. \tag{18}$$

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.,

$$\max(q, Q_\phi^*) \le 2r_x. \tag{19}$$

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,

$$\max(q, Q_\phi^q) \le \max(q, Q_\phi^*) \le 2r_x, \tag{20}$$

If we view q as o and r_x as γ, clearly $S_{o,\gamma} = Q_\phi^q$ satisfies (13). Line 3 in AMAX also finds $b = \mathcal{B}(c, r_q) = \text{MEB}(Q_\phi^q)$, by Lemma 4, we have:

$$d(q, c) + r_q \le 2\sqrt{2} r_x. \tag{21}$$

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:

$$\begin{aligned}
r_p &= \max_{y \in Q_\phi^p} d(p, y) \\
&\le \max_{y \in Q_\phi^q} d(p, y) \qquad (Q_\phi^p \text{ is the } \phi M \text{ NNs of } p \text{ in } Q) \\
&\le \max_{y \in Q_\phi^q} (d(p, c) + d(c, y)) \\
&\le \max_{y \in Q_\phi^q} (d(p^*, c) + d(c, y)) \qquad (p = nn(c, P)) \\
&= d(p^*, c) + r_q \qquad (\mathcal{B}(c, r_q) = \text{MEB}(Q_\phi^q)) \\
&\le d(p^*, q) + d(q, c) + r_q \\
&\le r^* + 2\sqrt{2} r_x \qquad (\text{due to } q \in Q_\phi^* \text{ and } (21)) \\
&\le (1 + 2\sqrt{2}) r^*. \qquad (\text{by } (18))
\end{aligned} \tag{22}$$

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 9 *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 (22), we have:

$$\begin{aligned}
r_{p'} &= \max_{y \in Q_\phi^{p'}} d(p', y) \\
&\le \max_{y \in Q_\phi^q} d(p', y) \\
&\le \max_{y \in Q_\phi^q} (d(p', c) + d(c, y)) \\
&\le \max_{y \in Q_\phi^q} (\beta d(p, c) + d(c, y)) \qquad (p' \text{ is } \beta\text{-approx. of } p) \\
&\le \beta \max_{y \in Q_\phi^q} (d(p, c) + d(c, y)) \\
&\le \beta(1 + 2\sqrt{2}) r^*, \ (\text{by the same derivation in } (22))
\end{aligned}$$

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

6.1.1 Reduce the cost of AMAX

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 10 *For any $0 < \lambda < 1$, 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 max* 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 8 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.

6.2 A two-dimensional 2-approximate algorithm

Next, we show that the idea presented in Section 5.2 (which resorts to sector nearest neighbors) can also be applied to obtain a 2-approximate algorithm, named Amax2, for max Fann queries in 2d space.

Algorithm 7: Amax2 (P, Q, ϕ, max)

1 Same as Asum2 (but note the change in the aggregate function).

Theorem 11 Amax2 *returns a 2-approximate answer to any max* Fann *query.*

Proof Let p^* be an optimal answer to the query, and q^* be the query point in $Q^{p^*}_\phi$ that is the *farthest* to p^*. Let p' be the point closest to q^*, among all the points of P that are in the same sector of q^* as p^*. By the same argument as in Theorem 5, we have $d(p', p^*) \leq d(p^*, q^*)$.

Thus, we complete the proof with:

$$\begin{aligned} r_{p'} &= \max_{x \in Q^{p'}_\phi} d(p', x) \leq \max_{x \in Q^{p^*}_\phi} d(p', x) \\ &\leq \max_{x \in Q^{p^*}_\phi} (d(p', p^*) + d(p^*, x)) \\ &\leq d(p', p^*) + r_{p^*} \\ &\leq d(p^*, q^*) + r_{p^*} \\ &= 2r_{p^*}.\square \end{aligned}$$

6.2.1 Improve the efficiency of Amax2

Similar to Asum2, Amax2 needs to perform $6M$ sector nearest neighbors. A sampling based approach, similar to that in Section 6.1.1 for Amax, can be adapted to dramatically reduce its cost, while maintaining its high approximation quality.

Theorem 12 *For any $0 < \lambda < 1$, executing the* Amax2 *algorithm only on a random subset of $f(\phi, \lambda)$ points of Q returns a 2-approximate answer to the max* 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 Following the proof of Theorem 11, we note that the approximation ratio is guaranteed as long as q^* is one of the points in Q that have gone through the algorithm. Hence, instead of trying all possible $q \in Q$, we can leverage the same intuition as followed by Theorem 3 for the SUM case.

We execute the algorithm only on a randomly chosen subset of Q, of course q^* may not be one of them. Nevertheless, if some other q' has been chosen that is among points in Q^*_ϕ (thus also in Q), i.e., q' is one of the ϕM closest points to p^* from Q, the proof can still go through except that now we have q' instead of q^* in Figure 3(b). Following the same argument as that in the proof of Theorem 6, we still have $d(p', p^*) \leq d(p^*, q')$.

On the other hand, note that since q^* in this case is the *furthest* point to p^* among the closest ϕM points to p^* from Q (which is $Q^{p^*}_\phi$), as long as $q' \in Q^*_\phi = Q^{p^*}_\phi$, we always have $d(q', p^*) \leq d(q^*, p^*)$.

Combining the above observations, following the same step from the proof of Theorem 11, we have, if $q' \in Q^*_\phi$:

$$\begin{aligned} r_{p'} &= \max_{x \in Q^{p'}_\phi} d(p', x) \leq \max_{x \in Q^{p^*}_\phi} d(p', x) \\ &\leq \max_{x \in Q^{p^*}_\phi} (d(p', p^*) + d(p^*, x)) \\ &\leq d(p', p^*) + r_{p^*} \\ &\leq d(p^*, q') + r_{p^*} \\ &\leq d(p^*, q*) + r_{p^*} = 2r_{p^*}. \end{aligned}$$

That said, we only need at least one of the points in Q^*_ϕ being considered by the algorithm Amax2. If we run Amax2 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.□

6.3 A $(1+\epsilon)$-approximate algorithm in low-dimensional spaces

In this section, we give an algorithm that guarantees $(1+\epsilon)$-approximate answers for any constant ϵ when the dimensionality d is a small constant. This algorithm, called AMAX3, is an adaptation of ASUM3, and also makes use of the grid-find-any operation defined in Section 5.3.

Algorithm 8: AMAX3 (P, Q, ϕ, max)

1 (p, Q_ϕ^p) = the output of AMAX $(P, Q, \phi, \max)$;
2 $r = r_p$;
3 set $C = \emptyset$;
4 **for** $i = 1, \ldots, M$ **do**
5 $\quad G_i$ = the $(2r, \epsilon r/((1+2\sqrt{2})\sqrt{d}))$-regular grid centered at q_i;
6 $\quad$ add to C the points returned by *grid-find-any*(G_i, P);
7 **return** $(p', Q_\phi^{p'})$ where p' is the point in C with the smallest $r_{p'}$;

Theorem 13 AMAX3 *returns a* $(1+\epsilon)$*-approximate answer for any max* FANN *query.*

Proof Let $(p^*, Q_\phi^{p^*})$ be an optimal answer to the query. Since (p, Q_ϕ^p) is $(1+2\sqrt{2})$-approximate, we know that $r_{p^*} \le r_p \le (1+2\sqrt{2}) r_{p^*}$. The fact $r = r_p \ge r_{p^*}$ implies that p^* is within distance r from at least one query point in $Q_\phi^{p^*}$; let us denote this query point by q^*.

Let G^* be the $(2r, \epsilon r/((1+2\sqrt{2})\sqrt{d}))$-regular grid centered at q^*. Since $d(p^*, q^*) \le r$, we know that p^* is covered by G^*, and hence, falls in a certain cell in G^*. We denote this cell by c.

AMAX3 performs a *grid-find-any*(G^*, P) operation. Let $\hat{p}$ be the point this operation returns for cell c. If $\hat{p} = p^*$, then ASUM3 returns an optimal answer; and we are done. Next, we focus on the scenario where $\hat{p} \ne p^*$. Since c is a (hyper) square with length $\epsilon r/((1+2\sqrt{2})\sqrt{d})$ on each dimension, we know:

$$d(\hat{p}, p^*) \le \epsilon r/(1+2\sqrt{2}). \quad (23)$$

We complete the proof with the following argument:

$$\begin{aligned} r_{\hat{p}} &= \max_{x \in Q_\phi^{\hat{p}}} d(\hat{p}, x) \le \max_{x \in Q_\phi^{p^*}} d(\hat{p}, x) \\ &\le \max_{x \in Q_\phi^{p^*}} (d(\hat{p}, p^*) + d(p^*, x)) \\ &= d(\hat{p}, p^*) + r_{p^*} \\ &\le \epsilon r/(1+2\sqrt{2}) + r_{p^*} \qquad \text{(by (23))} \\ &\le \epsilon \cdot r_{p^*} + r_{p^*} \\ &= (1+\epsilon) r_{p^*}. \square \end{aligned}$$

As in ASUM3, every grid-find-any operation performed by AMAX3 has $O(1)$ cells, and can have all the corresponding find-any operations executed on an R-tree concurrently.

More specifically, AMAX3 does M iterations of grid-find-any operations. Each grid-find-any opertaion does $(\frac{(2+4\sqrt{2})\sqrt{d}}{\epsilon})^d$ number of find-any operation in parallel, where each find-any operation does a range-search-like operation on an R-tree over P, using a d-dimension rectangle ρ with length $\frac{2r}{\epsilon r/((1+2\sqrt{2})\sqrt{d})}$ in each extent, such that it terminates and returns the first point found from the R-tree.

7 Extensions

This section contains a collection of useful extensions to the proposed algorithms.

7.1 The k-FANN problem

Given a query group Q and an aggregate function g (= sum or max), a *top-k* FANN query returns the k points $p \in P$ with the smallest aggregate distances r_p. We explain how to process such queries next.

Exact methods. In 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.

To adapt 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.

Approximation methods. All our approximation algorithms can be extended to return k answers, such that the i-th $(1 \le i \le k)$ one approximates the i-th best answer with the same approximation ratios we have already proved for $k = 1$.

For the ASUM algorithm, Line 3 of Algorithm 2 finds, for each $q \in Q$, 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 optimization in Section 5.1.1 can still be applied.

Similarly, in ASUM1, after finding the geometric centroid of the query group, we find its top k nearest neighbors from P and return them as the answer. In ASUM2, Line 3 of Algorithm 4 now should add to C the k sector

nearest neighbors of q_i; that is, for each sector of q_i, add to C the k points closest to q_i among all the points of P falling in the sector. The output then consists of the k points in C with the best aggregate distances.

To adapt ASUM3, at Line 1 of Algorithm 5 we invoke (the top-k version of the) ASUM to obtain k answers. Let $p_1, ..., p_k$ be the k points in these answers. For each p_i, execute through Lines 2-7 to obtain an approximate answer by setting $p = p_i$.

Regarding AMAX, at Line 5 of Algorithm 6, after obtaining $\mathcal{B}(c, r) = \text{MEB}(Q_\phi^q)$ for each point q in 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 optimization in Section 6.1.1 can still be applied.

Finally, AMAX2 and AMAX3 can be modified in the same manner as ASUM2 and ASUM3, respectively.

7.2 Other metric spaces

Our presentation so far considers that the distance between two points is their Euclidean distance. Next, we discuss how to support other distance metrics.

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 [6]. 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.

Algorithms ASUM1, ASUM2, ASUM3, AMAX, AMAX2, and AMAX3 are tailored for Euclidean distance, i.e., L_p norm with $p = 2$. However, it is well-known that, given two points p, q, their L_p distances under different $p \in [1, \infty)$ are all within a factor that depends only on d, and hence, is a constant when $d = O(1)$ (e.g., in 2d space, the L_p distances of all $p \in [1, \infty)$ are within twice the L_∞ distance). This means that ASUM1, ASUM2, AMAX, and AMAX2 all guarantee constant approximation factors for *every* such L_p norm when $d = O(1)$ (for this purpose, we can simply return the answers under L_2 norm regardless of p). This also means that, again when $d = O(1)$, ASUM3 and AMAX3 can be used to obtain $(1 + \epsilon)$-approximate answers for any L_p norm (for this purpose, return the answers under L_2 norm after adjusting ϵ within a constant factor). In all the above cases, the approximation ratio is adjusted by a constant factor depending on d, which is $\sqrt{d}$ for $p \in [1, \infty)$ that can be treated as a constant if we assume $d = O(1)$.

8 Experiments

We implemented all algorithms in C++, and executed our experiments on a machine with an Intel Xeon 3.07 GHz CPU and 24GB memory. For all index structures and data files, the page size is set to 4KB. For all of our approximation algorithms, we used the optimized versions with $1/\phi$ sampled query points from Q as guided by Theorems 3, 6, 10, and 12. The sample sizes indicated in Theorems 3, 6, 10, and 12 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 all 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 [5] consisting of $68,040$ points in 32 dimensions, the *MNIST* dataset [14] with $60,000$ points in 50 dimensions, and the *Cortina* dataset [21] with $1,088,864$ points in 74 dimensions.

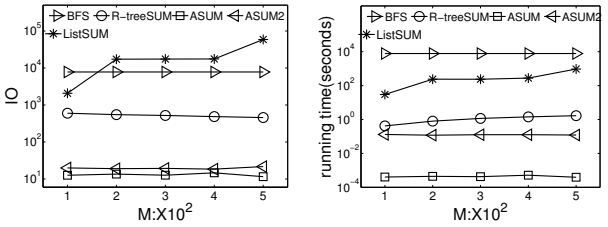
Query groups. For the FANN problem, the cost of the query depends on several critical factors, including the location of the center (either the geometric median or the center of the minimum enclosing ball) of Q, the covering range of Q (i.e., the space enclosed by MEB(Q)), how points are distributed 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.

8.1 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.

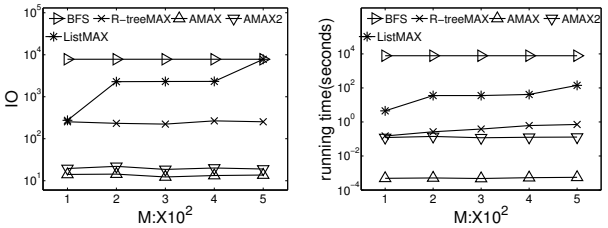
Quality of approximation. The approximation ratios of ASUM, ASUM2, AMAX and AMAX2 for the 4 sets of experiments are shown in Figure 8 and Figure 9. Clearly, in all these cases, all methods achieved excellent approximation quality for the sum and max FANN problems, respectively. The average approximation ratio is between 1.1 to 1.3 in all these experiments. More importantly, all algorithms behave quite stably, with the 95% percentile at 1.5 for most scenarios. Figures 8(a),8(b), 9(a), and 9(b) show that their approximation qualities are not affected by the size of the query group or the dataset. Figure 8(c) and 9(c) indicate that a larger ϕ value leads to better approximation quality. Note that when $\phi = 1$, we used the ASUM1 algorithm for the sum FANN, and the special-case algorithm from the group enclosing query [15] for the max FANN. 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. In all cases, it is clear that ASUM2 has achieved better approximation quality than the approximation returned by ASUM, whereas in the case of max, the approximation quality of AMAX and AMAX2 are similar in most cases (but keep in mind that AMAX2 does have a tighter theoretical bound, which is still of great interest).

Finally, we study the impact of dimensionality on the ASUM and AMAX methods (note that ASUM2 and AMAX2 work well in 2d and it becomes difficult to find sector nearest neighbors in higher dimensions). Figure 8(d) and 9(d) show that ASUM and AMAX 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.



(a) IO. (b) running time.

Fig. 10 Methods for sum FANN in low dimensions: vary M.

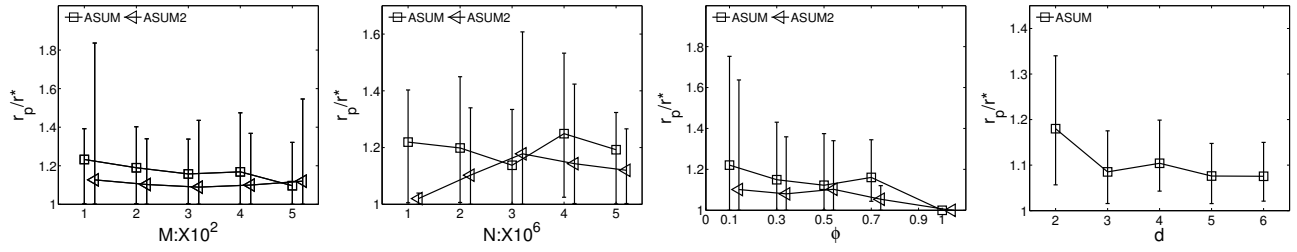


(a) IO. (b) running time.

Fig. 11 Methods for max FANN in low dimensions: vary M.

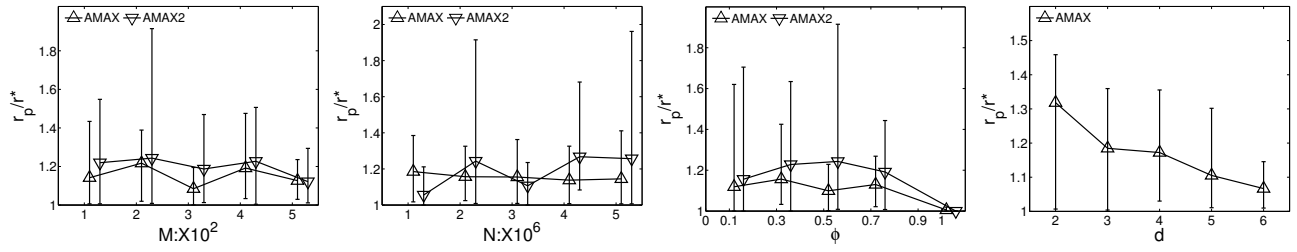
Efficiency. We next focus on the efficiency of different algorithms. Since some algorithms incur both considerable disk IOs and CPU costs, we reported both the number of IOs and the end-to-end running time. For the exact methods in low dimensions, we observe that the *List* algorithm is strictly worse than the *R-tree* method, as shown in Figures 10 and 11 for SUM and MAX respectively. Hence, in subsequent evaluations, we report the *R-tree* method as the representative of our exact methods in the other figures.

Figure 10 and 11 show 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. The *List* algorithm only runs efficiently when M is small, in terms of both IOs and running time and becomes much worse than the *R-tree* method when M increases. Our approximation algorithms, ASUM, AMAX, ASUM2 and AMAX2, are 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 ASUM2 and AMAX2 methods are able to answer a query Q with 300 query points over 2 million points in P in just about 0.1 second and 10 IOs. Both ASUM and AMAX methods have even better performance, 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



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

Fig. 8 Approximation quality of ASUM and ASUM2 methods in low dimensions.



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

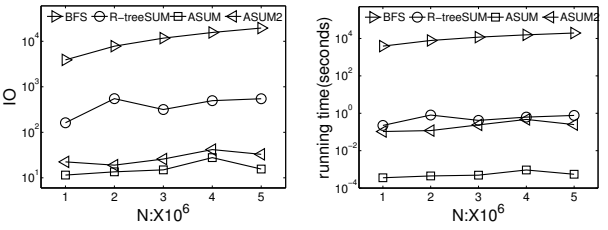
Fig. 9 Approximation quality of AMAX, and AMAX2 methods in low dimensions.

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).

The running time of ASUM2 and AMAX2 are relatively higher compared to those of ASUM and AMAX. That is because, both algorithms need to run the 6 constrained NN queries in parallel, which is IO efficient but incurs more (cpu-bound) geometry computations. However, even though ASUM2 and AMAX2 are more expensive compared to ASUM and AMAX, they do have strictly better approximation quality in theory, and in many cases too in practice. Note that the approximation bounds for ASUM and AMAX are tight as well (e.g., in Appendix A for AMAX and Theorem 4 for ASUM), which means that there are cases where ASUM2 and AMAX2 will have noticeably better approximation bounds than ASUM and AMAX respectively.

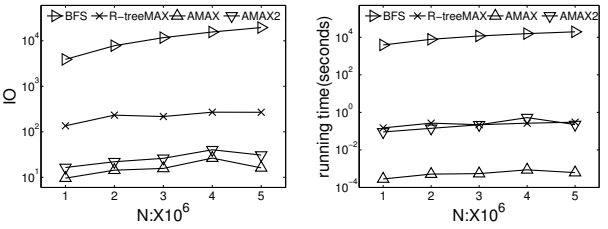
Also note that both ASUM2 and AMAX2 have much better IOs than the exact R-tree based methods. In terms of running time, they both outperform the R-tree methods (note that the y-axis is in log-scale), even though the performance improvement is not as dramatic as the gaps achieved by ASUM and AMAX. However, since they both have much better IOs than the R-tree methods (by at least one order of magnitude), this indicates that they have better scalability with smaller memory and/or bigger data size.

We next study the effect of dataset size. Figure 12 and 13 show 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. The ASUM2 and AMAX2 have the similar performance as the ASUM and AMAX in terms of IO, but issue a higher running time, for the similar reason as that explained before.



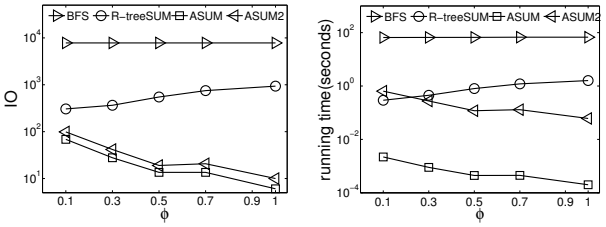
(a) IO. (b) running time.

Fig. 12 Methods for sum FANN in low dimensions: vary N.



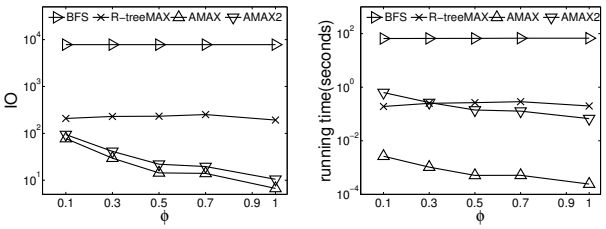
(a) IO. (b) running time.

Fig. 13 Methods for max FANN in low dimensions: vary N.



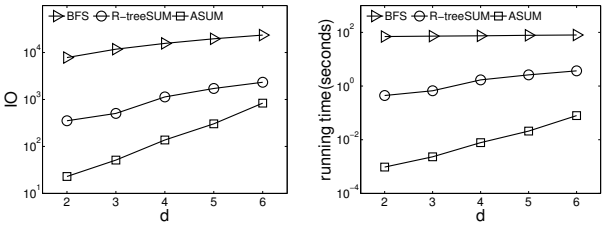
(a) IO. (b) running time.

Fig. 14 Methods for sum FANN in low dimensions: vary ϕ.



(a) IO. (b) running time.

Fig. 15 Methods for max FANN in low dimensions: vary ϕ.

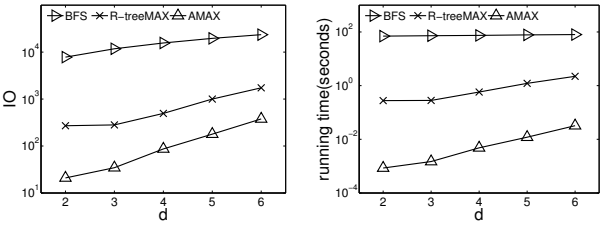


(a) IO. (b) running time.

Fig. 16 Methods for sum FANN in low dimensions: vary d.

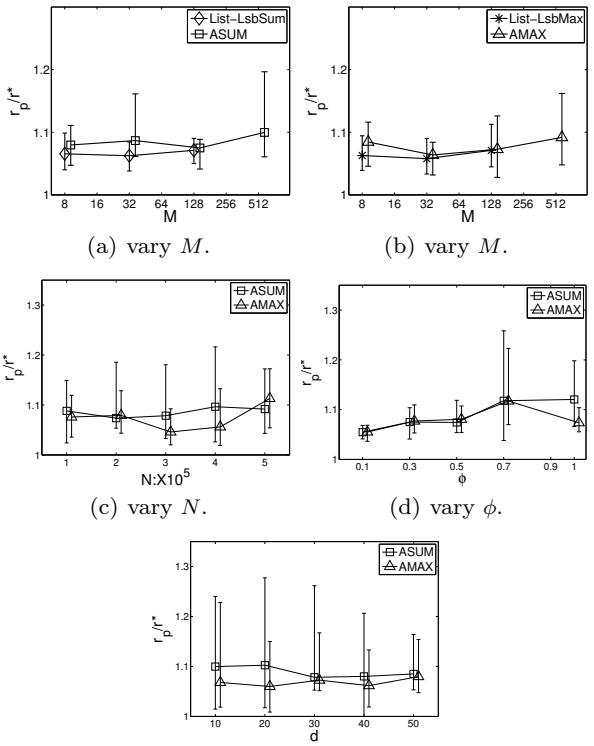
Our next experiment investigates the effect of ϕ. Since ASUM and AMAX both issue $1/\phi$ NN queries on an R-tree indexing the P points, when ϕ is small, their query costs would be higher. This trend was observed in Figures 14 and 15. In particular, Figures 14(a) and 15(a) show that ASUM and AMAX have similar IO cost as the *R-tree* method when $\phi = 0.1$, but has much lower IO costs for larger ϕ. On the other hand, ASUM2 and AMAX2 issue $6/\phi$ NN queries, and have the similar trend. In terms of the running time, ASUM and AMAX are both still much lower than all other methods for all ϕ values as shown in Figures 14(b) and 15(b). Except when ϕ is really small, both ASUM2 and AMAX2 have outperformed the R-tree based exact methods.

Next experiment studies the effect of dimensionality, where we tested all algorithms on the *RC* datasets from 2 to 6 dimensions as shown in Figures 16 and 17. 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



(a) IO. (b) running time.

Fig. 17 Methods for max FANN in low dimensions: vary d.



(a) vary M. (b) vary M.

(c) vary N. (d) vary ϕ.

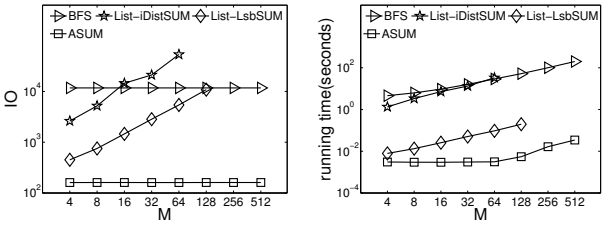
(e) vary d.

Fig. 18 Approximation quality of ASUM and AMAX in high dimensions.

dimensions. Nevertheless, ASUM and AMAX are clearly the winner in all dimensions.

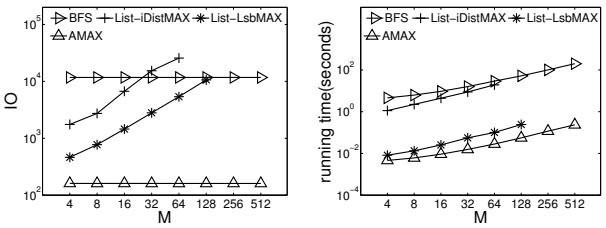
8.2 High dimensions

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 [24], the state of the art for answering approximate NN queries in high dimensions. We also tested



(a) IO. (b) running time.

Fig. 19 Methods for sum FANN in high dimensions: vary M.



(a) IO. (b) running time.

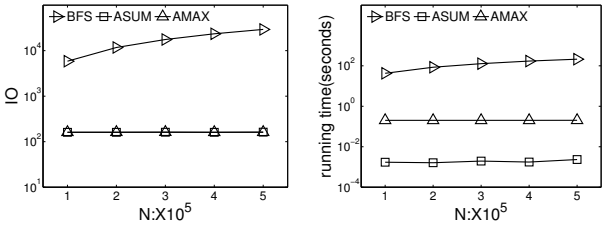
Fig. 20 Methods for max FANN in high dimensions: vary M.

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

Efficiency. Figures 19 and 20 show the efficiency of all methods when varying M. It is clear that in all cases,



(a) IO. (b) running time.

Fig. 21 High dimensions: vary N.

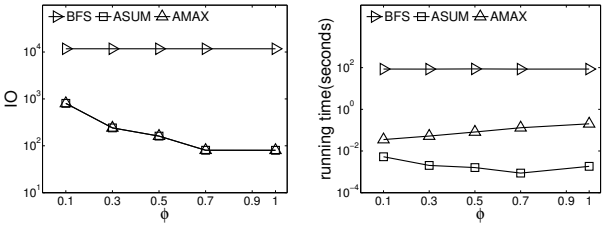
ASUM and AMAX maintain their superiority over other methods by 2–4 orders of magnitude in terms of both IOs and running time. In 30 dimensions, for 256 points in a query group over $200,000$ data points, ASUM takes only 0.01 second per query; AMAX is more expensive, due to the higher computation cost 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.

We next study the efficiency of the methods by varying the size of the dataset. The results are shown in Figure 21. 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.

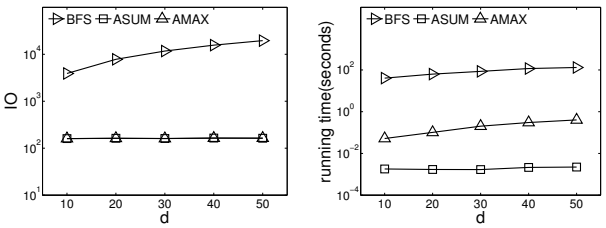
Figure 22 shows the experimental results when we vary ϕ. Similar to the results in low dimensions, smaller ϕ values lead to higher costs for both ASUM and AMAX, due to the $1/\phi$ sample size. Nevertheless, they are still much more efficient than *BFS*.

We also tested their performances in different dimensions as shown in Figure 23. 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



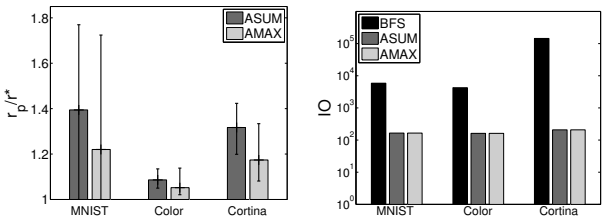
(a) IO. (b) running time.

Fig. 22 High dimensions: vary ϕ.



(a) IO. (b) running time.

Fig. 23 High dimensions: vary d.



(a) approximation ratio. (b) IO.

Fig. 24 All datasets in high dimensions.

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.

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 24. 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.

8.3 Remarks on the sample size from Q

If we use W to denote the sample size of the sampled set from Q in our optimized versions of Asum, Asum 2, Amax, and Amax 2. As guided by Theorems 3, 6, 10, and 12 respectively, formally W depends only on $1/\phi$ multiplied by a small constant of some sort. Clearly, using a smaller W value is only going to lead to worse approximation quality, yet even in this case when $W = 1/\phi$, the approximation quality of all these algorithms are still excellent, as shown in Figures 8 and 9 in low dimensions, and in Figure 18 in high dimensions.

The impact to efficiency is actually fairly easy to understand. Each algorithm only depends linearly on W, since the framework of each of these algorithms is to iterate through each point in the sampled subset of Q and carry out the same procedure in each iteration (note that the procedure in each iteration is of course different for different algorithms, but is the same across different iterations for the same algorithm). The number of iterations only depends on the sample size which is W.

That said, this effect can be seen in Figures 14 and 15 in low dimensions, and Figure 22 in high dimensions, when we vary ϕ. Effectively when $W = 1/\phi$, this can also be interpreted as the impact to efficiency as if we fixed a ϕ value but varied the sample size W that depends on a linear factor multiplied by $1/\phi$.

Being able to run only a small number of iterations that linearly depend on $1/\phi$ is indeed a major strength of our algorithms. For example, in the default set up when $\phi = 0.5$, this means that our algorithms only need to run 2 iterations of a simple procedure in an approximation, e.g., 2 NN queries from P for Asum.

9 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 include extending our investigation on Fann problem for probabilistic data and query groups.

10 Acknowledgment

Feifei Li was supported in part by NSF grants 1053979 and 1251019, and a Google Faculty Award. Ke Yi was supported by HKRGC grants GRF-621413, GRF-16211614, and GRF-16200415, and by a Microsoft grant MRA14EG05.

Yufei Tao was supported in part by GRF grants 4168/13 and 142072/14 from HKRGC. Bin Yao was supported by the National Basic Research Program (973 Program, No.2015CB352403), the NSFC (No. 61202025), and the EU FP7 CLIMBER project (No. PIRSES-GA-2012-318939). Feifei Li and Bin Yao were also supported by NSFC grant 61428204.

References

1. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. Journal of ACM **45**(6), 891–923 (1998)
2. Berchtold, S., Böhm, C., Keim, D.A., Kriegel, H.P.: A cost model for nearest neighbor search in high-dimensional data space. In: PODS (1997)
3. Berg, M., Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational geometry: algorithms and applications. Springer (1997)
4. Böhm, C.: A cost model for query processing in high dimensional data spaces. ACM Transaction on Database Systems **25**(2), 129–178 (2000)
5. Chakrabarti, K., Porkaew, K., Mehrotra, S.: The Color Data Set (2006). Http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.data.html
6. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB (1997)
7. Fagin, R., Kumar, R., Sivakumar, D.: Efficient similarity search and classification via rank aggregation. In: SIGMOD (2003)
8. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS (2001)
9. Ferhatosmanoglu, H., Stanoi, I., Agrawal, D., El Abbadi, A.: Constrained nearest neighbor queries. In: SSTD, pp. 257–278 (2001)
10. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB (1999)
11. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. **24**(2) (1999)
12. Jagadish, H.V., Ooi, B.C., Tan, K.L., Yu, C., Zhang, R.: iDistance: An adaptive B$^+$-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. **30**(2), 364–397 (2005)
13. Kumar, P., Mitchell, J.S.B., Yildirim, E.A.: Approximate minimum enclosing balls in high dimensions using core-sets. ACM Journal of Experimental Algorithmics **8** (2003)
14. LeCun, Y., Cortes, C.: The MNIST Data Set (1998). Http://yann.lecun.com/exdb/mnist
15. Li, F., Yao, B., Kumar, P.: Group enclosing queries. IEEE TKDE (2010)
16. Li, H., Lu, H., Huang, B., Huang, Z.: Two ellipse-based pruning methods for group nearest neighbor queries. In: GIS (2005)
17. Li, Y., Li, F., Yi, K., Yao, B., Wang, M.: Flexible aggregate similarity search. In: SIGMOD, pp. 1009–1020 (2011)
18. Papadias, D., Shen, Q., Tao, Y., Mouratidis, K.: Group nearest neighbor queries. In: ICDE (2004)
19. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. ACM TODS **30**(2), 529–576 (2005)
20. Razente, H.L., Barioni, M.C.N., Traina, A.J.M., Faloutsos, C., Traina Jr., C.: A novel optimization approach to efficiently process aggregate similarity queries in metric access methods. In: CIKM (2008)
21. Rose, K., Manjunath, B.S.: The CORTINA Data Set (2004). Http://www.scl.ece.ucsb.edu/datasets/index.htm
22. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD (1995)
23. Stanoi, I., Agrawal, D., El Abbadi, A.: Reverse nearest neighbor queries for dynamic databases. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 44–53 (2000)
24. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: SIGMOD (2009)
25. Yiu, M.L., Mamoulis, N., Papadias, D.: Aggregate nearest neighbor queries in road networks. IEEE TKDE **17**(6), 820–833 (2005)

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 25 where ϵ is an arbitrarily small positive.

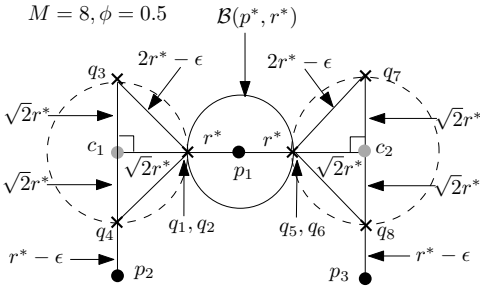


Fig. 25 Amax's approximation bound is tight.

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 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 $\max(p^*, Q_\phi^*) = r^*$.