Approximate Weighted Farthest Neighbors and Minimum Dilation Stars

John Augustine, David Eppstein, and Kevin A. Wortman

Computer Science Department
University of California, Irvine
Irvine, CA 92697, USA
{jea,eppstein,kwortman}@ics.uci.edu

Abstract. We provide an efficient reduction from the problem of querying approximate multiplicatively weighted farthest neighbors in a metric space to the unweighted problem. Combining our techniques with core-sets for approximate unweighted farthest neighbors, we show how to find $(1+\varepsilon)$-approximate farthest neighbors in time $O(\log n)$ per query in D-dimensional Euclidean space for any constants D and ε. As an application, we find an $O(n \log n)$ expected time algorithm for choosing the center of a star topology network connecting a given set of points, so as to approximately minimize the maximum dilation between any pair of points.

1 Introduction

Data structures for proximity problems such as finding closest or farthest neighbors or maintaining closest or farthest pairs in sets of points have been a central topic in computational geometry for a long time [2]. Due to the difficulty of solving these problems exactly in high dimensions, there has been much work on approximate versions of these problems, in which we seek neighbors whose distance is within a $(1+\varepsilon)$ factor of optimal [4]. In this paper, we consider the version of this problem in which we seek to approximately answer farthest neighbor queries for point sets with multiplicative weights. That is, we have a set of points p_i, each with a weight $w(p_i)$, and for any query point q we seek to approximate $\max_i w(p_i) d(p_i, q)$.

We provide the following new results:

- We describe in Theorem 3 a general reduction from the approximate weighted farthest neighbor query problem to the approximate unweighted farthest neighbor query problem. In any metric space, suppose that there exists a data structure that can answer unweighted $(1+\varepsilon)$-approximate farthest neighbor queries for a given n-item point set in query time $Q(n,\varepsilon)$, space $S(n,\varepsilon)$, and preprocessing time $P(n,\varepsilon)$. Then our reduction provides a data structure for answering $(1+\varepsilon)$-approximate weighted farthest neighbor queries in time $O(\log n + (\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})Q(n,\varepsilon/2))$ per query, that uses space $O(n + (\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})S(n,\varepsilon/2))$, and can be constructed in preprocessing time $O(n\log n + (\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})P(n,\varepsilon/2))$.
- We apply core-sets [1] to find a data structure for the approximate unweighted farthest neighbor query problem in $\mathbb{R}^D$, for any constant D, with query time $O(\varepsilon^{\frac{1-D}{2}})$, space $O(\varepsilon^{\frac{1-D}{2}})$, and preprocessing time $O(n + \varepsilon^{\frac{3}{2}-D})$ (Theorem 5). Applying our reduction results in a data structure for the approximate weighted farthest neighbor query problem in $\mathbb{R}^D$, with preprocessing time $O(n\log n + \varepsilon^{\frac{1}{2}-D}\log\frac{1}{\varepsilon})$, query time $O(\log n + \varepsilon^{\frac{-D-1}{2}}\log\frac{1}{\varepsilon})$, and space $O(n)$ (Corollary 6).
- As a motivating example for our data structures, we consider the problem of finding a star-topology network for a set of points in $\mathbb{R}^D$, having one of the input points as the hub of the network, and minimizing the maximum dilation of any network path between any pair of points. By

results of Eppstein and Wortman [5], this problem can be solved exactly in time $O(n\,2^{\alpha(n)} \log^2 n)$ in the plane, and $O(n^2)$ in any higher dimension. By using our data structure for approximate weighted farthest neighbor queries, we find in Corollary 16 a solution to this problem having dilation within a $(1+\varepsilon)$ factor of optimal, for any constant ε and constant dimension, in expected time $O(n \log n)$. More generally, as shown in Theorem 15, we can approximately evaluate the dilations that would be achieved by using every input point as the hub of a star topology, in expected $O(n \log n)$ total time.

2 Problem Definition

We first define the *weighted farthest neighbor query problem* in a metric space, and then extend its definition to relevant variants and special cases. Let $\mathcal{M} = (S, d)$ be a metric space. For any two points p and q in S, $d(p,q)$ denotes the distance between them. We assume we are given as input a finite set $P \subset S$; we denote by p_i the points in P. In addition, we are given positive weights on each point of P, which we denote by a function $w : P \mapsto \mathbb{R}^+$. We wish to preprocess the point set such that for any query point $q \in S$, we can quickly produce a point $f \in P$ that maximizes the weighted distance from q to r. More precisely, f should be the point in P that maximizes (or approximately maximizes) the weighted distance $d_w(q, p_i) = w(p_i) \cdot d(q, p_i)$ from the query point q. If we restrict the weights of all points to be equal, then this problem simplifies to the *unweighted farthest neighbor query problem*.

Finding the exact farthest neighbor to a query point can be computationally expensive. Hence, we are interested in approximate solutions to this problem, with sufficient guarantees that our algorithm produces a reasonably distant point in the weighted sense. To achieve this, we define the *approximate weighted farthest neighbor query problem*, wherein we are given an input of the same type as the weighted farthest neighbor query problem, and wish to preprocess the point set so that we can efficiently produce an ε-approximate weighted farthest point. That is, we wish to find a point $r \in P$ such that $d_w(q, r) \geq (1-\varepsilon) d_w(q, p_i)$ for some predefined ε and all $p_i \in P$. A similar definition holds for the *approximate unweighted farthest neighbor query problem* as well. When the metric space under consideration is Euclidean, we call the problem the *Euclidean approximate weighted farthest neighbor query problem*.

Without loss of generality, we can assume that $0 < w(p) \leq 1$ for all $p \in P$, and that exactly one point p_1 has weight 1. The first assumption can be made to hold by dividing all weights by the largest weight; this leaves approximations to the weighted farthest neighbor unchanged. The second assumption can be made to hold by arbitrarily breaking ties among contenders for the maximum-weight point.

3 Reduction from Weighted to Unweighted

First, let us suppose that we have a family of algorithms $\mathcal{A}_\varepsilon$ for the approximate unweighted farthest neighbor query problem defined over any metric space $\mathcal{M} = (S, d)$. $\mathcal{A}_\varepsilon$ consists of two components, one for preprocessing and the other for querying. For an input instance of the approximate unweighted farthest neighbor query problem, let $f \in P$ maximize the distance between our query point q and f. Using $\mathcal{A}_\varepsilon$, we can find an ε-approximate farthest point r such that $(1-\varepsilon) d(q, f) \leq d(q, r)$. The running time of $\mathcal{A}_\varepsilon$ has two components, namely preprocessing time and query time denoted $P(n, \varepsilon)$, and $Q(n, \varepsilon)$ respectively. It has a space requirement of $S(n, \varepsilon)$. Our real concern is the

weighted version of the problem and in this section, we provide a data structure to reduce an instance of the approximate weighted farthest neighbor query problem such that it can be solved by invoking A_ε. We call this family of reduction algorithms $\mathcal{R}_\varepsilon$.

The preprocessing and querying components of $\mathcal{R}_\varepsilon$ are provided in Procedures 1 and 2 respectively. For convenience, we separate out the points in S into two subsets S_1 and S_2. The first subset S_1 contains all points in P with weights in $[\varepsilon/2, 1]$, and S_2 contains $P \setminus S_1$. We use different approaches for the two subsets producing three candidate approximate farthest neighbors. Our final result is the farther of the three candidates in the weighted sense. We now show that the algorithm is correct both when $f \in S_1$ and when $f \in S_2$.

Case $f \in S_1$:

In this case, we only consider points in S_1. Therefore, we can assume that the weights are within a factor of $2/\varepsilon$ of each other. We partition the points set S_1 into buckets such that each bucket B_i contains all points in S_1 with weight in $[(\varepsilon/2)(1+\varepsilon/2)^{(i-1)}, (\varepsilon/2)(1+\varepsilon/2)^i)$. The number of buckets needed to cover S_1 is $\Theta((1/\varepsilon)\ln(1/\varepsilon))$.

Our data structure consists of separate instances (consisting of both the preprocessing and querying components) of A_ε with the error parameter $\varepsilon/2$ for each bucket B_i. In addition, each B_i is preprocessed by the preprocessing instance of A_ε. To query our data structure for the $(\varepsilon/2)$-approximate farthest neighbor $k_i \in B_i$ from q, we run the querying instance of $\mathcal{A}_\varepsilon$ on each preprocessed bucket B_i. Our candidate approximate weighted farthest neighbor for this case will be the $r_1 = \arg\max_{k_i} d(q, k_i)$.

Lemma 1 *If $f \in S_1$, then*

$$d_w(q, r_1) \geq (1-\varepsilon)d_w(q, f), \tag{1}$$

where f is the exact weighted farthest point and r_1 is as defined above.

Proof. Let us just consider bucket B_i whose weights are between some $(\varepsilon/2)(1+\varepsilon/2)^{(i-1)}$ and $(\varepsilon/2)(1+\varepsilon/2)^i$. Let f_i be the exact weighted farthest point from q in B_i. Within the bucket B_i, k_i is an $(\varepsilon/2)$-approximate farthest point if we don't consider the weights. I.e.,

$$(1-\varepsilon/2)d(q, f_i) \leq d(q, k_i).$$

However, when we consider the weighted distance, $w(k_i)$ and $w(f_i)$ differ by a factor of at most $(1+\varepsilon/2)$ and hence our inequality for the weighted case becomes

$$\frac{1-\varepsilon/2}{1+\varepsilon/2}d_w(q, f_i) \leq d_w(q, k_i).$$

Since $\frac{1-\varepsilon/2}{1+\varepsilon/2} > (1-\varepsilon)$ for $\varepsilon > 0$, we get

$$(1-\varepsilon)d_w(q, f_i) \leq d_w(q, k_i). \tag{2}$$

Let the bucket B_{i^*} contain the weighted farthest point f. By definition, $d_w(q, r_1) \geq d_w(q, k_{i^*})$, which proves the lemma when combined with Equation 2. $\square$

3

Procedure 1 Preprocessing for the approximate unweighted farthest neighbor problem

1: **INPUT:** Set of points P with $n = |P|$ and a distance metric d. A fixed error parameter ε.
   {Farthest point is in S_1.}
2: **Let** $S_1 \subseteq P$ be points with weights in $[2/\varepsilon, 1]$.
3: **Partition** S_1 into buckets $B_0, B_1, \ldots$ such that each B_i contains all points in S_1 whose weights are in $((\varepsilon/2)(1 + \varepsilon/2)^{i-1}, (\varepsilon/2)(1 + (\varepsilon/2))^i]$.
4: **Preprocess** each bucket by calling the preprocessing component of A_ε instance with error parameter $\varepsilon/2$.
   {Farthest point is in S_2.}
5: **Assign** p_1 to be point in P with highest weight (assumed to be 1).
6: **Sort** $S_2 = P \setminus S_1$ in non-decreasing order of $d(p_1, s)$ for all $s \in S_2$.
7: **for** each s taken in reverse sorted order from S_2 **do**
8:     **if** $\exists x \succ s$ in S_2 such that $d_w(p_1, s) \leq d_w(p_1, x)$ **then**
9:         $S_2 \leftarrow S_2 \setminus \{s\}$.
           {Note that this can be done in $O(n)$ time by updating the x that maximizes $d_w(p_1, x)$ at each iteration.}
10:     **end if**
11: **end for**
12: **Ensure** that S_2 is stored in a manner suitable for binary searching.

Procedure 2 Query step of the approximate unweighted farthest neighbor problem

1: **INPUT:** Query point q, B_i for all $1 \leq i \leq 1/\varepsilon \ln(1/\varepsilon)$, and S_2.
   {Farthest point is in S_1.}
2: **Assign** $r_1 \leftarrow q$
3: **for** each B_i **do**
4:     $k_i \leftarrow A_{(\varepsilon/2)}(B_i)$
5:     **if** $d_w(q, r_1) \leq d_w(q, k_i)$ **then**
6:         $r_1 \leftarrow k_i$
7:     **end if**
8: **end for**
   {Farthest point is in S_2.}
9: **Binary Search** through S_2 for point r_2 such that $d(p_1, r_2) \geq d(p_1, q)/\varepsilon$ and $d_w(p_1, r_2)$ is maximized.
10: **Return** $r \leftarrow \arg\max_{\{r_1, r_2 \, p_1\}}(d_w(q, r_1), d_w(q, r_2), d_w(q, p_1))$

4

Case $f \in S_2$:

We sort the points in S_2 in non-decreasing order of their distance from p_1; recall that p_1 is the only point in P with $w(p_1) = 1$. We then pare down S_2 by preserving points x such that $d_w(p_1, x) > d_w(p_1, y)$ for all $y \succ x$ in the sorted sequence of points, and discarding the rest. Our data structure for this case consists of the remaining reduced sorted list, stored in a manner suitable for performing binary searches (e.g. an array or binary search tree). To query the data structure, we binary search for the point r_2 that is the first point in the reduced sorted S_2 farther than $d(p_1, q)/\varepsilon$ from p_1. Our two candidates for this case are r_2 and p_1.

Lemma 2 *If $f \in S_2$, and p_1 is not an ε-approximate weighted farthest point from q, then $d_w(q, r_2) \geq (1 - \varepsilon)d_w(q, f)$.*

Proof. Consider the ball whose center is p_1 and radius is $2d(q, p_1)/\varepsilon$. Since $f \in S_2$, we can say that $d(q, f) > 2d(q, p_1)/\varepsilon$. Otherwise, $d_w(q, f) \leq (\varepsilon/2)d(q, f) \leq d(q, p_1) = d_w(q, p_1)$ (since $w(p_1) = 1$) and this contradicts our assumption that $f \in S_2$. In other words, both f and r_2 fall outside this ball, the former by argument and the latter by construction. Hence,

$$d(p_1, r_2) \geq 2d(p_1, q)/\varepsilon. \tag{3}$$

In addition, r_2 is chosen over f as being the farthest weighted neighbor of p_1 because (by construction) it is the weighted farthest point from p_1 outside the ball in consideration. Hence,

$$d_w(p_1, r_2) \geq d_w(p_1, f). \tag{4}$$

Consider $\triangle p_1 q r_2$.

$$
\begin{aligned}
d(p_1, r_2) &\leq d(p_1, q) + d(q, r_2) \\
&\leq (\varepsilon/2)d(p_1, r_2) + d(q, r_2) \qquad \text{(due to 3)} \\
\Rightarrow \quad d(q, r_2) &\geq (1 - \varepsilon/2)d(p_1, r_2).
\end{aligned}
$$

Again,

$$
\begin{aligned}
d(p_1, r_2) + d(p_1, q) &\geq d(q, r_2) \\
d(p_1, r_2) + (\varepsilon/2)d(p_1, r_2) &\geq d(q, r_2) \qquad \text{(due to 3)} \\
(1 + \varepsilon/2)d(p_1, r_2) &\geq d(q, r_2).
\end{aligned}
$$

Therefore,

$$(1 - \varepsilon/2)d(p_1, r_2) \leq d(q, r_2) \leq (1 + \varepsilon/2)d(p_1, r_2). \tag{5}$$

With similar treatment of $\triangle p_1 q f$, we get

$$(1 - \varepsilon/2)d(p_1, f) \leq d(q, f) \leq (1 + \varepsilon/2)d(p_1, f). \tag{6}$$

From Equation 6, we have

$$
\begin{aligned}
d_w(q, f) &\leq (1 + \varepsilon/2)d_w(p_1, f) \\
&\leq (1 + \varepsilon/2)d_w(p_1, r_2) \qquad \text{(From 4)} \\
&\leq \left(\frac{1 + \varepsilon/2}{1 - \varepsilon/2}\right) d_w(q, r_2) \qquad \text{(From 5)} \\
\Rightarrow \quad d_w(q, r_2) &\geq (1 - \varepsilon)d_w(q, f).
\end{aligned}
$$

$\square$

Theorem 3 *The family of reduction algorithms $\mathcal{R}_\varepsilon$ answer $(1+\varepsilon)$-approximate weighted farthest neighbor queries in time $O(\log n + (\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})Q(n,\varepsilon/2))$ per query, use space $O(n + (\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})S(n,\varepsilon/2))$, and the data structure can be constructed in preprocessing time $O(n\log n + (\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})P(n,\varepsilon/2))$.*

Proof. The proof of approximation guarantee follows from Lemmas 1 and 2 in a straightforward manner. Preprocessing takes time $O((\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})P(n,\varepsilon/2))$ for partitioning into buckets and calling the preprocessing component of $\mathcal{A}_\varepsilon$, and $O(n\log n)$ for sorting the points, hence accounting for the total preprocessing time. While querying, we call the querying component of A_ε for each bucket costing us $O((\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})Q(n,\varepsilon/2))$, and the binary search takes time $O(\log n)$ adding to the total querying time. Similarly, the space taken to store the preprocessed buckets and the sorted sequence of points is $O((\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})S(n,\varepsilon/2))$ and $O(n)$ respectively, adding to the total space required.

$\square$

The preprocessing and querying components (Procedures 1 and 2 respectively) are presented in pseudocode format. Each procedure has two parts to it corresponding to the two cases when $f \in S_1$ and $f \in S_2$. In Procedure 1, we first partition S_1 into buckets and preprocess them. Secondly, we sort S_2 and pare it down according to the requirements of Lemma 2. In Procedure 2, we choose the farthest neighbor r_1 of q in S_1 by querying each bucket and choosing the farthest from the pool of results. Secondly, we binary search in the reduced S_2 to obtain the point r_2 that is at or just farther than a distance of $2d(q,p_1)/\varepsilon$ from q. Our final ε-approximate farthest neighbor r of q is the weighted farthest point in $\{r_1, r_2, p_1\}$ from q.

4 Euclidean Approximate Unweighted Farthest Neighbor Queries

In this section, our problem definition remains intact except for the restriction that S is $\mathbb{R}^D$, where D is a constant, with the Euclidean distance metric. Notice that our points are unweighted and we are seeking a family of approximation algorithms $\mathcal{A}_\varepsilon$ that we assumed to exist as a black box in the previous section. Let $f \in P$ maximize the distance between our query point q and points in S. For a given ε, we are asked to find an ε-approximate farthest point r such that $d(q,r) \geq (1-\varepsilon)d(q,f)$. We can solve this using the ε-kernel technique surveyed in [1]. Let u be the unit vector in some direction. We denote the directional width of P in direction u by $w(u,P)$ and is given by

$$w(u,P) = \max_{s\in P}\langle u,s\rangle - \min_{s\in P}\langle u,s\rangle,$$

where $\langle u,s\rangle$ is the dot product of u and s. An ε-kernel is a subset K of P such that for all unit directions u,

$$(1-\varepsilon)w(u,P) \leq w(u,K).$$

It is now useful to state a theorem from [1, 3] that provides us an ε-kernel in time linear in n and polynomial in $1/\varepsilon$ with the dimension D appearing in the exponent.

Theorem 4 *Given a set P of n points in $\mathbb{R}^D$ and a parameter $\varepsilon > 0$, one can compute an ε-kernel of P of size $O(1/\varepsilon^{(D-1)/2})$ in time $O(n + 1/\varepsilon^{D-(3/2)})$.*

Consider points s_1 and s_2 in P and k_1 and k_2 in K that maximize the directional widths in the direction u, which is the unit vector in the direction of $\overrightarrow{qf}$.

$$(1-\varepsilon)w(u,P) = (1-\varepsilon)\langle u,s_1\rangle - (1-\varepsilon)\langle u,s_2\rangle \leq \langle u,k_1\rangle - \langle u,k_2\rangle.$$

Now, $\langle u, k_2 \rangle \geq \langle u, s_2 \rangle$, because otherwise, $w(u, P)$ can be maximized further. Hence with some substitution, we get

$$(1 - \varepsilon)\langle u, s_1 \rangle \leq \langle u, k_1 \rangle.$$

The point f maximizes the left hand side in the direction of $\overrightarrow{qf}$ because it is the farthest point from q. Therefore, the above inequality suggests that $k_1 \in K$ is an ε-approximate farthest neighbor of q. This implies that the point farthest from q obtained by sequentially searching the ε-kernel K will be an ε-approximate farthest neighbor of q. Hence, we can state the following theorem.

Theorem 5 *Given a set P of n points in $\mathbb{R}^D$ and a parameter $\varepsilon > 0$, there exists a family of algorithms $\mathcal{A}_\varepsilon$ that answers the ε-approximate unweighted farthest neighbor query problem in preprocessing time $O(n + e^{\frac{3}{2} - D})$, and the query time and space requirements are both $O(\varepsilon^{\frac{1-D}{2}})$.*

Combining this theorem with our general reduction from the weighted to the unweighted problem gives us the following corollary.

Corollary 6 *Given a set P of n points in $\mathbb{R}^D$, for any constant D, and given a weight function $w : P \mapsto \mathbb{R}^+$ and a parameter $\varepsilon > 0$, there exists a family of algorithms that answers the ε-approximate weighted farthest neighbor query problem in preprocessing time $O(n \log n + \varepsilon^{\frac{1}{2} - D} \log \frac{1}{\varepsilon})$ and query time $O(\log n + \varepsilon^{\frac{-D-1}{2}} \log \frac{1}{\varepsilon})$ with a space requirement of $O(n)$.*

Proof. We apply Theorem 3 to reduce the weighted problem to a collection of instances of the unweighted problem, which we solve using Theorem 5. The query time comes from combining the results of these two theorems, and the space bound comes from the fact that our overall data structure consists only of a sequence of points together with a family of disjoint subsets of points forming one core-set for each of the instances of the unweighted problem used by Theorem 3. To calculate the time bound, note that Theorem 3 takes time $O(n \log n)$ to sort the low-weight points by distance from p_1, and in the same time we may also perform the partition of high-weight points into buckets used by that theorem. Once we have partitioned the points into buckets, we construct for each bucket an instance of the data structure of Theorem 5; this takes time $O(n_i + e^{\frac{3}{2} - D})$ per bucket and adding these bounds over all buckets gives the stated preprocessing time bound. $\qquad \square$

5 Constrained Minimum Dilation Stars

The *dilation* between two vertices v and w of a weighted graph is defined as the ratio of the weight of the shortest path from v to w, divided by the direct distance between v and w. The dilation of the entire graph is defined as the greatest dilation between any pair of vertices in the graph. A *star* is a connected graph with exactly one internal vertex, called its *center*. Any collection of n points admits n possible stars, since any individual point may serve as a star center. In this section we consider the problem of computing the dilation of all of these n stars. A solution to this problem provides the foundation for a solution to the problem of choosing an optimal center: simply search for the point whose corresponding dilation value is smallest.

Eppstein and Wortman considered [5] the problem of selecting star centers that are optimal with respect to dilation. They showed that for any set of n points $P \subset \mathbb{R}^D$, there exists a set C of $O(n)$ pairs of points such that the worst pair for any center $c \in \mathbb{R}^D$ is contained in C. They give an $O(n \log n)$-time algorithm for constructing C, and go on to consider the problem of computing the

center that admits a star with minimal dilation. They present an $O(n \log n)$ expected-time algorithm for the case when c may be any point in $\mathbb{R}^D$ and D is an arbitrary constant, and an $O(n 2^{\alpha(n)} \log^2 n)$ expected-time algorithm for the case when c is constrained to be one of the input points and $D = 2$.

These results imply that the dilation of all n stars with centers from P may be computed in $O(n^2)$ time: construct C, then for each $c \in P$ and $(v, w) \in C$, evaluate the dilation of the path $\langle v, c, w \rangle$. In this section we improve on this time bound through approximation. We will show that a $(1 - \varepsilon)$-approximation of the dilation of all n stars may be computed in $O(n \log n)$ expected time, and hence an approximately optimal center $c \in P$ may be identified in $O(n \log n)$ expected time.

Our approximation algorithm first uses the results of [5] to compute the optimal $c_{OPT} \in \mathbb{R}^D$. Note that c_{OPT} may or may not be in P, and in general will not be. Our algorithm then partitions P into two subsets: the $k \in O(1)$ points nearest to c_{OPT}, and the other $n - k$ points. We say that a star center $c \in P$ is k-low if it is one of the k points nearest to c_{OPT}, or k-high otherwise. The dilation values for all the k-low centers are computed exactly in $O(n \log n)$ time using a combination of known techniques.

The dilation between any two points $p_i, p_j \in P$ through any k-high center c may be approximated by a weighted distance from c to the centroid of p_i and p_j. Hence the the dilation of the star centered on any k-high c may be approximated by the distance from c to the centroid farthest from c. We use the data structure described in Section 4 to answer each of these $n - k$ weighted farthest-neighbor queries in $O(\log n)$ time, which makes the overall running time of the approximation algorithm $O(n \log n)$.

Definition 7 *Let G be a Euclidean star with vertices V and center c. The dilation between any $v, w \in V \setminus \{c\}$ is*

$$\delta(c, v, w) = \frac{|vc| + |wc|}{|vw|}$$

and the dilation of the star is

$$\Delta(c) = \max_{v,w \in V \setminus \{c\}} \delta(c, v, w).$$

Definition 8 *Let*

- *$P = \langle p_0, \ldots, p_{n-1} \rangle$ be a set of n input points from $\mathbb{R}^D$ for some constant D, each with the potential to be the center of a star with vertices P,*
- *$c_{OPT} \in \mathbb{R}^D$ be the point minimizing $\Delta(c_{OPT})$,*
- *$S = \langle s_0, \ldots, s_{n-1} \rangle$ be the sequence formed by sorting P by distance from c_{OPT},*
- *$\varepsilon > 0$ be a constant parameter,*
- *$\Gamma = 2/\varepsilon - 1$,*
- *k be a constant depending only on Γ,*
- *$L = \{s_i \in S \mid 0 \le i < k\}$ be the k-low centers,*
- *and $H = P \setminus L$ be the k-high centers.*

We require the following claim, which is proved in [5]:

Claim 9 *Let c be the center of a Euclidean star in $\mathbb{R}^D$ for $D \in O(1)$ having vertices V. If $\Delta(c) \le \Gamma$ for some constant Γ, then there exists a constant ρ_Γ such that for any integer i, the D-dimensional annulus centered on c with inner radius ρ_Γ^i and outer radius ρ_Γ^{i+1} contains only $O(1)$ points from V.*
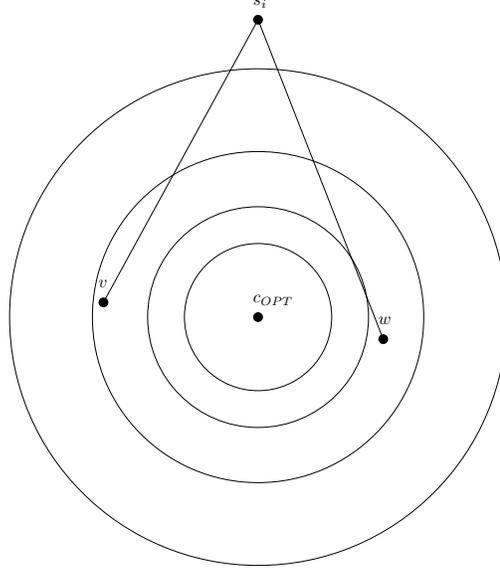
Fig. 1. Planar annuli containing the points s_i, v, and w.

Lemma 10 *For any Γ there exists a constant k depending only on Γ such that any center $s_i \in S$ with $i \geq k$ has dilation $\Delta(s_i) \geq \Gamma$.*

Proof. We first consider the case when $\Delta(c_{OPT}) > \Gamma$. By definition $\Delta(s_i) \geq \Delta(c_{OPT})$ for any $s_i \in S$, so we have that every $\Delta(s_i) > \Gamma$ regardless of the value of k.

We now turn to the case when $\Delta(c_{OPT}) \leq \Gamma$. Define ρ_Γ as in Claim 9, and let

$$A_j = \{x \in \mathbb{R}^d \mid \rho_\Gamma^j \leq |xc_{OPT}| \leq \rho_\Gamma^{j+1}\}$$

be the jth annulus centered on c_{OPT}. Define $l = 1 + \lceil \log_{\rho_\Gamma}(\Gamma - 1) \rceil$, and suppose $s_i \in A_j$. Let v, w be two input points that lie in the annulus A_{j-l}. By the definition of l we have

$$\rho_\Gamma^{l-1} - 1 \geq \Gamma$$
$$\frac{2(\rho_\Gamma^{j-1} - \rho_\Gamma^{j-l})}{2\rho_\Gamma^{j-l}} \geq \Gamma.$$

Since $v \in A_{j-l}$, we have $|s_i v| \geq (\rho_\Gamma^{j-1} - \rho_\Gamma^{j-l})$, and similarly $|s_i w| \geq (\rho_\Gamma^{j-1} - \rho_\Gamma^{j-l})$. Further, $|vw| \leq 2\rho_\Gamma^{j-l}$. So by substitution

$$\frac{(|s_i v|) + (|s_i w|)}{(|vw|)} \geq \Gamma$$
$$\Delta(s_i) \geq \Gamma.$$

Thus the lemma holds if the annuli $A_j, A_{j-1}, \ldots, A_{j-l}$ contain at least k points. We can ensure this by selecting k such that the points $s_{i-k}, \ldots, s_i$ necessarily span at least $l+1$ annuli. By Claim 9,

9

there exists some $m \in O(1)$ such that any annulus contains no more than m input points. So we set $k \geq (l+1)m$. The observation that l depends only on Γ completes the proof.

$\square$

Corollary 11 *For any $c \in L$, $\Delta(c) \leq \Gamma$; for any $c \in H$, $\Delta(c) \geq \Gamma$; and $|L| \in O(1)$.*

Proof. Each claim follows from Lemma 10 and Definition 8. $\square$

Lemma 12 *The set L, as well as the quantity $\Delta(c)$ for every $c \in L$, may be computed in $O(n \log n)$ expected time.*

Proof. The unconstrained center c_{OPT} may be found in $O(n \log n)$ expected time [5]. L may be constructed by sorting P by distance from c_{OPT}, and retaining the first k elements, where k is the constant defined in Lemma 10.

The set C may also be constructed in $O(n \log n)$ time [5]. Then any $\Delta(c)$ may be computed by evaluating

$$\Delta(c) = \max_{(v,w) \in C} \delta(c, v, w),$$

which takes $O(n)$ time since $|C| \in O(n)$ [5]. Evaluating all $|L| = k$ dilation values this way takes $O(kn)$ time. So the expected amount of time needed to compute every $\Delta(c)$ is $O(n \log n + kn)$, which is $O(n \log n)$ by Corollary 11. $\square$

Lemma 13 *If $c \in H$, $p_i, p_j \in P$ be the pair of points such that $\delta(c, p_i, p_j) = \Delta(c)$, and $|p_i c| \geq |p_j c|$, then $|p_j c| \geq (1 - \varepsilon)|p_i c|$.*

Proof. By the assumption $c \in H$ and Corollary 11,

$$\delta(c, p_i, p_j) = \frac{|p_i c| + |p_j c|}{|p_i p_j|} \geq \Gamma;$$

by the triangle inequality

$$|p_i p_j| + |p_j c| \geq |p_i c|$$
$$|p_i p_j| \geq |p_i c| - |p_j c|,$$

so

$$\frac{|p_i c| + |p_j c|}{(|p_i c| - |p_j c|)} \geq \Gamma$$
$$|p_i c| + |p_j c| \geq \Gamma(|p_i c| - |p_j c|)$$
$$|p_j c| + \Gamma|p_j c| \geq \Gamma|p_i c| - |p_i c|$$
$$|p_j c|(\Gamma + 1) \geq |p_i c|(\Gamma - 1)$$
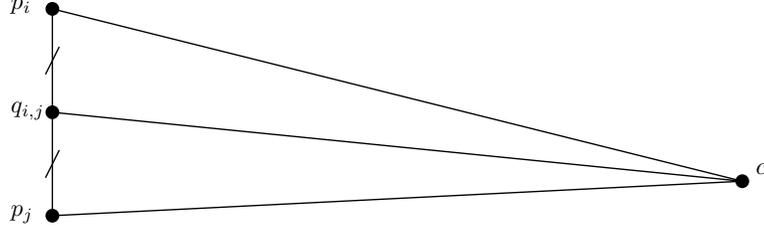$$|p_j c| \geq \frac{\Gamma - 1}{\Gamma + 1}|p_i c|.$$

By the definition of Γ,

10

Fig. 2. Example points p_i, p_j, $q_{i,j}$, and c.

$$|p_jc| \geq \frac{(2/\varepsilon - 1) - 1}{(2/\varepsilon - 1) + 1}|p_ic|$$
$$\geq \frac{2/\varepsilon - 2}{2/\varepsilon}|p_jc|$$
$$\geq (1-\varepsilon)|p_jc|.$$

$\square$

Lemma 14 *Define $q_{i,j} = \frac{1}{2}(p_i + p_j)$ and $w_{i,j} = \frac{2}{|p_ip_j|}$; then for any $c \in H$ and corresponding $p_i, p_j \in P$ such that $\delta(c, p_i, p_j) = \Delta(c)$,*

$$w_{i,j} \cdot |q_{i,j}c| \geq (1-\varepsilon) \cdot \delta(c, p_i, p_j).$$

Proof. By Lemma 13

$$|p_jc| \geq (1-\varepsilon)|p_ic|$$

and by construction $|q_{i,j}c| \geq |p_jc|$, so

$$|q_{i,j}c| \geq (1-\varepsilon)|p_ic|$$
$$2 \cdot |q_{i,j}c| \geq (1-\varepsilon)(|p_ic| + |p_ic|).$$

By assumption $|p_ic| \geq |p_jc|$, so

$$2 \cdot |q_{i,j}c| \geq (1-\varepsilon)(|p_ic| + |p_jc|)$$
$$\frac{2}{|p_ip_j|}|q_{i,j}c| \geq (1-\varepsilon)\frac{|p_ic| + |p_jc|}{|p_ip_j|},$$

and by substitution

$$w_{i,j} \cdot |q_{i,j}c| \geq (1-\varepsilon) \cdot \delta(c, p_i, p_j).$$

$\square$

11

Theorem 15 *A set of n values $\hat{\Delta}(p_i)$ subject to*

$$(1-\varepsilon)\Delta(p_i) \leq \hat{\Delta}(p_i) \leq \Delta(p_i)$$

may be computed in $O(n \log n)$ expected time.

Proof. By Lemma 12 is is possible to separate the k-low and k-high centers, and compute the exact dilation of all the k-low centers, in $O(n \log n)$ expected time. Section 4 describes a data structure that can answer approximate weighted farthest neighbor queries within a factor of $(1-\varepsilon)$ in $O(\log n)$ time after $O(n \log n)$ preprocessing. As shown in Lemma 14, the dilation function δ may be approximated for the k-high centers up to a factor of $(1-\varepsilon)$ using weights $w_{i,j} = \frac{2}{|p_i p_j|}$. So for each k-high p_i, $\Delta(p_i)$ may be approximated by the result of a weighted farthest neighbor query. Each of these $|H| = n - k$ queries takes $O(\log n)$ time, for a total expected running time of $O(n \log n)$. □

Corollary 16 *Let $\Delta_{OPT} = \min_{p_i \in P} \Delta(p_i)$. Then a point $\hat{c} \in P$ satisfying*

$$(1-\varepsilon)\Delta_{OPT} \leq \Delta(\hat{c}) \leq \Delta_{OPT}$$

may be identified in $O(n \log n)$ expected time.

Proof. Theorem 15 shows that the values $\hat{\Delta}(p_i)$ may be computed in $O(n \log n)$ expected time. A suitable $\hat{c}$ may be found by generating these n values, then searching for the smallest $\hat{\Delta}(p_i)$ and returning the corresponding p_i. □

References

1. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, number 52 in MSRI Publications, pages 1–30. Cambridge University Press, 2005.
2. S. Arya and D. Mount. Computational geometry: Proximity and location. In D. P. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 63–1–63–22. CRC Press, 2005.
3. T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proc. 20th Symp. Computational Geometry*, pages 152–159, New York, NY, USA, 2004. ACM Press.
4. C. Duncan and M. T. Goodrich. Approximate geometric query structures. In D. P. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 26–1–26–17. CRC Press, 2005.
5. D. Eppstein and K. A. Wortman. Minimum dilation stars. In *Proc. 21st Symp. Computational Geometry*, pages 321–326. ACM Press, Jun 2005.