# Fast Deterministic Approximation of Medoid in $\mathbb{R}^d$

Ovidiu Daescu* Ka Yaw Teo†

**Abstract**

For a set $P$ of $n$ points in $\mathbb{R}^d$, the *medoid* is the point in $P$ with the minimal sum of distances to $P$. We present two new deterministic algorithms for approximating the medoid of $P$ within a factor of $(1+\varepsilon)$ in time $O(n\varepsilon^{-d}\log n)$ and $O(n\varepsilon^{-d}+n\log n)$, respectively. Our results rely on a quick approximation of the sum of the distances between $P$ and any given point of $P$. Our algorithms are simple, versatile, and easily implementable.

## 1 Introduction

In this paper, we consider the following problem:

*Given a set $P$ of $n$ points in $\mathbb{R}^d$, locate a point in $P$ that minimizes the sum of the Euclidean distances between $P$ and the located point.*

The optimal point for the problem is commonly referred to as the *medoid*. One would encounter the problem of computing the medoid in various contexts such as clustering in data science [19, 21, 25], optimizing facility location in operations research [10, 11, 15, 22], and quantifying centrality in network analysis [6, 7, 16, 17, 26].

Naively, one can find the medoid of $P$ by simply computing all $\binom{n}{2}$ pairwise distances. However, it has been argued that an exact algorithm does not exist for solving the medoid problem in $o(n^2)$ time [23]. Different approaches have thus been developed to compute the medoid in sub-quadratic time either approximately or exactly under statistical assumptions.

Eppstein and Wang [13] proposed a randomized method that takes $O(n\varepsilon^{-2}\log n)$ distance computations to approximate the medoid within an additive error of $\varepsilon\mathcal{D}$ with high probability, where $\mathcal{D}$ is the diameter of $P$, which may not be known apriori. This result was later improved by Okamoto et al. [24], whose algorithm requires $O(n^{5/3}\log^{4/3} n)$ distance evaluations to return the exact medoid with high probability under certain statistical assumptions on $P$. Later on, Newling and Fleuret [23] presented an algorithm for finding the true medoid using $O(n^{3/2}2^{\Theta(d)})$ distance computations under certain assumptions on the distribution of the given points near the medoid. Soon after, a sampling-based algorithm was given by Bagaria et al. [4] for computing the exact medoid with high probability, and their algorithm takes a total of $O(n\log n)$ distance evaluations under a distributional assumption on the input points. By exploiting the underlying structure of the problem, Baharav and Tse [5] derived an improvement to Bagaria et al.'s algorithm, obtaining a gain of two to three orders of magnitude in number of distance computations.

Note that all the algorithms aforementioned have been derived in the context of network analysis, where $n$ is the number of nodes in an undirected graph, and the distance metric is the length of the shortest path between nodes. Nonetheless, the algorithms can be effectively applied to any point set under the Euclidean metric, in which case the time complexity of each said algorithm would be equal to its associated number of distance computations multiplied by a factor of $d$.

In addition, there are randomized algorithms based on coreset techniques [14, 27] capable of addressing the problem considered herein. Specifically, one can compute an $\varepsilon$-coreset of a point set $P$ in $\mathbb{R}^d$, which is a small weighted subset of $P$, such that for any point $q\in\mathbb{R}^d$, the distance sum $\sum_{p\in P}\|p-q\|$ can be approximated up to a factor of $(1+\varepsilon)$ by using the distances between $q$ and the weighted points in the coreset. A coreset of size $O(d\varepsilon^{-2})$ and $O(\mathrm{poly}(1/\varepsilon))$ can be constructed in time $O(dn+\log^2 n+d\varepsilon\log n)$ and $O(\mathrm{nnz}(A)+(n+d)\mathrm{poly}(1/\varepsilon)+\exp(\mathrm{poly}(1/\varepsilon)))$, respectively, where $\mathrm{nnz}(A)$ is the number of non-zero entries in the $n\times d$ matrix $A$ of the coordinates of $P$. As a result, one can find a $(1+\varepsilon)$-approximation to the medoid with high probability in $O(n\cdot\mathrm{poly}(1/\varepsilon))$ time.

For a set $P$ of $n$ points in $\mathbb{R}^2$, Har-Peled et al. [20] obtained an exact algorithm that computes the medoid in $O(n\log^2 n\cdot(\log n\log\log n+c_P))$ expected time, where $c_P$ is the size of the largest subset of $P$ in convex position. When the points of $P$ are located uniformly at random on the unit square, $c_P$ is bounded by $\Theta(n^{1/3})$ in expectation [2], and thus the medoid can be computed in $O(n^{4/3}\log^2 n)$ expected time.

## 2 Our results

Given an $\varepsilon>0$, a point $x$ is said to be a $(1+\varepsilon)$-approximate solution if the sum of the distances from $x$

*Department of Computer Science, University of Texas at Dallas, ovidiu.daescu@utdallas.edu

†Department of Computer Science, University of Texas at Dallas, ka.teo@utdallas.edu

to $P$ is at most $(1+\varepsilon)$ times the sum of the distances from the true medoid. Throughout the paper, we assume that $d$ and $\varepsilon$ are fixed constants independent of $n$; nevertheless, we include them in some of the asymptotic results to indicate their dependencies. In addition, we assume, without loss of generality, that $P$ has been scaled so that it is enclosed within a unit hypercube.

We begin in Section 3 by describing an algorithm to compute a $(1+\varepsilon)$-approximate medoid in time $O(n\varepsilon^{-d}\log n)$. This algorithm uses a new data structure named well separated subset decomposition (WSSD), which extends on the classical idea of the well separated pair decomposition (WSPD) by Callahan [9]. WSSD partitions $P$ into $O(\log n)$ clusters that preserve the distances of $P$ to each of the $n$ candidate points.

In Section 4, we encode the pairwise distances between the points of $P$ by directly using WSPD. We can then estimate the medoid within a factor of $(1+\varepsilon)$ in time $O(n\varepsilon^{-d}+n\log n)$, provided that the pairwise distances associated with the well separated pairs are computed and summed in the right order. If $\varepsilon^{-d}=O(\log n)$, our algorithm would run in $O(n\log n)$ time.

To the best of our knowledge, all the previous approximation methods for solving the medoid problem are randomized, making our algorithms the first *deterministic* fully polynomial-time approximation schemes (FPTASs) with a time complexity near-linear in $n$.

## 3 $O(n\varepsilon^{-d}\log n)$-time $(1+\varepsilon)$-approximation

We propose an $O(n\varepsilon^{-d}\log n)$-time approximation algorithm involving the following partitioning scheme.

### Well separated subset decomposition (WSSD)

Let $C$ denote a subset of $P$. Define $s>0$ to be a parameter called separation factor. With respect to a candidate point $p\in P$, for some $r\in\mathbb{R}_{\geq 0}$, if the points of $C$ can be enclosed within a Euclidean ball of radius $r$ such that the closest distance from this ball to $p$ is at least $sr$, then $C$ is said to be $s$-well separated from $p$.

**Definition 1 (WSSD)** *Given a set $P$ of $n$ points, a point $p$, and a separation factor $s>0$, an $s$-well separated subset decomposition ($s$-WSSD) with respect to $p$ is defined as a collection of subsets of $P$, denoted by $\{C_1,C_2,\ldots,C_k\}$, such that (I) $C_i\subseteq P$ for $1\leq i\leq k$, (II) $C_i\cap C_j=\emptyset$ for $1\leq i,j\leq k$ and $i\neq j$, (III) $\cup_{i=1}^{k}C_i=P$, and (IV) $C_i$ is $s$-well separated from $p$ for $1\leq i\leq k$.*

An $s$-WSSD can be constructed from either a kd-tree [8] or a balanced box decomposition (BBD) tree [3]. Both of these data structures are based on a hierarchical subdivision of space into rectilinear regions called cells. The size of a cell is given by the length of its longest side. For a set $P$ of $n$ points in $\mathbb{R}^d$, it is possible to build, in time $O(n\log n)$, an optimized kd-tree [18] or a BBD-tree with height $O(\log n)$ and space $O(n)$. In either tree, each internal node has two children, and each leaf node contains a single point. Unlike a kd-tree, the cells of a BBD-tree have a bounded aspect ratio, and the sizes of the cells decrease by (at least) a factor of $1/2$ with each descent of $2d$ levels in the tree.

**Theorem 1** *For a set $P$ of $n$ points and any $s>0$, with respect to a point $p$, one can construct an $s$-WSSD of size $O(s^d\log n)$ in time $O(n\log n+s^d\log n)$.*

**Proof.** We begin by building a kd-tree or a BBD-tree for $P$. Each leaf node, which contains a single point, is treated as having an infinitesimally small cell containing its point.

The construction of an $s$-WSSD, with respect to a point $p$, is based on a recursive process. Throughout the construction, we maintain a collection of sets that satisfy properties (I), (II), and (III) as stated in Definition 1. When the procedure terminates, all the sets generated will fulfill property (IV). Each set of the $s$-WSSD will be encoded as a node in the kd-tree or BBD-tree.

Let $u$ denote a node in either tree. Consider the smallest Euclidean ball that encloses the cell of node $u$. If the ball is $s$-well separated from point $p$, then we report node $u$ as an $s$-well separated subset. Otherwise, we apply the procedure recursively to each child node of $u$. Let $WSSD(u,p,s)$ denote said procedure.

Note that we can determine whether a node $u$ (i.e., its smallest enclosing Euclidean ball) is $s$-well separated from point $p$ in $O(1)$ time. This requires computing the smallest Euclidean ball enclosing the cell of node $u$, either at the time of determining the separation between node $u$ and point $p$ or in advance (when creating the tree data structure).

In the procedure $WSSD()$, we divide a node $u$ only if the call $WSSD(u,p,s)$ is non-terminal – that is, node $u$ is not an $s$-well separated subset. Each non-terminal call generates at most two recursive calls, through which a terminal call may arise. Note that each terminal call produces at most one well separated subset. Thus, the total number of well separated subsets is at most two times the number of non-terminal calls.

To evaluate the number of $s$-well separated subsets generated in the recursive process, we use a packing argument (to count the number of non-terminal calls), which slightly differs depending on either an optimized kd-tree or a BBD-tree is used as the basis for the construction of the $s$-WSSD.

**kd-tree-based $s$-WSSD.** Each of the nodes at a given level in an optimized kd-tree is associated with (nearly) the same number of points (which is a result of choosing the median as the cutting value). Consider

the nodes at a given level $\lambda$ of the kd-tree, where the cell associated with each node contains $b_\lambda$ points. Let $V$ denote the volume of the cell associated with any node at level $\lambda$ in the kd-tree. As shown by Friedman et al. [18], the expected volume of each such cell is approximately $E[V] = \frac{b_\lambda}{n+1} \cdot \frac{1}{P_\lambda}$, where $P_\lambda$ is the probability density averaged over the cell (assuming that the probability distribution of the points within the cell is approximately constant). Let $\alpha$ be the size of the cell. Then, the expected size $E[\alpha]$ of the cell is simply the $d$-th root of the expected volume of the cell – that is, $E[\alpha] = E[V]^{1/d}$. The expected number of nodes at level $\lambda$ being divided in the procedure must be bounded from above by the expected number of cells at level $\lambda$ overlapping the ball of radius $sr$ centered at $p$ – that is, $(1 + \lceil 2sr/E[\alpha] \rceil)^d$. Given that $r = E[\alpha]\sqrt{d}/2$, the upper bound becomes $(1 + s\sqrt{d})^d = O(s^d)$. Since there are $O(\log n)$ levels in the kd-tree, the expected number of non-terminal calls to the procedure *WSSD*() is $O(s^d \log n)$. Hence, the expected number of $s$-well separated subsets is $2 \cdot O(s^d \log n) = O(s^d \log n)$.

**BBD-tree-based $s$-WSSD.** For the case of a BBD-tree, we use a similar packing argument as that for a kd-tree. Recall that point set $P$ has been scaled so that it is enclosed within a unit hypercube. As a result, the cells of the BBD-tree have sizes that are powers of 1/2.

For analysis purposes, we congregate the nodes in the BBD-tree into groups according to the sizes of their associated cells. Define size group $i$ to be the set of nodes whose cell size is $1/2^i$. Note that a node and its child may have the same size, and thus we cannot apply the packing argument directly to each size group. Define base group $i$ to be the subset of nodes in size group $i$ that are leaf nodes or whose children belong to the next smaller size group. The cells corresponding to the nodes in a base group are pairwise interior-disjoint. For each base group $i$, the number of cells overlapping the ball of radius $sr$ centered at $p$ is bounded from above by $\left(1 + \left\lceil 2sr/\left(1/2^i\right)\right\rceil\right)^d$. Since $r = \left(1/2^i\right)\sqrt{d}/2$, the upper bound becomes $O(s^d)$. Note that at most $2d$ levels of ancestors above the nodes in the base group can be in the same size group. In addition, the BBD-tree is $O(\log n)$ in height, which implies that the total number of base groups is bounded by $O(\log n)$. So, the total number of non-terminal calls to *WSSD*() is $O(2d \cdot s^d \log n) = O(s^d \log n)$. As a result, the total number of $s$-well separated subsets generated with respect to point $p$ is $2 \cdot O(s^d \log n) = O(s^d \log n)$.

In both cases above, the asymptotic upper bound on the number of $s$-well separated subsets generated is $O(s^d \log n)$, with the distinction that the upper bound applies to the worst case for the BBD-tree, whereas the upper bound is derived with respect to the average (expected) case for the kd-tree. Together with $O(n \log n)$ time to build either tree, the overall running time is $O(n \log n + s^d \log n)$. □

We now describe a technical lemma associated with an $s$-WSSD, which will be used later in approximating the medoid.

**Lemma 2 (WSSD Utility Lemma)** *If subset $C$ is $s$-well separated from point $p$, and $c, c' \in C$, then we have $\|c' - p\| \leq \left(1 + \frac{2}{s}\right) \|c - p\|$.*

**Proof.** Due to the triangle inequality, we have $\|c' - p\| \leq \|c - p\| + \|c - c'\|$. Since $C$ is enclosed within a ball of radius $r$ and is $s$-well separated from $p$, we have $\|c' - p\| \leq \|c - p\| + 2r = \|c - p\| + \frac{2r}{sr} sr \leq \|c - p\| + \frac{2}{s}\|c - p\| = \left(1 + \frac{2}{s}\right) \|c - p\|$. □

### WSSD-based approximation

This section discusses the usage of a WSSD for approximating the medoid of $P$. We present the arguments only for the WSSD constructed from a BBD-tree, since the analysis is similar for the case of using a kd-tree, aside from that the resulting time complexity would be of the average case instead of the worst case.

**Theorem 3** *Given a set $P$ of $n$ points in $\mathbb{R}^d$, for any $\varepsilon > 0$, a $(1 + \varepsilon)$-approximation to the medoid of $P$ can be computed in time $O(n\varepsilon^{-d} \log n)$.*

**Proof.** First, we build a BBD-tree for $P$, using which we construct an $s$-WSSD with respect to each of the $n$ candidate points in $P$. According to Theorem 1, the total construction time is bounded by $O(n \log n + ns^d \log n) = O(ns^d \log n)$. We make a small augmentation to the construction of the WSSD as follows. When building a BBD-tree, we associate each node $u$ of the tree with a quantity $|u|$ indicating the number of points lying within its cell. When we output a node $u$ as an $s$-well separated subset with respect to a point $p$ in the decomposition process, we report $|u|$ and the farthest point within the cell of node $u$ from $p$ (which may not necessarily be a point of $P$). Since the farthest point within a hypercube from $p$ is one of the $2^d$ vertices of the hypercube, we can find the farthest point in $O(2^d)$ time, which is just $O(1)$ given that $d$ is treated as a constant. Thus, the overall running time for the construction of the WSSD remains the same as before.

Let $\{C_i \mid 1 \leq i \leq k_p\}$ be the collection of $s$-well separated subsets with respect to a point $p$. Let $\phi(C_i)$ denote the farthest point within the cell containing $C_i$ from point $p$, and let $|C_i|$ be the number of points in $C_i$. With respect to each candidate point $p \in P$, we compute the distance sum $\sum_i |C_i| \cdot \|\phi(C_i) - p\|$, and output the point $p$ achieving the smallest distance sum.

Suppose that the aforementioned approach yields point $x$ as an approximate medoid for $P$. Let $\{A_i \mid 1 \leq i \leq k_x\}$ be the set of $s$-well separated subsets with respect to $x$. Then, by Lemma 2, for each $s$-well separated subset $A_i$, we have $|A_i| \cdot \|\phi(A_i) - x\| \leq \sum_{a \in A_i} \left(1 + \frac{2}{s}\right) \|a - x\|$. Since $\phi(A_i)$ is the farthest point within the cell containing $A_i$ from $x$, by summing over all $i$, we obtain

$$\sum_{p \in P} \|p - x\| \leq \sum_i |A_i| \cdot \|\phi(A_i) - x\| \leq \left(1 + \frac{2}{s}\right) \sum_{p \in P} \|p - x\|$$

Let $m$ denote the exact medoid of $P$. Let $\{B_i : 1 \leq i \leq k_m\}$ be the set of $s$-well separated subsets with respect to $m$. We then have

$$\begin{aligned}\sum_{p \in P} \|p - m\| \leq \sum_{p \in P} \|p - x\| &\leq \sum_i |A_i| \cdot \|\phi(A_i) - x\| \\ &\leq \sum_i |B_i| \cdot \|\phi(B_i) - m\| \\ &\leq \left(1 + \frac{2}{s}\right) \sum_{p \in P} \|p - m\|\end{aligned}$$

Given any $\varepsilon > 0$, we set $s = 2/\varepsilon$. Then, we obtain

$$\sum_{p \in P} \|p - m\| \leq \sum_{p \in P} \|p - x\| \leq (1 + \varepsilon) \sum_{p \in P} \|p - m\|$$

This implies that the output point $x$ is a $(1 + \varepsilon)$-approximation to the medoid of $P$. The overall running time is bounded by $O(n(2/\varepsilon)^d \log n) = O(n\varepsilon^{-d} \log n)$. □

## 4 $O(n\varepsilon^{-d} + n \log n)$-time $(1 + \varepsilon)$-approximation

In this section, we derive an algorithm for computing a $(1 + \varepsilon)$-approximation to the medoid of $P$ in $O(n\varepsilon^{-d} + n \log n)$ time. First, we use a WSPD to represent the distances between the points of $P$. After obtaining such a representation, we carefully enumerate the pairwise distances in an order such that the sum of the distances from $P$ to each representative point is approximated correctly.

### Well separated pair decomposition (WSPD)

A WSPD [9] is formally defined as follows. Let $A$ and $B$ be subsets of $P$. Define $s > 0$ to be a separation factor. Denote by $r$ the smallest radius of a Euclidean ball such that each of $A$ and $B$ can be enclosed within such a ball. Set $A$ is said to be $s$-well separated from $B$ if the closest distance between the two balls enclosing $A$ and $B$ is at least $sr$.

**Definition 2 (WSPD)** *For a set $P$ of $n$ points and a separation factor $s > 0$, an $s$-well separated pair decomposition ($s$-WSPD) is a collection of pairs of subsets of $P$, denoted as $\{\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}\}$, such that (I) $A_i, B_i \subseteq P$ for $1 \leq i \leq k$, (II) $A_i \cap B_i = \emptyset$ for $1 \leq i \leq k$, (III) $\cup_{i=1}^{k} A_i \bigotimes B_i = P \bigotimes P$, and (IV) $A_i$ and $B_i$ are $s$-well separated for $1 \leq i \leq k$.*

When estimating the medoid of $P$, we will make use of the following utility property of an $s$-WSPD.

**Lemma 4 (WSPD Utility Lemma)** *If pair $\{A, B\}$ is $s$-well separated, $a, a' \in A$, and $b \in B$, then we have $\|a' - b\| \leq \left(1 + \frac{2}{s}\right) \|a - b\|$.*

**Proof.** The proof is similar to that of Lemma 2 and thus omitted. □

### WSPD-based approximation

**Theorem 5** *Given a set $P$ of $n$ points in $\mathbb{R}^d$, for any $\varepsilon > 0$, one can compute a $(1 + \varepsilon)$-approximation to the medoid of $P$ in $O(n\varepsilon^{-d} + n \log n)$ time.*

**Proof.** We begin by building a compressed octree for $P$. The octree can be built in $O(n \log n)$ time, and is of size $O(n)$ [1, 12]. For simplicity of arguments, we assume that the octree is not compressed but of size $O(n)$. This allows us to assume that nodes of the same level in the octree have the same cell size. By using the octree, we construct a WSPD for $P$ such that each well separated pair of nodes generated are of the same level in the octree. This requires a slight modification to the original algorithm given in [9] for creating a WSPD. Namely, when we fail to separate a pair of nodes $u$ and $v$, we proceed to recursively separate the $2^d$ children of $u$ from those of $v$, thus keeping the invariant that each pair of nodes considered are of the same size. The algorithm is presented as a pseudocode in Figure 1 (where the code in blue is an augmentation necessary for finding an approximate medoid, which will be discussed later). The initial call is $WSPD(u_0, u_0, s, \emptyset)$, where $u_0$ is the root of the octree.

To evaluate the total number of well separated pairs in the resulting WSPD, it suffices to count the number of terminal calls to $WSPD()$, each of which can generate $\Theta(2^{2d})$ well separated pairs. Since a terminal call may only arise as a call to $WSPD()$ in a non-terminal call, we instead bound the number of calls to $WSPD()$ made by all the non-terminal calls. We claim that, in any non-terminal call, for every node $u_i$ (iterated in the first outer for loops of the algorithm), the number of calls to $WSPD()$ (as in the final for loop in the algorithm) is bounded by $O(s^d)$. Since there are $O(n)$ nodes in the (compressed) octree, the total number of calls to $WSPD()$ is $O(s^d n)$.

We are now left to establish the claim that for any node $u_i$, the number of calls to $WSPD()$ is bounded by $O(s^d)$. For a node $u_i$, a call to $WSPD()$ is made only if $u_i$ is not $s$-well separated from some node $v_j$. Let $\alpha$

```
Algorithm WSPD(u, v, s, par)
1.  let u_1, ..., u_α be the children of u;
2.  let v_1, ..., v_β be the children of v;
3.  for i ← 1 to α
4.      do S ← T ← ∅;
5.         newpar ← par;
6.         for j ← 1 to β
7.             do if (u_i or v_j is empty) then ignore (u_i, v_j);
8.                    else if (u_i and v_j are leaves and u_i = v_j) then ignore (u_i, v_j);
9.                           else if (u_i and v_j are s-well separated)
10.                                then add (u_i, v_j) to S;
11.                                else add (u_i, v_j) to T;
12.                if (S ≠ ∅) then newpar ← u_i;
13.                for each (x, y) ∈ S do output (x, y, par);
14.                for each (x, y) ∈ T do call WSPD(x, y, s, newpar);
```

Figure 1: Augmented algorithm for constructing WSPD.

denote the side length of the cells of nodes $u_i$ and $v_j$. Let $r$ be the radius of the Euclidean ball enclosing each of the cells of nodes $u_i$ and $v_j$. Note that $r = \alpha\sqrt{d}/2$. Assume that $s \geq 1$; if $0 < s < 1$, we replace $s$ with $\max(s, 1)$. Let $c_{u_i}$ and $c_{v_j}$ be the centers of the balls enclosing the cells of nodes $u_i$ and $v_j$, respectively. Since $u_i$ is not $s$-well separated from $v_j$, the distance between $c_{u_i}$ and $c_{v_j}$ must be at most $2r + sr \leq 3sr$. Let $\beta$ denote the ball of radius $3sr$ centered at $c_{u_i}$. The set of nodes $v_j$ that are not $s$-well separated from $u_i$ must correspond to the cells of side length $\alpha$ overlapping $\beta$. Using a similar packing argument as in the proof of Theorem 1, for a node $u_i$, the number of such nodes $v_j$ is bounded by $O(s^d)$.

Finally, together with $O(n \log n)$ time to build the octree, the overall time for constructing the WSPD is $O(n \log n + s^d n)$.

For the convenience of the ensuing discussion, each well separated pair $(u, v)$ is represented (and produced by the algorithm *WSPD*()) as an ordered pair, where $u$ is referred to as the *anchor set* of the pair.

**Augmenting WSPD construction.** In the octree used for constructing the WSPD, we associate each node $u$ with i) a representative point $rep(u)$, which may be chosen arbitrarily among the points lying inside the cell of node $u$, and ii) a quantity $|u|$ indicating the number of points located within the cell of node $u$. In addition, we output each well separated pair $(u, v)$ along with a set $w$, if any, where $w$ is the lowest ancestor of $u$ such that $w$ is $s$-well separated from some node of the same level as $w$ (in the octree). We call $w$ the *parent set* of $(u, v)$ (and of $u$), and $u$ a *child set* of $w$. In the pseudocode *WSPD*(), variables *par* and *newpar* (i.e., code in blue) are used for keeping track of the parent set for each well separated pair.

**Finding an approximate medoid.** Let $\Gamma$ denote the set of well separated pairs in an $s$-WSPD for $P$. Each ordered pair of points $(p_i, q_i) \in P \times P$ (where $p_i \neq q_i$) occurs in a unique well separated pair $(A, B)$ in $\Gamma$. As a result, we could simply compute the approximate distance sum for each point $p_i \in P$ using $\Gamma$, and return the point with the minimum distance sum as the approximate solution.

To take a closer look at this idea, let $e_i$ denote any anchor set (of any well separated pair in $\Gamma$) that has no child anchor set. Recall that $rep(e_i)$ denotes the representative point associated with $e_i$. Note that a point $p_i \in P$ must belong to a (unique) childless anchor set $e_i$, for which $p_i$ may or may not be chosen as the representatives point.

Suppose that $p_i = rep(e_i)$. Let $S_i \subseteq \Gamma$ be the subset of well separated pairs, of which $e_i$ is either the anchor set or a child anchor set. We can obtain an approximate distance sum for point $rep(e_i)$ using $S_i$. Namely, for each pair $(A, B) \in S_i$, we compute $|B| \cdot \|rep(B) - rep(e_i)\|$. We then take the sum over all the pairs in $S_i$ to be the approximate distance sum for $p_i$. If $p_i \neq rep(e_i)$, then the approximate distance sum for $p_i$ only differs by at most a factor of $\left(1 + \frac{2}{s}\right)$ from that for $rep(e_i)$, according to Lemma 4.

Hence, we only need to compute the distance sum for each representative point associated with a childless anchor set (which will just be referred to as *representative points* for simplicity hereafter). However, observe that, for any two childless anchor sets $e_i$ and $e_j$, $S_i \cap S_j$ may not be empty. That is, there could be some well separated pair $(A, B) \in S_i \cap S_j$ such that $e_i \subseteq A$ and $e_j \subseteq A$. In other words, $e_i$ and $e_j$ could have some common ancestor anchor set. This implies, by computing the distance sum for each representative point one at a time, that the running time required to find the representative point with the minimum distance sum could

**Algorithm** *Approx-Medoid*($\Gamma$)
1. $F \leftarrow \emptyset$;
2. **for** each $\mathsf{C}_i \in \mathsf{C}$ (in increasing order)
3. **do for** each $\mathsf{P}_{ij} \in \mathsf{P}_i$
4. **do** let $w$ be the parent set of the well separated pairs in $\mathsf{P}_{ij}$;
5. **for** each $\mathsf{A}_{ijk} \in \mathsf{A}_{ij}$
6. **do** let $u$ be the anchor set of the well separated pairs in $\mathsf{A}_{ijk}$;
7. compute $\sigma[u]$ += $\sum_{(u,v)\in \mathsf{A}_{ijk}} (|v| \cdot \|rep(v) - rep(u)\|)$;
8. **if** $(w = \emptyset)$ **then** add $(rep(u), \sigma[u])$ to $F$;
9. **if** $(w \neq \emptyset)$
10. **then** let $A = \{u \mid (u, v) \in \mathsf{P}_{ij}\}$;
11. $u_{\min} \leftarrow \arg\min_{u\in A} \sigma[u]$;
12. $\sigma[w] \leftarrow \sigma[u_{\min}]$;
13. $rep(w) \leftarrow rep(u_{\min})$;
14. output $(x, \sigma) \in F$ with the minimum $\sigma$;

Figure 2: Algorithm for approximating medoid using WSPD.

be $\Omega(s^d n)$.

As it turns out, we do not have to compute the distance sums for all the representative points. If we compute and sum the distances for each representative point at each level in a bottom-up fashion, allow the representative point with the minimum "partial" distance sum thus far at each level to overtake the others with the same parent anchor set, then we could find the representative point with the minimum distance sum in $O(s^d n)$ time (since each well separated pair is only used once for distance sum computation).

Here are the details of the procedure. We group the well separated pairs in $\Gamma$ according to the cell sizes of their corresponding node pairs in the octree. Within each cell-size group, we collect the well separated pairs into groups with common parent sets. Within each such parent-set group, we further congregate the well separated pairs according to their anchor sets. Formally, for a given well separated pair $(u, v)$, we denote by $cel(u, v)$ and $par(u, v)$ the cell size and the parent set of $(u, v)$, respectively. Note that if a well separated pair $(u, v)$ has no parent set, then $par(u, v) = \emptyset$. Define:

i) $\mathsf{C} = \{\mathsf{C}_i \subseteq \Gamma \mid \forall (u, v), (x, y) \in \Gamma,\ cel(u, v) = cel(x, y) \implies (u, v) \in \mathsf{C}_i \wedge (x, y) \in \mathsf{C}_i\}$,

ii) $\mathsf{P}_i = \{\mathsf{P}_{ij} \subseteq \mathsf{C}_i \mid \forall (u, v), (x, y) \in \mathsf{C}_i,\ par(u, v) = par(x, y) \implies (u, v) \in \mathsf{P}_{ij} \wedge (x, y) \in \mathsf{P}_{ij}\}$, and

iii) $\mathsf{A}_{ij} = \{\mathsf{A}_{ijk} \subseteq \mathsf{P}_{ij} \mid \forall (u, v), (x, y) \in \mathsf{P}_{ij},\ u = x \implies (u, v) \in \mathsf{A}_{ijk} \wedge (x, y) \in \mathsf{A}_{ijk}\}$.

That is, $\mathsf{C} = \{\mathsf{C}_i \mid i = 1, 2, \ldots\}$ is the partition of set $\Gamma$ according to cell size, $\mathsf{P}_i = \{\mathsf{P}_{ij} \mid j = 1, 2, \ldots\}$ is the partition of set $\mathsf{C}_i \in \mathsf{C}$ according to parent set, and $\mathsf{A}_{ij} = \{\mathsf{A}_{ijk} \mid k = 1, 2, \ldots\}$ is the partition of set $\mathsf{P}_{ij} \in \mathsf{P}_i$ according to anchor set. Assume, without loss of generality, that for any $\mathsf{C}_i, \mathsf{C}_j \in \mathsf{C}$, if $i < j$, then $cel(u, v) < cel(x, y)$ for all $(u, v) \in \mathsf{C}_i$ and $(x, y) \in \mathsf{C}_j$. For any anchor set $u$, let $\sigma[u]$ denote the distance sum computed for $u$. Initially, we set $\sigma[u] = 0$ for all anchor sets $u$. We then process $\Gamma$ as described in the pseudocode given in Figure 2.

Briefly, we iterate the well separated pairs by cell size in ascending order. Within each cell-size group $\mathsf{C}_i \in \mathsf{C}$, we update $\sigma[u]$ for each anchor set $u$ sharing the same parent set $w$ by considering its associated well separated pairs (line 6 of the pseudocode). If $w \neq \emptyset$, then we find, among those having the same parent set $w$, the anchor set $u_{\min}$ with the minimum distance sum after the update. We record the distance sum for $u_{\min}$ as that for its parent set $w$, and replace the representative point for the parent set $w$ of $u_{\min}$ with that for $u_{\min}$. When the algorithm terminates, of all anchor sets without a parent set, we report the one with the minimum distance sum.

The time complexity of *Approx-Medoid*() is bounded by $O(s^d n)$, given that each well separated pair is processed by a constant number of operations in the procedure. Along with the construction time for WSPD, the overall time for approximating the medoid of $P$ is $O(n \log n + s^d n)$.

**Correctness of algorithm.** We now proceed to prove the correctness of the algorithm *Approx-Medoid*() to yield a solution within a multiplicative error of $\varepsilon$.

Let $m$ be the exact medoid of $P$. Let $x$ be the representative point returned as the approximate solution by the algorithm *Approx-Medoid*(). To establish the correctness of the algorithm, we have to show that

$$\sum_{p\in P} \|p - m\| \leq \sum_{p\in P} \|p - x\| \leq (1 + \varepsilon) \sum_{p\in P} \|p - m\|$$

The first inequality holds because no other point in $P$ can have a smaller distance sum than the exact medoid

$m$. To prove the second inequality, consider the anchor set $U$ such that i) $m \in U \wedge x \in U$ and ii) $m \notin U' \wedge x \notin U'$ for all child sets $U'$ of $U$.

First, we examine the set of distance computations for the levels above that of $U$. Let $\Lambda = \{(U_i'', V_i) \mid i = 1, 2, \dots\}$ denote the set of all well separated pairs such that for each pair $(U_i'', V_i) \in \Lambda$, $U_i''$ is an ancestor set of $U$. For each well separated pair $(U_i'', V_i) \in \Lambda$, according to Lemma 4, we have $|V_i| \cdot \|rep(V_i) - x\| \leq \left(1 + \frac{2}{s}\right) |V_i| \cdot \|rep(V_i) - m\|$. If we sum over all the pairs in $\Lambda$, we then have

$$\sum_i |V_i| \cdot \|rep(V_i) - x\| \leq \left(1 + \frac{2}{s}\right) \sum_i |V_i| \cdot \|rep(V_i) - m\|$$

Secondly, we examine the distance computations for the levels below that of $U$. For any anchor set $C$, let $\sigma[C, c]$ denote the distance sum computed for $C$ in the algorithm, where $c \in C$ is the representative point used in computing the distance sum. Let $M$ be the child set of $U$ such that $m \in M$. Similarly, let $X$ denote the child set of $U$ such that $x \in X$. Recall that $M \cap X = \emptyset$.

Let $M'$ be the lowest descendant set of $M$ such that $m \in M'$. Let $m'$ be the representative point associated with $M'$. If $m' = m$, then the distance sum computed for $M'$ is $\sigma[M', m'] = \sigma[M', m]$. Otherwise, according to Lemma 4, we have $\sigma[M', m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M', m]$.

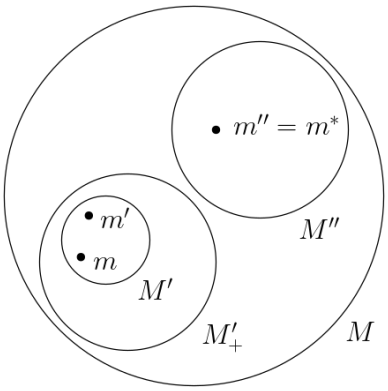


Figure 3: Point $m'' = m^*$ overtakes $m'$ in the distance sum computation during the execution of the algorithm *Approx-Medoid*() such that $\sigma[M, m^*] \leq \sigma[M, m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M, m]$.

Let $m^* \in M$ be the representative point used (in the algorithm) to compute the distance sum for anchor set $M$. Since $x$ is the representative point produced by the algorithm as the solution, we must have $\sigma[X, x] \leq \sigma[M, m^*]$. If $m^* = m'$, then $\sigma[X, x] \leq \sigma[M, m^*] = \sigma[M, m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M, m]$. Otherwise, at some level between that of $M$ and $M'$, $m'$ must be "overtaken" by some other point $m''$ such that i) $m''$ is the representative of some anchor set $M''$, and ii) $\sigma[M'', m''] \leq \sigma[M'_+, m']$, where $M'_+$ is an ancestor set of $M'$ and of the same level as $M''$ (see Figure 3 for an illustration). Clearly, this sort of "overtaking" could happen multiple times as we ascend the levels from that of $M'$ to $M$ in the algorithm. At the end of the ascension, $m^*$ prevails, and we have $\sigma[X, x] \leq \sigma[M, m^*] \leq \sigma[M, m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M, m]$.

As the algorithm terminates, $(x, \sigma)$ is yielded as the approximate solution, where

$$\sigma = \sum_i |V_i| \cdot \|rep(V_i) - x\| + \sigma[X, x]$$

is the minimum distance sum reported along with point $x$. By applying Lemma 4, we have

$$\begin{aligned}
&\sum_{p \in P} \|p - x\| \\
&\leq \left(1 + \frac{2}{s}\right) \left(\sum_i |V_i| \cdot \|rep(V_i) - x\| + \sigma[X, x]\right) \\
&\leq \left(1 + \frac{2}{s}\right)^2 \left(\sum_i |V_i| \cdot \|rep(V_i) - m\| + \sigma[M, m]\right) \\
&\leq \left(1 + \frac{2}{s}\right)^3 \sum_{p \in P} \|p - m\| \\
&= \left(1 + \frac{6}{s} + \frac{12}{s^2} + \frac{8}{s^3}\right) \sum_{p \in P} \|p - m\|
\end{aligned}$$

Since $s = \max(s, 1)$, we obtain

$$\sum_{p \in P} \|p - x\| \leq \left(1 + \frac{6}{s} + \frac{20}{s^2}\right) \sum_{p \in P} \|p - m\|$$

Given an $\varepsilon > 0$, if we set $s = \frac{3 + \sqrt{9 + 20\varepsilon}}{\varepsilon}$, then we have

$$\sum_{p \in P} \|p - x\| \leq (1 + \varepsilon) \sum_{p \in P} \|p - m\|$$

□

## 5 Conclusion

We have presented two deterministic, near-linear time algorithms for approximating the medoid of a point set in fixed dimensions within a factor of $(1 + \varepsilon)$. In the future, we propose to further explore the idea of pair decompositions for solving minsum location-based optimization problems involving more complex geometric objects.

## References

[1] S. Aluru. Quadtrees and octrees. In *Handbook of Data Structures and Applications*, pages 309–326. Chapman and Hall/CRC, 2018.

[2] G. Ambrus and I. Bárány. Longest convex chains. *Random Structures & Algorithms*, 35(2):137–162, 2009.

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

[4] V. Bagaria, G. Kamath, V. Ntranos, M. Zhang, and D. Tse. Medoids in almost-linear time via multi-armed bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 500–509, 2018.

[5] T. Z. Baharav and D. Tse. Ultra fast medoid identification via correlated sequential halving. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 3655–3664, 2019.

[6] A. Bavelas. Communication patterns in task-oriented groups. *Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

[7] M. A. Beauchamp. An improved index of centrality. *Behavioral Science*, 10(2):161–163, 1965.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[9] P. B. Callahan. *Dealing with higher dimensions: The well-separated pair decomposition and its applications*. PhD thesis, The Johns Hopkins University, 1995.

[10] Z. Drezner. *Facility location: A survey of applications and methods*. Springer Series in Operations, 1995.

[11] Z. Drezner and H. W. Hamacher. *Facility location: Applications and theory*. Springer Science & Business Media, 2004.

[12] D. Eppstein, M. T. Goodrich, and J. Z. Sun. Skip quadtrees: Dynamic data structures for multidimensional point sets. *International Journal of Computational Geometry & Applications*, 18(01n02):131–160, 2008.

[13] D. Eppstein and J. Wang. Fast approximation of centrality. *Journal of Graph Algorithms and Applications*, 8(1):39–45, 2004.

[14] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 569–578, 2011.

[15] R. L. Francis, L. F. McGinnis, and J. A. White. *Facility layout and location: An analytical approach*. Pearson College Division, 1992.

[16] L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.

[17] N. E. Friedkin. Theoretical foundations for centrality measures. *American journal of Sociology*, 96(6):1478–1504, 1991.

[18] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[19] J. Han, M. Kamber, and A. K. H. Tung. Spatial clustering methods in data mining: A survey. *Geographic data mining and knowledge discovery*, pages 188–217, 2001.

[20] S. Har-Peled, M. Jones, and S. Rahul. Active-learning a convex body in low dimensions. *Algorithmica*, 83(6):1885–1917, 2021.

[21] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. John Wiley & Sons, 1990.

[22] P. B. Mirchandani and R. L. Francis. *Discrete location theory*. Wiley-Interscience, 1990.

[23] J. Newling and F. Fleuret. A Sub-Quadratic Exact Medoid Algorithm. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 185–193, 2017.

[24] K. Okamoto, W. Chen, and X. Y. Li. Ranking of closeness centrality for large-scale social networks. In *International Workshop on Frontiers in Algorithmics*, pages 186–195, 2008.

[25] H. S. Park and C. H. Jun. A simple and fast algorithm for $k$-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

[26] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[27] C. Sohler and D. P. Woodruff. Strong coresets for $k$-median and subspace approximation: Goodbye dimension. In *Proceedings of the 59th Annual Symposium on Foundations of Computer Science*, pages 802–813, 2018.