A Sub-Quadratic Exact Medoid Algorithm

James Newling
Idiap Research Institute & EPFL

François Fleuret
Idiap Research Institute & EPFL

Abstract

We present a new algorithm `trimed` for obtaining the *medoid* of a set, that is the element of the set which minimises the mean distance to all other elements. The algorithm is shown to have, under certain assumptions, expected run time $O(N^{\frac{3}{2}})$ in $\mathbb{R}^d$ where N is the set size, making it the first sub-quadratic exact medoid algorithm for $d > 1$. Experiments show that it performs very well on spatial network data, frequently requiring two orders of magnitude fewer distance calculations than state-of-the-art approximate algorithms. As an application, we show how `trimed` can be used as a component in an accelerated K-medoids algorithm, and then how it can be relaxed to obtain further computational gains with only a minor loss in cluster quality.

1 Introduction

A popular measure of the centrality of an element of a set is its mean distance to all other elements. In network analysis, this measure is referred to as *closeness centrality*, we will refer to it as *energy*. Given a set $\mathcal{S} = \{x(1), \dots, x(N)\}$ the energy of element $i \in \{1, \dots, N\}$ is thus given by,

$$E(i) = \frac{1}{N-1} \sum_{j \in \{1,\dots,N\} \setminus \{i\}} \text{dist}(x(i), x(j)).$$

An element in $\mathcal{S}$ with minimum energy is referred to as a *1-median* or a *medoid*. Without loss of generality, we will assume that $\mathcal{S}$ contains a unique medoid. The problem of determining the medoid of a set arises in the contexts of clustering, operations research, and network analysis. In clustering, the Voronoi iteration K-medoids algorithm (Hastie et al., 2001; Park and Jun, 2009) requires determining the medoid of each of K clusters at each iteration. In operations research, the facility location problem requires placing one or several facilities so as to minimise the cost of connecting to clients. In network analysis, the medoid may represent an influential person in a social network, or the most central station in a rail network.

1.1 Medoid algorithms and our contribution

A simple algorithm for obtaining the medoid of a set of N elements computes the energy of all elements and selects the one with minimum energy, requiring $\Theta(N^2)$ time. In certain settings $\Theta(N)$ algorithms exist, such as in 1-D where the problem is solved by Quickselect (Hoare, 1961), and more generally on trees. However, no general purpose $o(N^2)$ algorithm exists. An example illustrating the impossibility of such an algorithm is presented in Supplementary Material B (SM-A). Related to finding the medoid of a set is finding the *geometric median*, which in vector spaces is defined as the point in the vector space with minimum energy. The relationship between the two problems is discussed in Section 2.1.

Much work has been done to develop approximate algorithms in the context of network analysis. The `RAND` algorithm of Eppstein and Wang (2004) can be used to estimate the energy of all nodes in a graph. The accuracy of `RAND` depends on the diameter of the network, which motivated Cohen et al. (2014) to use pivoting to make `RAND` more effective for large diameter networks. The work most closely related to ours is that of Okamoto et al. (2008), where `RAND` is adapted to the task of finding the k lowest energy nodes, $k = 1$ corresponding to the medoid problem. The resulting `TOPRANK` algorithm of Okamoto et al. (2008) has run time $\tilde{O}(N^{5/3})$ under certain assumptions, and returns the medoid with probability $1 - O(1/N)$, that is *with high probability* (w.h.p.). Note that only their run time result requires any assumption, obtaining the medoid w.h.p. is guaranteed. `TOPRANK` is discussed in Section 2.2.

Proceedings of the 20^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54.

In this paper we present an algorithm which has expected run time $O(N^{3/2})$ under certain assumptions and always returns the medoid. In other words, we present an exact medoid algorithm with improved complexity over the state-of-the-art approximate algorithm, `TOPRANK`. We show through experiments that the new algorithm works well for low-dimensional data in $\mathbb{R}^d$ and for spatial network data. Our new medoid algorithm, which we call `trimed`, uses the triangle inequality to quickly eliminate elements which cannot be the medoid. The $O(N^{3/2})$ run time follows from the surprising result that all but $O(N^{1/2})$ elements can be eliminated in this way.

The complexity bound on expected run time which we derive contains a term which grows exponentially in dimension d, and experiments show that in very high dimensions `trimed` often ends up computing $O(N^2)$ distances.

1.2 K-medoids algorithms and our contribution

The K-medoids problem is to partition a set into K clusters, so as to minimise the sum over elements of dissimilarites with their nearest medoids. That is, to choose $\mathcal{M} = \{m(1), \dots, m(K)\} \subset \{1, \dots, N\}$ to minimise,

$$\mathcal{L}(\mathcal{M}) = \sum_{i=1}^{N} \min_{k \in \{1,\dots,K\}} \text{diss}(x(i), x(m(k))).$$

We focus on the special case where the dissimilarity is a distance (diss = dist), which is still more general than K-means which only applies to vector spaces. K-medoids is used in bioinformatics where elements are genetic sequences or gene expression levels (Chipman et al., 2003) and has been applied to clustering on graphs (Rattigan et al., 2007). In machine vision, K-medoids is often preferred, as a medoid is more easily interpretable than a mean (Frahm et al., 2010).

The K-medoids problem is NP-hard, but there exist approximation algorithms. The Voronoi iteration algorithm, appearing in Hastie et al. (2001) and later in Park and Jun (2009), consists of alternating between updating medoids and assignments, much in the same way as Lloyd's algorithm works for the K-means problem. We will refer to it as `KMEDS`, and to Lloyd's K-means algorithm as `lloyd`.

One significant difference between `KMEDS` and `lloyd` is that the computation of a medoid is quadratic in the number of elements per cluster whereas the computation of a mean is linear. By incorporating our new medoid algorithm into `KMEDS`, we break the quadratic dependency of `KMEDS`, bringing it closer in performance to `lloyd`. We also show how ideas for accelerating `lloyd` presented in Elkan (2003) can be used in `KMEDS`.

It should be noted that algorithms other than `KMEDS` have been proposed for finding approximate solutions to the K-medoids problem, and have been shown to be very effective in Newling and Fleuret (2016b). These include `PAM` and `CLARA` of Kaufman and Rousseeuw (1990), and `CLARANS` of Ng et al. (2005). In this paper we do not compare cluster qualities of previous algorithms, but focus on accelerating the `lloyd` equivalent for K-medoids as a test setting for our medoid algorithm `trimed`.

2 Previous works

2.1 A related problem: the geometric median

A problem closely related to the medoid problem is the geometric median problem. In the vector space $\mathbb{R}^d$ the geometric median, assuming it is unique, is defined as,

$$g(\mathcal{S}) = \arg\min_{v \in \mathcal{V}} \left(\sum_{y \in \mathcal{S}} \|v - y\| \right). \tag{1}$$

While the medoid of a set is defined in any space with a distance measure, the geometric median is specific to vector spaces, where addition and scalar multiplication are defined. The convexity of the objective function being minimised in (1) has enabled the development of fast algorithms. In particular, Cohen et al. (2016) present an algorithm which obtains an estimate for the geometric median with relative error $1 + O(\epsilon)$ with complexity $O(nd \log^3(\frac{n}{\epsilon}))$. In $\mathbb{R}^d$, one may hope that such an algorithm can be converted into an exact medoid algorithm, but it is not clear how to do this.

Thus, while it may be possible that fast geometric median algorithms can provide inspiration in the development of medoid algorithms, they do not work out of the box. Moreover, geometric median algorithms cannot be used for network data as they only work in vector spaces, thus they are useless for the spatial network datasets which we consider in Section 5.

2.2 Medoid Algorithms : `TOPRANK` and `TOPRANK2`

In Eppstein and Wang (2004), the `RAND` algorithm for estimating the energy of all elements of a set $\mathcal{S} = \{x(1), \dots, x(N)\}$ is presented. While `RAND` is presented in the context of graphs, where the N elements are nodes of an undirected graph and the metric is shortest path length, it can equally well be applied to any set endowed with a distance. The simple idea of `RAND` is to estimate the energy of each element from a

sample of *anchor* nodes I, so that for $j \in \{1, \dots, N\}$,

$$\hat{E}(j) = \frac{N}{|I|(N-1)} \sum_{i \in I} \text{dist}(x(j), x(i)).$$

An elegant feature of `RAND` in the context of sparse graphs is that Dijkstra's algorithm needs only be run from anchor nodes $i \in I$, and not from every node. The key result of Eppstein and Wang (2004) is the following. Suppose that $\mathcal{S}$ has diameter Δ, that is

$$\Delta = \max_{(i,j) \in \{1,\dots,N\}^2} \text{dist}(x(i), x(j)),$$

and let $\epsilon > 0$ be some error tolerance. If I is of size $\Omega(\log(N)/\epsilon)$, then $\mathbb{P}(|E(j) - \hat{E}(j)| > \epsilon\Delta)$ is $O\left(\frac{1}{N^2}\right)$ for all $j \in \{1, \dots, N\}$. Using the union bound, this means there is a $O\left(\frac{1}{N}\right)$ probability that at least one energy estimate is off by more than $\epsilon\Delta$, and so we say that *with high probability* (w.h.p.) all errors are less than $\epsilon\Delta$.

`RAND` forms the basis of the `TOPRANK` algorithm of Okamoto et al. (2008). Whereas `RAND` w.h.p. returns an element which has energy within ϵ of the minimum, `TOPRANK` is designed to w.h.p. return the true medoid. In motivating `TOPRANK`, Okamoto et al. (2008) observe that the expected difference between consecutively ranked energies is $O(\Delta/N)$, and so if one wishes to correctly rank all nodes, one needs to distinguish between energies at a scale $\epsilon = \Delta/N$, for which the result of Eppstein and Wang (2004) dictates that $\Theta(N \log N)$ anchor elements are required with `RAND`, which is more elements than $\mathcal{S}$ contains. However, to obtain just the highest ranked node should require less information than obtaining a full ranking of nodes, and it is to this task that `TOPRANK` is adapted.

The idea behind `TOPRANK` is to accurately estimate only the energies of promising elements. The algorithm proceeds in two passes, where in the first pass promising elements are earmarked. Specifically, the first pass runs `RAND` with $N^{2/3} \log^{1/3}(N)$ anchor elements to obtain $\hat{E}(i)$ for $i \in \{1, \dots, N\}$, and then discards elements whose $\hat{E}(i)$ lies below threshold τ given by,

$$\tau = \min_{j \in \{1,\dots,N\}} \hat{E}(j) + 2\hat{\Delta}\alpha' \left(\frac{\log n}{n}\right)^{\frac{1}{3}}, \qquad (2)$$

where $\hat{\Delta}$ is an upper bound on Δ obtained from the anchor nodes, and α' is some constant satisfying $\alpha' > 1$. The second pass computes the true energy of the undiscarded elements, returning the one with lowest true energy. Note that a smaller α' value results in a lower (better) threshold, we discuss this point further in SM-C.

To obtain run time guarantees, `TOPRANK` requires that the distribution of node energies is non-decreasing near to the minimum, denoted by E^*. More precisely, letting f_E be the probability distribution of energies, the algorithms require the existence of $\epsilon > 0$ such that,

$$E^* \leq \tilde{e} < e < E^* + \epsilon \implies f_E(\tilde{e}) \leq f_E(e). \qquad (3)$$

If assumption 3 holds, then the run time is $\tilde{O}(N^{\frac{5}{3}})$. A second algorithm presented in Okamoto et al. (2008) is `TOPRANK2`, where the anchor set I is grown incrementally until some heuristic criterion is met. There is no runtime guarantee for `TOPRANK2`, although it has the potential to run much faster than `TOPRANK` under favourable conditions. Pseudocode for `RAND`, `TOPRANK` and `TOPRANK2` is presented in SM-C.

2.3 K-medoids algorithm : `KMEDS`

The Voronoi iteration algorithm, which we refer to as `KMEDS`, is similar to `lloyd`, the main difference being that cluster medoids are computed instead of cluster means. It has been desribed in the literature at least twice, once in Hastie et al. (2001) and then in Park and Jun (2009), where a novel initialisation scheme is developed. Pseudocode is presented in SM-B.

All N^2 distances are computed and stored upfront with `KMEDS`. Then, at each iteration, KN comparisons are made during assignment and $\Omega(N^2/K)$ additions are made during medoid update. The initialisation scheme of `KMEDS` requires all N^2 distances. Each iteration of `KMEDS` requires retrieving at least $\max\left(KN, N^2/K\right)$ distinct distances, as can be shown by assuming balanced clusters.

As an alternative to computing all distances upfront, one could store per-cluster distance matrices which get updated on-the fly when assignments change. Using such an approach, the best one could hope for would be $\max\left(KN, N^2/K\right)$ distance calculations and $\Theta(N^2/K)$ memory. If one were to completely forego storing distances in memory and calculate distances only when needed, the number of distance calculations would be at least $r(KN + N^2/K)$, where r is the number of iterations.

The initialisation scheme of Park and Jun (2009) selects K well centered elements as initial medoids. This goes against the general wisdom for K-means initialisation, where centroids are initialised to be well separated (Arthur and Vassilvitskii, 2007). While their new scheme of Park and Jun (2009) performs well on a limited number of small 2-D datasets, we show in Section 5.2 that in general uniform initialisation performs as well or better.

3 Our new medoid algorithm : `trimed`

We present our new algorithm, `trimed`, for determining the medoid of set $\mathcal{S} = \{x(1), \dots, x(N)\}$. Whereas the approach with `TOPRANK` is to empirically *estimate* $E(i)$ for $i \in \{1, \dots, N\}$, the approach with `trimed`, presented as Alg. 1, is to *bound* $E(i)$. When `trimed` terminates, an index $m^* \in \{1, \dots, N\}$ has been determined, along with lower bounds $l(i)$ for all $i \in \{1, \dots, N\}$, such that $E(m^*) \leq l(i) \leq E(i)$, and thus $x(m^*)$ is the medoid. The bounding approach uses the triangle inequality, as depicted in Figure 1.

Algorithm 1 The `trimed` algorithm for computing the medoid of $\{x(1), \dots, x(N)\}$.

1: $l \leftarrow \underline{0}_N$ // lower bounds on energies, maintained such that $l(i) \leq E(i)$ and initialised as $l(i) = 0$.
2: $m^{cl}, E^{cl} \leftarrow -1, \infty$ // index of best medoid candidate found so far, and its energy.
3: **for** $i \in$ `shuffle` $(\{1, \dots, N\})$ **do**
4: **if** $l(i) < E^{cl}$ **then**
5: **for** $j \in \{1, \dots, N\}$ **do**
6: $d(j) \leftarrow \text{dist}(x(i), x(j))$
7: **end for**
8: $l(i) \leftarrow \frac{1}{N-1} \sum_{j=1}^{N} d(j)$ // set $l(i)$ to be tight, that is $l(i) = E(i)$.
9: **if** $l(i) < E^{cl}$ **then**
10: $m^{cl}, E^{cl} \leftarrow i, l(i)$
11: **end if**
12: **for** $j \in \{1, \dots, N\}$ **do**
13: $l(j) \leftarrow \max(l(j), |l(i) - d(j)|)$ // using $E(i)$ and $\text{dist}(x(i), x(j))$ to possibly improve bound on $E(j)$.
14: **end for**
15: **end if**
16: **end for**
17: $m^*, E^* \leftarrow m^{cl}, E^{cl}$
18: **return** $x(m^*)$

The algorithm `trimed` iterates through the N elements of $\mathcal{S}$. Each time a new element with energy lower than the current lowest energy (E^{cl}) is found, the index of the current best medoid (m^{cl}) is updated (line 10). Lower bounds on energies are used to quickly eliminate poor medoid candidates (line 4). Specifically, if lower bound $l(i)$ on the energy of element i is greater than or equal to E^{cl}, then i is eliminated. If the bound test fails to eliminate element i, then it is *computed*, that is, all distances to element i are computed (line 6). The computed distances are used to potentially improve lower bounds for all elements (line 13). Theorem 3.1 states that `trimed` finds the medoid. The proof relies on showing that lower bounds remain consistent when updated (line 13).

The algorithm is very straightforward to implement,

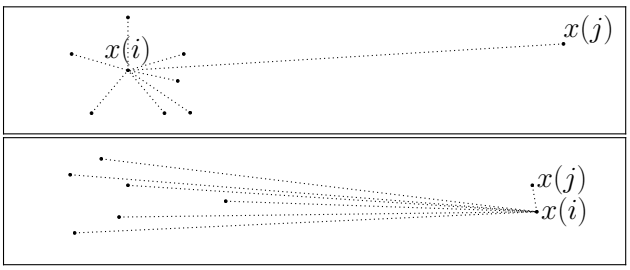

Figure 1: Using the inequality $E(j) \geq |E(i) - \text{dist}(x(i), x(j))|$ to eliminate $x(j)$ as a medoid candidate. Computed element $x(i)$ with energy $E(i) \geq E^{cl}$ is used as a pivot to lower bound $E(j)$. The two cases where the inequality is effective are when (case 1, above) $\text{dist}(x(i), x(j)) - E(i) \geq E^{cl}$ and (case 2, below) $E(i) - \text{dist}(x(i), x(j)) \geq E^{cl}$, as both lead to $E(j) \geq E^{cl}$ which eliminates $x(j)$ as a medoid candidate.

and requires only two additional floating point values per datapoint: for sample i, one for $l(i)$ and one for $d(i)$. Computing either all or no distances from a sample makes particularly good sense for network data, where computing all distances to a single node is efficiently performed using Dijkstra's algorithm.

Theorem 3.1. *`trimed` returns the medoid of set $\mathcal{S}$.*

Proof. We need to prove that $l(j) \leq E(j)$ for all $j \in \{1, \dots, N\}$ at all iterations of the algorithm. Clearly, as $l(j) = 0$ at initialisation, we have $l(j) \leq E(j)$ at initialisation. $E(j)$ does not change, and the only time that $l(j)$ may change is on line 13, where we need to check that $|l(i) - d(j)| \leq E(j)$. At line 13, $l(i) = E(i)$ from line 8, and $d(j) = \text{dist}(x(i), x(j))$, so at line 13 we are effectively checking that $|E(i) - \text{dist}(x(i), x(j))| \leq E(j)$. But this is a simple consequence of the triangle inequality, as we now show. Using the definition, $E(j) = \frac{1}{N} \sum_{l=1}^{N} \text{dist}(x(l), x(j))$, we have on the one hand,

$$\begin{aligned} E(j) &\geq \frac{1}{N} \sum_{l=1}^{N} \text{dist}(x(l), x(i)) - \text{dist}(x(i), x(j)) \\ &\geq E(i) - \text{dist}(x(i), x(j)), \end{aligned} \tag{4}$$

and on the other hand,

$$\begin{aligned} E(j) &\geq \frac{1}{N} \sum_{l=1}^{N} \text{dist}(x(i), x(j)) - \text{dist}(x(l), x(i)) \\ &\geq \text{dist}(x(i), x(j)) - E(i). \end{aligned} \tag{5}$$

Combining (4) and (5) we obtain the required inequality $|E(i) - \text{dist}(x(i), x(j))| \leq E(j)$. □

The bound test (line 4) becomes more effective at later iterations, for two reasons. Firstly, whenever an element is computed, the lower bounds of other samples may increase. Secondly, E^{cl} will decrease whenever a better medoid candidate is found. The main result of this paper, presented as Theorem 3.2, is that in $\mathbb{R}^d$ the expected number of computed elements is $O(N^{\frac{1}{2}})$ under some weak assumptions. We show in Section 5 that the $O(N^{\frac{1}{2}})$ result holds even in settings where the assumptions are not valid or relevent, such as for network data.

The shuffle on line 3 is performed to avoid w.h.p. pathological orderings, such as when elements are ordered in descending order of energy which would result in all N elements being computed.

Theorem 3.2. *Let $\mathcal{S} = \{x(1), \dots, x(N)\}$ be a set of N elements in $\mathbb{R}^d$, drawn independently from probability distribution function f_X. Let the medoid of $\mathcal{S}$ be $x(m^*)$, and let $E(m^*) = E^*$. Suppose that there exist strictly positive constants ρ, δ_0 and δ_1 such that for any set size N with probability $1 - O(1/N)$*

$$x \in \mathcal{B}_d(x(m^*), \rho) \implies \delta_0 \leq f_X(x) \leq \delta_1, \qquad (6)$$

where $\mathcal{B}_d(x, r) = \{x' \in \mathbb{R}^d \ : \ \|x' - x\| \leq r\}$. Let $\alpha > 0$ be a constant (independent of N) such that with probability $1 - O(1/N)$ all $i \in \{1, \dots, N\}$ satisfy,

$$x(i) \in \mathcal{B}_d(x(m^*), \rho) \implies E(i) - E^* \geq \alpha \|x(i) - x(m^*)\|^2. \qquad (7)$$

Then, the expected number of elements computed by `trimed` *is $O\left(\left(V_d[1]\delta_1 + d\left(\frac{4}{\alpha}\right)^d\right) N^{\frac{1}{2}}\right)$, where $V_d[1] = \pi^{\frac{d}{2}}/(\Gamma(\frac{d}{2}+1))$ is the volume of $\mathcal{B}_d(0,1)$.*

3.1 On the assumptions in Theorem 3.2

The assumption of constants ρ, δ_0 and δ_1 made in Theorem 3.2 is weak, and only pathological distributions might fail it, as we now discuss. For the assumptions to fail requires that f_X vanishes or diverges at the distribution medoid. Any reasonably behaved distribution does not have this behaviour, as illustrated in Figure 2. The constant α is a strong convexity constant. The existence of $\alpha > 0$ is guaranteed by the existence of ρ, δ_0 and δ_1, as the mean of a sum of uniformly spaced cones converges to a quadratic function. This is illustrated in 1-D in Figure 5 in SM-G, but holds true in any dimension.

Note that the assumptions made are on the distribution f_X, and not on the data itself. This must be so in order to prove complexity results in N.

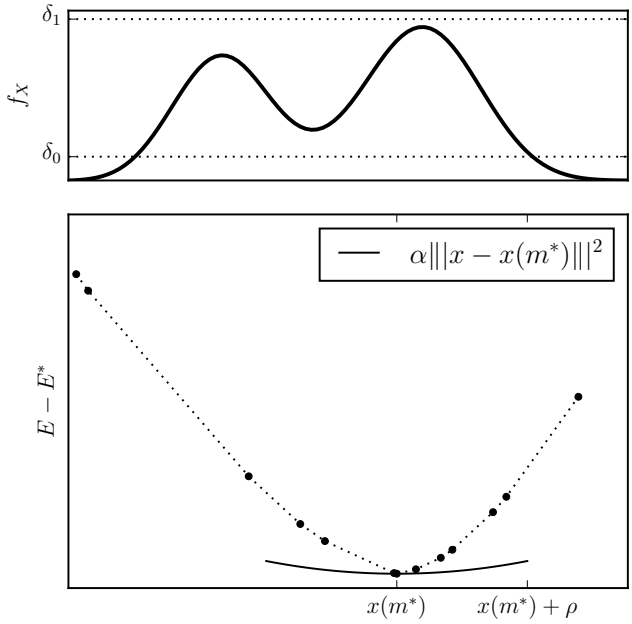

Figure 2: Illustration in 1-D of the constants used in Theorem 3.2. Above, δ_0 and δ_1 bound the probability density function in a region containing the distribution medoid. Below, the energy of samples grows quadratically around the medoid $x(m^*)$. The energy E is a sum of cones centered on samples, which is approximately quadratic unless f_X vanishes or explodes, guaranteeing the existence of $\alpha > 0$ required in Theorem 3.2.

3.2 Sketch of proof of Theorem 3.2

We now sketch the proof of Theorem 3.2, showing how (6) and (7) are used. A full proof is presented in SM-G. Firstly, let the index of the first element after the shuffle on line 3 be i'. Then, no elements beyond radius $2E(i')$ of $x(i')$ will subsequently be computed, due to type 1 eliminations (see Figure 1). Therefore, all computed elements are contained within $\mathcal{B}_d(x(i'), 2E(i'))$.

Next, notice that once an element $x(i)$ has been computed in `trimed`, no elements in the ball $\mathcal{B}_d(x(i), E(i) - E^{cl})$ will subsequently be computed, due to type 2 eliminations (see Figure 1). We refer to such a ball as an *exclusion ball*. By upper bounding the number of exclusion balls contained in $\mathcal{B}_d(x(i'), 2E(i'))$ using a volumetric argument, we can obtain a bound on the number of computed elements, but obtaining such an upper bound requires that the radii of exclusion ball $E(i) - E^{cl}$ be bounded below by a strictly positive value. However, by using a volumetric argument only beyond a certain positive radius of the medoid (a radius $N^{-1/2d}$), we have $\alpha > 0$ in (15) which provides a lower bound on exclusion ball radii, assuming $E^{cl} \approx E^*$. Using δ_0 and δ_1 we can show that E^{cl}

approaches E^* sufficiently fast to validate the approximation $E^{cl} \approx E^*$.

It then remains to count the number of computed elements within radius $N^{-1/2d}$ of the medoid. One cannot find a strict upper bound here, but using the boundedness of f_X provided by δ_1, we have w.h.p. that the number of elements computed within $N^{-1/2d}$ is $O(\delta_1 N^{1/2})$, as the volume of a sphere scales as the dth power of its radius.

4 Our accelerated K-medoids algorithm : `trikmeds`

We adapt our new medoid algorithm `trimed` and borrow ideas from Elkan (2003) to show how `KMEDS` can be accelerated. We abandon the initial N^2 distance calculations, and only compute distances when necessary. The accelerated version of `lloyd` of Elkan (2003) maintains KN bounds on distances between points and centroids, allowing a large proportion of distance calculations to be eliminated. We use this approach to accelerate assignment in `trikmeds`, incurring a memory cost $O(KN)$. By adopting the algorithm of Newling and Fleuret (2016a) or that of Hamerly (2010), the memory can be reduced to $O(N)$. We accelerate the medoid update step by adapting `trimed`, reusing lower bounds between iterations, so that `trimed` is only run from scratch once at the start. Details and pseudocode are presented in SM-H.

One can relax the bound test in `trimed` so that for $\epsilon > 0$ element i is computed if $l(i)(1+\epsilon) < E^{cl}$, guaranteeing that an element with energy within a factor $1+\epsilon$ of E^* is found. It is also possible to relax the bound tests in the assignment step of `trikmeds`, such that the distance to an assigned cluster's medoid is always within a factor $1+\epsilon$ of the distance to the nearest medoid. We denote by `trikmeds-`ϵ the `trikmeds` algorithm where the update and assignment steps are relaxed as just discussed, with `trikmeds-0` being exactly `trikmeds`. The motivation behind such a relaxation is that, at all but the final few iterations, it is probably a waste of computation obtaining medoids and assignments at high resolution, as in subsequent iterations they may change.

5 Results

We first compare the performance of the medoid algorithms `TOPRANK`, `TOPRANK2` and `trimed`. We then compare the K-medoids algorithms, `KMEDS` and `trikmeds`. Our C++ implementations and Python binding will be made available under a free open source license.

5.1 Medoid algorithm results

We compare our new exact medoid algorithm `trimed` with state-of-the-art approximate algorithms `TOPRANK` and `TOPRANK2`. Recall, Okamoto et al. (2008) prove that the approximate algorithms return w.h.p. the true medoid. We confirm that this is the case in all our experiments, where the approximate algorithms return the same element as `trimed`, which we know to be correct by Theorem 3.1. We now focus on comparing computational costs, which are proportional to the number of computed points.

Results on artificial datasets are presented in Figure 3, where our two main observations relate to scaling in N and dimension d. The artificial data are (left) uniformly drawn from $[0,1]^d$ and (right) drawn from $\mathcal{B}_d(0,1)$ with probability of lying within radius $1/2^{1/d}$ of $1/200$, as opposed to $1/2$ as would be the case under uniform density. Details about sampling from this distribution can be found in SM-F. Results on a mix of publicly available real and artificial datasets are presented in Table 1 and discussed in Section 5.1.2.

5.1.1 Scaling with N and d on artificial datasets

In Figure 3 we observe that the number of points computed by `trimed` is $O(N^{1/2})$, as predicted by Theorem 3.2. This is illustrated (right) by the close fit of the number of computed points to exact square root curves at sufficiently large N for $d \in \{2, 6\}$.

Recall that `TOPRANK` consists of two passes, a first where $N^{2/3}\log^{1/3} N$ anchor points are computed, and a second where all sub-threshold points are computed. We observe that for small N `TOPRANK` computes all N points, which corresponds to all points lying below threshold. At sufficiently large N the threshold becomes low enough for all points to be eliminated after the first pass. The effect is particularly dramatic in high dimensions ($d = 6$ on right), where a phase transition is observed between all and no points being computed in the second pass.

Dimension d appears in Theorem 3.2 through a factor $d(4/\alpha)^d$, where α is the strong convexity of the energy at the medoid. In Figure 3, we observe that the number of computed points increases with d for fixed N, corresponding to a relatively small α. The effect of α on the number of computed elements is considered in greater detail in SM-F.

In contrast to the above observation that the number of computed points increases as dimension increases for `trimed`, `TOPRANK` appears to scale favourably with dimension. This observation can be explained in terms of the distribution of energies, with energies close to E^*

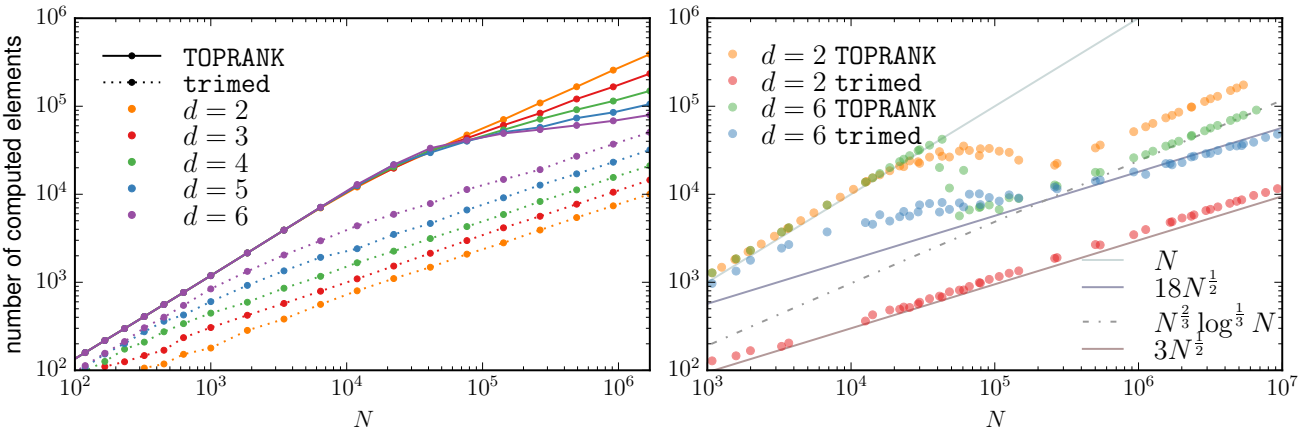

Figure 3: Comparison of TOPRANK and our algorithm trimed on simulated data. On the left, points are drawn uniformly from $[0,1]^d$ for $d \in \{2,\dots,6\}$, and on the right they are drawn from $\mathcal{B}_d(0,1)$ for $d \in \{2,6\}$, with an increased density near the edge of the ball. Fewer points (elements) are computed by trimed than by TOPRANK in all scenarios. For small N, TOPRANK computes $O(N)$ points, before transitioning to $\tilde{O}(N^{2/3})$ computed points for large N. trimed computes $O(N^{1/2})$ points. Note that trimed performs better in low-d than in high-d, with the reverse trend being true for TOPRANK. These observations are discussed in further detail in the text.

being less common in higher dimensions, as discussed in SM-J.

5.1.2 Results on publicly available real and simulated datasets

We present the datasets used here in detail in SM-I. For all datasets, algorithms TOPRANK, TOPRANK2 and trimed were run 10 times with a distinct seed, and the mean number of iterations ($\hat{n}$) over the 10 runs was computed. We observe that our algorithm trimed is the best performing algorithm on all datasets, although in high-dimensions (MNIST-0) and on social network data (Gnutella) no algorithm computes significantly fewer than N elements. The failure in high-dimensions (MNIST-0) of trimed is in agreement with Theorem 3.2, where dimension appears as the exponent of a constant term. The small world network data, Gnutella, can be embedded in a high-dimensional Euclidean space, and thus the failure on this dataset can also be considered as being due to high-dimensions. For low-dimensional real and spatial network data, trimed consistently computes $O(N^{1/2})$ elements.

5.1.3 But who needs the exact medoid anyway?

A valid criticism that could be raised at this stage would be that for large datasets, finding the exact medoid is probably overkill, as any point with energy reasonably close to E^* suffices for most applications. But consider, the RAND algorithm requires computing $\log N/\epsilon^2$ elements to confidently return an element with energy within ϵE^* of E^*. For $N = 10^5$ and $\epsilon = 0.05$, this is 4600, already more than trimed requires to obtain the exact medoid on low-d datasets of comparable size.

5.2 K-medoids algorithm results

With N elements to cluster, KMEDS is $\Theta(N^2)$ in memory, rendering it unusable on even moderately large datasets. To compare the initialisation scheme proposed in Park and Jun (2009) to random initialisation, we have performed experiments on 14 small datasets, with $K \in \{10, \lceil N^{1/2} \rceil, \lceil N/10 \rceil\}$. For each of these 42 experimental set-ups, we run the deterministic KMEDS initialisation once, and then uniform random initialisation, 10 times. Comparing the mean final energy of the two initialisation schemes, in only 9 of 42 cases does KMEDS initialisation result in a lower mean final energy. A Table containing all results from these experiments in presented in SM-E.

Having demonstrated that random uniform initialisation performs at least as well as the initialisation scheme of KMEDS, and noting that trikmeds-0 returns exactly the same clustering as would KMEDS with uniform random initialisation, we turn our attention to the computational performance of trikmeds. Table 2 presents results on 4 datasets, each described in SM-I. The first numerical column is the relative number of distance calculations using trikmeds-0 and KMEDS, where large savings in distance calculations, especially in low-dimensions, are observed. Columns ϕ_c and ϕ_E are the number of distance calculations and energies respectively, using $\epsilon \in \{0.01, 0.1\}$, relative to $\epsilon = 0$.

			TOPRANK	TOPRANK2	trimed
dataset	type	N	$\hat{n}$	$\hat{n}$	$\hat{n}$
Birch 1	2-d	1.0×10^5	57944	100180	**2180**
Birch 2	2-d	1.0×10^5	66062	100180	**2208**
Europe	2-d	1.6×10^5	176095	169535	**2862**
U-Sensor Net	u-graph	3.6×10^5	113838	327216	**1593**
D-Sensor Net	d-graph	3.6×10^5	99896	176967	**1372**
Pennsylvania road	u-graph	1.1×10^6	216390	time-out	**2633**
Europe rail	u-graph	4.6×10^4	35913	47041	**518**
Gnutella	d-graph	6.3×10^3	7043	6407	**6328**
MNIST	784-d	6.7×10^3	7472	6799	**6514**

Table 1: Comparison of TOPRANK, TOPRANK2 and our algorithm trimed on publicly available real and simulated datasets. Column 2 provides the type of the dataset, where 'x-d' denotes x-dimensional vector data, while 'd-graph' and 'u-graph' denote directed and undirected graphs respectively. Column $\hat{n}$ gives the mean number of elements computed over 10 runs. Our proposed trimed algorithm obtains the true medoid with far fewer computed points in low dimensions and on spatial network data. On the social network dataset (Gnutella) and the very high-d dataset (MNIST), all algorithms fail to provide speed-up, computing approximately N elements.

We observe large reductions in the number of distance computations with only minor increases in energy.

6 Conclusion and future work

We have presented our new trimed algorithm for computing the medoid of a set, and provided strong theoretical guarantees about its performance in $\mathbb{R}^d$. In low-dimensions, it outperforms the state-of-the-art approximate algorithm on a large selection of datasets. The algorithm is very simple to implement, and can easily be extended to the general ranking problem. In the future, we propose to explore the idea of using more complex triangle inequality bounds involving several points, with as goal to improve on the $O(N^{1/2})$ number of computed points.

We have demonstrated how trimed, when combined with the approach of Elkan (2003), can greatly reduce the number of distance calculations required by the Voronoi iteration K-medoids algorithm of Park and Jun (2009). In the future we would like to replace the strategy of Elkan (2003) with that of Hamerly (2010), which will be better adapted to graph clustering as either all or no distances are computed with it, making it more amenable to Dijkstra's algorithm.

Acknowledgements

The authors are grateful to Wei Chen for helpful discussions of the TOPRANK algorithm. James Newling was funded by the Hasler Foundation under the grant 13018 MASH2.

References

Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Chipman, H., Hastie, T., and Tibshirani, R. (2003). *Statistical Analysis of Gene Expression Microarray Data*. Chapman & Hall. Chapter 4.

Cohen, E., Delling, D., Pajor, T., and Werneck, R. F. (2014). Computing classic closeness centrality, at scale. In *Proceedings of the Second ACM Conference on Online Social Networks*, COSN '14, pages 37–50, New York, NY, USA. ACM.

Cohen, M. B., Lee, Y. T., Miller, G. L., Pachocki, J. W., and Sidford, A. (2016). Geometric median in nearly linear time. In *STOC16*. submitted.

Elkan, C. (2003). Using the triangle inequality to accelerate k-means. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 147–153.

Eppstein, D. and Wang, J. (2004). Fast approximation of centrality. *J. Graph Algorithms Appl.*, 8(1):39–45.

Frahm, J.-M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S., and Pollefeys, M. (2010). Building rome on a cloudless day. In *Proceedings of the 11th European Conference on Computer Vision:*

			$K = 10$					$K = \lceil\sqrt{N}\rceil$				
			$\epsilon = 0$	$\epsilon = 0.01$		$\epsilon = 0.1$		$\epsilon = 0$	$\epsilon = 0.01$		$\epsilon = 0.1$	
Dataset	N	d	N_c/N^2	ϕ_c	ϕ_E	ϕ_c	ϕ_E	N_c/N^2	ϕ_c	ϕ_E	ϕ_c	ϕ_E
Europe	1.6×10^5	2	0.067	0.33	1.004	0.01	1.054	0.008	0.68	1.031	0.39	1.090
Conflong	1.6×10^5	3	0.042	0.67	1.001	0.08	1.014	0.006	0.92	1.003	0.61	1.026
Colormo	6.8×10^4	9	0.163	0.92	1.000	0.35	1.015	0.011	0.98	1.000	0.82	1.005
MNIST50	6.0×10^4	50	0.280	0.99	1.000	0.95	1.001	0.019	0.99	1.001	0.97	1.001

Table 2: Relative numbers of distance calculations and final energies using `trikmeds-`ϵ for $\epsilon \in \{0, 0.01, 0.1\}$. The number of distance calculations with `trikmeds-0` is N_c, presented here relative to the number computed using `KMEDS` (N^2) in column N_c/N^2. The number of distance calculations with $\epsilon \in \{0.01, 0.1\}$ relative to `trikmeds-0` are given in columns ϕ_c, so $\phi_c = 0.33$ means $3\times$ fewer calculations than with $\epsilon = 0$. The final energies with $\epsilon \in \{0.01, 0.1\}$ relative to `trikmeds-0` are given in columns ϕ_E. We see that `trikmeds-0` uses significantly fewer distance calculations than would `KMEDS`, especially in low-dimensions where a greater than $K\times$ reduction is observed ($N_C/N^2 < 1/K$). For low-d, additional relaxation further increases the saving in distance calculations with little cost to final energy.

Part IV, ECCV'10, pages 368–381, Berlin, Heidelberg. Springer-Verlag.

Hamerly, G. (2010). Making k-means even faster. In *SDM*, pages 130–140.

Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2001). *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. Springer, New York.

Hoare, C. A. R. (1961). Algorithm 65: Find. *Commun. ACM*, 4(7):321–322.

Kaufman, L. and Rousseeuw, P. J. (1990). *Finding groups in data : an introduction to cluster analysis*. Wiley series in probability and mathematical statistics. Wiley, New York. A Wiley-Interscience publication.

Newling, J. and Fleuret, F. (2016a). Fast k-means with accurate bounds. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 936–944.

Newling, J. and Fleuret, F. (2016b). K-medoids for k-means seeding. arXiv:1609.04723. Under review.

Ng, R. T., Han, J., and Society, I. C. (2005). Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, pages 1003–1017.

Okamoto, K., Chen, W., and Li, X.-Y. (2008). Ranking of closeness centrality for large-scale social networks. In *Proceedings of the 2Nd Annual International Workshop on Frontiers in Algorithmics*, FAW '08, pages 186–195, Berlin, Heidelberg. Springer-Verlag.

Park, H.-S. and Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl.*, 36(2):3336–3341.

Rattigan, M. J., Maier, M., and Jensen, D. (2007). Graph clustering with network structure indices. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 783–790, New York, NY, USA. ACM.