One Hop Greedy Permutations*

Donald R. Sheehy†

Abstract

We adapt and generalize a heuristic for k-center clustering to the permutation case, where every prefix of the ordering is a guaranteed approximate solution. The *one-hop greedy permutations* work by choosing at each step the farthest unchosen point and then looking in its local neighborhood for a point that covers the most points at a certain scale. This balances the competing demands of reducing the coverage radius and also covering as many points as possible. This idea first appeared in the work of Garcia-Diaz et al. [6] and their algorithm required $O(n^2 \log n)$ time for a fixed k (i.e. not the whole permutation). We show how to use geometric data structures to approximate the entire permutation in $O(n \log \Delta)$ time for metrics sets with spread Δ. Notably, this running time is asymptotically the same as the running time for computing the ordinary greedy permutation.

1 Introduction

Greedy permutations of points in a metric space are useful for many standard computations, such as proximity search data structures, sampling, and k-center clustering. They were developed independently by Gonzalez [7] and also Dyer and Frieze [4] in 1985 for k-center clustering. Starting from any point the next point in a greedy permutation is chosen to be the farthest remaining point from the previously chosen points. Greedy permutations are effective for so many tasks because every prefix of the ordering gives a good, mostly uniform sample, keeping points as far apart as possible while (approximately) minimizing the maximum distance from any point to the sample.

The coverage radius of a subset $S \subset P$ is the minimum r such that $P \subseteq \bigcup_{x \in S} \texttt{ball}(x, r)$. The metric k-center problem is a search for k points that minimize the coverage radius. The first k points of a greedy permutation provide a 2-approximate k-center, i.e. the coverage radius is at most twice the optimal solution. This is known to be the best possible unless $P = NP$. However, there are several heuristics that have been shown to produce better solutions in practice. One such heuristic developed by Garcia-Diaz et al.[6] achieves a worse theoretical guarantee, but consistently outperforms the greedy approach on benchmarks. That approach works for a fixed r by choosing at each step, not the farthest point, but instead a point within distance r of the farthest point that covers the most previously uncovered points in its radius r ball. Then the algorithm binary searches for a good value of r. We call this the *one-hop k-center* algorithm.

Given that the greedy permutation is widely used because of its ability to provide a sequence of good covers, it makes sense to import these efficient heuristics from the setting with fixed r and k into the permutation setting. In this paper, we will generalize this approach to give a permutation of one-hop k-centers and show how to relate it to approximate greedy permutations. We call these *one-hop greedy permutations.* We will then show how to compute a one-hop greedy permutation in $O(n \log \Delta)$ time, where Δ is the *spread* of the input (the ratio of the largest to smallest pairwise distances). As (almost) always with such an analysis, the true worst case is $O(n^2)$, but would require point sets with exponential spread to achieve. In theory, this can be brought down to $O(n \log n)$ using elaborate theoretical techniques [8]; however, given that our interest is in the practical performance, we describe our algorithm in terms of a standard practical approach. In theory, the greedy approach is optimal anyways. In practice, it is very rare to find inputs with super-polynomial spread. A proof of concept implementation was used to generate some examples visualized in Section 5.

2 Background

2.1 Metrics Spaces

Let P be a finite subset of a metric space. Let $\mathbf{d}(a, b)$ denote the distance between a and b. The *metric ball* centered at x with radius r is

$$\texttt{ball}(x, r) := \{p \in P \mid \mathbf{d}(x, p) \leq r\}.$$

For a subset $S \subset P$, let $\mathbf{d}(a, S) := \min_{x \in S} \mathbf{d}(a, x)$. The *Hausdorff* distance between two subsets is defined as

$$\mathbf{d}_H(S, T) := \max\{\max_{s \in S} \mathbf{d}(s, T), \max_{t \in T} \mathbf{d}(t, S)\}.$$

So, for a subset $S \subset P$, this simplifies to $\max_{p \in P} \mathbf{d}(p, S)$, also known as the *coverage radius* of

*This work was partially supported by the NSF under grant CCF-1652218.

†Department of Computer Science, North Carolina State University, `don.r.sheehy@gmail.com`

S. A subset $S \subseteq P$ is an (α, β)*-net* if it has coverage radius at most β and all pairs of points are at least α apart. The constant α controls the *packing* and β controls the *covering*. If $\alpha = \beta$, then we call it an α-net, and if the constants are unimportant, we just call it a net. The *spread* Δ of a point set is the ratio of the largest distance to the smallest distance. The quantity $\log \Delta$ features naturally in the analysis of many geometric algorithms and data structures as it roughly counts the number of levels in a hierarchical data structure in which the scale drops by a constant factor at each level.

The *doubling dimension* for a metric is the minimum number ρ such that every ball of radius $2r$ can be covered by 2^ρ balls of radius r. A metric is said to be a *doubling metric* if the doubling dimension is bounded by a constant. The natural appeal of doubling metrics is that they give a notion of low dimensionality for general metric spaces. In particular, packing and covering arguments similar to those used in Euclidean space can be used. For example, an (α, β)-net of the points in a ball of radius r will have at most $O(2^{r/\alpha})$ points.

2.2 From Approximate Greedy to k-Center

Let $P = (p_1, \ldots, p_n)$ be an ordered set of points and let $P_i = (p_1, \ldots, p_i)$ be the ith *prefix*. A *c-approximate greedy permutation* is an ordering of P such that for all $i = 1 \ldots n-1$,

$$\mathbf{d}_H(P_i, P) \leq c\mathbf{d}(p_{i+1}, P_i).$$

A 1-approximate greedy permutation is simply called a *greedy permutation*.

Below, we explain how the standard proof that the greedy permutation gives 2-approximate k-centers for all k can be extended to the approximate greedy case. More specifically, we show that that a c-approximate greedy permutation yields a $2c$-approximate k-center clustering.

Lemma 1 *If P is ordered according to a c-approximate greedy permutation, then every prefix P_k is an $(\frac{r}{c}, r)$-net where $r = \mathbf{d}_H(P_k, P)$.*

Proof. Fix any value of k. The coverage radius of P_k is r by definition. For the packing condition, we observe that for any $i < j \leq k$, we have

$$\begin{aligned} \mathbf{d}(p_i, p_j) &\geq \mathbf{d}(p_j, P_{j-1}) \\ &\geq \frac{1}{c}\mathbf{d}_H(P, P_{j-1}) \\ &\geq \frac{1}{c}\mathbf{d}_H(P, P_k) \\ &= \frac{r}{c}. \end{aligned}$$

□

Lemma 2 *If $P = (p_1, \ldots, p_n)$ be a c-approximate greedy permutation for some $c \geq 1$, then P_k is a $2c$-approximate k-center for all k.*

Proof. Let $r = \mathbf{d}_H(P_k, P)$ be the coverage radius of P_k. Let r_* be the radius of the optimal k-center, *Opt*. The goal is to show that $r \leq 2cr_*$.

By Lemma 1, every two points in P_k are at least $\frac{r}{c}$ apart. If there are two points of P_k within distance r_* of one center in the optimal solution, then their distance is at most $2r_*$. Thus, it would follow that $\frac{r}{c} \leq 2r_*$ and therefore $r \leq 2cr_*$ as desired. If no two points have this property, then every ball of radius r_* in the optimal solution contains a unique point of P_k and thus $\mathbf{d}_H(P_k, Opt) \leq r_*$. By the triangle inequality, $r \leq 2r_* \leq 2cr_*$ because

$$r = \mathbf{d}_H(P, P_k) \leq \mathbf{d}_H(P_k, Opt) + \mathbf{d}(P, Opt) \leq 2r_*.$$

□

3 The One-Hop Greedy Permutation

The heuristic proposed by Garcia-Diaz et al., which we refer to as *one-hop k-center* was motivated by the observation that the greedy permutation achieves its factor of 2 approximation factor by choosing points very conservatively. Another perspective is that the greedy approach reduces the coverage radius by actively seeking out and covering the extremes, which is at odds with the goal of finding "centers".

A simple one-dimensional example shows how the greedy algorithm can make poor selections (see Figure 1). Suppose we start with a unit-length line segment densely sampled with points and a sample point on one end. The next point taken in the greedy ordering is the other end of the segment. The coverage radius is one half. A better choice would be to take the point at $\frac{2}{3}$. This would reduce the coverage radius to $\frac{1}{3}$.

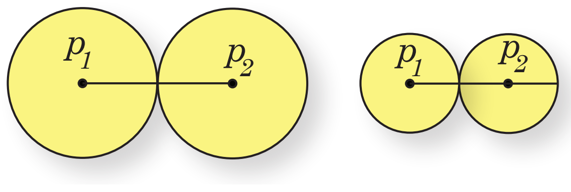

Figure 1: On the left, a greedy permutation takes the end point as its second point, resulting in a coverage radius of $\frac{1}{2}$. On the right, the one-hop greedy permutation backs of from the purely greedy choice resulting in a coverage radius of $\frac{1}{3}$.

There are several challenges that naturally arise in generalizing the one-hop approach from fixed radius (i.e. fixed k) setting to the permutation setting. The main one is the changing radius; the radius decreases with each prefix. This means we must choose what the target radius should be at each step rather than using binary search as in the original work. If we did use binary search, we would potentially get a completely different set for each k instead of a single permutation of the input. The natural choice for the scale at each step is to be some fraction of the current radius. We will allow this to be a tunable parameter, which also allows us to prove some other theoretical properties below.

Let P be the input metric. We will define an ordering $(p_1, \dots p_n)$ of P. Let $P_i = (p_1, \dots, p_i)$ denote the prefixes of the ordering. For each i, let q_i denote the point in P that maximizes the distance to P_{i-1}. In other words, q_i is the point that would be added next if the permutation were greedy. Fix a parameter α. The ordering is *α-one-hop greedy* if for each i, the point p_i is the point in $\mathtt{ball}(q_i, \alpha r)$ with $r = \mathbf{d}(q_i, P_{i-1})$ that maximizes

$$\left| \mathtt{ball}(p_i, \alpha r) \setminus \bigcup_{j=1}^{i-1} \mathtt{ball}(p_j, \alpha r) \right| .$$

The point is to balance between covering the farthest point and also covering many points.

Lemma 3 *If P is ordered according to an α-one-hop greedy permutation, then P is $\frac{1}{1-\alpha}$-approximate greedy.*

Proof. By definition, the point p_i is chosen in a ball of radius αr centered at the point of distance $r = \mathbf{d}_H(P_{i-1}, P)$ from P_{i-1}. So, by the triangle inequality, $\mathbf{d}(p_i, P_{i-1}) \le (1-\alpha)r = (1-\alpha)\mathbf{d}_H(P_{i-1}, P)$. Therefore, P is $\frac{1}{1-\alpha}$-approximate greedy as desired. $\square$

The Lemma 2 and Lemma 3 imply that our one-hop greedy permutations will give $\frac{2}{1-\alpha}$-approximate k-center solutions. As α goes to zero, we get the standard greedy permutation and its corresponding 2-approximate k-centers.

The choice of α is nonobvious. On the one hand, a large value provides flexibility in choosing the next point. On the other hand, that takes more time and the guarantee gets worse. Algorithmically, the cleanest choice for α is to let it equal $\frac{1}{3}$. For any value of $\alpha \le \frac{1}{3}$, the points in $\mathtt{ball}(p_i, \alpha r)$ are disjoint from $\bigcup_{j=1}^{i-1} \mathtt{ball}(p_j, \alpha r)$. This saves us the trouble checking if points are already covered when choosing the next point. Perhaps accidentally, choosing $\alpha = \frac{1}{3}$ leads to a 3-approximate k-center, which is the approximation ratio achieved by Garcia-Diaz et al. in the fixed radius case. Whether such a choice is actually best will likely vary based on peculiarities of the input.

4 Efficient Approximations

The original algorithm for one-hop k-centers started by sorting all distances in $O(n^2 \log n)$ time. It then, used this ordering to compute a one-hop k-center for a given radius r in quadratic time in a manner similar to the original Gonzalez greedy ordering algorithm. The extra step in each iteration was the search in the neighborhood of the greedy choice for a point that covers more points in its radius r ball (that were not already covered by a previously inserted point). This also takes quadratic time. Thus, the total running time is $O(n^2 \log n)$.

In the low-dimensional setting, one can hope to do better with approximations. In the case of greedy permutations, Har-Peled and Mendel [8] showed that the approach of Clarkson [2, 3] runs in $O(n \log \Delta)$ time. We will augment this approach with a data structure that does approximate range sampling, to make the local improvement step faster. In the end, we will have an algorithm that runs in $O(n \log \Delta)$ time. Thus, it matches the asymptotic running time of the greedy permutation computation, albeit with worse constants.

A $(1+\varepsilon)$-*approximate one-hop greedy permutation* of P is an ordering of P that could be a one-hop greedy permutation if all distances are perturbed by a factor of at most $1+\varepsilon$. For such an approximation, it will suffice to choose the next point among an εr-net where r is the distance to the current farthest point. It also suffices to count points in metric balls only approximately as described below.

4.1 Approximate Range Sampling

A *metric range search* takes a point x and a radius r as input and returns the points in $\mathtt{ball}(x, r)$. A *c-approximate metric range search* returns a set of points S such that

$$\mathtt{ball}(x, r) \subseteq S \subseteq ball(x, cr).$$

That is, it may return some extra points that are close to the desired ball. Similarly, *metric range counting* and *c-approximate range counting* return the number of points that would be returned by the corresponding search.

Many data structures for proximity search on metric spaces can easily be adapted to perform *approximate range sampling*, which is a hybrid between range search and range counting. A range sampling query (x, r, ε) for a given resolution ε returns an ε-net of $\mathtt{ball}(x, r)$ along with a weight for each point. The points in the range are each assigned to a point in the sample within distance ε and the weight of a point is the number of points assigned to it. In the c-approximate variant, the range may include points of distance up to cr away from x.

Perhaps the simplest data structure to use for approximate range sampling would be the cover tree of Beygelzimer et al. [1], particularly in the variant by Izbicki and Shelton [9]. Other hierarchical data structures such as navigating nets [12], net-trees [8, 11, 10], or deformable spanners [5] could also be used. In any of these data structures, one searches through a hierarchy of nets, where the ith level is a 2^i net. They are all designed for range searching, though they are sold as nearest neighbor search data structures, which is, of course, a kind of range search with a range that shrinks as you proceed. A search through such a data structure maintains a 2^i-approximate range sample at each level. Moreover, for doubling metrics, the number of points in the sample is $2^{O(2^i/r)}$. So, any search that stops at a level i such that $r = \Omega(2^i)$ will only return a constant sized set of (weighted) points. The running time of such a search is proportional to the number of levels searched, which is $O(\log \Delta)$ in the worst case.

4.2 Putting it all together

A standard approach to computing a greedy permutation is to use a kind of discrete Voronoi diagram. At step i, each point is associated with its nearest neighbor in P_i. A *cluster* is the set of points associated with a given point. When the point p_i is added to the permutation, a search is performed to find which points now have p_i as their nearest neighbor. To speed up the search, one stores a graph with vertex set P_i that connects two points p_a and p_b if adding a point from the cluster of p_a could affect the cluster of p_b. Armed with this graph, the update only checks points within a constant factor of the current coverage radius. This approach first formalized [2] and implemented [3] by Clarkson was shown to only require $O(n \log \Delta)$ time for greedy permutations in doubling metrics by Har-Peled and Mendel [8]. The same analysis easily holds for approximately greedy permutations. The two key steps are that, one, the graph has constant degree on a net, and, two, the distance to a point is computed only a constant number of times before the coverage radius must go down by a factor of two. We call this structure the *cluster graph*.

Our algorithm uses a cluster graph and approximate range sampling to compute an approximate one-hop greedy permutation. Let $\alpha < 1$ be the hop parameter. Let $\varepsilon <= 1$ be the desired approximation factor. At each iteration, we add one point using the following steps. First, we use a heap to quickly find the existing point p whose cluster has the largest radius. The farthest point f in this cluster is the point that would be added in a pure greedy permutation. Let r be the distance from f to p. We compute an approximate range sample S in $\texttt{ball}(f, 2\alpha r)$ at scale εr. We can then compute approximate range counts for $\texttt{ball}(q, \alpha r)$ for each $q \in S$ in constant time by adding the weights of points in S, careful not to count points that are within αr of a previously added point. The previously added points that could be within this radius are all neighbors of p in the cluster graph, so there are only a constant number of them. There are a constant number of points in S and each takes constant time. We add the point with the largest count. In total, we get the following.

Theorem 4 *Let P be points in a metric space with constant doubling dimension. Let $\alpha < 1$ and $\varepsilon < 1$ be constants. Then, a $(1+\varepsilon)$-approximate α-one-hop greedy permutation of P can be computed in $O(n \log \Delta)$ time.*

5 A Proof of Concept

We integrated an implementation of the one-hop greedy permutations into our Python library for greedy permutations. It can be found on Github at `https://github.com/donsheehy/greedypermutation`. This library also includes standard algorithms for computing pure greedy permutations and its approximations.

We found that on small, low-dimensional examples, there appears to be a very slight improvement with the one-hop greedy permutations. The example in Figure 2 is typical. We set $\alpha = 1/3$ and took a uniform sample of points in a square. The figure show three different scales in the permutation at $k = 10$, $k = 25$, and $k = 50$. The graph in Figure 3 shows how the radii change with the number of points. Even in this simple example, the one-hop greedy permutation often has the smaller coverage radius.

It remains to see if the improvements attained with one-hop k-centers on large benchmark instances coming from TSP datasets [6] are realized by the one-hop greedy permutations.

6 Conclusions

We have generalized the one-hop k-center heuristic to define one-hop greedy permutations. We have also given an efficient algorithm to compute approximations in $O(n \log \Delta)$ time.

There are several future directions to consider. In our implementation, we used a standard greedy permutation to build the data structure for approximate range sampling. This is novel in that it does a kind of boot strapping from an ordinary greedy permutation to a one-hop greedy permutation. Another direction to consider is whether the final choice should really count points by weight or just count points in the approximate range sample. It seems that the latter approach, though farther from the one-hop k-center approach could be more stable to drastically varying density in the underlying points set.

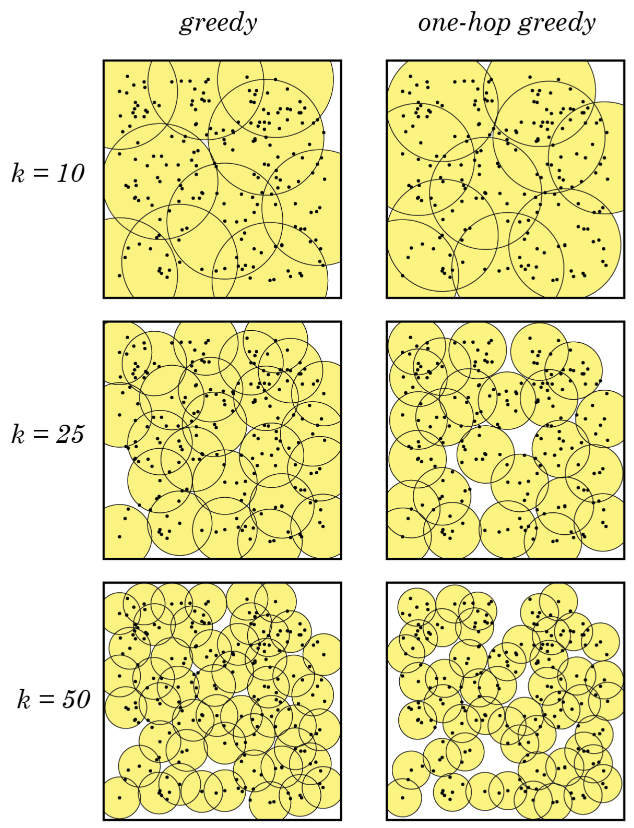

Figure 2: For points in the plane, there is little visual difference between the two cases except at the boundary. The one-hop greedy permutation tends to overshoot the boundary less.

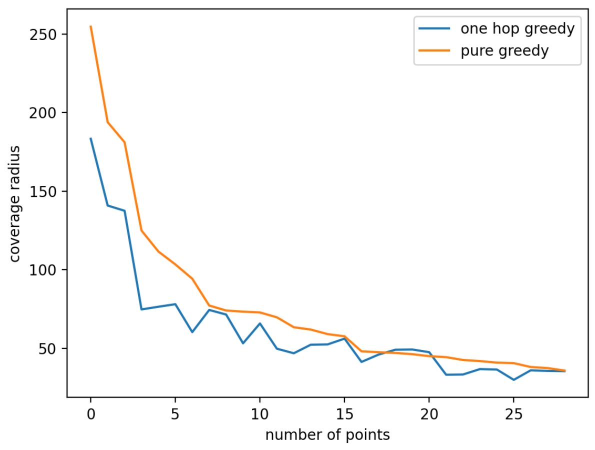

Figure 3: This is the graph of the coverage radii for the two approaches on the small example shown above.

References

[1] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.

[2] K. L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

[3] K. L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for ‘sb(s)‘. Preliminary version presented at ALENEX99, 2003.

[4] M. Dyer and A. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985.

[5] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Applications*, 35:2–19, 2006.

[6] J. Garcia-Diaz, J. Sanchez-Hernandez, R. Menchaca-Mendez, and R. Menchaca-Mendez. When a worse approximation factor gives better performance: a 3-approximation algorithm for the vertex k-center problem. *Journal of Heuristics*, 23(5):349–366, 2017.

[7] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[8] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[9] M. Izbicki and C. R. Shelton. Faster cover trees. In *Proceedings of the Thirty-Second International Conference on Machine Learning*, 2015.

[10] M. Jahanseir and D. Sheehy. Nettrees. Available from http://dx.doi.org/10.5281/zenodo.1409233, 2018.

[11] M. Jahanseir and D. R. Sheehy. Transforming hierarchical trees on metric spaces. In *Proceedings of the Canadian Conference on Computational Geometry*, 2016.

[12] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *SODA*, 2004.