Greedy Permutations and Finite Voronoi Diagrams

Oliver A. Chubet
North Carolina State University, Raleigh, NC, USA

Paul Macnichol
North Carolina State University, Raleigh, NC, USA

Parth Parikh
North Carolina State University, Raleigh, NC, USA

Donald R. Sheehy
North Carolina State University, Raleigh, NC, USA

Siddharth S. Sheth
North Carolina State University, Raleigh, NC, USA

Abstract

We illustrate the computation of a greedy permutation using finite Voronoi diagrams. We describe the neighbor graph, which is a sparse graph data structure that facilitates efficient point location to insert a new Voronoi cell. This data structure is not dependent on a Euclidean metric space. The greedy permutation is computed in $O(n \log \Delta)$ time for low-dimensional data using this method [4, 6].

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases greedy permutation, Voronoi diagrams

Digital Object Identifier 10.4230/LIPIcs.SoCG.2023.64

Category Media Exposition

Supplementary Material *Audiovisual*: `https://youtu.be/zMlpHV6Y1SM`

Funding This research was supported by the NSF under grant CCF-2017980.

1 Introduction

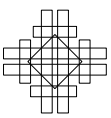
Given a finite metric space X, and a subset P, there is a natural decomposition of X into *Voronoi cells* defined for each $p \in P$ as

$$\mathsf{V}(p) := \{x \in X \mid \mathbf{d}(x,p) = \mathbf{d}(x,P)\}$$

This is a discrete analog of the Voronoi diagrams studied in geometry.

The most glaring difference between finite Voronoi diagrams and, say, Euclidean Voronoi diagrams is that there is no natural dual (i.e., no Delaunay triangulation) that connects neighboring cells. To store some local structure, a finite Voronoi diagram is endowed with a *neighbor graph* G. The requirement is that if there are two cells $\mathsf{V}(a)$ and $\mathsf{V}(b)$ such that inserting a new point $a' \in \mathsf{V}(a)$ will cause some points to move from $\mathsf{V}(b)$ to the newly formed $\mathsf{V}(a')$, then ab is an edge of G. In other words, G stores sufficient information to perform an incremental insertion of a new cell. The movement of points into a new cell is called *point location*. The neighbor graph also stores all the information needed to update itself efficiently after an insertion.

Greedy permutations, also known as farthest point traversals) are a standard way to provide a sequence of good samples of a metric space at different scales. The points are ordered so that each point is the farthest among the remaining points. We describe a simple, deterministic algorithm to compute greedy permutations.



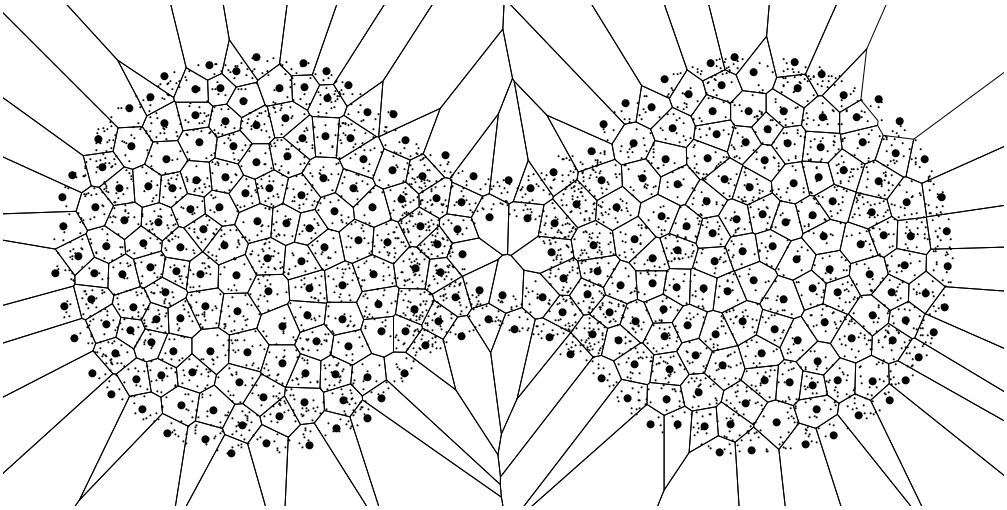


Figure 1 The geometric Voronoi cells cover the points in the cells of the finite Voronoi diagram.

2 Greedy Permutations

The greedy permutation, or the farthest first traversal, is a powerful heuristic used to approximate many hard problems since 1977 from TSP [5] to k-center clustering [3, 2]. Let $P = (p_0, \ldots, p_{n-1})$ be a finite sequence of points in a metric space with distance $\mathbf{d}$. The i^{th}-*prefix* is the set $P_i = \{p_0, \ldots, p_{i-1}\}$ containing the first i points of P. The sequence P is a *greedy permutation* if for all i,

$$\mathbf{d}(p_i, P_i) = \max_{p \in P} \mathbf{d}(p, P_i).$$

The point $p_0 \in P$ is the *seed* of the greedy permutation and may be chosen arbitrarily. A greedy permutation can be computed in $O(n \log \Delta)$ time for low-dimensional data [4, 6]. An $O(n \log n)$-time randomized approximation algorithm exists [4], but, to our knowledge, it has never been implemented.

3 Finite Voronoi Diagram

A *finite Voronoi diagram* consists of the cells $\mathsf{V}(p)$ and a neighbor graph G. It can be constructed incrementally as follows. It starts with a single cell containing all the points. We insert a point $p' \in \mathsf{V}(p)$ by constructing $\mathsf{V}(p')$ as a subset of the points $\mathsf{V}(q)$ where q is a neighbor of p. We then update the neighbor graph by selecting neighbors of $\mathsf{V}(p')$ among the neighbors of neighbors of p. There is a final step in which we prune excess edges that are clearly no longer needed in the neighbor graph. This maintains sparsity of the neighbor graph, resulting in efficient point location.

4 Point Location

An obvious disadvantage of finite Voronoi diagrams as compared to their geometric cousins is that the points of the cell must be explicitly enumerated and moved when there is an insertion. The analysis of Clarkson's algorithm by Har-Peled and Mendel [4] counts the number of times that each point is considered for moving. They show that by a standard

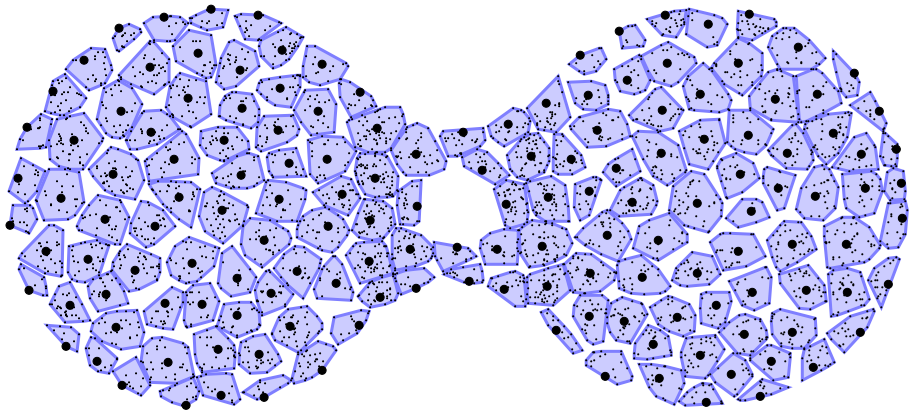


Figure 2 We depict the finite Voronoi cells by drawing the convex hull of the points it contains.

packing argument, only a constant number (i.e., $2^{O(d)}$) of points can touch a given point before the maximum distance of the farthest point from the center of a cell goes down by at least half. It follows immediately that each point is touched at most $2^{O(d)} \log \Delta$ times.

5 Incremental Construction of Finite Voronoi Diagrams

An incremental construction inserts the points one at a time, updating the cells and a neighbor graph. Let $(X, \mathbf{d})$ be a metric space and a let $P \subseteq X$ be a finite subset. The points of X are partitioned into cells $\mathsf{V}(p)$, where $p \in P$. The cells of a finite Voronoi diagram satisfy the following invariant.

The Cell Invariant: For all $p \in P$ and all $p' \in \mathsf{V}(q)$, $\mathbf{d}(p, p') = \min_{q \in P} \mathbf{d}(p', q)$.

The cell invariant says that if $a' \in \mathsf{V}(a)$, then a is a nearest neighbor of a' among the points of P. If we insert a' into the finite Voronoi diagram, then we add a' to P. The point a is called the *parent* of a'. To recover the cell invariant after an insertion, points from other cells are moved into the new cell in a process called *point location*. For a cell $\mathsf{V}(q)$, its *out-radius* is $R_q := \max_{a \in \mathsf{V}(q)} \mathbf{d}(q, a)$.

In addition to the cells, we also maintain the *neighbor graph* on P that allows point location to be performed locally. That is, there is an edge from a to b if inserting a new point into P from $\mathsf{V}(a)$ would require a point from $\mathsf{V}(b)$ to be moved (or vice versa). Formally, the neighbor graph maintains the following invariant.

The Neighbor Invariant: For all cells $\mathsf{V}(a)$ and $\mathsf{V}(b)$, if there exist points $a' \in \mathsf{V}(a)$ and $b' \in \mathsf{V}(b)$ such that $\mathbf{d}(a', b') < \mathbf{d}(b', b)$, then there is an edge from a to b.

Suppose we are inserting a' with parent a. The neighbor invariant directly implies that every point b' that must move into a new cell centered at a point a' is contained in a cell $\mathsf{V}(b)$, where b is a neighbor of a. Thus, point location only requires iterating over the cells of the neighbors of a to find all the points that will move into the newly constructed $\mathsf{V}(a')$.

To avoid excess distance computations, the stored neighbor graph contains some edges that are not strictly required by the neighbor invariant. After inserting a', the neighbors

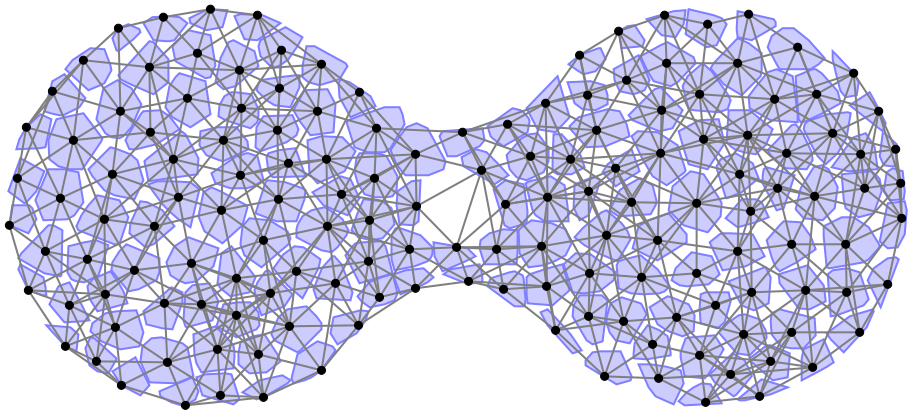


Figure 3 The neighbor graph during the finite Voronoi diagram construction.

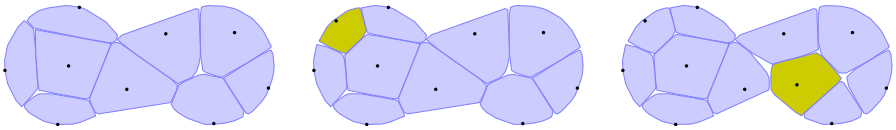


Figure 4 From left two right, we see two two iterations of the incremental finite Voronoi diagram construction.

are selected from point a, the neighbors of a, and the neighbors of the neighbors of a. One can simply add edges from a' to each of these points. Any graph that satisfies the neighbor invariant will continue to satisfy the neighbor invariant after this "neighbors of neighbors" update to support insertions.

As Clarkson observed [1], it is possible to use the triangle inequality to identify edges that are too long. Specifically, if the out-radii of two cells $\mathsf{V}(a)$ and $\mathsf{V}(b)$ are R_a and R_b respectively, then if the neighbor invariant requires an edge from a to b, then $\mathbf{d}(a,b) \leq R_a + R_b + \max\{R_a, R_b\}$. Edges longer than this length can be *pruned* from the neighbor graph.

So, inserting a new point a' with parent a into a finite Voronoi diagram has three steps:

1. Compute $\mathsf{V}(a')$ by moving points from cells of neighbors of a.
2. Compute the neighbors of a' by iterating over neighbors of neighbors of a.
3. Prune the edges incident to any point whose cell or neighbors changed.

By the analysis of Har-Peled and Mendel [4], the total point location cost is at most $2^{O(d)} n \log \Delta$. Storing points in a cell as a max-heap keyed by distance from the center gives ready access to the cell's out-radius. Heaps are updated for all cells which are modified in an iteration. If centers are added in a greedy fashion, it follows from a packing argument that the degree of a vertex in the neighbor graph is $2^{O(d)}$. So, the heap operations per iteration cost $2^{O(d)} \log n$. Updating the neighbor graph also takes $2^{O(d)}$ time per iteration. Therefore, a finite Voronoi diagram can be computed in $2^{O(d)} n \log \Delta$ time.

References

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

2 M.E Dyer and A.M Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985. doi:10.1016/0167-6377(85)90002-1.

3 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

4 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

5 Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

6 Donald R. Sheehy. greedypermutations. https://anonymous.4open.science/r/greedypermutation-C50B, 2020.