Higher-Order Voronoi Diagrams for Nearest and Farthest Points

Mejía Yáñez José Ehécatl

December 26, 2024

Abstract

Voronoi diagrams are fundamental tools in computational geometry, widely applied in fields such as computer graphics, artificial intelligence, and network design. This paper focuses on constructing orders of Voronoi diagrams for both the nearest and farthest points, exploring their geometric properties, and presenting an optimized approach for their computation that outperforms brute-force methods. The proposed method combines theoretical principles with efficient algorithmic strategies, enabling improved performance on large and complex data sets.

1 Introduction

Voronoi diagrams are quite common and widely used. Personally, I have encountered them in various forms in several competitive programming problems, in other algorithms presented in different articles, and they are also useful in geographic analysis related to data analysis. Therefore, in this brief article, we will explain in a simple manner how to construct a high-order Voronoi diagram based on the closest point and provide an idea of how to compute the farthest-point Voronoi diagrams.

2 Development

The farthest-point Voronoi diagram is similar to the nearest point version, but focuses on the farthest distance between a given point and all other points in the dataset. Constructing this diagram involves identifying the regions that are furthest from each point, rather than the nearest.

2.1 Nearest

The way to analyze how to construct the Voronoi diagram will be through the intersection of the half-planes for each point. This, as is well known, gives us the diagram mentioned, although with a complexity of $O(n^2 \log n)$ or $O(n^2 k)$. Here, k can be defined as the number of points that share a side in a Voronoi diagram between two points. This can be treated as a constant because the number of edges in the Delaunay triangulation is linear, which allows us to understand that the number of edges of a vertex is constant, and therefore its Voronoi neighbors as well.

2.1.1 Construction

To compute the second-order Voronoi diagram, given a region of the first-order diagram, we can divide this region into subregions for any other point using the same procedure mentioned previously. With this method, we could calculate the second-order diagram, but the complexity would be $O(n^3 \log n)$. To reduce this complexity, we can observe that the Voronoi region will only be divided by the neighbors of the point, thus reducing the complexity to $O(n^2 \log n + n \cdot k^2 \cdot \log k)$. We can repeat this idea for any region of any order and simply compute the next regions of the following order. This results in a final complexity of $O(n \cdot k^d \cdot \log k)$.

2.1.2 Algorithm

Input: A set P of data points (of size n) and the number of the order k.
Output: The vector V_i of regions per point $i = 1, 2, \ldots, n$.

Functions and variables used:

- $VD(R, P)$: A function to construct the Voronoi diagram for the region R and the points P.
- $ID(R, p)$: When called, a function finds the neighbors of the point p with the region R.
- $init$: Region indicating the universe.
- $Regions[][]$: An array $2D$ of regions to store the Voronoi regions (initially empty), where the first dimension corresponds to each point (of size n) and the second to the number of regions per point.
- $Neighbors[][][]$: An array $3D$ to store the neighbors of regions (initially empty) where the first dimension represents each point (of size n), the second corresponds to the regions per point, and the third indicates the index of the neighbors.
- $Temp[][]$: A temporary array $2D$ of regions (initially empty), where the first dimension corresponds to the number of points (of size n) and the second to the number of regions per point.
- i: a temporary variable (initially one).
- j: a temporary variable (initially zero).
- l: a temporary variable (initially zero).

Steps:

```
1. Regions = $VD(init, S)$

2. Set $i \leftarrow 1$.

3. For $j = 0$ to $|P|$:
    • For $l = 0$ to $|Regions[j]|$:
        – Neighbors[$j$] $\leftarrow$ Neighbors[$j$] $\cup$ $ID$(Regions[$i$][$j$], $P[j]$)

4. For $j = 0$ to $|P|$:
    • For $l = 0$ to $|Regions[j]|$:
        – Temp[$j$] $\leftarrow$ Temp[$j$] $\cup\, VD$(Regions[$i$][$j$], $Neighbors[j][l]$)

5. Regions $\leftarrow$ Temp ;

6. Neighbors && Temp $\leftarrow$ ;

7. If $i + 1 = k$, go to step 8; otherwise:
    • $i \leftarrow i + 1$, go to step 3.

8. Return Regions.
```

2.2 Farthest

To calculate the Voronoi region of the farthest point, a simple observation can be made: if we change the orientation of the half-planes (choosing the right side) and compute the intersection, we will obtain the corresponding diagram. Similarly, it can be computed in $O(n^2 \log n)$ complexity. Another observation is that only those points that belong to the convex hull will have a first-order Voronoi region of the farthest point. This significantly reduces the complexity if the origin of the points tends to follow a random order.

2.2.1 Construction

To construct the second-order Voronoi region of the farthest point, the regions can be divided between points that share a side between their regions, and similarly, they will have a constant number of neighbors (k). However, as mentioned before, only points that belong to the convex hull are considered up to this point. When calculating for a point, it is necessary to also consider the points that would belong to the new convex hull (excluding the point in question). This can be computed in linear complexity for a single point, yielding a complexity of $O(n^2 \log n + n \cdot k^2 \log k)$.

3 Conclusion

In this paper, we have explored higher-order Voronoi diagrams for both the nearest and farthest points, presenting methods to compute them efficiently. By leveraging observations about the convex hull and the properties of Voronoi regions, we were able to propose optimizations that reduce computational complexity compared to traditional brute-force approaches. The algorithms discussed allow for the construction of Voronoi diagrams of higher orders with a manageable time complexity, making them suitable for large and complex datasets. Future work can extend these methods to more general cases or investigate further optimizations to handle even larger-scale problems.