Algorithms for neighborhood searches: The Voronoi diagram approach

Course in Computational Geometry

BORELLI ROBERTO
borelli.roberto@spes.uniud.it

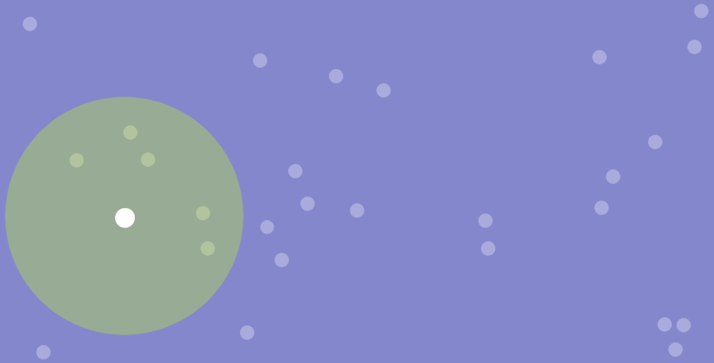
University of Udine

Abstract

In this presentetion I consider the neighborhood selection problem which I started to study in my bachelor's thesis and in a previous work[1]. I briefly recap the main results which I have already studied, namely, I define the problem in a precise way and I show a straightforward solution, then I review the common algorithmic approaches toghether with complexity analisies and I show the curse of dimensionality problem. Then I consider the Voronoi diagram approach to solve the neighborhood selection problem. First, I present the algorithm which shows how to properly use the data structure to implement an efficient solution, and second, I compare this method by the previous ones both with theoretical and experimental results.

[1] R. Borelli, A. Dovier, and F. Fogolari (2022). "Data Structures and Algorithms for k-th Nearest Neighbours Conformational Entropy Estimation". In: *Biophysica* 2.4, pp. 340–352. DOI: 10.3390/biophysica2040031.

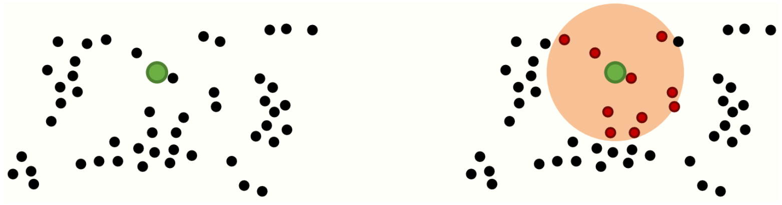
Contents



Problem overview

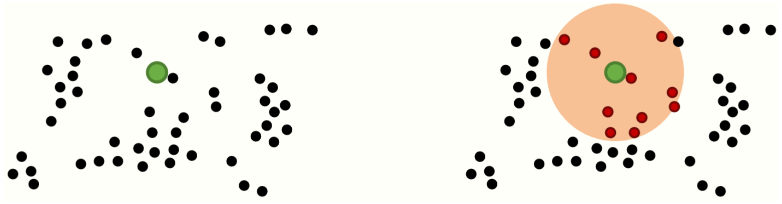
Problem formulation

- ALL-MNN: Given n point in a d dimensional space, find for each point p its m nearest neighbors



Problem formulation

- ALL-MNN: Given n point in a d dimensional space, find for each point p its m nearest neighbors



- The problem has a wide range of real applications
  - Conformational entropy estimation on biomolecular systems
  - Machine learning

Problem formulation

- I propose a very simple formulation (which I didn't found in the literature) based on integer linear programming
- It evidences the *optimization* nature of the problem
- It's usefull to define precisely the broblem
- It doesn't touch the geometry of the problem and it's very inefficient

Problem formulation

Le S the be the inpunt set $S = \vec{p_1}, \ldots, \vec{p_n}$ and let m be the number of neighbors $0 < m < n$

We call $N_1, \ldots, N_n$ the sets of m neighbors for $\vec{p_1}, \ldots, \vec{p_n}$.

Problem formulation

Le S the be the inpunt set $S = \vec{p_1}, \ldots, \vec{p_n}$ and let m be the number of neighbors $0 < m < n$
We call $N_1, \ldots, N_n$ the sets of m neighbors for $\vec{p_1}, \ldots, \vec{p_n}$. Let us introduce $n^2 - n$ binary variables of the form $x_{ij}, i \neq j$ with the meaning

$$x_{ij} = \begin{cases} 1 & \text{if } \vec{p_j} \in N_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall i, j \in [n], i \neq j \tag{1}$$

Problem formulation

Le S the be the inpunt set $S = \vec{p_1}, \ldots, \vec{p_n}$ and let m be the number of neighbors $0 < m < n$
We call $N_1, \ldots, N_n$ the sets of m neighbors for $\vec{p_1}, \ldots, \vec{p_n}$. Let us introduce $n^2 - n$ binary variables of the form $x_{ij}, i \neq j$ with the meaning

$$x_{ij} = \begin{cases} 1 & \text{if } \vec{p_j} \in N_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall i, j \in [n], i \neq j \tag{1}$$

The integer programming model is now the following

$$\min \sum_{i \in [n]} \sum_{j \in [n] \setminus \{i\}} (x_{ij} dist(\vec{p_i}, \vec{p_j})) \tag{2}$$

subject to:

$$\sum_{j \in [n] \setminus \{i\}} x_{ij} = m \qquad \forall i \in [n] \tag{3}$$

Lower bound

- First lower bound: we have to read the input and print the output:

$$\Omega(nd) + \Omega(nmd) \tag{4}$$

Lower bound

- First lower bound: we have to read the input and print the output:

$$\Omega(nd) + \Omega(nmd) \tag{4}$$

- I can improve the lower bound by a reduction through ELEMENT-UNIQUENESS:

ELEMENT-UNIQUENESS $\preceq$ CLOSEST-PAIR $\preceq$ ALL-1NN

Lower bound

ELEMENT-UNIQUENESS $\preceq$ CLOSEST-PAIR $\preceq$ ALL-1NN

1. Start from a list of numbers (points in one dimension) S on which we want to check the uniqueness property

Lower bound

ELEMENT-UNIQUENESS $\preceq$ CLOSEST-PAIR $\preceq$ ALL-1NN

1. Start from a list of numbers (points in one dimension) S on which we want to check the uniqueness property
2. For each point p_i, obtain the nearest neighbor q_i by solving the ALL-1NN problem

Lower bound

ELEMENT-UNIQUENESS $\preceq$ CLOSEST-PAIR $\preceq$ ALL-1NN

1. Start from a list of numbers (points in one dimension) S on which we want to check the uniqueness property
2. For each point p_i, obtain the nearest neighbor q_i by solving the ALL-1NN problem
3. Compute the vector $\vec{d}$ such that $d_i = dist(p_i, q_i)$

Lower bound

ELEMENT-UNIQUENESS $\preceq$ CLOSEST-PAIR $\preceq$ ALL-1NN

1. Start from a list of numbers (points in one dimension) S on which we want to check the uniqueness property
2. For each point p_i, obtain the nearest neighbor q_i by solving the ALL-1NN problem
3. Compute the vector $\vec{d}$ such that $d_i = dist(p_i, q_i)$
4. Look for the index j such that d_j is minimum and so we have solved the CLOSEST-PAIR problem

Lower bound

ELEMENT-UNIQUENESS $\preceq$ CLOSEST-PAIR $\preceq$ ALL-1NN

1. Start from a list of numbers (points in one dimension) S on which we want to check the uniqueness property
2. For each point p_i, obtain the nearest neighbor q_i by solving the ALL-1NN problem
3. Compute the vector $\vec{d}$ such that $d_i = dist(p_i, q_i)$
4. Look for the index j such that d_j is minimum and so we have solved the CLOSEST-PAIR problem
5. To solve the ELEMENT-UNIQUENESS problem, check if $d_j = 0$

Lower bound

ELEMENT-UNIQUENESS $\preceq$ CLOSEST-PAIR $\preceq$ ALL-1NN

1. Start from a list of numbers (points in one dimension) S on which we want to check the uniqueness property
2. For each point p_i, obtain the nearest neighbor q_i by solving the ALL-1NN problem
3. Compute the vector $\vec{d}$ such that $d_i = dist(p_i, q_i)$
4. Look for the index j such that d_j is minimum and so we have solved the CLOSEST-PAIR problem
5. To solve the ELEMENT-UNIQUENESS problem, check if $d_j = 0$

- Steps 3 and 4 take linear time
- Solving the ELEMENT-UNIQUENESS takes $\Omega(n \log n)$ time
- Solving the ALL-1NN problem in $o(n \log n)$ implies a contraddiction

Lower bound

Some details on the reduction:

- Note that Element-Uniqueness is a decision problem while the other 2 are functional problems. So we are referring to the more general definition of reduction between functional problems
- One (implicit) key assumption: the ALL-1NN takes as input a list (and not a set) of points. Otherwise the reduction doesn't make sense: a set always satisfies the uniqueness property
- In real world situations S is always a set
- By implementing the classical algorithms I observed that the code doesn't change much if we allow or not S to be a list
- To be precise I should also prove that $\text{ALL-1NN}_{\text{lists}} \preceq \text{ALL-1NN}_{\text{sets}}$

The naive approach

- If we consider just one dimension we can easily come up with an optimal algorithm which takes linear space and $\Theta(n \log n + mn)$ time

The naive approach

- If we consider just one dimension we can easily come up with an optimal algorithm which takes linear space and $\Theta(n \log n + mn)$ time
- If we consider more dimensions:

```
NAIVE-ALL-MNN(S)
    1. For each point $p$ in $S$, compute the vector $\vec{d_p}$ which contains the distances between $p$ and all the other points
    2. Order each vector $\vec{d_p}$
    3. Return the $m$ nearest neighbors for each $p$
```

The naive approach

- If we consider just one dimension we can easily come up with an optimal algorithm which takes linear space and $\Theta(n \log n + mn)$ time
- If we consider more dimensions:

NAIVE-ALL-MNN(S)

```
1. For each point $p$ in $S$, compute the vector $\vec{d_p}$ which contains the distances between $p$ and all the other points
2. Order each vector $\vec{d_p}$
3. Return the $m$ nearest neighbors for each $p$
```

- Ordering (which takes $\Theta(n \log n)$) is not really necessary. After computing $\vec{d_p}$ we could just select (in linerar time) the m-th smallest distance
- So in the general case we have an algorithm which always takes linear space and $O(dn^2)$ time
- ... which of course is unaceptable

Classical approaches and results

In the literature we can find algorithms based on tree strcutures (like VP trees, K-D trees, Quad trees, ...) which are made up of the following steps:

Classical approaches and results

In the literature we can find algorithms based on tree strcutures (like VP trees, K-D trees, Quad trees, ...) which are made up of the following steps:

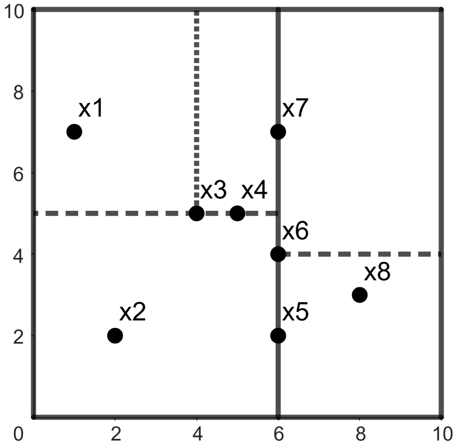
- **Preprocessing:** In this step we prepare a data structure (for example a tree) which tries to partition the space in small regions containing points. For each point in S the we then invoke a query algorithm which should take advantage of the structure created.

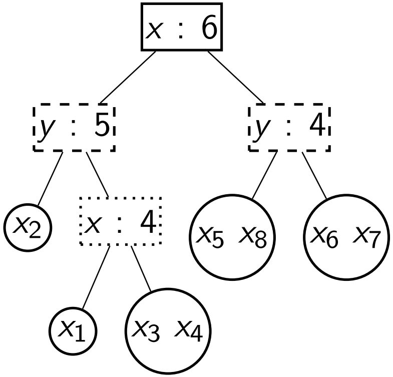
Classical approaches and results

In the literature we can find algorithms based on tree strcutures (like VP trees, K-D trees, Quad trees, ...) which are made up of the following steps:

- **Preprocessing:** In this step we prepare a data structure (for example a tree) which tries to partition the space in small regions containing points. For each point in S the we then invoke a query algorithm which should take advantage of the structure created.
- **Querying:** Given a point $p \in S$ we visit the data structure (possibly a small part of it) in order to find the m nearest neighbors of p.

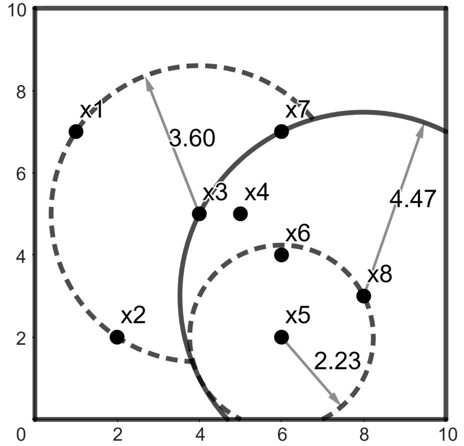
Classical approaches and results: The K-D Tree

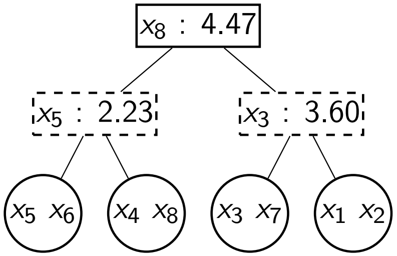




- Each internal node corresponds to a binary test which involves the projection operation
- At each node the set of points is split into two almost-balanced sets

Classical approaches and results: The VP Tree



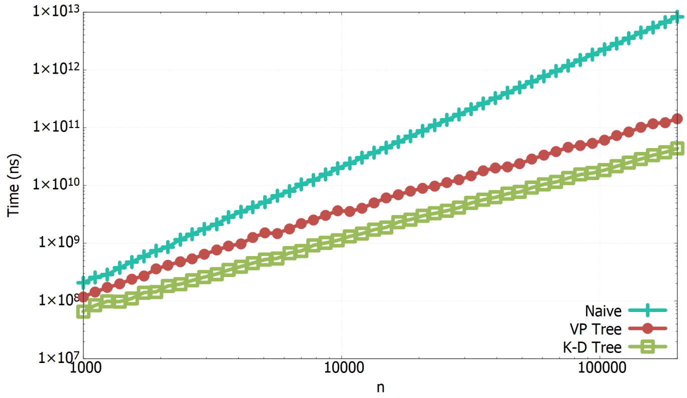


- Similar idea to K-D Trees but more general
- Each internal node is no more associated with a projection but only involves the dicotomy far/near with respect to the vantage point

Classical approaches and results: Costs

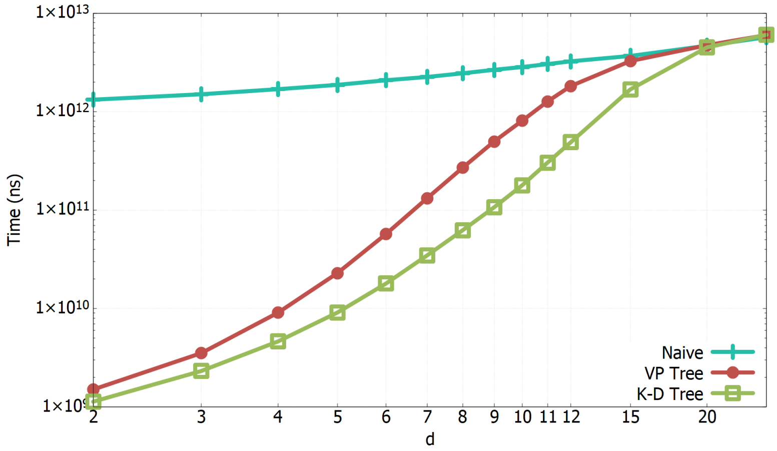
- The preprocessing steps is in some sense *easy* and takes $\Theta(n \log n)$ time
- The query should take $\Theta(\log n)$ time, but with currently-avalaible methods this happens only in the avarage case
- We get the worst case when d is sufficiently large. In such case the query algorithm takes linear time (we essentialy visit the entire tree structure) and the entire ALL-MNN algorithm degenretes to an exhaustive search which takes quadratic time
- This problem is known in the literature as **curse of dimensionality** and we currently don't know if the problem admits an optimal solution when the number of dimension increases

Classical approaches and results: Comparison



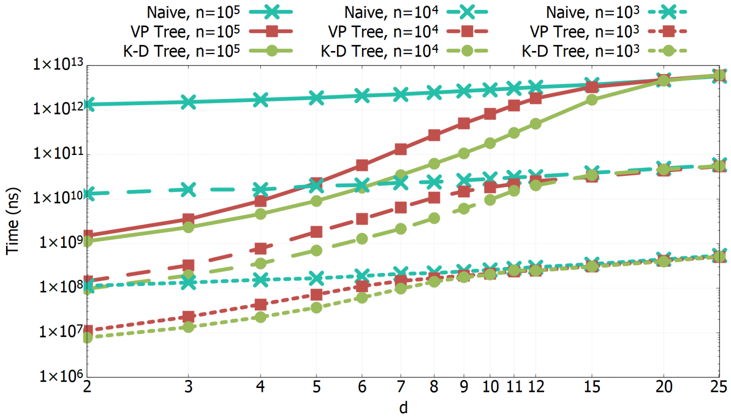
- Whith $d = 6$ dimensions and $m = 20$ VP Tree and the K-D tree performances are the same expect for a multiplicative constant
- Recall: In double logarithmic scale different lines inclinations means different polinomials

Classical approaches and results: Curse of dimensionality



- While working with $n = 100000$ points and $m = 20$ neighbors per point, advantages of tree-based methods are completely lost at $d = 20$

Classical approaches and results: Curse of dimensionality



- We observed that *curse of dimensionality* tends to attenuate when we increase n
- We also observed that the tree-based approaches degenerate in an exhaustive search when $d \in \Omega(\log n)$

The Voronoi diagram approach

The Voronoi diagram

Optimal solution to ALL-1NN in 2d

The *k*-th order Voronoi diagram

Optimal solution to ALL-MNN in 2d

Comparison

Comparison

Conclusion

Concluding remarks

Bibliography I

Borelli, R., A. Dovier, and F. Fogolari (2022). "Data Structures and Algorithms for k-th Nearest Neighbours Conformational Entropy Estimation". In: *Biophysica* 2.4, pp. 340–352. DOI: 10.3390/biophysica2040031.

Chazelle, B. and H. Edelsbrunner (1987). "An Improved Algorithm for Constructing kth-Order Voronoi Diagrams". In: *IEEE Transactions on Computers* C-36.11, pp. 1349–1354. DOI: 10.1109/TC.1987.5009474.

Lee, D.-T. (1982). "On k-Nearest Neighbor Voronoi Diagrams in the Plane". In: *IEEE Transactions on Computers* C-31.6, pp. 478–487. DOI: 10.1109/TC.1982.1676031.

Bibliography II

Shamos, M. I. and D. Hoey (1975). “Closest-point problems”. In: *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pp. 151–162. DOI: 10.1109/SFCS.1975.8.