CSE520 Computational Geometry
Lecture 1
Fixed-Radius Near Neighbors Searching

Antoine Vigneron

Ulsan National Institute of Science and Technology

March 5, 2020

Introduction

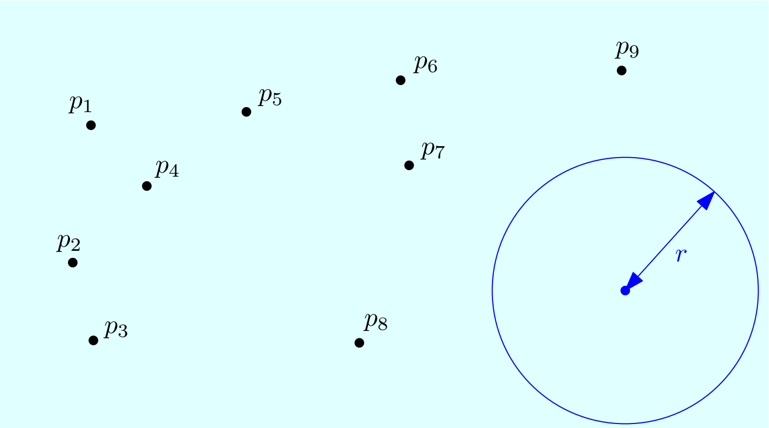
Reference:

- Dave Mount's lecture notes, lecture 2.
- Wikipedia.

An old problem.

- Applications: Air traffic control, molecular dynamics ...
- Simple and very fast algorithm.

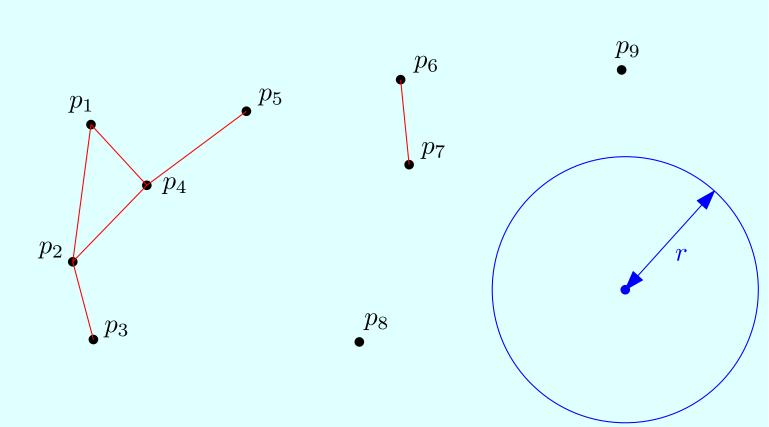
Problem Formulation



INPUT: a set of points $P = \{p_1, \ldots, p_n\}$ and a radius r.

We assume we are in *fixed dimension*, that is, $P \subset \mathbb{R}^d$ with $d = O(1)$.

Problem Formulation



OUTPUT: all pairs $(p_i, p_j) \in P^2$ such that the distance $|p_i p_j|$ is at most r.

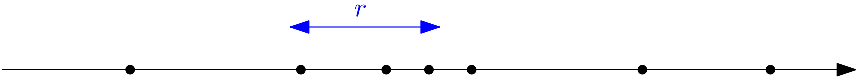
Here, the output is: $(p_1, p_2), (p_1, p_4), (p_2, p_3), (p_2, p_4), (p_4, p_5), (p_6, p_7)$.

Brute-Force Approach

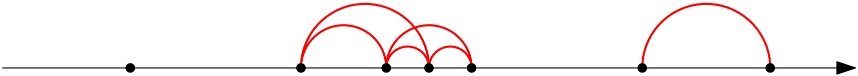
- For any two points p_i, p_j, we can compute $|p_i p_j|^2$ in constant time.
  - Example: if $d = 2$, $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$, then
$$|p_i p_j|^2 = (x_i - x_j)^2 + (y_i - y_j)^2.$$
- So we can just check for each pair (i, j) whether $|p_i p_j|^2 \leqslant r^2$.
  - Running time? $\Theta(n^2)$.
- Can we do better than $O(n^2)$?
  - Let k denote the *output size*: k is the number of pairs (p_i, p_j) such that $|p_i p_j| \leqslant r$.
  - $o(n^2)$ is impossible, because in the worst case
$$k = \binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2).$$
  - So we will give an *output sensitive* algorithm that runs in time $\Theta(n + k)$.

The One-Dimensional Case

- **INPUT**: $x_1, \ldots, x_n \in \mathbb{R}$ and $r > 0$.



- **OUTPUT**: the k pairs (x_i, x_j) such that $|x_i - x_j| \leqslant r$.



The One-Dimensional Case

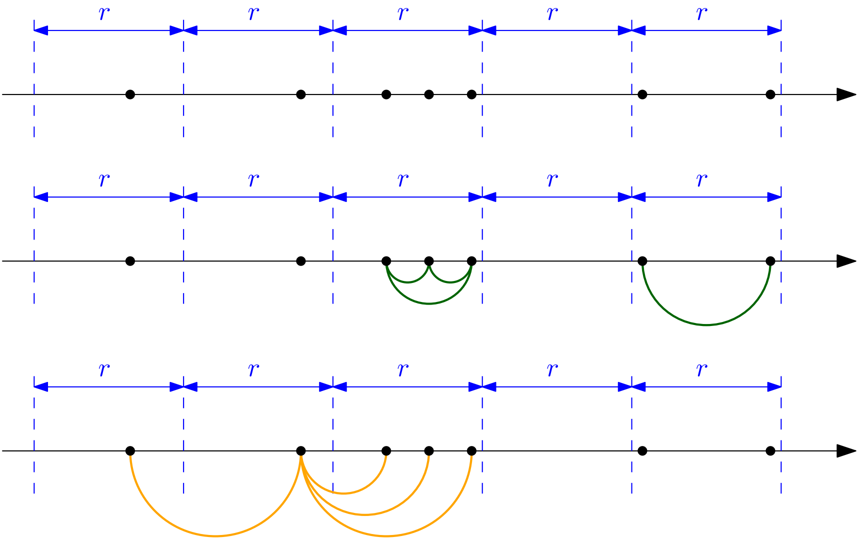
Exercise

Find a simple $O(n \log(n) + k)$ algorithm.

Solution

- Sort the numbers in increasing order.
- Traverse this sorted list from left to right.
- Pseudocode:

A Bucketing Approach



A Bucketing Approach

- We partition $\mathbb{R}$ using intervals (buckets) of size r:

$$\ldots, [-3r, -2r), [-2r, -r), [-r, 0), [0, r), [r, 2r), [2r, 3r) \ldots$$

- So each interval $[br, (b+1)r)$ corresponds to some integer b.
- A point $x \in \mathbb{R}$ is in bucket $b = \lfloor x/r \rfloor$.
  - So we can determine the bucket containing x in $O(1)$ time.
- Observation: We may only have $|x_i - x_j| \leqslant r$ if
  - x_i and x_j lie in the same bucket b ...
  - or x_i is in bucket b and x_j is in bucket $b+1$.
  - or x_j is in bucket b and x_i is in bucket $b+1$.
- So our algorithm only checks pairs in the same bucket, or in adjacent buckets.
- We store the nonempty buckets in a *dictionary* data structure:
  - It allows deletion, insertion, lookup.

Pseudocode

1D-Bucketing algorithm

Create an empty dictionary D.
for $i \leftarrow 1, n$ **do**
    $b_i \leftarrow \lfloor x_i / r \rfloor$
    if $b_i \notin D$ **then**
        insert bucket b_i into D
    insert x_i into bucket b_i
for each nonempty bucket b **do**
    report all pairs (x_i, x_j) in bucket b
    report all pairs $(x_i, x_j) \in b \times (b+1)$ such that $x_i - x_j \leqslant r$

Analysis

Theorem

The bucketing algorithm runs in time $O(nT(n) + k)$, where $T(n)$ is the time needed for one dictionary operation.

- So it is $O(n \log(n) + k)$ if we implement the dictionary with a balanced binary search tree,
- and is (randomized) expected time $O(n + k)$ using a hash table.

- We now prove the theorem.
- First observe that we make $O(n)$ dictionary operations so they take $O(nT(n))$ time.
- Line 8 takes time $O(\sum_{b \in \mathbb{Z}} n_b^2)$ and line 9 takes time $O(\sum_{b \in \mathbb{Z}} n_b n_{b+1})$ where n_b is the number of points in bucket b.
- So Lines 8 and 9 take time $O(T')$ where

$$T' = \sum_{b \in \mathbb{Z}} n_b^2 + \sum_{b \in \mathbb{Z}} n_b n_{b+1}.$$

Analysis

- For any $x, y \in \mathbb{R}$ we have $xy \leqslant \frac{x^2+y^2}{2}$.
- Therefore

$$T' \leqslant \sum_{b\in\mathbb{Z}} n_b^2 + \sum_{b\in\mathbb{Z}} \frac{n_b^2 + n_{b+1}^2}{2}$$
$$= \frac{3}{2}\sum_{b\in\mathbb{Z}} n_b^2 + \frac{1}{2}\sum_{b\in\mathbb{Z}} n_{b+1}^2 = 2\sum_{b\in\mathbb{Z}} n_b^2 = 2S.$$

where $S = \sum_{b\in\mathbb{Z}} n_b^2$.

Analysis

- As all numbers in b are at distance less than r from each other, the number of output pairs in bucket b is

$$k_b = \frac{n_b(n_b - 1)}{2},$$

so

$$S = \sum_{b \in \mathbb{Z}} 2k_b + n_b = 2k + n.$$

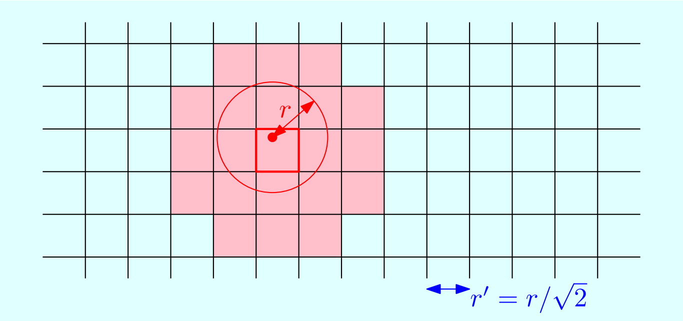
- It follows that $T' \leqslant 4k + 2n$ hence $T' = O(n + k)$.
- Therefore the total running time is $O(nT(n) + T') = O(nT(n) + n + k) = O(nT(n) + k)$.

Two-Dimensional Case

- The bucketing approach applies.
- The integer pair (b_x, b_y) correspond to the square

$$[b_x r', (b_x + 1)r') \times [b_y r', (b_y + 1)r').$$

  where $r' = r/\sqrt{2}$.
- For a bucket $b = (b_x, b_y)$, we only need to check pairs $(p, p') \in b \times b'$, where b' is one of the 21 buckets pictured below.



Higher Dimension

- Bucketing applies to any dimension d.
- When $d = O(1)$, it has the same time bound: $O(nT(n) + k)$.
- But in general, it is exponential in d.
- In high dimension, the best known time bounds are not much better than brute force.