Project 2 – Finding the Closest Pair

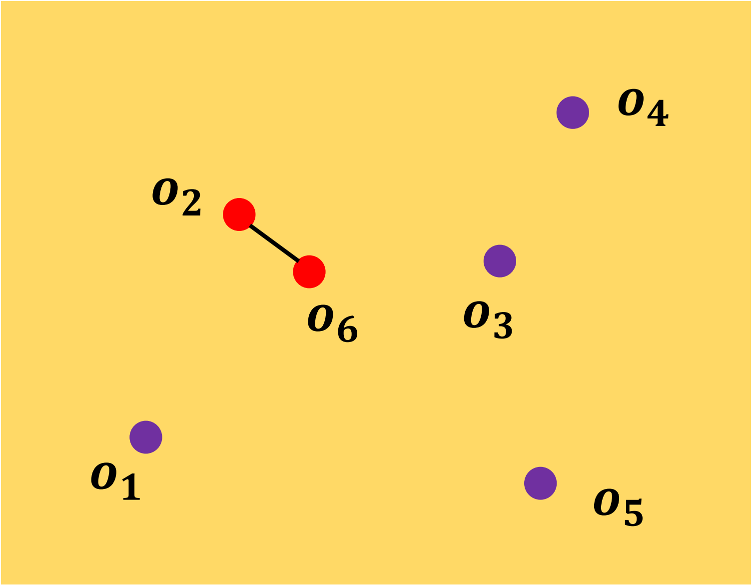
Qiang Huang
huangq25@mail2.sysu.edu.cn
School of Data and Computer Science
Sun Yat-Sen University

Introduction

- The purpose of the project:
  - Implement an approximation method with random projection to find the closest pair of a set of objects in high-dimensional space;
  - Use Euclidean distance as the distance metric.

Closest Pair Problem

- Given n objects $\{o_1, o_2, \ldots, o_n\}$ in d-dimensional Euclidean space R^d, the closest pair problem is to find a pair of objects o_i and o_j that minimizes $\|o_i - o_j\|$.



The closest pair is $\{o_2, o_6\}$

Euclidean Distance

- Given two objects $\boldsymbol{o_i = (o_{i1}, o_{i2}, \ldots, o_{id})}$ and $\boldsymbol{o_j = (o_{j1}, o_{j2}, \ldots, o_{jd})}$ from R^d, the Euclidean distance between o_i and o_j is computed as follows:

$$\|o_i - o_j\| = \sqrt{\sum_{k=1}^{d} \left(o_{ik} - o_{jk}\right)^2}$$

- An Example:
  - Suppose $o_i = (4, 2)$ and $o_j = (1, 6)$, then
  - $\|o_i - o_j\| = \sqrt{(4-1)^2 + (2-6)^2} = \sqrt{3^2 + 4^2} = 5$

Overview of the Algorithm

- Step 1: Make a random projection to project the n data objects from R^d to m random lines $\{S_1, S_2, \dots, S_m\}$;

- Step 2: For each random line S_i, we find the closest pair cp as a candidate; (closest pair in R^d → closest pair in 1-Dimension)

- Step 3: We compute the Euclidean distance for the m candidates (each candidate is the closest pair of S_i), and return the closest pair among the m candidates;

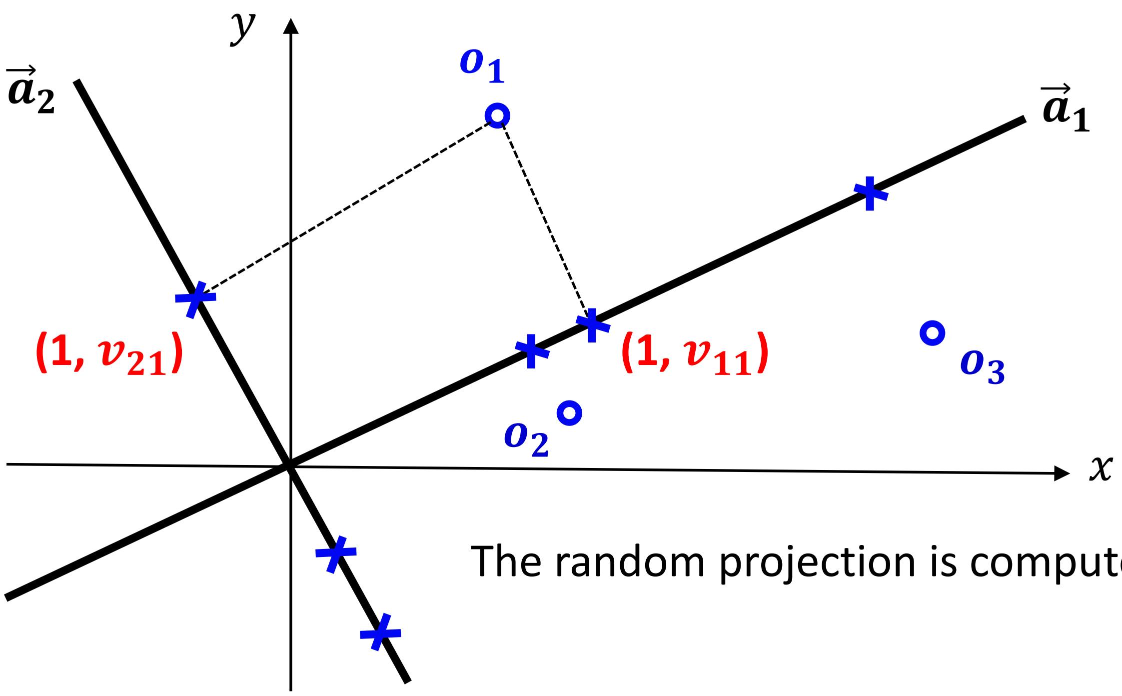
Overview of the Algorithm

Algorithm 1: Closest-Pair

```
Input: a database D of n data objects
       {o_1, o_2, ..., o_n} and the number of
       random projections m.
Output: the closest pair cp.
{S_1, S_2, ..., S_m} = Random-Projection();
Let cp be the closest pair, cp = null;
Let min be the closest distance, min = ∞;
for i = 1 to m do
    {δ, cp_i} = Closest-Pair-Line(S_i);
    Compute Euclidean distance dist for the
    closet pair cp_i found in S_i;
    if dist < min then
        min = dist;
        cp = cp_i;
return cp;
```

Random Projection

- Project all n objects onto the m random lines $S_1, S_2, \ldots, S_m$;



The random projection is computed by $v_{ij} = \vec{a}_i \cdot \vec{o}_j$

Random Projection

- Generate random projection vectors $\vec{a}_1, \vec{a}_2, \dots, \vec{a}_m$
  - Each $\vec{a}_i$ is a d-dimensional vector where each component is drawn from standard Normal distribution $N(0,1)$.

- There is a Box-Muller method to generate a random variable from $N(0,1)$.
  - Suppose U_1 and U_2 are random variable distributed uniformly from $(0,1)$;
  - Two independent random variables X_1 and X_2 from $N(0,1)$ are generated as follows:
  - $X_1 = \sqrt{-2\log U_1}\cos(2\pi U_2)$
  - $X_2 = \sqrt{-2\log U_1}\sin(2\pi U_2)$

Random Projection

Algorithm 2: Random-Projection

Input: a database D of n data objects $\{o_1, o_2, \ldots, o_n\}$ and the number of random projections m.

Output: $\{S_1, S_2, \ldots, S_m\}$.

Generate a set of m random projection vectors $\{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_m\}$;

$S_j = \emptyset$ for $j \in \{1, 2, \ldots, m\}$;

for $i = 1$ *to* n **do**

    for $j = 1$ *to* m **do**

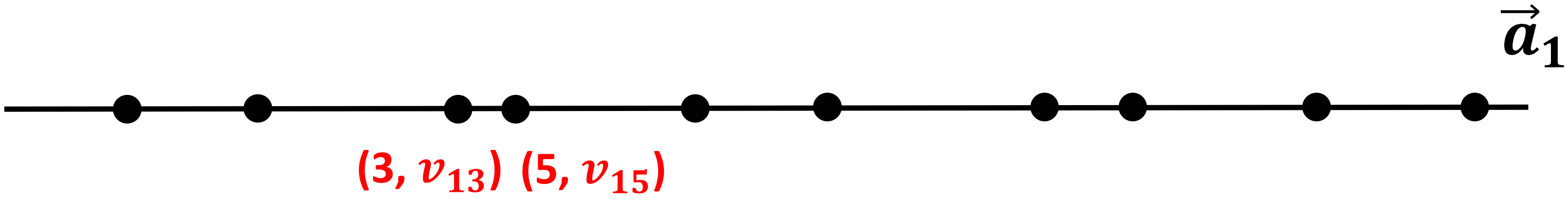
        $c_{i,j} = i$;

        $v_{i,j} = \vec{a}_j \cdot \vec{o}_i$;

        $S_j = S_j \cup (c_{i,j}, v_{i,j})$;

return $\{S_1, S_2, \ldots, S_m\}$;

Closest Pair in 1-Dimension

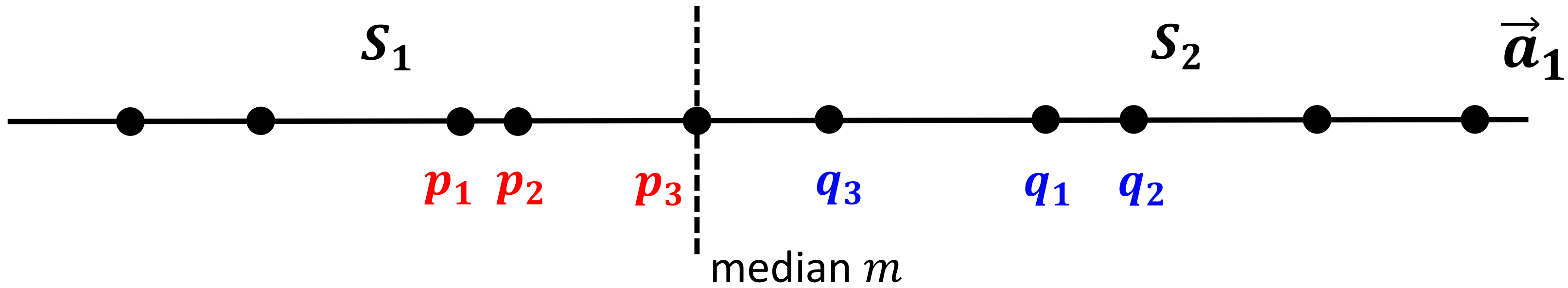


- 1D problem can be solved in $O(n \log n)$ via sorting (i.e. quick sort).

- We can sort all objects on the line, and find the closest pair with a linear scan.
  - i.e., the closest pair is $\{o_3, o_5\}$ for the random line $\vec{a}_1$

- However, can we find the closest pair during the sorting?

1-Dimension Divide and Conquer

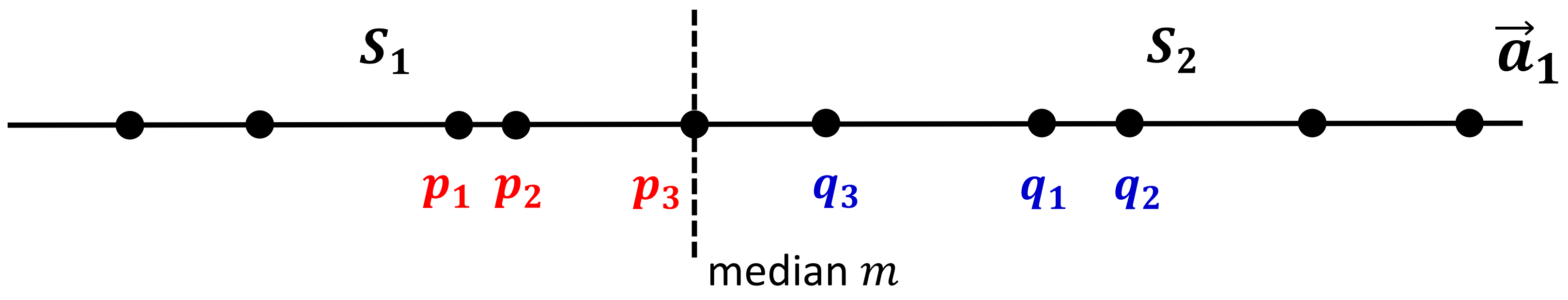


- Yes, use divide-and-conquer!

- Divide:
  - Divide all the objects into two sets S_1 and S_2 by a **median $\boldsymbol{m}$** so that $p < q$ for all $p \in S_1$ and $q \in S_2$;

  - Recursively compute the closest pair $\{p_1, p_2\}$ in S_1 and $\{q_1, q_2\}$ in S_2.

1-Dimension Divide and Conquer



- Conquer:
  - Let δ be the smallest separation found so far:

$$\delta = \min(|p_2 - p_1|, |q_2 - q_1|)$$

  - The closest pair is $\{p_1, p_2\}$, or $\{q_1, q_2\}$, or $\{p_3, q_3\}$.
  - In 1-Dimension, p_3 is the rightmost object of S_1 and q_3 is the leftmost object of S_2 (Why?)

1-Dimension Divide and Conquer

Algorithm 3: Closest-Pair-Median(S)

Input: a set S of n data objects on a line

Output: the samllest separation δ and the closest pair cp

if $|S| = 1$ **then**
    return $\{\delta = \infty, cp = null\}$;

if $|S| = 2$ **then**
    if $p_1 > p_2$ **then** swap(p_1, p_2);
    return $\{\delta = |p_2 - p_1|, cp = \{p_1, p_2\}\}$;

Let $m = median(S)$;
Divide S into S_1 and S_2 at m;
$\{\delta_1, cp_1\}$ = Closest-Pair-Median(S_1);
$\{\delta_2, cp_2\}$ = Closest-Pair-Median(S_2);
$\{\delta_{12}, cp_{12}\}$ is the result across the cut;
return $\{\delta, cp\} =$ $\min(\{\delta_1, cp_1\}, \{\delta_2, cp_2\}, \{\delta_{12}, cp_{12}\})$;

Finding the Median

- We can use the divide-and-conquer to find the median in $O(n)$.

- Given a set of objects S
- Select(S, k):
  - choose a splitter $a_i \in S$ **uniformly at random**; (add randomization)
  - Split S into two sub sets S^- and S^+; (divide)
    - For all $a_j \in S^-$, $a_j < a_i$;
    - For all $a_j \in S^+$, $a_j > a_i$;

  - If $|S^-| = k - 1$ then a_i is the answer; (conquer)
  - Else if $|S^-| \geq k$ then Select(S^-, k);
  - Else Select($S^+, k - 1 - |S^-|$);

Finding the Median

- An Example:
- Suppose $S = \{4, 1, 7, 3, 5, 2, 6\}$ and $\underline{k = 6}$.

- Find a splitter $a_i = 3$, then $S = \{2, 1, 3, 4, 5, 7, 6\}$,
- Since $|\boldsymbol{S}^-| = 2 < k - 1 = 5$, consider $S^+ = \{4, 5, 7, 6\}$ and $\underline{k = 6 - 1 - |\boldsymbol{S}^-| = 3}$

- Find a splitter $a_i = 6$, then $S = \{2, 1, 3, 4, 5, 6, 7\}$,
- Since $|S^-| = 2 = k - 1$, a_i is the answer

Finding the Median

- Why the algorithm Select(S, k) is $O(n)$? (average case, not worst case)

- choose a splitter $a_i \in S$ ***uniformly at random*** (add a **randomization**)

- In each recursion j (Event X_j), we can prune S from $n\left(\frac{3}{4}\right)^{j-1}$ to $n\left(\frac{3}{4}\right)^{j}$ (the best case is ½ , here ¾ is the average case)
  - i.e., n: $7 \rightarrow 4$ after 1st recursion in the example of the previous slide.

- When j increases, $\boldsymbol{n\left(\frac{3}{4}\right)^{j}} \rightarrow \mathbf{1}$; Finally we can find the k^{th} object.

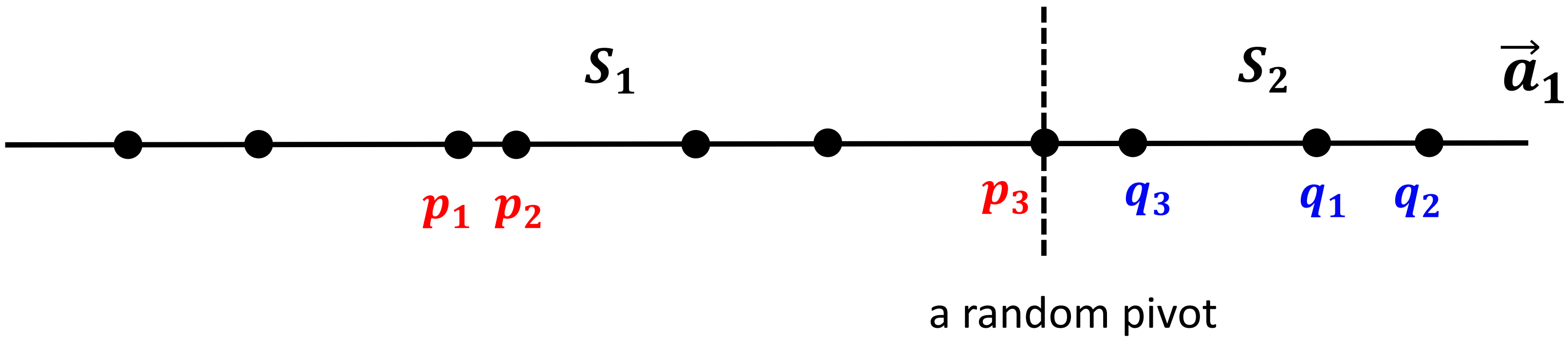
- Thus, the average time is

- $E[x] = \sum_j E[X_j] \le \sum_j cn\left(\frac{3}{4}\right)^j = cn \sum_j \left(\frac{3}{4}\right)^j = O(n)$ (here $\sum_j \left(\frac{3}{4}\right)^j = 4$)

1-Dimension Divide and Conquer

- Shall we have to choose **median** for splitting?

- Actually we can generalize the **median** to a **random pivot**:
  - choose a pivot $p_i \in S$ uniformly at random;
  - Split S into two sub sets S_1 and S_2;
    - For all $p_j \in S_1, p_j \leq p_i$;
    - For all $p_j \in S_2, p_j > p_i$;



1-Dimension Divide and Conquer

Algorithm 4: Closest-Pair-Pivot(S)

Input: a set S of n data objects on a line
Output: the samllest separation δ and the closest pair cp

if $|S| = 1$ **then**
  return $\{\delta = \infty, cp = null\}$;
if $|S| = 2$ **then**
  if $p_1 > p_2$ **then** swap(p_1, p_2);
  return $\{\delta = |p_2 - p_1|, cp = \{p_1, p_2\}\}$;
Choose a pivot $p_i \in S$ uniformly at random;
for *each element* $p_j \in S$ **do**
  Put p_j in S_1 if $p_j \leq p_i$;
  Put p_j in S_2 if $p_j > p_i$;
$\{\delta_1, cp_1\}$ = Closest-Pair-Pivot(S_1);
$\{\delta_2, cp_2\}$ = Closest-Pair-Pivot(S_2);
$\{\delta_{12}, cp_{12}\}$ is the result across the cut;
return $\{\delta, cp\} = \min(\{\delta_1, cp_1\}, \{\delta_2, cp_2\}, \{\delta_{12}, cp_{12}\})$;

Dataset

- Please download from http://yann.lecun.com/exdb/mnist/

- Use TRAINING SET IMAGE FILE (train-images-idx3-ubyte) as dataset Mnist.ds ($n = 60{,}000$ and $d = 784$)

- Each object is an image with pixels 28×28

- Please read the file format on the website and extract the dataset from the train-images-idx3-ubyte.gz

Input Format of the Project

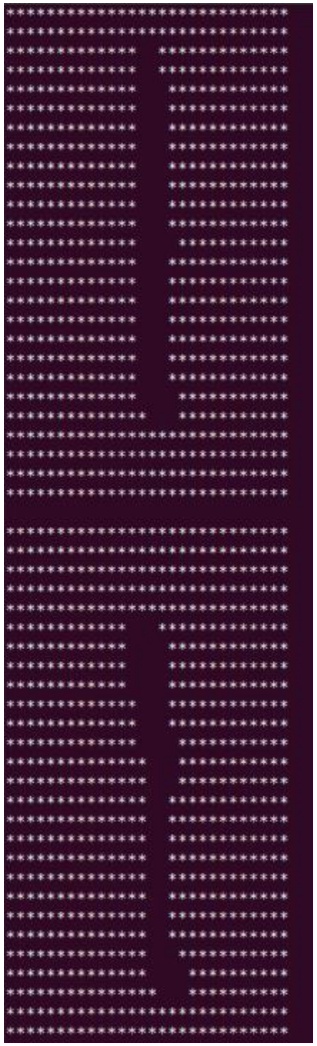
- The input format of the dataset is described as follows:

```
1 object1,1 object1,2 ...  object1,d
2 object2,1 object2,2 ...  object2,d
...
N objectN,1 objectN,2 ...  objectN,d
```

- The 1st element is the object id, the 2nd element to the $(d+1)^{\text{th}}$ element are the coordinate of the object itself.

Requirements

- Implementations:
  - We limit $m = 100$
  - Programming language: C/C++
  - implement Algorithm 1 with Algorithm 2&3, and display the running time and the closest pair
  - implement Algorithm 1 with Algorithm 2&4, and display the running time and the closest pair
  - the running time is defined by the wall clock time of Algorithm 1 to find the closest pair.
  - the closest pair can be displayed as the right figure:



- Write an experimental report:
  - Depict how to develop the two implementations
  - Discuss these two implementations with the running time and your analyses.

Submission

- 提交时间：2016年12月11日 23:59分
- 提交方式：提交给各方向学委，学委统一交给TA（张楚涵）

- 提交文件:
  - **README**
  - **源代码:** 保存在目录 “`/src`”
  - **数据集:** 假设保存在 “`/src`” **（不用提交数据集）**
  - **Makefile**: 如果你使用了多个.h 和.cpp文件， 需要写一个makefile来编译，并生成一个可执行的程序 `cp`。程序必须使用以下命令执行:

```
./cp -n 60000 -d 784 -f Mnist.ds
```

  - **实验报告（列明小组成员及分工）**
  - **请注意不要抄袭！**

- 所有文件打包成一个.zip文件（不接受.rar）
  - 命名规范：“小组序号_组长学号_组长姓名（拼音）_Project2.zip”.
  - i.e., 1_14141414_zhangsan_Project2.zip