

Algorithmic Thinking

Luay Nakhleh

Clustering and the Closest Pair Problem

The Divide-and-Conquer Algorithmic Technique

1 Clustering

Definition 1 A clustering of a set P of points into k clusters is a partition of P into sets C_1, \dots, C_k , such that

- $\forall 1 \leq i \leq k, C_i \subseteq P$,
- $\forall 1 \leq i \leq k, C_i \neq \emptyset$,
- $\forall 1 \leq i, j \leq k, i \neq j, C_i \cap C_j = \emptyset$, and
- $\cup_{i=1}^k C_i = P$.

Fig. 1 shows a clustering of 10 points into two clusters.

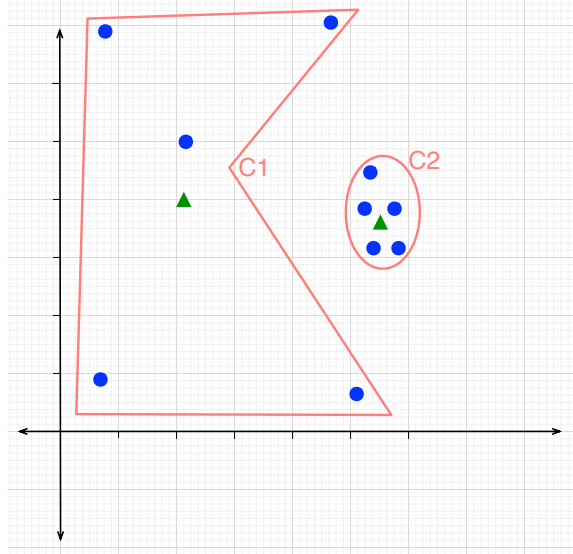


Figure 1: Two clusters C_1 and C_2 on the set of points (blue solid circles) and their centers are shown (green triangle).

We define the center of a cluster C_u as

$$\text{center}(C_u) = \frac{1}{|C_u|} \sum_{p_i \in C_u} (x_i, y_i).$$

For example, if $C_u = \{p_1, p_7, p_9\}$, with $p_1 = (1, 2)$, $p_7 = (4, 6)$, and $p_9 = (4, 4)$, then

$$\text{center}(C_u) = \frac{1}{3}((1, 2) + (4, 6) + (4, 4)) = \frac{1}{3}(9, 12) = (3, 4).$$

Fig. 1 shows the centers of the two clusters C1 and C2.

While many clusterings of the points in P exist, a desired property is that the partition results in clusters with higher similarity of points within a cluster than of points between clusters. Algorithms **HierarchicalClustering** and **KMeansClustering** below are two heuristics for generating clustering with this desired property. In both algorithms, we will use k to denote the number of clusters.

A word on implementation. While P is defined as a set in both clustering algorithms, it is more convenient to implement it using a list, since each point can be accessed directly in the list. Further, both algorithms return a set C of clusters; here, C is a set of elements, where each element is a set of points, and C satisfies the properties in Definition 1.

Algorithm 1: HierarchicalClustering.

Input: A set P of points whose i th point, p_i , is a pair (x_i, y_i) ; k , the desired number of clusters.

Output: A set C of k clusters that provides a clustering of the points in P .

```

1  $n \leftarrow |P|$ ;
2 Initialize  $n$  clusters  $C = \{C_1, \dots, C_n\}$  such that  $C_i = \{p_i\}$ ;
3 while  $|C| > k$  do
4    $(C_i, C_j) \leftarrow \operatorname{argmin}_{C_i, C_j \in C, i \neq j} d_{C_i, C_j}$ ;
5    $C \leftarrow C \cup \{C_i \cup C_j\}$ ;
6    $C \leftarrow C \setminus \{C_i, C_j\}$ ;
7 return  $C$ ;
```

Algorithm 2: KMeansClustering.

Input: A set P of points whose i th point, p_i , is a pair (x_i, y_i) ; k , the desired number of clusters; q , a number of iterations.

Output: A set C of k clusters that provides a clustering of the points in P .

```

1  $n \leftarrow |P|$ ;
2 Initialize  $k$  centers  $\mu_1, \dots, \mu_k$  to initial values (each  $\mu$  is a point in the 2D space);
3 for  $i \leftarrow 1$  to  $q$  do
4   Initialize  $k$  empty sets  $C_1, \dots, C_k$ ;
5   for  $j = 0$  to  $n - 1$  do
6      $\ell \leftarrow \operatorname{argmin}_{1 \leq f \leq k} d_{p_j, \mu_f}$ ;
7      $C_\ell \leftarrow C_\ell \cup \{p_j\}$ ;
8   for  $f = 1$  to  $k$  do
9      $\mu_f = \operatorname{center}(C_f)$ 
10 return  $\{C_1, C_2, \dots, C_k\}$ ;
```

2 2D Points, the Euclidian Distance, and Cluster Error

Both algorithms, **HierarchicalClustering** and **KMeansClustering**, make use of a distance measure, d . In the case of **HierarchicalClustering**, d_{C_i, C_j} is the distance between the two clusters C_i and C_j . In the case of **KMeansClustering**, d_{p_j, μ_f} is the distance between the point p_j and center μ_f of cluster C_f . But, how is this distance measure d defined?

In our case, we will only deal with points in the 2D space, such that each point p_i is given by two features: its horizontal (or, x) and vertical (or, y) coordinates, so that $p_i = (x_i, y_i)$. One natural way to quantify the distance between two points p_i and p_j in this case is the standard Euclidian distance:

$$d_{p_i, p_j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

While this distance measure applies directly to compute d_{p_j, μ_f} in **KMeansClustering**, how does it apply to computing d_{C_i, C_j} in **HierarchicalClustering**? By computing the distance between their centers:

$$d_{C_i, C_j} \equiv d_{center(C_i), center(C_j)}.$$

In our discussion above, we assumed a known number of clusters, k , and sought a clustering with that number of clusters. However, in general, the number of clusters is unknown. One way to determine the number of clusters is to vary the value of k , such that $k = 1, 2, 3, \dots$, and for each value of k to “inspect” the quality of the clusters obtained. One measure of such quality is the error of a cluster, which reflects how tightly packed around the center the cluster’s points are, and is defined for cluster C_i as

$$error(C_i) = \sum_{p \in C_i} (d_{p, center(C_i)})^2.$$

To illustrate, consider the two clusters in Fig. 1. Cluster C2 has a larger *error* value than C1 and, indeed, compared to C1, it is hard to argue that the points in C1 form a cluster.

3 The Closest Pair Problem

Now that we have defined the distance between points and clusters, we need an algorithm that finds, among a set of clusters, two clusters that are closest to each other (in the case of **HierarchicalClustering**), or, among a set of centers, a closest center to a given point (in the case of **KMeansClustering**). In this Module, we will approach this task by solving the Closest Pair problem, defined as follows:

- **Input:** A set P of (distinct) points and a distance measure d defined on every two points in P .
- **Output:** A pair of distinct points in P that are closest to each other under the distance measure d .

Notice that the solution of the problem might not be unique (that is, more than a single pair of points might be closest), in which case we are interested in an arbitrary one of those pairs with the smallest pairwise distance.

A simple brute-force algorithm can solve this problem, as given by the pseudo-code of Algorithm **BFClosestPair**. Notice the notation we use for finding the minimum of two tuples $\min\{(d_1, p_1, q_1), (d_2, p_2, q_2)\}$, which returns the tuple that has the smallest first element (that is, it returns tuple (d_1, p_1, q_1) if $d_1 < d_2$, and (d_2, p_2, q_2) otherwise). In case the two tuples have the same first element, one of them is returned arbitrarily.

Algorithm 3: BFClosestPair.

Input: A set P of (≥ 2) points whose i th point, p_i , is a pair (x_i, y_i) .

Output: A tuple (d, i, j) where d is the smallest pairwise distance of points in P , and i, j are the indices of two points whose distance is d .

```

1  $(d, i, j) \leftarrow (\infty, -1, -1);$ 
2 foreach  $p_u \in P$  do
3   foreach  $p_v \in P$  ( $u \neq v$ ) do
4      $(d, i, j) \leftarrow \min\{(d, i, j), (d_{p_u, p_v}, u, v)\}$ 
5 return  $(d, i, j);$ 
```

Can we do better than **BFClosestPair** in terms of running time? We will now consider two divide-and-conquer algorithms for this problem, **SlowDCClosestPair** and **FastDCClosestPair**, that differ in their implementation details and, consequently, running times.

Algorithm 4: SlowDCClosestPair.

Input: A set P of (≥ 2) points whose i th point, p_i , is a pair (x_i, y_i) .

Output: A tuple (d, i, j) where d is the smallest pairwise distance of the points in P , and i, j are the indices of two points whose distance is d .

```

1  $n \leftarrow |P|$ ;
2 if  $n \leq 3$  then
3   return BFClosestPair( $P$ );
4 else
5   Let  $H$  be a sorted list of the points in  $P$  in nondecreasing order of their horizontal ( $x$ ) coordinates;
6    $m \leftarrow \lceil n/2 \rceil$ ; // the number of points in each half
7    $mid \leftarrow \frac{1}{2}(x_{H[m-1]} + x_{H[m]})$ ; // the horizontal coordinate of the vertical dividing line
8    $P_\ell \leftarrow \{H[i] : 0 \leq i \leq m-1\}$ ;  $P_r \leftarrow \{H[i] : m \leq i \leq n-1\}$ ;
9    $(d_\ell, i_\ell, j_\ell) \leftarrow \text{SlowDCClosestPair}(P_\ell)$ ;
10   $(d_r, i_r, j_r) \leftarrow \text{SlowDCClosestPair}(P_r)$ ;
11   $(d, i, j) \leftarrow \min\{(d_\ell, i_\ell, j_\ell), (d_r, i_r, j_r)\}$ ; // min is based on the first element of the tuple
12  Let  $S$  be a list of the set  $\{p_i : |x_i - mid| < d\}$  sorted in nondecreasing order of their vertical ( $y$ ) coordinates;
13  Let  $k$  be the number of elements in  $S$ ;
14  for  $u \leftarrow 0$  to  $k-2$  do
15    for  $v \leftarrow u+1$  to  $\min\{u+3, k-1\}$  do
16       $(d, i, j) \leftarrow \min\{(d, i, j), (d_{S[u], S[v]}, S[u], S[v])\}$ ;
17 return  $(d, i, j)$ ;
```

Algorithm 5: FastDCClosestPair.

Input: A set P of (≥ 2) points whose i th point, p_i , is a pair (x_i, y_i) ; two lists H and V such that H contains the indices of the points sorted in nondecreasing order of their horizontal (x) coordinates, and V contains the indices of the points sorted in nondecreasing order of their vertical (y) coordinates.

Output: A tuple (d, i, j) where d is the smallest pairwise distance of the points in P , and i, j are the indices of two points whose distance is d .

```

1 Let  $n$  be the number of elements in  $H$ ;
2 if  $n \leq 3$  then
3    $Q \leftarrow \{p_{H[i]} : 0 \leq i \leq n-1\}$ ;
4   return BFClosestPair( $Q$ );
5 else
6    $m \leftarrow \lceil n/2 \rceil$ ; // the number of points in each half
7    $mid \leftarrow \frac{1}{2}(x_{H[m-1]} + x_{H[m]})$ ; // the horizontal coordinate of the vertical dividing line
8    $H_\ell \leftarrow H[0..m-1]$ ;  $H_r \leftarrow H[m..n-1]$ ;
9   Copy to  $V_\ell$ , in order, the elements  $V[i]$  that are elements of  $H_\ell$ ;
10  Copy to  $V_r$ , in order, the elements  $V[i]$  that are elements of  $H_r$ ;
11   $(d_\ell, i_\ell, j_\ell) \leftarrow \text{FastDCClosestPair}(P, H_\ell, V_\ell)$ ; // Use the original  $P$ 
12   $(d_r, i_r, j_r) \leftarrow \text{FastDCClosestPair}(P, H_r, V_r)$ ; // Use the original  $P$ 
13   $(d, i, j) \leftarrow \min\{(d_\ell, i_\ell, j_\ell), (d_r, i_r, j_r)\}$ ; // min is based on the first element of the tuple
14  Copy to  $S$ , in order, every  $V[i]$  for which  $|x_{V[i]} - mid| < d$ ;
15  Let  $k$  be the number of elements in  $S$ ;
16  for  $u \leftarrow 0$  to  $k-2$  do
17    for  $v \leftarrow u+1$  to  $\min\{u+3, k-1\}$  do
18       $(d, i, j) \leftarrow \min\{(d, i, j), (d_{S[u], S[v]}, S[u], S[v])\}$ ;
19 return  $(d, i, j)$ ;
```