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 $C_1, \ldots, C_k$, 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.

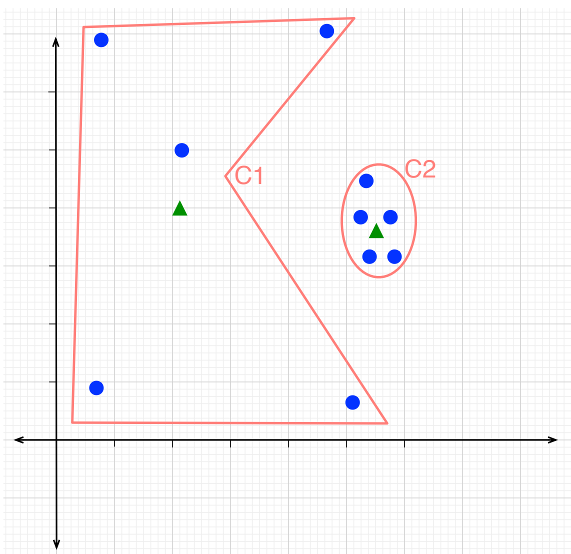

Figure 1: Two clusters C1 and C2 on the set of points (blue solid circles) and their centers are shown (green triangle).

We define the center of a cluster C_u as

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

For example, if $C_u = \{p_1, p_7, p_9\}$, with $p_1 = (1, 2)$, $p_2 = (4, 6)$, and $p_3 = (4, 4)$, then

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

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 ith 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.

$n \leftarrow |P|$;
Initialize n clusters $C = \{C_1, \ldots, C_n\}$ such that $C_i = \{p_i\}$;
while $|C| > k$ **do**
    $(C_i, C_j) \leftarrow \text{argmin}_{C_i, C_j \in C, i \neq j} d_{C_i, C_j}$;
    $C \leftarrow C \cup \{C_i \cup C_j\}$;
    $C \leftarrow C \setminus \{C_i, C_j\}$;
return C;

Algorithm 2: KMeansClustering.

Input: A set P of points whose ith 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.

$n \leftarrow |P|$;
Initialize k centers $\mu_1, \ldots, \mu_k$ to initial values (each μ is a point in the 2D space);
for $i \leftarrow 1$ **to** q **do**
    Initialize k empty sets $C_1, \ldots, C_k$;
    for $j = 0$ **to** $n - 1$ **do**
        $\ell \leftarrow \text{argmin}_{1 \leq f \leq k} d_{p_j, \mu_f}$;
        $C_\ell \leftarrow C_\ell \cup \{p_j\}$;
    for $f = 1$ **to** k **do**
        $\mu_f = \mathit{center}(C_f)$
return $\{C_1, C_2, \ldots, C_k\}$;

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 $k = 1, 23, \ldots$, and for 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 the two clusters in Fig. 1. Cluster C1 has a larger *error* value than C2 and, indeed, compared to C2, 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 **SlowClosestPair**. 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: SlowClosestPair.

Input: A set P of (≥ 2) points whose ith 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.

$(d, i, j) \leftarrow (\infty, -1, -1)$;
foreach $p_u \in P$ **do**
  foreach $p_v \in P$ *($u \neq v$)* **do**
    $(d, i, j) \leftarrow \min\{(d, i, j), (d_{p_u,p_v}, u, v)\}$; `// min compares the first element of each tuple`
return (d, i, j);

Can we do better than **SlowClosestPair** in terms of running time? We will now consider a divide-and-conquer algorithm for this problem, **FastClosestPair**.

Algorithm 4: FastClosestPair.

Input: A set P of (≥ 2) points whose ith point, p_i, is a pair (x_i, y_i), sorted in nondecreasing order of their horizontal (x) 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.

$n \leftarrow |P|$;
if $n \leq 3$ **then**
    $(d, i, j) \leftarrow \textbf{SlowClosestPair}(P)$;
else
    $m \leftarrow \lfloor n/2 \rfloor$;
    $P_\ell \leftarrow \{p_i : 0 \leq i \leq m-1\}$; $P_r \leftarrow \{p_i : m \leq i \leq n-1\}$; // P_ℓ and P_r are also sorted
    $(d_\ell, i_\ell, j_\ell) \leftarrow \textbf{FastClosestPair}(P_\ell)$;
    $(d_r, i_r, j_r) \leftarrow \textbf{FastClosestPair}(P_r)$;
    $(d, i, j) \leftarrow \min\{(d_\ell, i_\ell, j_\ell), (d_r, i_r + m, j_r + m)\}$;
    $mid \leftarrow \frac{1}{2}(x_{m-1} + x_m)$; // center line of strip
    $(d, i, j) \leftarrow \min\{(d, i, j), \textbf{ClosestPairStrip}(P, mid, d)\}$;
return (d, i, j);

Algorithm 5: ClosestPairStrip.

Input: A set P of points whose ith point, p_i, is a pair (x_i, y_i); mid and w, both of which are real numbers.

Output: A tuple (d, i, j) where d is the smallest pairwise distance of points in P whose horizontal (x) coordinates are within w from mid.

Let S be a list of the set $\{i : |x_i - mid| < w\}$;
Sort the indices in S in nondecreasing order of the vertical (y) coordinates of their associated points;
$k \leftarrow |S|$;
$(d, i, j) \leftarrow (\infty, -1, -1)$;
for $u \leftarrow 0$ **to** $k-2$ **do**
    for $v \leftarrow u+1$ **to** $\min\{u+3, k-1\}$ **do**
        $(d, i, j) \leftarrow \min\{(d, i, j), (d_{p_{S[u]}, p_{S[v]}}, S[u], S[v])\}$;
return (d, i, j);