CLUSTERING

Partitional Algorithms

Partitional Clustering

- Hierarchical methods produce nested sets of clusters
- Partitional clustering creates one set of clusters
- The desired number of clusters, k, is usually an input to the algorithm
  - Is this good?
- A metric is usually used to measure the goodness of the clustering
  - Goal is to maximize the similarities within same cluster and to minimize the similarities among different clusters

Partitional Algorithms

- MST
- Squared Error
- K-Means
- Nearest Neighbor
- PAM

MST (Minimum Spanning Tree) Algorithm

Input:

$D = \{t_1, t_2, ..., t_n\}$ // **Set of elements**

A // **Adjacency matrix showing distance between elements.**

k // **Number of desired clusters.**

Output:

f // **Mapping represented as a set of ordered pairs.**

Partitional MST Algorithm:

$M = MST(A)$;

identify inconsistent edges in M;

remove $k-1$ **inconsistent edges;**

create output representation;

Squared Error

- A metric to asses goodness of a clustering is the **squared error** measure
- It measures the squared distance of each object to the centroid of its cluster
- The squared error for a cluster $K_i = \{t_{i1}, ti_2, \ldots, t_{im}\}$ with centroid C_i is given by $\mathbf{se_{Ki}}$
- The squared error for a clustering $K = \{K_1, K_2, K_k\}$ is given by $\mathbf{se_K}$
- Minimize squared error

$$se_{K_i} = \sum_{j=1}^{m} ||t_{ij} - C_k||^2$$

$$se_K = \sum_{j=1}^{k} se_{K_j}$$

The Squared Error Clustering Algorithm

- Idea is to partition the database into k clusters so that the squared error distance for the clustering, se_K, is minimized
- Algorithm consists of two phases
- Phase 1:
  - objects are assigned at random to the k clusters
- Phase 2:
  - objects are moved around clusters in order to minimize the squared error
  - an object is always moved to the cluster with the closest center
  - keep moving objects until the difference between successive squared errors is below a predefined threshold value ($se_{Ki} - se_{K(i+1)} \leq$ threshold)

Squared Error Algorithm

Input:
$D = \{t_1, t_2, ..., t_n\}$ // Set of elements
k // Number of desired clusters.
Output:
K // Set of clusters.
Squared Error Algorithm:
assign each item t_i to a cluster;
calculate center for each cluster;
repeat
assign each item t_i to the cluster which has the closest center ;
calculate new center for each cluster;
calculate squared error;
until the difference between successive squared errors is below a threshold;

- Running time is O(knt)
- Space complexity is O(n)

K-Means

- It is similar to the Squared Error Algorithm
- Initial set of clusters is randomly chosen
- Iteratively, items are moved among sets of clusters until the desired set is reached
- High degree of similarity among elements in a cluster is obtained
- Given a cluster $K_i=\{t_{i1},t_{i2},\ldots,t_{im}\}$, the ***cluster mean*** is $m_i = (1/m)(t_{i1} + \ldots + t_{im})$

$$m_i = \frac{t_{i1} + ti_2 + \ldots + tim}{m}$$

K-Means Algorithm

Input:

$D = \{t_1, t_2, ..., t_n\}$ // Set of elements

~~A // Adjacency matrix showing distance between elements.~~

k // Number of desired clusters.

Output:

K // Set of clusters.

K-Means Algorithm:

assign initial values for means $m_1, m_2, ..., m_k$;

repeat

assign each item t_i to the cluster which has the closest mean ;

calculate new mean for each cluster;

until convergence criteria is met;

K-Means Example

- Given: {2,4,10,12,3,20,30,11,25}, k=2
- Randomly assign means (seeds): $m_1=2, m_2=4$
- $K_1=\{2,3\}$, $K_2=\{4,10,12,20,30,11,25\}$, $m_1=2.5, m_2=16$
- $K_1=\{2,3,4\}$, $K_2=\{10,12,20,30,11,25\}$, $m_1=3, m_2=18$
- $K_1=\{2,3,4,10\}$, $K_2=\{12,20,30,11,25\}$, $m_1=4.75, m_2=19.6$
- $K_1=\{2,3,4,10,11,12\}$, $K_2=\{20,30,25\}$, $m_1=7, m_2=25$
- $K_1=\{2,3,4,10,11,12\}$, $K_2=\{20,30,25\}$, $m_1=7, m_2=25$
- Stop as the clusters do not change. (until the means do not change)

Remarks on the K-Means Algorithm

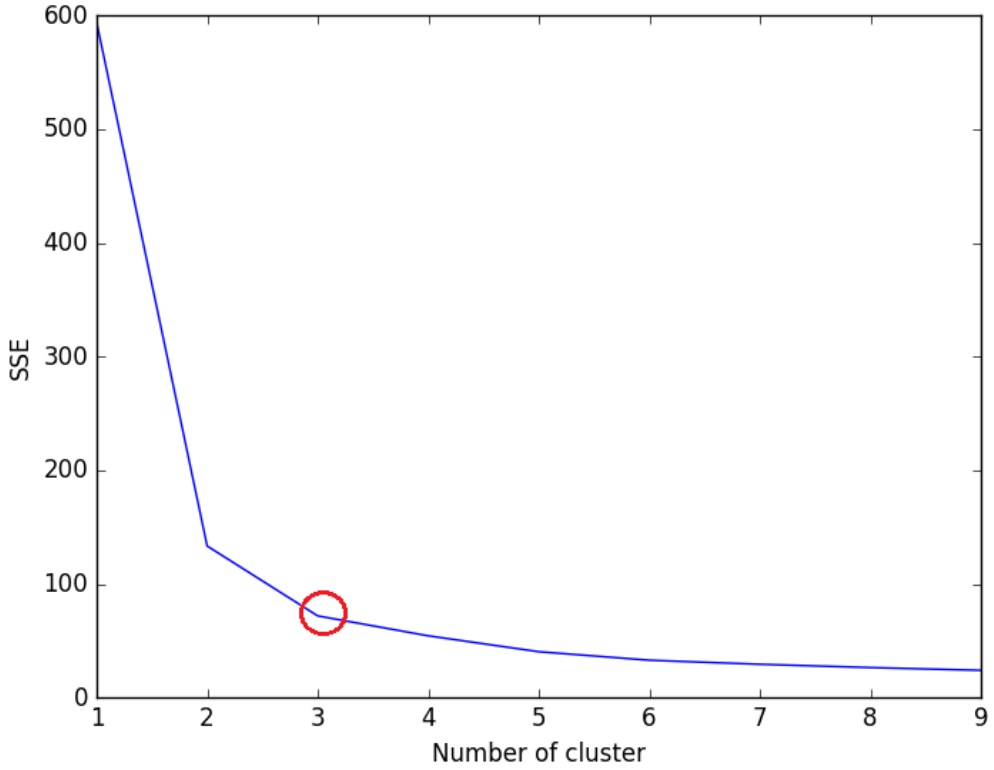
- Running time: O(tkn), where *n* is # objects, *k* is # clusters, and *t* is # iterations. Normally, $k, t << n$.
- Often terminates at a *local optimum*
- Applicable only when *mean* is defined
  - What about categorical data?
  - K-modes is a variant for categorical data
- Need to specify *k*, the *number* of clusters, in advance
- Hard to handle *outliers*
- Not suitable for discovering clusters with nonconvex shape

Variations of the K-Means Method

- A few variants of the *k-means* which differ in
  - Selection of the initial *k* means
  - Dissimilarity calculations

The Elbow Method to Determine Number of Clusters (k)

- Plot the squared error ($\mathbf{se}_k$) for different values of k (Number of clusters) and look for an elbow



Nearest Neighbor

- An object x will be added to the cluster that has the object y, which is closest to x only if dist(x, y) $\leq$ threshold value
- Otherwise, x will be added to a new cluster
- Items are iteratively merged into the existing clusters that are closest
- Incremental
- Threshold, t, used to determine if items are added to existing clusters or a new cluster is created

Nearest Neighbor Algorithm

Input:

$D = \{t_1, t_2, ..., t_n\}$ // **Set of elements**

A // **Adjacency matrix showing distance between elements.**

Output:

K // **Set of clusters.**

Nearest Neighbor Algorithm:

$K_1 = \{t_1\}$;

$K = \{K_1\}$;

$k = 1$;

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

  find the t_m **in some cluster** K_m **in** K **such that** $dis(t_i, t_m)$ **is the smallest;**

  if $dis(t_i, t_m) \leq t$ **then**

    $K_m = K_m \cup t_i$

  else

    $k = k + 1$;

    $K_k = \{t_i\}$;

Nearest Neighbor Algorithm - Example

- Apply the algorithm to the data set represented by the following adjacency matrix. Use threshold value, t = 2.
- K = {}
- K_1 = {A}, K = {K_1}
- K_1 = {A, B}
- K_1 = {A, B, C}
- K_1 = {A, B, C, D}
- K_2 = {E}, K = {K_1, K_2 }

	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0

Remarks on Nearest Neighbor Algorithm

- Time complexity is $O(n^2)$
- Space complexity is also $O(n^2)$
- Nearest Neighbor is a special case of the k-nearest neighbors algorithm
- Number of clusters, k, is not an input to the algorithm

The PAM Algorithm

- PAM ≡ Partitioning Around Medoids
- aka K-Medoids
- The k-means algorithm is sensitive to outliers
- K-Medoids: Instead of taking the mean as the cluster representative, the medoid is used
- Remember: medoid is the most centrally located object in a cluster
- Handles outliers well

k-means algorithm is sensitive to outliers

- D = $\{1, 2, 3, 8, 9, 10, 25\}$; where 25 is an outlier
- K1 = $\{1, 2, 3\}$, K2 = $\{8, 9, 10\}$
- If we apply k-means using k = 2
- K= $\{\{1, 2, 3\}, \{8, 9, 10, 25\}\}$
- m1 = 2, m2 = 13
- $se_K = (1 - 2)^2 + (2 - 2)^2 + (3 - 2)^2 + (8 - 13)^2 + \ldots + (25 - 13)^2 = 196$
- K′ = $\{\{1, 2, 3, 8\}, \{9, 10, 25\}\}$
- m1 = 3.5, m2 = 14.67
- $se_{K'} = (1 - 3.5)^2 + (2 - 3.5)^2 + (3 - 3.5)^2 + (8 - 3.5)^2 + (9 - 14.67)^2 + (10 - 14.67)^2 + (25 - 14.67)^2 = 189.67$

Idea of the PAM Algorithm

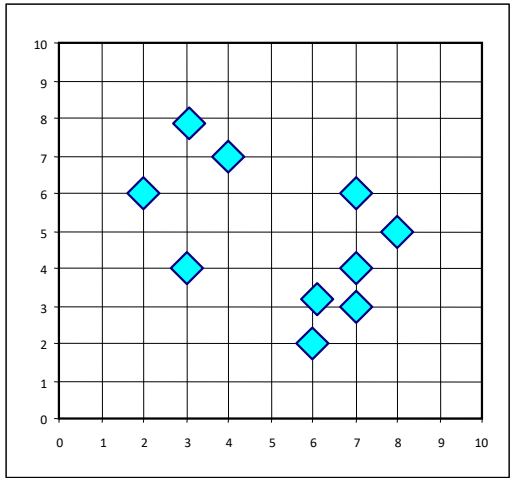
- Partition the data set into k clusters based on the principle of minimizing the sum of dissimilarities between each object in a cluster and the reference point “medoid”
- Starts with an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids as long as this improves the clustering

The PAM Algorithm

- Algorithm consists of the following steps:

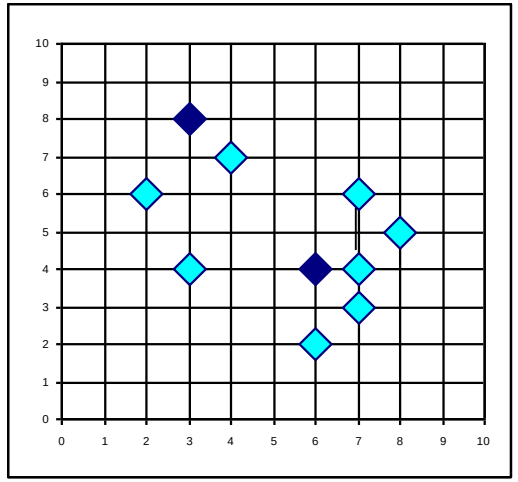
1. choose k objects as the k medoids
2. each remaining object is clustered with the medoid that is most similar to it
3. iteratively, replace one of the medoids by one of the non-medoids as long as the quality of the clustering is improved

A Typical K-Medoids Algorithm (PAM)

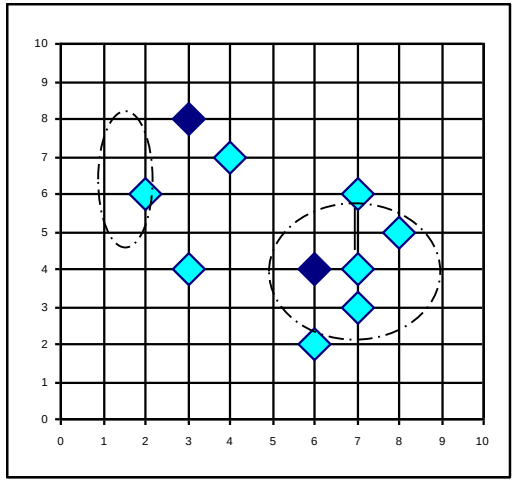


K=2

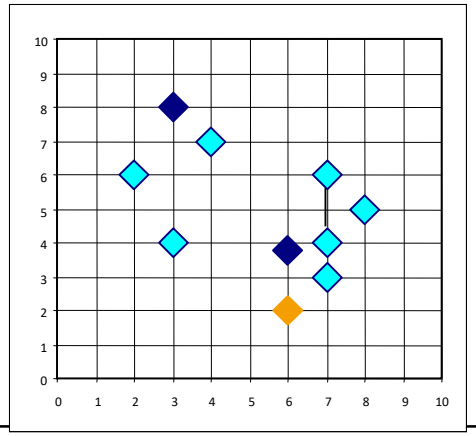
Arbitrary choose k objects as initial medoids



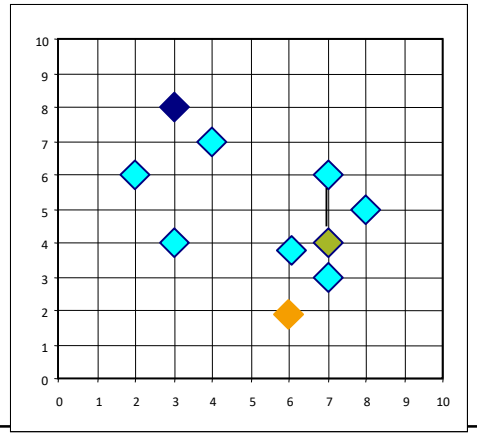
Assign each remaining object to nearest medoid



Select a nonmedoid object, t_h



Compute total cost of swapping



Swapping t_i and t_h if quality is improved.

Do loop

Until no change

The PAM Algorithm – Cost Function

- Quality of the clustering is estimated using a **cost function**
- cost function measures the distance between an object and the medoid of its cluster
- Let C_{jih} be the cost change for an object t_j associated with swapping medoid t_i by non-medoid t_h
- Notations:
  - K_i is a specified cluster
  - t_i is the current medoid of cluster K_i
  - t_h is a non-medoid object

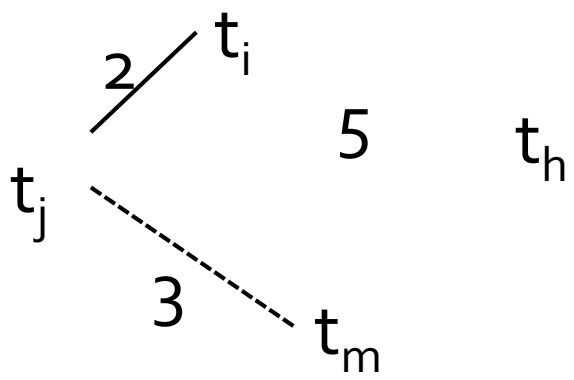
PAM - Cost Calculation

- medoids are changed if the overall cost is improved
- C_{jih} – cost change for an item t_j associated with swapping medoid t_i with non-medoid t_h.
- We have the following four cases for calculating C_{jih}

1. $t_j \in K_i$, **but** $\exists$ **another medoid** t_m *where* $dis(t_j, t_m) \leq dis(t_j, t_h)$
2. $t_j \in K_i$, **but** $dis(t_j, t_h) \leq dis(t_j, t_m) \forall$ *other medoids* t_m;
3. $t_j \in K_m$, $\notin K_i$, **and** $dis(t_j, t_m) \leq dis(t_j, t_h)$; **and**
4. $t_j \in K_m$, $\notin K_i$, **but** $dis(t_j, t_h) \leq dis(t_j, t_m)$.

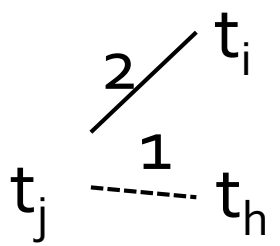
Case 1: $t_j \in K_i$ but $\exists$ another medoid t_m where $dist(t_j, t_m) < dist(t_j, t_h)$

- t_j will be reassigned to K_m
- $C_{jih} = dist(t_j, t_m) - dist(t_j, t_i)$



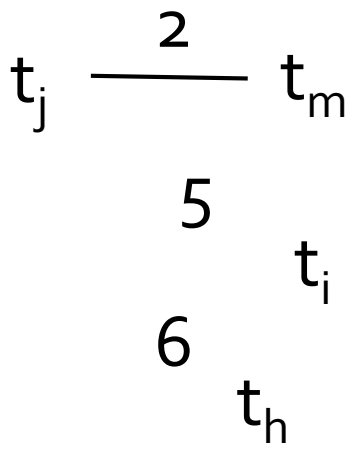
Case 2: $t_j \in K_i$ but $dist(t_j, t_h) \leq dist(t_j, t_m)$ ∀ other medoid t_m

- t_j will remain in the same cluster
- $C_{jih} = dist(t_j, t_h) - dist(t_j, t_i)$



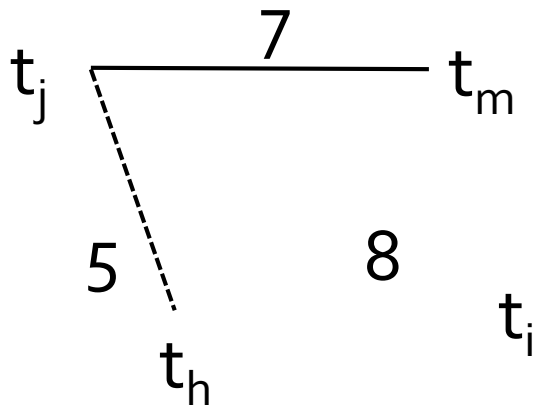
Case 3: $t_j \in K_m \notin K_i$, and $dist(t_j, t_m) \leq dist(t_j, t_h)$

- t_j will remain in K_m
- $C_{jih} = 0$



Case 4: $t_j \in K_m \notin K_i$, but $dist(t_j, t_h) \leq dist(t_j, t_m)$

- t_j will be reassigned to t_h
- $C_{jih} = dist(t_j, t_h) - dist(t_j, t_m)$



PAM - Cost Calculation

- Let TC_{ih} denote the total impact on the clustering that is associated with swapping medoid t_i by non-medoid t_h
- $TC_{ih} = \sum_{j=1}^{n} C_{jih}$

PAM Algorithm

Input:
$D = \{t_1, t_2, ..., t_n\}$ // **Set of elements**
A // **Adjacency matrix showing distance between elements.**
k // **Number of desired clusters.**
Output:
K // **Set of clusters.**
PAM Algorithm:
arbitrarily select k **medoids from** D**;**
repeat
for each t_h **not a medoid do**
for each medoid t_i **do**
calculate TC_{ih}**;**
find i, h **where** TC_{ih} **is the smallest;**
if $TC_{ih} < 0$ **then**
replace medoid t_i **with** t_h**;**
until $TC_{ih} \geq 0$**;**
for each $t_i \in D$ **do**
assign t_i **to** K_j **where** $dis(t_i, t_j)$ **is the smallest over all medoids;**

Remarks

- Ordering of input does not impact results
- PAM is more robust than k-means in the presence of noise and outliers
- PAM does not **scale well** for large data set
  - $O(k(n-k)^2)$ for each iteration

    where n is # of data, k is # of clusters
- K-means is O(ntk), but t and k are usually much less than n
- Both PAM and k-means require the user to specify k

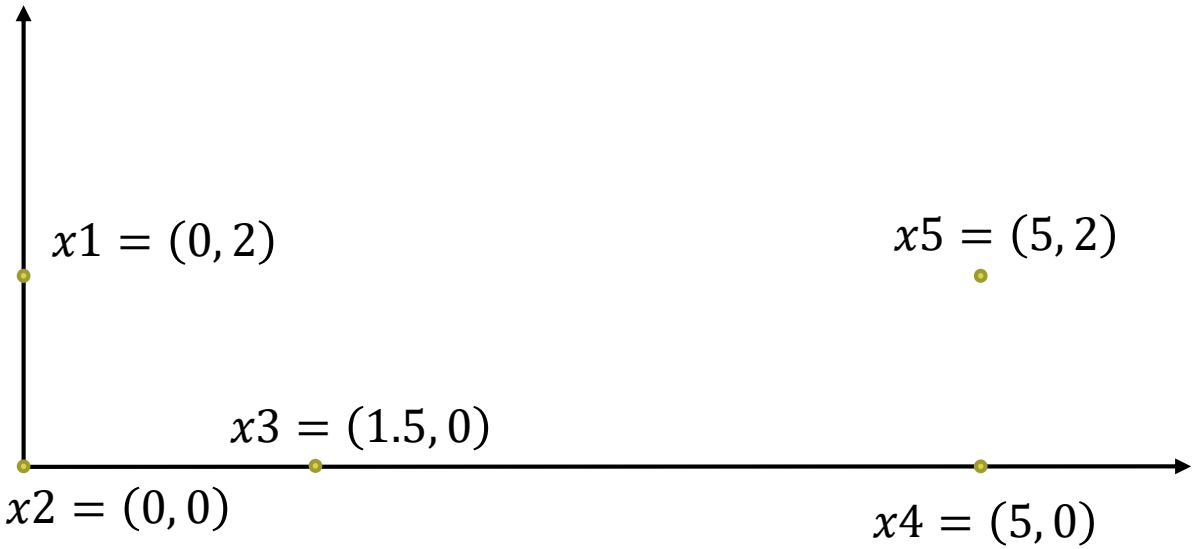
➔Sampling based method,

CLARA(Clustering LARge Applications)

PAM – Example (1 – 3)

- Apply PAM to the data set consisting of : $x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0)$, and $x5 = (5, 2)$
- Let $k = 2$

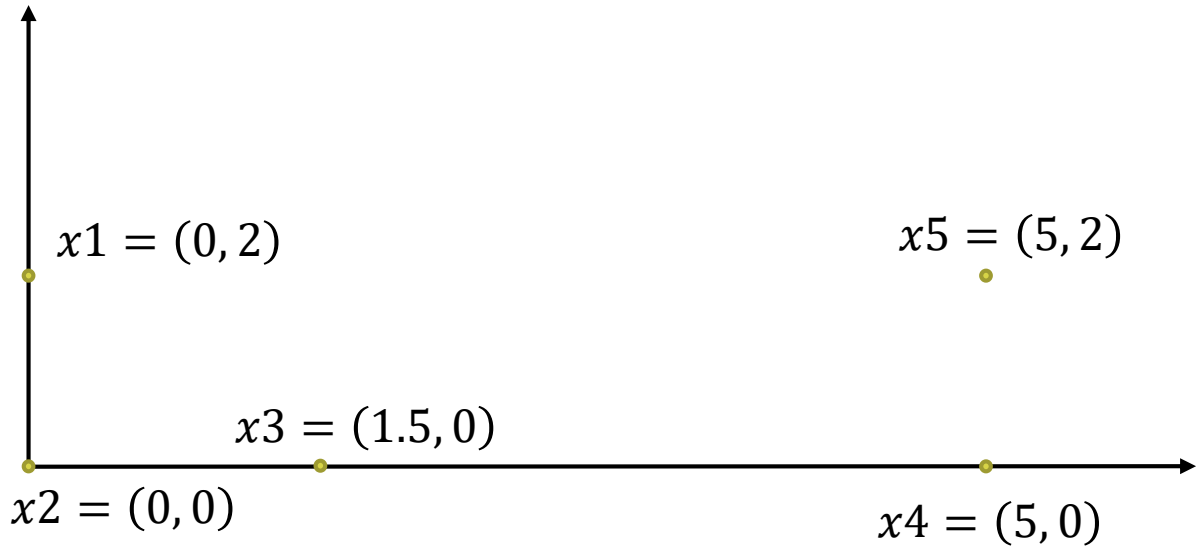
	$x1$	$x2$	$x3$	$x4$	$x5$
$x1$	0	2	2.5	5.39	5
$x2$		0	1.5	5	5.39
$x3$			0	3.5	4.03
$x4$				0	2
$x5$					0



PAM – Example (2 – 3)

- Assume initial medoids are: $x1$ and $x2$
- Initial clusters: $K1 = \{x1, x5\}, K2 = \{x2, x3, x4\}$
- Non-medoids: $x3, x4, x5$
- To decide if a non-medoid, t_h, should replace a medoid, t_i, calculate TC_{ih}
- Choose the pair (t_i, t_h) that gives the least value for TC_{ih}
- We need to examine the following costs: TC_{13}, TC_{14}, TC_{15}, TC_{23}, TC_{24}, and TC_{25}

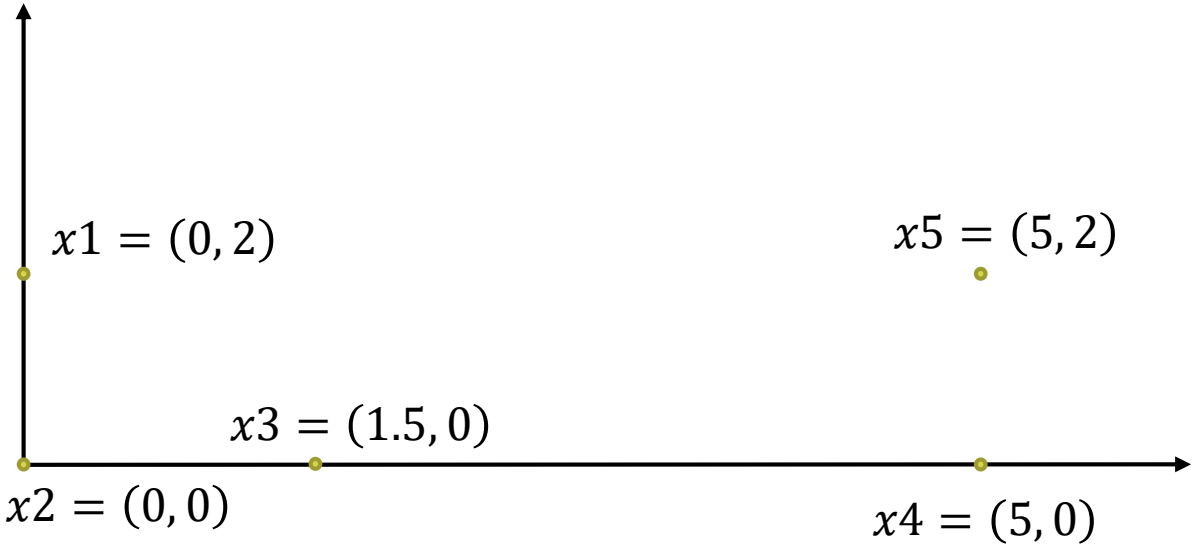
	$x1$	$x2$	$x3$	$x4$	$x5$
$x1$	0	2	2.5	5.39	5
$x2$		0	1.5	5	5.39
$x3$			0	3.5	4.03
$x4$				0	2
$x5$					0



PAM – Example (3 – 3)

- $TC_{ih} = \sum_{j=1}^{n} C_{jih}$
- $TC_{13} = C_{113} + C_{213} + C_{313} + C_{413} + C_{513}$
- $TC_{13} = (2 - 0) + 0 + (0 - 1.5) + (3.5 - 5) + (4.03 - 5)$
- $TC_{13} = 2 - 1.5 - 1.5 - 0.97 = -1.97$

	$x1$	$x2$	$x3$	$x4$	$x5$
$x1$	**0**	**2**	**2.5**	**5.39**	**5**
$x2$		**0**	**1.5**	**5**	**5.39**
$x3$			**0**	**3.5**	**4.03**
$x4$				**0**	**2**
$x5$					**0**



CLARA (Clustering Large Applications)

- PAM has complexity $O(k(n-k)^2)t$ where t is number of iterations
- A sampling method called *CLARA* was introduced to deal with large datasets
- It works by talking a random sample from the dataset
- Medoids are found by applying PAM to this sample
- These medoids are then used as the medoids for the whole data set
- Clustering is done using these medoids

CLARA (Clustering Large Applications) cont.

- Strength: deals with larger data sets better than *PAM*
- Weakness:
  - Efficiency depends on the sample size
  - A good clustering based on samples will not necessarily represent a good clustering of the whole data set
- To improve the accuracy of CLARA, ***multiple samples*** can be drawn and PAM is applied to them
- Take medoids from the sample that performs the best

CLARANS ("Randomized" CLARA)

- *CLARANS* ≡ Clustering Large Applications based upon Randomized Search
- The clustering process can be presented as searching a graph where every node is a potential solution, that is, a set of *k* medoids
- Two nodes are neighbours if their sets differ by only one medoid
- Each node can be assigned a cost that is defined to be the total dissimilarity between every object and the medoid of its cluster
- The problem corresponds to search for a minimum on the graph
- At each step in PAM, all neighbours of current_node node are searched; the neighbour which corresponds to the deepest descent in cost is chosen as the next solution

CLARANS ("Randomized" CLARA)

- For large values of n and k, examining k(n-k) neighbours is time consuming
- CLARANS starts with a random sample, finds the medoids, clusters the objects, and computes se_K
- CLARANS then draws a neighbor at random from the whole dataset to examine and computes $se_{K'}$ for the neighbor
- Notice: X' is a neighbor for X if X – X' is only one element
- If $se_{K'} < se_K$ than K' becomes the current set of medoids
- Repeat same process by comparing with a certain number of neighbors as specified by the parameter **max_neighbor**

CLARANS ("Randomized" CLARA)

- CLARANS has the benefit of not confining the search to a restricted area
- CLARANS uses another parameter **numlocal** that specifies the total number of times the algorithm is run from scratch
- CLARANS is more efficient and scalable than both PAM and CLARA; returns higher quality clusters

Algorithm CLARANS

- 1. Input parameters *num_local* and *max_neighbor*. Initialize *i* to 1, and *mincost* to a large number.
- 2. Set *current_node* to an arbitrary node in $G_{n,k}$.
- 3. Set j to 1.
- 4. Consider a random neighbor S of *current_node*, and based on S, calculate the cost differential of the two nodes.
- 5. If S has a lower cost, set current_node to S, and go to Step 3.
- 6. Otherwise, increment j by 1. If j <= max_neighbor, go to Step 4.
- 7. Otherwise, when j > max_neighbor, compare the cost of current_node with mincost. If the former is less than mincost, set mincost to the cost of current_node and set bestnode to current_node.
- 8. Increment i by 1. If i > num_local, output bestnode and halt. Otherwise, go to Step 2.