Cluster Analysis I

What is Cluster Analysis

- Cluster: A collection of data objects
  - **similar** (or **related**) to one another within the same group
  - dissimilar (or unrelated) to the objects in other groups
- Cluster analysis (or *clustering*, *data segmentation*, …)
  - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters
- Unsupervised learning: no predefined classes (i.e., *learning by observations* vs. learning by examples: supervised)
- Typical applications
  - As a stand-alone tool to gain insight into *data distribution*
  - As a preprocessing step for other algorithms

Clustering Examples

- ***Biology***: taxonomy of living things: kingdom, phylum, class, order, family, genus and species
- Information retrieval: document clustering
- ***Land use***: Identification of areas of similar land use in an earth observation database
- ***Marketing***: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- ***City-planning***: Identifying groups of houses according to their house type, value, and geographical location
- ***Earth-quake studies***: Observed earth quake epicenters should be clustered along continent faults
- ***Climate***: understand earth climate, find atmospheric and oceanic patterns
- …

Clustering as a Preprocessing Tool

- Binning:
  - Converting numerical variables to categorical
- Summarization:
  - Preprocessing for regression, PCA, classification, and association analysis
- Compression:
  - Image processing: vector quantization
- Finding K-nearest Neighbors
  - Localizing search to one or a small number of clusters
- Outlier detection
  - Outliers are often viewed as those “far away” from any cluster

What is Good Clustering?

- A good clustering method will produce high quality clusters
  - **high *intra-class*** similarity: **cohesive** within clusters
  - **low *inter-class*** similarity: **distinctive** between clusters

- The quality of a clustering method depends on
  - the similarity measure used by the method
  - its implementation, and
  - Its ability to discover some or all of the hidden patterns

Cluster Quality

- Dissimilarity/Similarity metric
  - Expressed in terms of a **distance function**, typically metric:

  $d(i, j)$
  - The definitions of distance functions are usually rather different for interval-scaled, Boolean, categorical, ordinal ratio, and vector variables, etc.
  - **Weights** should be associated with different variables based on applications and data semantics

- Quality of clustering:
  - There is usually a separate "**quality function**" that measures the "*goodness*" of a cluster.
  - It is hard to define "similar enough" or "good enough"
    - The answer is typically highly subjective

Considerations for Cluster Analysis

- Partitioning criteria
  - Single level vs. hierarchical partitioning (often, *multi-level* hierarchical partitioning is desirable)
- Separation of clusters
  - Exclusive (e.g., one customer belongs to only one region)
  - Non-exclusive (e.g., one document may belong to more than one class)
- Similarity measure
  - Distance-based (e.g., Euclidian, road network, vector) vs. connectivity-based (e.g., density or contiguity)
- Clustering space
  - Full space (often when low dimensional) vs. subspaces (often in high-dimensional clustering)

Challenges

- **Scalability**
  - Clustering all the data instead of only on samples
- Ability to deal with different types of attributes
  - Numerical, binary, categorical, ordinal, linked, and mixture of these
- Constraint-based clustering
  - User may give inputs on constraints
  - Use domain knowledge to determine input parameters
- Interpretability and usability
- Others
  - Discovery of clusters with arbitrary shape
  - Ability to deal with noisy data
  - Incremental clustering and insensitivity to input order
  - High dimensionality

Major Clustering Approaches

- Partitioning approaches:
  - Construct various partitions and then evaluate them by some criterion, e.g., minimizing the sum of square errors
  - Typical methods: k-means, k-medoids, CLARANS
- Hierarchical approaches:
  - Create a hierarchical decomposition of the set of data (or objects) using some criterion
  - Typical methods: Diana, Agnes, BIRCH, CAMELEON
- Density-based approaches:
  - Based on connectivity and density functions
  - Typical methods: DBSCAN, OPTICS, DenClue
- Grid-based approaches:
  - based on a multiple-level granularity structure
  - Typical methods: STING, WaveCluster, CLIQUE
- Model-based approaches:
  - A model is hypothesized for each of the clusters and tries to find the best fit of that model to each other
  - Typical methods: EM, SOM, COBWEB

Partitioning Algorithms: Basic Concept

- Partitioning method: Partitioning a database ***D*** of ***n*** objects into a set of ***k*** clusters, such that the sum of squared distances is minimized (where c_i is the centroid or medoid of cluster C_i)

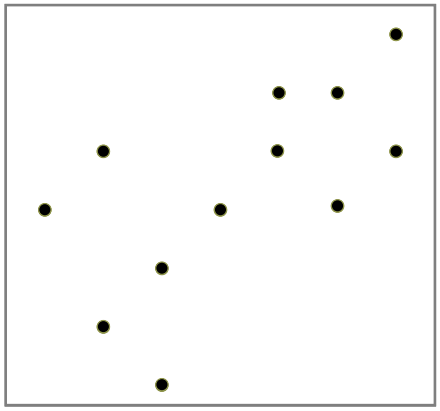
$$E = \sum_{i=1}^{k} \sum_{p \in C_i} (p - c_i)^2$$

- Given ***k***, find a partition of *k clusters* that optimizes the chosen partitioning criterion
  - ❖ Global optimal: exhaustively enumerate all partitions
  - – Heuristic methods: *k-means* and *k-medoids* algorithms
  - – *k-means* (MacQueen'67, Lloyd'57/'82): Each cluster is represented by the center of the cluster
  - – *k-medoids* or PAM (Partitioning around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster

The *K*-means Clustering Method

- Given *k*, the *k-means* algorithm is implemented in four steps:
  1. Partition objects into ***k*** nonempty subsets
  2. Compute seed points as the centroids of the clusters of the current partitioning (the centroid is the center, i.e., ***mean point***, of the cluster)
  3. Assign each object to the cluster with the nearest seed point
  4. Go back to Step 2, stop when the assignment does not change

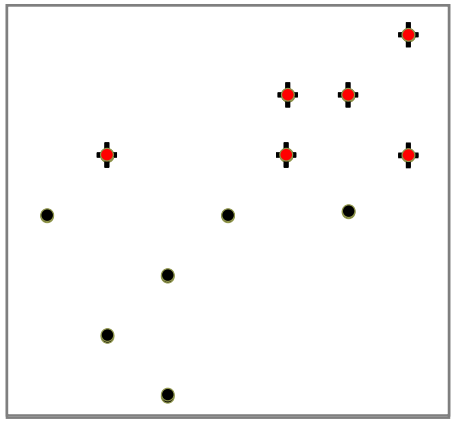
K-means Example



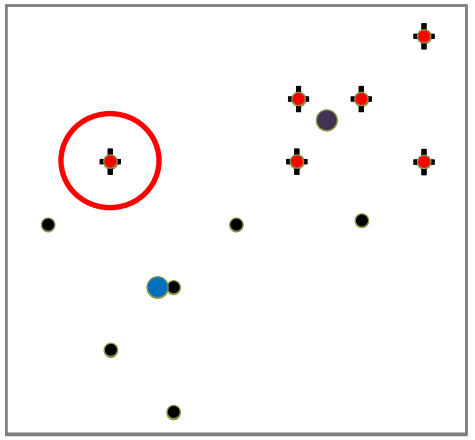
The initial data set

K=2

Arbitrarily partition objects into k groups

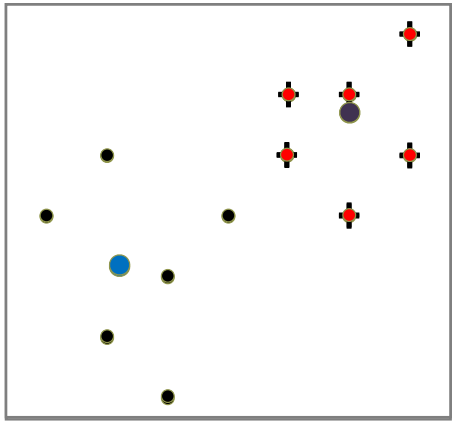


Update the cluster centroids

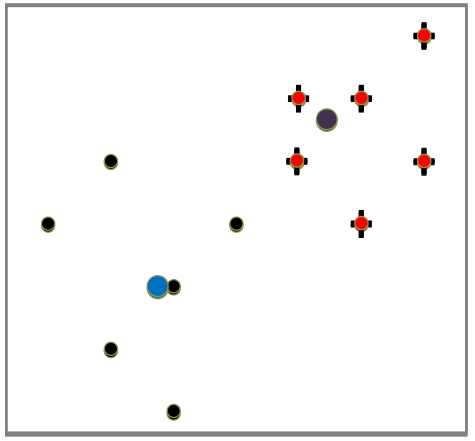


Reassign objects

Loop if needed



Update the cluster centroids



- Partition objects into *k* nonempty subsets
- Repeat
  - Compute centroid (i.e., mean point) for each partition
  - Assign each object to the cluster of its nearest centroid
- Until no change

Example of *k*-Means Clustering at Work

- Assume $k = 2$ to cluster following data points

a	b	c	d	e	f	g	h
(1, 3)	(3, 3)	(4, 3)	(5, 3)	(1, 2)	(4, 2)	(1, 1)	(2, 1)

- Step 1: $k = 2$ specifies number of clusters to partition
- Step 2: Randomly assign $k = 2$ cluster centers

  For example, $m_1 = (1, 1)$ and $m_2 = (2, 1)$

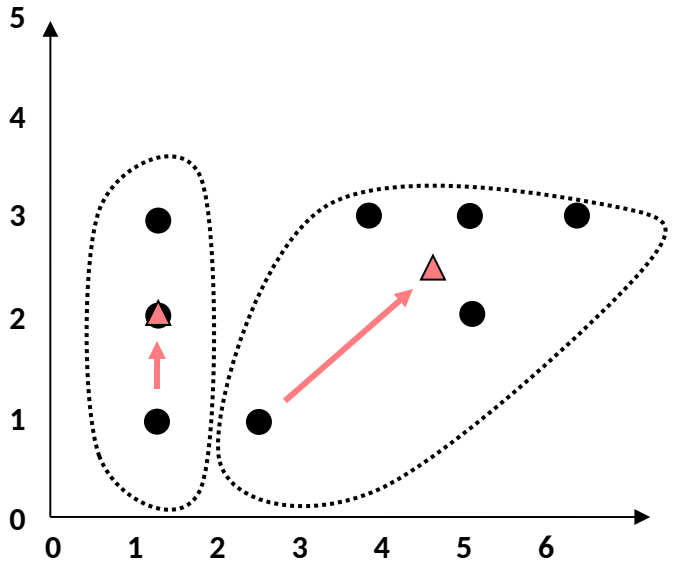
- First Iteration
  - Step 3: For each record, find nearest cluster center

    Euclidean distance from points to m_1 and m_2 shown

Point	a	b	c	d	e	f	g	h
Distance from m_1	2.00	2.83	3.61	4.47	1.00	3.16	0.00	1.00
Distance from m_2	2.24	2.24	2.83	3.61	1.41	2.24	1.00	0.00
Cluster Membership	C_1	C_2	C_2	C_2	C_1	C_2	C_1	C_2

Example of *k*-Means Clustering at Work *(cont'd)*

- Step 4: For *k* clusters, find cluster centroid, update location
- Cluster 1 = [(1 + 1 + 1)/3, (3 + 2 + 1)/3] = (1, 2), Cluster 2 = [(3 + 4 + 5 + 4 + 2)/5, (3 + 3 + 3 + 2 + 1)/5] = (3.6, 2.4)
- Figure shows movement of clusters m_1 and m_2 (triangles) after first iteration of algorithm



Example of *k*-Means Clustering at Work *(cont'd)*

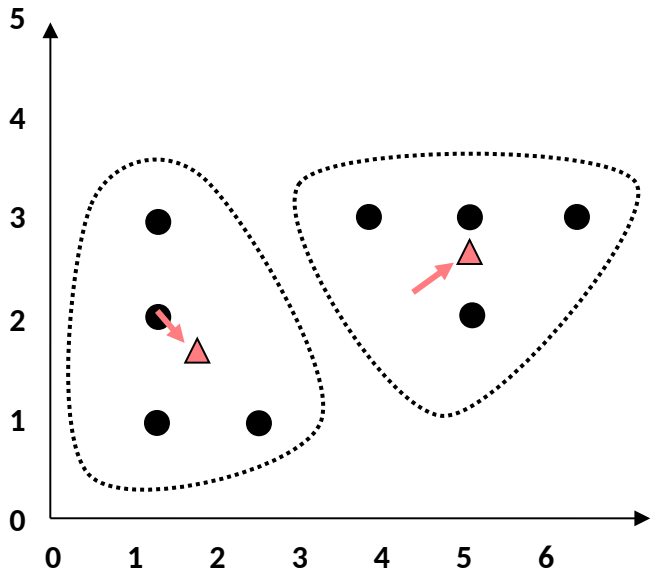
- Step 5: Repeats Steps 3 – 4 until convergence or termination

- Second Iteration
  - Repeat procedure for Steps 3 – 4
  - Again, for each record find nearest cluster center m*1* = (1, 2) or m*2* = (3.6, 2.4)
  - Table below shows how *h* moved to cluster 1

Point	a	b	c	d	e	f	g	h
Distance from m_1	2.00	2.83	3.61	4.47	1.00	3.16	0.00	1.00
Distance from m_2	2.24	2.24	2.83	3.61	1.41	2.24	1.00	0.00
Cluster Membership	C_1	C_2	C_2	C_2	C_1	C_2	C_1	C_1

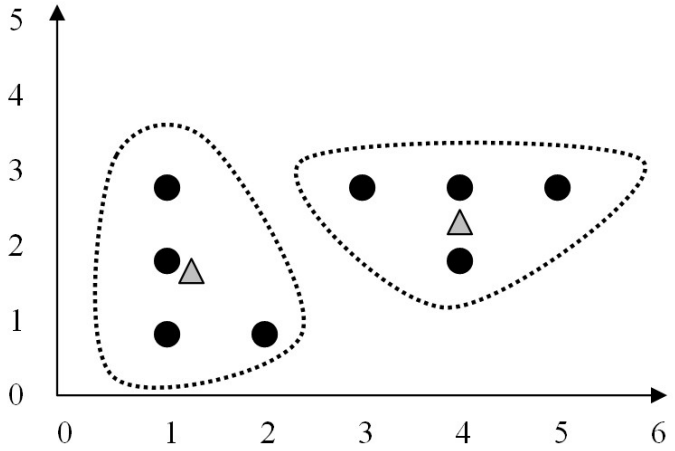
Example of *k*-Means Clustering at Work *(cont'd)*

- Cluster centroids updated to m_1 = (1.25, 1.75) or m_2 = (4, 2.75)
- After Second Iteration, cluster centroids shown to move slightly



Example of *k*-Means Clustering at Work *(cont'd)*

- Third (Final) Iteration
  - Repeat procedure for Steps 3 – 4
  - Now, for each record find nearest cluster center m_1 = (1.25, 1.75) or m_2 = (4, 2.75)
  - This time, no records shift cluster membership
  - Centroids remain unchanged, therefore algorithm terminates



Comments on the *K*-means Method

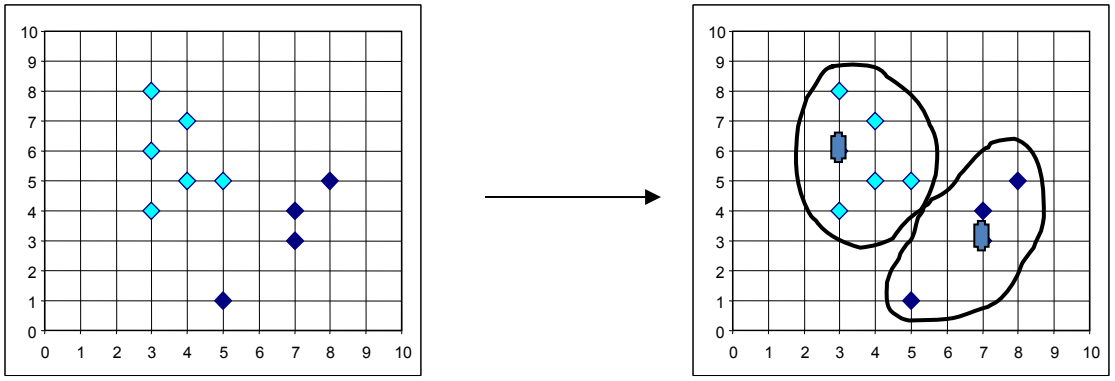
- Strength: ***Efficient***: $O(tkn)$, where n is # objects, k is # clusters, and t is # iterations. Normally, $k, t << n$.
    - Comparing: PAM: $O(k(n-k)^2)$, CLARA: $O(ks^2 + k(n-k))$
- Comment: Often terminates at a *local optima*. (Greedy)
- Weakness
  - Applicable only to objects in a **continuous** n-dimensional space
    - Using the k-modes method for categorical data
    - In comparison, k-medoids can be applied to a wide range of data
  - Need to specify *k*, the *number* of clusters, in advance (there are ways to automatically determine the best k (see Hastie et al., 2009)
  - Sensitive to noisy data and *outliers*
  - Not suitable to discover clusters with *non-convex shapes*

Variations of the *K*-means Method

- Most of the variants of the *k-means* which differ in
  - Selection of the initial *k* means
  - Dissimilarity calculations
  - Strategies to calculate cluster means
- Handling *categorical data*: *k-modes*
  - Replacing means of clusters with modes
  - Using new dissimilarity measures to deal with categorical objects
  - Using a frequency-based method to update modes of clusters
  - A mixture of categorical and numerical data: *k-prototype* method

K-Medoids

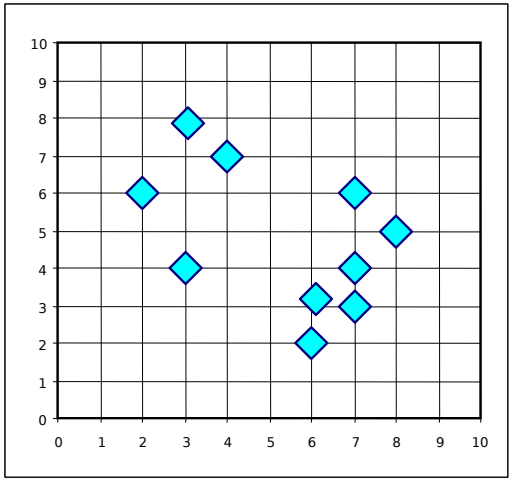
- The k-means algorithm is sensitive to outliers
  - Since an object with an extremely large value may substantially distort the distribution of the data
- K-Medoids: Instead of taking the **mean** value of the object in a cluster as a reference point, **medoids** can be used, which is the **most centrally located** object in a cluster



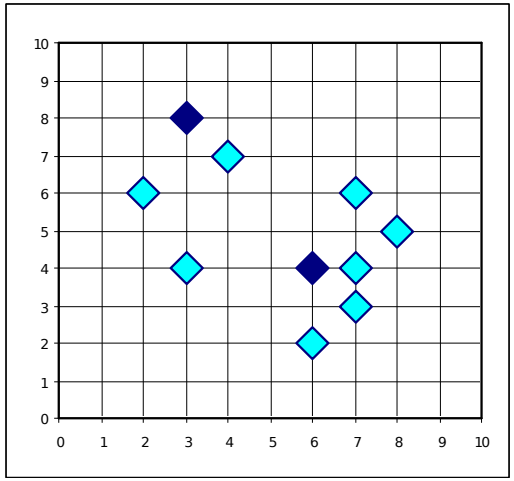
The *K-medoid Clustering Method*

- *K-Medoids* Clustering: Find *representative* objects (medoids) in clusters
  - *PAM* (Partitioning Around Medoids, Kaufmann & Rousseeuw 1987)
    - Starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering
    - *PAM* works effectively for small data sets, but does not scale well for large data sets (due to the computational complexity)
- Efficiency improvement on PAM
  - *CLARA* (Kaufmann & Rousseeuw, 1990): PAM on samples
  - *CLARANS* (Ng & Han, 1994): Randomized re-sampling

PAM: A Typical K-Medoids Algorithm

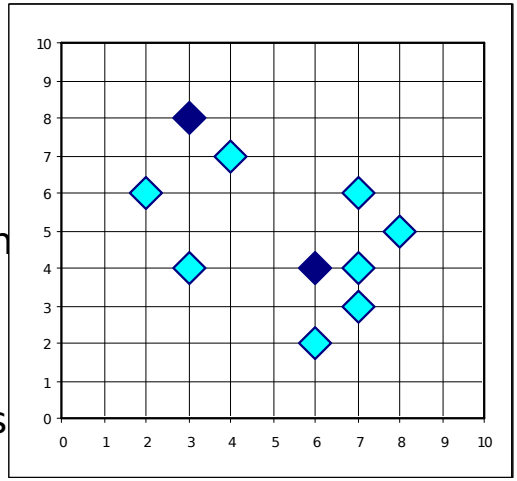


Arbitrarily choose k objects as initial medoids



Assign each remaining object to nearest medoids

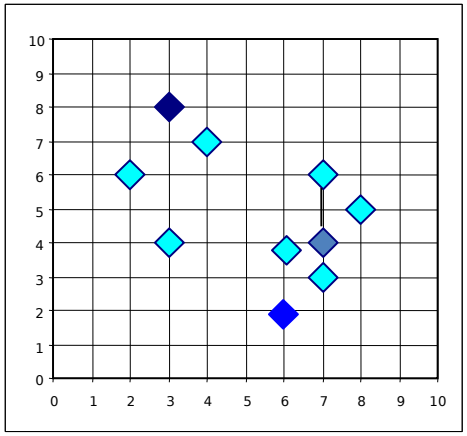
Total Cost = 20



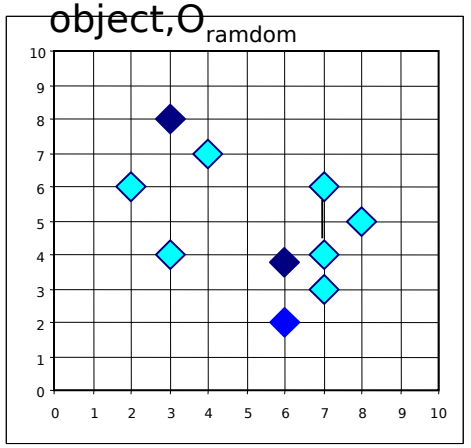
K=2

Randomly select a nonmedoid object,O_{ramdom}

Total Cost = 26



Compute total cost of swapping



Do loop

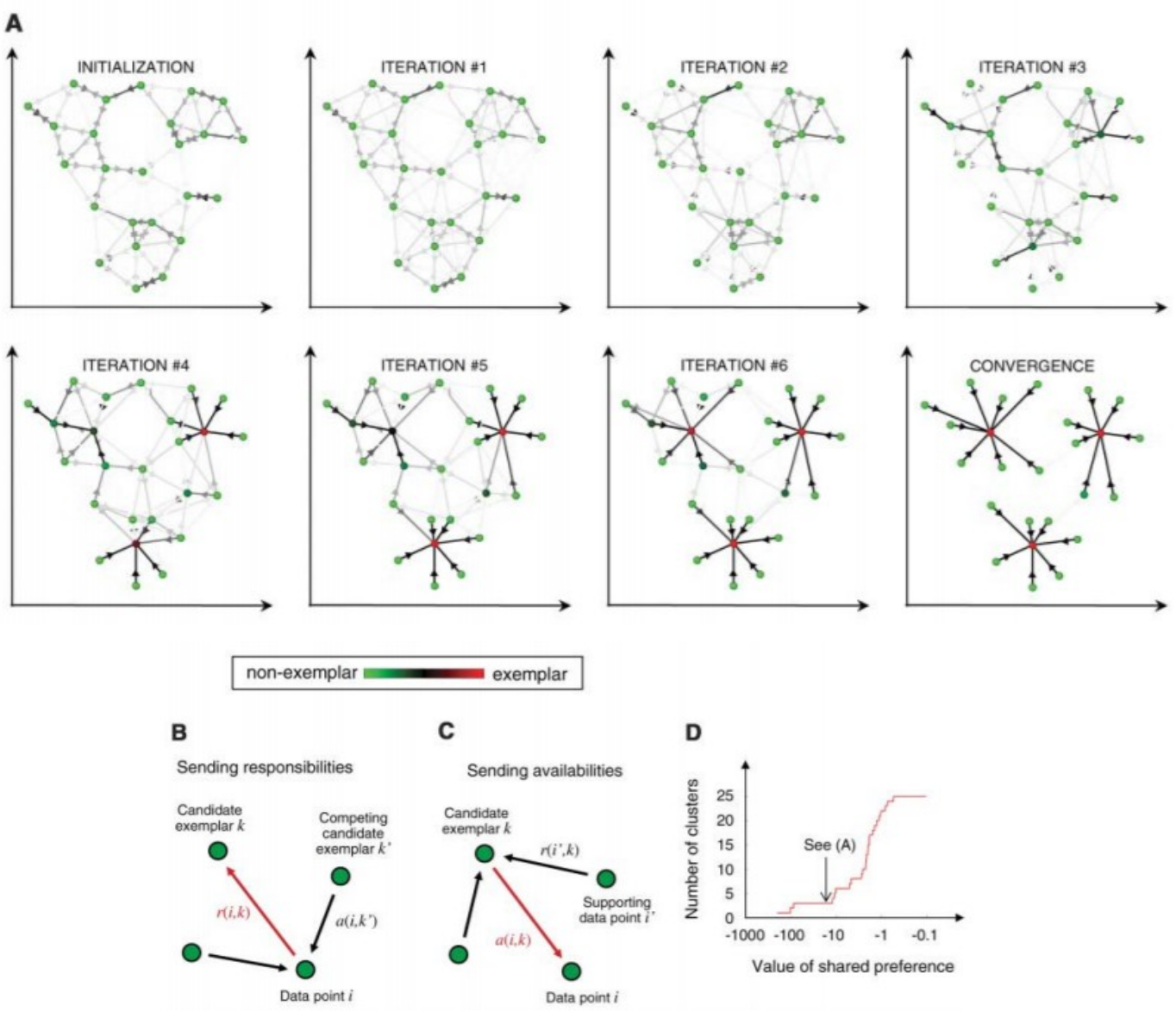
Until no change

Swapping O and O_{ramdom}

If quality is improved.

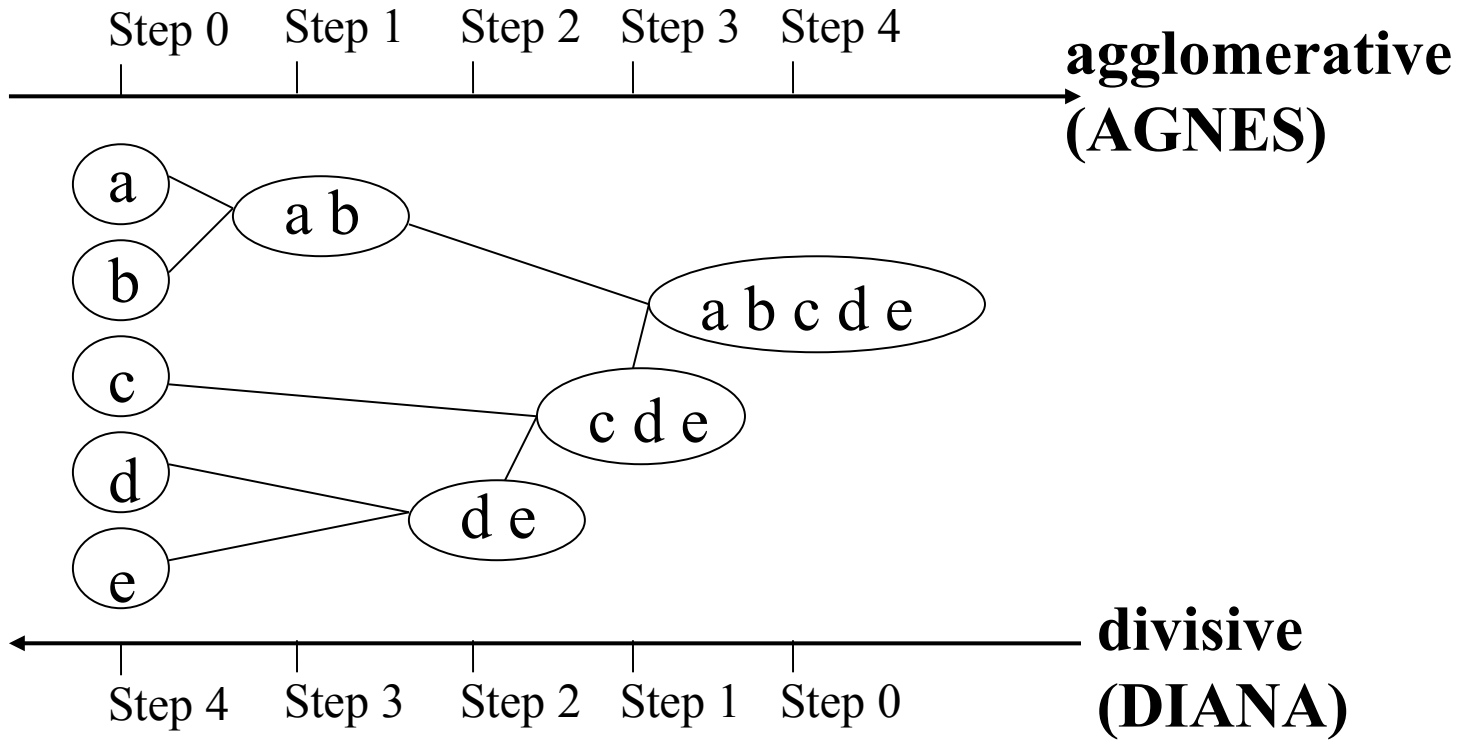
$$cost(x,c)=\sum_{i=1}^{d} ¿\, x_i - c_i \vee ¿¿$$

Affinity Propagation

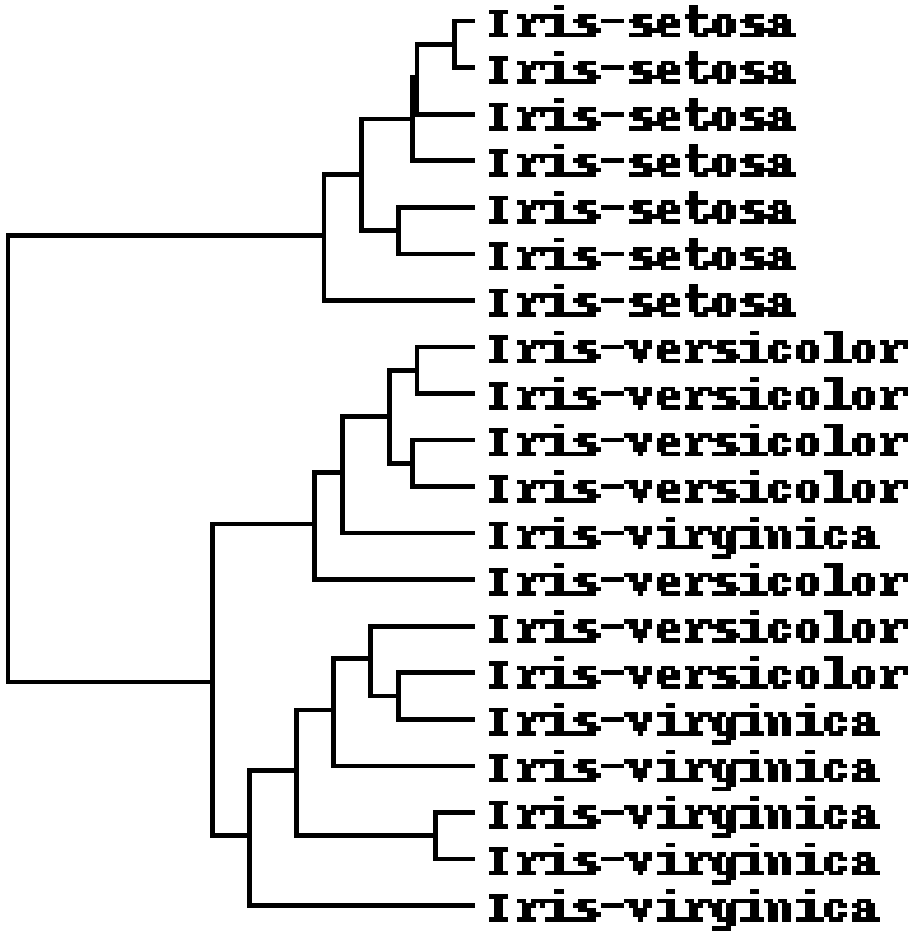


Hierarchical Clustering

- Use *distance matrix* as clustering criteria.
- Does not require the number of clusters ***k*** as an input.
- Needs a termination condition

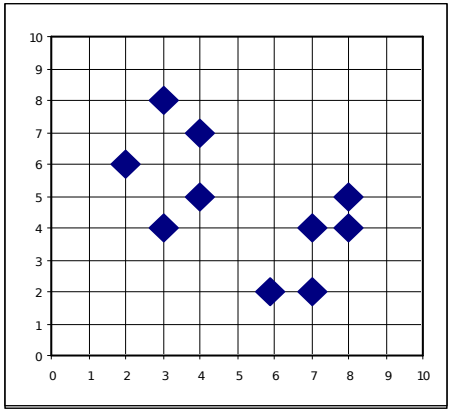


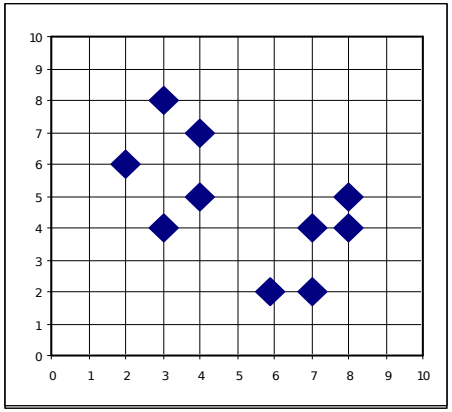
Dendrogram

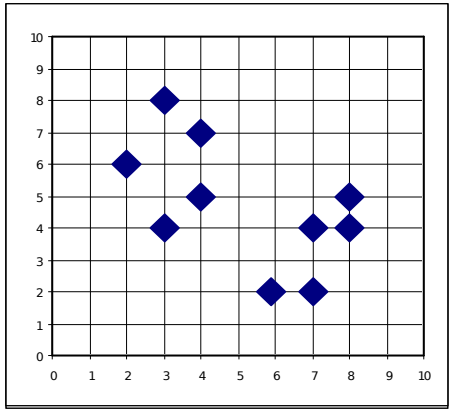

Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-virginica
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-virginica
Iris-virginica
Iris-virginica
Iris-virginica
Iris-virginica

AGNES (Agglomerative Nesting)

- Introduced in Kaufmann and Rousseeuw (1990)
- Use the **single-linkage** method and the dissimilarity matrix
- Merge nodes that have the lowest dissimilarity
- Progress in a non-descending fashion
- Eventually all nodes belong to the same cluster

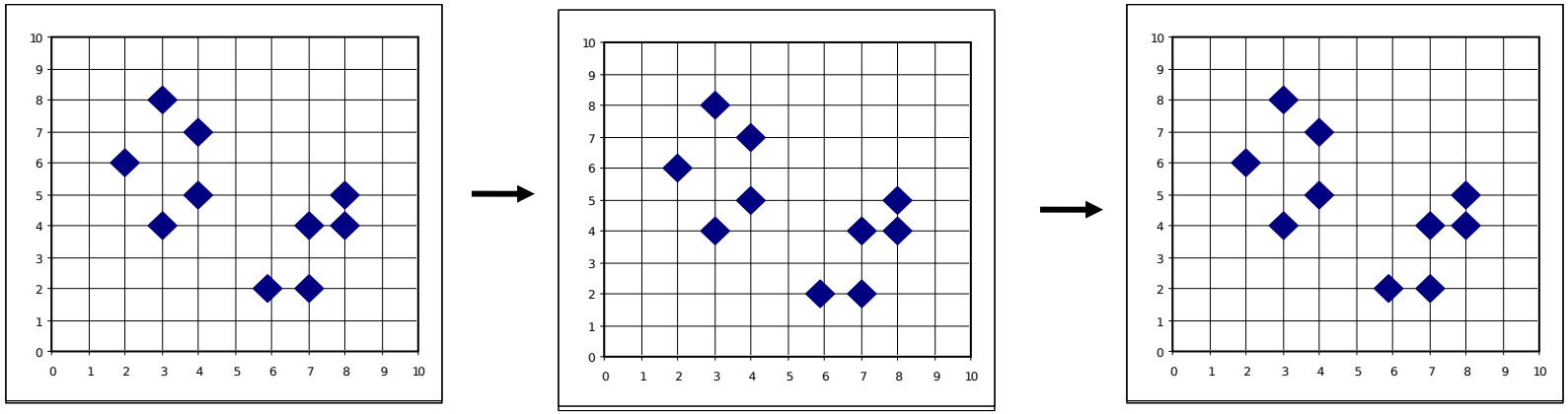






DIANA (Divisive Analysis)

- Introduced in Kaufmann and Rousseeuw (1990)
- Implemented in statistical analysis packages
- Inverse order of AGNES
- Eventually each node forms a cluster on its own

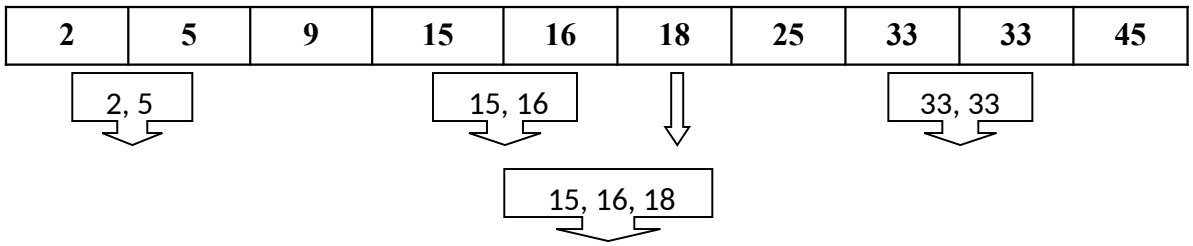


Distance Between Hierarchical Clusters

- A core need is to be able to measure the distance between two hierarchical clusters.
- **Single linkage:** smallest distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = min(t_{ip}, t_{jq})$
- **Complete linkage:** largest distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = max(t_{ip}, t_{jq})$
- **Average linkage:** avg distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = avg(t_{ip}, t_{jq})$
- **Centroid:** distance between the centroids of two clusters, i.e., $dist(K_i, K_j) = dist(C_i, C_j)$
- **Medoid:** distance between the medoids of two clusters, i.e., $dist(K_i, K_j) = dist(M_i, M_j)$
  - Medoid: a chosen, centrally located object in the cluster

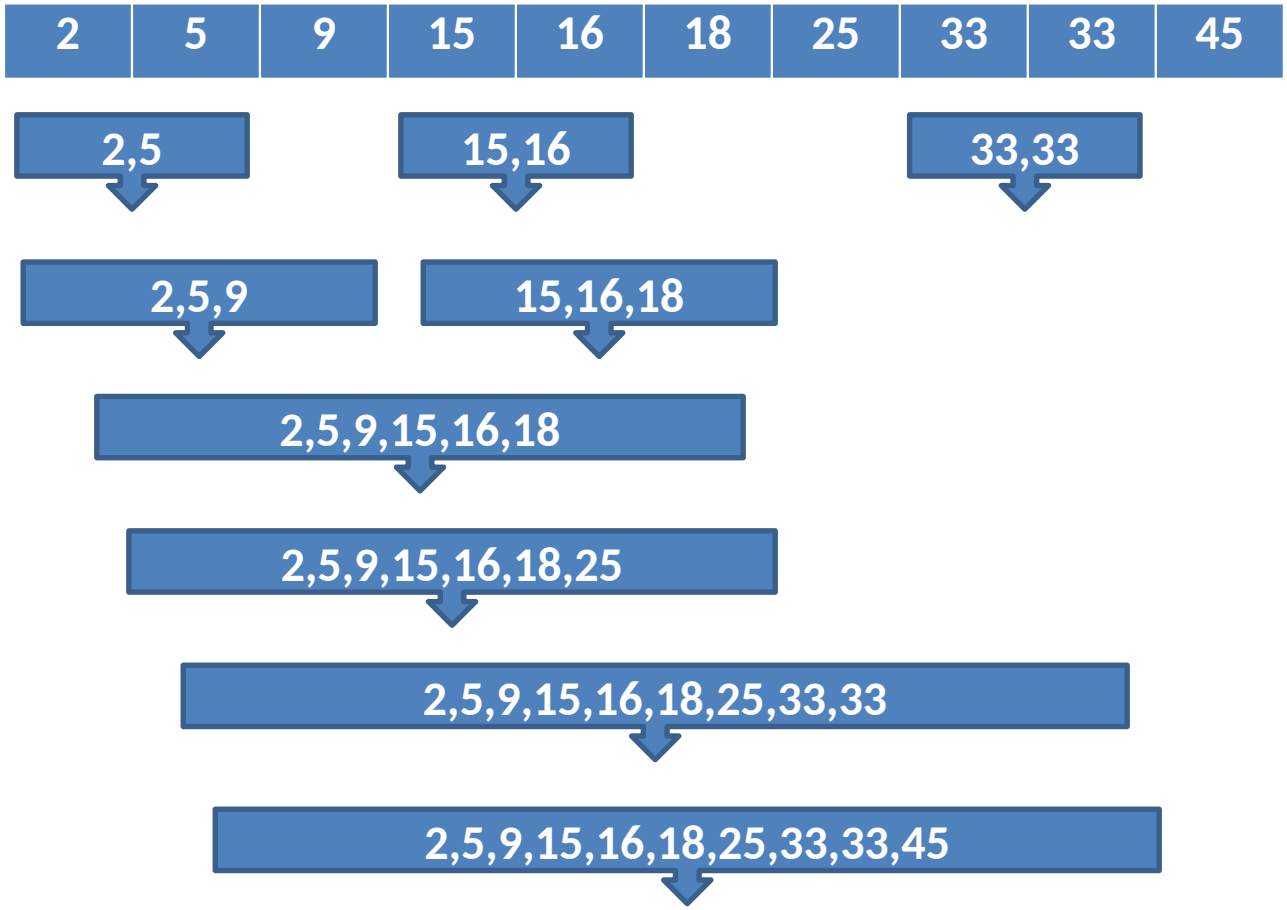
Single-Linkage Clustering

- To begin, each record assigned to own cluster
- Single-linkage seeks minimum distance between any two records, in separate clusters
- Step 1: Minimum cluster distance is between clusters {33} and {33}. Distance = 0, clusters combined
- Step 2: Clusters {15} and {16} combined, where distance = 1
- Step 3: Cluster {15, 16} combined with cluster {18}
- Step 4: Clusters {2} and {5} combined



Single Linkage Example (Agglomerative)

$dist(K_i, K_j) = min(t_{ip}, t_{jq})$



Complete-Linkage Clustering

- Complete-linkage explored using sample data

  `2  5  9  15  16  18  25  33  33  45`

- Distance among records in two clusters farthest from each other minimized

- Step 1: Each cluster contains single record

  No difference between single and Complete-linkage

  Clusters {33} and {33} combined

- Step 2: Clusters {15} and {16} combined

  No difference between single and Complete-linkage

Complete-Linkage Clustering *(cont'd)*

- Step 3: Complete-linkage diverges from Single-linkage

  Farthest neighbors between {15, 16} and {18} are 15 and 18, distance = 3

  Clusters {2} and {5} also have distance = 3

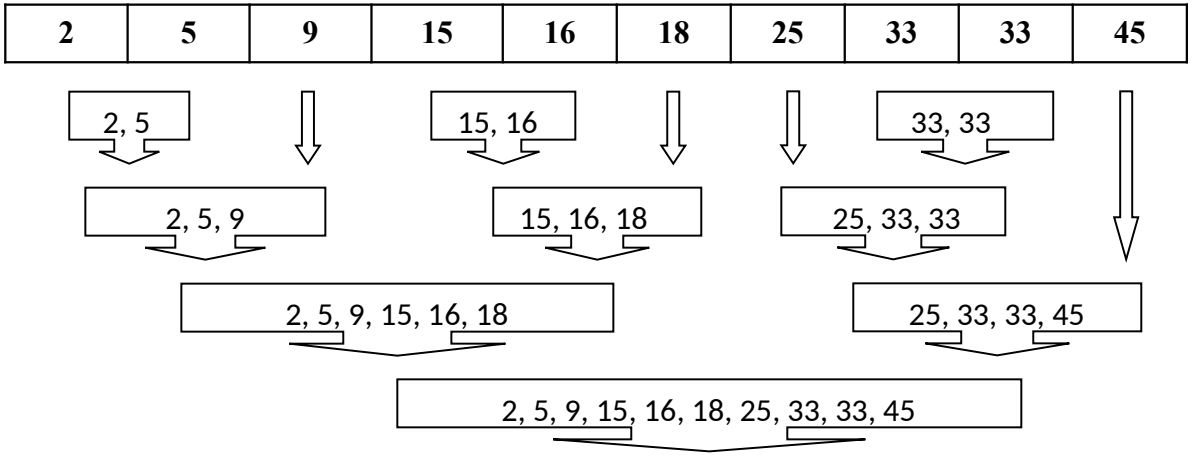
  Algorithm silent regarding ties

  Result, {2, 5} arbitrarily chosen

- Complete-linkage procedure continues for Steps 4 – 9, until all records contained in same cluster

Complete-Linkage Clustering *(cont'd)*

- Figure illustrates Complete-linkage agglomerative clustering on sample data



Extensions to Hierarchical Clustering

- Major weakness of agglomerative clustering methods
  - Can never undo what was done previously
  - Do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects
- Integration of hierarchical & distance-based clustering
  - BIRCH (1996): uses CF-tree and incrementally adjusts the quality of sub-clusters
  - CHAMELEON (1999): hierarchical clustering using dynamic modeling

BIRCH (Balanced iterative Reducing and Clustering Using Hierarchies

- Zhang, Ramakrishnan & Livny, SIGMOD'96
- Integrates hierarchical clustering and iterative partitioning
- Incrementally construct a CF (Clustering Feature) tree, a hierarchical data structure for multiphase clustering
  - Phase 1: scan DB to build an initial in-memory CF tree (a multi-level compression of the data that tries to preserve the inherent clustering structure of the data)
  - Phase 2: use an arbitrary clustering algorithm to cluster the leaf nodes of the CF-tree
- *Scales **linearly***: finds a good clustering with a single scan and improves the quality with a few additional scans
- *Weakness:* handles only numeric data, and is sensitive to the order of the data record

BIRCH

- Designed for *very large* data sets
  - Time and memory are limited
  - Incremental and dynamic clustering of incoming objects
  - Only one scan of data is necessary
  - Does not need the whole data set in advance

Similarity Metric (1)

Given a cluster of instances $\{\vec{X}_i\}$*, we define:*

Centroid: $\vec{X0} = \frac{\sum_{i=1}^{N} \vec{X}_i}{N}$

Radius: *average distance from member points to centroid*

$$R = \left(\frac{\sum_{i=1}^{N} (\vec{X}_i - \vec{X0})^2}{N}\right)^{\frac{1}{2}}$$

Diameter: *average* ***pair-wise*** *distance within a cluster*

$$D = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\vec{X}_i - \vec{X}_j)^2}{N(N-1)}\right)^{\frac{1}{2}}$$

Similarity Metric (2)

Centroid Euclidean distance: $D0 = ((\vec{X0}_1 - \vec{X0}_2)^2)^{\frac{1}{2}}$

Centroid Manhattan distance: $D1 = |\vec{X0}_1 - \vec{X0}_2| = \sum_{i=1}^{d} |\vec{X0}_1^{(i)} - \vec{X0}_1^{(i)}|$

Average inter-cluster: $D2 = (\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{N_1 N_2})^{\frac{1}{2}}$

Average intra-cluster: $D3 = (\frac{\sum_{i=1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{(N_1+N_2)(N_1+N_2-1)})^{\frac{1}{2}}$

Variance increase: $(\sum_{k=1}^{N_1+N_2} (\vec{X}_k - \frac{\sum_{l=1}^{N_1+N_2} \vec{X}_l}{N_1+N_2})^2$

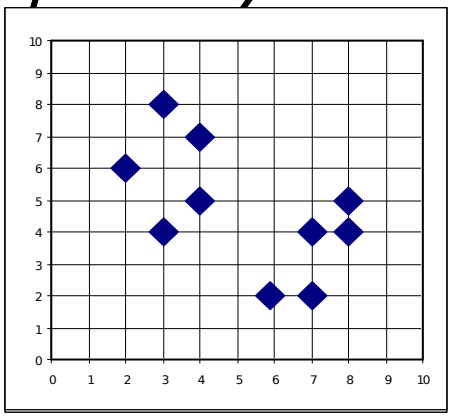
$$- \sum_{i=1}^{N_1} (\vec{X}_i - \frac{\sum_{l=1}^{N_1} \vec{X}_l}{N_1})^2 - \sum_{j=N_1+1}^{N_1+N_2} (\vec{X}_j - \frac{\sum_{l=N_1+1}^{N_1+N_2} \vec{X}_l}{N_2})^2)^{\frac{1}{2}}$$

Clustering Feature Vector in BIRCH

- Clustering Feature (CF): $CF = (N, LS, SS)$
- N: Number of data points
- *LS: linear sum of N points:* $\sum_{i=1}^{N} X_i$
- *SS: square sum of N points*

$$\sum_{i=1}^{N} X_i^2$$

CF = (5, (16,30),(54,190))



(3,4)
(2,6)
(4,5)
(4,7)
(3,8)

Properties of Clustering Feature

- CF entry is more **compact**
  - Stores significantly less than all of the data points in the sub-cluster
- A CF entry has sufficient information to calculate D0-D4
- Additivity theorem allows us to merge sub-clusters incrementally & consistently

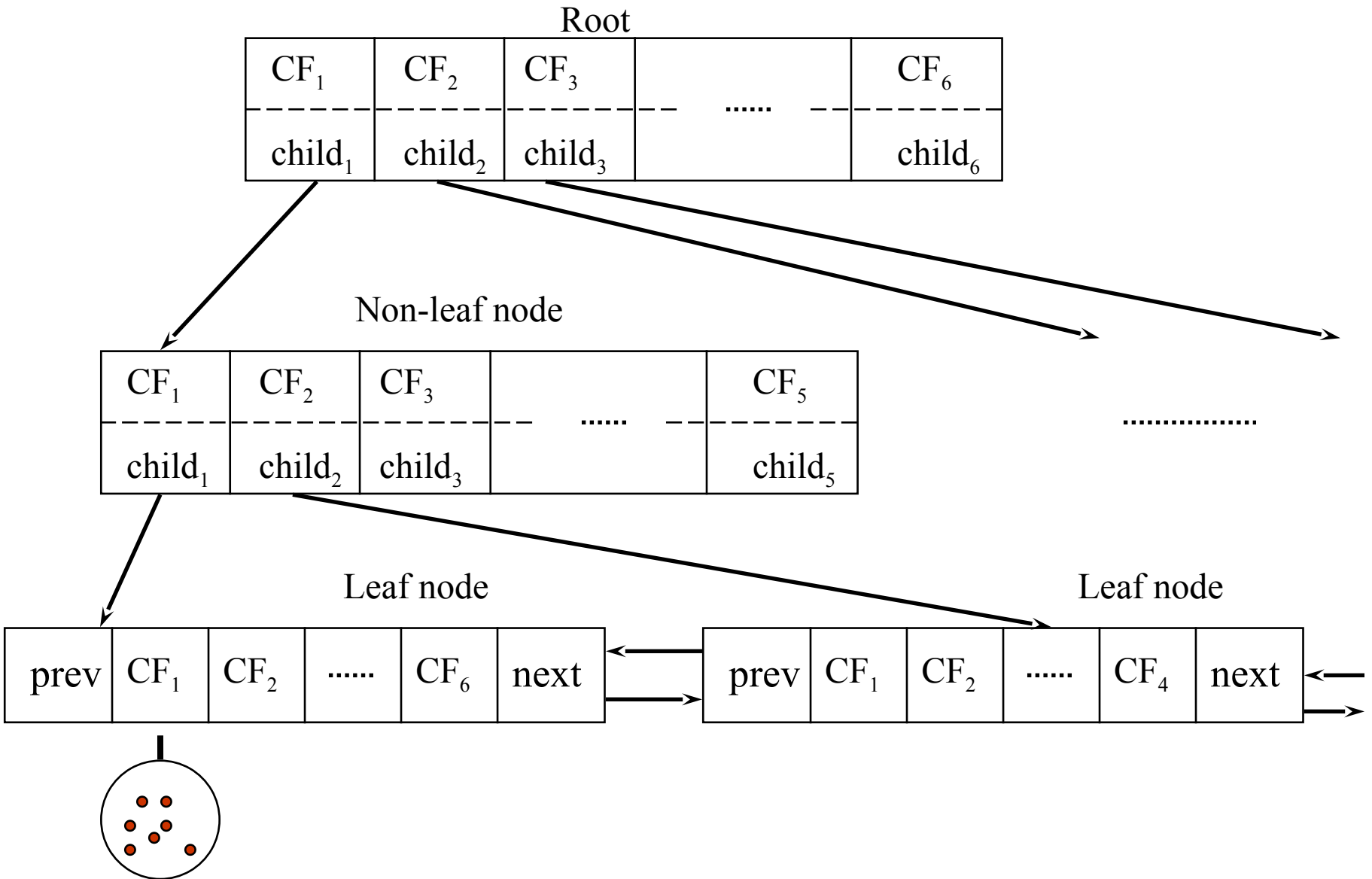
$$\mathbf{CF_1} + \mathbf{CF_2} = (N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, SS_1 + SS_2)$$

CF-Tree in BIRCH

- Clustering feature:
  - ***Summary*** of the statistics for a given sub-cluster
  - Registers crucial measurements for computing cluster and utilizes storage efficiently
- A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering
  - A nonleaf node in a tree has descendants or "children"
  - The nonleaf nodes store sums of the CFs of their children
- A CF tree has two parameters
  - Branching factor: max # of children
  - Threshold: max diameter of sub-clusters stored at the leaf nodes

$$\sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}$$

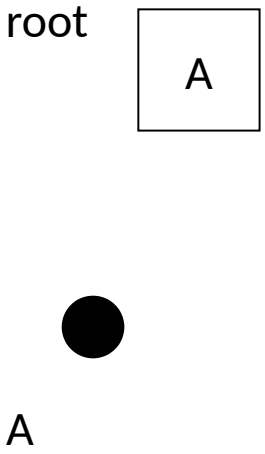
The CF Tree Structure



The BIRCH algorithm

- An example of the CF Tree

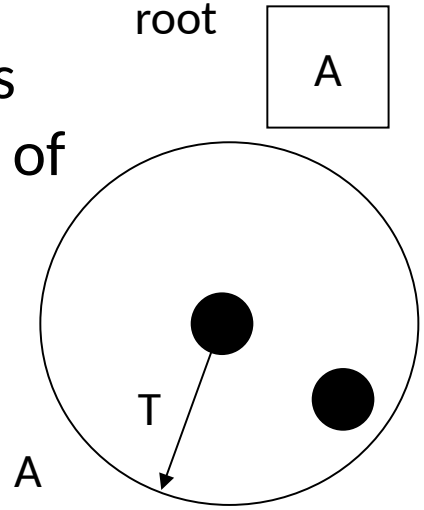
Initially, the data points in one cluster.



The BIRCH algorithm

- An example of the CF Tree

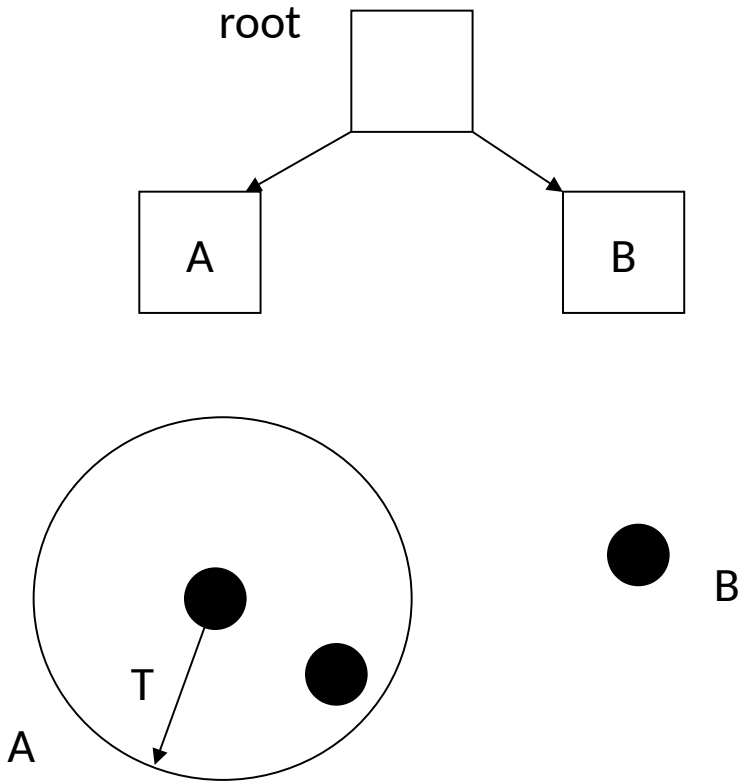
The data arrives, and a check is made whether the size of the cluster does not exceed *T*.



The BIRCH algorithm

- An example of the CF Tree

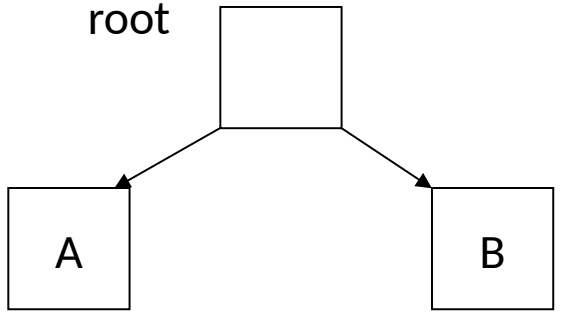
If the cluster size grows
too big, the cluster is split
into two clusters,
and the points
are redistributed.

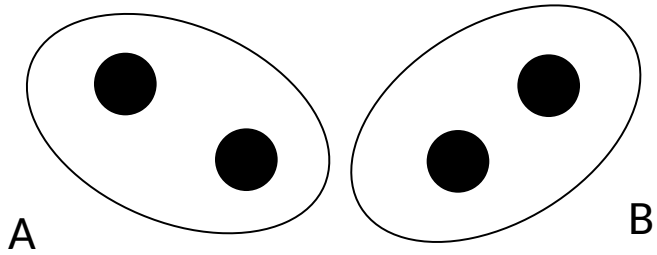


The BIRCH algorithm

- An example of the CF Tree

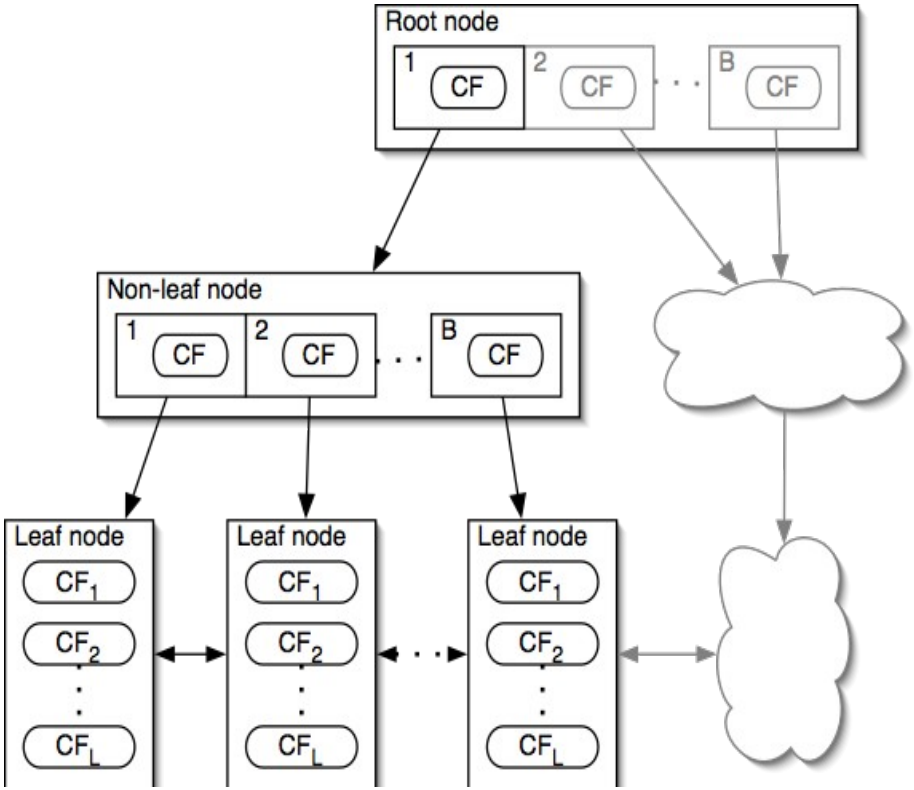
At each node of the tree, the CF tree keeps information about the mean of the cluster, and the mean of the sum of squares to compute the size of the clusters efficiently.





The BIRCH Algorithm

- Each non-leaf node has at most *B* entries
- Each leaf node has at most *L* CF entries which each satisfies threshold *T*
- Node size is determined by dimensionality of data space and input parameter *P* (page size)



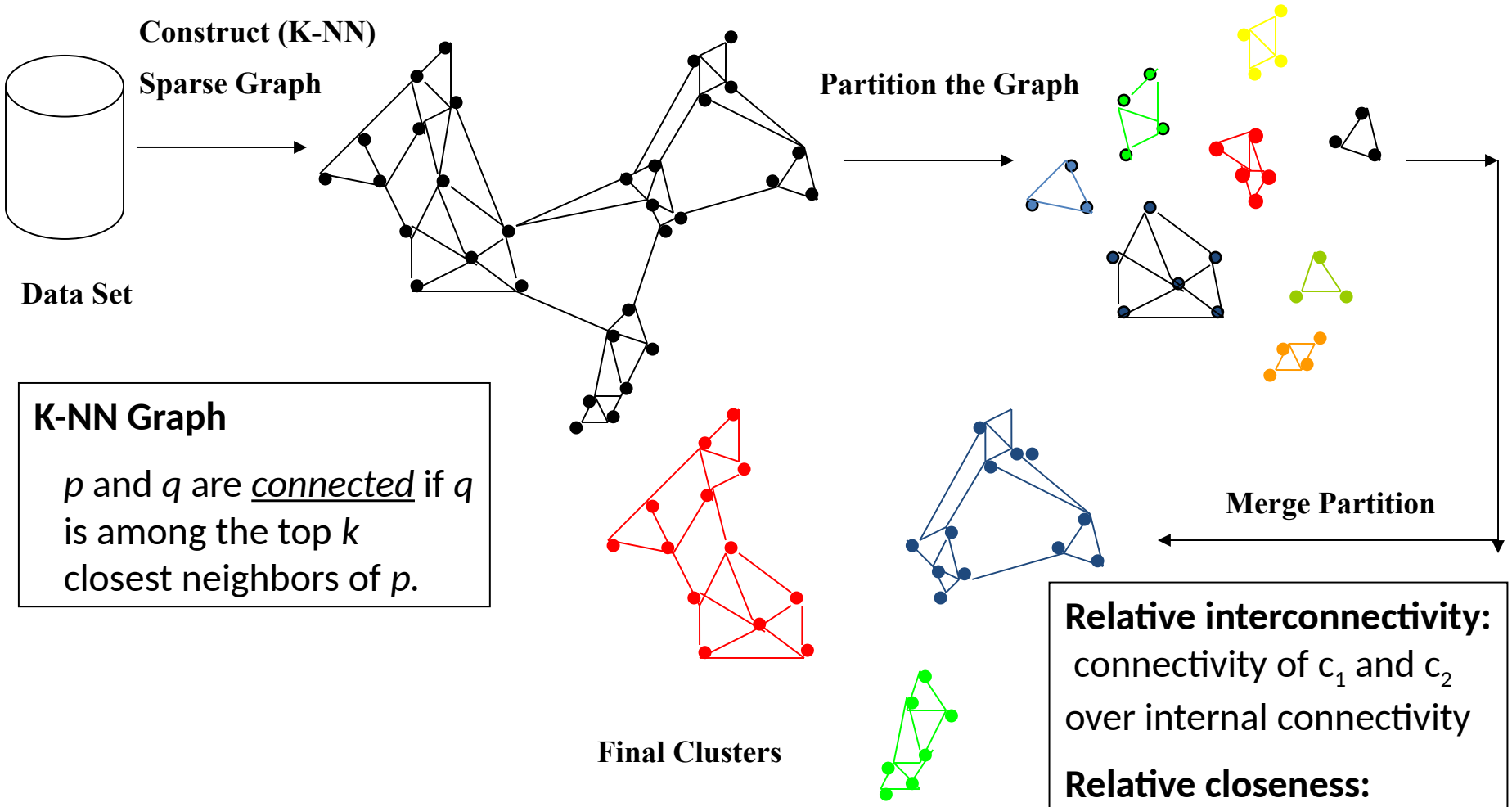
The BIRCH Algorithm

- For each point in the input
  - Find closest leaf entry
  - Add point to leaf entry and update CF
  - If *entry diameter* > *max_diameter*, then split leaf, and possibly parents
- Algorithm is O(n)
- Concerns
  - Sensitive to insertion order of data points (not order invariant)
  - With fixed size of leaf nodes, clusters may not be so natural
  - Clusters tend to be spherical given the radius and diameter measures

CHAMELEON: Hierarchical Clustering Using Dynamic Modeling (1999)

- CHAMELEON: G. Karypis, E. H. Han, and V. Kumar, 1999
- Measures the similarity based on a dynamic model
  - Two clusters are merged *only if* the ***interconnectivity*** and ***closeness (proximity)*** between two clusters are highly *relative to* the internal interconnectivity of the clusters and closeness of items within the clusters
- Graph-based, and a two-phase algorithm
  1. Use a graph-partitioning algorithm: cluster objects into a large number of relatively small sub-clusters
  2. Use an agglomerative hierarchical clustering algorithm: find the genuine clusters by repeatedly combining these sub-clusters

Overall Framework of CHAMELEON



K-NN Graph

p and *q* are *connected* if *q* is among the top *k* closest neighbors of *p*.

Relative interconnectivity: connectivity of c_1 and c_2 over internal connectivity

Relative closeness: closeness of c_1 and c_2 over internal closeness

CHAMELEON (Complex Objects)