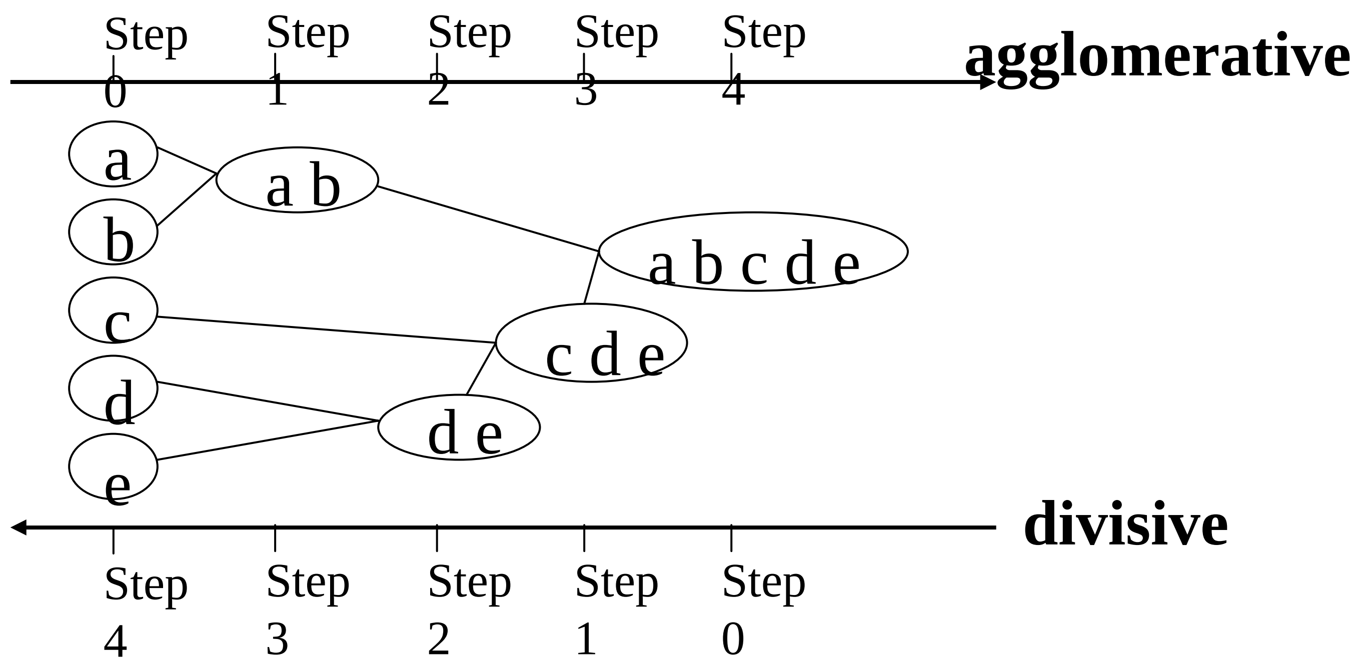
Clustering

Part Two:

Hierarchical clustering approaches

Hierarchical Clustering

- Use distance matrix as clustering criteria. This method does not require the number of clusters ***k*** as an input, but needs a termination condition



Single-link vs. Complete-link

- difference: the way to characterize the similarity between a pair of clusters
  - **single link**: ***minimum*** of the distances between all pairs of patterns drawn from the two clusters
  - **complete link**: ***maximum*** of the distances between all pairs of patterns drawn from the two clusters
  - **Complete link**: average of the distances between all pairs off
- Both use agglomerative clustering control structure

Agglomerative clustering

- Step 1: place each pattern in its own cluster

  construct a list of inter-pattern distances for all distinct unordered pairs of patterns, and sort this list in ascending order
- Step through the sorted list of distances, forming for each distinct dissimilarity value d_k, a graph on the patterns where pairs of patterns closer than d_k are connected by a graph edge.

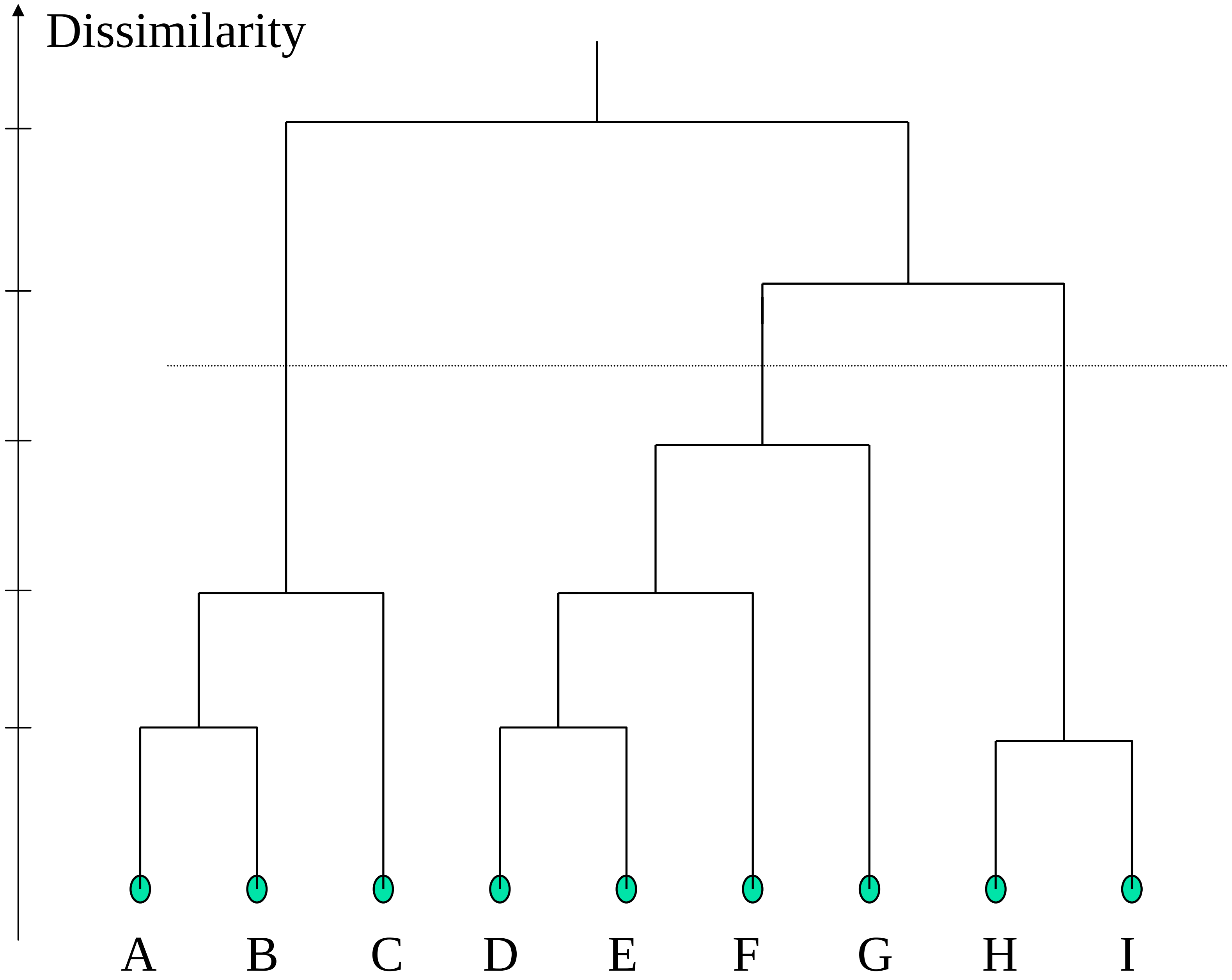
  If all patterns are members of a completely connected graph, stop.

Dendrogram

A *Dendrogram* Shows How the Clusters are Merged Hierarchically:

- Decompose data objects into a several levels of nested partitioning (<u>tree</u> of clusters), called a <u>dendrogram</u>.

- A <u>clustering</u> of the data objects is obtained by <u>cutting</u> the dendrogram at the desired level, then each <u>connected component</u> forms a cluster.


Dissimilarity
A
B
C
D
E
F
G
H
I

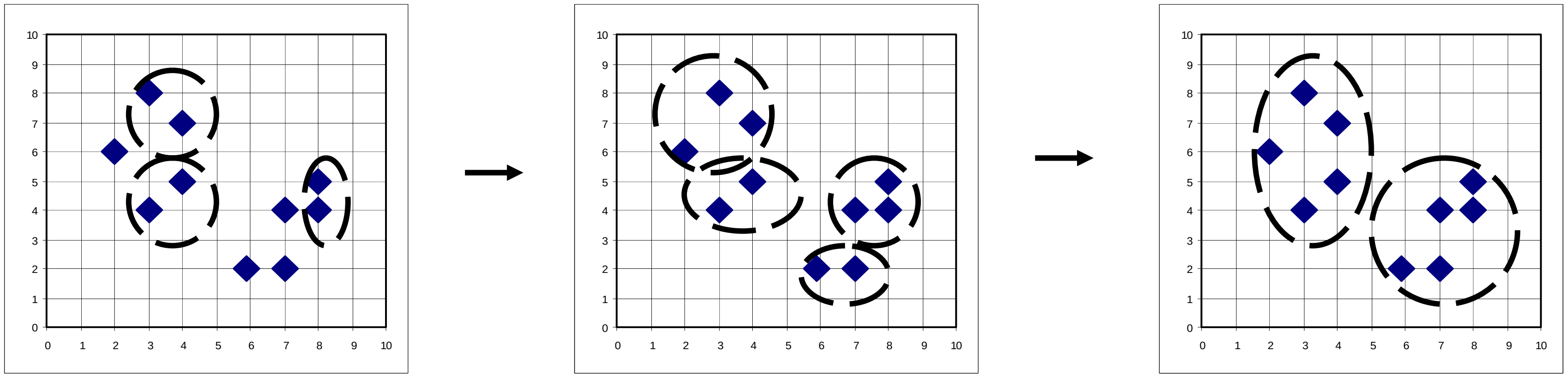
Practice question

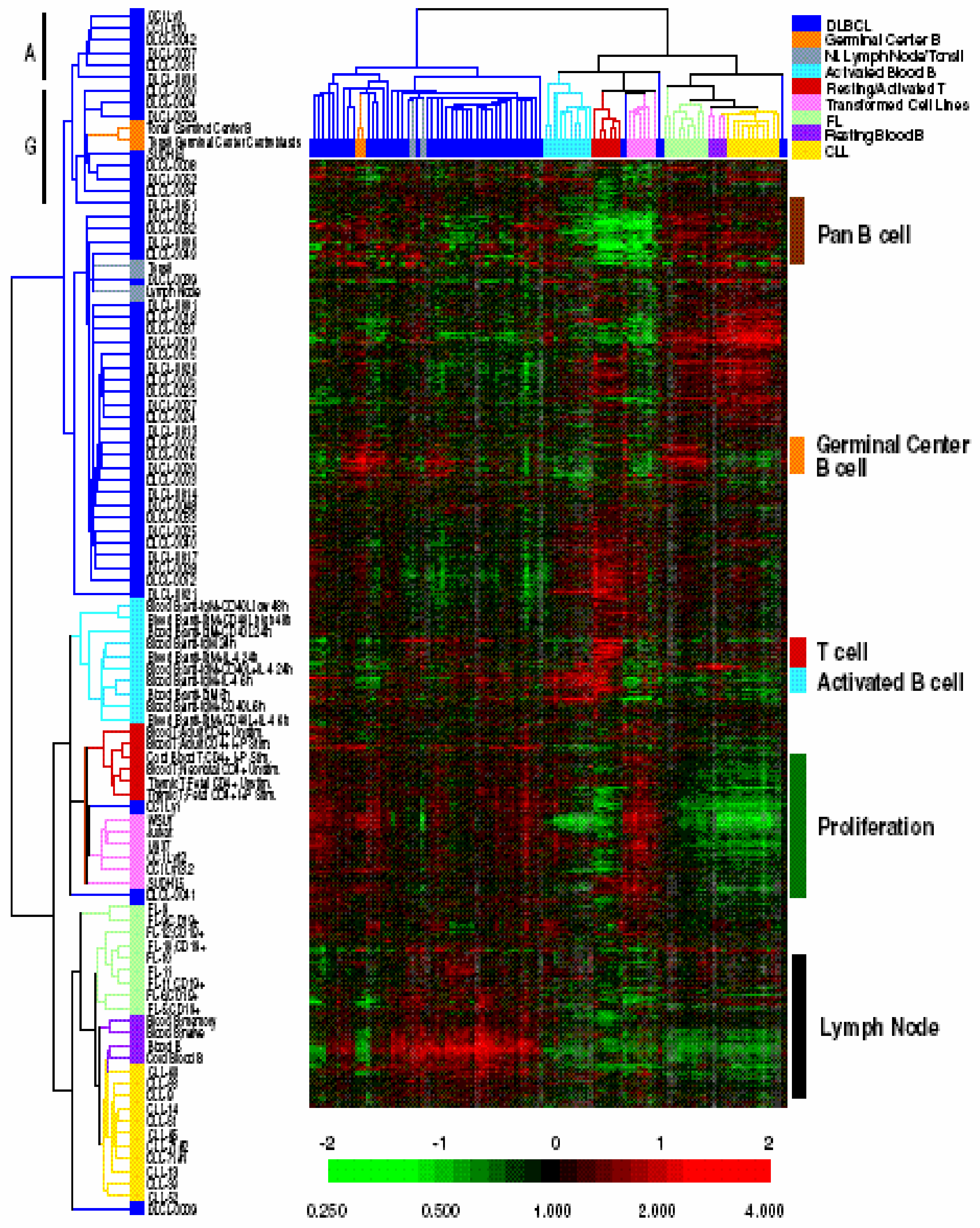
- Cluster the following six objects, using single-link and complete link agglomerative clustering methods:

	Gender	Age	time	Fever	cough
Obj1:	F	23	2	Y	N
Obj2:	M	2	0.5	N	N
Obj3:	F	15	3	Y	Y
Obj4:	F	18	0.5	Y	N
Obj5:	M	58	4	N	Y
Obj6:	F	44	14	N	Y

AGNES (Agglomerative Nesting)

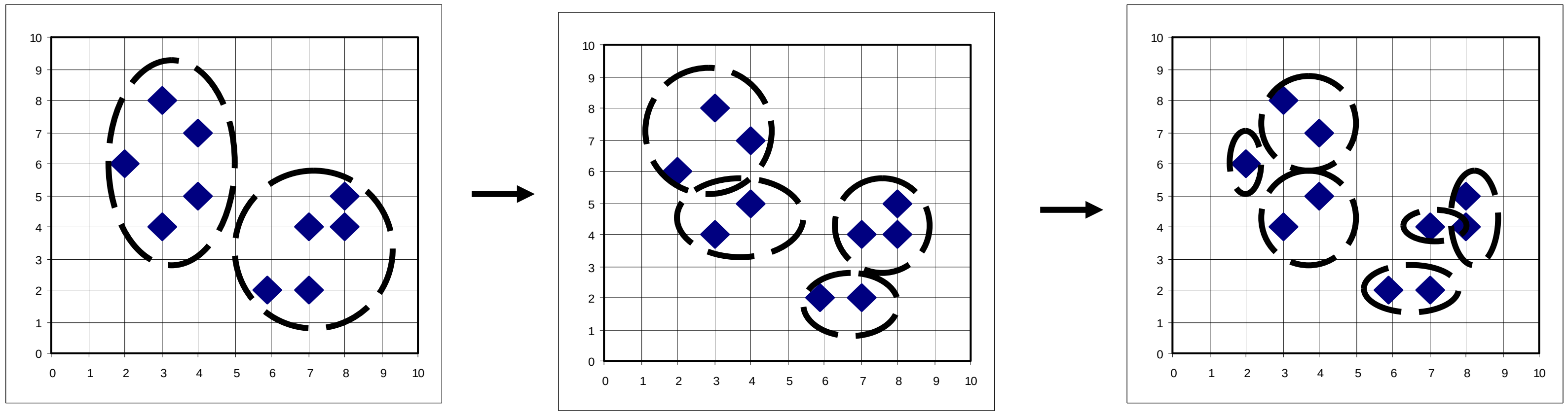
- Implemented in statistical analysis packages, e.g., Splus
- Use the Single-Link method and the dissimilarity matrix.
- Merge nodes that have the least dissimilarity
- Go on in a non-descending fashion
- Eventually all nodes belong to the same cluster




A
G
DLBCL
Germinal Center B
NL Lymph Node/Tonsil
Activated Blood B
Resting/Activated T
Transformed Cell Lines
FL
Resting Blood B
CLL
Pan B cell
Germinal Center B cell
T cell
Activated B cell
Proliferation
Lymph Node
-2
-1
0
1
2
0.250
0.500
1.000
2.000
4.000

DIANA (Divisive Analysis)

- Inverse order of AGNES
- Eventually each node forms a cluster on its own



More on Hierarchical Clustering Methods

- **Major weakness of agglomerative clustering methods**
  - do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects
  - can never undo what was done previously
- **Integration of hierarchical with distance-based clustering**
  - BIRCH (1996): uses CF-tree and incrementally adjusts the quality of sub-clusters
  - CURE (1998): selects well-scattered points from the cluster and then shrinks them towards the center of the cluster by a specified fraction
  - …

Introduction to BIRCH

- Only works with "metric" attributes
  - *Must have Euclidean coordinates*
- Designed for very large data sets
  - *Time and memory constraints are explicit*
  - *Treats dense regions of data points as sub-clusters*
    - *Not all data points are important for clustering*
  - *Only one scan of data is necessary*

Introduction to BIRCH

- Incremental, distance-based approach
  - *Does not need the whole data set in advance*
  - *Unique approach: distance based algorithms generally need all the data points to work*
- Does not assume that the probability distributions on attributes is independent

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

- 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 sensitive to the order of the data record.

Clustering Feature Vector

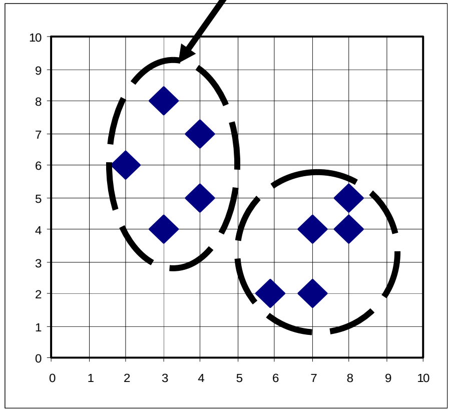
Clustering Feature: $CF = (N, \overrightarrow{LS}, SS)$

N: **Number of data points**

LS: $\sum_{i=1}^{N} \overrightarrow{X_i}$

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

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



(3,4)

(2,6)

(4,5)

(4,7)

(3,8)

The algorithm: background

Given a cluster of instances $\{\vec{X}_i\}$*, we define the* ***centroid****, the* ***radius****, and the* ***diameter****:*

$$\vec{X}0 = \frac{\sum_{i=1}^{N} \vec{X}_i}{N}$$

$$R = \left(\frac{\sum_{i=1}^{N} (\vec{X}_i - \vec{X}0)^2}{N}\right)^{\frac{1}{2}}$$

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

The algorithm: background

We define the ***Euclidean*** *and* ***Manhattan*** *distance between any two clusters as:*

$$D0 = ((\vec{X}0_1 - \vec{X}0_2)^2)^{\frac{1}{2}}$$

$$D1 = |\vec{X}0_1 - \vec{X}0_2| = \sum_{i=1}^{d} |\vec{X}0_1^{(i)} - \vec{X}0_1^{(i)}|$$

The algorithm: background

We define the ***average inter-cluster****, the* ***average intra-cluster****, and the* ***variance increase*** *distances as:*

$$D2 = \left(\frac{\sum_{i=1}^{N_1}\sum_{j=N_1+1}^{N_1+N_2}(\vec{X}_i - \vec{X}_j)^2}{N_1 N_2}\right)^{\frac{1}{2}}$$

$$D3 = \left(\frac{\sum_{i=1}^{N_1+N_2}\sum_{j=1}^{N_1+N_2}(\vec{X}_i - \vec{X}_j)^2}{(N_1+N_2)(N_1+N_2-1)}\right)^{\frac{1}{2}}$$

$$D4 = \Bigg(\sum_{k=1}^{N_1+N_2}\left(\vec{X}_k - \frac{\sum_{l=1}^{N_1+N_2}\vec{X}_l}{N_1+N_2}\right)^2 - \sum_{i=1}^{N_1}\left(\vec{X}_i - \frac{\sum_{l=1}^{N_1}\vec{X}_l}{N_1}\right)^2 - \sum_{j=N_1+1}^{N_1+N_2}\left(\vec{X}_j - \frac{\sum_{l=N_1+1}^{N_1+N_2}\vec{X}_l}{N_2}\right)^2\Bigg)^{\frac{1}{2}}$$

The algorithm: background

- Preprocessing data is optional
- Can not affect relative placement
  - *If point A is left of B, then after preprocessing A must still be to the left of B*
- Avoids bias caused by dimensions with a large spread
- Large spread may naturally describe data

The algorithm: CF

*A **Clustering Feature** (CF) summarizes a sub-cluster of data points.*

Given a cluster $\{\vec{X}_1, \vec{X}_2, \ldots, \vec{X}_N\}$

$$\mathbf{CF} = (N, \vec{LS}, SS)$$

N is the number of data points

$\vec{LS} = \sum_{i=1}^{N} \vec{X}_i$

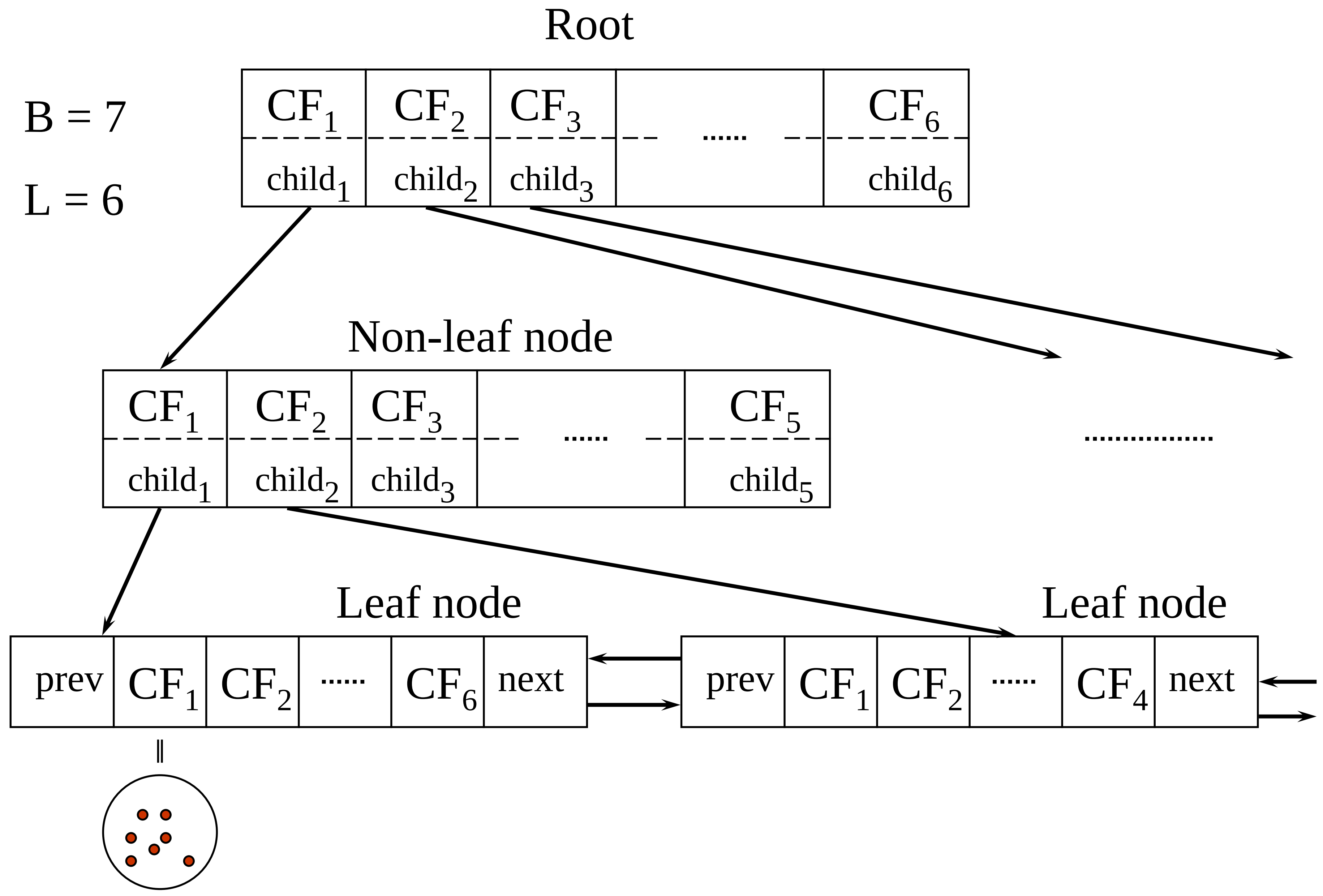
$SS = \sum_{i=1}^{N} \vec{X}_i^{\,2}$

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

The algorithm: CF

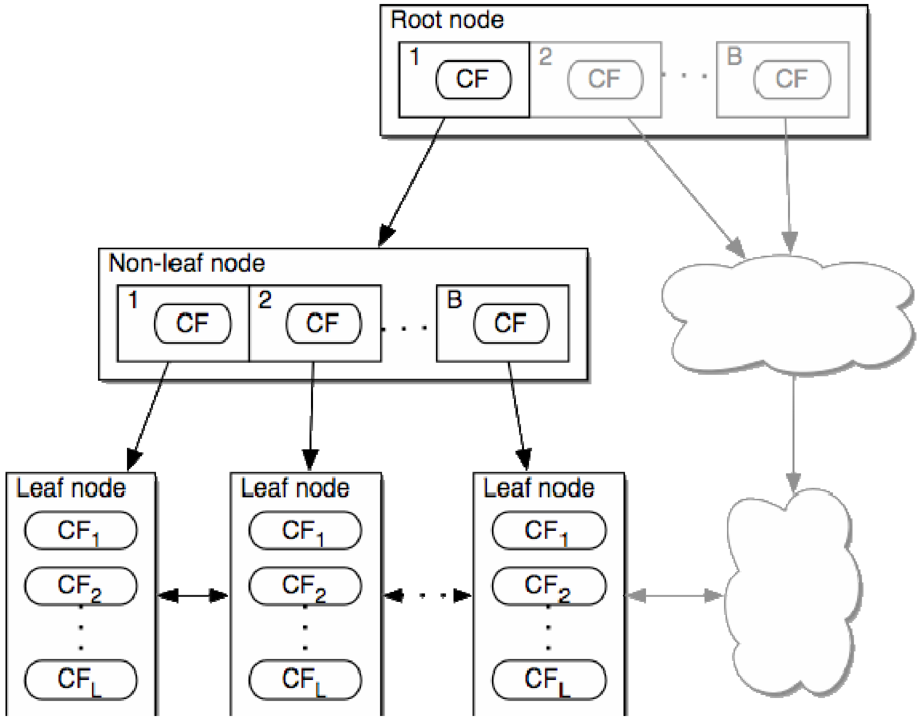
- CF entry is more compact
  - *Stores significantly less then all of the data points in the sub-cluster*
- A CF entry has enough information to calculate D0-D4
- Additivity theorem allows us to incrementally merge sub-clusters

The algorithm: CF-tree


Root
B = 7
L = 6
CF1 CF2 CF3 ...... CF6
child1 child2 child3 child6
Non-leaf node
CF1 CF2 CF3 ...... CF5
child1 child2 child3 child5
....
Leaf node
prev CF1 CF2 ...... CF6 next
Leaf node
prev CF1 CF2 ...... CF4 next
||

The algorithm: CF-tree

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



The algorithm: CF-tree insert

- Recurse down from root
  - *Choose the "closest" CF and go to that node*
- Modify the leaf
  - *If the closest CF in the leaf can not absorb, make a new CF entry. If there is no room, split the node*
- Traverse back up
  - *Modifying CFs or splitting nodes*

The algorithm: CF-tree rebuild

- If we run out of space, increase T
  - *By increasing the threshold, CFs absorb more data*
- Rebuilding "pushes" CFs over
  - *The larger* T *allows different CFs to group together*
- Reducibility theorem
  - *Increasing* T *will result in a CF-tree as small or smaller then the original*

The algorithm: BIRCH

- Phase 1: Load data into memory
  - *Build a CF-tree with the data*
- Phase 2: Condense data
  - *Rebuild the CF-tree with a larger* T
  - *Condensing is optional*

The algorithm: BIRCH

- Phase 3: Global clustering
  - *Use existing clustering algorithm on CF entries*
  - *Helps fix problem where natural clusters span nodes*
- Phase 4: Cluster refining
  - *Do additional passes over the data set and reassign data points to the closest centroid from phase 3*
  - *Refining is optional*

The algorithm: BIRCH

- Why have optional phases?
  - *Phase 2 allows us to resize the data set so Phase 3 runs on an optimally sized data set*
  - *Phase 4 fixes a problem with CF-trees where some data points may be assigned to different leaf entries*
  - *Phase 4 will always converge to a minimum*
  - *Phase 4 allows us to discard outliers*

The algorithm: BIRCH

- Rebuild CF-tree with smallest T
  - *Start with T=0 and try rebuilding the tree*
- Get rid of outliers
  - *Write outliers to special place outside of the tree*
- Delayed split
  - *Treat data points that force a split like outliers*

Experimental results

- Input parameters:
  - *Memory (*M*): 5% of data set*
  - *Disk space (*R*): 20% of* M
  - *Distance equation:* D2
  - *Quality equation: weighted average diameter (*D*)*
  - *Initial threshold (*T*): 0.0*
  - *Page size (*P*): 1024 bytes*

Experimental results

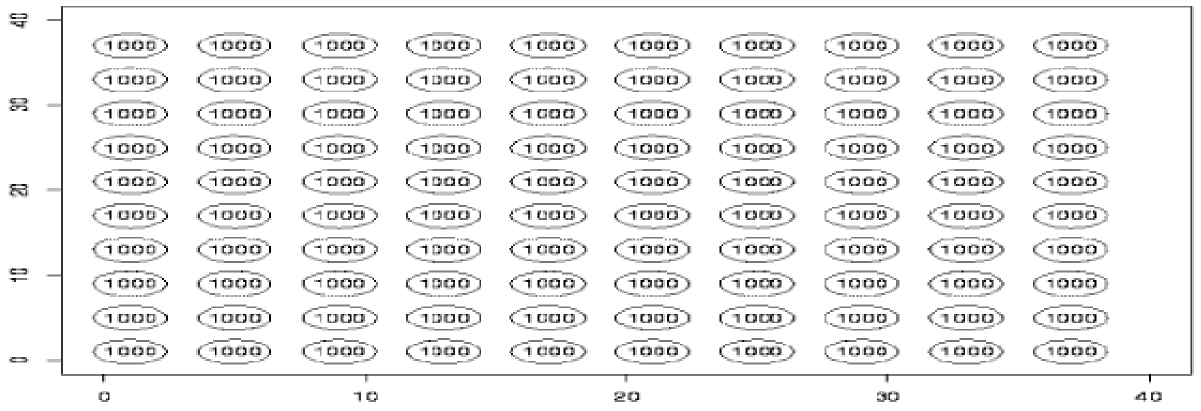
- The Phase 3 algorithm used is called an agglomerative Hierarchical Clustering (HC) algorithm
- One refinement pass
  - *Outlier discarding off*
- Delay-split is on
  - *This is what we use disk space R for*

Experimental results

- Create 3 synthetic data sets for testing
  - *Also create an ordered copy for testing input order*
- KMEANS and CLARANS require entire data set to be in memory
  - *Initial scan is from disk, subsequent scans are in memory*

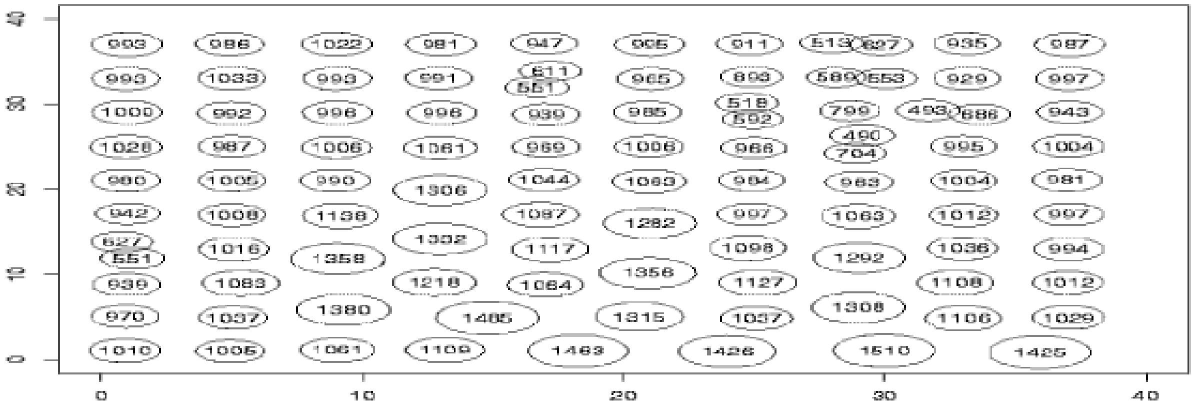
Experimental results

Intended clustering



Experimental results

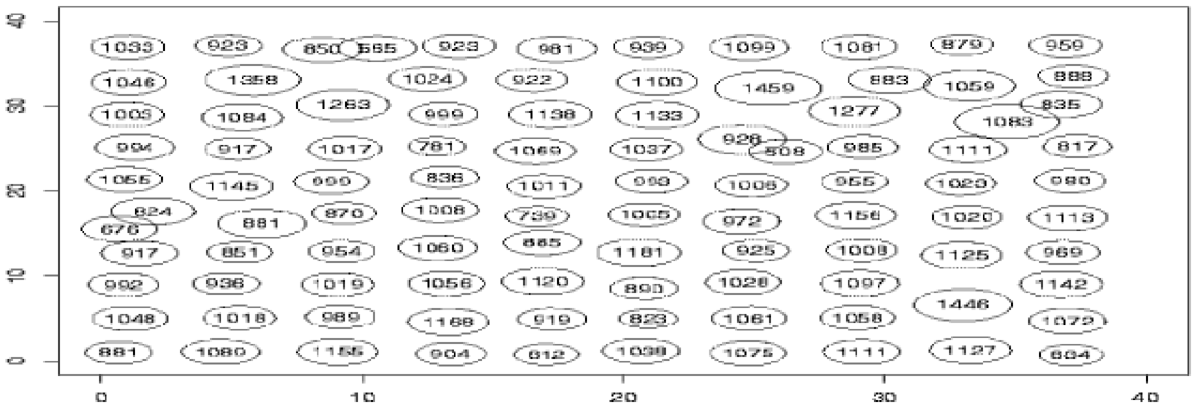
KMEANS clustering



DS	*Time*	*D*	*# Scan*	*DS*	*Time*	*D*	*# Scan*
1	**43.9**	**2.09**	**289**	1o	33.8	1.97	197
2	13.2	4.43	51	2o	12.7	4.20	29
3	32.9	3.66	187	3o	36.0	4.35	241

Experimental results

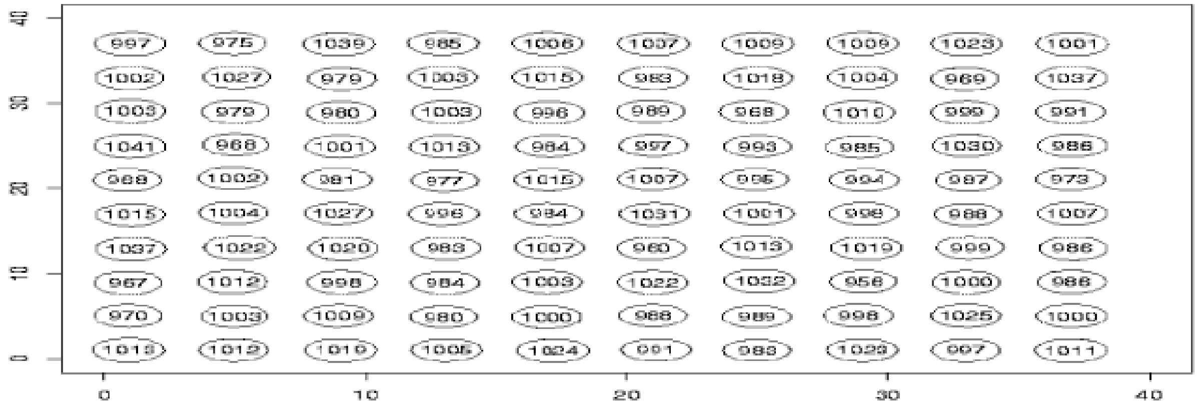
CLARANS clustering



DS	*Time*	*D*	*# Scan*	*DS*	*Time*	*D*	*# Scan*
1	**932**	**2.10**	**3307**	1o	794	2.11	2854
2	758	2.63	2661	2o	816	2.31	2933
3	835	3.39	2959	3o	924	3.28	3369

Experimental results

BIRCH clustering



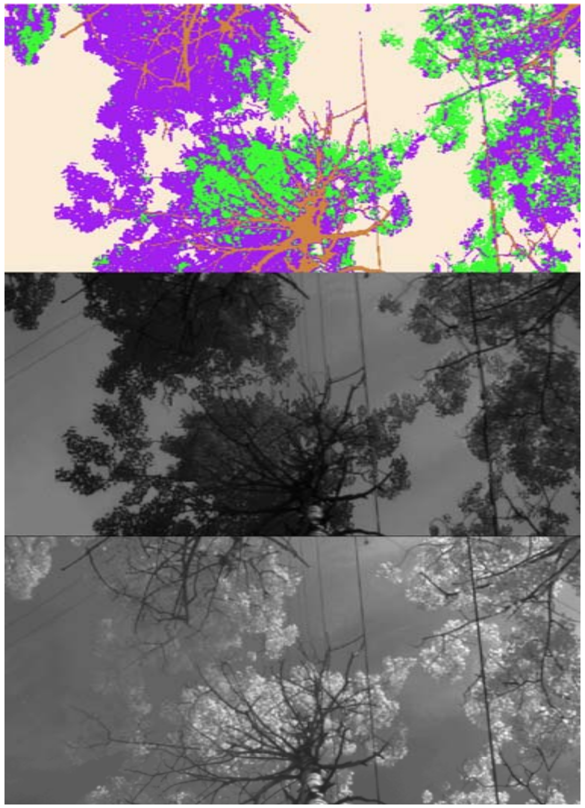
DS	*Time*	*D*	*# Scan*	*DS*	*Time*	*D*	*# Scan*
1	**11.5**	**1.87**	**2**	1o	13.6	1.87	2
2	10.7	1.99	2	2o	12.1	1.99	2
3	11.4	3.95	2	3o	12.2	3.99	2

Experimental results

- Page size
  - *When using Phase 4 P can vary from 256 to 4096 with out much effect on the final results*
- Memory vs. Time
  - *Results generated with low memory can be compensated for by multiple iterations of Phase 4*
- Scalability

Conclusions and practical use

- Pixel classification in images
- From top to bottom:
  - *BIRCH classification*
  - *Visible wavelength band*
  - *Near-infrared band*



Conclusions and practical use

- Image compression using vector quantization
- Generate codebook for frequently occurring patterns
- BIRCH performs faster then CLARANS or LBG, while getting better compression and nearly as good quality

Conclusions and practical use

- BIRCH works with very large data sets
- Explicitly bounded by computational resources
  - *Runs with specified amount of memory (P)*
- Superior to CLARANS and KMEANS
  - *Quality, speed, stability and scalability*

CF tree

- CF tree is height balanced tree CF (B,T)
  - Branching factor (B)
  - Threshold (T)
  - Node (leaf node and non-leaf node) represents a cluster made up of all the subclusters represented by its entries
    - Non-leaf node contains at most B entries of the form [CFi, childi]
    - Leaf node contains at most L entries(objects), each entry represents a sub-cluster, which absorbs many data points with diameter (or radius) under the specified threshold value.

Insertion into a CF tree

- To insert an object into a CF tree:
  1. Identifying the appropriate leaf
  2. Modifying the leaf
     1. Absorb by a leaf entry
     2. Create a new leaf entry
        1. No leaf splitting
        2. Leaf splitting
  3. Modifying the path to the leaf
     1. No leaf/node split
     2. Leaf/node split
  4. Merging refinement
     1. No resplitting
     2. resplitting