# Data Mining: Foundation, Techniques and Applications

## Lesson 10: Mining and Searching High Dimensional Data



Li Cuiping(李翠平)
School of Information
Renmin University of China


Anthony Tung(鄧锦浩)
School of Computing
National University of Singapore

# Outline

- Sources of HDD
- Challenges of HDD
- Foundation
  - Similarity Function
  - High Dimensional Distance Join
- Techniques & Application
  - Finding Nonlinear Correlated Clusters in High Dimensional Data
  - Finding Patterns in Extremely High Dimensional Data

# Sources of High Dimensional Data

- Microarray gene expression
- Text documents
- Images
- Features of Sequences, Trees and Graphs
- Audio, Video, Human Motion Database (spatio-temporal as well!)

# Challenges of High Dimensional Data

- Indistinguishable
  - Distance between two nearest points and two furthest points could be almost the same
- Sparsity
  - As a result of the above, data distribution are very sparse giving no obvious indication on where the interesting knowledge is
- Large number of combination
  - Efficiency: How to test the number of combinations
  - Effectiveness: How do we understand and interpret so many combinations?

# Outline

- Sources of HDD
- Challenges of HDD
- Foundation
  - Similarity Function
  - High Dimensional Distance Join
- Techniques & Application
  - Finding Nonlinear Correlated Clusters in High Dimensional Data
  - Finding Patterns in Extremely High Dimensional Data

# Similarity Search : Traditional Approach

- **Objects represented by multidimensional vectors**

| Elevation | Aspect | Slope | Hillshade (9am) | Hillshade (noon) | Hillshade (3pm) | ... |
|---|---|---|---|---|---|---|
| 2596 | 51 | 3 | 221 | 232 | 148 | |
| ... | | | | | | |

- **The traditional approach to similarity search: kNN query**

$Q$ = ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

| ID | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 | Dist |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 1.1 | 1 | 1.2 | 1.6 | 1.1 | 1.6 | 1.2 | 1.2 | 1 | 1 | **0.93** |
| P2 | 1.4 | 1.4 | 1.4 | 1.5 | 1.4 | 1 | 1.2 | 1.2 | 1 | 1 | **0.98** |
| P3 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 2 | **1.73** |
| P4 | 20 | 20 | 21 | 20 | 22 | 20 | 20 | 19 | 20 | 20 | **57.7** |
| P5 | 19 | 21 | 20 | 20 | 20 | 21 | 18 | 20 | 22 | 20 | **60.5** |
| P6 | 21 | 21 | 18 | 19 | 20 | 19 | 21 | 20 | 20 | 20 | **59.8** |

# Deficiencies of the Traditional Approach

- **Deficiencies**
  - Distance is affected by a few dimensions with high dissimilarity
  - Partial similarities can not be discovered

- **The traditional approach to similarity search: kNN query**

  $Q$ = ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

| ID | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 | Dist |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 1.1 | **100** | 1.2 | 1.6 | 1.1 | 1.6 | 1.2 | 1.2 | 1 | 1 | **99.0** |
| P2 | 1.4 | 1.4 | 1.4 | 1.5 | 1.4 | **100** | 1.2 | 1.2 | 1 | 1 | **99.0** |
| P3 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | **100** | 2 | 2 | **99.0** |
| P4 | 20 | 20 | 21 | 20 | 22 | 20 | 20 | 19 | 20 | 20 | **57.7** |
| P5 | 19 | 21 | 20 | 20 | 20 | 21 | 18 | 20 | 22 | 20 | **60.5** |
| P6 | 21 | 21 | 18 | 19 | 20 | 19 | 21 | 20 | 20 | 20 | **59.8** |

# Thoughts

- Aggregating too many dimensional differences into a single value result in too much information loss. Can we try to reduce that loss?
- While high dimensional data typically give us problem when in come to similarity search, can we turn what is against us into advantage?
- Our approach: Since we have so many dimensions, we can compute more complex statistics over these dimensions to overcome some of the “noise” introduce due to scaling of dimensions, outliers etc.

# The *N*-Match Query : Warm-Up

- **Description**
  - Matches between two objects in *n* dimensions. ($n \leqslant d$)
  - The *n* dimensions are chosen dynamically to make the two objects match best.
- **How to define a "match"**
  - Exact match
  - Match with tolerance $\delta$
- **The similarity search example**

  $Q$ = ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)  ***n* = 6**

| ID | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 | Dist |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 1.1 | **100** | 1.2 | 1.6 | 1.1 | 1.6 | 1.2 | 1.2 | 1 | 1 | 0.2 |
| P2 | 1.4 | 1.4 | 1.4 | 1.5 | 1.4 | **100** | 1.2 | 1.2 | 1 | 1 | 0.4 |
| P3 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | **100** | 2 | 2 | 0 |
| P4 | 20 | 20 | 21 | 20 | 22 | 20 | 20 | 19 | 20 | 20 | 19 |
| P5 | 19 | 21 | 20 | 20 | 20 | 21 | 18 | 20 | 22 | 20 | 19 |
| P6 | 21 | 21 | 18 | 19 | 20 | 19 | 21 | 20 | 20 | 20 | 19 |

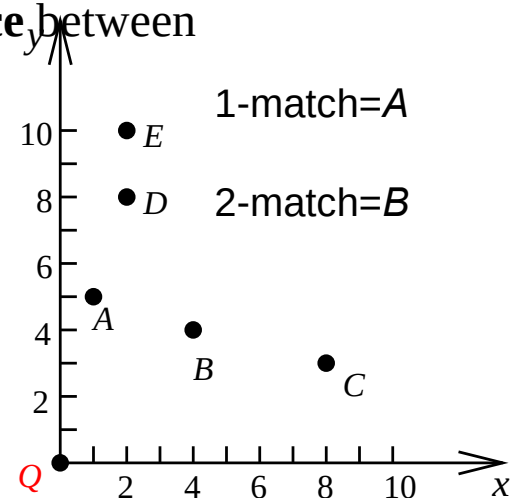
# The *N*-Match Query : The Definition

- **The *n*-match difference**

  Given two *d*-dimensional points $P(p_1, p_2, \ldots, p_d)$ and $Q(q_1, q_2, \ldots, q_d)$, let $\delta_i = |p_i - q_i|$, $i=1,\ldots,d$. Sort the array $\{\delta_1, \ldots, \delta_d\}$ in increasing order and let the sorted array be $\{\delta_1', \ldots, \delta_d'\}$. Then $\delta_n'$ is the **n-match difference** between *P* and *Q*.

- **The *n*-match query**

  Given a *d*-dimensional database *DB*, a query point *Q* and an integer *n* ($n \leqslant d$), find the point $P \in DB$ that has the smallest *n*-match difference to *Q*. *P* is called the ***n*-match** of *Q*.




- **The similarity search example**

  $Q$ = ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

  ***n* = 8**

| ID | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 | Dist |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 1.1 | **100** | 1.2 | 1.6 | 1.1 | 1.6 | 1.2 | 1.2 | 1 | 1 | 0.6 |
| P2 | 1.4 | 1.4 | 1.4 | 1.5 | 1.4 | **100** | 1.2 | 1.2 | 1 | 1 | 0.4 |
| P3 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | **100** | 2 | 2 | 1 |
| P4 | 20 | 20 | 21 | 20 | 22 | 20 | 20 | 19 | 20 | 20 | 19 |
| P5 | 19 | 21 | 20 | 20 | 20 | 21 | 18 | 20 | 22 | 20 | 19 |
| P6 | 21 | 21 | 18 | 19 | 20 | 19 | 21 | 20 | 20 | 20 | 19 |

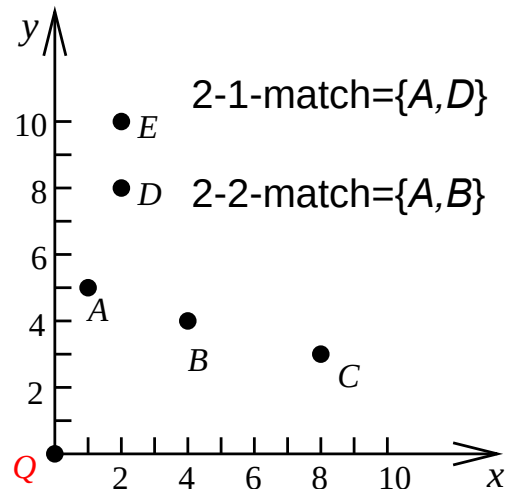
# The *N*-Match Query : Extensions

- **The *k*-*n*-match query**

  Given a *d*-dimensional database *DB*, a query point *Q*, an integer *k*, and an integer *n*, find a set *S* which consists of *k* points from *DB* so that for any point $P1 \in S$ and any point $P2 \in DB\text{-}S$, *P1*'s n-match difference is smaller than *P2*'s *n*-match difference. *S* is called the ***k*-*n*-match** of *Q*.

- **The frequent *k*-*n*-match query**

  Given a *d*-dimensional database *DB*, a query point *Q*, an integer *k*, and an integer range $[n_0, n_1]$ within [1,*d*], let $S_0, \ldots, S_i$ be the answer sets of *k*-$n_0$-match, …, *k*-$n_1$-match, respectively, find a set *T* of *k* points, so that for any point $P1 \in T$ and any point $P2 \in DB\text{-}T$, *P1*'s number of appearances in $S_0, \ldots, S_i$ is larger than or equal to *P2*'s number of appearances in $S_0, \ldots, S_i$ .




- **The similarity search example**

$Q$ = ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)  **$n$ = 6**

| ID | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 | Dist |
|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 1.1 | **100** | 1.2 | 1.6 | 1.1 | 1.6 | 1.2 | 1.2 | 1 | 1 | 0.2 |
| P2 | 1.4 | 1.4 | 1.4 | 1.5 | 1.4 | **100** | 1.2 | 1.2 | 1 | 1 | 0.4 |
| P3 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | **100** | 2 | 2 | 0 |
| P4 | 20 | 20 | 21 | 20 | 22 | 20 | 20 | 19 | 20 | 20 | 19 |
| P5 | 19 | 21 | 20 | 20 | 20 | 21 | 18 | 20 | 22 | 20 | 19 |
| P6 | 21 | 21 | 18 | 19 | 20 | 19 | 21 | 20 | 20 | 20 | 19 |

# Cost Model

- **The multiple system information retrieval model**
  - Objects are stored in different systems and scored by each system
  - Each system can sort the objects according to their scores
  - A query retrieves the scores of objects from different systems and then combine them using some aggregation function

**Q : color="red" & shape="round" & texture "cloud"**

| System 1: Color | |
|---|---|
| Object ID | Score |
| 1 | 0.4 |
| 2 | 2.8 |
| 5 | 6.5 |
| 3 | 9.5 |
| 4 | 9.6 |

| System 2: Shape | |
|---|---|
| Object ID | Score |
| 1 | 1.0 |
| 5 | 5.5 |
| 2 | 7.8 |
| 3 | 9.0 |
| 4 | 1.5 |

| System 3: Texture | |
|---|---|
| Object ID | Score |
| 1 | 1.0 |
| 2 | 2.0 |
| 3 | 5.0 |
| 5 | 9.0 |
| 4 | 8.0 |

- **The cost**
  - Retrieval of scores – proportional to the number of scores retrieved
- **The goal**
  - To minimize the scores retrieved

# The AD Algorithm

- **The AD algorithm for the *k*-*n*-match query**
  - Locate the query's attributes value in every dimension
  - Retrieve the objects' attributes value from the query's attributes in both directions
  - The objects' attributes are retrieved in Ascending order of their Differences to the query's attributes. An n-match is found when it appears n times.

**Q : color="red" and shape="round" and texture "cloud"**

| d1 | |
|---|---|
| Object ID | Attr |
| 1 | 0.4 |
| 2 | 2.8 |
| 5 | 3.5 |
| 3 | 6.5 |
| 4 | 9.0 |

3.0

| d2 | |
|---|---|
| Object ID | Attr |
| 1 | 1.0 |
| 5 | 1.5 |
| 2 | 5.5 |
| 3 | 7.8 |
| 4 | 9.0 |

7.0

| d3 | |
|---|---|
| Object ID | Attr |
| 1 | 1.0 |
| 2 | 2.0 |
| 3 | 5.0 |
| 5 | 8.0 |
| 4 | 9.0 |

4.0

## Auxiliary structures

- Next attribute to retrieve *g*[2d]

| d1 | | d2 | | d3 | |
|---|---|---|---|---|---|
| 1 , 2.6 | 3 , 3.5 | 2 , 1.5 | 4 , 2.0 | 2 , 2.0 | 5 , 4.0 |

- Number of appearances *appear*[c]

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 0 | 2 | 2 | 0 | 1 |

- Answer set *S*

**{ 3 , 2 }**

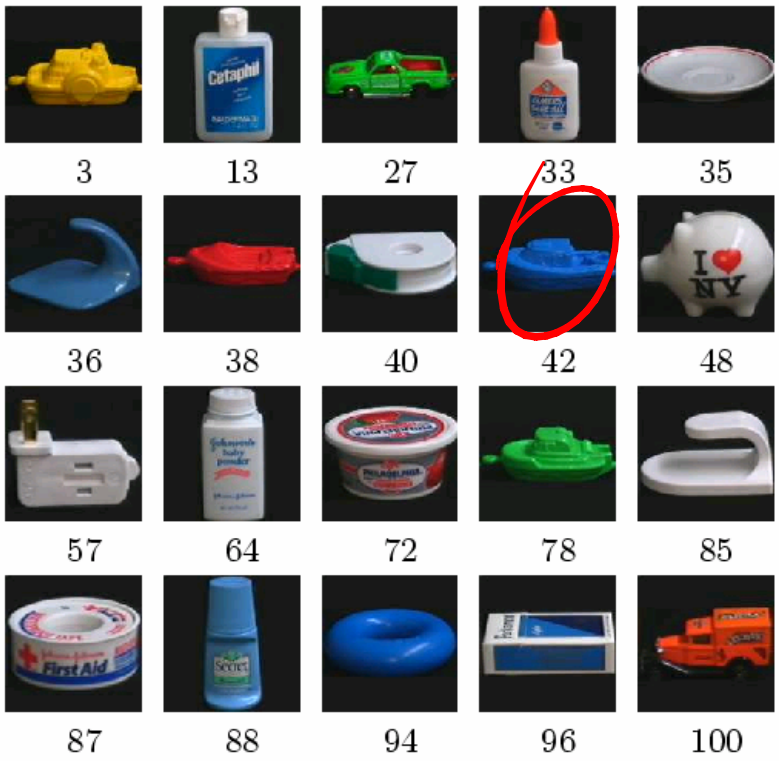
# The AD Algorithm : Extensions

- **The AD algorithm for the frequent *k*-*n*-match query**
  - The frequent *k*-*n*-match query
    - Given an integer range $[n_0, n_1]$, find *k*-$n_0$-match, *k*-($n_0$+1)-match, … , *k*-$n_1$-match of the query, $S_0, S_1, \ldots, S_i$.
    - Find *k* objects that appear most frequently in $S_0, S_1, \ldots, S_i$.
  - Retrieve the same number of attributes as processing a *k*-$n_1$-match query.
- **Disk based solutions for the (frequent) *k*-*n*-match query**
  - Disk based AD algorithm
    - Sort each dimension and store them sequentially on the disk
    - When reaching the end of a disk page, read the next page from disk
  - Existing indexing techniques
    - Tree-like structures: R-trees, k-d-trees
    - Mapping based indexing: space-filling curves, iDistance
    - Sequential scan
    - Compression based approach (VA-file)

# Experiments : Effectiveness

## Searching by *k*-*n*-match

- COIL-100 database
- 54 features extracted, such as color histograms, area moments




| *k*-*n*-match query, *k*=4 | |
|---|---|
| *n* | Images returned |
| 5 | 36, 42, 78, 94 |
| 10 | 27, 35, 42, 78 |
| 15 | 3, 38, 42, 78 |
| 20 | 27, 38, 42, 78 |
| 25 | 35, 40, 42, 94 |
| 30 | 10, 35, 42, 94 |
| 35 | 35, 42, 94, 96 |
| 40 | 35, 42, 94, 96 |
| 45 | 35, 42, 94, 96 |
| 50 | 35, 42, 94, 96 |

| *k*NN query | |
|---|---|
| *k* | Images returned |
| 10 | 13, 35, 36, 40, 42<br>64, 85, 88, 94, 96 |

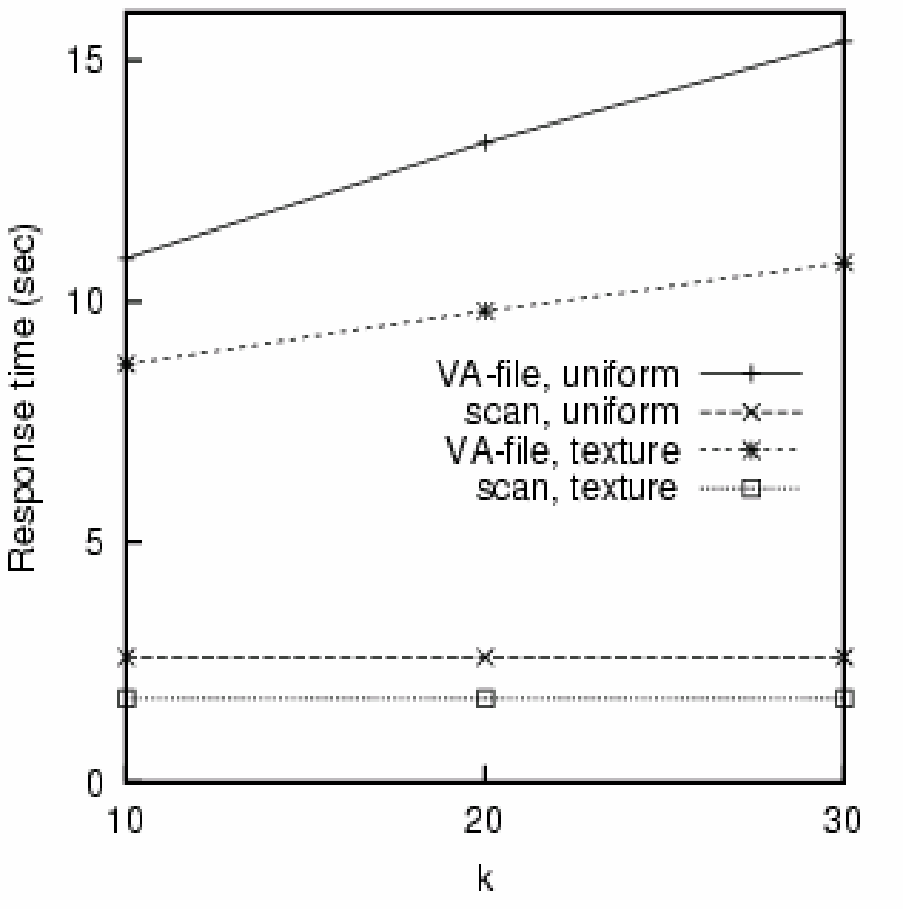
## Searching by frequent *k*-*n*-match

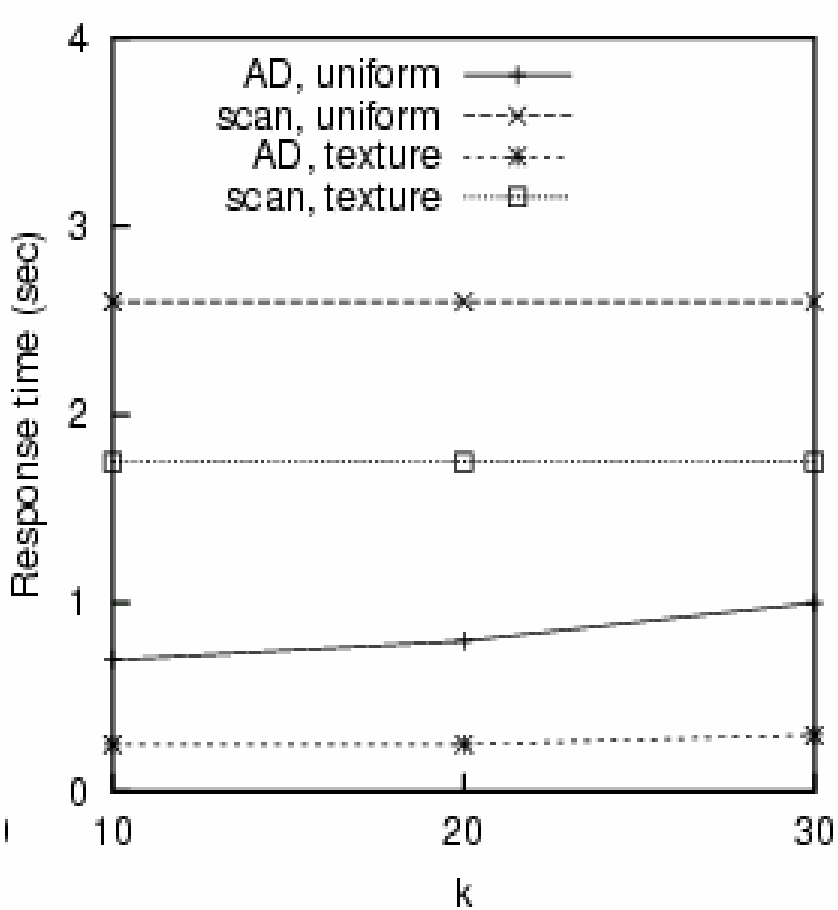
- UCI Machine learning repository
- Competitors:
  - IGrid
  - Human-Computer Interactive NN search (HCINN)

| Data sets (*d*) | IGrid | HCINN | Freq. *k*-*n*-match |
|---|---|---|---|
| Ionosphere (34) | 80.1% | 86% | 87.5% |
| Segmentation (19) | 79.9% | 83% | 87.3% |
| Wdbc (30) | 87.1% | N.A. | 92.5% |
| Glass (9) | 58.6% | N.A. | 67.8% |
| Iris (4) | 88.9% | N.A. | 89.6% |

# Experiments : Efficiency

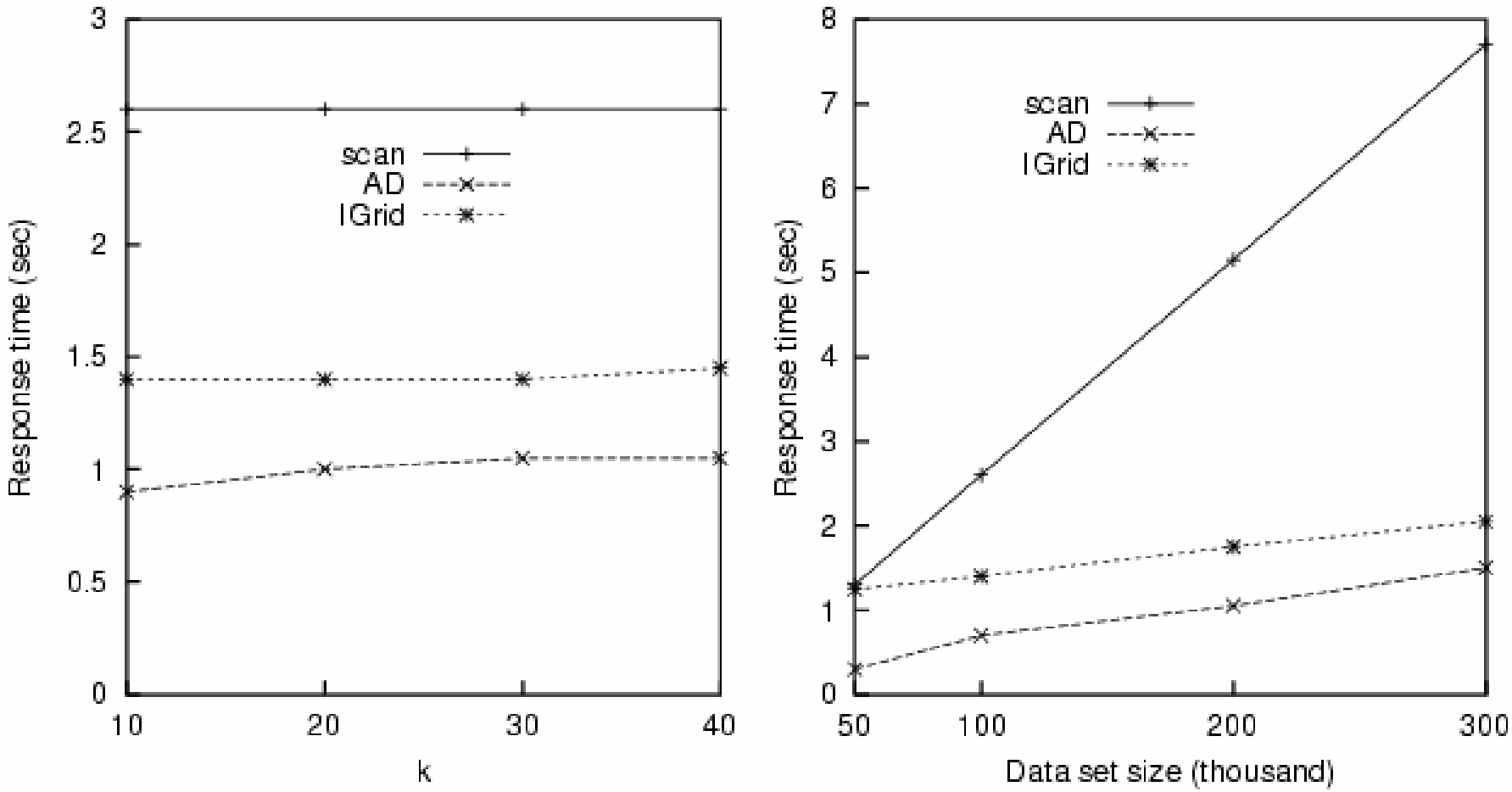
- **Disk based algorithms for the Frequent *k*-*n*-mach query**
  - Texture dataset (68,040 records); uniform dataset (100,000 records)
  - Competitors:
    - The AD algorithm
    - VA-file
    - Sequential scan



# Experiments : Efficiency (continued)

- **Comparison with other similarity search techniques**
  - Texture dataset ; synthetic dataset
  - Competitors:
    - Frequent *k*-*n*-match query using the AD algorithm
    - IGrid
    - scan



# Future Work(I)

- We now have a natural way to handle similarity search for data with categorical , numerical and attributes. Investigating k-n-match performance on such mixed-type data is currently under way
- Likewise, applying k-n-match on data with missing or uncertain attributes will be interesting
- Query={1,1,1,1,1,1,1,M,No,R}

| ID | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 |
|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 1.1 | 1 | 1.2 | 1.6 | 1.1 | 1.6 | 1.2 | M | Yes | R |
| P2 | 1.4 | 1.4 | 1.4 | 1.5 | 1.4 | 1 | 1.2 | F | No | B |
| P3 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | M | No | B |
| P4 | 20 | 20 | 21 | 20 | 22 | 20 | 20 | M | Yes | G |
| P5 | 19 | 21 | 20 | 20 | 20 | 21 | 18 | F | Yes | R |
| P6 | 21 | 21 | 18 | 19 | 20 | 19 | 21 | F | Yes | Y |

# Future Work(I)

- We now have a natural way to handle similarity search for data with categorical , numerical and attributes. Investigating k-n-match performance on such mixed-type data is currently under way
- Likewise, applying k-n-match on data with missing or uncertain attributes will be interesting
- Query={1,1,1,1,1,1,1,M,No,R}

| ID | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 |
|---|---|---|---|---|---|---|---|---|---|---|
| P1 | | 1 | 1.2 | 1.6 | 1.1 | 1.6 | 1.2 | M | | R |
| P2 | 1.4 | 1.4 | | 1.5 | | 1 | 1.2 | F | No | B |
| P3 | 1 | 1 | 1 | 1 | 1 | | 2 | M | No | B |
| P4 | 20 | 20 | | 20 | 22 | 20 | 20 | M | | G |
| P5 | 19 | 21 | 20 | 20 | 20 | | 18 | | Yes | R |
| P6 | 21 | | 18 | | 20 | | 21 | F | Yes | Y |

# Future Work(II)

- In general, three things affect the result from a similarity search: noise, scaling and axes orientation. K-n-match reduce the effect of noise. Ultimate aim is to have a similarity function that is robust to noise, scaling and axes orientation
- Eventually will look at creating mining algorithms using k-n-match

# Outline

- Sources of HDD
- Challenges of HDD
- Foundation
  - Similarity Function
  - High Dimensional Distance Join
- Techniques & Application
  - Finding Nonlinear Correlated Clusters in High Dimensional Data
  - Finding Patterns in Extremely High Dimensional Data

# KNN-Join

- KNN-Join between R and S
  - assigns to each point in R its **K-nearest neighbours** in S
  - based on a distance function
- Important in many data mining algorithms
  - k-mean clustering
  - LOF (intensity based outlier detection)

# KNN-Join: Existing Algorithms

- Only up-to-date algorithm: MuX
- uses **R-tree** as index structure
- R-tree problems with **many dimensions**
  - R-tree performance degenerates
  - High memory requirement for the index structure (for storing high dimensional bounding boxes)

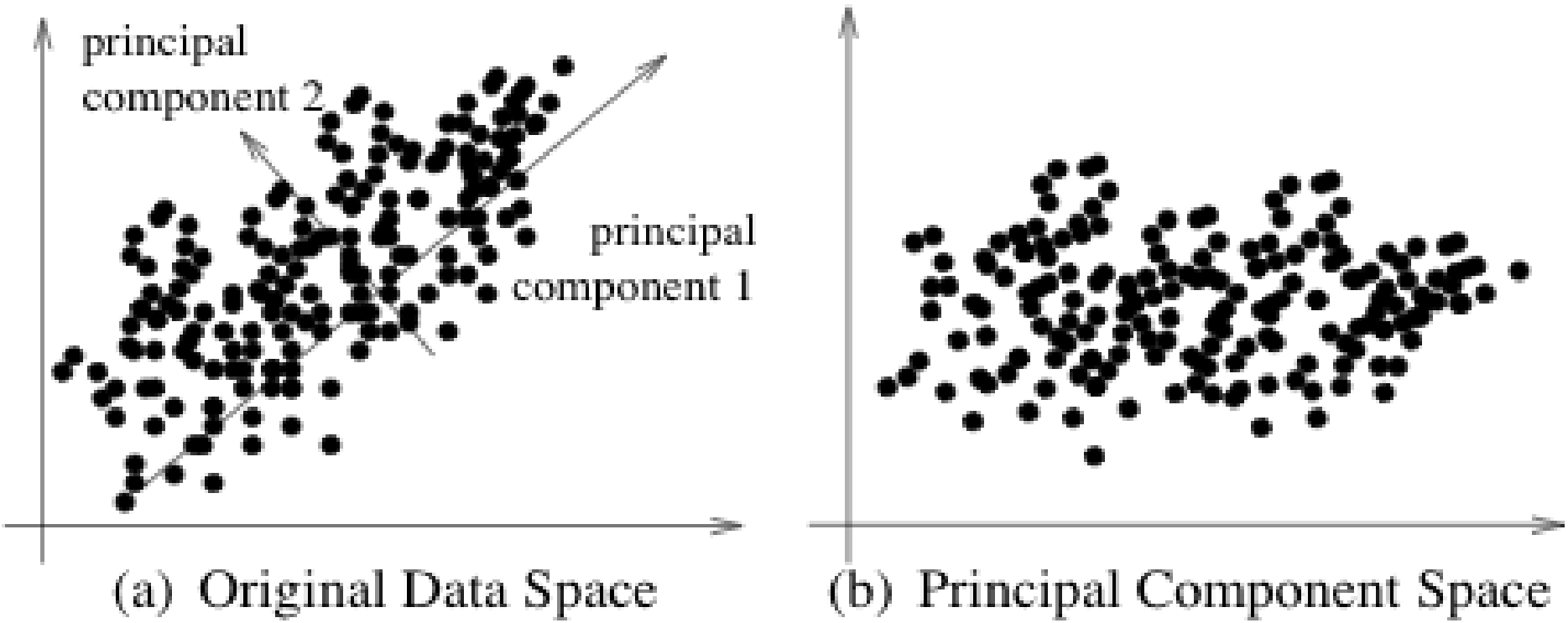
# GORDER KNN-Join

- Sorts the datasets in the **Grid-Order**
- Uses a scheduled block nested loop join
- Advantages
  - reduces random reads
  - uses pruning to reduce I/O and similarity computation
  - reduces the cost of the distance computation by doing it in lower dimensions

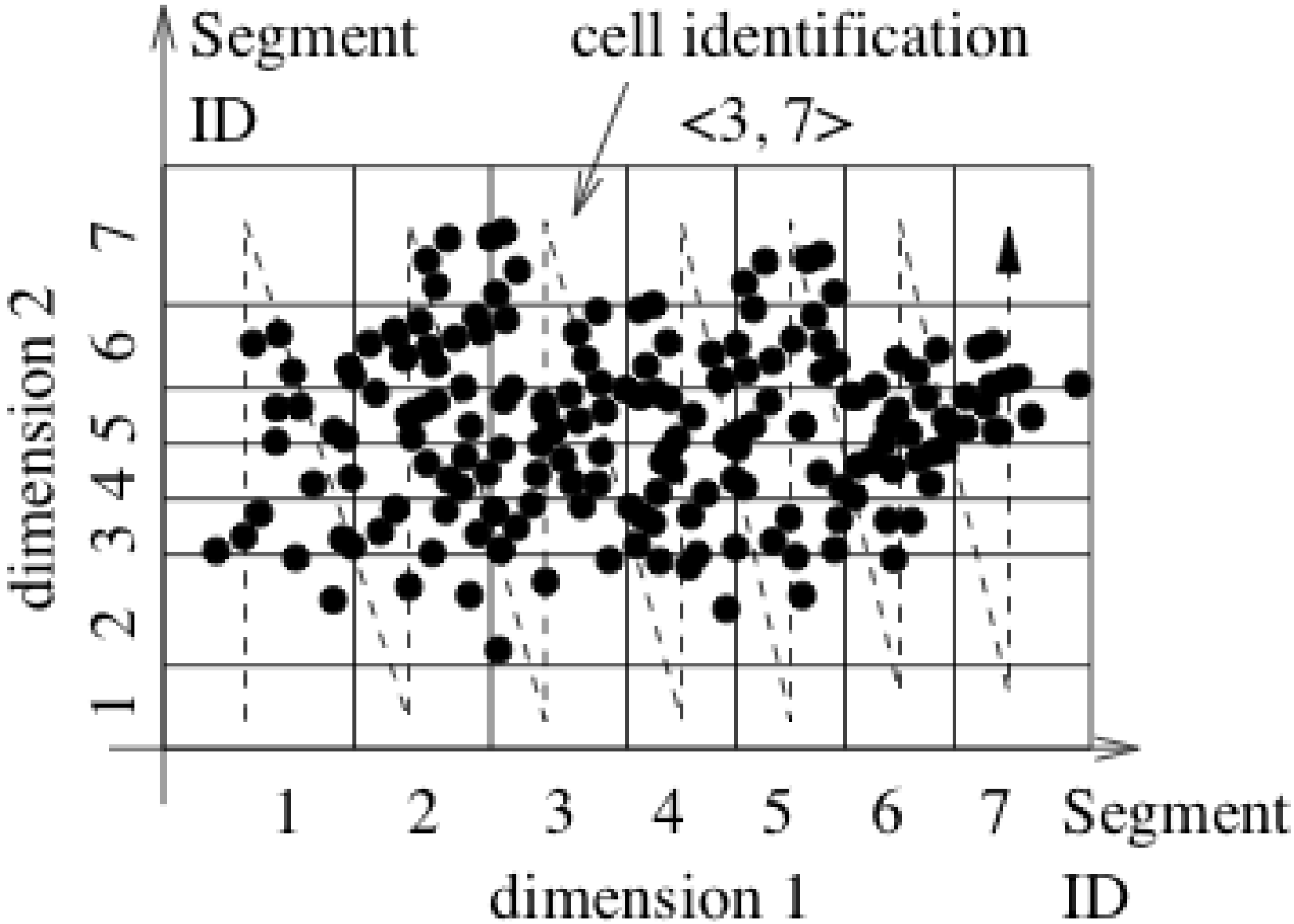
# G-Ordering

- Consists of two steps
  - Principal component analysis
  - Grid order sorting
- Grid Order
  - partition space into $l^d$ cells
  - identification vector $<s_1..s_n>$ for each cell
  - sort lexicographically by id vector

# G-ordering: Example



# G-ordering: Example



# G-ordered data: properties

- estimation of distance between p, q by projection to the first k dimensions
  - first dimensions are the most important ones
- for a **block** of points ($p_1 \ldots p_m$) bounding box can easily be calculated
  - check the first and the last point
  - check the first dimension in where they differ (**active dimension** of the block)

# Distance between blocks

- MinDist($B_r$, $B_s$)
  - minimum distance of the bounding boxes
- MaxDist($B_r$, $B_s$)
  - maximum distance of the bounding boxes
- bdist($B_r$, $B_s$): first consider MinDist(), if equal then consider MaxDist()
- Observation
  - MinDist() is a **lower bound** for the real distance of points in the block

# Pruning

- Idea: if a set of points are farer away than the kth candidate of p ($\in B_r$), **do not consider them** when joining with p
- Point pruning: MinDist($B_r$, $B_s$) > prunedst(p)
  - prunedst(p): distance to the k-th NN candidate
- Block pruning: MinDist($B_r$, $B_s$) > prunedst(Br)
  - prunedst(Br) = max (prunedst(p), p in $B_r$)

# Join GORDERed Data

- foreach Block $B_r$ of R:
  - foreach Block $B_s$ of S orderd by bdist:
    - if (MinDist($B_r$, $B_s$) > prunedst($B_r$)) break;
    - MemoryBlocking($B_r$, $B_s$)
  - Output KNN of $B_r$
- Observations:
  - bounding box of each S block in memory
    - easy sorting
  - loading of block data only if doing the join

# Joining: MemoryBlocking($B_r$, $B_s$)

- To reduce computational cost: subdivide into smaller blocks
- Subdivide $B_r$, $B_s$ into smaller blocks

- foreach Block $B_r'$ of $B_r$:
  - foreach Block $B_s'$ of S orderd by bdist:
    - if (MinDist($B_{r'}$, $B_{s'}$) >= prunedst($B_{r'}$)) break;
    - MemoryJoin($B_{r'}$, $B_{s'}$)

# Joining: MemoryJoin($B_r$, $B_s$)

- foreach point p in $B_r$
  - if (MinDist($B_r$, $B_s$) <= prunedst(p)):
    - for each point q in $B_s$
      - ComputeDist(p, q)

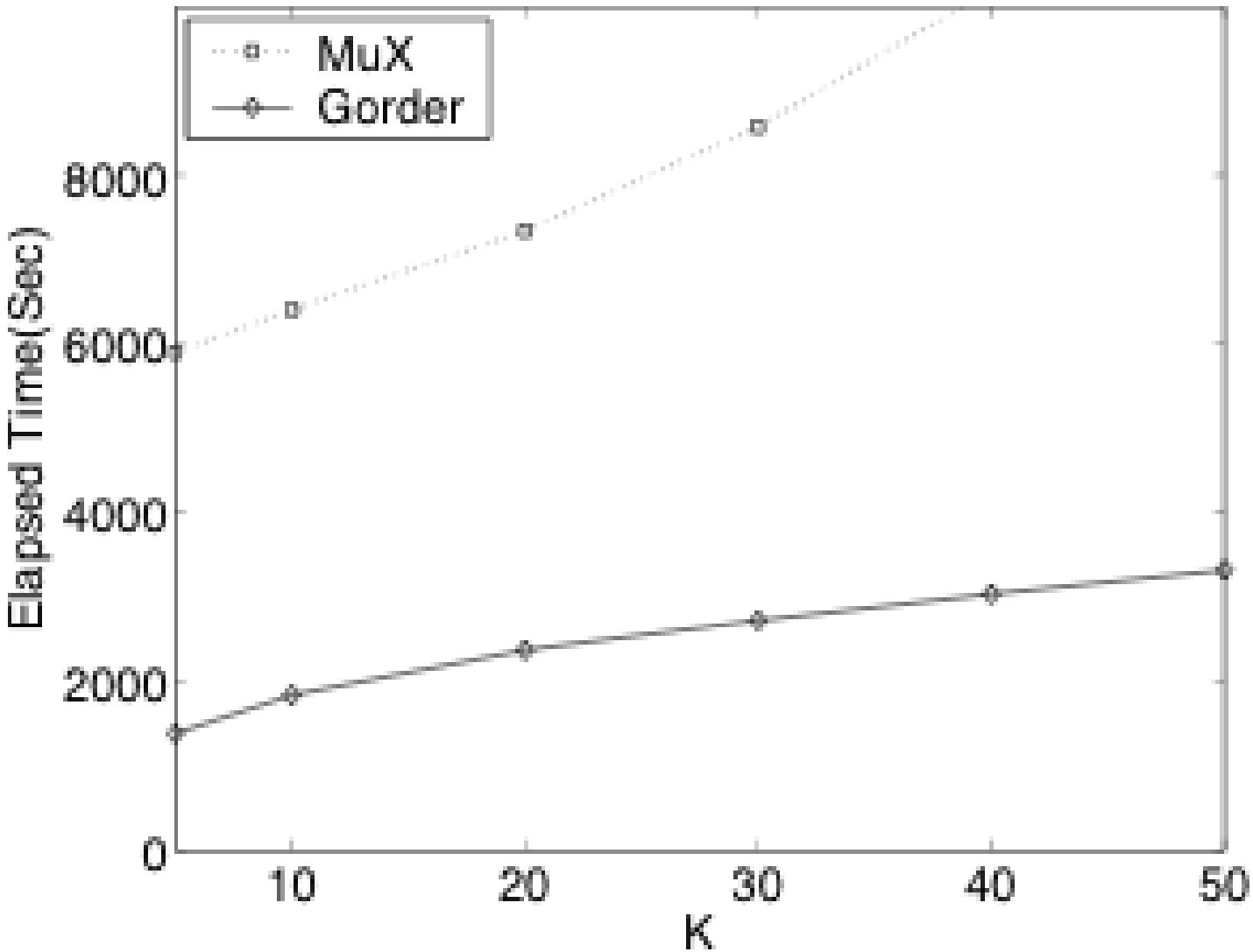
# Calculating the Distance

- $p \in B_r$, $q \in B_s$
- Consider the distance in a – 1 dimensions
- a = min(active dimension $B_r$, $B_s$)
- MinDist($B_{r,a-1}$; $B_{s,a-1}$) <= $d_{\{1..a-1\}}(p, q)$
- MinDist($B_{r,a-1}$; $B_{s,a-1}$) + $d_{\{a..d\}}(p, q)$ <= d(p,q)
- Use the above expression as a estimation for pruning

# Calculating the Distance

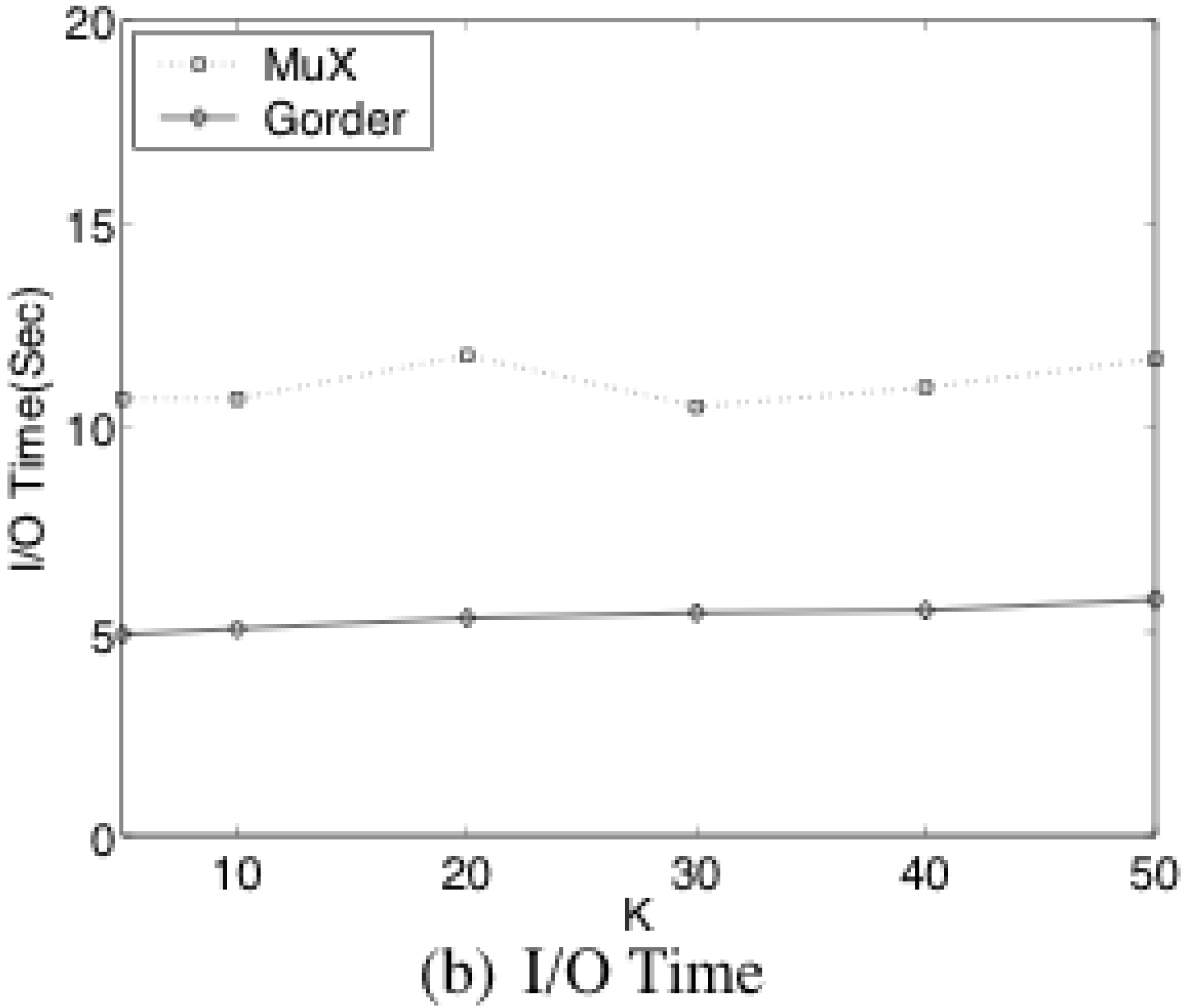
- pdist = MinDist($B_{r,a-1}$; $B_{s,a-1}$)
- for (k = a to d):
  - pdist += $(p.x_k - q.x_k)^2$
  - if (pdist > prunedst(p)) return;
- pdist -= MinDist($B_{r,a-1}$; $B_{s,a-1}$)
- for (k = 1 to a – 1)
  - pdist += $(p.x_k - q.x_k)^2$
  - if (pdist > prunedst(p)) return;
- Add q to the KNN of p

# Performance Evaluations (1)



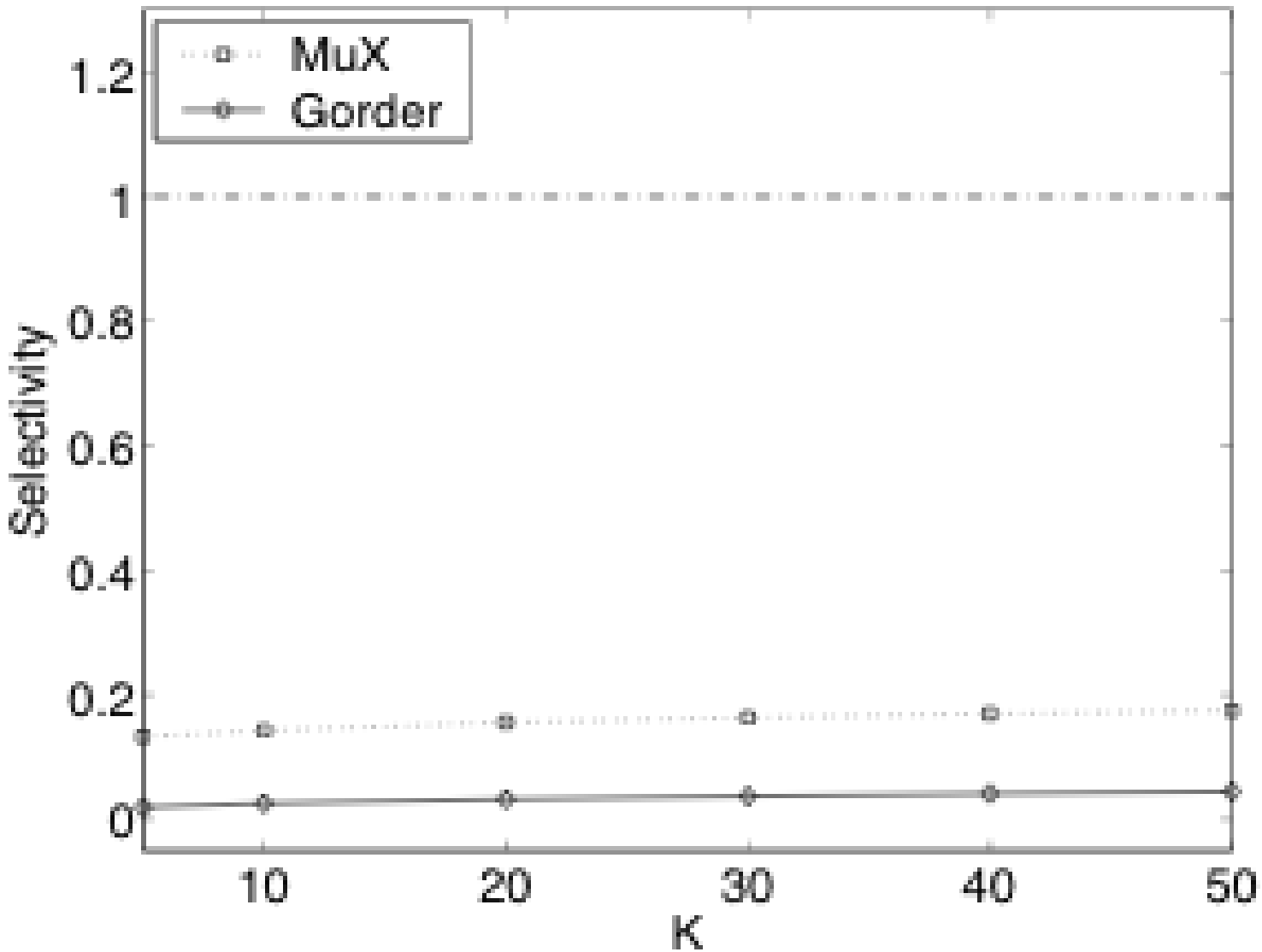

(a) Elapsed Time

# Performance Evaluations (1)



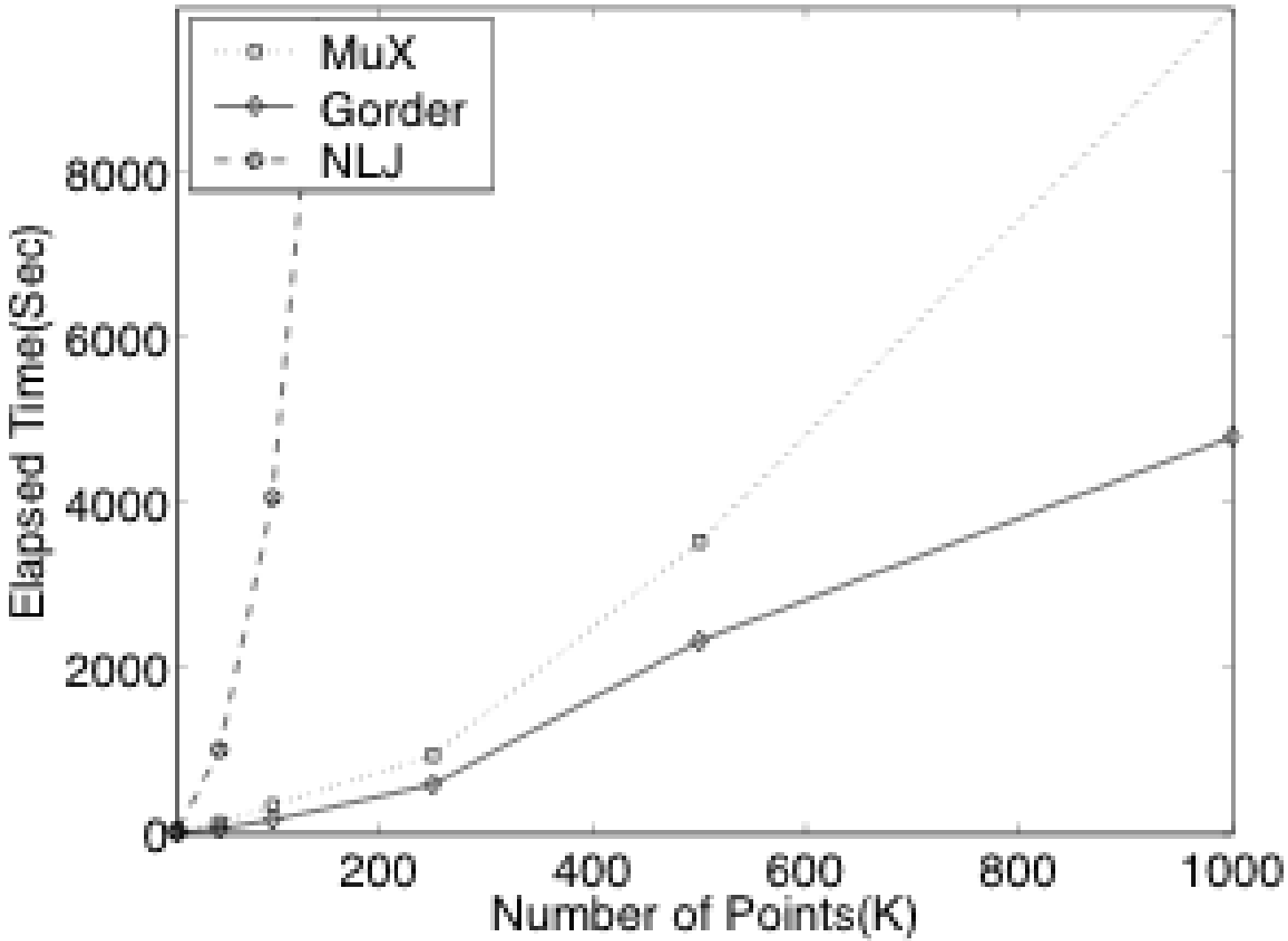

(b) I/O Time

# Performance Evaluations (1)



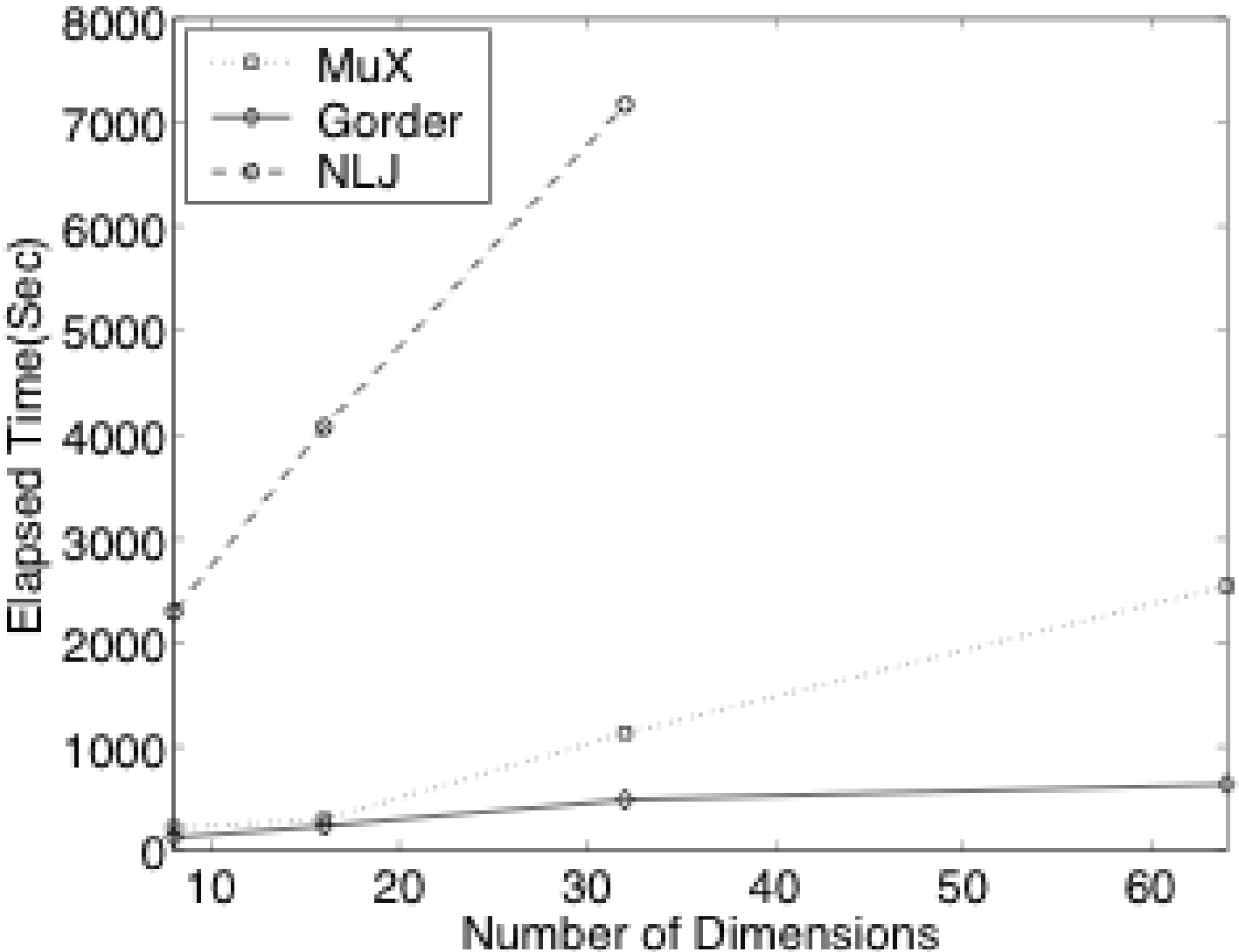

(c) Distance Computation Selectivity

# Performance Evaluations: Scaling

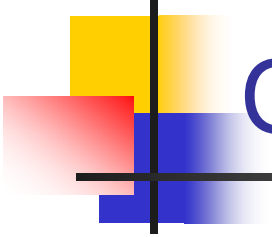



(a) Elapsed Time

# Performance Evaluations: Scaling




(a) Elapsed Time



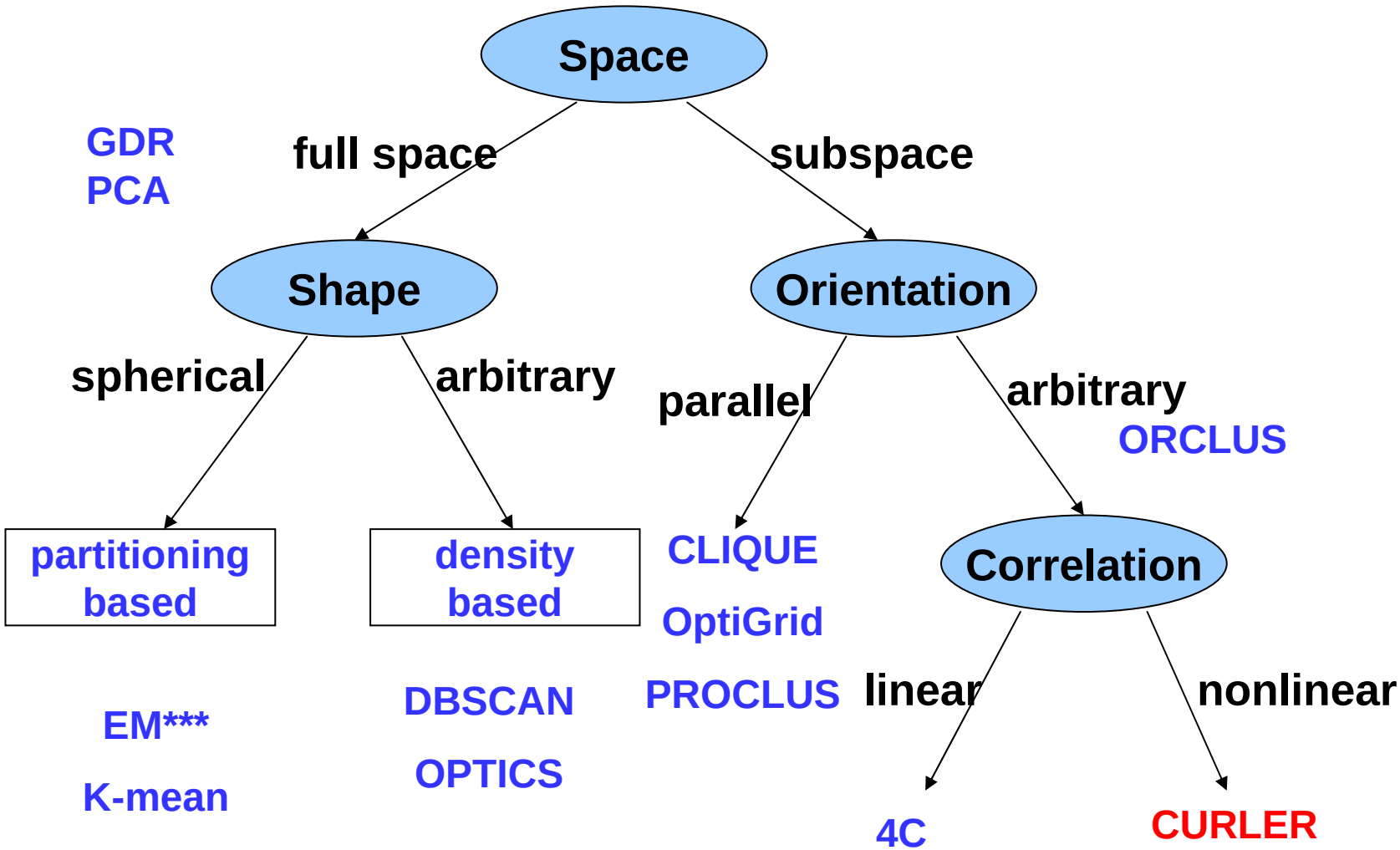
# Outline

- Sources of HDD
- Challenges of HDD
- Foundation
  - Similarity Function
  - High Dimensional Distance Join
- Techniques & Application
  - Finding Nonlinear Correlated Clusters in High Dimensional Data
  - Finding Patterns in Extremely High Dimensional Data

# Motivation

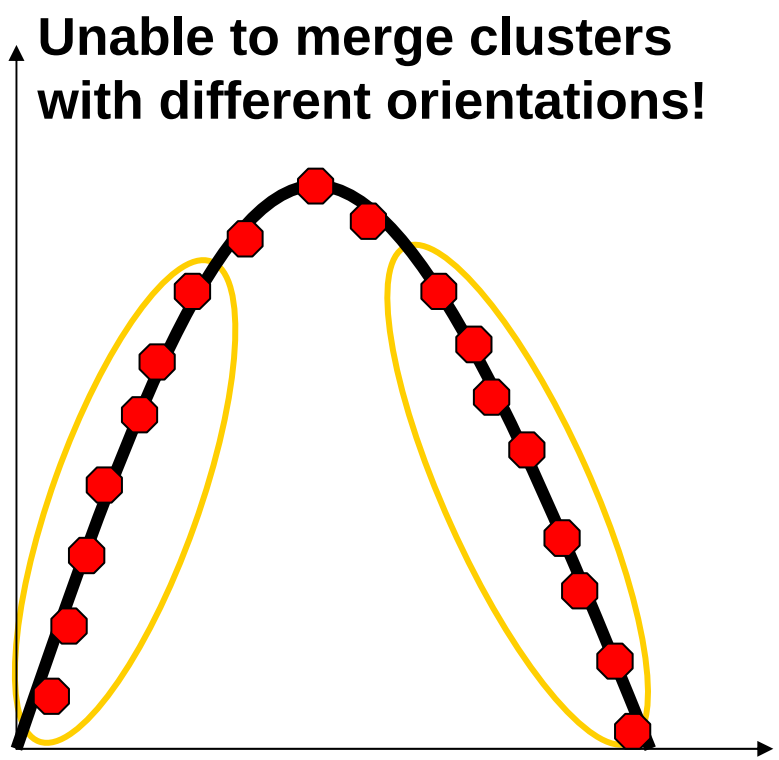
- Data objects are strongly correlated only in a subset of features instead of globally correlated in all the features.
- The correlation among the subset of features can be nonlinear as well as linear, e.g., the co-expression patterns of genes in a gene network, the Gas Laws in physic

# Existing clustering algorithms



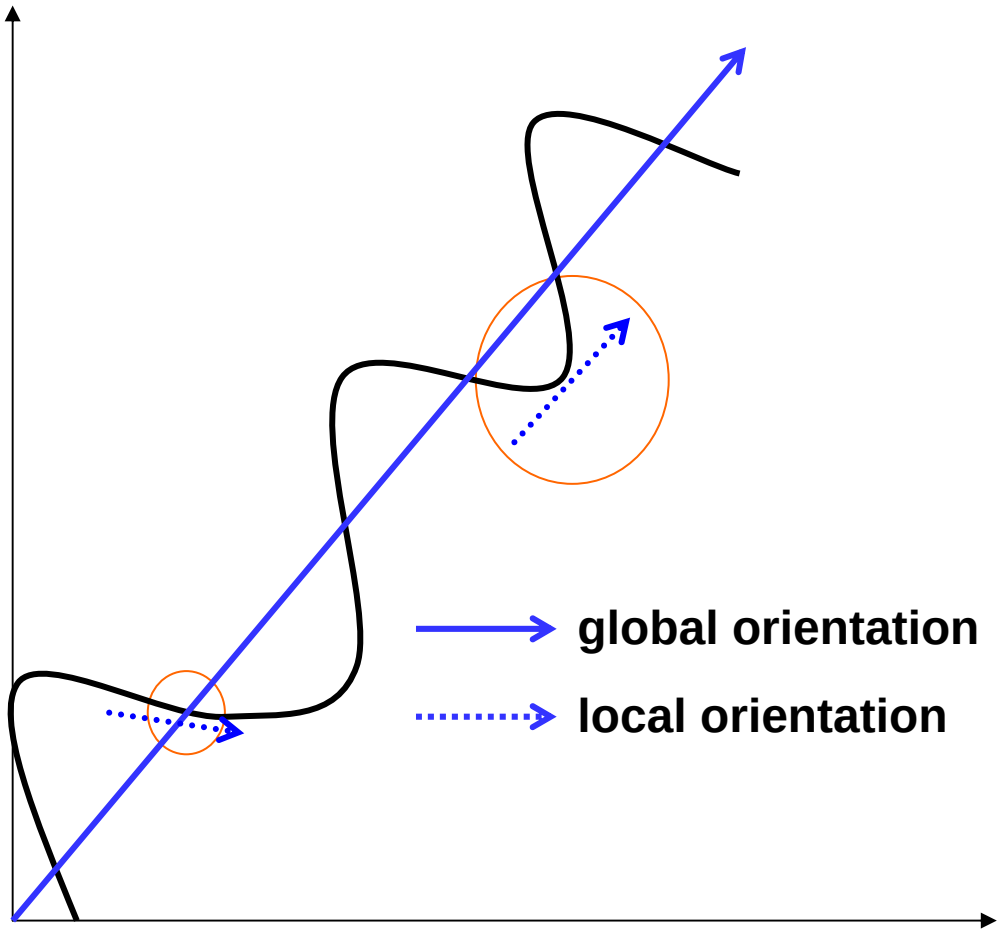
# ORCLUS and 4C

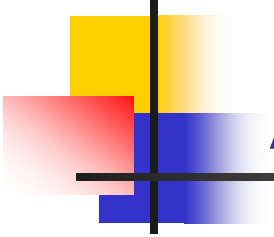
- Microclusters (seeds, core objects) are initialized to group nearby data objects together.
- Microclusters that are close in proximity and similar in orientation are merged together



# Challenges

- Determination of an appropriate neighborhood to capture the local orientation correctly
- Merge microclusters of different orientations to capture the nonlinear correlation





# Algorithm

- Fuzzy EM clustering
- Cluster expansion
- NNCO plot: Visualization
- Top-down clustering
- Time complexity analysis

# EM Clustering

- Clustering model
  - Mean vector
  - Covarience Matrix
- Iterative adjustment of model
  - E Step: update the member probability of each data for each microcluster
  - M Step:  update the clustering model for each microcluster
  - Stopping criteria: log likelihood of mixture model is converged or MaxLoopNum is achieved.

# Desirable Characteristic of EM

- Fuzzy Clustering
  - In real life dataset, each point can belong to different subspace, E.g., a patient may suffer from two types of disease A and B
  - Points become “stretchable” like bond in metals. Can define microclusters similarity based on number of points shared
- Iterative
  - Overcoming the catch-22 situation. Neighbors in full space might not be neighbors in subspace. But how do we know neighbors in subspace?

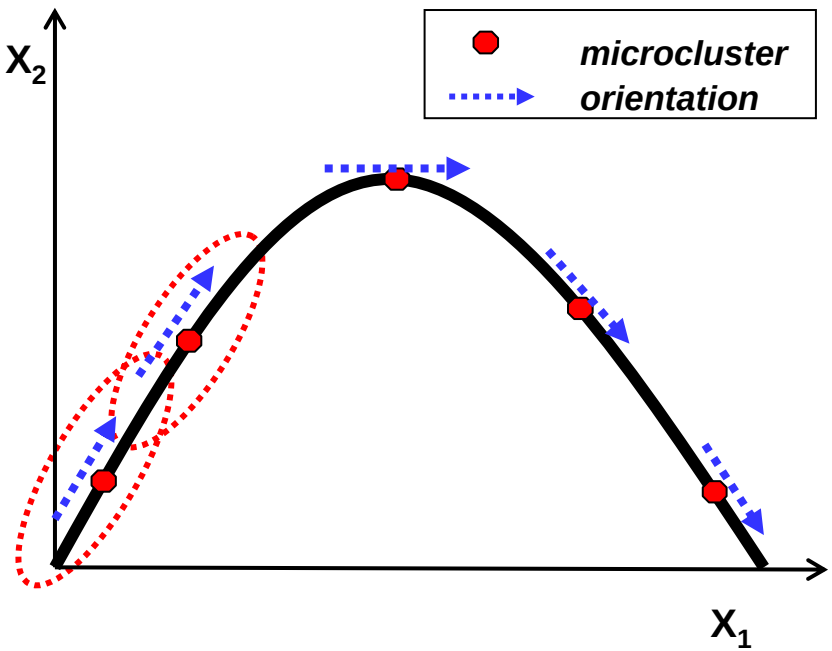
# Co-sharing Level

- Co-sharing level:

$$\cosh are(M_i, M_j) = \sum_{x \in D} [PR(M_i \mid x) * PR(M_j \mid x)]$$

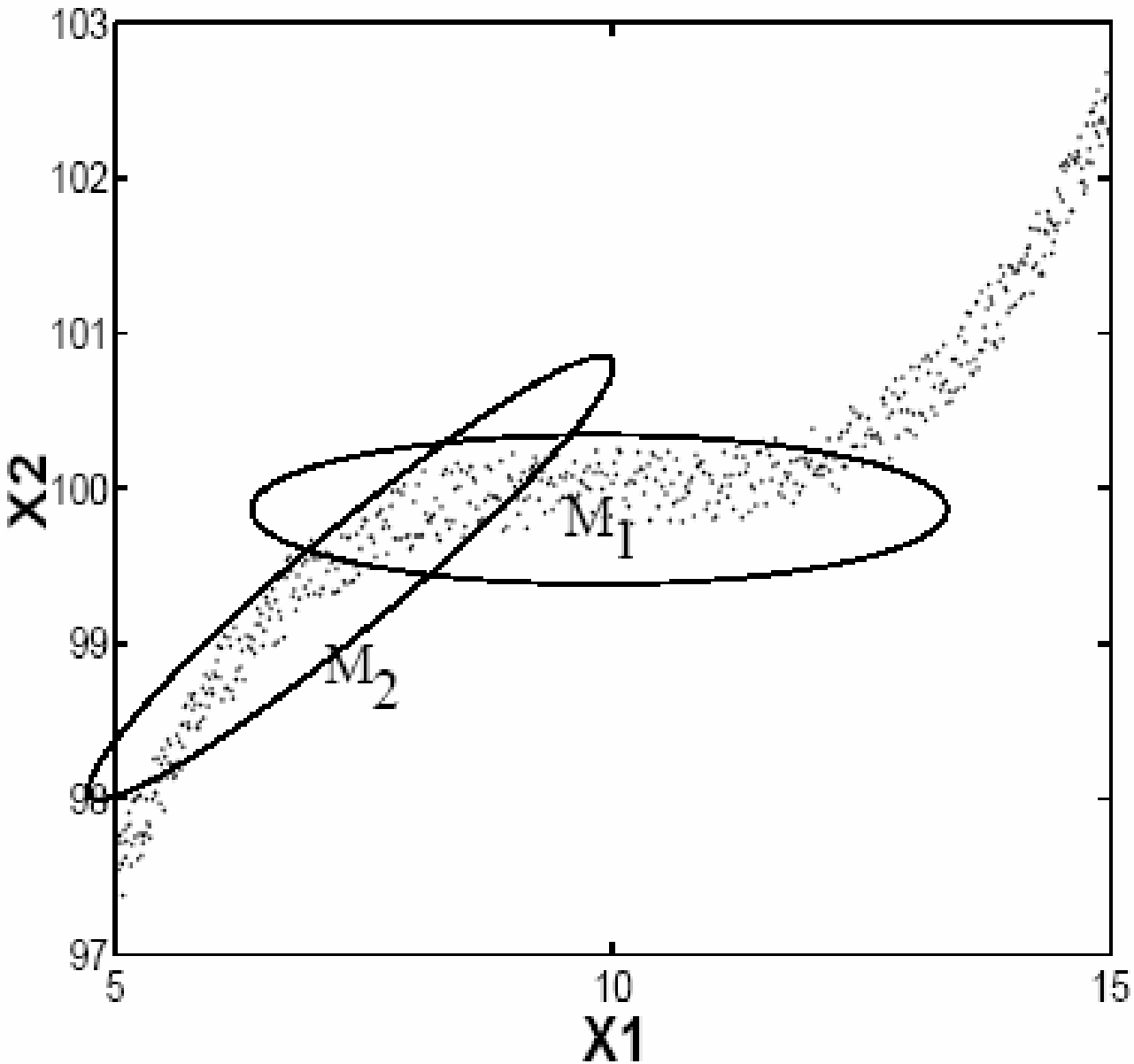
- Co-sharing level matrix: M, each entry

$$M_{ij} = \cosh are(M_i, M_j)$$




| coshare | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
|---|---|---|---|---|---|
| $M_1$ | - | 0.9 | 0.4 | 0.3 | 0.2 |
| $M_2$ | | - | 0.5 | 0.2 | 0.3 |
| $M_3$ | | | - | 0.5 | 0.4 |
| $M_4$ | | | | - | 0.9 |
| $M_5$ | | | | | - |

# Co-sharing Example



# Cluster Expansion I

- Iterative merge $M_c$ with the highest co-sharing value to current cluster C

$$\cosh are(C, M_c) = Max\{\cosh are(C, M_i)\}$$

- Matrix updating in cluster expansion

$$\cosh are(C, M_k) = Max(\cosh are(C, M_k), \cosh are(M_c, M_k))$$

- Optimization: only keep the top $l_{top}$ $PR(M_i|x)$ for each data x

# Cluster Expansion Example

| coshare | C | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
|---|---|---|---|---|---|
| C | - | **0.9** | 0.4 | 0.3 | 0.2 |
| $M_2$ | | - | 0.5 | 0.2 | 0.3 |
| $M_3$ | | | - | 0.5 | 0.4 |
| $M_4$ | | | | - | 0.9 |
| $M_5$ | | | | | - |

**C={$M_1$}**

| coshare | C | $M_3$ | $M_4$ | $M_5$ |
|---|---|---|---|---|
| C | - | **0.5** | 0.3 | 0.3 |
| $M_3$ | | - | 0.5 | 0.4 |
| $M_4$ | | | - | 0.9 |
| $M_5$ | | | | - |

**C={$M_1$, $M_2$}**

| coshare | C | $M_5$ |
|---|---|---|
| C | - | **0.9** |
| $M_5$ | | - |

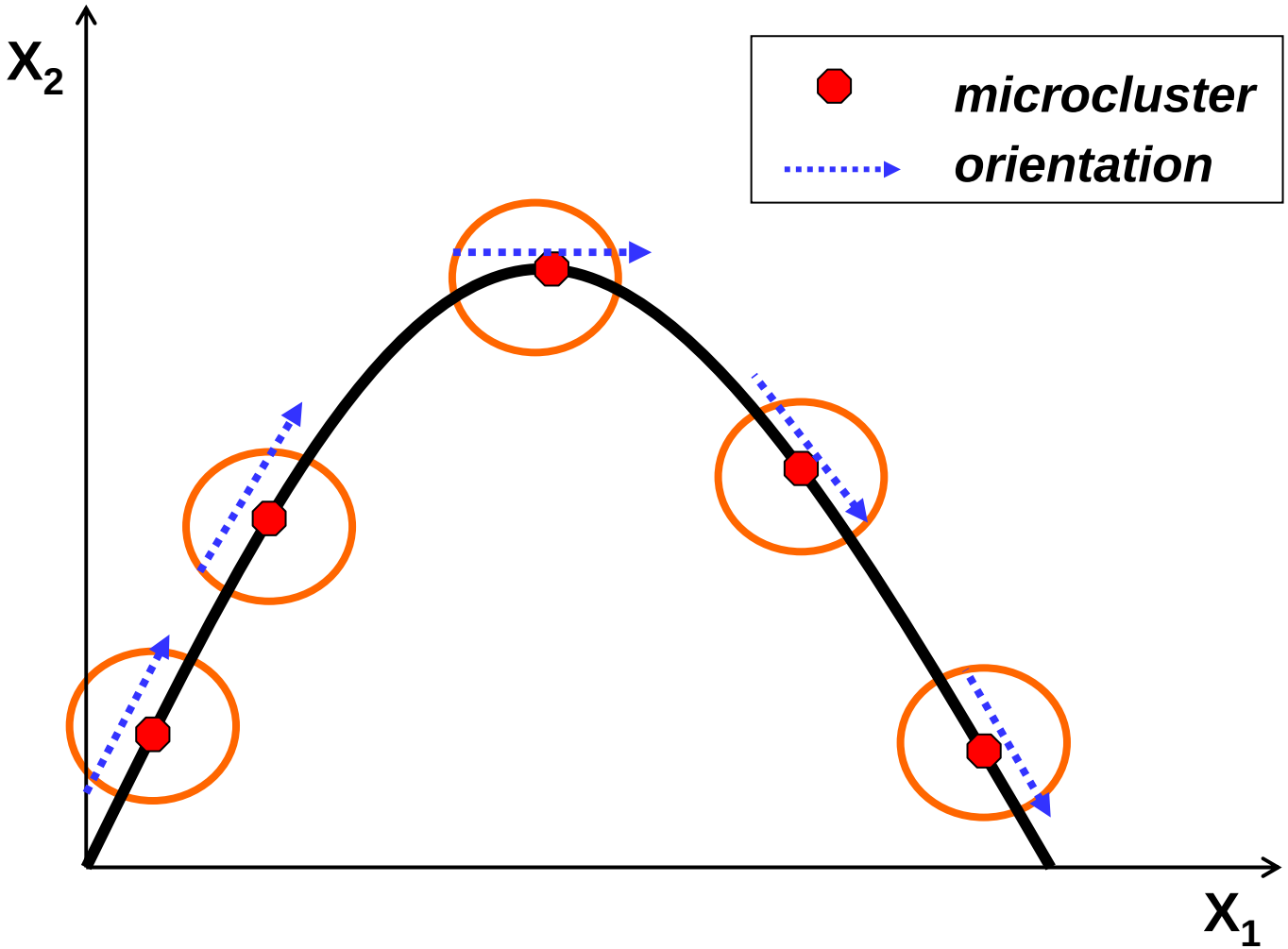
**C={$M_1$, $M_2$, $M_3$,$M_4$,$M_5$}**

**C={$M_1$, $M_2$, $M_3$,$M_4$}**

| coshare | C | $M_4$ | $M_5$ |
|---|---|---|---|
| C | - | **0.5** | 0.4 |
| $M_4$ | | - | 0.9 |
| $M_5$ | | | - |

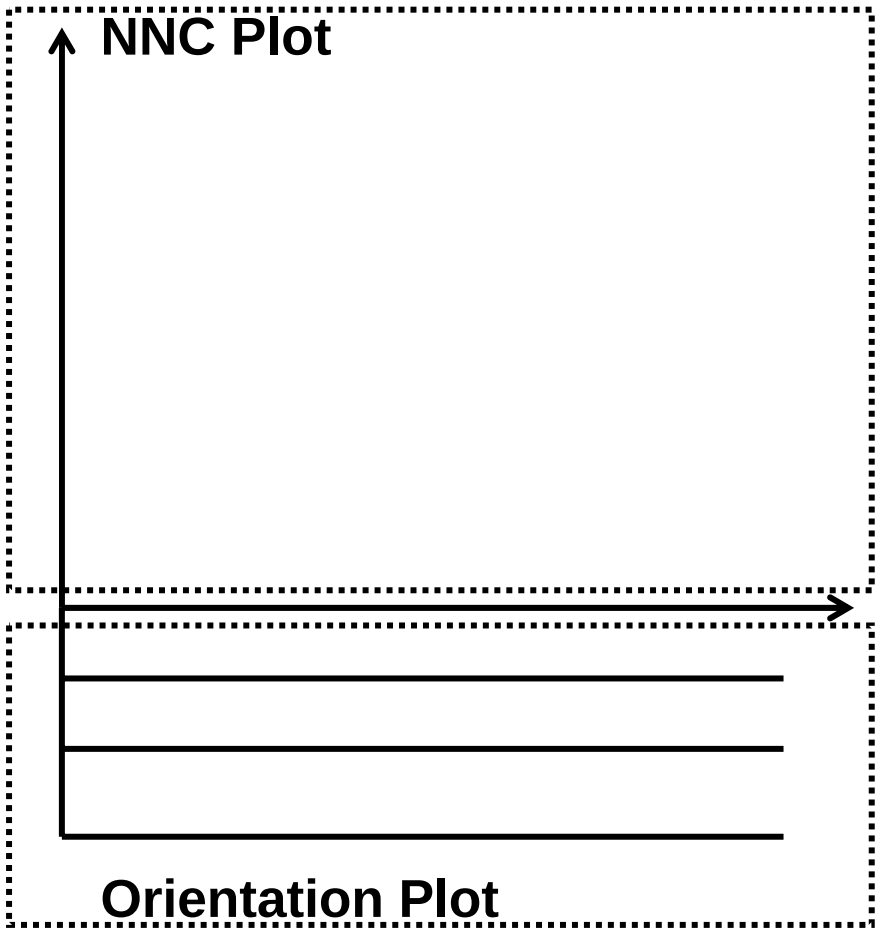
**C={$M_1$, $M_2$, $M_3$}**

# Cluster Expansion Example Cont.



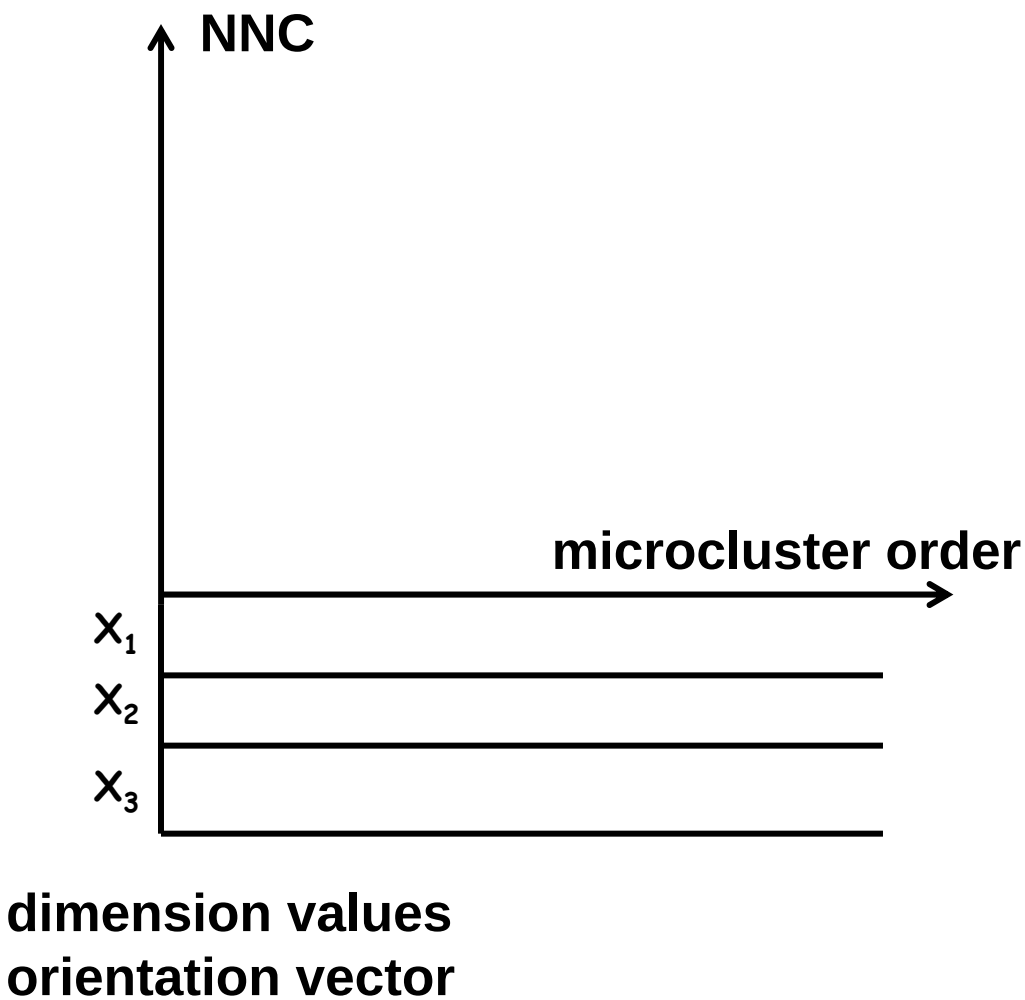
# NNCO Plot

- Nearest neighbor co-sharing level plot (NNC plot)
- Orientation plot



# Axes of NNCO Plot

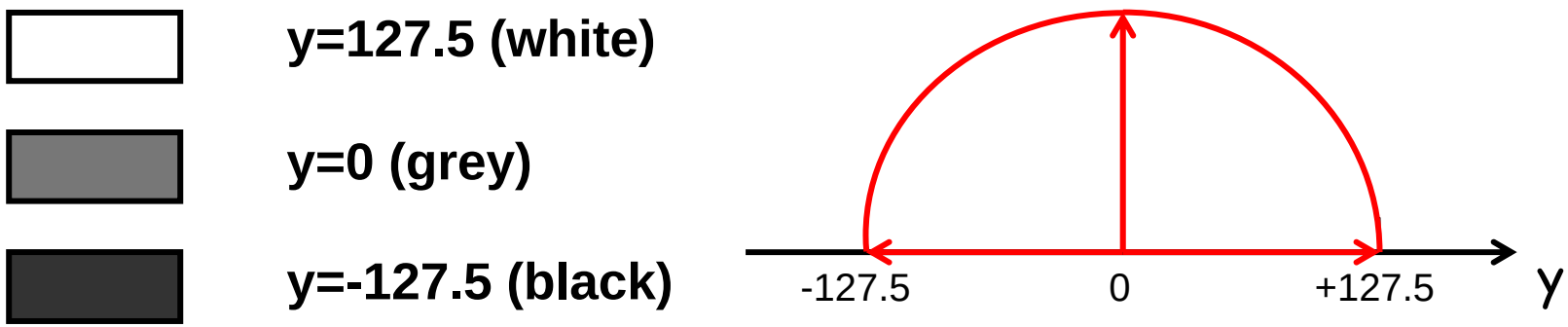
- Horizontal axis: the microcluster order in cluster expansion.
- Vertical axis above: the co-sharing level between the microcluster and the cluster being processed to which it is added.
- Vertical axis below: dimension values of the orientation vector for each microcluster



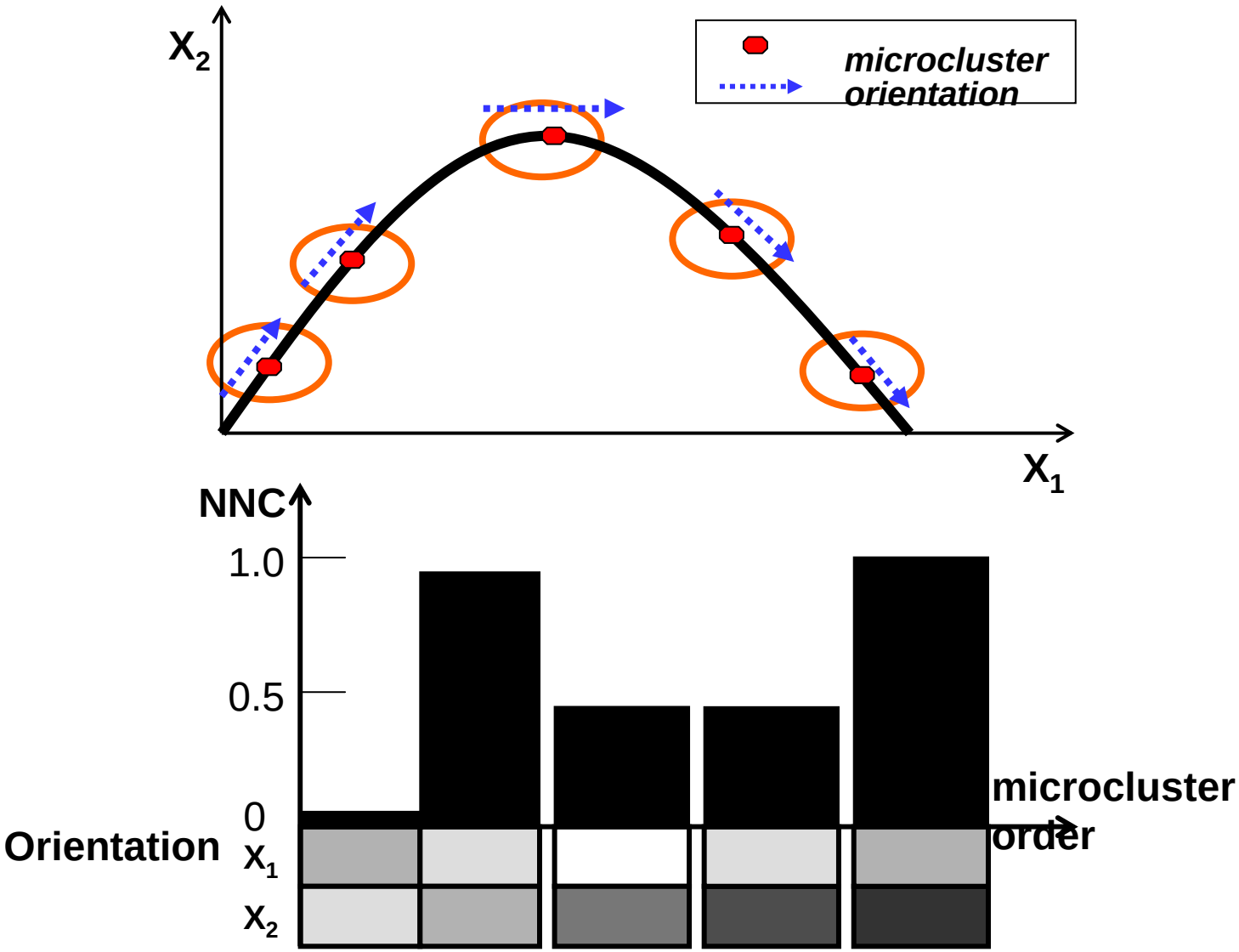
# Color Mapping

- Normalization: each dimension value y of the microcluster orientation vector is normalized to the range of [-127.5, +127.5].
- Color Mapping:

  *Color(y)=[R(y+127.5), G(y+127.5), B(y+127.5)]*

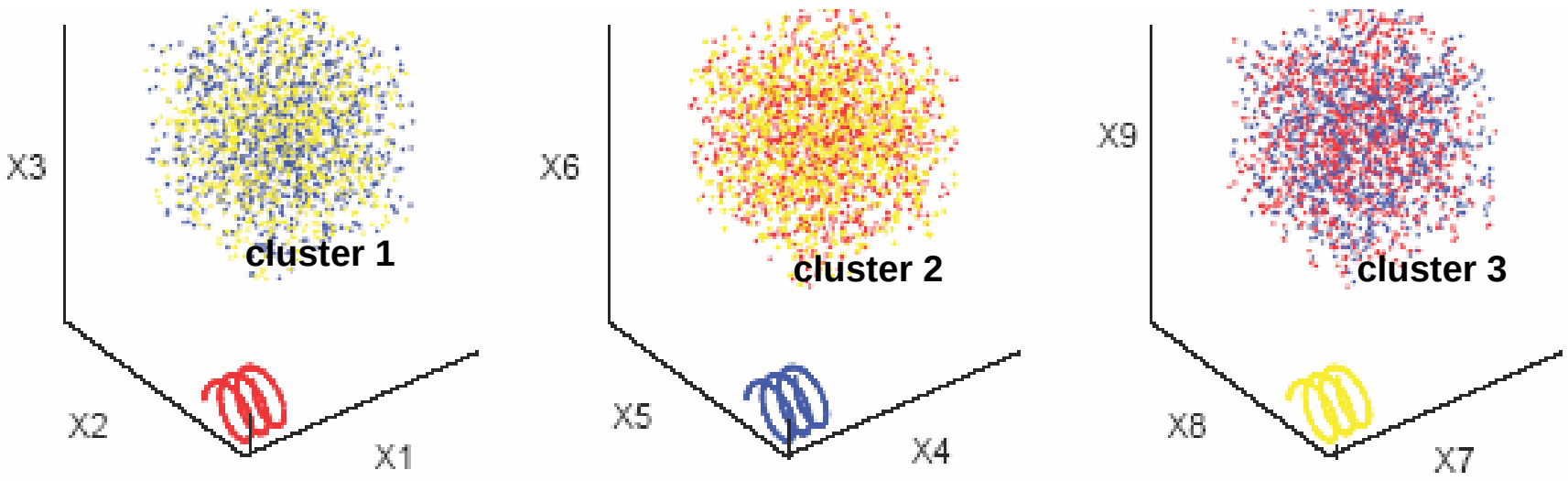


# NNCO Plot Example



# Top-down Clustering

- Motivation: the first round clustering on the global data space might only capture the global orientation of each cluster.



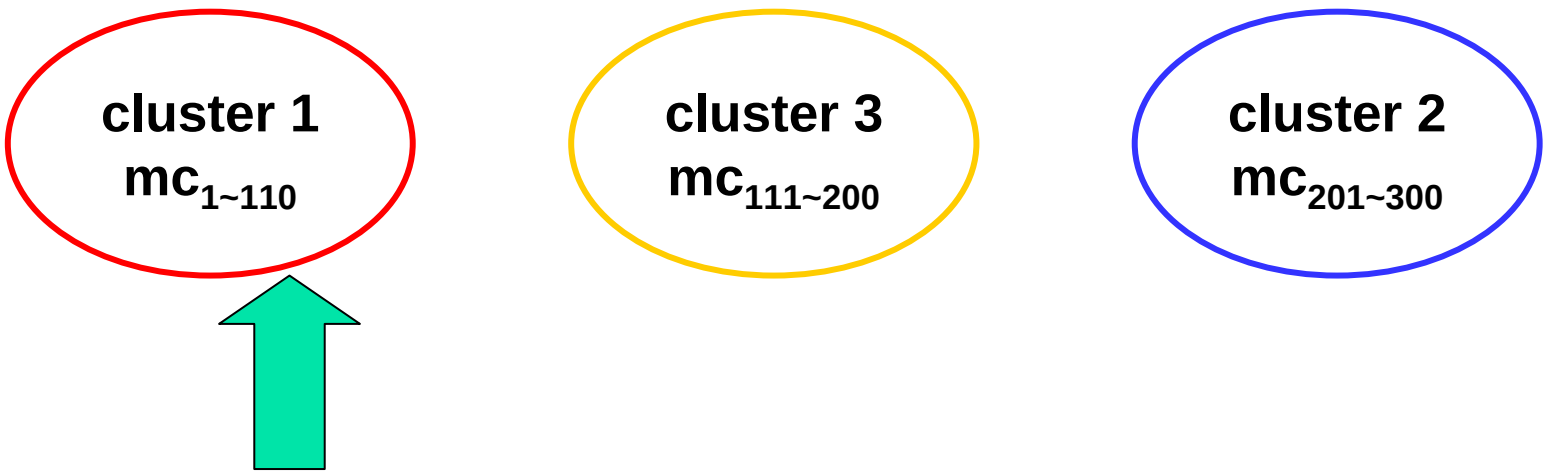

**A synthetic 9D dataset with 3 clusters in different existing space**

# Top-down Clustering I

- Identify clusters in global space $\{X_1, X_2, \ldots, X_d\}$ which are separated by NNC gaps.

# Top-down Clustering II

- Identify the data members for each cluster




**Max $\{PR(M_i|x)\}=PR(mc_{50}|x)$, i=1, 2, …, 300.**

# Top-down Clustering III

- Project the data members of each cluster $C_i$ into the corresponding cluster existing space $\{e_{i1}, e_{i2}, \ldots, e_{il}\}$ ($l <= d$)

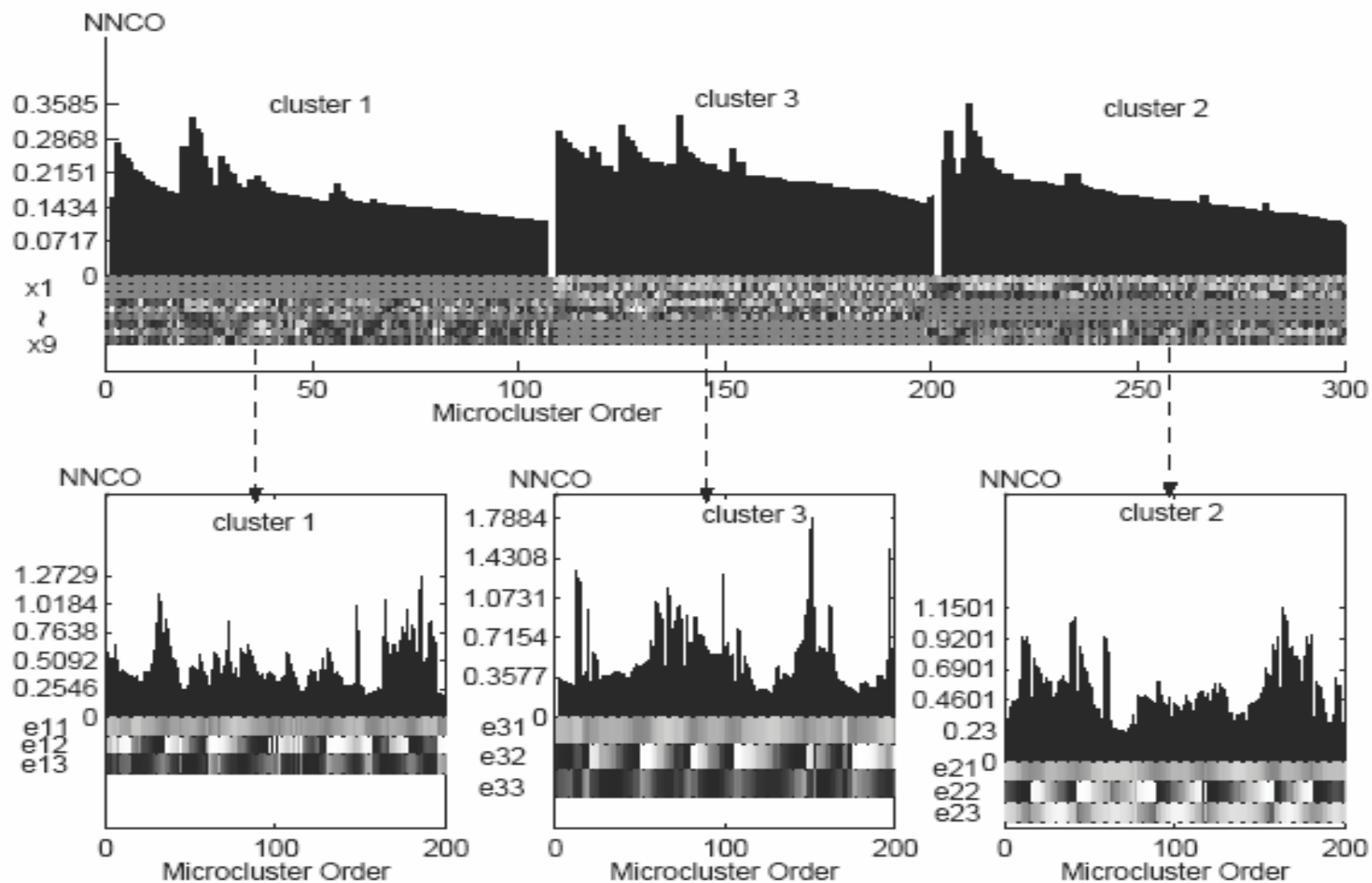
$$x' = (x \cdot e_{i1}, x \cdot e_{i2}, \ldots\ldots, x \cdot e_{il})$$

x: d-dimensional vector.

$e_{ij}$: l-dimensional vector, eigenvector with the minimum eigenvalue decomposed from the covariance matrix of cluster $C_i$'s datamembers.

# Top-down Clustering IV

- Run CURLER again to capture the local cluster structure of the interested cluster in the new space




Data Mining: Foundation, Techniques and Applications

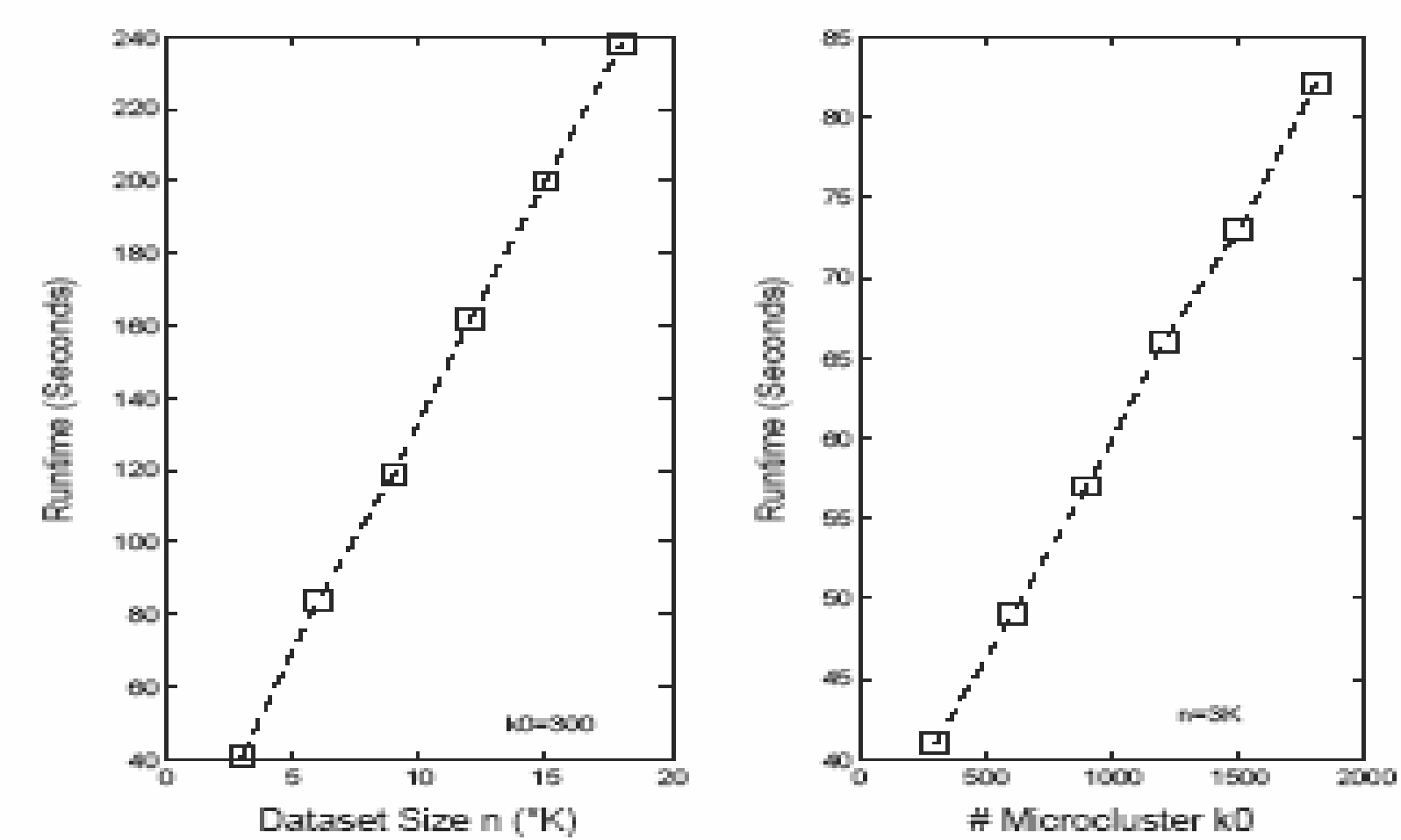
# Time Complexity Analysis

n: number of data objects

$k_0$: number of microclusters

- EM Clustering
  - matrix operation: $O(d^3)$
  - matrix operation for $k_0$ mcs: $O(k_0 \cdot d^3)$
  - computation of $PR(x|M_i)$: $O(d^2)$
  - total: $O(k_0 \cdot n \cdot d^2 + k_0 \cdot d^3)$
- Cluster Expansion
  - cosharing level matrix initialization: $O(n \cdot l_{top}^2)$
  - matrix updating: $O(k_0)$
  - maximal number of updating: $k_0$
  - total: $O(n \cdot l_{top}^2 + k_0^2)$

# Evaluation of Efficiency

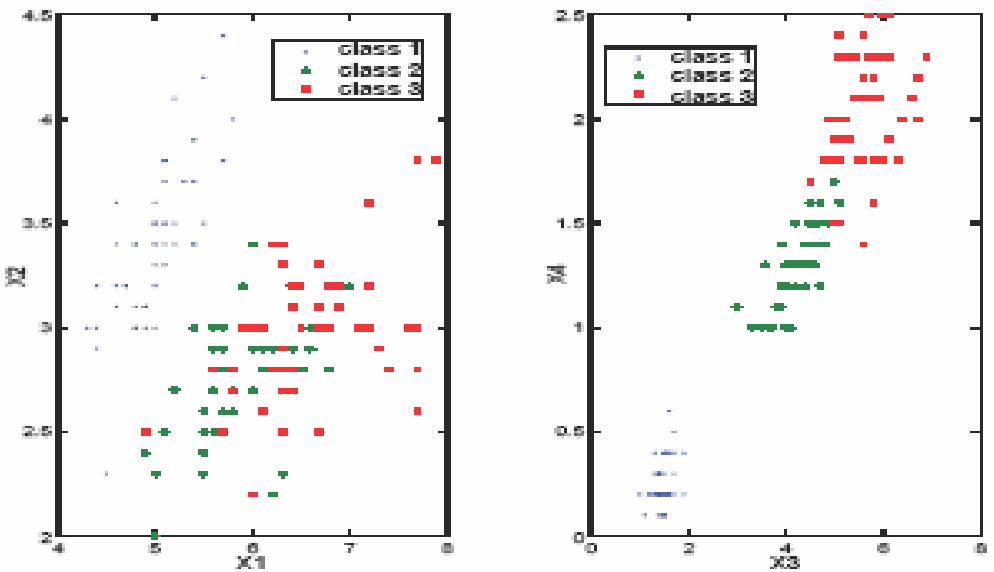



**Runtime of CURLER when varying dataset size and microcluster number on the 9D synthetic dataset**

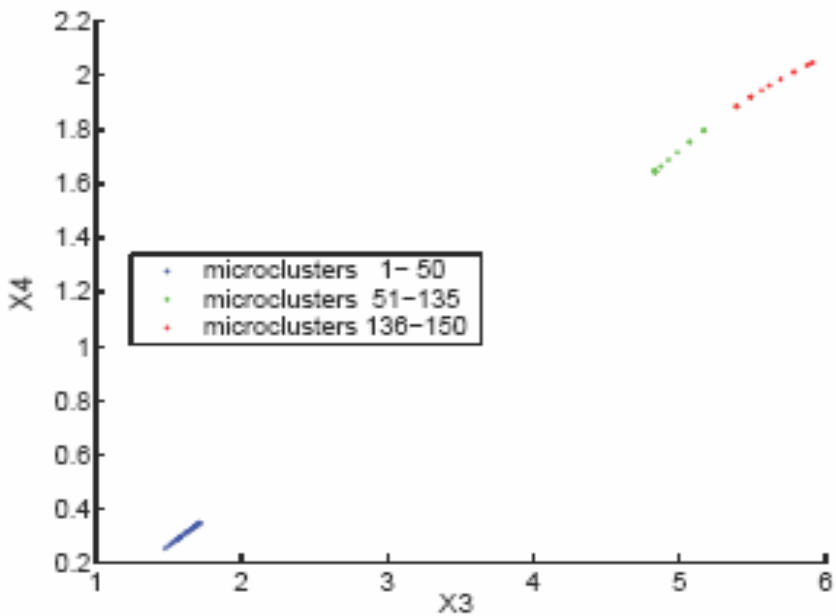
# Evaluation of Effectiveness

- Synthetic 9D dataset
- Iris and Image datasets from UCI repository of machine learning databases and domain theories
- Iyer time series gene expression data

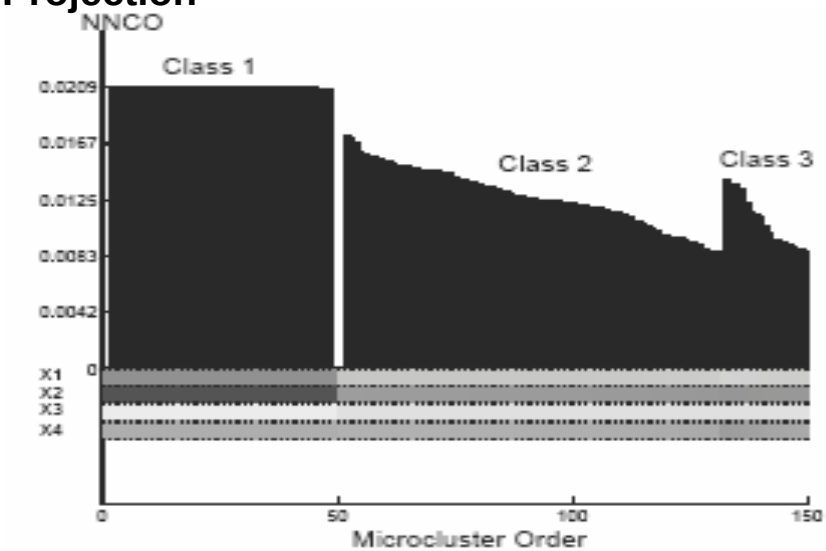
# 4D Iris Dataset of 3 Classes




**Data Projection**

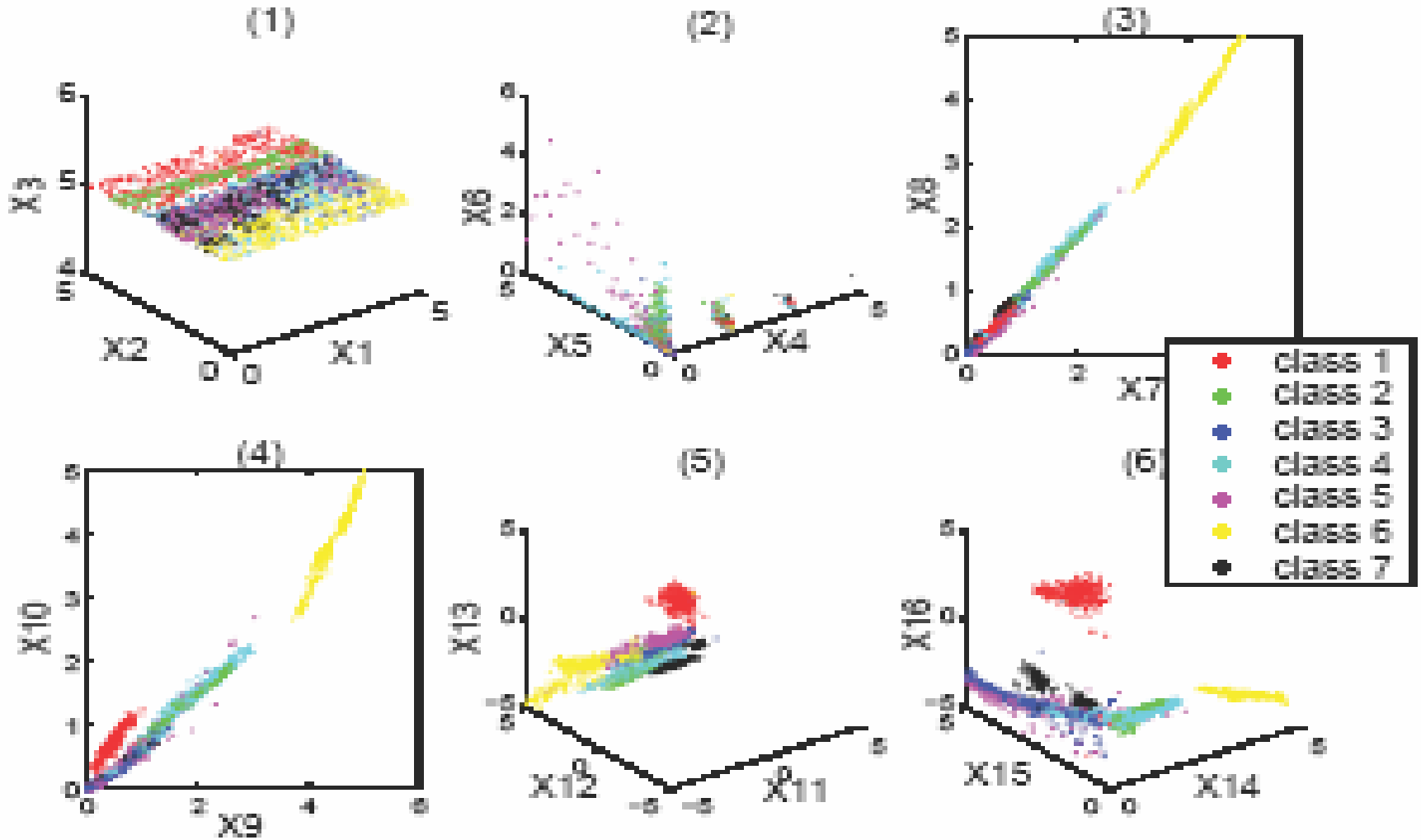



**Microcluster Traversal**

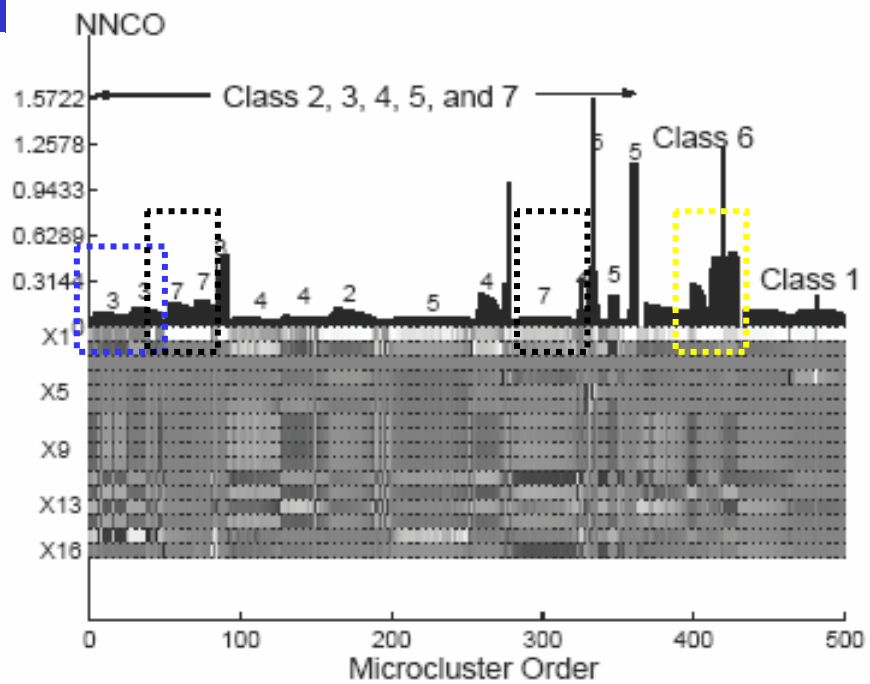



**NNCO Plot**

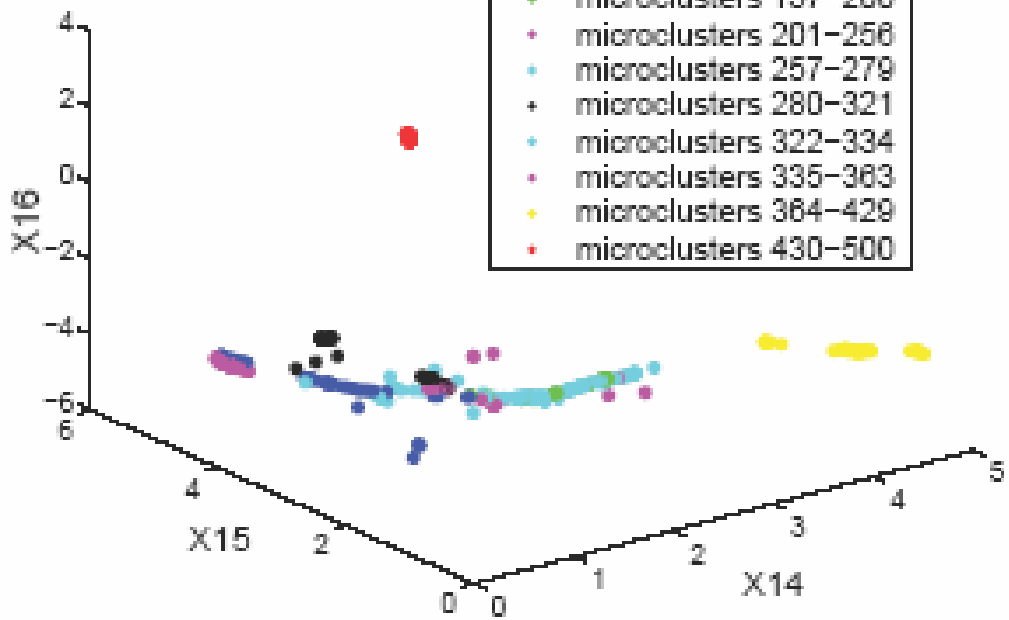
# 16D Image Dataset of 7 Classes



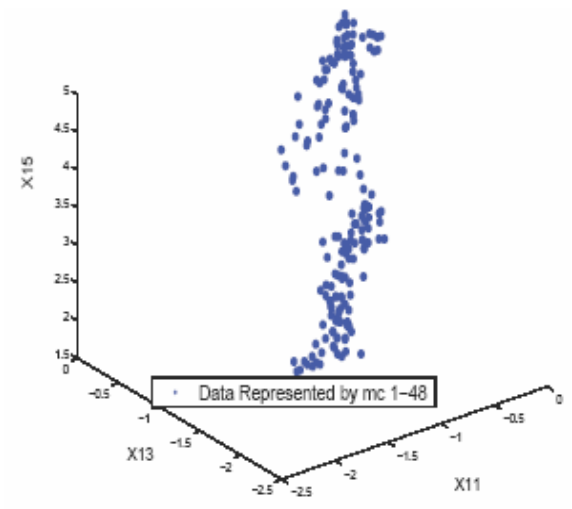

**Data Projection**

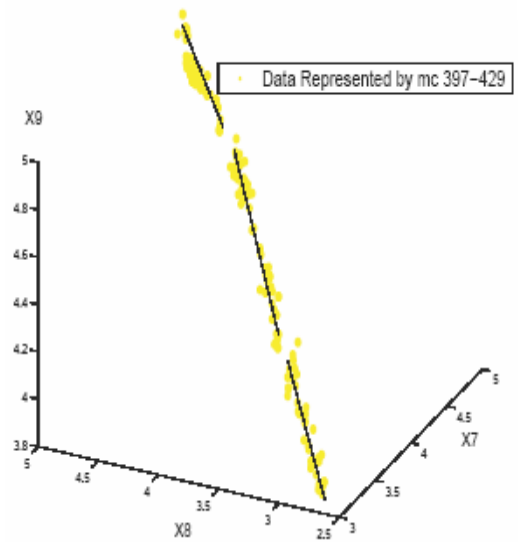


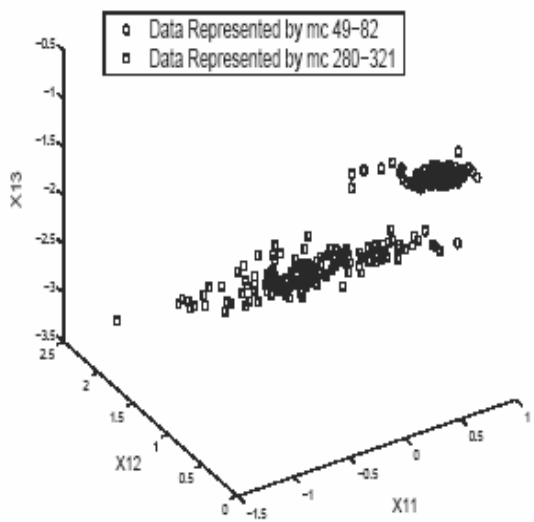

**NNCO Plot**



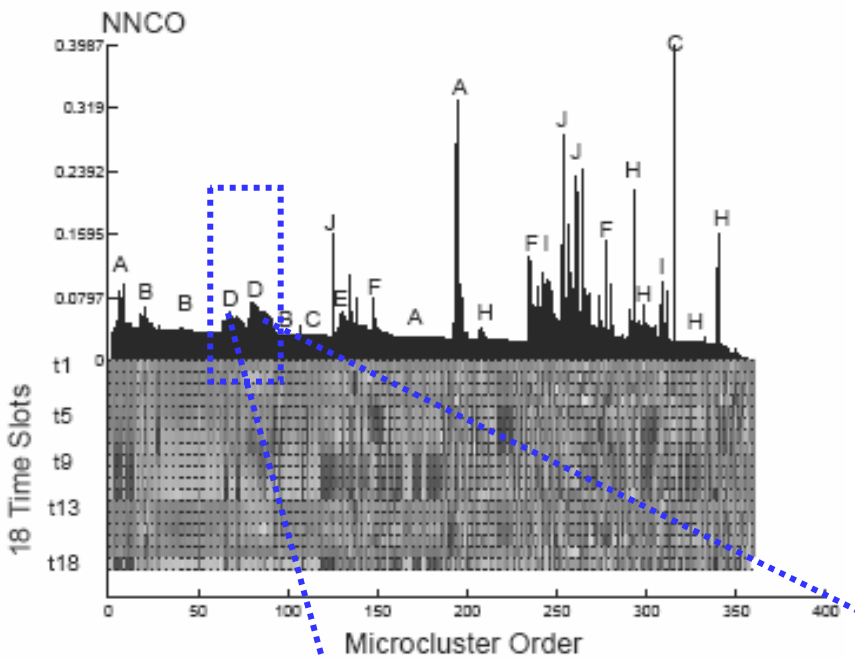

**Microcluster Traversal**

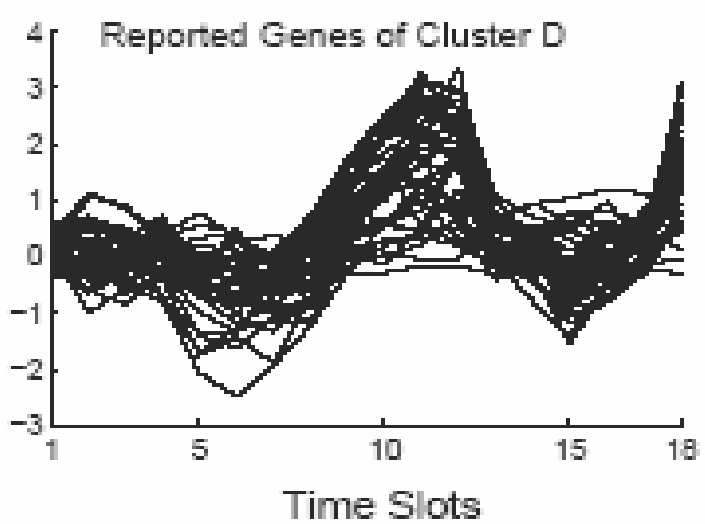


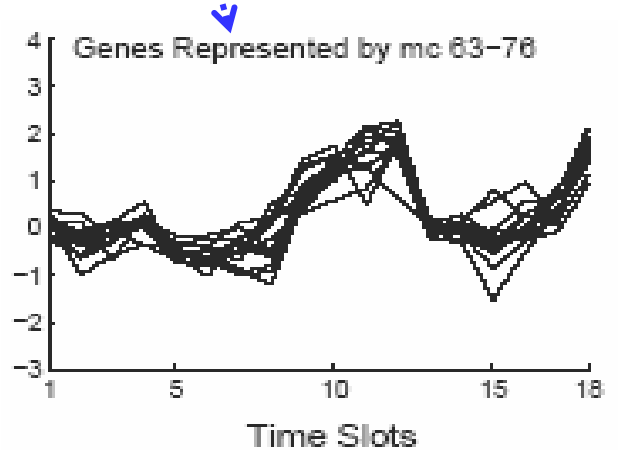



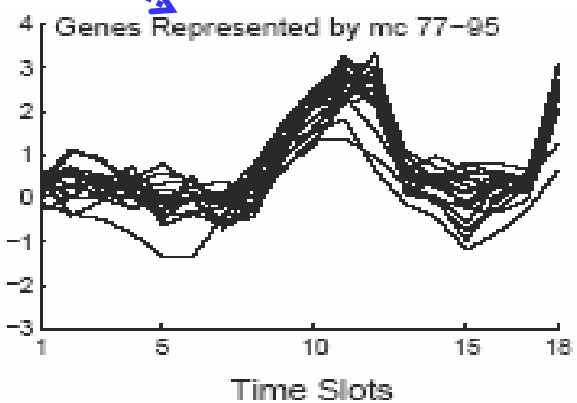



# Iyer Dataset



# Summary

- CURLER is pretty simple after explaining. Main contributions are the insights:
  - Identify the concepts of global and local orientation
  - Realize the characteristic need for various components of CURLER
  - The engineering work needed to put things together
- Future Work
  - When can we stop the sublevel clustering ? Can we do it automatically using a modification of residue analysis ?
  - Can we make use of CURLER for visual classification ?
  - A look at how the catch-22 situation can be avoid by looking at better similarity function

# Outline

- Sources of HDD
- Challenges of HDD
- Foundation
  - Similarity Function
  - High Dimensional Distance Join
- Techniques & Application
  - Finding Nonlinear Correlated Clusters in High Dimensional Data
  - Finding Patterns in Extremely High Dimensional Data

# A Microarray Dataset

1000 - 100,000 columns

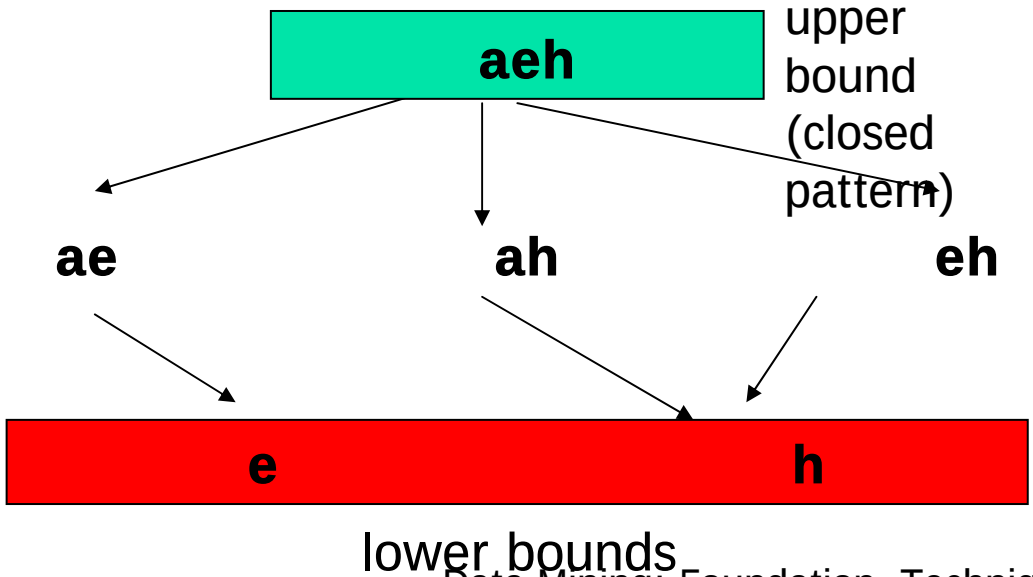
100-500 rows

| | Class | Gene1 | Gene2 | Gene3 | Gene4 | Gene 5 | Gene 6 | Ge |
|---|---|---|---|---|---|---|---|---|
| Sample1 | Cancer | | | | | | | |
| Sample2 | Cancer | | | | | | | |
| . . . | | | | | | | | |
| SampleN-1 | ~Cancer | | | | | | | |
| SampleN | ~Cancer | | | | | | | |

- Find closed patterns which occur frequently among genes.
- Find rules which associate certain combination of the columns that affect the class of the rows
  - Gene1,Gene10,Gene1001 -> Cancer

# Challenge I

- Large number of patterns/rules
  - number of possible column combinations is extremely high
- Solution: Concept of a closed pattern
  - Patterns are found in exactly the same set of rows are grouped together and represented by their upper bound
- Example: the following patterns are found in row 2,3 and 4




| i | ri | Class |
|---|---|---|
| 1 | a,b,c,l,o,s | C |
| 2 | a,d, e, h ,p,l,r | C |
| 3 | a,c, e, h ,o,q,t | C |
| 4 | a, e ,f, h ,p,r | ~C |
| 5 | b,d,f,g,l,q,s,t | ~C |

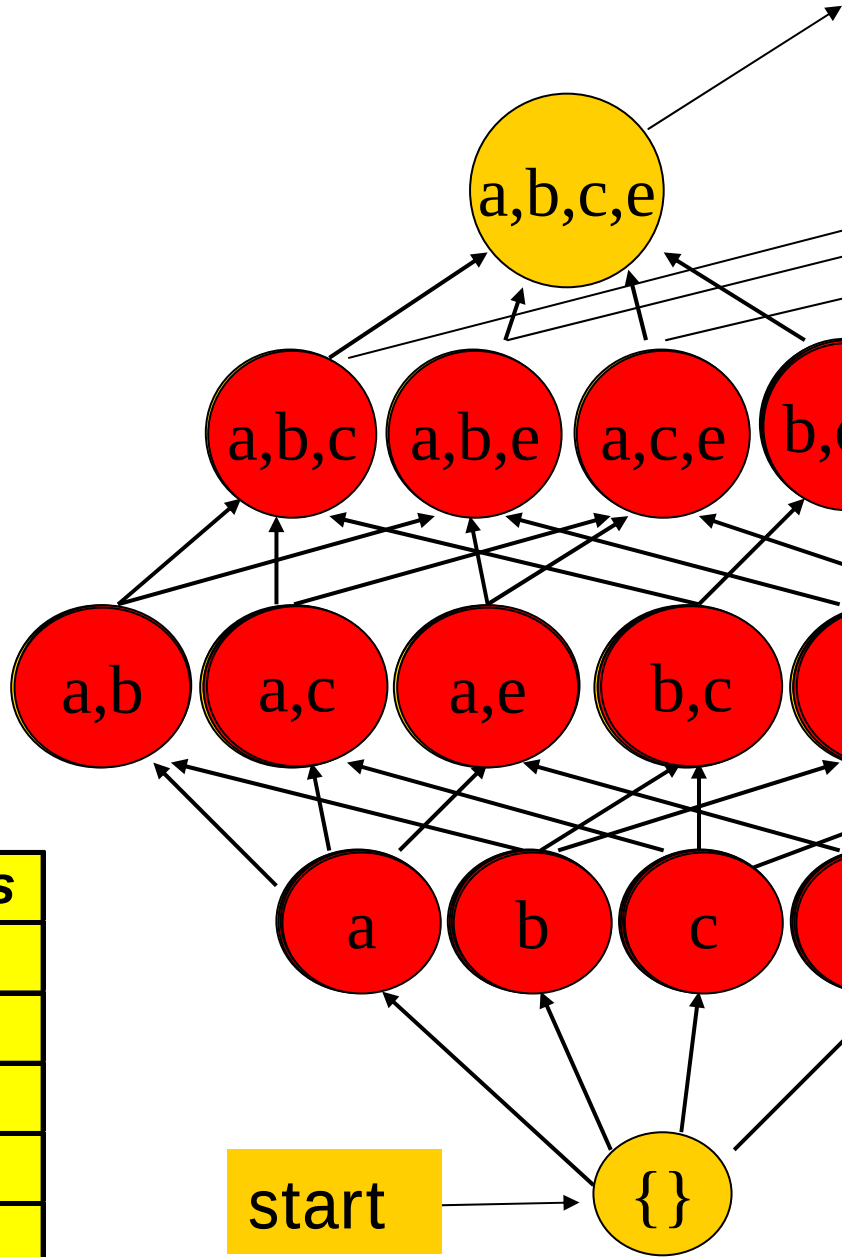
"a" however not part of the group

# Challenge II

- Most existing frequent pattern discovery algorithms perform searches in the column/item enumeration space i.e. systematically testing various combination of columns/items
- For datasets with 1000-100,000 columns, this search space is enormous
- Instead we adopt a novel row/sample enumeration algorithm for this purpose. CARPENTER (SIGKDD'03) is the *FIRST* algorithm which adopt this approach

# Column/Item Enumeration Lattice

- Each nodes in the lattice represent a combination of columns/items
- An edge exists from node A to B if A is subset of B and A differ from B by only 1 column/item
- Search can be done breadth first

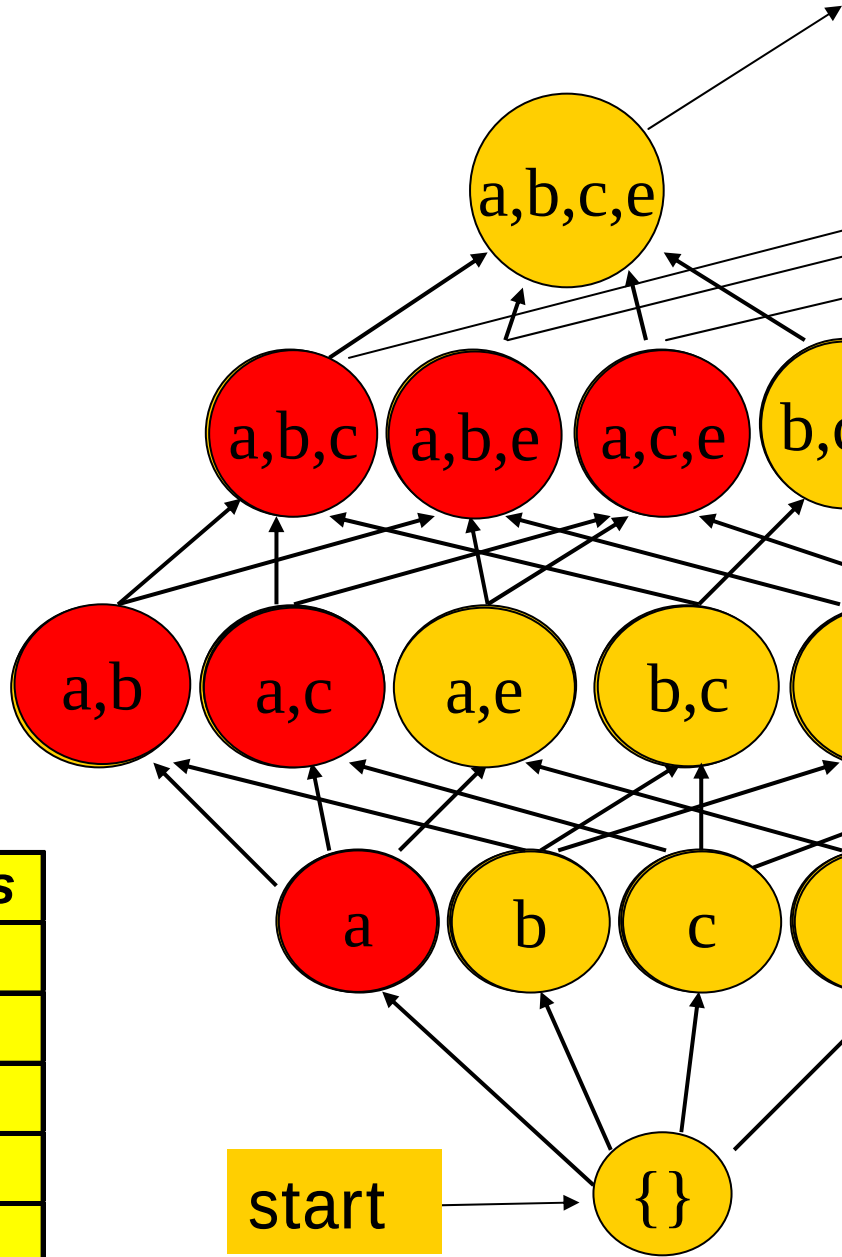



| i | ri | Class |
|---|---|---|
| 1 | a,b,c,l,o,s | C |
| 2 | a,d,e,h,p,l,r | C |
| 3 | a,c,e,h,o,q,t | C |
| 4 | a,e,f,h,p,r | ~C |
| 5 | b,d,f,g,l,q,s,t | ~C |

# Column/Item Enumeration Lattice

- Each nodes in the lattice represent a combination of columns/items
- An edge exists from node A to B if A is subset of B and A differ from B by only 1 column/item
- Search can be done depth first
- Keep edges from parent to child only if child is the prefix of parent

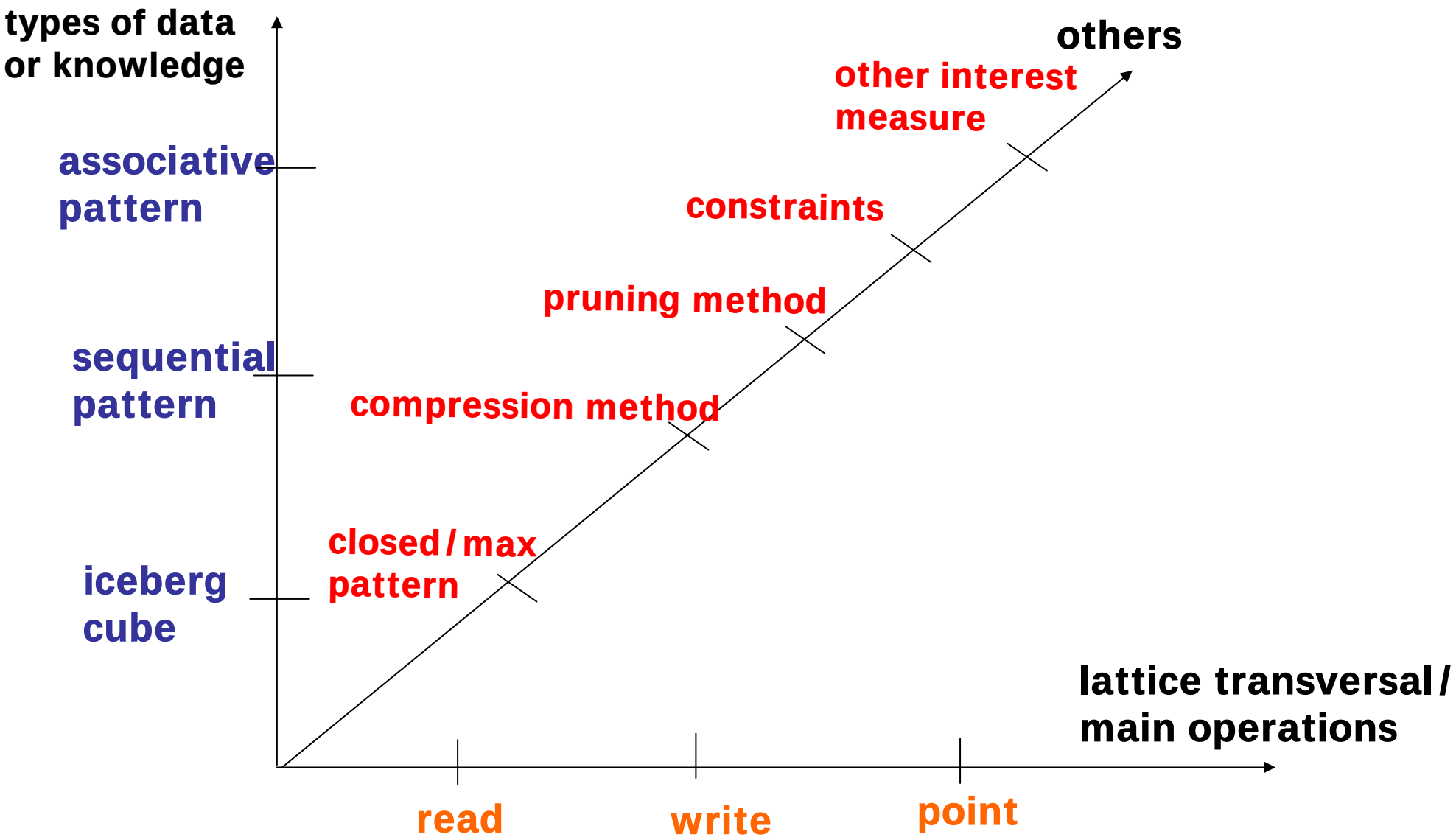
| *i* | *ri* | *Class* |
|---|---|---|
| *1* | *a,b,c,l,o,s* | *C* |
| *2* | *a,d,e,h,p,l,r* | *C* |
| *3* | *a,c,e,h,o,q,t* | *C* |
| *4* | *a,e,f,h,p,r* | *~C* |
| *5* | *b,d,f,g,l,q,s,t* | *~C* |



# General Framework for Column/Item Enumeratio

| | Read-based | Write-based | Point-based |
|---|---|---|---|
| Association Mining | Apriori[AgSr94], DIC | Eclat, MaxClique[Zaki01], FPGrowth [HaPe00] | Hmine |
| Sequential Pattern Discovery | GSP[AgSr96] | SPADE [Zaki98,Zaki01], PrefixSpan [PHPC01] | |
| Iceberg Cube | Apriori[AgSr94] | | BUC[BeRa99], H-Cubing [HPDW01] |

# A Multidimensional View



# Sample/Row Enumeration Algorihtms

- To avoid searching the large column/item enumeration space, our mining algorithm search for patterms/rules in the sample/row enumeration space
- Our algorithms does not fitted into the column/item enumeration algorithms
- They are not YAARMA (Yet Another Association Rules Mining Algorithm)
- Column/item enumeration algorithms simply does not scale for microarray datasets

# Existing Row/Sample Enumeration Algorithms

- CARPENTER(SIGKDD'03)
  - Find closed patterns using row enumeration
- FARMER(SIGMOD’04)
  - Find interesting rule groups and building classifiers based on them
- COBBLER(SSDBM'04)
  - Combined row and column enumeration for tables with large number of rows and columns
- Topk-IRG(SIGMOD’05)
  - Find top-k covering rules for each sample and build classifier directly

# Concepts of CARPENTER

| i | ri | Class |
|---|---|---|
| 1 | a,b,c,l,o,s | C |
| 2 | a,d,e,h,p,l,r | C |
| 3 | a,c,e,h,o,q,t | C |
| 4 | a,e,f,h,p,r | ~C |
| 5 | b,d,f,g,l,q,s,t | ~C |

Example Table

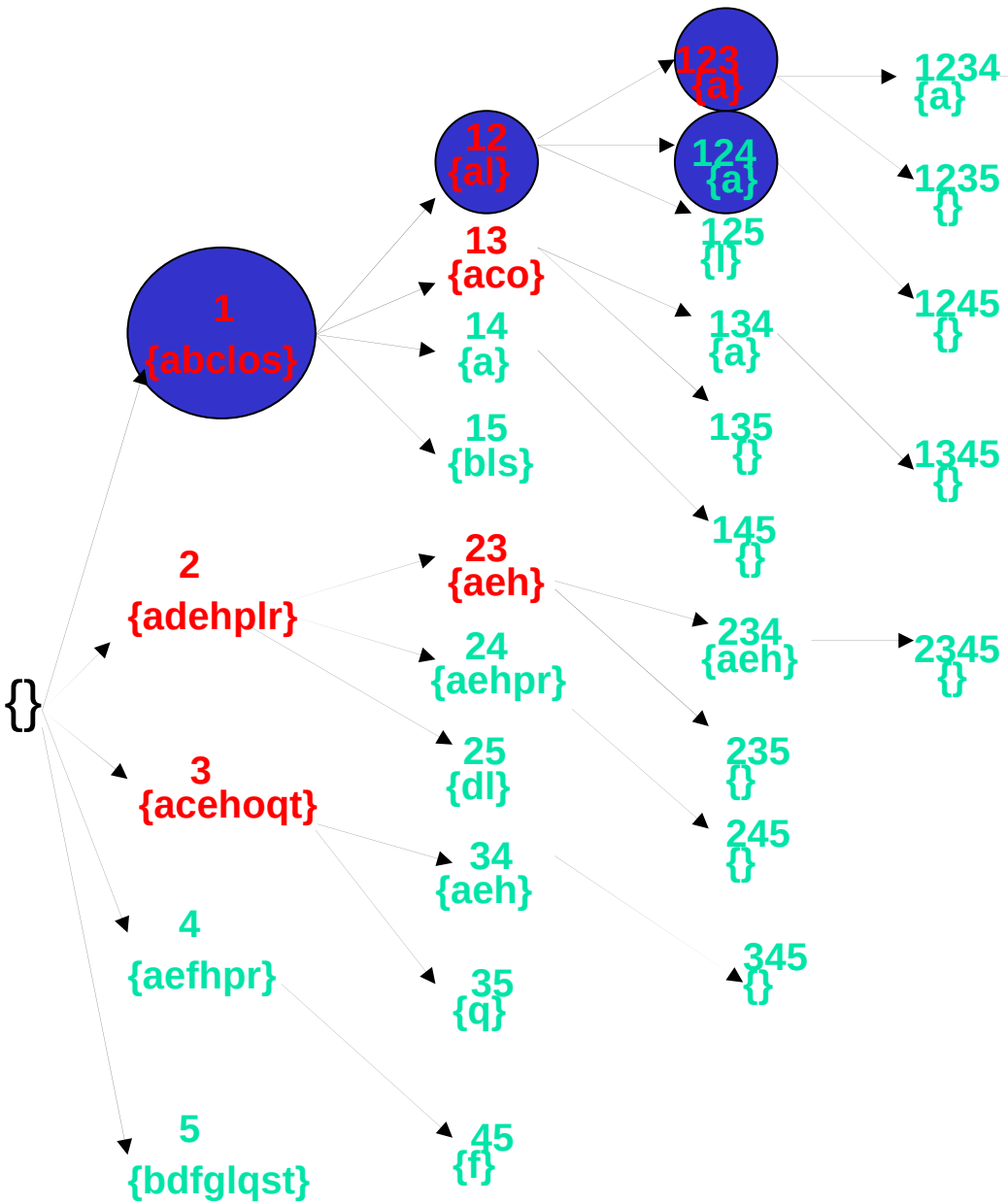
| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| b | 1 | 5 |
| c | 1,3 | |
| d | 2 | 5 |
| e | 2,3 | 4 |
| f | | 4,5 |
| g | | 5 |
| h | 2,3 | 4 |
| l | 1,2 | 5 |
| o | 1,3 | |
| p | 2 | 4 |
| q | 3 | 5 |
| r | 2 | 4 |
| s | 1 | 5 |
| t | 3 | 5 |

Transposed Table,TT

| | C | ~C |
|---|---|---|
| a | 1,2,3 | 4 |
| e | 2,3 | 4 |
| h | 2,3 | 4 |

$TT|_{\{2,3\}}$

# Row Enumeration




$TT|_{\{1\}}$

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| b | 1 | 5 |
| c | 1,3 | |
| l | 1,2 | 5 |
| o | 1,3 | |
| s | 1 | 5 |

$TT|_{\{12\}}$

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| l | 1,2 | 5 |

$TT|_{\{123\}}$ $TT|_{\{124\}}$

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| b | 1 | 5 |
| c | 1,3 | |
| d | 2 | 5 |
| e | 2,3 | 4 |
| f | | 4,5 |
| g | | 5 |
| h | 2,3 | 4 |
| l | 1,2 | 5 |
| o | 1,3 | |
| p | 2 | 4 |
| q | 3 | 5 |
| r | 2 | 4 |
| s | 1 | 5 |
| t | 3 | 5 |

# Pruning Method 1

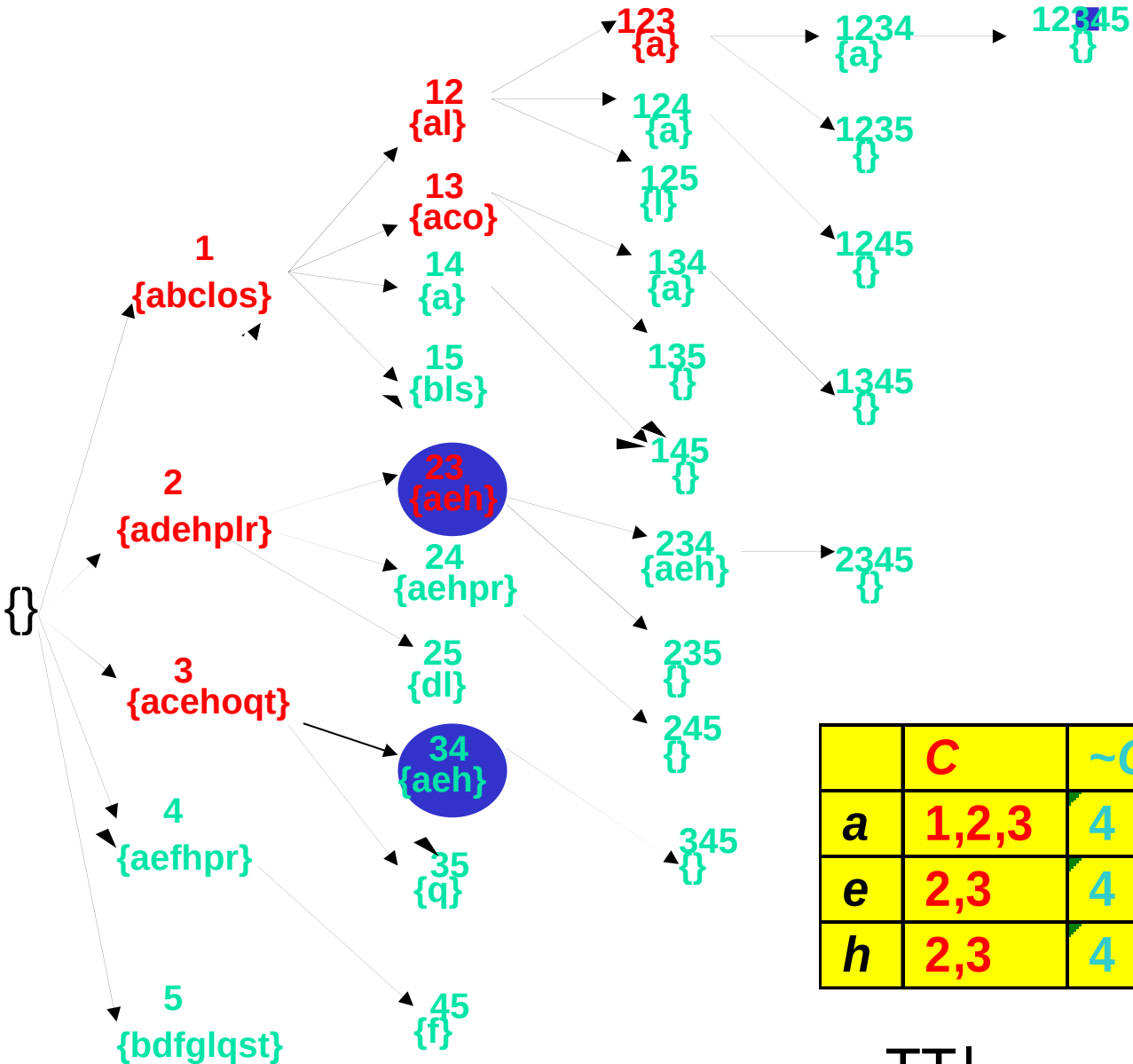
- Removing rows that appear in all tuples of transposed table will not affect results

| | C | ~C |
|---|---|---|
| *a* | 1,2,3 | 4 |
| *e* | 2,3 | 4 |
| *h* | 2,3 | 4 |

$TT|_{\{2,3\}}$


r4 has 100% support in the conditional table of "r2r3", therefore branch "r2 r3r4" will be pruned.

# Pruning method 2




| | C | ~C |
|---|---|---|
| *a* | 1,2,3 | 4 |
| *e* | 2,3 | 4 |
| *h* | 2,3 | 4 |

- if a rule is discovered before, we can prune enumeration below this node
  - Because all rules below this node has been discovered before
  - For example, at node 34, if we found that {aeh} has been found, we can prune off all branches below it

# Pruning Method 3: Minimum Support

- Example: From $TT|_{\{1\}}$, we can see that the support of all possible pattern below node {1} will be at most 5 rows.

$TT|_{\{1\}}$

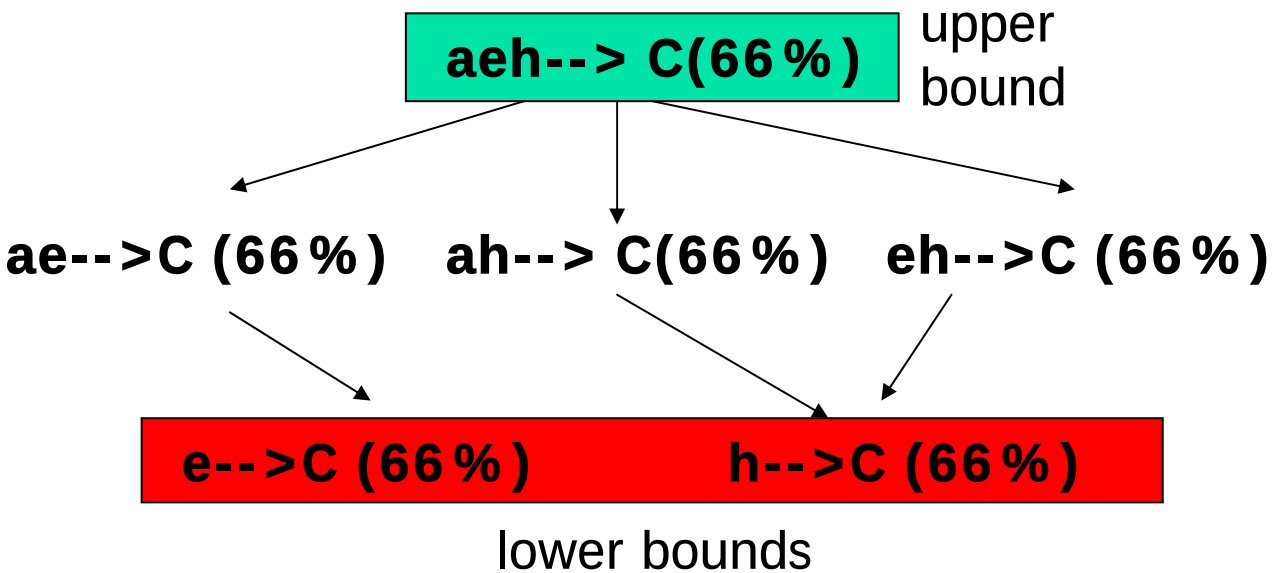
| *ij* | *R*(*ij*) | |
|---|---|---|
| | *C* | ~*C* |
| *a* | 1,2,3 | 4 |
| *b* | 1 | 5 |
| *c* | 1,3 | |
| *l* | 1,2 | 5 |
| *o* | 1,3 | |
| *s* | 1 | 5 |

# From CARPENTER to FARMER

- What if classes exists ? What more can we do ?
- Pruning with Interestingness Measure
  - Minimum confidence
  - Minimum chi-square
- Generate lower bounds for classification/ prediction

# Interesting Rule Groups

- Concept of a rule group/equivalent class
  - rules supported by exactly the same set of rows are grouped together
- Example: the following rules are derived from row 2,3 and 4 with 66% confidence



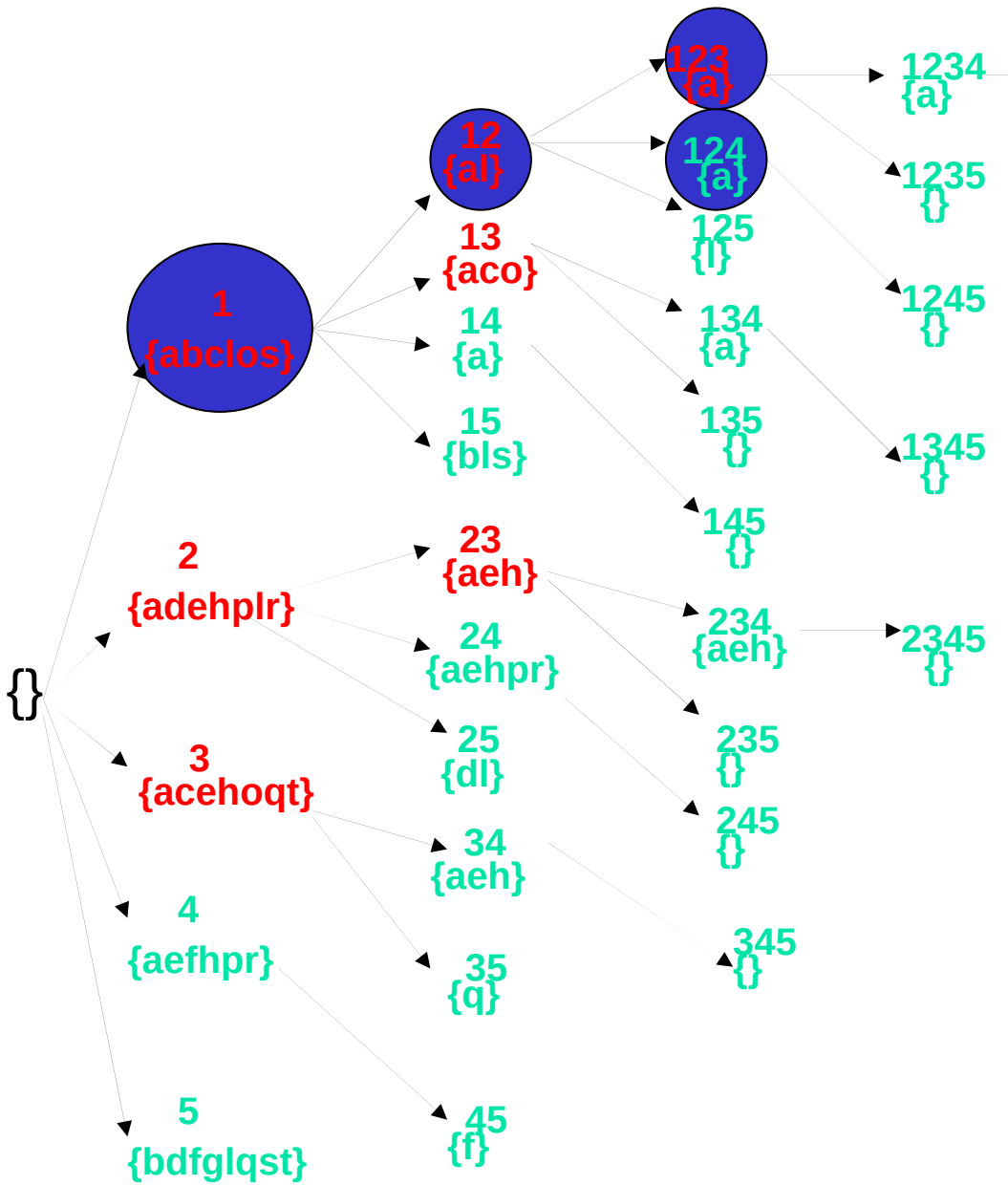

| i | ri | Class |
|---|---|---|
| 1 | a,b,c,l,o,s | C |
| 2 | a,d,e,h,p,l,r | C |
| 3 | a,c,e,h,o,q,t | C |
| 4 | a,e,f,h,p,r | ~C |
| 5 | b,d,f,g,l,q,s,t | ~C |

a-->C however is not in the group

# Pruning by Interestingness Measure

- In addition, find only interesting rule groups (IRGs) based on some measures:
  - minconf: the rules in the rule group can predict the class on the RHS with high confidence
  - minchi: there is high correlation between LHS and RHS of the rules based on chi-square test
- Other measures like lift, entropy gain, conviction etc. can be handle similarly

# Ordering of Rows: All Class C before ~C




| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| b | 1 | 5 |
| c | 1,3 | |
| d | 2 | 5 |
| e | 2,3 | 4 |
| f | | 4,5 |
| g | | 5 |
| h | 2,3 | 4 |
| l | 1,2 | 5 |
| o | 1,3 | |
| p | 2 | 4 |
| q | 3 | 5 |
| r | 2 | 4 |
| s | 1 | 5 |
| t | 3 | 5 |

$TT|_{\{1\}}$

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| b | 1 | 5 |
| c | 1,3 | |
| l | 1,2 | 5 |
| o | 1,3 | |
| s | 1 | 5 |

$TT|_{\{12\}}$

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| l | 1,2 | 5 |

$TT|_{\{123\}}$ $TT|_{\{124\}}$

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |

# Pruning Method: Minimum Confidence

- Example: In TT|{2,3} on the right, the maximum confidence of all rules below node {2,3} is at most 4/5

| | C | ~C |
|---|---|---|
| *a* | 1,2,3,6 | 4,5 |
| *e* | 2,3,7 | 4,9 |
| *h* | 2,3 | 4 |

$TT|_{\{2,3\}}$

# Pruning method: Minimum chi-square

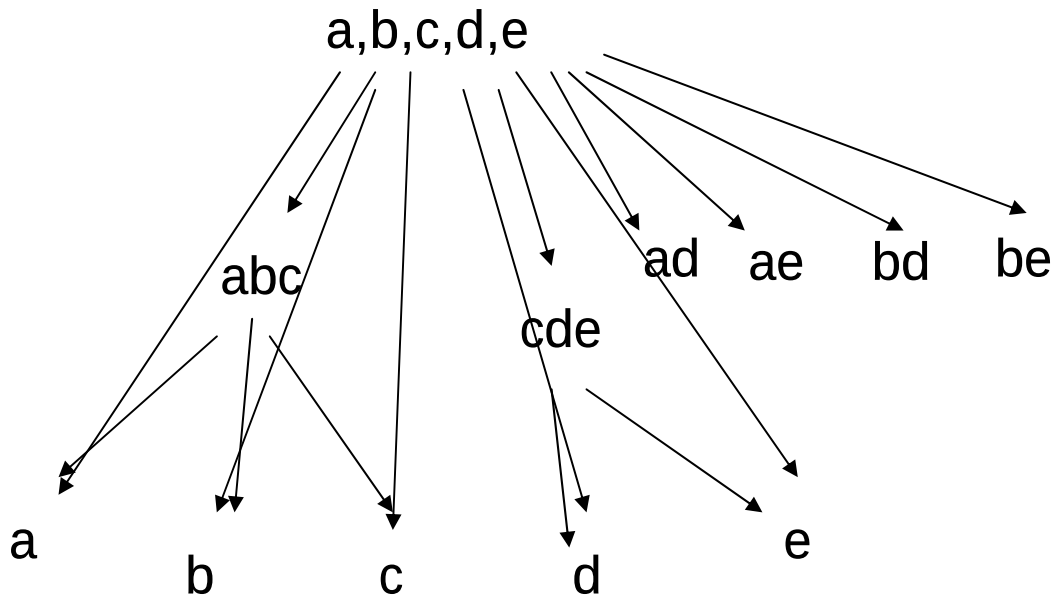
- Same as in computing maximum confidence

| | C | ~C |
|---|---|---|
| ***a*** | **1,2,3,6** | **4,5** |
| ***e*** | **2,3,7** | **4,9** |
| ***h*** | **2,3** | **4** |

$TT|_{\{2,3\}}$

| | C | ~C | Total |
|---|---|---|---|
| A | max=5 | min=1 | Computed |
| ~A | Computed | Computed | Computed |
| | Constant | Constant | Constant |

# Finding Lower Bound, MineLB




- Example: An upper bound rule with antecedent A=abcde and two rows (r1 : abcf ) and (r2 : cdeg)
- Initialize lower bounds {a, b, c, d, e}
- add “abcf”--- new lower {d ,e}
- Add “cdeg”--- new lower bound{ad, bd, ae, be}

Candidate lower bound: a, b, c, d, e, ad, ae, bd, be, cd, ce

Kept since no lower bound is a subset of them

# Implementation

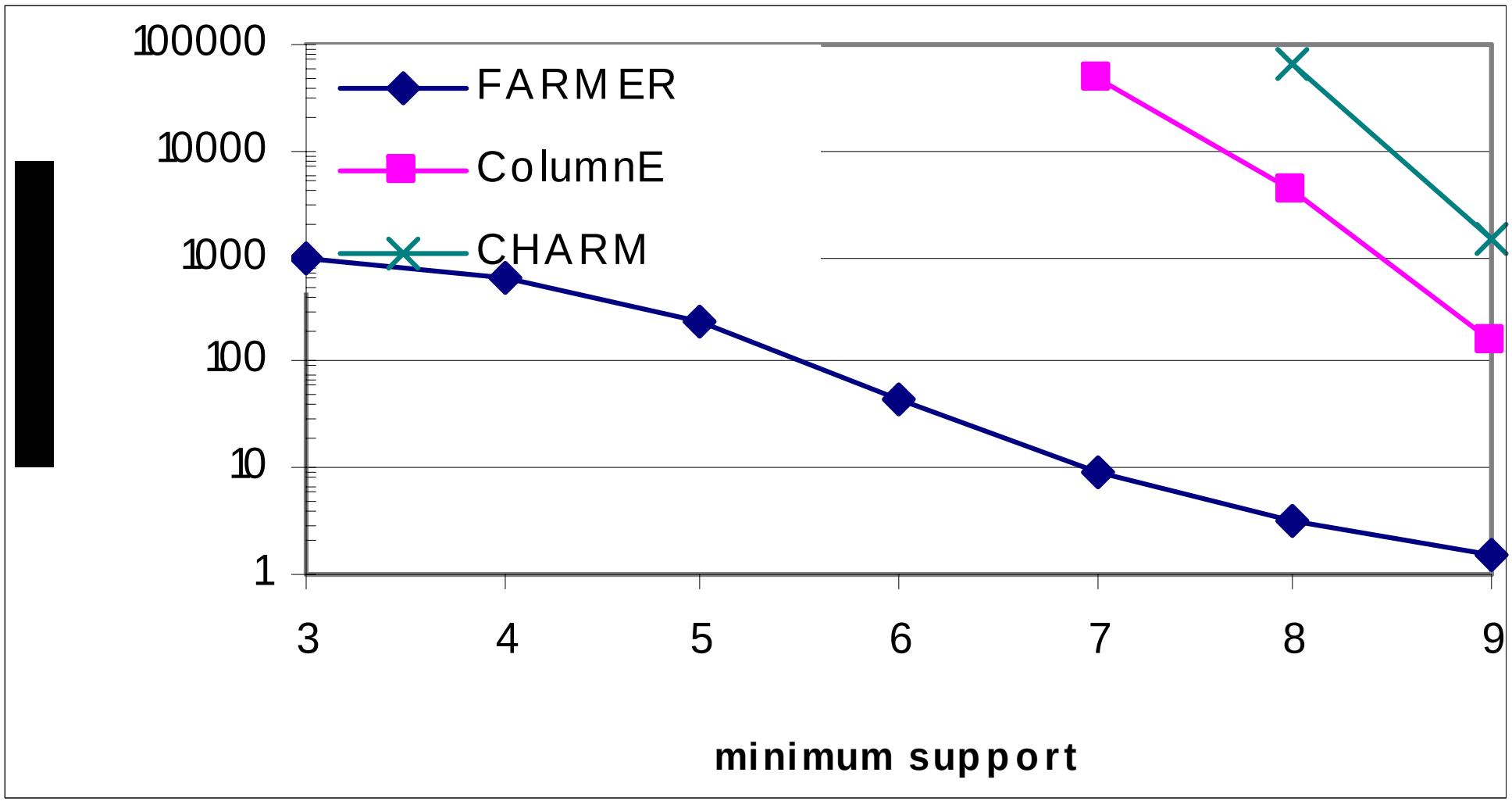
- In general, CARPENTER FARMER can be implemented in many ways:
  - FP-tree
  - Vertical format
- For our case, we assume the dataset can be fitted into the main memory and used pointer-based algorithm similar to BUC

| ij | R (ij ) | |
|---|---|---|
| | C | ~C |
| a | 1,2,3 | 4 |
| b | 1 | 5 |
| c | 1,3 | |
| d | 2 | 5 |
| e | 2,3 | 4 |
| f | | 4,5 |
| g | | 5 |
| h | 2,3 | 4 |
| l | 1,2 | 5 |
| o | 1,3 | |
| p | 2 | 4 |
| q | 3 | 5 |
| r | 2 | 4 |
| s | 1 | 5 |
| t | 3 | 5 |

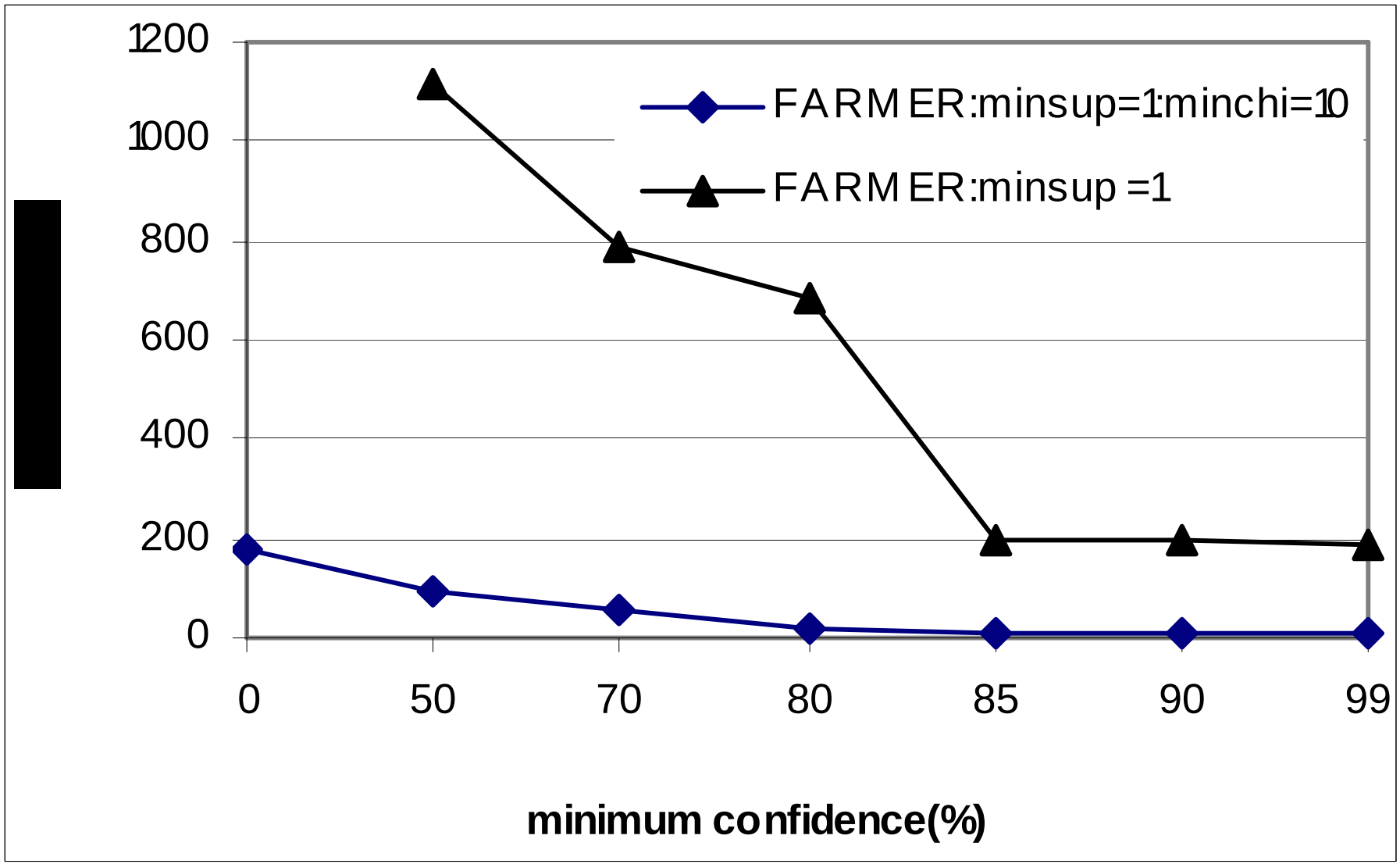
# Experimental studies

- Efficiency of FARMER
  - On five real-life dataset
    - lung cancer (LC), breast cancer (BC) , prostate cancer (PC), ALL-AML leukemia (ALL), Colon Tumor(CT)
  - Varying minsup, minconf, minchi
  - Benchmark against
    - CHARM [ZaHs02] ICDM'02
    - Bayardo's algorithm (ColumE) [BaAg99] SIGKDD'99
- Usefulness of IRGs
  - Classification

# Example results--Prostate



# Example results--Prostate



# Top k Covering Rule Groups

- Rank rule groups (upper bound) according to
  - Confidence
  - Support
- Top k Covering Rule Groups for row *r*
  - k highest ranking rule groups that has row r as support and support > *minimum support*
- Top k Covering Rule Groups = TopKRGS for each row

# Usefulness of Rule Groups

- Rules for every row
- Top-1 covering rule groups sufficient to build CBA classifier
- No min confidence threshold, only min support
- #TopKRGS = $k$ x #rows

# Top-k covering rule groups

- For each row, we find the most significant k rule groups:
  - based on confidence first
  - then support
- Given minsup=1, Top-1
  - row 1: abc→C1(sup = 2, conf= 100%)
  - row 2: abc→C1
    - abcd→C1(sup=1,conf = 100%)
  - row 3: cd→C1(sup=2, conf = 66.7%)
    - If minconf = 80%, ?
  - row 4: cde→C2 (sup=1, conf = 50%)

| class | Items |
|---|---|
| C1 | a,b,c |
| C1 | a,b,c,d |
| C1 | c,d,e |
| C2 | c,d,e |

# Main advantages of Top-k coverage rule group

- The number is bounded by the product of k and the number of samples
- Treat each sample equally → provide a complete description for each row (small)
- The minimum confidence parameter-- instead k.
- Sufficient to build classifiers while avoiding excessive computation

# Top-k pruning

- At node X, the maximal set of rows covered by rules to be discovered down X-- rows containing X and rows ordered after X.
  - minconf ← MIN confidence of the discovered TopkRGs for all rows in the above set
  - minsup ← the corresponding minsup
- Pruning
  - If the estimated upper bound of confidence down X < minconf → prune
  - If same confidence and smaller support → prune
- Optimizations

# Classification based on association rules

- Step 1: Generate the complete set of association rules for each class ( minimum support and minimum confidence.)
  - CBA algorithm adopts apriori-like algorithm -fails at this step on microarray data.
- Step 2:Sort the set of generated rules
- Step 3: select a subset of rules from the sorted rule sets to form classifiers.

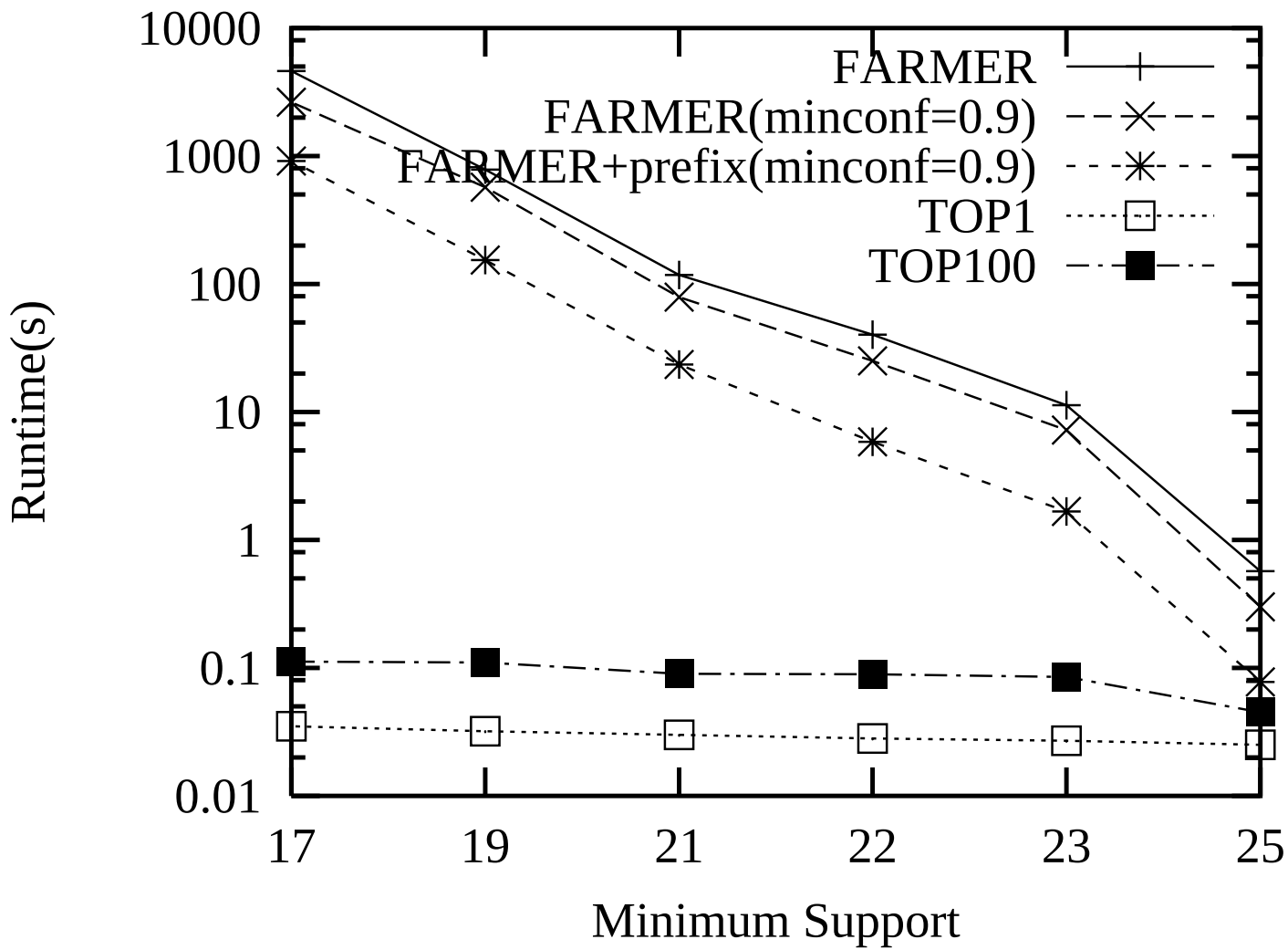
# Features of RCBT classifiers

| Problems | RCBT |
|---|---|
| To discover, store, retrieve and sort a large number of rules | Mine those rules to be used for classification.e.g.Top-1 rule group is sufficient to build CBA classifier |
| Default class not convincing for biologists | Main classifier + some back-up classifiers |
| Rules with the same discriminating ability, how to integrate?<br>Upper bound rules: specific<br>Lower bound rules: general | A subset of lower bound rules—integrate using a score considering both confidence and support. |

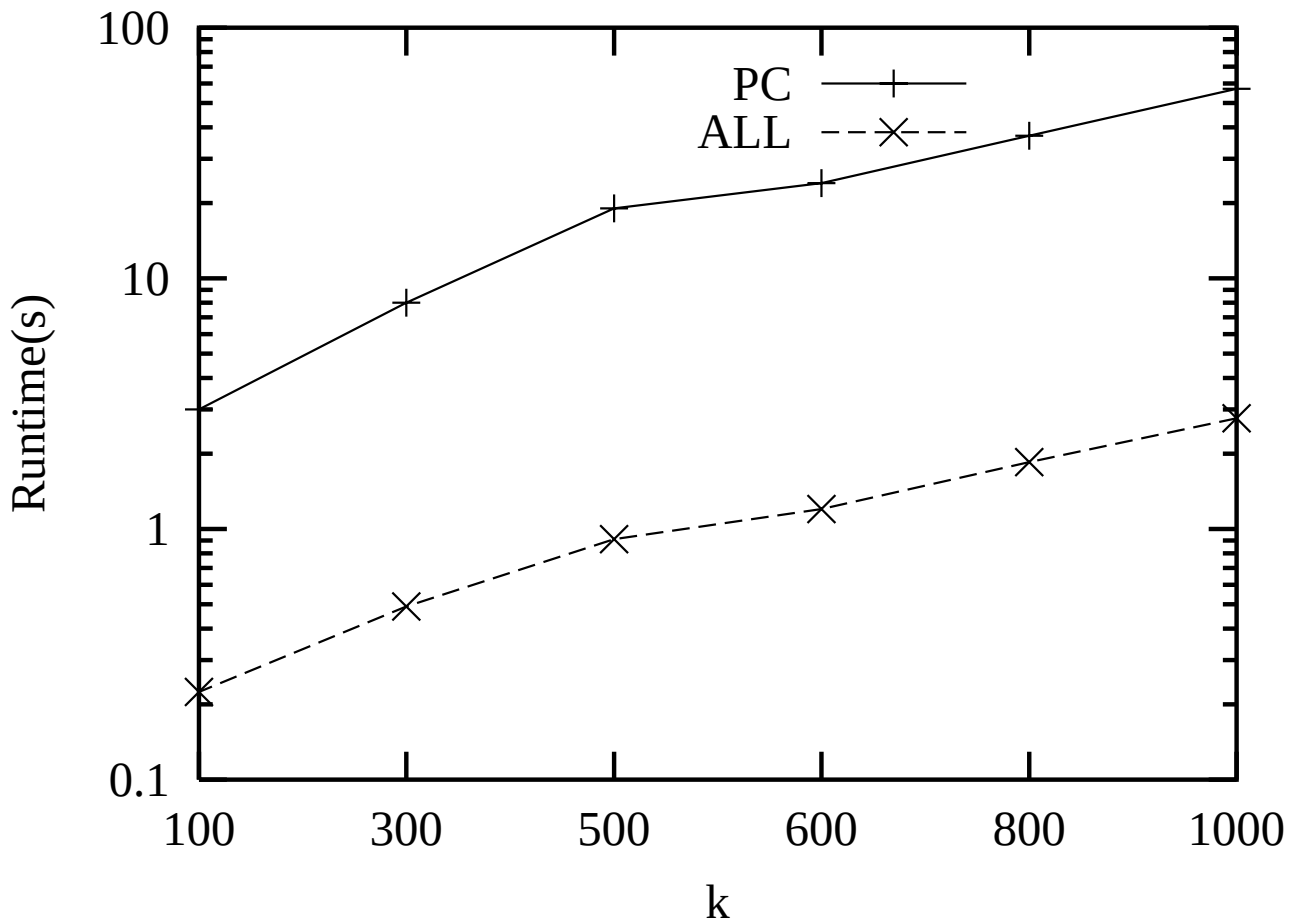
# Experimental studies

- Datasets: 4 real-life data
- Efficiency of Top-k Rule mining
  - Benchmark: Farmer, Charm, Closet+
- Classification Methods:
  - CBA (build using top-1 rule group)
  - RCBT (our proposed method)
  - IRG Classifier
  - Decision trees (single, bagging, boosting)
  - SVM

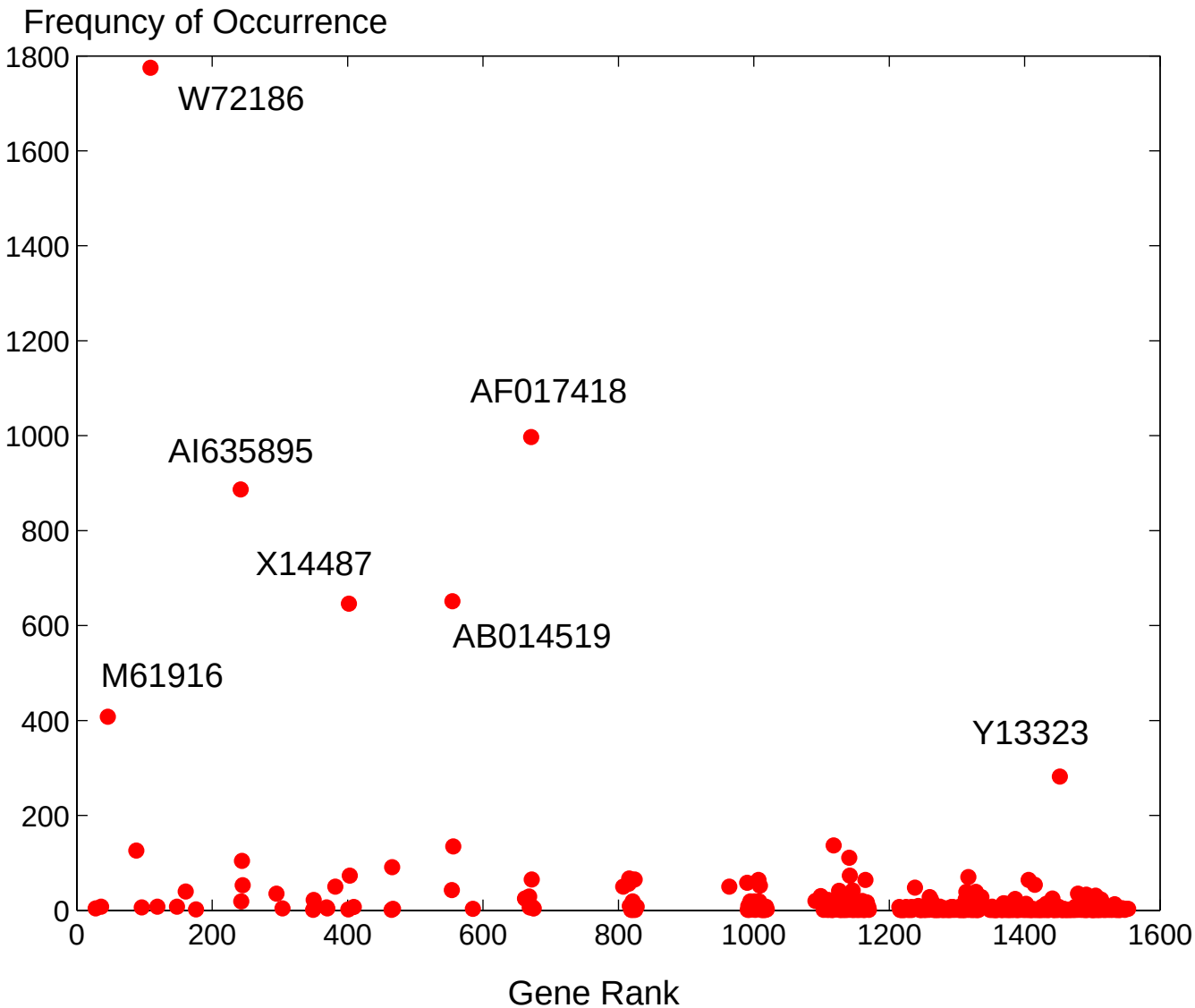
# Runtime v.s. Minimum support on ALL-AML dataset



# Scalability with k



# Biological meaning –Prostate Cancer Data



# Classification results

| Dataset | # Original Genes | # Genes after Discretization | Class 1 | Class 0 | # Training | # Test |
|---|---|---|---|---|---|---|
| ALL/AML (ALL) | 7129 | 866 | ALL | AML | 38 (27 : 11) | 34 |
| Lung Cancer (LC) | 12533 | 2173 | MPM | ADCA | 32 (16 : 16) | 149 |
| Ovarian Cancer (OC) | 15154 | 5769 | tumor | normal | 210 (133 : 77) | 43 |
| Prostate Cancer (PC) | 12600 | 1554 | tumor | normal | 102 (52 : 50) | 34 |

Table 1: Gene Expression Datasets

# Classification results

| Dataset | RCBT | CBA | IRG Classifier | C4.5 family | | | SVM |
|---|---|---|---|---|---|---|---|
| | | | | single tree | bagging | boosting | |
| AML/ALL (ALL) | 91.18% | 91.18% | 64.71% | 91.18% | 91.18% | 91.18% | 97.06% |
| Lung Cancer(LC) | 97.99% | 81.88% | 89.93% | 81.88% | 96.64% | 81.88% | 96.64% |
| Ovarian Cancer(OC) | 97.67% | 93.02% | - | 97.67% | 97.67% | 97.67% | 97.67% |
| Prostate Cancer(PC) | 97.06% | 82.35% | 88.24% | 26.47% | 26.47% | 26.47% | 79.41% |
| Average Accuracy | 95.98% | 87.11% | 80.96% | 74.3% | 77.99% | 74.3% | 92.70% |

**Table 2: Classification Results**

# References


- Anthony K. H. Tung, Rui Zhang, Nick Koudas, Beng Chin Ooi. "Similarity Search: A Matching Based Approach", VLDB'06.
- C. Xia, H. Lu B. C. Ooi, and J. Hu "GORDER: An Efficient Method for KNN Join Processing" VLDB 2004
- Anthony K. H. Tung, Xin Xu, Beng Chin Ooi. "CURLER: Finding and Visualizing Nonlinear Correlated Clusters" In Proceedings, SIGMOD'05, Baltimore, Maryland 2005.
- Gao Cong, Kian-Lee Tan, Anthony K. H. Tung, Xin Xu. "Mining Top-k Covering Rule Groups for Gene Expression Data". In Proceedings SIGMOD'05,Baltimore, Maryland 2005


**Optional References:**


- Feng Pan, Gao Cong, Anthony K. H. Tung, Jiong Yang, Mohammed Zaki, "CARPENTER: Finding Closed Patterns in Long Biological Datasets", In Proceedings KDD'03, Washington, DC, USA, August 24-27, 2003.
- Gao Cong, Anthony K. H. Tung, Xin Xu, Feng Pan, Jiong Yang. "FARMER: Finding Interesting Rule Groups in Microarray Datasets". Iin SIGMOD'04, June 13-18, 2004, Maison de la Chimie, Paris, France.
- Feng Pang, Anthony K. H. Tung, Gao Cong, Xin Xu. "COBBLER: Combining Column and Row Enumeration for Closed Pattern Discovery". SSDBM 2004 Santorini Island Greece.
- Gao Cong, Kian-Lee Tan, Anthony K.H. Tung, Feng Pan. "Mining Frequent Closed Patterns in Microarray Data". In IEEE International Conference on Data Mining, (ICDM). 2004
- Xin Xu, Ying Lu, Anthony K.H. Tung, Wei Wang. "Mining Shifting-and-Scaling Co-Regulation Patterns on Gene Expression Profiles". ICDE 2006.