Skimmer: Rapid Scrolling of Relational Query Results

Manish Singh, Arnab Nandi and H.V. Jagadish

Information Overload

- Hard for users to specify the query results of interest
  - Empty or many-answers problem
- User's often use a non-optimal strategy
  - Order or rank result w.r.t. their most preferred attribute(s), and
  - Choose from the top few results

Finding a needle in haystack

Example: A Realtor's Database

- Contains many attributes
- Buyers have specific needs, but they cannot accurately express it in terms of precise queries.
- Typical buyer's workflow:
  - Express selection predicates on few attributes, and
  - Plow through the large result set, ordered by some attribute(s).
- Example of **"Information Overload"**

Sample Query Result

HID	# Bedroom	School Dist	Crime Rate	Price
2134	4	3.5	0.4	20000
1253	2	1.4	0.1	30000
2341	3	10	0.1	40000
2221	4	25.5	0.7	60000
1531	3	1	0.5	40000
2345	6	30	0.4	70000
2211	2	1.5	0.1	30000
1252	2	1.6	0.9	10000
2349	4	5	0.1	20000
1245	3	0.6	0.2	25000

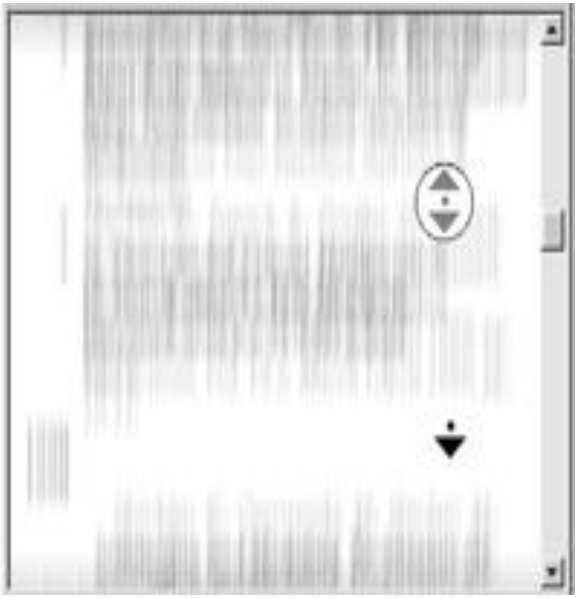
Browsing

- User interface studies have shown:
  - Users use both browsing and querying to search information
  - Users tend to do exploratory browsing before formulating precise queries

- **How to support fast browsing over relational data?**
  - Relational data lacks visuals aids, and
  - Without visual aids browsing speed is restricted to user's reading speed.

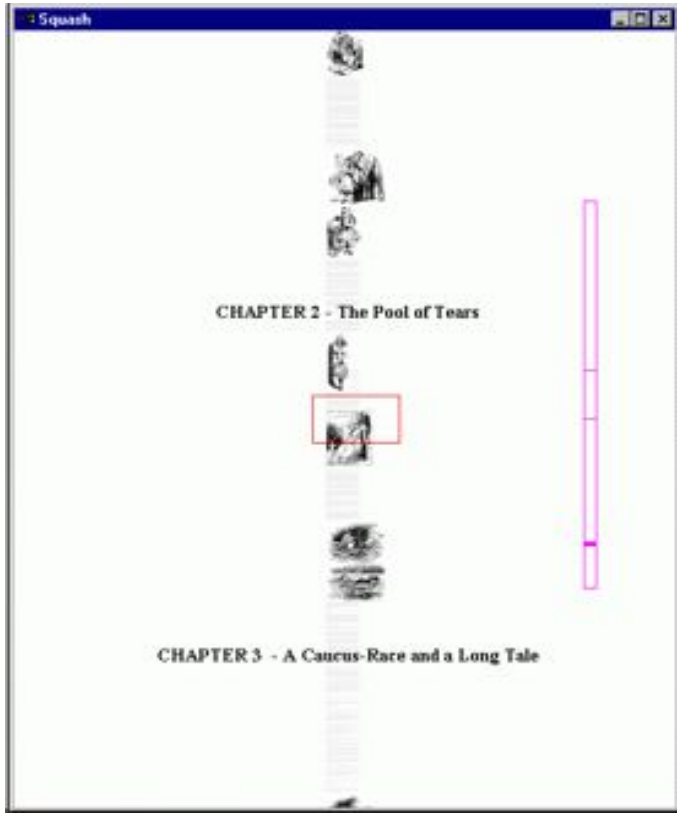
Fast Browsing Using Scrolling Interface

- Scrolling is a widely used interface
- Constraints in fast scrolling:
  - Technical constraints: Data distortion
  - Bodily constraints: Visual perception, memory retention etc.
- **Solution:** Integrate scrolling with automatic zooming
  - Easy transition between overview and detailed information.



Scrolling Large Text Documents

- Our eyes follows visuals hints, such as
  - Difference in font size, colors
  - Section headings, graphics etc.
  - Known structural outline
- Variable speed scrolling [Igarashi, UIST 2000]
  - Automatic zooming based on scrolling rate



Relational Data Vs. Text Documents

- Alphanumeric values with few visual markers
- Lacks structural outline

Results: Page **3** of **9**

Date	Price ▲	Bed.	Bath.
14 Jan	455K	2	2
13 Jun	477K	4	3
8 Dec	480K	4	4
6 Feb	505K	5	3
19 May	555K	5	4
12 Jan	578K	3	3
16 May	593K	4	5
5 Jun	650K	5	3

Sorted Column

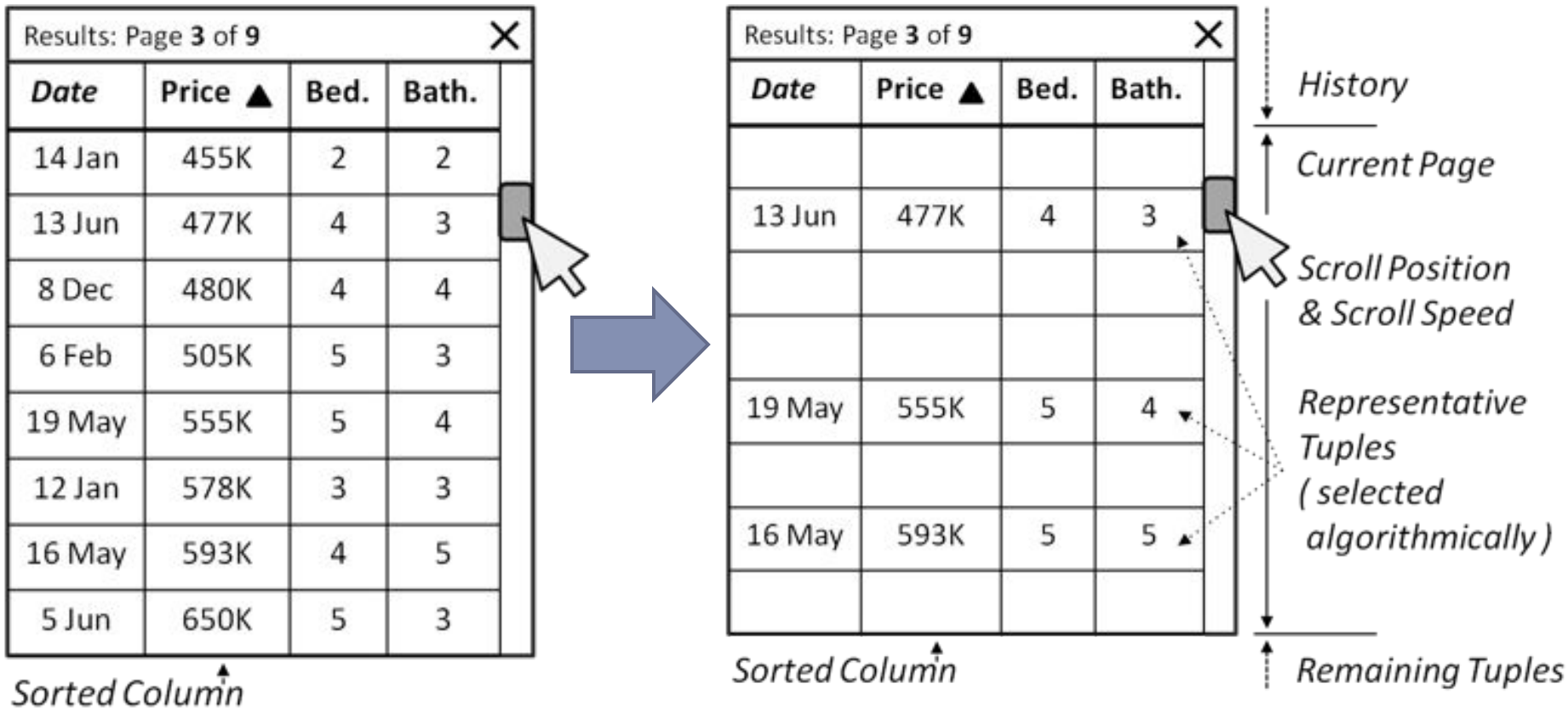
Problem Definition

- Design a variable-speed scrolling system for relational data
  - Use user's scrolling behavior to determine how much and what information to display
- **Skimmer**: Allows users to skim through information at different browsing speeds.

Outline

- Motivation
- Scrolling Interface
  - User Interface
  - Goodness Metric
- Algorithms
- Experimental Evaluation
- Conclusion

User Interface



User Interface

- **Input:** Sorted query result R
- **Output:** R requires S pages $\{P_1, P_2, \ldots, P_S\}$ for display
  - Display representatives: $\{D_1, D_2, \ldots, D_S\}$
  - $D_i \subseteq P_i$ and it is computed based on:
    - User's current scrolling speed
    - Contents of page P_i
    - User's current browsing history
- **Benefit:** Reduces information overload by showing summarized, non-redundant and diverse information

Fast Browsing Over Houses

Goodness Metric: Information Loss

- **Tuplewise information loss** of a non-displayed tuple, t_{nd} from P_i where t_d is most similar tuple from D_i and $H(sid)$

$$TIL(t_{nd}, sid) = V(t_{nd}, t_d)$$

- **Pagewise information loss** score of page P_i:

$$PIL(P_i, sid) = \sum_{t_p \in P_i} TIL(t_p, sid)$$

- **Cumulative information loss** for result set R and scroll log SL

$$CIL(SL, R) = \sum_{sid=1}^{|SL|} PIL(P_i, sid)$$

Outline

- Motivation
- Scrolling Interface
- Algorithms
  - Two naïve algorithms
  - Two K-Medoids based algorithms
    - Relationship between K-Medoid and *PIL* Score
  - Three K-Means based algorithms
- Experimental Evaluation
- Conclusion and Future Work

Naïve Sampling

- Compute set $D_i = K_i$ tuples from page P_i
  - K_i is determined based on user's current scrolling speed
- Random sampling
  - Pick K_i random tuples from P_i
- Uniform sampling
  - Pick K_i evenly spaced tuples from P_i

K-Medoid

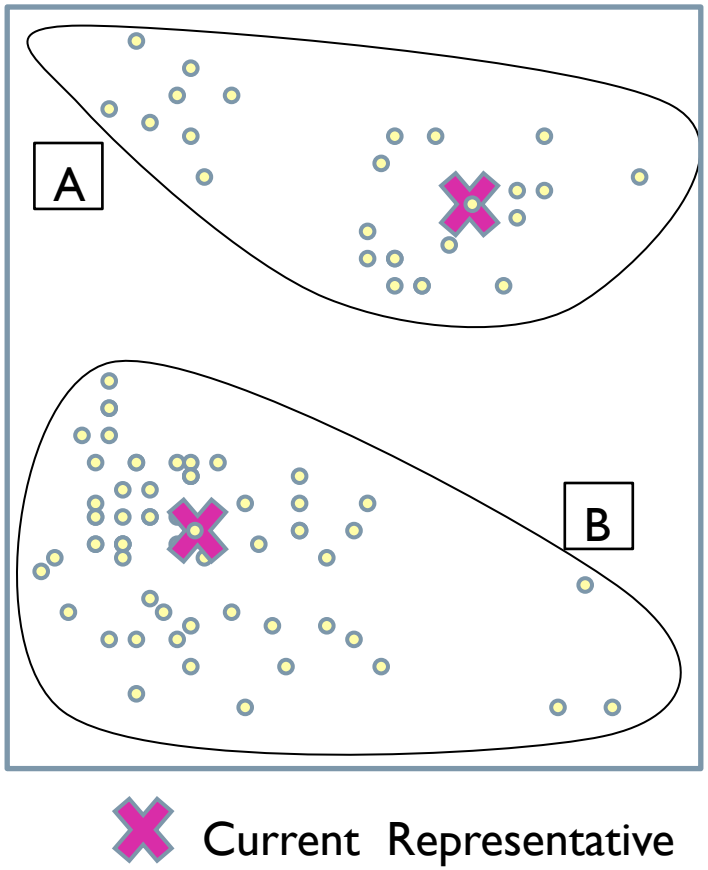
- It's a clustering algorithm that partitions a dataset D, containing N elements, into K partitions
- Each partition is represented by an actual sample point
- It minimizes the following absolute error criterion:

$$E_{KMedoids}(P) = \sum_{j=1}^{K} \sum_{p \in C_j} V(p, o_j)$$

- Best known heuristic solutions: PAM, CLARA and CLARANS

Local K-Medoid (*LKMed*)

- $D_i = PAM\ (P_i, K_i)$
- PAM Algorithm:
  - Initialize clusters centers
  - Repeat until convergence
    - Assignment: Assign each point to nearest cluster
    - Update: Swap based greedy update of cluster centers
- CLARA and CLARANS not suitable for small datasets



Importance of History

- **Our goal:** Show non-redundant, diverse information to the user

House A
10,000$

House C
40,000$

House B
40,000$

Relationship: PIL Score and K-Medoid

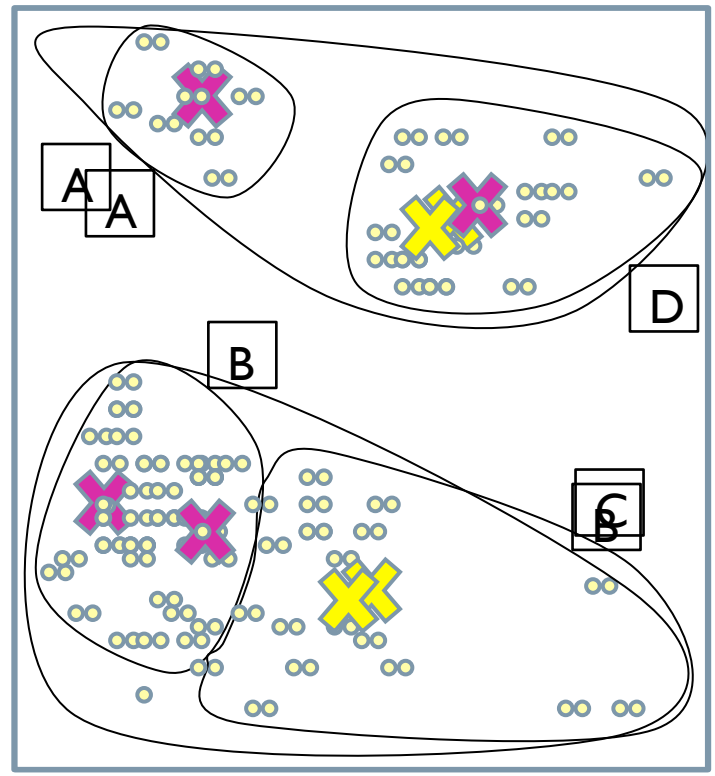
- K-Medoid optimizes:

$$EKMedoids(P) = \sum_{j=1}^{K} \sum_{p \in C_j} V(p, o_j)$$

- *PIL* Score of page *P*

$$PIL(P) = \sum_{j=1}^{L} \sum_{p \in C_j} V(p, o_j)$$

$$where\ L = \sum_{l=1}^{i} K_l \text{ and } \sum_{l=1}^{i-1} K_l < j \leq \sum_{l=1}^{i} K_l$$

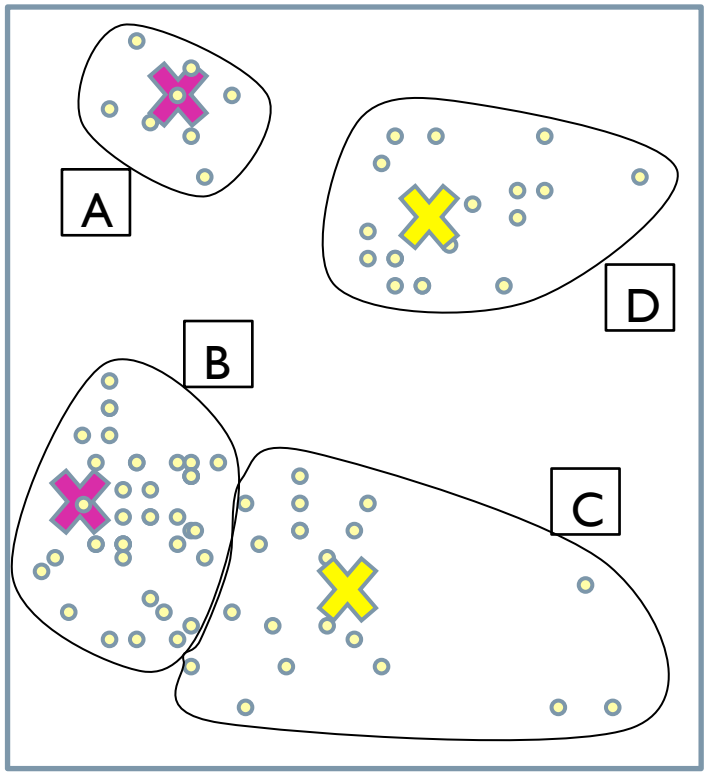


Historical Representative

Current Representative

Historical K-Medoid (*HKMed*)

- $D_i = HKMed\ (P_i, K_i)$
  - Minimizes the exact *PIL* score
- *HKMed* Algorithm
  - Initialize the cluster centers
  - Repeat until convergence
    - Assignment: Assign each point to nearest cluster.
    - Update: Update unfixed cluster centers based on greedy swap



Historical Representative

Current Representative

Performance Issues

- Computational constraints: Satisfy user's non-linear scrolling behavior
  - Next page representative is selected based on:
    - Past displayed content
    - User's current scroll rate
  - Desired computation time: Less than 100 ms

- PAM : $O(K*(N-K)^2)$ dist computations per iteration

Approximate K-Medoid

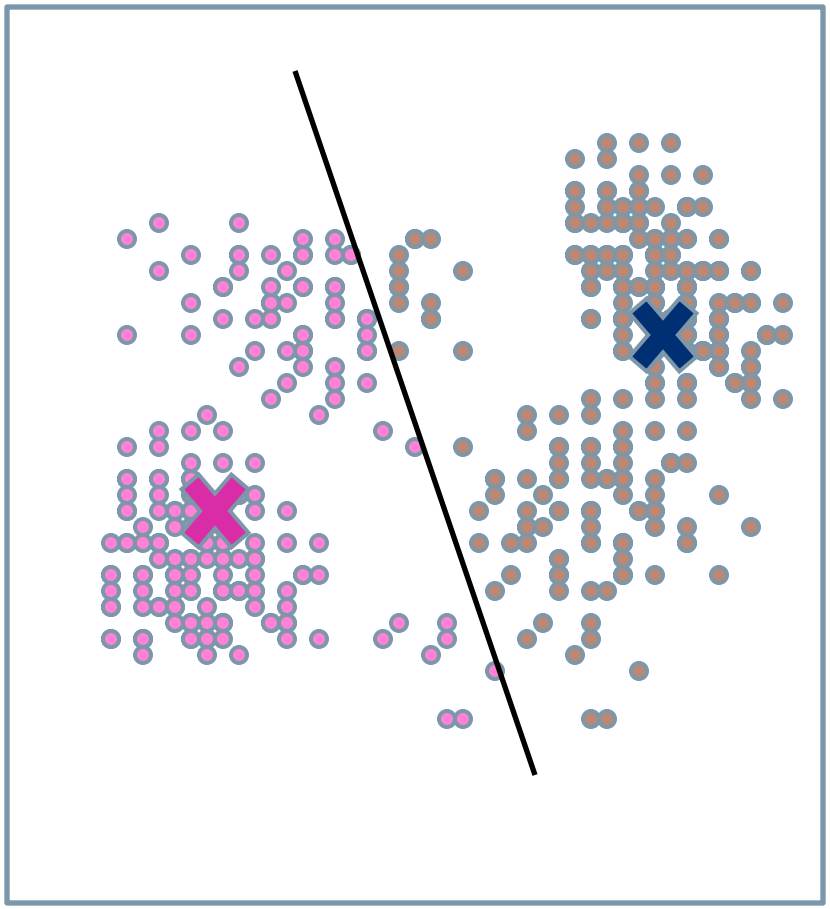
- K-Means is an efficient partition based clustering algorithm. It divides a dataset into 'K' partitions.
  - It is O(K*N) as compared to $O(K(N-K)^2)$ in K-Medoid
- Each partition is represented by partition centroid.
- It minimizes the following square-error criterion:

$$EKMeans(P) = \sum_{j=1}^{K} \sum_{p \in C_j} |p - m_i|^2$$

- It can only be used for numerical attributes and Euclidean distance function.

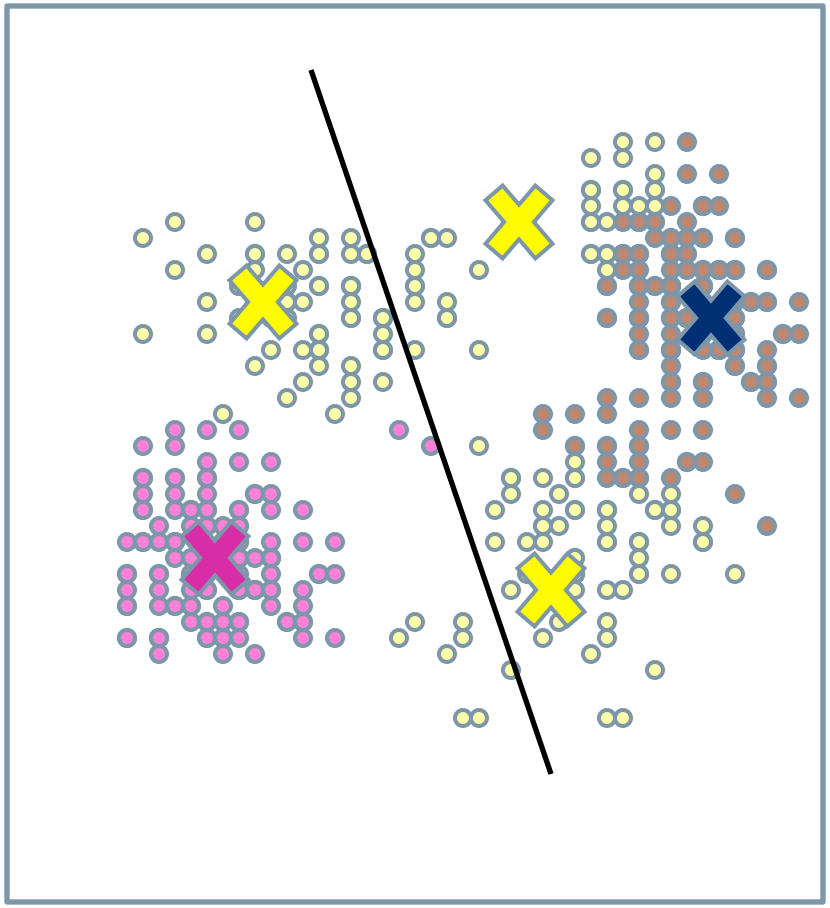
Local K-Means (LKMeans)

- Algorithm
  - KCenters = KMeans (P_i, K_i)
  - D_i = NN (KCenters, P_i)
- KMeans Algorithm
  - Initialize cluster centers
  - Repeat until convergence
    - Assignment: Assign each point to nearest cluster.
    - Update: New cluster centers by computing mean of all assigned points.



Historical K-Means (HKMeans)

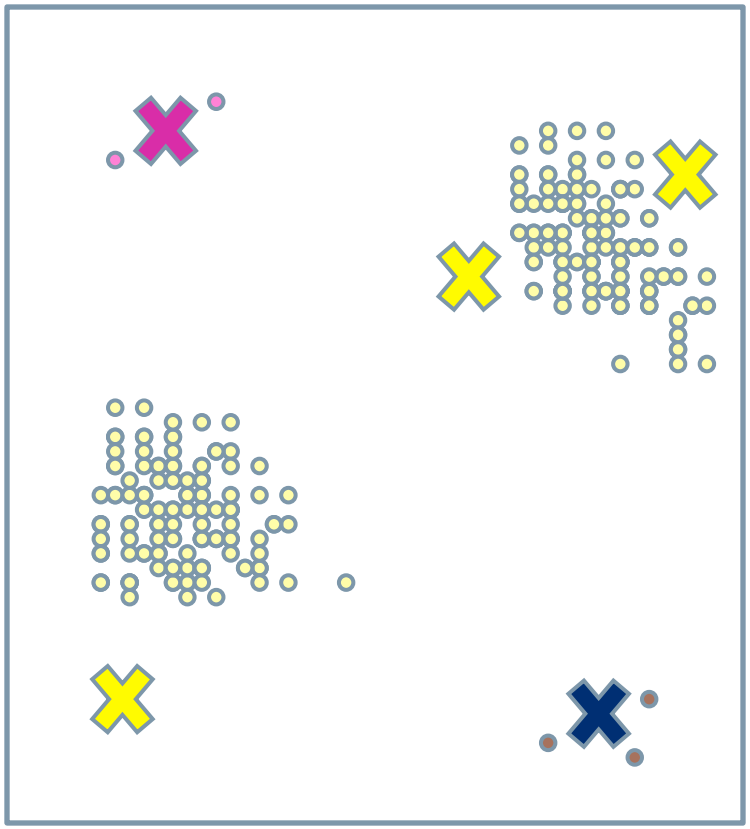
- Similar motivation as that of historical K-Medoid.
- Algorithm
  - KCenters = HKMeans (P_i, K_i)
  - D_i = NN (KCenters, P_i)
- HKMeans Algorithm
  - Initialize cluster centers
  - Repeat until convergence
    - Assignment: Assign each point to nearest cluster.
    - Update: New unfixed cluster centers by computing mean of all assigned points.



Historical Representative

Effect of Initialization

- HKMeans worse than LKMeans in terms of CIL Score
- Unlike *HKMed*, *HKMeans can* get caught in local minimum
  - Bad initial cluster centers
  - Representatives being determined based on the outliers

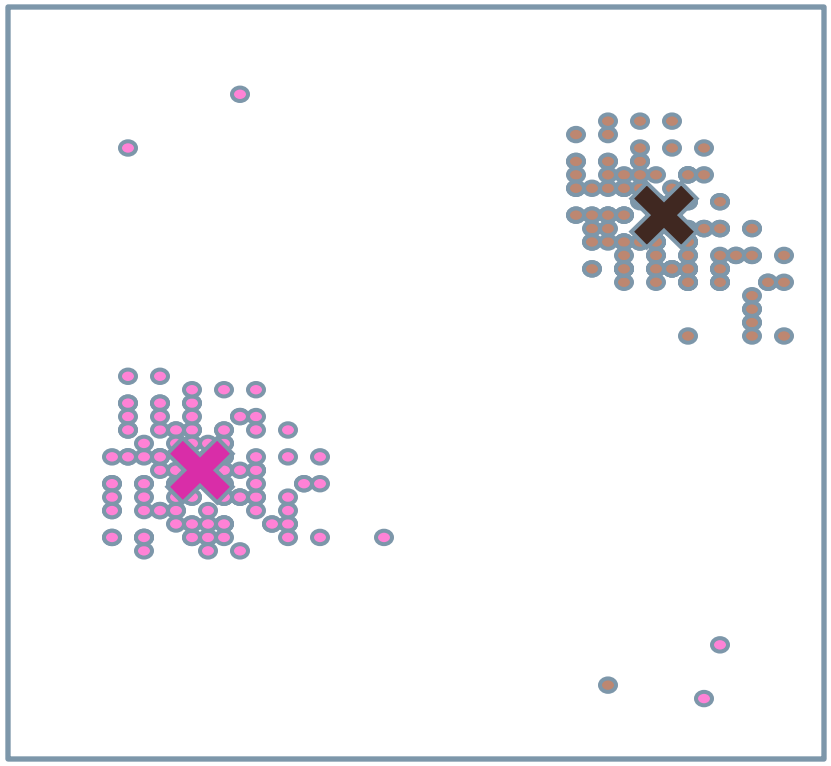


Historical Representative

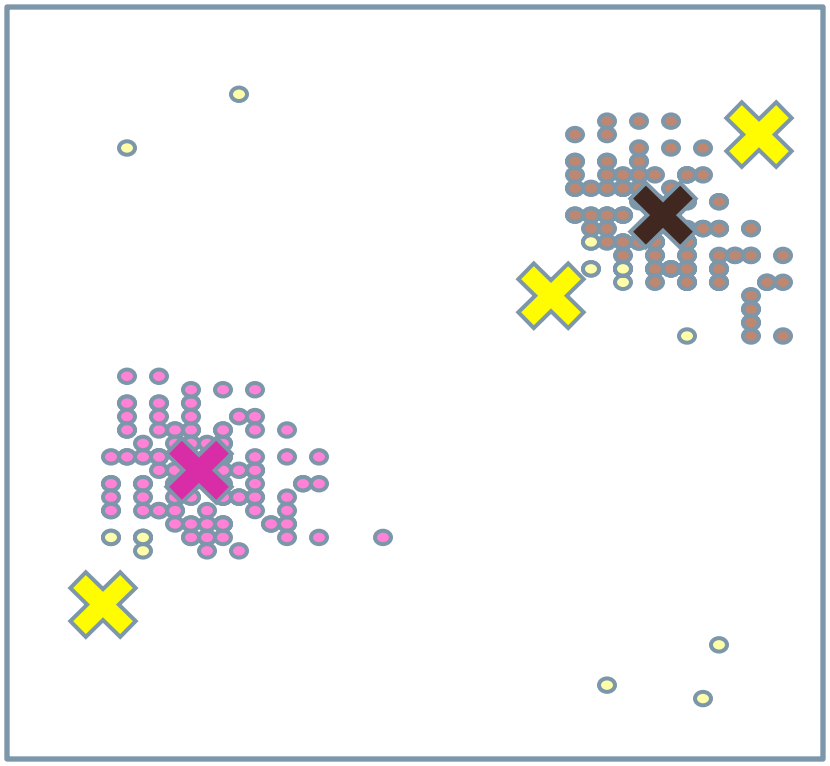
Two-Phase K- Means (TPKMeans)

- Phase 1
  - Choose good initial cluster centers using LKMeans
- Phase 2
  - Select non-redundant representatives using HKMeans
- Benefits
  - Information quality quite close to HKMed
  - Runs almost N times faster as compared to K-Medoids based algorithms

Two Phase K-Means (TPKMeans)



Local K-Means



Historical K-Means

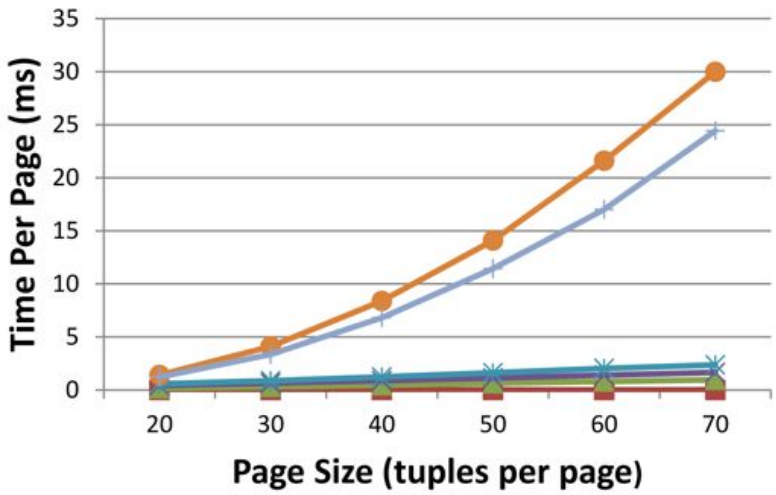
Outline

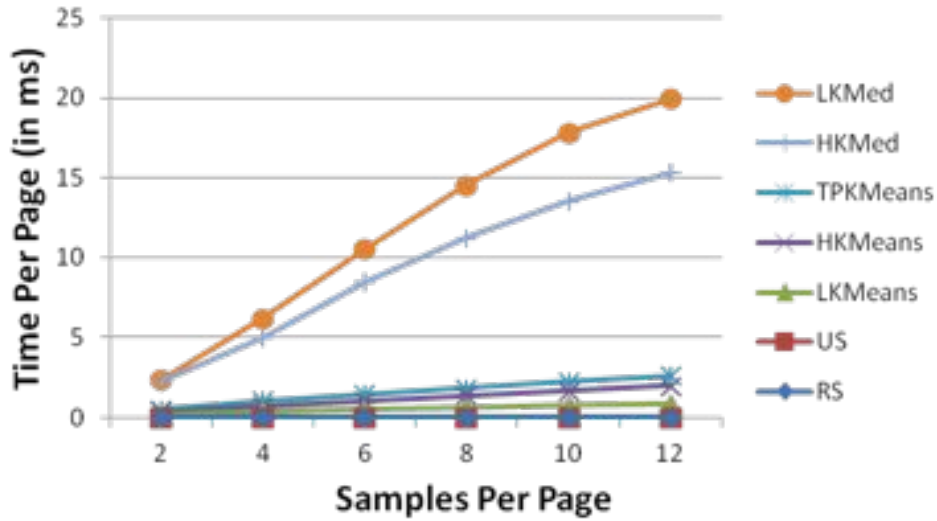
- Motivation
- Scrolling Interface
- Algorithms
- Experimental Evaluation
  - Performance
  - Information Quality
  - User Study
- Conclusion and Future Work

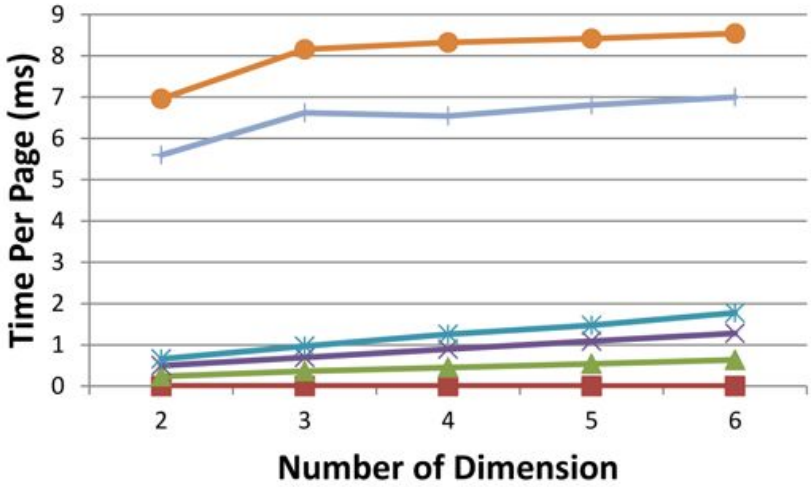
Experimental Goals

- Computational Performance
  - Page size
  - Number of dimensions
  - Sampling rate
- Information Quality
- User Study

Performance







- HKMed and LKMed need more time
  - Not suitable for large page size or high sampling rate
  - HKMed is faster than LKMed
- All algorithms satisfy interactive response constraint

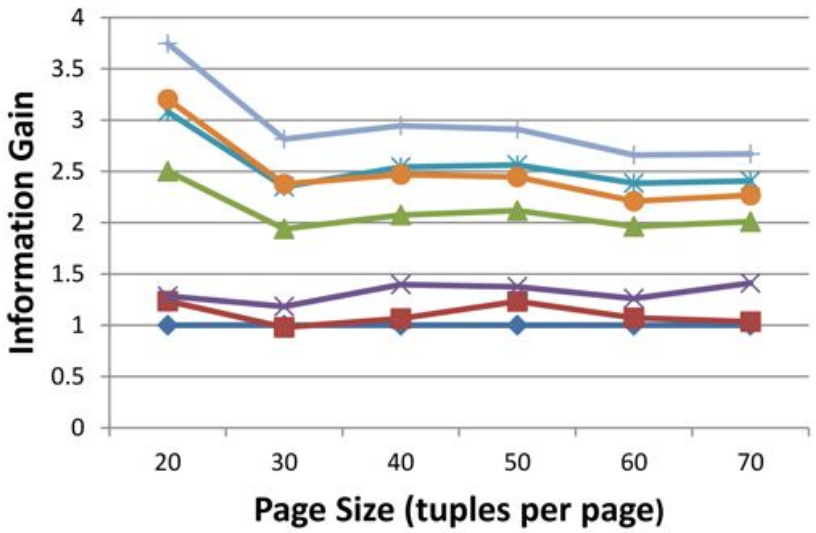
Experimental Goals

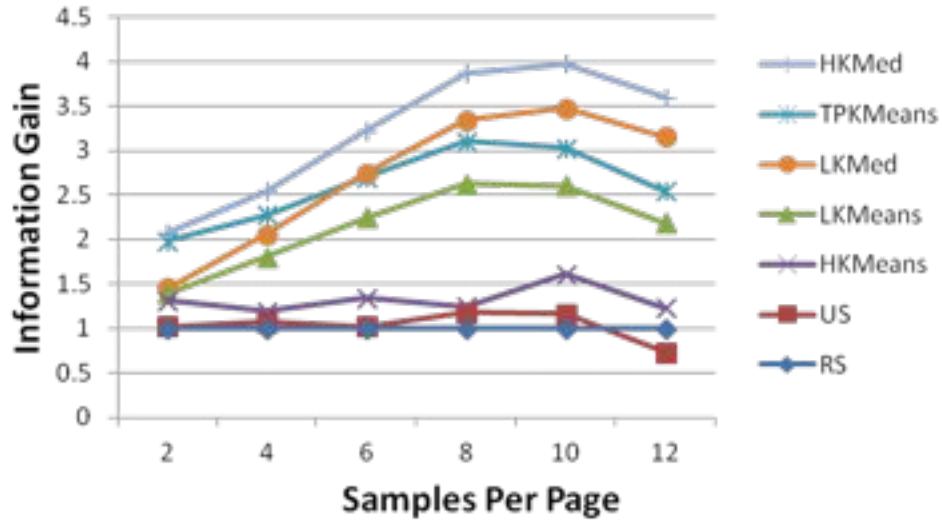
- Computational Performance
- Information Quality
  - Information Gain: We use Random Sampling as baseline B

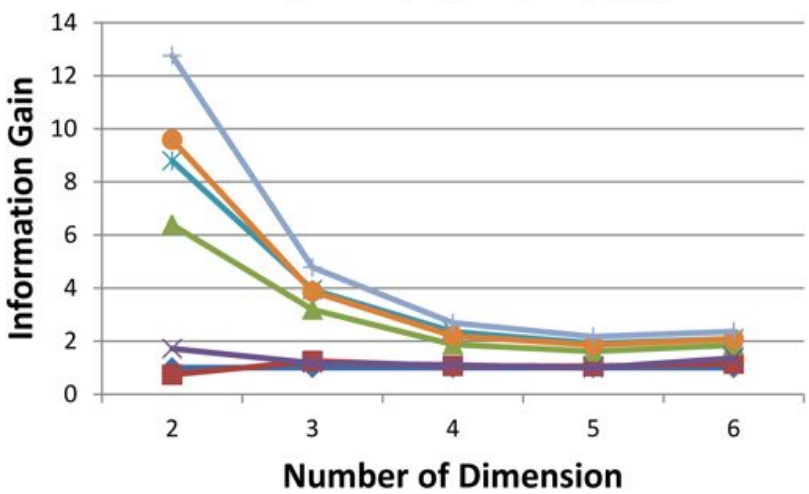
$$IG(A,B) = \frac{CIL_B(SL,R)}{CIL_A(SL,R)}$$

  - Page size
  - Number of dimensions
  - Sampling rate
- User Study

Information Quality

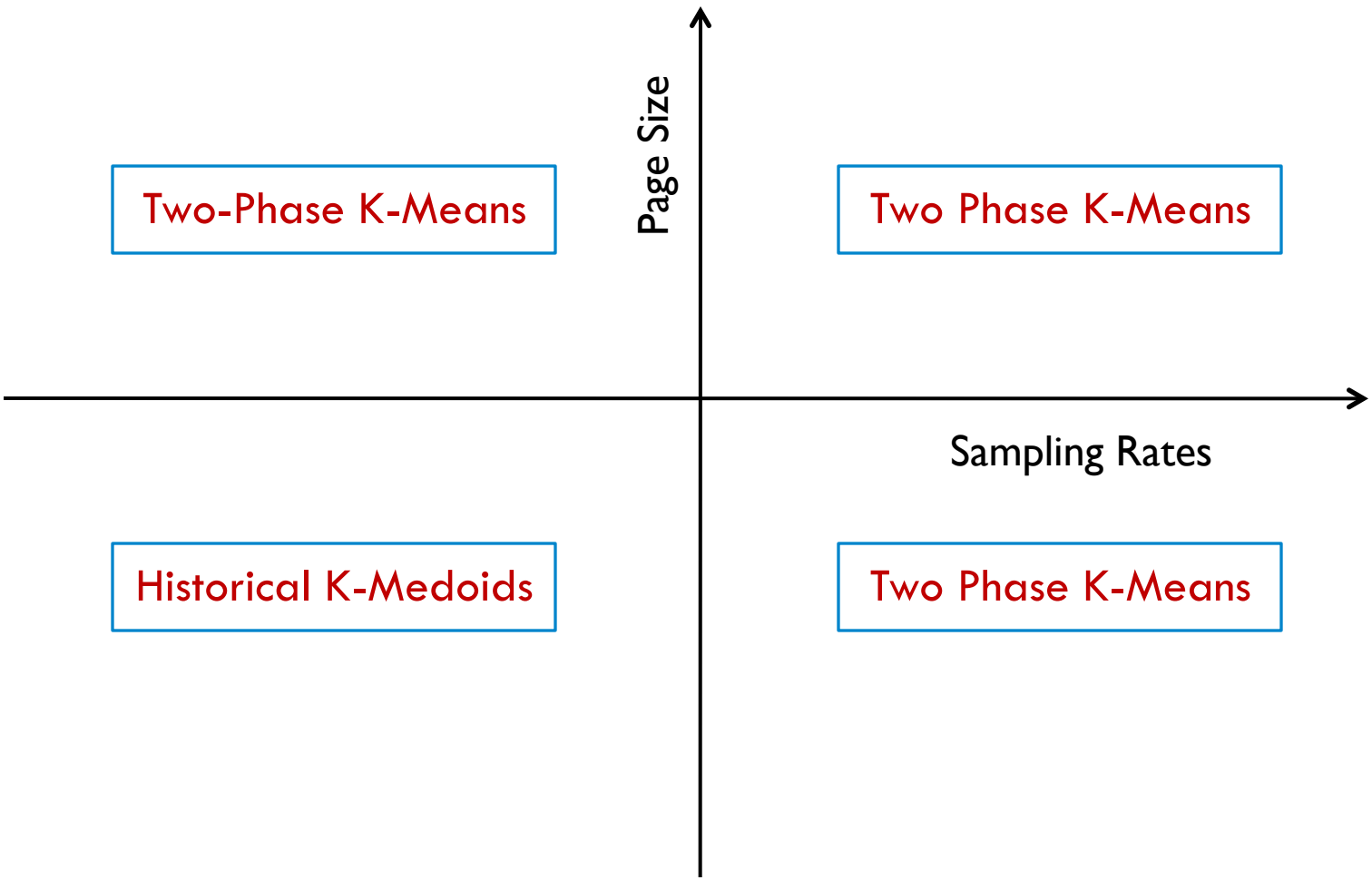






- *HKMed* is best followed by *TPKMeans* and *LKMed*
- *HKMeans* is almost close to random sampling
- Information gain decreases with increasing # dimensions

Summary Recommendations

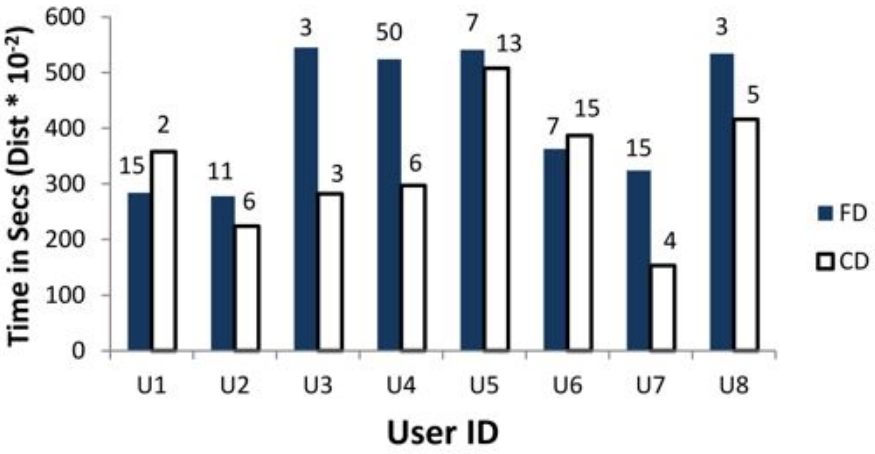


Experimental Goals

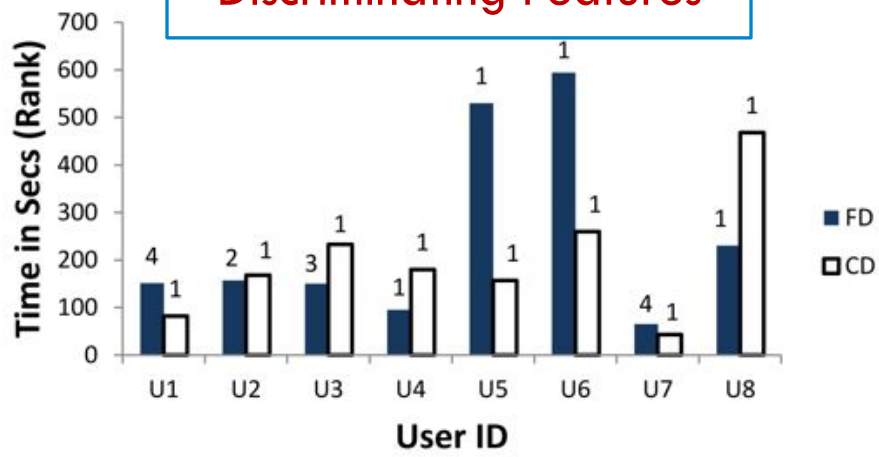
- Computational Performance
- Information Quality
- User Study
  - Users' efficiency and quality of response to three tasks

User Study

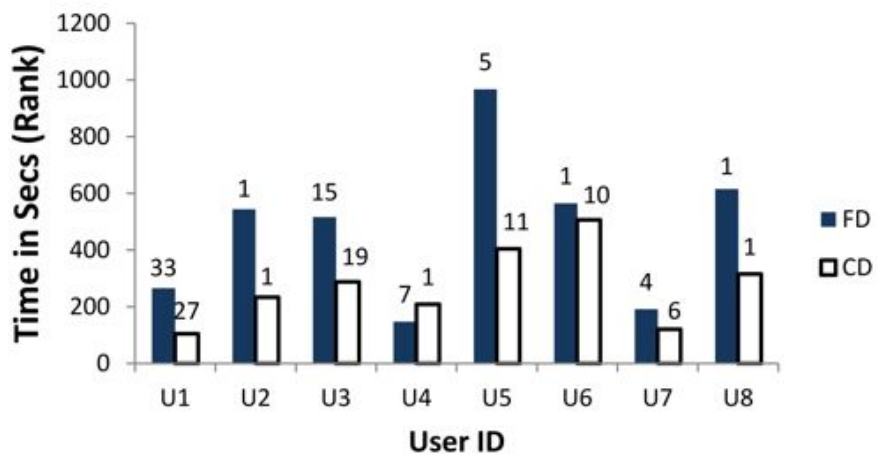
Regression Task



Discriminating Features



Interesting Patterns



- Almost similar or better quality of response for all three tasks
- Users are able to do the tasks 1.5 - 2 times faster
- Less stress due to reduced information

Conclusions

- Scrolling-aware browsing: Introduced the idea of selecting representative tuples to enable variable-speed scrolling through relational data
- Information loss metric: Quantified loss of information incurred due to browsing representative tuples
- Algorithms: Developed and compared five new scrolling based sampling algorithms that minimize information loss
- Interaction constraints: Proposed efficiently computable algorithms that satisfy fast scrolling requirement.