Skimmer: RapidScrolling of RelationalQuery 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
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

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Price</th>
<th>Bed.</th>
<th>Bath.</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 Jan</td>
<td>455K</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>13 Jun</td>
<td>477K</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8 Dec</td>
<td>480K</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6 Feb</td>
<td>505K</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>19 May</td>
<td>555K</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>12 Jan</td>
<td>578K</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16 May</td>
<td>593K</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5 Jun</td>
<td>650K</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
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:

\[
E_{K\text{Medoids}}(P) = \sum_{j=1}^{K} \sum_{p \in C_j} V(p, o_j)
\]

- Best known heuristic solutions: PAM, CLARA and CLARANS.
Local K-Medoid (LKMmed)

\[ D_i = PAM (P_i, K_i) \]

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 B
40,000$

House C
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** \( P \)
  \[ PIL(P) = \sum_{j=1}^{L} \sum_{p \in C_j} V(p, o_i) \]
  where \( L = \sum_{i=1}^{i} K_i \) and \( \sum_{i}^{i-1} K_i < j \leq \sum_{i}^{i} K_i \)

- 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

![Diagram showing historical and current representatives]
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 $O(K^2N)$ 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**
  - \( \text{KCenters} = \text{KMeans} (P_i, K_i) \)
  - \( D_i = \text{NN} (\text{KCenters}, P_i) \)

- **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
- $K_{Centers} = HKMeans (P_i, K_i)$
- $D_i = NN (K_{Centers}, P_i)$

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.
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
HKMed is best followed by TPKMeans and LKMed

HKMeans is almost close to random sampling

Information gain decreases with increasing # dimensions
Summary Recommendations

- Two-Phase K-Means
- Historical K-Medoids
- Two Phase K-Means
- Two Phase K-Means

Page Size

Sampling Rates
Experimental Goals

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

- **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**

### Discriminating Features

<table>
<thead>
<tr>
<th>User ID</th>
<th>Time in Secs (FD)</th>
<th>Time in Secs (CD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>U2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>U3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>U4</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>U5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>U6</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>U7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>U8</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Regression Task

<table>
<thead>
<tr>
<th>User ID</th>
<th>Time in Secs (Dist * 10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>15</td>
</tr>
<tr>
<td>U2</td>
<td>11</td>
</tr>
<tr>
<td>U3</td>
<td>3</td>
</tr>
<tr>
<td>U4</td>
<td>50</td>
</tr>
<tr>
<td>U5</td>
<td>7</td>
</tr>
<tr>
<td>U6</td>
<td>15</td>
</tr>
<tr>
<td>U7</td>
<td>3</td>
</tr>
<tr>
<td>U8</td>
<td>5</td>
</tr>
</tbody>
</table>

### Interesting Patterns

<table>
<thead>
<tr>
<th>User ID</th>
<th>Time in Secs (Rank)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>33</td>
</tr>
<tr>
<td>U2</td>
<td>27</td>
</tr>
<tr>
<td>U3</td>
<td>15</td>
</tr>
<tr>
<td>U4</td>
<td>19</td>
</tr>
<tr>
<td>U5</td>
<td>11</td>
</tr>
<tr>
<td>U6</td>
<td>10</td>
</tr>
<tr>
<td>U7</td>
<td>4</td>
</tr>
<tr>
<td>U8</td>
<td>1</td>
</tr>
</tbody>
</table>
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.