Top-k queries and selecting representative points

Koen van Walstijn and Sepanta Zeighami

April 26, 2016
Outline

Introduction
  Problem definition and top-\(k\) queries

Top-\(k\) query algorithms
  Naive methods
  Fagin’s algorithm
  Threshold algorithm
  No-Random-Access algorithm

Selecting representative points
  Introduction
  Motivating Example
  Top-\(k\) Representative Skyline Points
  \(k\)-regret Operator
  Diversification
  \(r\)-DisC Query
Outline

Introduction
- Problem definition and top-k queries

Top-k query algorithms
- Naive methods
- Fagin’s algorithm
- Threshold algorithm
- No-Random-Access algorithm

Selecting representative points
- Introduction
- Motivating Example
- Top-k Representative Skyline Points
- k-regret Operator
- Diversification
- r-DisC Query
Problem definition

- Rather than the exact result of a query, we might be interested in approximate matches
- We want to return the top-$k$ results that best correspond to certain criteria
Let’s say we are interested in finding the cheapest places to live. We define a *scoring function* as $Price + 10 \times Tuition$ [5]:

**Example**

<table>
<thead>
<tr>
<th>HID</th>
<th>Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lafayette</td>
<td>90,000</td>
</tr>
<tr>
<td>2</td>
<td>W.Lafayette</td>
<td>110,000</td>
</tr>
<tr>
<td>3</td>
<td>Indianapolis</td>
<td>111,000</td>
</tr>
<tr>
<td>4</td>
<td>Kokomo</td>
<td>118,000</td>
</tr>
<tr>
<td>5</td>
<td>Lafayette</td>
<td>125,000</td>
</tr>
<tr>
<td>6</td>
<td>Kokomo</td>
<td>154,000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>......</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>Location</th>
<th>Tuition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indianapolis</td>
<td>3000</td>
</tr>
<tr>
<td>2</td>
<td>W.Lafayette</td>
<td>3500</td>
</tr>
<tr>
<td>3</td>
<td>Lafayette</td>
<td>6000</td>
</tr>
<tr>
<td>4</td>
<td>Lafayette</td>
<td>6200</td>
</tr>
<tr>
<td>5</td>
<td>Indianapolis</td>
<td>7000</td>
</tr>
<tr>
<td>6</td>
<td>Indianapolis</td>
<td>7900</td>
</tr>
<tr>
<td>7</td>
<td>Kokomo</td>
<td>8200</td>
</tr>
<tr>
<td>8</td>
<td>Kokomo</td>
<td>8200</td>
</tr>
</tbody>
</table>

**Join Result**

<table>
<thead>
<tr>
<th>HID</th>
<th>SID</th>
<th>Price + 10 x Tuition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>150000</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>152000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>145000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>141000</td>
</tr>
</tbody>
</table>

The following examples illustrate real-world scenarios where efficient top-$k$ processing is crucial. The examples highlight the importance of adopting efficient top-$k$ processing techniques in traditional database environments.
Assumptions

- The top-k query results depend on the scoring function, which aggregates scores of an object on several dimensions and allows us to rank the objects.
- The first part of this presentation will deal with an explicit scoring function that is *monotone* in the underlying object attributes: \( S(x_1, x_2, \ldots, x_n) \leq S'(x'_1, x'_2, \ldots, x'_n) \) if \( x_i \leq x'_i \) for every \( i \).
- The object attributes are assumed to be stored in *separate, ordered lists*.
Assumptions

- The second part of this presentation will deal with situations where an explicit scoring function is not specified: rather, top-k representative points need to be returned that might be of interest to the user.
Outline

Introduction
Problem definition and top-
queries
Top-
query algorithms
Naive methods
Fagin’s algorithm
Threshold algorithm
No-Random-Access algorithm

Selecting representative points
Introduction
Motivating Example
Top-
Representative
Skyline Points
-regret Operator
Diversification
-
-DisC Query
Straightforward solutions

- One can perform random access to look up the scores for all objects in the different lists, compute the total scores, sort the total scores and return the top-$k$ results.
- Similarly, one can perform a join operation on the lists, compute the total score for each object, sort the new table by total score and return the top-$k$ objects.
- However, these are costly operations when databases get large!
Outline

Introduction
- Problem definition and top-k queries

Top-k query algorithms
- Naive methods
- Fagin’s algorithm
- Threshold algorithm
- No-Random-Access algorithm

Selecting representative points
- Introduction
- Motivating Example
- Top-k Representative Skyline Points
- $k$-regret Operator
- Diversification
- $r$-DisC Query
Steps of the algorithm [4]

1. Access sorted lists in parallel until there are \( k \) objects seen in every list
2. Perform random access to obtain the missing scores for objects not seen in every list
3. Compute the total score for each object and return the top \( k \) objects
Introduction
Top-k query algorithms
Selecting representative points
References

Worked example

1. Access sequentially until $k$ objects are seen in all lists:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>

Koen van Walstijn and Sepanta Zeighami
Top-k queries and selecting representative points

Naive methods
Fagin’s algorithm
Threshold algorithm
No-Random-Access algorithm
### Worked example

1. Access sequentially until $k$ objects are seen in all lists:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td></td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td></td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td></td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td></td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td></td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
</tr>
</tbody>
</table>
Worked example

1. Access sequentially until $k$ objects are seen in all lists:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>
Worked example

1. Access sequentially until $k$ objects are seen in all lists:

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th></th>
<th>$s_2$</th>
<th></th>
<th>$s_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>
## Worked example

2. Perform random access to retrieve missing scores:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>
## Worked example

3. Compute total scores and return top-\(k\) objects:

<table>
<thead>
<tr>
<th></th>
<th>(S_1)</th>
<th>(S_2)</th>
<th>(S_3)</th>
<th>(S_{\text{total}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O_1)</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(O_2)</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(O_4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(O_3)</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(O_5)</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
S_{\text{total}} = \begin{cases} 
10 + 8 + 8 & = 26 \\
8 + 7 + 6 & = 21 \\
4 + 5 + 3 & = 12 \\
0 + 2 + 3 & = 5 \\
0 + 1 + 1 & = 2 \\
\end{cases}
\]

\(O_4\) is the top-\(k\) object with the highest total score.
Advantages and disadvantages

+ Not so much random access: only at the end to find the missing scores
– Large buffer space is required to keep track of all the objects encountered
Outline

Introduction
Problem definition and top-k queries

Top-k query algorithms
Naive methods
Fagin’s algorithm
Threshold algorithm
No-Random-Access algorithm

Selecting representative points
Introduction
Motivating Example
Top-k Representative Skyline Points
k-regret Operator
Diversification
r-DisC Query
Steps of the algorithm [4]

1. Access sorted lists in parallel and after each sequential access:
   - Compute the threshold value as the *scoring function over the last seen object in every list* (example follows)
   - Perform random access to obtain scores of objects seen so far, compute their total scores and keep a buffer of top-$k$ objects so far

2. Stop when all objects in the buffer are above the threshold
Worked example

1: Access sorted lists sequentially, look up scores and compute threshold:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>
## Worked example

1: Access sorted lists sequentially, look up scores and compute threshold:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
<th></th>
<th>$T$</th>
<th></th>
<th>$S_{total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
<td></td>
<td>26</td>
<td></td>
<td>$O_4$ 21</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>$O_1$ 19</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Worked example

1: Access sorted lists sequentially, look up scores and compute threshold:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
<th></th>
<th>$T$</th>
<th></th>
<th>$S_{total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
<td></td>
<td>21</td>
<td></td>
<td>$O_4$</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>$O_1$</td>
</tr>
<tr>
<td>$O_3$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_4$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Worked example

1: Access sorted lists sequentially, look up scores and compute threshold:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
<th>$T$</th>
<th>$S_{total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Worked example

2: Stop when $k$ objects are above threshold and return top-$k$ objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$T$</th>
<th>$S_{total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>$O_5$</td>
<td></td>
<td>$O_4$</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>$O_4$</td>
<td></td>
<td>$O_1$</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>$O_1$</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>$O_2$</td>
<td></td>
<td>$O_1$</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>$O_3$</td>
<td></td>
<td>19</td>
</tr>
</tbody>
</table>
Advantages and disadvantages

- Buffer space is bounded by $k$
- Less objects are accessed than in Fagin’s algorithm: in the latter, when $k$ objects are seen in all lists, their total scores are greater than or equal to the threshold
- Likely to need more random access than Fagin’s algorithm, which only performs random access at the end
Outline

Introduction
Problem definition and top-k queries
Top-k query algorithms
Naive methods
Fagin’s algorithm
Threshold algorithm
No-Random-Access algorithm

Selecting representative points
Introduction
Motivating Example
Top-k Representative Skyline Points
k-regret Operator
Diversification
r-DisC Query
Steps of the algorithm [5]

1. Access sorted lists in parallel and after each sequential access:
   - Compute the lower and upper bounds of the total scores of objects seen so far
   - Maintain a buffer with the updated lower and upper bounds

2. Stop when there are $k$ objects in the buffer with their lower bound higher than the upper bound of all other objects and return these top-$k$ objects
Worked example

1: Access sorted lists sequentially and compute lower and upper bounds:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_3$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_4$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>
## Worked example

1: Access sorted lists sequentially and compute lower and upper bounds:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td></td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td></td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td></td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td></td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$S_{LB}$</th>
<th>$S_{UP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>26</td>
</tr>
<tr>
<td>$O_4$</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>$O_5$</td>
<td>8</td>
<td>26</td>
</tr>
</tbody>
</table>
Worked example

1: Access sorted lists sequentially and compute lower and upper bounds:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_{LB}$</th>
<th>$S_{UP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
</tr>
</tbody>
</table>
## Worked example

1: Access sorted lists sequentially and compute lower and upper bounds:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th></th>
<th>$S_2$</th>
<th></th>
<th>$S_3$</th>
<th></th>
<th>$S_{LB}$</th>
<th>$S_{UP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>10</td>
<td>$O_4$</td>
<td>8</td>
<td>$O_5$</td>
<td>8</td>
<td>$O_4$</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>$O_2$</td>
<td>8</td>
<td>$O_2$</td>
<td>7</td>
<td>$O_4$</td>
<td>6</td>
<td>$O_1$</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>$O_4$</td>
<td>7</td>
<td>$O_1$</td>
<td>4</td>
<td>$O_1$</td>
<td>5</td>
<td>$O_2$</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>$O_3$</td>
<td>5</td>
<td>$O_3$</td>
<td>2</td>
<td>$O_2$</td>
<td>3</td>
<td>$O_5$</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>$O_5$</td>
<td>3</td>
<td>$O_5$</td>
<td>0</td>
<td>$O_3$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Worked example

2: Stop when there are \( k \) objects in the buffer with their \( S_{LB} \) above all other \( S_{UB} \) and return these top-\( k \) objects:

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\text{Object} & S_1 & S_2 & S_3 & S_{LB} & S_{UP} \\
\hline
O_1 & 10 & \text{O_4} & 8 & \text{O_5} & 8 & \text{O_4} & 21 & 21 \\
O_2 & 8 & \text{O_2} & 7 & \text{O_4} & 6 & \text{O_1} & 19 & 19 \\
O_4 & 7 & \text{O_1} & 4 & \text{O_1} & 5 & \text{O_2} & 15 & 16 \\
O_3 & 5 & \text{O_3} & 2 & \text{O_2} & 3 & \text{O_5} & 8 & 16 \\
O_5 & 3 & \text{O_5} & 0 & \text{O_3} & 1 & \text{O_1} & 19 & 19 \\
\end{array}
\]
Advantages and disadvantages

+ No random access required

– Might not report exact scores but instead report a range
Comparison

Fagin’s algorithm:

+ Not so much random access: only at the end to find the missing scores

- Larger buffer space than Threshold algorithm is required

Threshold algorithm:

+ Buffer space is bounded by $k$

+ Less objects are accessed than in Fagin’s algorithm

- Likely to need more random access than Fagin’s algorithm

No-Random-Access algorithm:

+ No random access required

- Might not report exact scores but instead report a range
Introduction

Selection of a few datapoints from a database so that we can:

- Give suggestions to a user
- Provide a summary of the database

What does ”representative” mean?
Outline

Introduction
- Problem definition and top-\(k\) queries

Top-\(k\) query algorithms
- Naive methods
- Fagin’s algorithm
- Threshold algorithm
- No-Random-Access algorithm

Selecting representative points
- Introduction
- Motivating Example
- Top-\(k\) Representative Skyline Points
- \(k\)-regret Operator
- Diversification
- \(r\)-DisC Query
Motivating Example: A hotel booking website

- Different criteria involved in decision making, such as:
  - Price
  - Location
  - Size of the room

- Consider a website that gives suggestions to a user.
- Or, when a user does not know what type of room he or she is exactly looking for.
  - Hence, the result of the user’s query will be a large subset of the database.
  - For instance, all the hotels with the price less than 500HKD per night.

The problem is how to select a few points to show to a user instead of the many points in the database.
Representativeness

- Need to select points that are “representative” of the whole database
- Different ways to define representativeness
Outline

Introduction
  Problem definition and top-k queries
Top-k query algorithms
  Naive methods
  Fagin’s algorithm
  Threshold algorithm
  No-Random-Access algorithm

Selecting representative points
  Introduction
  Motivating Example
  Top-k Representative Skyline Points
  k-regret Operator
  Diversification
  r-DisC Query
Top-k Representative Skyline Points by Lin et al. [6]

Measures **representativeness** of a set $S$ by a measureing $|D(S)|$, where $D(S) =$ the set of points that the points in $S$ dominate.

- **Dominates:** A point, $p$, dominates the point, $q$, if $p$ has a lower value than $q$ in all dimensions
- **Assuming the lower the value, the more desirable a criterion is**

Given a database $D$, top-k RPS returns $k$ data points from the skyline of $D$ that together dominate the maximum possible number of points in the database.

- top-k RPS is NP-hard in more than 2 dimensions.
- A greedy algorithm with the approximation ratio $1 - \frac{1}{e}$ is proposed.
Algorithm for top-$k$ RPS

### Figure: Hotel database

<table>
<thead>
<tr>
<th>id</th>
<th>dist (km)</th>
<th>price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>4</td>
<td>150</td>
</tr>
<tr>
<td>$p_2$</td>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>$p_3$</td>
<td>2.5</td>
<td>240</td>
</tr>
<tr>
<td>$p_4$</td>
<td>2</td>
<td>180</td>
</tr>
<tr>
<td>$p_5$</td>
<td>1.7</td>
<td>270</td>
</tr>
<tr>
<td>$p_6$</td>
<td>1</td>
<td>195</td>
</tr>
<tr>
<td>$p_7$</td>
<td>1.2</td>
<td>210</td>
</tr>
</tbody>
</table>

### Figure: Greedy Algorithm

**Algorithm 1 Greedy ($k, P$)**

**Input:** $k$: an integer; $P$: a set of data points.

**Output:** $k$ skyline points.

**Description:**

1. Compute $S_P$.
2. **Step 2:** $\forall s \in S_P$: compute $D\{s\}$.
3. **Step 3:**
   1. $S_1 := \emptyset$.
   2. **while** $|S| < k$ and $S_P - S \neq \emptyset$ **do**
   3. Choose $s \in S_P - S$ such that $|D\{s\} \cup S|$ is maximized;
   4. $S := \{s\} \cup S$;
5. **return** $S$;
Algorithm for top-k RPS

- Calculation of $D(\{s\})$, $\forall s \in S_p$:
  - $\forall p \in P$, use a query window from origin to $p$, and add $p$ to all the skyline points that dominate it.

- Calculation of $D(\{s\} \cup S)$:
  - Sort $D(\{s\}) \forall s \in S$ according to the first dimension and merge them.
Introduction

Top-k query algorithms
Selecting representative points

References

Motivating Example

Top-k Representative Skyline Points

k-regret Operator
Diversification
r-DisC Query

An example

<table>
<thead>
<tr>
<th>id</th>
<th>dist (km)</th>
<th>price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>4</td>
<td>150</td>
</tr>
<tr>
<td>$p_2$</td>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>$p_3$</td>
<td>2.5</td>
<td>240</td>
</tr>
<tr>
<td>$p_4$</td>
<td>2</td>
<td>180</td>
</tr>
<tr>
<td>$p_5$</td>
<td>1.7</td>
<td>270</td>
</tr>
<tr>
<td>$p_6$</td>
<td>1</td>
<td>195</td>
</tr>
<tr>
<td>$p_7$</td>
<td>1.2</td>
<td>210</td>
</tr>
</tbody>
</table>

(a) Hotels

Calculate the skyline using the branch and bound algorithm.

Calculate $D(\{s\}) \forall s \in S_p = \{p_2, p_4, p_6\}$.

$D(\{p_2\}) = \{p_1\}$, $D(\{p_4\}) = \{p_3\}$, $D(\{p_6\}) = \{p_7, p_5, p_3\}$

For $k = 2$, first iteration selects $p_6$ and second iteration selects $p_2$.

Figure: Hotel database
Reducing the space complexity

- To reduce the space complexity of storing $D(\{s\})$, we can use another counting technique.
- Let $h(p)$ be a hash function and $P$ a set of points.
- $Fm = \{B : |B| = L, \forall 0 \leq j \leq L-1, B[j] = 1 \text{ iff } \exists p \in P, h(p) = j\}$.
- Let $fm(P)$ represent the set of $z$ FM sketches generated over $P$ with independent hash functions.
- We can approximate the size of $P$ by $\frac{1}{\phi}2^z \sum_{i=1}^{z} \frac{\min(Fm_i)}{z}$
- where $\min(B)$ denotes the least bit (from left) of a bitmap $B$ with value 0; if no such bit exists then $\min(B) = L$. 
Reducing the space complexity

- For the sets $D(\{s\}), \forall s \in S_p$ we can create $fm(\{s\})$, the set of FM sketches.
- To calculate $D(\{s\} \cup S)$, we can use bitwise or between the bitmap sketches.
Outline

Introduction
- Problem definition and top-\(k\) queries

Top-\(k\) query algorithms
- Naive methods
- Fagin’s algorithm
- Threshold algorithm
- No-Random-Access algorithm

Selecting representative points
- Introduction
- Motivating Example
- Top-\(k\) Representative Skyline Points
  - \(k\)-regret Operator
- Diversification
- \(r\)-DisC Query
**k-regret Operator by Nanongkai et al. [7]**

- Aims at minimising the ratio between the utility a user gains from the $k$ selected points and the original database.
- Maximum regret ratio (MRR) is defined as:
  
  $$\sup_{f \in F} \frac{\max_{p \in D} f(p) - \max_{p \in S} f(p)}{\max_{p \in D} f(p)}$$

- $k$-regret operator returns a set $S$, $|S| = k$, given a Database $D$ and a set of utility functions $F$, for which the maximum regret ratio is minimised.
- Proven by Chester et al. [1] to be NP-hard.
Algorithm for $k$-regret Operator

<table>
<thead>
<tr>
<th>Car</th>
<th>MPG</th>
<th>HP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toyota Prius</td>
<td>51</td>
<td>134</td>
</tr>
<tr>
<td>Honda Civic Hybrid</td>
<td>40</td>
<td>110</td>
</tr>
<tr>
<td>Ford Fusion</td>
<td>41</td>
<td>191</td>
</tr>
<tr>
<td>Nissan Altima Hybrid</td>
<td>35</td>
<td>198</td>
</tr>
<tr>
<td>Volkswagen Jetta TDI</td>
<td>30</td>
<td>140</td>
</tr>
</tbody>
</table>

Figure: Cars database

Algorithm 2  \text{GREEDY}(D, k)

\begin{enumerate}
\item Input: A set of $d$-dimensional points $D = \{p_1, p_2, ..., p_n\}$ and an integer $k$, the desired output size.
\item Output: A subset of $D$ of size $k$, denoted by $S$.
\begin{enumerate}
\item Let $S = \{p_1\}$ where $p_1^* = \arg \max_{p \in D} p[1]$.
\item In the loop below, we find a point $p \in D \setminus S$ with maximum regret ratio.
\begin{enumerate}
\item for $i=1$ to $k-1$ do
\item Let $r^* = 0$ and $p^* = \text{null}$.
\item for each $p \in D \setminus S$ do
\item Compute $rr_{S \cup \{p\}}(S, L)$
\item if $r^* < rr_{S \cup \{p\}}(S, L)$ then
\item $r^* = rr_{S \cup \{p\}}(S, L)$
\item $p^* = p$
\item end if
\item end for
\item if $r^* = 0$ then return $S$
\item else $S = S \cup \{p^*\}$ endif
\item end for
\item return $S$
\end{enumerate}
\end{enumerate}
\end{enumerate}

Figure: Utility functions

Figure: Greedy Algorithm
An example

<table>
<thead>
<tr>
<th>Car</th>
<th>$f_{(0.2,0.8)}$</th>
<th>$f_{(0.4,0.6)}$</th>
<th>$f_{(0.6,0.4)}$</th>
<th>$f_{(0.8,0.2)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>117.4</td>
<td>100.8</td>
<td>84.2</td>
<td>67.6</td>
</tr>
<tr>
<td>$p_2$</td>
<td>96</td>
<td>82</td>
<td>68</td>
<td>54</td>
</tr>
<tr>
<td>$p_3$</td>
<td>161</td>
<td>131</td>
<td>101</td>
<td>71</td>
</tr>
<tr>
<td>$p_4$</td>
<td>165.4</td>
<td>132.8</td>
<td>100.2</td>
<td>67.6</td>
</tr>
<tr>
<td>$p_5$</td>
<td>118</td>
<td>96</td>
<td>74</td>
<td>52</td>
</tr>
</tbody>
</table>

**Figure**: Utility functions

Let $S = \{p_1, p_2\}$ and $F$ the set of utility functions. Regret ratio, $rr_D(S, \{f_{(0.2,0.8)}\}) = \frac{165.4-117.4}{165.4} = 0.29$

- Linear programming is used to calculate $rr_D(S, L)$, where $L$ is the class of the linear utility functions. (functions of the form $f(p) = \sum_{i=1}^{d} a_i p[i]$)
An example

Using the greedy algorithm to find the solution for $k = 2$:

- First, set $S = p_1$ (maximises the first dimension).
- Add $p_2$ to $S$ and compute $rr_D(S, f), \forall f \in F$. Take their maximum.
- So, we have $rr_D(S, \{f(0.2,0.8)\}) = 0.29$, $rr_D(S, \{f(0.4,0.6)\}) = 0.24$, $rr_D(S, \{f(0.6,0.4)\}) = 0.16$, $rr_D(S, \{f(0.8,0.2)\}) = 0$
- MRR will be $\max(0.29, 0.24, 0.16, 0) = 0.29$.
- Then, we remove $p_2$ from $S$, add $p_3$ and calculate regret ratio again.
- We’ll do the same for all points that are not in $S$ and take the point that results in the lowest MRR.
- In this iteration, we’ll add $p_4$ because the corresponding MRR will be zero.
Outline

Introduction
Problem definition and top-k queries
Top-k query algorithms
Naive methods
Fagin’s algorithm
Threshold algorithm
No-Random-Access algorithm

Selecting representative points
Introduction
Motivating Example
Top-k Representative Skyline Points
k-regret Operator
Diversification
r-DisC Query
Query result diversification [2]

- Return $k$ items as a result of a query that are the most related to the query but as different from each other as possible.
- To avoid homogeneous items in the result.
- For example, a sales person might want to give suggestions to the users on gifts to buy.
Query result diversification cont.

- Measure relevance and dissimilarity:
  - Relevance function: $\delta_{rel}(t, Q)$ measures the relevance of the point $t$ to the query $Q$.
  - Distance function: $\delta_{dis}(t_1, t_2)$ measures the dissimilarity of the points $t_1$ and $t_2$.
- The problem is to return $k$ points to maximise an objective function defined in terms of $\delta_{dis}(t_1, t_2)$ and $\delta_{rel}(t, Q)$.
- One of the introduced objective functions for a set $U$ consisting of $k$ points and a parameter $\lambda$ is:

$$F(U) = (k - 1)(1 - \lambda) \cdot \sum_{t \in U} \delta_{rel}(t, Q) + \lambda \cdot \sum_{t, t' \in U} \delta_{dis}(t, t')$$

- NP-hard problem, but a 2-approximation greedy solution exists.
Query result diversification cont.

- Consider a database consisting of
  - catalog(item, type, price, inStock)
  - history(item, buyer, recipient, gender, age, rel, event, rating)
- A recommender system wants to suggest a user to buy gifts for a 14 years old girl.
- $\delta_{dis}$ can be a function defined using the type of the items the shop has.
  - Two jewellery will score less than a doll and a jewellery
- $\delta_{rel}$ can be a function defined using history.
  - If a gift has been purchased before for a 14 years old girl, it’s more relevant to the query.
- The greedy algorithm will add to the solution set the point whose addition will result in the highest score.
An example

- Assume the shop has items: jewellery, dolls, cutlery, shirt.
- Based on the historic data each of these items have been bought for a 14 years old the following number of times: jewellery = 15, doll = 20, cutlery = 0, shirt = 10.
- We also have the distance function as follows: $\delta_{\text{dis}}$ (jewellery, dolls) = 6, $\delta_{\text{dis}}$ (jewellery, cutlery) = 8, $\delta_{\text{dis}}$ (jewellery, shirt) = 3, $\delta_{\text{dis}}$ (cutlery, dolls) = 7, $\delta_{\text{dis}}$ (shirt, dolls) = 5, $\delta_{\text{dis}}$ (cutlery, shirt) = 8.
- for $U = \{\text{cutlery, dolls}\}$ we can calculate $F(U) = 0.5 \times (0 + 20) + 0.5 \times (7) = 13.5$
Outline

Introduction
Problem definition and top-$k$ queries

Top-$k$ query algorithms
Naive methods
Fagin’s algorithm
Threshold algorithm
No-Random-Access algorithm

Selecting representative points
Introduction
Motivating Example
Top-$k$ Representative Skyline Points
$k$-regret Operator
Diversification
$r$-DisC Query
Result Diversification based on Dissimilarity and Coverage by Drosou and Pitoura [3]

- Return results that are as different to each other as possible and cover the whole database.
- A distance function, \( dist(.,.) \), to measure similarity of two points.
- Neighbourhood of an object is defined as:
  \[
  N_r(p_i) = \{ p_j | p_i \neq p_j, \; dist(p_i, p_j) \leq r \}
  \]
- Select a minimal set \( S \) such that:
  - Union of the neighbourhood of the points in \( S \) cover the whole database
  - No point in \( S \) is in the neighbourhood of any other point in \( S \)
**r-DisC Query**

- We can represent the database by a graph, where points in the neighbourhood of each other are connected by an edge.
- The coverage problem is the same as finding the dominating set in a graph.
- The diversification problem is the same as finding the an independent set in a graph.
- The problem is the same as finding a minimum dominating independent set in a graph.
**r-DisC Query**

- **Black**: selected point
- **Grey**: covered by a black point
- **White**: not grey or black

Let $\Delta$ be the maximum number of neighbours of any object in the database. Algorithm 3 is a $\Delta$-approximation for the problem.

---

**Algorithm 3 Greedy-DisC**

**Input**: A set of objects $\mathcal{P}$ and a radius $r$.

**Output**: An $r$-DisC diverse subset $S$ of $\mathcal{P}$.

1. $S \leftarrow \emptyset$
2. for all $p_i \in \mathcal{P}$ do
   3. color $p_i$ white
4. end for
5. while there exist white objects do
   6. select the white or grey object $p_i$ with the largest $|N^W_r(p_i)|$
   7. $S = S \cup \{p_i\}$
   8. color $p_i$ black
   9. for all $p_j \in N^W_r(p_i)$ do
      10. color $p_j$ grey
11. end for
12. end while
13. return $S$
r-DisC Query
Bibliography I


Bibliography II

