Progressive Join Algorithms Considering User Preference

Mengsu Ding, **Shimin Chen***
Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China

Nantia Makrynioti, Stefan Manegold
CWI (Centrum Wiskunde & Informatica)
Amsterdam, the Netherlands
Preference-aware Progressive Join

- Progressive query processing for exploratory data analysis
- Return join results ordered according to preference

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.9</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.7</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Join keys: L → R

Output:
- (1, 0, 1, 1.4)
- (1, 0, 2, 1.3)
- (2, 0, 1, 1.2)

Preference combining function:
\[ f(x, y) = Ax + By \]

Goal: Fast early results & Fast full results
Problems of Existing Solutions

- **Blocking approach:** Join + Sort
  - No early results

- **Extending top-\(k\) join algorithms:**
  - e.g., RankJoin [VLDB’03] symmetric hash join + priority queue
  - Slow full results
  - Significant sorting overhead

We want to reduce or eliminate sorting overhead.
Our Idea: Exploiting Contour Lines

Preference values of join results

Contour Line
Points with equal preference value

Preference combining function
\( f(x, y) = Ax + By \)

Table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.5</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.3</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.2</td>
<td>0.7</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0.1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Output:

\((1, 0, 1.4)\)
\((1, 0, 2, 1.3)\)
\((2, 0, 1, 1.2)\)
Our Idea: Exploiting Contour Lines

\[ f(x, y) = Ax + By \]
\[ A = B = 1 \]
Inputs Follow Contour Lines (Algorithm 1: CJpI)

Check bound: upper($U_i$) < lower($Buf_{i-1}$)

- Avoid sorting across buffers
- Only need to sort within each buffer
Both Inputs and Outputs Follow Contour Lines (Algorithm2: CJpB)

- Reduce intermediate result size
- But join cost may be higher

Check bound: upper($U_i$) < lower($Buf_i%2$)
Relaxation to Remove Sorting

- **Relaxation of Order**: **no intra-buffer sorting**
  - Relaxation: tuples $t_i$ and $t_j$ are regarded as in order if $|t_i.pval - t_j.pval| \leq \epsilon$
  - Judiciously select input intervals

### 4 Variants of Contour Joins

<table>
<thead>
<tr>
<th>Variants</th>
<th>Follow Contour Lines</th>
<th>Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Join Inputs</td>
<td>Join Results</td>
</tr>
<tr>
<td>CJpI</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CJpB</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CJrI</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CJrB</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

p: precise, r: relaxed;  I: Inputs, B: Both inputs & outputs
Experiments

- **Data Set:** Based on TPC-H Lineitem and Partsupp
  - Preference values based on: \( l\_\text{discount}, ps\_\text{availqty} \)
  - 3 datasets (can fit into main memory)

<table>
<thead>
<tr>
<th>Scale Factor</th>
<th>Datasets</th>
<th>#Inputs</th>
<th>#Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_1 )</td>
<td>datasets#1</td>
<td>fixed</td>
<td>increasing</td>
</tr>
<tr>
<td>( m_2 )</td>
<td>datasets#2</td>
<td>increasing</td>
<td>fixed</td>
</tr>
<tr>
<td>( m_3 )</td>
<td>datasets#3</td>
<td>increasing</td>
<td></td>
</tr>
</tbody>
</table>

- **Query**
  
  \[
  f(x, y) = Ax + By
  \]
  
  ```
  select L.key, \( f(L.pval, PS.pval) \) as score
  from Lineitem as L, Partsupp as PS
  where L.key = PS.key
  order by score ASC progressively
  ```

- **Machine**
  - Intel Core i7-4770 CPU @3.40GHz (8MB cache) and 32GB memory
  - 64-bit Ubuntu 16.04 LTS with 4.15.0-62-generic Linux kernel
Overall Results

- Compared to RJ (RankJoin)
  - 1% early results
    - best precise contour join: up to 7x improvements
    - best relaxed contour join: up to 14x improvements
  - Full results
    - best precise contour join: up to 10.6x improvements
    - best relaxed contour join: up to 39.4x improvements

- Comparable or better performance to JS (blocking JoinSort)
Fix Input Size, Increase Join Result Size

- As join result size increases, the fraction of sorting increases
- RankJoin becomes very poor
- CJrB is the best performing algorithm
Conclusion

• Preference-aware joins in progressive query processing

• Idea: exploit contour lines in the join result space

• ContourJoin: a promising solution
  - Faster early and full results generation (vs. RankJoin)
  - Good total result generation performance (vs. JoinSort)

- More in the paper
  - Algorithms and proofs
  - Extensive experimental results
  - Discussion on preference combining functions, unsorted inputs, multi-way joins