Interval Join based on plane sweep

1. Preprocessing:
   - Each interval -> a start and an end domain point
   - Sort endpoints of each input collection

2. Join:
   - Sweep through lists (as in sort-merge)
   - If a start domain point (e.g. from R) is met, insert interval to active set (e.g. A^s)
   - Scan other active set (e.g. A^t) to produce join results
   - If an end domain point (e.g. from R) is met, delete interval from active set (e.g. A^t)

Example: John's intervals are stored in R, and Mary's intervals are stored in S. We want to find all pairs where John and Mary overlap.

<table>
<thead>
<tr>
<th>Company R</th>
<th>Company S</th>
</tr>
</thead>
<tbody>
<tr>
<td>employee</td>
<td>start</td>
</tr>
<tr>
<td>John</td>
<td>1993</td>
</tr>
<tr>
<td>Mary</td>
<td>1995</td>
</tr>
<tr>
<td>Tom</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>employee</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1994</td>
<td>1996</td>
</tr>
<tr>
<td>Bob</td>
<td>1998</td>
<td>2002</td>
</tr>
<tr>
<td>Hugo</td>
<td>2007</td>
<td>2010</td>
</tr>
<tr>
<td>Helen</td>
<td>2014</td>
<td>2017</td>
</tr>
</tbody>
</table>

Interval Join

Find all pairs of employees whose working periods at companies R and S overlap.

Interval Count Semi-Join

For each employee r from company R, count how many employees from company S worked at a period of time overlapping with r’s employment.

Applications

- Temporal databases
- Selecting and/or ranking objects

Related work

- Plane-sweep based interval joins [1], [2]
- Top-k count semi-joins
  - Relational [3], spatial [4]

Baseline Approaches

Naïve Algorithm:

- Compute interval join
- Sort and aggregate join results

Simple Counting Algorithm:

- Adapt Interval Join algorithm to count ICSJ results instead of producing IJ results

Smart Counting Algorithm

Observation: to compute r.count we need:

- The number |A^s| of active intervals from S when r.start is accessed
- The number of intervals formed S which became active after r.start and before r.end

Approach: use a global counter g to track for how many intervals s ∈ S their start point has been seen so far.

X High cost: join + aggregation

Simple Counting Algorithm:

- Adapt Interval Join algorithm to count ICSJ results instead of producing IJ results

X Cost similar to interval join

X Cost sensitive to join output

Experiments

Setup

- In-memory processing
- Gapless hash map [2]

Datasets

<table>
<thead>
<tr>
<th>FLIGHTS</th>
<th>BOOKS</th>
<th>GREEND</th>
<th>WEKIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardinality</td>
<td>445,827</td>
<td>2,312,602</td>
<td>110,155,441</td>
</tr>
<tr>
<td>Domain duration (secs)</td>
<td>2,759,280</td>
<td>31,507,200</td>
<td>283,356,410</td>
</tr>
<tr>
<td>Shortest interval (secs)</td>
<td>1,261</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Longest interval (secs)</td>
<td>42,301</td>
<td>31,406,400</td>
<td>59,468,008</td>
</tr>
<tr>
<td>Avg. interval duration (secs)</td>
<td>8,790</td>
<td>2,010,320</td>
<td>16</td>
</tr>
<tr>
<td>Distinct domain points</td>
<td>41,975</td>
<td>5,339</td>
<td>182,028,123</td>
</tr>
</tbody>
</table>

Execution time (varying |R|/|S|)

Breakdown (|R| = |S|)

References