Efficient Rank Join with Aggregation Constraints

Min Xie†, Laks V.S. Lakshmanan†, Peter Wood‡
† University of British Columbia
‡ Birkbeck, University of London
Outline

• Introduction
• Aggregation Constraints
• Deterministic Optimization
• Probabilistic Optimization
• Empirical Results
Top-k Query Processing

- Top-k query [Ilyas et al., CSUR’11]
  - Information retrieval, recommender system and etc.
  - Extremely fruitful area with lots of interesting work

- Rank join [Ilyas et al., VLDB’03, Natsev et al., VLDB’01]
  - Well studied top-k operator in the DB community with many applications
    - Multi-criteria selection
    - Information retrieval
    - Data mining
## Rank Join Operator

- Rank join
  - Extremely useful for building preferred packages of items
  - **Travel Planning**: a package of one museum & one restaurant

### Rank Join Operator

<table>
<thead>
<tr>
<th>Museum</th>
<th>Location</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3.5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Location</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Museum.Location = Restaurant.Location} \]

\[ \text{Order By} \]

\[ \text{Museum.Rating + Restaurant.Rating} \]

Keep top-k

Wednesday, 31 August, 11
Aggregation constraints

- Constraints on **attribute values of each join result**
- Extremely common for applications such as travel packages, course recommendations and etc.

---

**Museum**

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>13.5</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>a</td>
<td>15</td>
<td>4.5</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

**Restaurant**

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>50</td>
<td>4.5</td>
</tr>
<tr>
<td>b</td>
<td>20</td>
<td>4.5</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

---

\[ Museum.Location = Restaurant.Location \]

**Order By**

\[ Museum.Rating + Restaurant.Rating \]

**Keep top-k**

Constrained by

\[ Museum.Cost + Restaurant.Cost \leq 50 \]
Review of Existing Rank Join Algorithms

- **Existing algorithms** [Ilyas et al., VLDB’03] [Schnaitter and Polyzotis, PODS’08]
  - **Settings**: Tuples in each table pre-sorted based on the score attribute(s)
  - Threshold-based algorithm
    - Accessing tuples iteratively from each table
    - Determine a upper bound after a new tuple is accessed
    - Stop if the current top-k results of accessed tuples are better than the upperbound

- **Cruxes of the rank join algorithms**
  - **Item accessing strategy** (Round Robin/Adaptive)
  - **Bounding schemes** (Corner Bound/FR(*) Bound)
  - Significantly affect the performance of the underlying rank join algorithms
Review Existing Rank Join Algorithms

• Performance of rank join algorithm
  • Number of items accessed
  • In memory computation cost

• Rank join algorithms with FR(*) bounding scheme is **Instance Optimal** [Schnaitter and Polyzotis, PODS’08]
  • Within a broad class of algorithms, the # of items accessed is always bounded by a constant factor compared with other algorithm

• Instance optimality alone doesn’t guarantee **good overall performance**! [Finger and Polyzotis, SIGMOD’09]
  • In memory computational cost may dominate the cost
Leveraging Existing Rank Join Algorithms

• How to support aggregation constraints?
  • A naive solution: post-filtering
  • Threshold-based algorithm
    • Accessing tuples iteratively from each table
    • Determine a upper bound after a new tuple is accessed
    • Stop if seen top-k results of accessed tuples, which satisfies all aggregation constraints, are better than the upper bound

• How good is this naive algorithm?
  • Instance Optimal! (Proof in the paper)
  • Yet bad empirical performance
    • In memory processing cost is high
Optimization Opportunity (i)

- Number of tuples kept for each relation
  - Museum : 5
  - Restaurant : 4

- Number of *join probes* performed (Round Robin)
  - 20

<table>
<thead>
<tr>
<th>Museum</th>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁: a</td>
<td>13.5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>t₂: a</td>
<td>15</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>t₃: b</td>
<td>10</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>t₄: a</td>
<td>15</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>t₅: b</td>
<td>5</td>
<td>3.5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₆: c</td>
<td>50</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>t₇: b</td>
<td>20</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>t₈: b</td>
<td>10</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>t₉: a</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>t₁₀: a</td>
<td>10</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Constraint**

$SUM(Cost) \leq 20$

Top-2 results

$\{ t₃, t₈ \} : 9$
$\{ t₁, t₉ \} : 8$

Upperbound : 8
Optimization Opportunity (ii)

- Deterministic optimization

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁: a</td>
<td>13.5</td>
<td>5</td>
</tr>
<tr>
<td>t₂: a</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>t₃: b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>t₄: a</td>
<td>15</td>
<td>4.5</td>
</tr>
<tr>
<td>t₅: b</td>
<td>5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₆: c</td>
<td>50</td>
<td>4.5</td>
</tr>
<tr>
<td>t₇: b</td>
<td>20</td>
<td>4.5</td>
</tr>
<tr>
<td>t₈: b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>t₉: a</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>t₁₀: a</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

Constraint

$$\text{SUM}(\text{Cost}) \leq 20$$

Top-2 results

Deterministic tuple pruning can save many unnecessary join probes during the query processing
Outline

- Aggregation Constraints
  - Deterministic Optimization
  - Probabilistic Optimization
  - Empirical Results
Aggregation Constraints

- Aggregation constraint definition
  - Let $A$ be an attribute, $\lambda$ be a constant value, $\theta$ be a comparison operator and $AGG$ be an aggregation function \{MIN, MAX, SUM\}

- Primitive aggregation constraint (PAC)
  \[ pac ::= AGG(A) \theta \lambda \]

- Aggregation constraint (AC)
  \[ ac ::= pac \mid pac \land ac \]

<table>
<thead>
<tr>
<th>Museum</th>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$:</td>
<td>a</td>
<td>13.5</td>
<td>5</td>
</tr>
<tr>
<td>$t_2$:</td>
<td>a</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>$t_3$:</td>
<td>b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_4$:</td>
<td>a</td>
<td>15</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_5$:</td>
<td>b</td>
<td>5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_6$:</td>
<td>c</td>
<td>50</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_7$:</td>
<td>b</td>
<td>20</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_8$:</td>
<td>b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_9$:</td>
<td>a</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$t_{10}$:</td>
<td>a</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

Constraint: $SUM(Cost) \leq 20$

Top-2 results: $\{ t_3, t_8 \}$

\{ $t_1, t_9$ \}
Problem Definition

• **Rank Join with Aggregation Constraints**

  • Given a set of relations $\mathbf{R}$, a join condition $\mathbf{jc}$, a monotonic score function $\mathbf{S}$ and an aggregation constraint $\mathbf{ac}$

  • Find top-k join results which satisfy $\mathbf{ac}$
Outline

• Aggregation Constraints
• Deterministic Optimization
• Probabilistic Optimization
• Empirical Results
Deterministic Optimization (i)

• Basic properties of aggregation constraints
  • When AGG is MIN and $\theta$ is $\geq$, the corresponding PAC can leverage on **direct-pruning**.
  • If a tuple $t$ doesn’t satisfy the PAC, $t$ can be directly pruned
Example (i)

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$: a</td>
<td>13.5</td>
<td>5</td>
</tr>
<tr>
<td>$t_2$: a</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>$t_3$: b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_4$: a</td>
<td>15</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_5$: b</td>
<td>5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_6$: c</td>
<td>50</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_7$: b</td>
<td>20</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_8$: b</td>
<td>10</td>
<td>4.5</td>
</tr>
<tr>
<td>$t_9$: a</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$t_{10}$: a</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

Constraint

$\min(Rating) \geq 4$

Top-2 results
Deterministic Optimization (i)

• Basic properties of aggregation constraints
  • When AGG is MAX and $\theta$ is $\ge$, the corresponding PAC is **monotone**.
    • If a tuple $t$ satisfies the PAC, join results of $t$ with any tuple also satisfy the PAC
  • When AGG is SUM and $\theta$ is $\le$, the corresponding PAC is **anti-monotone**.
    • If a tuple $t$ doesn’t satisfy the PAC, join results of $t$ with any tuple also don’t satisfy the PAC
Deterministic Optimization (i)

- Basic properties of aggregation constraints

<table>
<thead>
<tr>
<th>AGG ( \theta )</th>
<th>( \leq )</th>
<th>( \geq )</th>
<th>=</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>monotone</td>
<td>direct-pruning</td>
<td>monotone after pruning</td>
</tr>
<tr>
<td>MAX</td>
<td>direct-pruning</td>
<td>monotone</td>
<td>monotone after pruning</td>
</tr>
<tr>
<td>SUM</td>
<td>anti-monotone</td>
<td>monotone</td>
<td>c-anti-monotone</td>
</tr>
</tbody>
</table>

Pruning based on investigating each individual tuple
Deterministic Optimization (ii)

- Subsumption-based Pruning (Motivation)

<table>
<thead>
<tr>
<th>Museum</th>
<th>Restaurant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Cost</td>
</tr>
<tr>
<td>t₁: a</td>
<td>13.5</td>
</tr>
<tr>
<td>t₂: a</td>
<td>15</td>
</tr>
<tr>
<td>t₃: b</td>
<td>10</td>
</tr>
<tr>
<td>t₄: a</td>
<td>15</td>
</tr>
<tr>
<td>t₅: b</td>
<td>5</td>
</tr>
</tbody>
</table>

Constraint

\[ \text{SUM}(\text{Cost}) \leq 20 \]

Top-2 results

Pruning based on comparing tuples
Deterministic Optimization (ii)

• **pac-Dominance Relationship**
  - Comparing two tuples w.r.t. a single PAC
  - Given two tuples \( t, t' \) from the same relation \( R \)
  - \( t \) pac-dominates \( t' \) (or \( t \succ_{\text{pac}} t' \)), if
    - for any tuple \( t'' \) which can join with \( t' \) without violating pac
    - \( t'' \) can also join with \( t \) without violating pac
  - For the common scenario where we have one aggregation constraint per attribute
  - **Sufficient** and **necessary** conditions for determining pac-dominance relationship of each possible aggregation constraint
Deterministic Optimization (ii)

• Example

− Consider AGG is SUM, and θ is ≥, t \succeq_{pac} t' iff.
  − t, t' has the same join attribute value
  − Either
    − t satisfies the PAC
    − Or t.A \geq t'.A

• Similar conditions can be derived for other aggregation constraints (details in the paper)

<table>
<thead>
<tr>
<th>Location</th>
<th># of Review</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁: a</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>t₂: a</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>t₃: a</td>
<td>8</td>
<td>4.5</td>
</tr>
<tr>
<td>t₄: a</td>
<td>8</td>
<td>4.5</td>
</tr>
<tr>
<td>t₅: a</td>
<td>5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Quasi-order: reflexive, transitive anti-symmetric

# of Review ≥ 10

Top-1
Deterministic Optimization (ii)

• **Tuple Subsumption**

  • Let $ac = pac_1 \land \ldots \land pac_m$ be the aggregation constraint
  
  • $t$ subsumes $t'$ (or $t \supseteq_t t'$) if
    
    • score of $t$ is larger than or equal to $t'$
    
    • for all $pac$ in $ac$
      
      • $t \supseteq_{pac} t'$
Deterministic Optimization (ii)

- **Theorem 1:**
  - A tuple \( t \) from relation \( R \) can be directly dropped iff. \( t \) is subsumed by at least \( k \) other tuples in \( R \)

- *Small improvement:* after we have found \( k' \) join result which are guaranteed to be the top-\( k' \) results (\( k' < k \))
  - A tuple \( t \) from relation \( R \) can be directly dropped iff. \( t \) is subsumed by at least \( k - k' \) other tuples in \( R \)

- Adaptive subsumption based pruning
Optimized Algorithm for Rank Join with Aggregation Constraints

- Procedure kRJAC

1. Access new items from each relation

2. **Using the basic property of aggregation constraints to prune tuples which are not promising**

3. **Use subsumption based pruning to further prune away unpromising tuples**

4. If a new tuple isn’t pruned, join it with accessed tuples from other relations

5. Update upperbound threshold and check the stopping criteria
Outline

• Aggregation Constraints
• Deterministic Optimization
• Probabilistic Optimization
• Empirical Results
Probabilistic Optimization

• Rank join algorithms with deterministic pruning can save lots of in memory computations

• Can we further speedup the algorithm?

• Utilize a probabilistic procedure inspired by the previous work on probabilistic top-k algorithms [Theobald et al., VLDB’04]

• Don’t need 100% guarantee that the returned top-k results are actual top-k results

• Stop the algorithm once we can guarantee the current top-k results are correct with a certain confidence threshold
Probabilistic Optimization

• Let $ac = pac_1 \land \ldots \land pac_m$ be the aggregation constraint
• Let $jc$ be the join condition
• Given a set $s$ of tuples, consider the join result of $s$
  • The probability of it satisfying $jc$ can be estimated using existing work in RDBMS [Lipton et al., SIGMOD’90], let it be $P_{jc}$
  • For common data distributions such as uniform and exponential, the probability of the join result of $s$ satisfying each $pac$ can also be estimated (details in the paper), let it be $P_{pc}$
Probabilistic Optimization

• Assume all PACs and the join condition are mutually independent

\[ P_{jc \land ac} = P_{jc} \times \prod_{pac \in ac} P_{pc} \]

• Let N be the estimated number of possible join results which are better than the current top-k result [Theobald et al., VLDB’04]

• based on histogram

• The probability of having a future join result which is better than current top-k result can be estimated as

\[ P = 1 - (1 - P_{jc \land ac})^N \]

• We stop the algorithm if \( P \leq \varepsilon \)
Outline

- Aggregation Constraints
- Deterministic Optimization
- Probabilistic Optimization
- Empirical Results
Data Setting

• Consider synthetic two relation datasets
  • For join attribute, the join selectivity fixed at 0.01
  • For other attributes, we consider two settings
    • Uniform attribute value distribution
    • Exponential attribute value distribution
  • Values are normalized to [0,1]
Efficiency Study (Single PAC)

- $\text{SUM}(A) \geq \lambda$, selectivity $10^{-5}$
- Subsumption-based pruning
Efficiency Study (Single PAC)

- \[ \text{SUM}(A) \leq \lambda, \text{ selectivity } 10^{-5} \]
- Anti-monotone & Subsumption-based pruning

![Graphs showing execution time and number of tuples pruned](image)
Efficiency Study (Multiple PACs)

- \( \text{SUM}(A) \geq \lambda, \text{SUM}(B) \geq \lambda, \) overall selectivity \( 10^{-5} \)
Quality of Probabilistic Algorithm

- Often much faster than deterministic algorithm
- The value of the top-k result get from the probabilistic algorithm is very close to the exact top-k result

![Result quality of probabilistic algorithm graph](image)
Related Work

• Aggregation constraints

  • Well studied in the database community [Levy et al., VLDB’94][Ng et al., SIGMOD’98][Pei and Han, KDD’00][Ross et al., TCS’98]

  • Allows users to impose application-specific preferences

  • Optimizes the performance of the underlying algorithms
Related Work

• **Top-k query processing** [Ilyas et al. CSUR'11]

• **Threshold algorithm** [Fagin, PODS'01]

• **Rank Join**

  • **Implemented inside RDBMS engines** [Ilyas et al., SIGMOD'04, Li et al., SIGMOD'05]

  • **Indexing schemes** [Tsaparas et al., ICDE'03]

  • **Many variations** [Martinenghiand and Tagliasacchi, PVLDB'10]
Related Work

- Top-k package recommendation
- Fixed size package recommendation [Angel et al., EDBT'09]
- Flexible size package recommendation [Xie et al., RecSys'10] [Parameswaran et al., TOIS'11]

- The underlying problem is significantly harder
  - Outer join instead of natural/inner join

- Techniques proposed in this work can still be applied to optimize the performance of the algorithm
Conclusion

• Applications: trip planning and curriculum planning
  • Aggregation constrained top-k query processing
• Naive algorithm works yet high memory computation cost
  • Deterministic optimization: tuple pruning
  • Probabilistic optimization
• Future work
  • Consider flexible size package recommendation under the current framework
  • Broader classes of constraints
Thank you.
Backup Slides