Joining Ranked Input In Practice

Ihab F. Ilyas
Purdue University
Joint work with
Walid G. Aref
Purdue University
Ahmed K. Elmagarmid
Hewlett Packard
Motivation

- Almost every application that depends on ranking the results of the user queries will need a way to combine results of multiple rankings:
  - Similarity queries on multiple features.
  - Query by multiple examples.
  - Joining multiple ordered streams.
  - Documents ranking on multiple keywords search.
  - Document search results from multiple search engines.

- Most of these applications are only interested in the top K results to be presented to the user.
Motivation

• We need a ranking operator that takes as an input multiple ranked streams and produce the combined ranking with minimal access to the inputs.

• The only alternative is to consume all the inputs and sort on the computed overall score. Sometimes not even feasible!
The Driving Application

• STEAM: The Continuous Media Streaming Database Project at Purdue  
  *(Demo - ICDE2002)*

• Objectives of STEAM:
  – Build a database system prototype that has the capabilities to manipulate continuous media objects.
  – Allow for flexible querying of these objects.
  – Store media objects inside the database and present the results as output streams from the database.

• Challenge:
  – Modify the different engine components to satisfy the new functionality requirements and the new data types.

• Application:
  – Medical video database application on top of the system.
The Driving Application

- Use the feature approach.
- Many physical features can be extracted.
- Individual features are multi-dimensional.
- Query Types:
  - Single-Feature Queries:
    “get 4 video shots that are most similar in color to a given query image”
  - Multi-Feature Queries:
    “get 4 video shots that are most similar in color, texture and edge histogram to a given query image”
Single-Feature Query

Color Histogram

Edge Histogram

Texture Tamura

Query
Single-Feature Query

• For low to medium dimensionality, use k-Nearest Neighbor (K-NN) algorithm on a high-dimensional index structure.

• For very high dimensional features, sequential scan over the objects.
Multi-Feature Query

- Color Histogram
- Edge Histogram
- Texture Tamura

Query
Multi-Feature Query

A score function is needed to combine features.

– Alternative 1:
  • Compute the score of each object in the database.
  • Sort the results on the combined score!

– Alternative 2:
  • Concatenate features in one single feature vector.
  • Treat it as a single-feature query.
  • Dimensionality effect!
Multi-Feature Query

– Alternative 3:
  • Index on each feature separately.
  • Get the top K objects for each feature separately.
  • Join the results.
  • Not even correct !!

– Alternative 4:
  • Index on each feature separately.
  • Retrieve objects from each index ranked on one feature.
  • Try to combine the score of each ranking.
  • Stop when you have K objects that are “guaranteed” to have the top K total scores.
Theory?

Efficient solutions:

- Fagin’s Algorithm (FA) [JCSS’99]
- Fagin et. al (TA, NRA, CA) [PODS’01]
- Nepal and Ramakrishna (Multi-step) [ICDE’99]
- Natsev et. al (J*) [VLDB’01]
- Güntzer et. al (Quick-Combine, Stream-Combine) [VLDB’00, ITCC’01]
NRA (No-Random-Access)

A(15-19)  D(13-13)

A,10  A,5  D,7
B,4  D,4  C,6
C,3  C,3  E,5
D,2  B,2  B,4
E,1  E,1  A,3

Buffer Queue


k = 2
The J* Algorithm

- Based on the A* search algorithm.
- Handle general joining conditions.
- Transform the problem to a navigation in a graph of nodes aiming to reach a node with a valid join combination
- Have to see all the scores of an object before reporting it!
The J* Algorithm

D,D (9)
C,A (9)
D,C (9)
# Rank Join Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Random Access</th>
<th>Pipelinnable</th>
<th>Join Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA</td>
<td>Yes</td>
<td>No</td>
<td>key</td>
</tr>
<tr>
<td>TA</td>
<td>Yes</td>
<td>No</td>
<td>key</td>
</tr>
<tr>
<td>Multi-step ≈ TA</td>
<td>Yes</td>
<td>No</td>
<td>key</td>
</tr>
<tr>
<td>Quick-Combine ≈ TA</td>
<td>Yes</td>
<td>No</td>
<td>key</td>
</tr>
<tr>
<td>Stream-Combine</td>
<td>No</td>
<td>Yes</td>
<td>key</td>
</tr>
<tr>
<td>NRA</td>
<td>No</td>
<td>No</td>
<td>key</td>
</tr>
<tr>
<td>J*</td>
<td>No</td>
<td>Yes</td>
<td>General</td>
</tr>
</tbody>
</table>
Systems?

Few attempts – No Practical Solution

• Garlic (IBM): FA algorithm
• HERON: Quick-Combine
How to Integrate into Engine?

• Table Functions:

```sql
SELECT Images.name
FROM GlobalRank( Images, Color, Texture, QueryImage.jpg, ScoreFunction, 10) ;
```

• Query Operator
  – Implementation outside the SQL engine → loose efforts of the query optimizer.
  
  – A chance to be shuffled with other operators in the plan for better performance (pushing down predicates and projections)
  
  – In brief, under the optimizer control!
Our Objective

• Realize the NRA-like algorithms as a physical *query operator* to be used practically in current database engines.
  – Perform necessary modifications.
  – Identify several optimization issues.

• Compare the new operator with the J* (operator ready).
The NRA-RJ Operator

- Implements a logical ranking operator RJOIN.
- RJOIN takes $n$ sorted inputs, a join condition and a score function.
- Each input attaches a score to each object in that input.
- The output is the joined stream with an overall score attached to each object computed using the score function.
- The output is sorted on that overall score.
The NRA-RJ Operator

Sorted output

<oid, f(score₁, score₂, ...scoreₙ), ...>

RJOIN

Sorted Input 1
<oid, score₁, ...>

Sorted Input 2
<oid, score₂, ...>

......

Sorted Input n
<oid, scoreₙ, ...>
The NRA-RJ Operator

• Example Inputs:
  – Different views of the same database objects sorted on different criteria: e.g., the feature ranking of video shots w.r.t different features.
  – External sources: e.g., web search results from different search engines.

• Usage covers a wide range of queries in many new applications: similarity queries in multimedia databases, multiple streams processing, etc.
Why Query Operator?

![Diagram of query operator with NRA-RJ and sorted input nodes]
From an Algorithm To a Query Operator
Important Properties

• Incremental
  – So the operator can fit into the Open/Get-Next/Close framework.

• Pipelined
  – If one is going to sort, one is better off sorting once for all the scores.
  – Most of the queries that use the operator are interested in getting the first answer fast.
NRA

- Output stream has no specific grade/score attached → not a valid input.
- Consider all the inputs together → more context, faster termination.
- Less flexible. A multi-way one-layer operator at the leaf-level of the query plan.

NRA-RJ

- Allow inputs to have ranges of score.
- Report output objects with a range from worst to best grade.
- NRA-RJ: Consider some of the inputs at each stage → less context.
- NRA-RJ: More flexible, pipelined special join operator can be integrated well in the query plan.
NRA-RJ

- **Open()**: Open left and right inputs.
- **Close()**: Close left and right inputs.
- **GetNext()**: 

  ```
  If Queue.Top.WorstGrade > Maximum (Best Grades of all other objects)
  Return the tuple.
  LOOP
  Left.GetNext()
  Right.GetNext()
  Compute threshold.
  Update best and worst grades of objects in Queue.
  If Queue.Top.WorstGrade > Maximum (Best Grades of all other objects)
  break
  End Loop
  Remove from Queue
  Return tuple.
  ```
Optimizing NRA-RJ

• The *local ranking* problem

**Solution : A Balancing Factor !**
Optimizing NRA-RJ

Input A  Input B

NRA RJ

n/p  n  m'  m'

Input C

Input A  Input B  Input C
NRA-RJ vs. J*

- Stopping condition

<table>
<thead>
<tr>
<th></th>
<th>A : 10</th>
<th>B : 5</th>
<th>C : 4</th>
<th>D : 3</th>
<th>E : 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>10-14</td>
<td>10-10</td>
<td>4-9</td>
<td>1-10</td>
<td>1-2</td>
</tr>
</tbody>
</table>

NRA-RJ at $d = 2$
NRA-RJ vs. J*

• Space Complexity
  – Worst Case: (important for allocation)
    • NRA-RJ: A queue of \((N)\) objects
    • J*: A queue of \((2N - 1)\) objects
  – Best Case:
    • NRA-RJ: Empty Queue
    • J*: A queue of \((2K)\) objects.
Performance Study

• The Prototype:
  – Begin with Predator (ORDBMS) on top of Shore (SM).
  – Modify/add all necessary components.
    • Incorporate the GiST index structure inside Shore to provide HD indexes.
    • Change the query engine to add the NRA-RJ, the STOP_AFTER and the NN operators.
    • Modify the buffer manager to handle large media segments.
    • Add a stream manager to handle streaming of media In/Out the database.
  – Runs on Sun Enterprise 450 with UltraSparc-II processors running SunOS 5.6.
Empirical Results
NRA-RJ is sensitive to input ordering.

<table>
<thead>
<tr>
<th>K</th>
<th>NRA-RJ</th>
<th>NRA</th>
<th>J*</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3.692383</td>
<td>2.699707</td>
<td>4.881836</td>
</tr>
<tr>
<td>10</td>
<td>12.87939</td>
<td>9.37793</td>
<td>10.77881</td>
</tr>
<tr>
<td>15</td>
<td>17.28613</td>
<td>13.97217</td>
<td>15.12305</td>
</tr>
<tr>
<td>20</td>
<td>19.15967</td>
<td>15.3457</td>
<td>16.49658</td>
</tr>
<tr>
<td>25</td>
<td>20.65723</td>
<td>16.62451</td>
<td>17.71289</td>
</tr>
<tr>
<td>30</td>
<td>23.05957</td>
<td>19.27686</td>
<td>20.36523</td>
</tr>
<tr>
<td>35</td>
<td>23.83545</td>
<td>20.27148</td>
<td>21.39111</td>
</tr>
<tr>
<td>40</td>
<td>25.38232</td>
<td>21.78711</td>
<td>22.87549</td>
</tr>
<tr>
<td>45</td>
<td>26.87256</td>
<td>23.25537</td>
<td>25.11572</td>
</tr>
<tr>
<td>50</td>
<td>28.7417</td>
<td>25.71826</td>
<td>27.45361</td>
</tr>
<tr>
<td>55</td>
<td>29.88037</td>
<td>27.04443</td>
<td>28.93604</td>
</tr>
<tr>
<td>60</td>
<td>30.51514</td>
<td>27.9917</td>
<td>29.72705</td>
</tr>
<tr>
<td>65</td>
<td>31.16211</td>
<td>28.60742</td>
<td>30.37402</td>
</tr>
<tr>
<td>70</td>
<td>32.28662</td>
<td>30.07568</td>
<td>31.77979</td>
</tr>
<tr>
<td>80</td>
<td>33.29834</td>
<td>32.1123</td>
<td>33.78516</td>
</tr>
</tbody>
</table>
**Input Ordering**

J* is less sensitive to input ordering
Choosing the input ordering is an important optimizer decision.
NRA-RJ vs. NRA vs. J*

Graphs showing CPU Time, Page Accesses, and Buffer Size for NRA-RJ, NRA, and J*.
Conclusion

• NRA-RJ is a practical binary pipelined query operator.
• Can be adopted by real database engine as a new join operator.
• For equi-join on key attributes the proposed NRA-RJ with the local-ranking minimization is the best solution.
• For general join conditions, the J* is the only rank-join algorithm proposed.
• Choosing the input order and the balancing factor are important optimizer decisions.
Future Work

• Heuristics to choose a “good” input order. Sampling on input or consuming the first k tuples are possible approaches.

• How to correctly cost the NRA-RJ operator.

• Adaptive NRA-RJ. What happens if one of the streams stopped or became very slow?