Nearest Neighbor Join with Groups and Predicates

F. Cafagna, M. Boehlen,
Department of Computer Science
University of Zürich

A. Bracher
Agroscope
Switzerland

DOLAP, Melbourne
23.10.2015
Table of Contents

- The Problem
- Related Work: Indexing, Sorting, Segmenting
- The Robust NNJ
- Analytical Evaluation
- Experimental Evaluation
The Problem

Input:
- **Outer Relation** $R$
  - Table $G \times T$
  - Example tuple $r_0$: Pea 1

- **Fact Table** $S$
  - Table $G \times T \times P$
  - Example tuple $s_5$: Soy 3 CP

Query:
- For computing the *Gross Energy*, the CP value is needed. What is the *Crude Protein* value for each tuple in $R$?

Output:
- Table $G_R \times G_S \times T_R \times T_S \times P$
  - Values for Pea and Soy in CP columns
Query: What is the *Crude Protein* value for each row in $R$?

$$R \bowtie^T [G, P = 'CP'] S$$

Result: For each $r \in R$, find the tuple(s) in $S$ that:

1. minimize the *distance* with $r.T$ (closest in time)
2. have the same *group* as $r.G$ (same feed)
3. satisfy a *predicate* $\theta$, e.g., $P = 'CP'$ (nutrient is Crude Protein)
Related Work (1): Indexing

Yao, B., Li, F., Kumar, P.: *K nearest neighbor queries and knn-joins in large relational databases (almost) for free.* ICDE 2010

- Implements a NNJ using an index on similarity attribute $T$
- Helps to efficiently find the nearest measurement in time
- With a predicate (e.g., $P = \text{dOM}$) we get many false hits

Example:

- **Query:** Nearest measurement for $(\text{Soy}, 5)$ where $P = \text{dOM}$?
- **Problem:** The nearest measurement to $(\text{Soy}, 5)$ may not be $\text{dOM}$
- **How bad?** Only 190 among 1M measurements store a $\text{dOM}$ value! $\Rightarrow$ 5000 useless measurements are fetched before finding a $\text{dOM}$ value
Related Work (2): Segmenting & Sorting

Galindo-Legaria, Joshi: *Orthogonal optimization of subqueries and aggregation*. SIGMOD 2001

Silva, Aref, Ali: *The similarity join database operator*. ICDE 2010

- SegmentApply generalizes an operator w/o groups.
- For each group in $R$:
  1. fetch from $R$ and $S$ the tuples of that group
  2. apply a SortMerge NNJ to those tuples
- Multiple accesses to the fact table cause redundant fetches

Example:
- How: compute a NNJ for Pea, then a NNJ for Soy
- Problem: Blocks 1, 2, 3, 6, and 8 are fetched twice!
Robust NNJ Query Tree (1)

- **Problem:** Searching the entire fact table in the Swiss Feed Data Warehouse takes \( \approx 1 \) min (Agroscope has lots of data :-)

- **Goal:** Minimize the number of accesses to the fact table.

- **Idea:** Use the groups of the query to limit the fact table to its relevant portions.

- **How:** Make **at most one** access to the fact table and eliminate tuples with unneeded groups.

- **Example:** It is useless to access tuples with groups \( \neq \{Pea, Soy\} \).

\[ R \begin{array}{ccc}
1 & 5 & 5 \\
\end{array} \]

\[ S \begin{array}{ccccc}
& cp & k & na & & \\
1 & 9 & 6 & & \\
& ca & k & & & \\
4 & 8 & & & \\
& om & vit.C & & & \\
3 & 5 & & & \\
& cp & k & i & & \\
6 & 1 & 7 & & \\
\end{array} \]
Robust NNJ Query Tree (2)

- **Problem:** Processing groups separately causes redundant fetches

- **Idea:** Sorting one big set of tuples w/o redundancies is more efficient than sorting many small sets w redundancies.

- **How:** After the relevant portions are fetched, our roNNJ:
  1. Sorts the tuples by $G$ and $T$ $\Rightarrow$ Tuples are clustered
  2. Compute the Join with one scan of the two inputs

- **Example:** Each block is accessed only once

Input:

$$R = \begin{array}{c}
5 & 6 & 2
\end{array}$$

$$S = \begin{array}{cccc}
cp & k & na & \text{om} & \text{vit.C} & cp & k & i \\
1 & 9 & 5 & 3 & 3 & 6 & 1 & 7
\end{array}$$

Merge:

$$\sigma_{G \in \pi_G(R)} S$$

$$R = \begin{array}{c}
5 & 6 & 2
\end{array}$$

$$S = \begin{array}{cccccccc}
1 & 3 & 4 & 6 & 7 & 9 & 1 & 3 & 5 & 8
\end{array}$$

Francesco Cafagna - University of Zurich - DOLAP '15
Robust NNJ Query Tree (3)

- **Problem:** Useless tuples are fetched multiple times.
- **Goal:** Avoid useless backtracking, i.e., tuples that are not Nearest Neighbours should never be refetched.
- **How:** Always eliminate tuples not satisfying predicate $\theta$
  - after the selection on the groups
    $\Rightarrow$ the Query Tree can use an index on $G$
  - before the SortMerge
    $\Rightarrow$ the nearest neighbours of $r_i$ will all be adjacent in $S$.

- **Example:** Eliminate tuples with $P \neq 'cp'$ before the SM
  $\Rightarrow$ for $(Pea, 5)$ only the nearest neighb. are refetched in $S$.

$R$

$\sigma_{G \in \pi_G(R)}S$

Francesco Cafagna - University of Zurich - DOLAP '15
Robust NNJ Query Tree (4)

This Query Tree uses the left subtree (relation $R$) to limit the right subtree (the fact table $S$) to its relevant portions.

1. The left branch fetches and sorts $R$ by $G, T$
2. In the right branch, the fact table $S$ is accessed once: make one scan of the FactTable and keep only its relevant tuples, i.e., the tuples with the groups of $R$
3. evaluate $\theta$ after the selection on the group (otherwise an index on $G$ is not useful) but before the join (so that no useless backtracking is computed in the Join)
4. Sort the relevant portion by $G, T$ and compute the NNJ at the top.
Analysis

The critical parameters for a NNJ query tree are:

1. the predicate selectivity
2. the group selectivity

Why?

- A low predicate selectivity implies (many) False Hits
  \textbf{Example:} \textit{Nutrient} = \textquote{cp} \Rightarrow 250 \text{ False hits per } R \text{ tuple}

- A high group selectivity implies (many) redundant fetches
  \textbf{Example:} \pi_G(R) = \{Soy, Pea, Beans, \ldots\} (i.e., all Legume seeds)
    \Rightarrow \text{ each block redundantly fetched up to 20 times}
Predicate Selectivity

Let: $\Theta$ be the *predicate selectivity*, i.e., % of tuples satisfying predicate $\theta$

**Example:**
- Tuples satisfying $\theta \equiv P = 'cp'$ are:
  $\{ (Pea, 1), (Pea, 3), (Pea, 7), (Soy, 3), (Soy, 5) \}$
- Predicate selectivity $\Theta = \frac{5}{10} = 0.5$

**Swiss Feed DW:** $P = 'cp'$ $\Rightarrow$ $\Theta = 0.004$
Group Selectivity

Let: \( \Psi = \frac{|\Pi_G(R) \cap \Pi_G(S)|}{|\Pi_G(S)|} \) be the group selectivity, i.e., % groups that are relevant

Example: Grouping Attribute: \( G \)

Relevant groups = \{Pea, Soy\}

Group Selectivity \( \Psi = \frac{2}{4} = 0.5 \)

Swiss Feed DW: \( \pi_G(R) = \{Soy, Pea, Beans, \ldots\} \Rightarrow \Psi = 3\% \)
Evaluation & Experiments (1)

Predicate Selectivity $\Theta$, and Group Selectivity $\Psi$

- **Costs to FactTable** $S = \text{Fetch}(S) + \text{Sort(Relevant Portion)}$
  
  $$= |S| + (\Theta \Psi |S|) \log(\Theta \Psi |S|)$$

- $\Theta$ and $\Psi$ effectively reduce the amount of data to sort

Example: - Nutrient = CP $\Rightarrow \Theta = 0.4$
  - $\pi_G(R) = \{\text{Soy, Pea, Beans, ...} \} \Rightarrow \Psi = 3\%$

<table>
<thead>
<tr>
<th>$S$</th>
<th>$\Theta$</th>
<th>$\Psi$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>1</td>
<td>1</td>
<td>26 sec</td>
</tr>
<tr>
<td>10M</td>
<td>1</td>
<td>1</td>
<td>11 min</td>
</tr>
<tr>
<td>90M</td>
<td>1</td>
<td>1</td>
<td>1 h</td>
</tr>
<tr>
<td>90M</td>
<td>0.004</td>
<td>0.03</td>
<td>0.8 sec</td>
</tr>
</tbody>
</table>

Francesco Cafagna - University of Zurich - DOLAP '15
Setting: $|S| = 90M; \text{Predicate Selectivity} = 0.4\%; \text{Group Selectivity} = 3\%$

When the # of relevant tuples among the 90M of the fact table grows:

- # of index false hits of the **B-Tree** stays the same (i.e., $\frac{1}{0.004}$)

- The # of blocks that **SegApply** fetches redundantly grows ($> 900k$ tuples Sorting on Disk)

- **roNNJ** fetches and sorts $S$ once (always Sorting on Disk but no redundant fetches)
Evaluation & Experiments (3)

Settings: Group Selectivity = 3%

When the predicate gets more selective:

- # of index false hits of the B-Tree grows
- for a predicate (almost) never true, checking all possible combinations (i.e., Antijoin) is cheaper than computing false hits
- SegApply and roNNJ sort less tuples and improve
Evaluation & Experiments (4)

Setting: Predicate Selectivity = 0.4%

When the # of involved groups grows:

- # index false hits of the B-Tree stays the same but caching is less effecting for the false hits

- SegApply computes more NNJs ⇒ higher probability of redundant fetches

- # tuples for roNNJ to sort increases but no redundant fetches or false hits.
Conclusions & Future Work

- We have proposed the Nearest Neighbour Join with Groups and Predicates.
- Its Query Tree uses the groups of the left subtree to limit the portion of the right subtree (i.e., the fact table) to fetch.
- Its evaluation Algorithm, roNNJ, is independent on the physical organization of the fact table.
- It does not suffer from index false hits or redundant fetches.

Future Work:
NNJ with similarity on timestamps with different granularities (e.g., instead of a complete date, only the month or season may be available).
Nearest Neighbor Join with Groups and Predicates

F. Cafagna, M. Boehlen, A. Bracher
Department of Computer Science
University of Zürich
Agroscope
Switzerland

DOLAP, Melbourne
23.10.2015