Query Optimization
Exercise Session 8

Andrey Gubichev

December 8, 2014
Plan for today

- Two heuristics: Iterative DP, Quick Pick
- Meta-heuristics
Iterative DP

- Create all join trees with size up to \( k \), get the cheapest one
- Replace the cheapest tree with the compound relation, start all over again
Iterative Dynamic Programming
Iterative Dynamic Programming

\[
\begin{array}{ccccccc}
R_1 & R_3 & R_4 & R_5 & R_6 & R_7 & R_8 \\
30 & 30 & 20 & 10 & 100 & 15 & 100 \\
0.16 & 0.20 & 0.34 & 0.25 & 0.25 & 0.33 & 0.33 \\
150 & 300 & 30 & 150 & 200 & 6 & 500 \\
\end{array}
\]
Quick Pick

- \( Trees = \{ R_1, \ldots, R_n \}, \ Edge = \text{list of edges} \)
- pick a random edge \( e \in \text{Edges} \) that connects two trees in \( Trees \)
- exclude two selected trees from \( Trees \), add the new tree to
  \( Trees, \ Edge = \text{Edges} \setminus \{e\} \)
- repeat until the complete join tree is constructed

Question for the homework: How to check that an edge connects two trees? what data structures to use?
Metaheuristics
Iterative Improvement

- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

⇒ You’ll do a homework exercise on this.

- Rules for left-deep trees: \textit{swap} and \textit{3cycle}
- Rules for bushy trees: commutativity, associativity, left/right join exchange

Simulated Annealing

- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an “equilibrium”:
Iterative Improvement

- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

⇒ You’ll do a homework exercise on this.
- Rules for left-deep trees: *swap* and *3cycle*
- Rules for bushy trees: commutativity, associativity, left/right join exchange

Simulated Annealing

- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an “equilibrium”:
  - # iterations = # relations
Iterative Improvement
- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

⇒ You’ll do a homework exercise on this.
- Rules for left-deep trees: swap and 3cycle
- Rules for bushy trees: commutativity, associativity, left/right join exchange

Simulated Annealing
- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an “equilibrium”:
  - # iterations = # relations
  - # iterations = 16 × # relations
Iterative Improvement

- Get pseudo-random join tree
- Improve with random operation until local minimum is found
- If this yields a cheaper tree than previously known, keep it, else throw it away

⇒ You’ll do a homework exercise on this.

- Rules for left-deep trees: swap and 3cycle
- Rules for bushy trees: commutativity, associativity, left/right join exchange

Simulated Annealing

- Similar to II, but may keep worse tree (with decreasing probability) to escape local minimum
- Parameter tuning is a nightmare. Consider the following proposals for an “equilibrium”:
  - # iterations = # relations
  - # iterations = 16 × # relations
  - “Would you bet your business on these numbers?”
Possible transformations

- **Swap** $A \Join B \rightarrow B \Join A$
- **3Cycle** $A \Join (B \Join C) \rightarrow C \Join (A \Join B)$ (if possible)
- **Associativity** $(A \Join B) \Join C \rightarrow A \Join (B \Join C)$
- **Left Join exchange** $(A \Join B) \Join C \rightarrow (A \Join C) \Join B$
- **Right Join exchange** $A \Join (B \Join C) \rightarrow B \Join (A \Join C)$
Iterative Improvement

- left deep trees only
  (commutativity for base relations, 3Cycle)
- cost function: $C_{out}$

<table>
<thead>
<tr>
<th></th>
<th>$R_1 R_2$</th>
<th>$R_2 R_3$</th>
<th>$R_3 R_4$</th>
<th>$R_1 R_4$</th>
<th>$R_1 R_2 R_3$</th>
<th>$R_2 R_3 R_4$</th>
<th>$R_1 R_3 R_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2500</td>
<td>40</td>
<td>2000</td>
<td>100</td>
<td>2000</td>
<td>400</td>
<td>2000</td>
</tr>
</tbody>
</table>
Tabu Search

- In each step, take cheapest neighbor\(^1\) (even if more expensive than current)
- Avoid cycles by keeping visited trees in a tabu-set

\(^1\)i.e. join tree that can be produced with a single transformation
Genetic Algorithms

Big picture
- Create a “population”, i.e. create $p$ random join trees
- Encode them using ordered list or ordinal number encoding
- Create the next generation
  - Randomly mutate some members (e.g. exchange two relations)
  - Pairs members of the population and create “crossovers”
- Select the best, kill the rest

Details
- Encodings
- Crossovers
Encoding

Ordered lists
- Simple
- Left-deep trees: Straight-forward
- Bushy trees: Label edges in join-graph, encode the processing tree just like the execution engine will evaluate it

Ordinal numbers
- Are slightly more complex
- Manipulate a list of relations (careful: indexes are 1-based)
  - Left-deep trees: $$(((R_1 \bowtie R_4) \bowtie R_3) \bowtie R_2) \bowtie R_5$$
  - Bushy trees: $$((R_3 \bowtie (R_1 \bowtie R_2)) \bowtie (R_4 \bowtie R_5))$$
Ordered lists

- Simple
- Left-deep trees: Straight-forward
- Bushy trees: Label edges in join-graph, encode the processing tree just like the execution engine will evaluate it

Ordinal numbers

- Are slightly more complex
- Manipulate a list of relations (careful: indexes are 1-based)
- Left-deep trees: \(((R_1 \Join R_4) \Join R_3) \Join R_2) \Join R_5 \mapsto 13211
- Bushy trees: \((R_3 \Join (R_1 \Join R_2)) \Join (R_4 \Join R_5)\)
Encoding

Ordered lists

- Simple
- Left-deep trees: Straight-forward
- Bushy trees: Label edges in join-graph, encode the processing tree just like the execution engine will evaluate it

Ordinal numbers

- Are slightly more complex
- Manipulate a list of relations (careful: indexes are 1-based)
- Left-deep trees: \(((R_1 \Join R_4) \Join R_3) \Join R_2) \Join R_5 \mapsto 13211
- Bushy trees: \((R_3 \Join (R_1 \Join R_2)) \Join (R_4 \Join R_5) \mapsto 12\ 21\ 23\ 12\)
Crossover

Subsequence exchange for ordered list encoding
- Select subsequence in parent 1, e.g. \textit{abcdefgh}
- Reorder subsequence according to the order in parent 2

Subsequence exchange for ordinal number encoding
- Swap two sequences of same length and same offset
- What if we get duplicates?

Subset exchange for ordered list encoding
- Find random subsequences in both parents that have the same length and contain the same relations
- Exchange them to create two children
Quick Pick, Genetic Algorithm
Submit exercises to Andrey.Gubichev@in.tum.de
Due December 15, 2014.