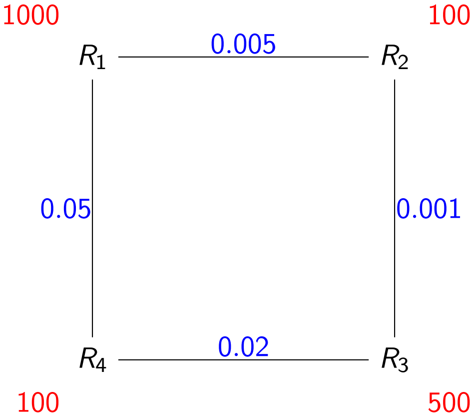
# Query Optimization

## Exercise Session 4

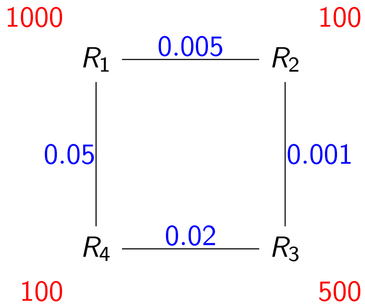
Bernhard Radke

November 28, 2016

# MVP



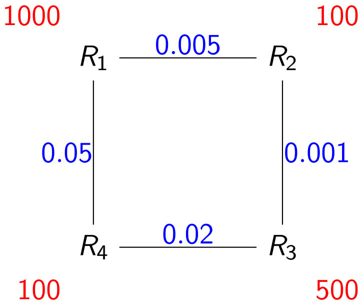
# MVP

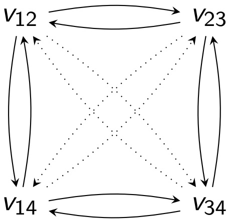



Query graph to Weighted Directed Join Graph:

- nodes = joins
- physical edges between "adjacent" joins (share one relation)
- virtual edges - everywhere else
- WDJG is a clique

# MVP

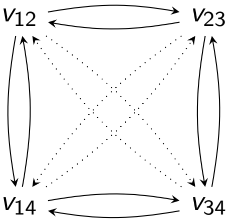


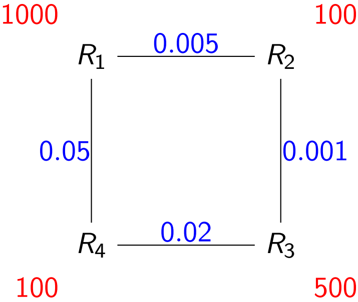
# MVP

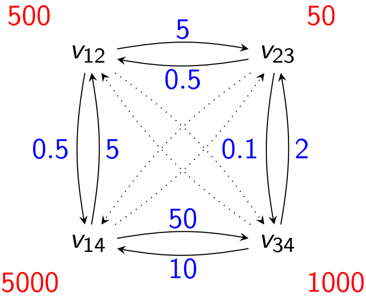
Annotations:

- edge weight $w_{u,v} = \frac{|\bowtie_u|}{|u \sqcap v|}$
- the cost of a node = the cost of the join $C_{out}$



# MVP


1000
100
$R_1$
0.005
$R_2$
0.05
0.001
$R_4$
0.02
$R_3$
100
500


500
50
5
$v_{12}$
$v_{23}$
0.5
0.5
5
0.1
2
50
$v_{14}$
$v_{34}$
10
5000
1000

# Effective spanning tree (informally)

ESP corresponds to an "effective" execution plan (no extra joins). Three conditions:

1. $T$ is binary
2. For every non-leaf node $v_i$, for every edge $v_j \rightarrow v_i$ there is a common base relation between $v_i$ and the subtree with the root $v_j$
3. For every node $v_i = R \bowtie S$ with two incoming edges $v_k \rightarrow v_i$ and $v_j \rightarrow v_i$
   - 3.1 $R$ or $S$ can be present at most in one of the subtrees $v_k$ or $v_j$
   - 3.2 unless the subtree $v_j$ (or $v_k$) contains both $R$ and $S$

# MVP (informally)

Construct an effective spanning tree in two steps:

Step 1 (**Choose an edge to reduce the cost of an expensive operation**)
Start with the most expensive node, find the incoming edge that can reduce the cost the most. Update the cost of the node. Add the edge to the ESP, check the conditions. Repeat until

- no more edges can reduce any cost
- no more join nodes to consider

# MVP (informally)

Construct an effective spanning tree in two steps:

Step 1 (**Choose an edge to reduce the cost of an expensive operation**)
Start with the most expensive node, find the incoming edge that can reduce the cost the most. Update the cost of the node. Add the edge to the ESP, check the conditions. Repeat until

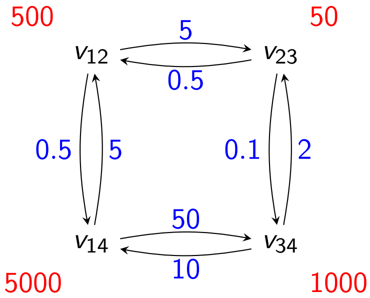
- no more edges can reduce any cost
- no more join nodes to consider

Step 2 (**Find edges causing minimum increase to the result of joins**)
Similar to Step 1, but start with the cheapest node.

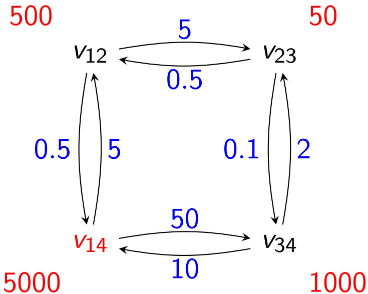
# MVP - example

We start with a graph without virtual edges.
Two cost lists:

- for the Step 1: $Q_1 = v_{14}, v_{34}, v_{12}, v_{23}$
- for the Step 2: $Q_2 = \emptyset$

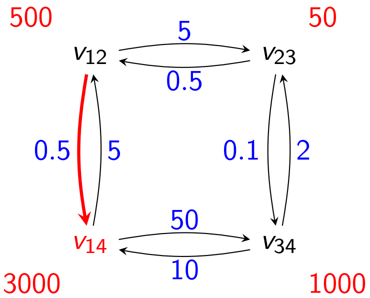


# MVP - example



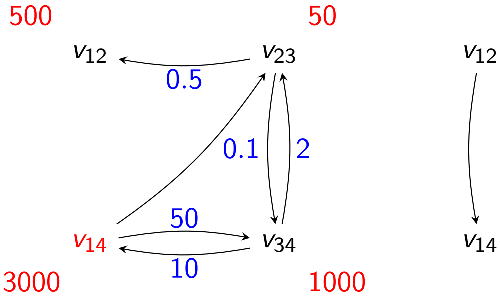

Start with $v_{14}$,

# MVP - example




Start with $v_{14}$, select the edge $v_{12} \rightarrow v_{14}$.
After $v_{12}$ is executed, $|R_1 \bowtie R_2| = 500$
We replace $R_1$ by $R_1 \bowtie R_2$ in $v_{14} = R_1 \bowtie R_4$:
$v_{14} = (R_1 \bowtie R_2) \bowtie R_4$
$cost(v_{14}) = 500 * 100 * 0.05 + 500 = 3000$

# MVP - example




Add edge to EST.

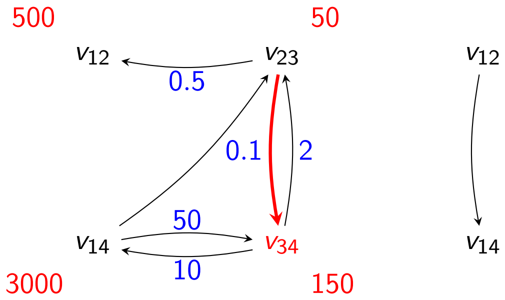
Add new edge $v_{14} \to v_{23}$.

Consider $v_{14}$, no incoming edge with weight $< 1$:

$Q_1 = v_{34}, v_{12}, v_{23}$.

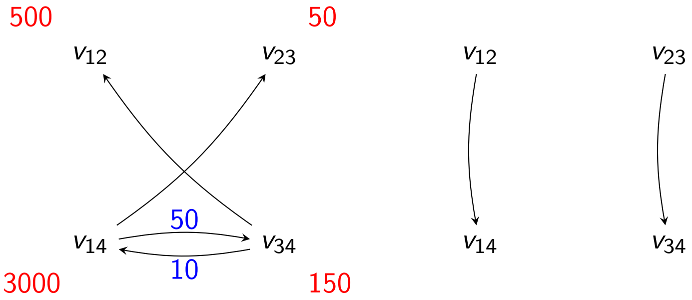
$Q_2 = v_{14}$

# MVP - example




Consider $v_{34}$, one incoming edge with weight $< 1$:
Recompute cost: $cost(v_{34}) = 50 * 100 * 0.02 + 50 = 150$
$Q_1 = v_{12}, v_{34}, v_{23}$.
$Q_2 = v_{14}$

# MVP - example

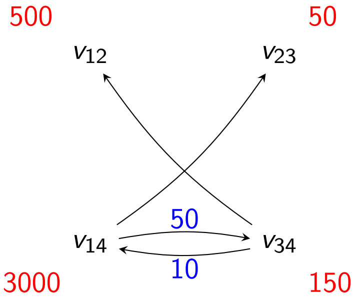


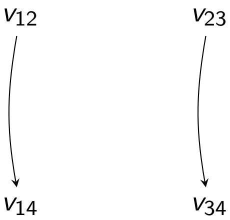

Remove edges, add to EST.

$Q_1 = v_{12}, v_{34}, v_{23}$.

$Q_2 = v_{14}$

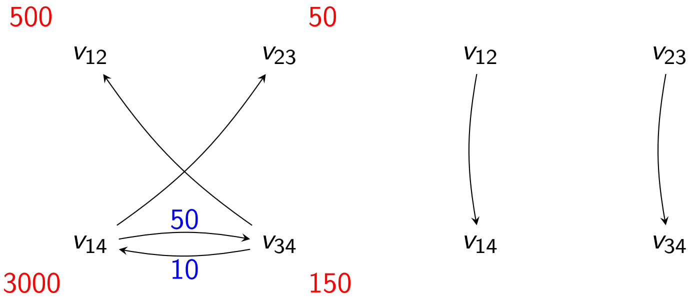
# MVP - example




$v_{12}$: no incoming edge with the weight $< 1$
$Q_1 = v_{34}, v_{23}$.
$Q_2 = v_{12}, v_{14}$

# MVP - example



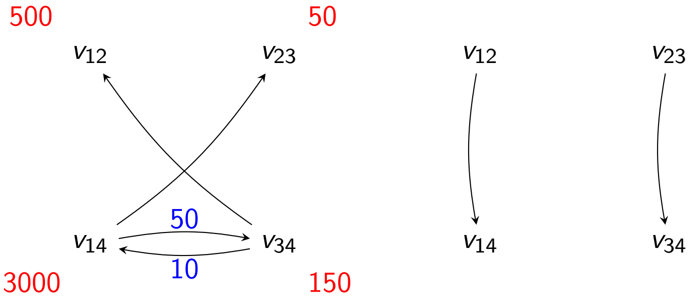

$v_{34}$,$v_{23}$: no incoming edges with the weights $> 1$

$Q_1 = \emptyset$.

$Q_2 = v_{23}, v_{34}, v_{12}, v_{14}$

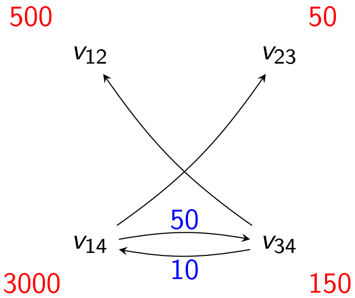
End of Step 1.

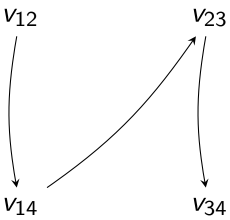
# MVP - example



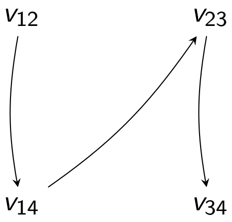

Step 2: try to increase the cost of the EST as little as possible.
$v_{23}$: one incoming edge, does not violate the EST conditions. Add it and stop.

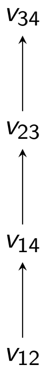
# MVP - example



# MVP - example



# See also

C.Lee, C.Shih and Y.Chen. *Optimizing large join queries using a graph-based approach*. In *IEEE Transactions on Knowledge and Data Engineering*, 2001.

# Overview Dynamic Programming Strategy

- generate optimal join trees bottom up
- start from optimal join trees of size one (relations)
- build larger join trees by (re-)using those of smaller sizes

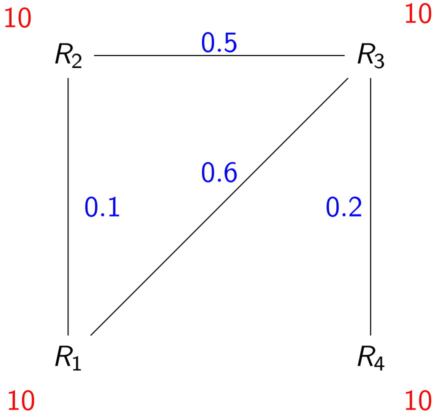
# DP: Generating Linear Trees

DPsizeLinear($R$)
**Input:** a set of relations $R = \{R_1, \ldots, R_n\}$ to be joined
**Output:** an optimal left-deep (right-deep, zig-zag) join tree
$B$ = an empty DP table $2^R \rightarrow$ join tree
**for each** $R_i \in R$
  $B[\{R_i\}] = R_i$
**for each** $1 < s \leq n$ **ascending** {
  **for each** $S \subset R, R_i \in R : |S| = s - 1 \wedge R_i \notin S$ {
    **if** $\neg$cross products $\wedge \neg S$ connected to $R_i$ **continue**
    $p_1 = B[S], p_2 = B[\{R_i\}]$
    **if** $p_1 = \epsilon$ **continue**
    $P$ = CreateJoinTree($p_1, p_2$);
    **if** $B[S \cup \{R_i\}] = \epsilon \vee C(B[S \cup \{R_i\}]) > C(P)$
      $B[S \cup \{R_i\}] = P$
  }
}
**return** $B[\{R_1, \ldots, R_n\}]$

# DPsize

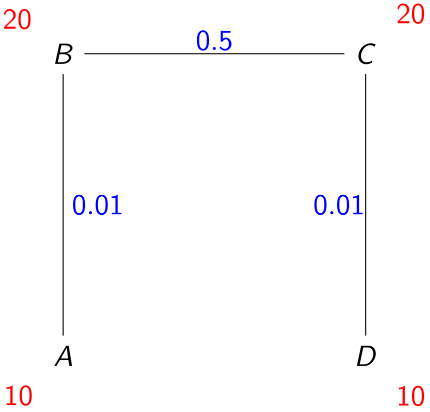
- iterate over subsets of the set of relations, the size is increasing
- $S_1$, $S_2$: $S_1 \cap S_2 = \emptyset$, $S_1$ is connected to $S_2$

# DPsize - example



# Bushy vs. Linear trees

- Linear: add one more relation every time, i.e. add $R$ to optimal $T_1$ to get optimal $T = T_1 \bowtie R$
- Bushy: consider all pairs of optimal $T_1$ and $T_2$ to find optimal $T = T_1 \bowtie T_2$



# DPsub

- Iterate over subsets in the integer order
- Before a join tree for $S$ is generated, all the relevant subsets of $S$ must be available

# DPsub: Integer Enumeration

Enumerate $\{R_1, R_2, R_3, R_4\}$ in Integer order.

# NB

The ability to build the DP table is crucial for passing the exam!

# Homework: Task 1 (15 points)

- Give an example query qraph with join selectivities for which the greedy operator ordering (GOO) algorithm does not give the optimal (with regards to $C_{out}$) join tree. Specify the optimal join tree.
- For that example perform the IKKBZ-based heuristics

# Homework: Task 2 (15 points)

- Using the program from the the last exercise as basis, construct the query graph for each connected component.

# Info

- Exercises due: 9 AM, Dezember 05, 2016