CSE 544
Principles of Database Management Systems

Lecture 11 – Optimization Wrap-up

Announcments

- HW3 due on Friday!
- Don't neglect the project

Will Discuss the Paper

- How good are query optimizers, Really?

Basic Cardinality Estimation

- What are the basic assumptions made in cardinality estimation?

- How is the join size estimated?

Basic Cardinality Estimation

- What are the basic assumptions made in cardinality estimation?
  - uniformity: all values, except for the most-frequent ones, are assumed to have the same number of tuples
  - independence: predicates on attributes (in the same table or from joined tables) are independent
  - principle of inclusion: the domains of the join keys overlap such that the keys from the smaller domain have matches in the larger domain
- How is the join size estimated?

$$|T_1 \bowtie_{x=y} T_2| = \frac{|T_1||T_2|}{\max(\mathrm{dom}(x), \mathrm{dom}(y))},$$

Benchmarks

- What are the traditional database benchmarks?
- Why are they poor tools for evaluating cardinality estimators?

Benchmarks

- What are the traditional database benchmarks?
  - TPC with several benchmarks: TPC/H, TPC/DS, ...
- Why are they poor tools for evaluating cardinality estimators?

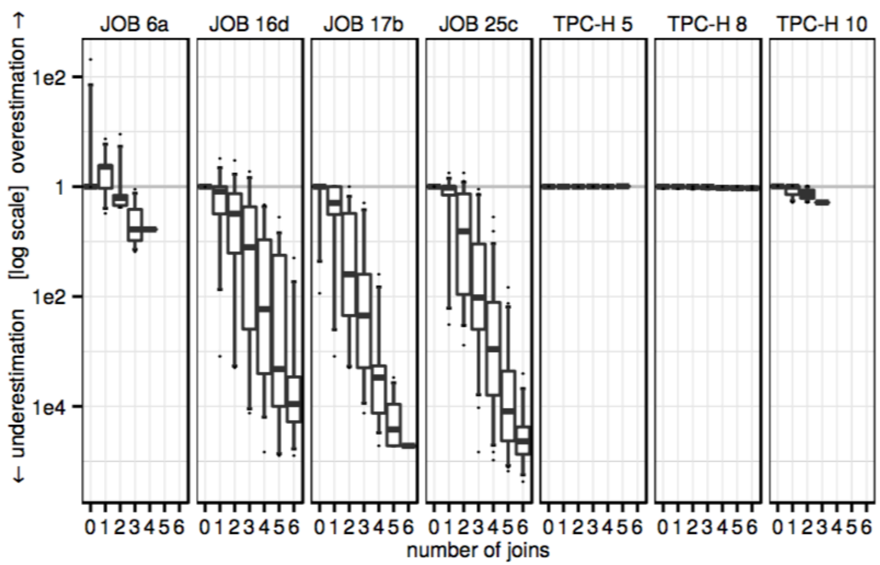


Figure 4: PostgreSQL cardinality estimates for 4 JOB queries and 3 TPC-H queries

Subplans

- For which subplans does an optimizer need to estimate the cardinality?

  $\sigma_{x=5}(A) \bowtie_{A.bid=B.id} B \bowtie_{B.cid=C.id} C$

  where id = primary key;   bid, cid = foreign keys

Subplans

- For which subplans does an optimizer need to estimate the cardinality?

  $\sigma_{x=5}(A) \bowtie_{A.bid=B.id} B \bowtie_{B.cid=C.id} C$

  where id = primary key;   bid, cid = foreign keys

- $\sigma_{x=5}(A)$
  $\sigma_{x=5}(A) \bowtie_{A.bid=B.id} B$
  $B \bowtie_{B.cid=C.id} C$
  $\sigma_{x=5}(A) \bowtie_{A.bid=B.id} B \bowtie_{B.cid=C.id} C$
  If index on the fk A.bid, then $A \bowtie_{A.bid=B.id} B$ why??

Discuss Main Graph

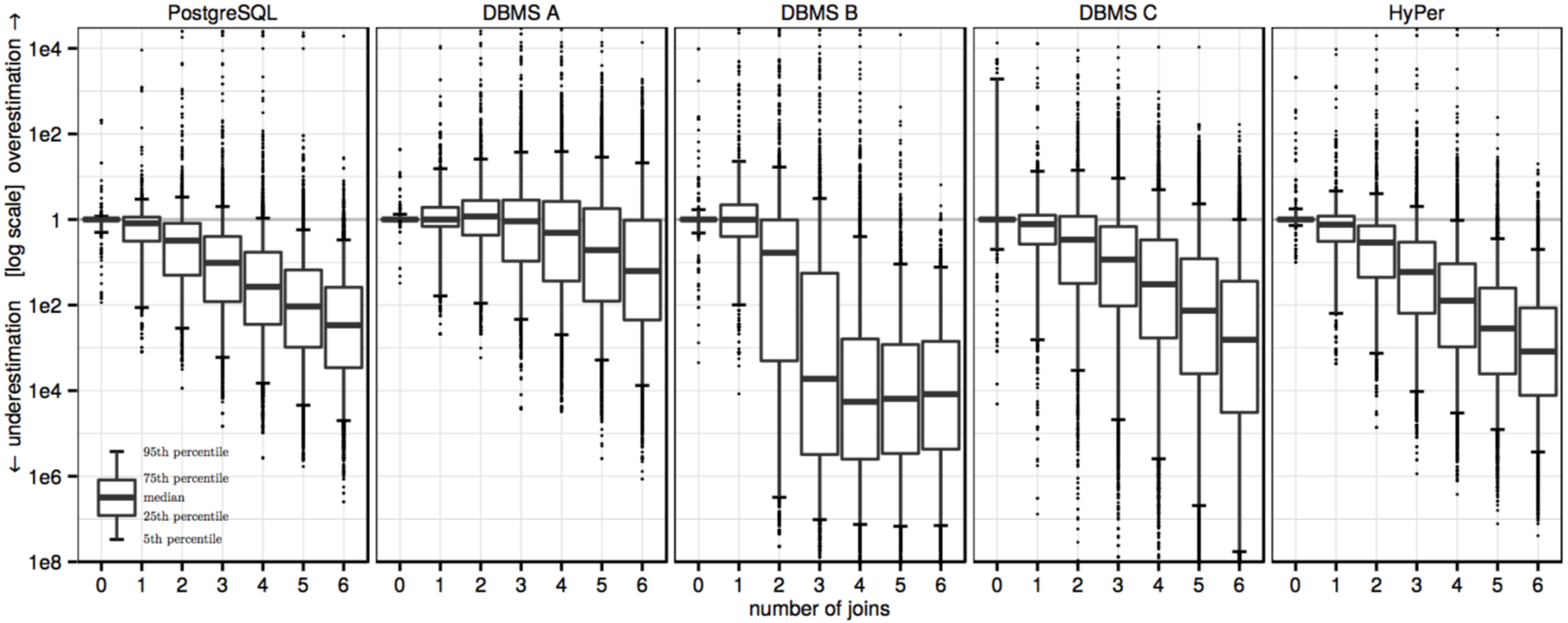


Figure 3: Quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. Each boxplot summarizes the error distribution of all subexpressions with a particular size (over all queries in the workload)

Discuss Main Graph

Sample-based??

Sample-based

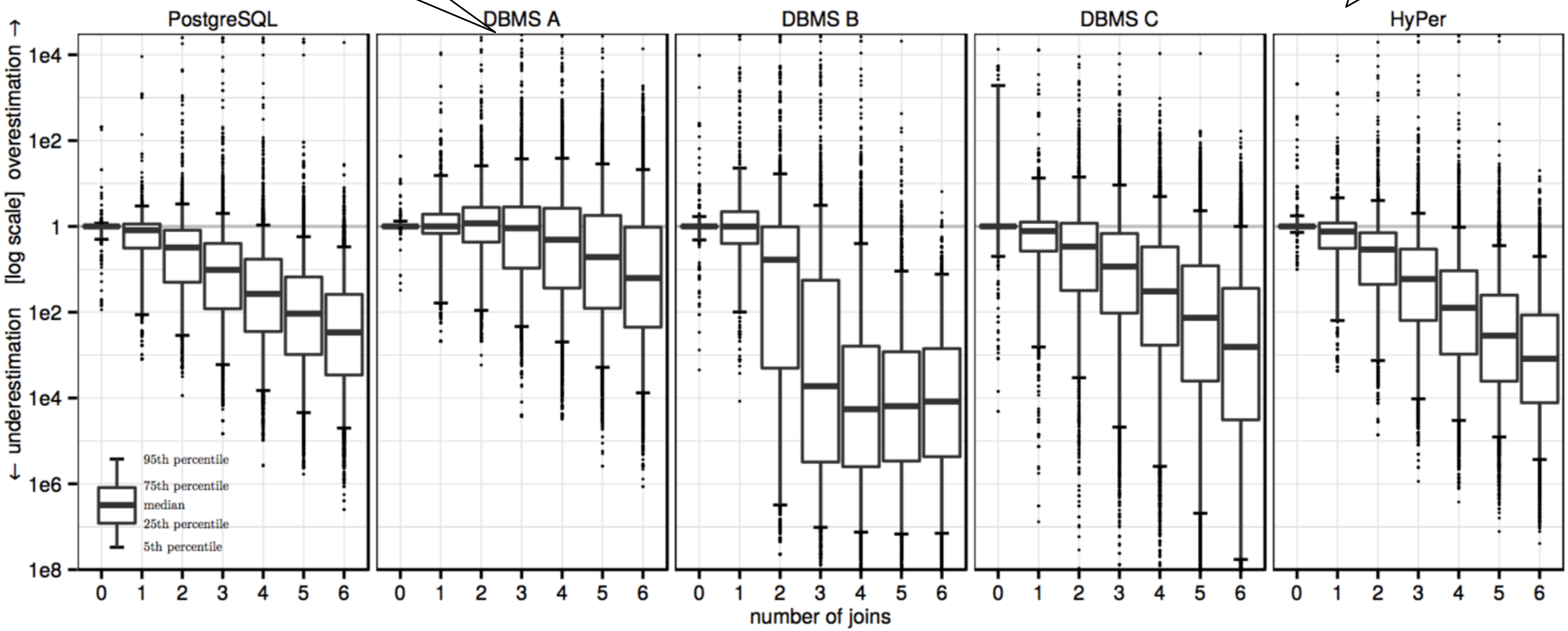


Figure 3: Quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. Each boxplot summarizes the error distribution of all subexpressions with a particular size (over all queries in the workload)

Samples v.s. Distinct Values

- Estimate $\sigma_{x=5}(A)$
  - Distinct values: $|\sigma_{x=5}(A)| \approx T(A) / V(A,x)$ (= $|A|$ / $Dom(A.x)$)
  - Sample: keep a sample SA, use Thomson's estimator:
    $|\sigma_{x=5}(A)| \approx |\sigma_{x=5}(SA)| * |A| / |SA|$
  - HyPer and possibly System A use samples

- Discuss pros and cons of sampling-based estimate

Samples v.s. Distinct Values

- Estimate $\sigma_{x=5}(A)$
  - Distinct values: $|\sigma_{x=5}(A)| \approx T(A) / V(A,x)$ (= $|A|$ / $Dom(A.x)$)
  - Sample: keep a sample SA, use Thomson's estimator:
    $|\sigma_{x=5}(A)| \approx |\sigma_{x=5}(SA)| * |A| / |SA|$
  - HyPer and possibly System A use samples

- Discuss pros and cons of sampling-based estimate
  - Pros: very good for single table; correlated attributes; complex predicates (A.x like "%Johnson%")
  - Cons: return estimate 0 if sample doesn't contain predicate; do not work for joins (explain in class)

End-Effect on Query Runtime

Do poor cardinality estimators lead to worse runtime?

- Case 1: simple access paths (i.e. indices on keys only)
- Case 2: complex access paths (add indices on fk's)

End-Effect on Query Runtime

Do poor cardinality estimators lead to worse runtime?

- Case 1: simple access paths (i.e. indices on keys only)
- Case 2: complex access paths (add indices on fk's)

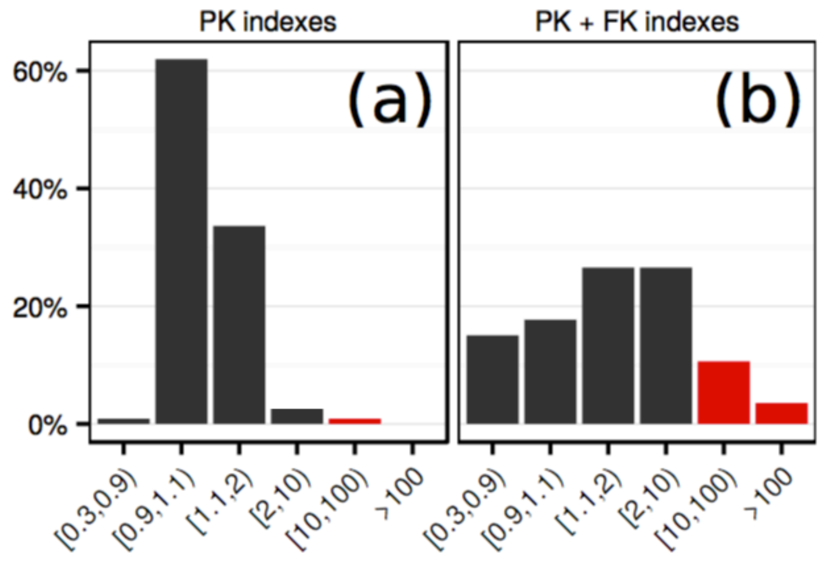


Figure 7: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (different index configurations)

End-Effect on Query Runtime

Do poor cardinality estimators lead to worse runtime?

- Case 1: simple access paths (i.e. indices on keys only)
- Case 2: complex access paths (add indices on fk's)

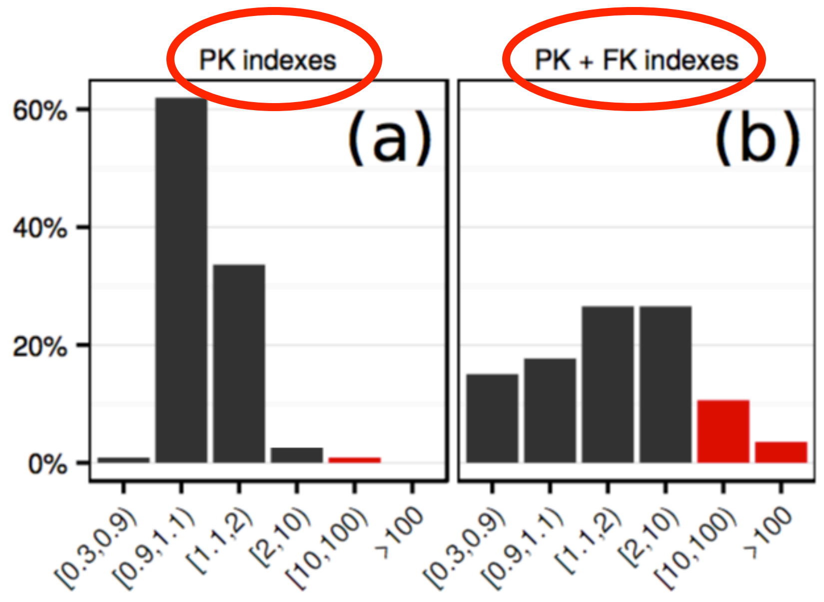


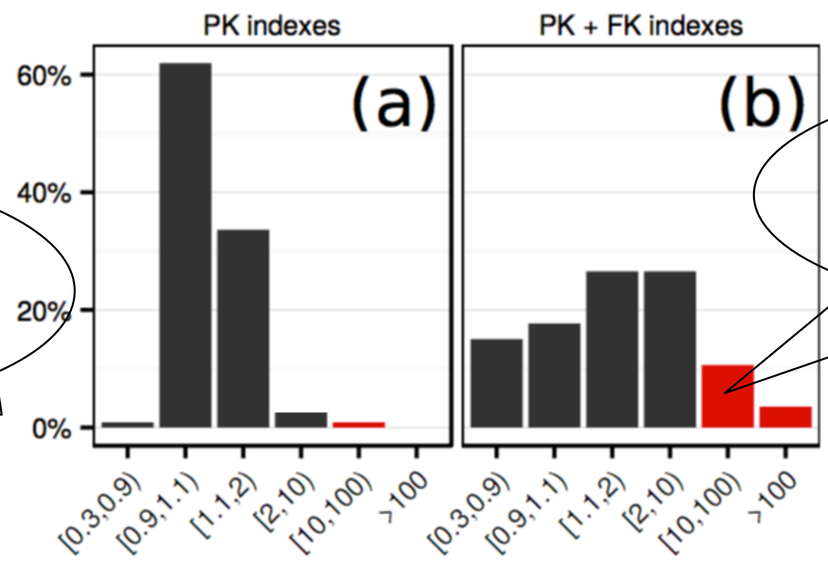
Figure 7: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (different index configurations)

End-Effect on Query Runtime

Do poor cardinality estimators lead to worse runtime?

- Case 1: simple access paths (i.e. indices on keys only)
- Case 2: complex access paths (add indices on fk's)

Base runtime:
query optimized
with true cardinalities



Several queries suffered
significantly from poor
estimates

Figure 7: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (different index configurations)

Cost Model

- Given the estimated cardinality, need to estimate actual cost = weighted sum of I/O cost plus CPU cost (x400)
- What are the main takeaways?

Cost Model

- Given the estimated cardinality, need to estimate actual cost = weighted sum of I/O cost plus CPU cost (x400)
- What are the main takeaways?

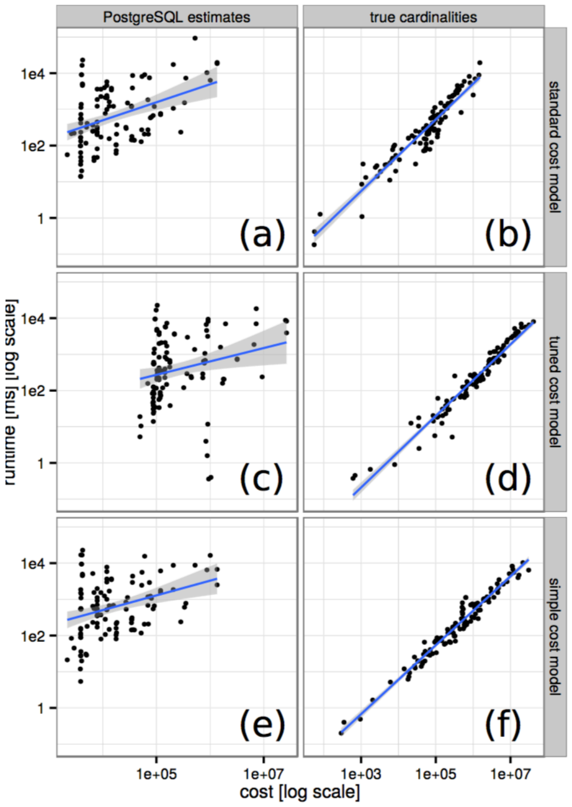


Figure 8: Predicted cost vs. runtime for different cost models

Cost Model

- Given the estimated cardinality, need to estimate actual cost = weighted sum of I/O cost plus CPU cost (x400)
- What are the main takeaways?

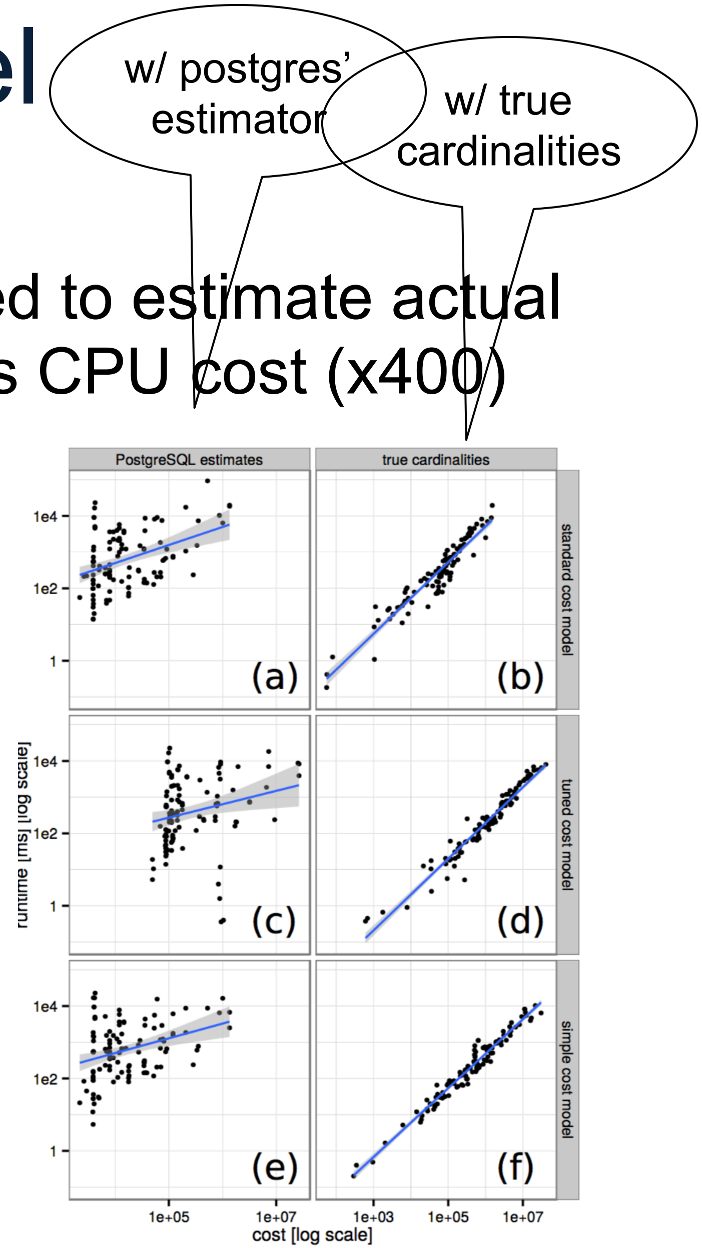


Figure 8: Predicted cost vs. runtime for different cost models

Cost Model

w/ postgres' estimator

w/ true cardinalities

- Given the estimated cardinality, need to estimate actual cost = weighted sum of I/O cost plus CPU cost (x400)
- What are the main takeaways?

Current cost model

Increase CPU weight x50

Just use a simple formula

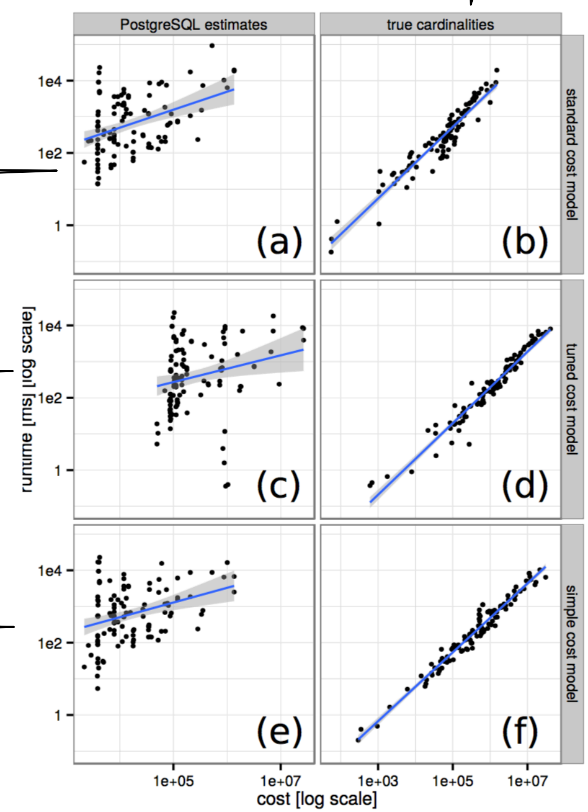


Figure 8: Predicted cost vs. runtime for different cost models

Structural Query Optimization

- Studied by the theory community, little implementation
- Most critical for "aggregate push-down"

  select count(*) from Author, Publication;
  -- takes forever! But should take 2-3 seconds (why?)

Conjunctive Queries

- Definition:
  Q(X) :- R1(X1), R2(X2), ..., Rm(Xm)
- Same as a single datalog rule
- Terminology:
  - Atoms
  - Head variables
  - Existential variables
- CQ = denotes the set of conjunctive queries

Examples

- Example of CQ

$$q(x,y) = \exists z.(R(x,z) \wedge \exists u.(R(z,u) \wedge R(u,y)))$$

$$q(x) = \exists z.\exists u.(R(x,z) \wedge R(z,u) \wedge R(u,y))$$

- Examples of non-CQ:

$$q(x,y) = S(x,y) \wedge \forall z.(R(x,z) \rightarrow R(y,z))$$

$$q(x) = T(x) \vee \exists z.S(x,z)$$

Types of CQ

- **Full** CQ: head variables are all variables
  Q(x,y,z,u) :- R(x,y),S(y,z),T(z,u)

- **Boolean** CQ: no head variables
  Q() :- R(x,y),S(y,z),T(z,u)

- With or without **self-joins**:
  Q(x,u) :- R(x,y),S(y,z),R(z,u)
  Q(x,u) :- R(x,y),S(y,z),T(z,u)

Extensions

- With **inequalities** $CQ^{<}$:
  Q(x) :- R(x,y),S(y,z),T(z,u),y<u

- With **disequalities** $CQ^{\neq}$:
  Q(x) :- R(x,y),S(y,z),T(z,u),y≠u

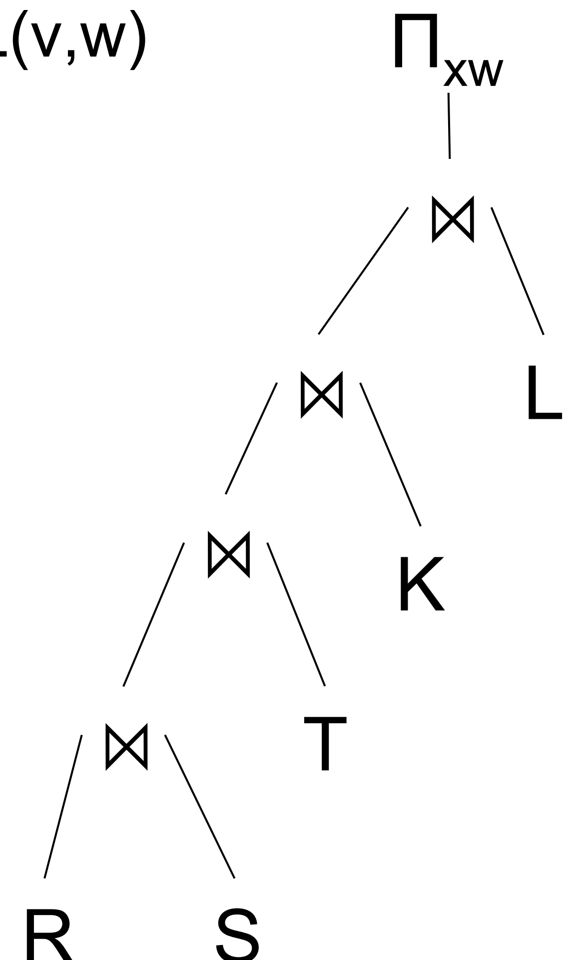
- With **aggregates**:
  Q(x,count(*)) :- R(x,y),S(y,z),T(z,u)
  Q(x, sum(u)) :- R(x,y),S(y,z),T(z,u)

Question in Class

- $Q(x,w) \text{ :- } R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)$
- Assume $|R|=|S|=|T|=|K|=|L| = N$
- What is the complexity of Q?

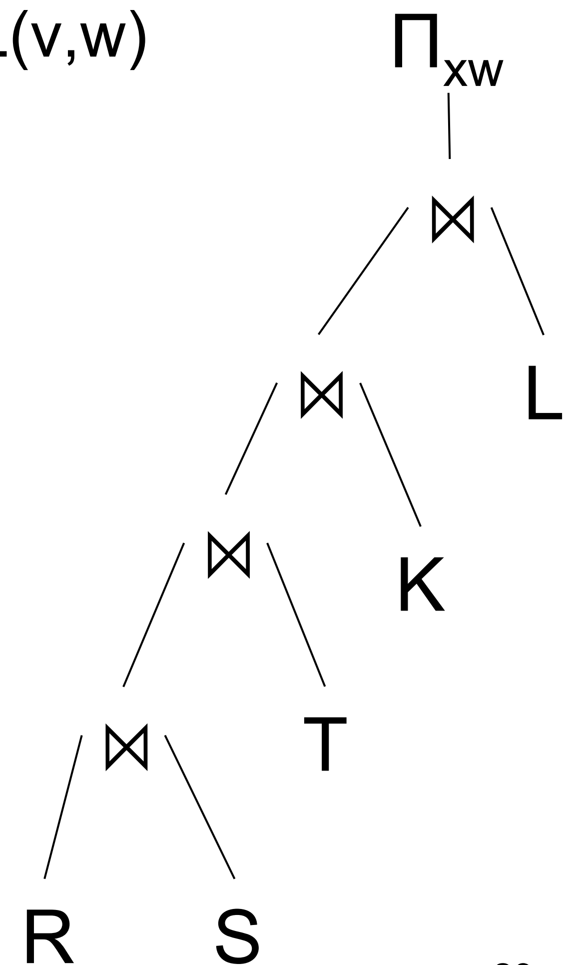
Question in Class

- Q(x,w) :- R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)
- Assume |R|=|S|=|T|=|K|=|L| = N
- What is the complexity of Q?
- What is the complexity of this plan?



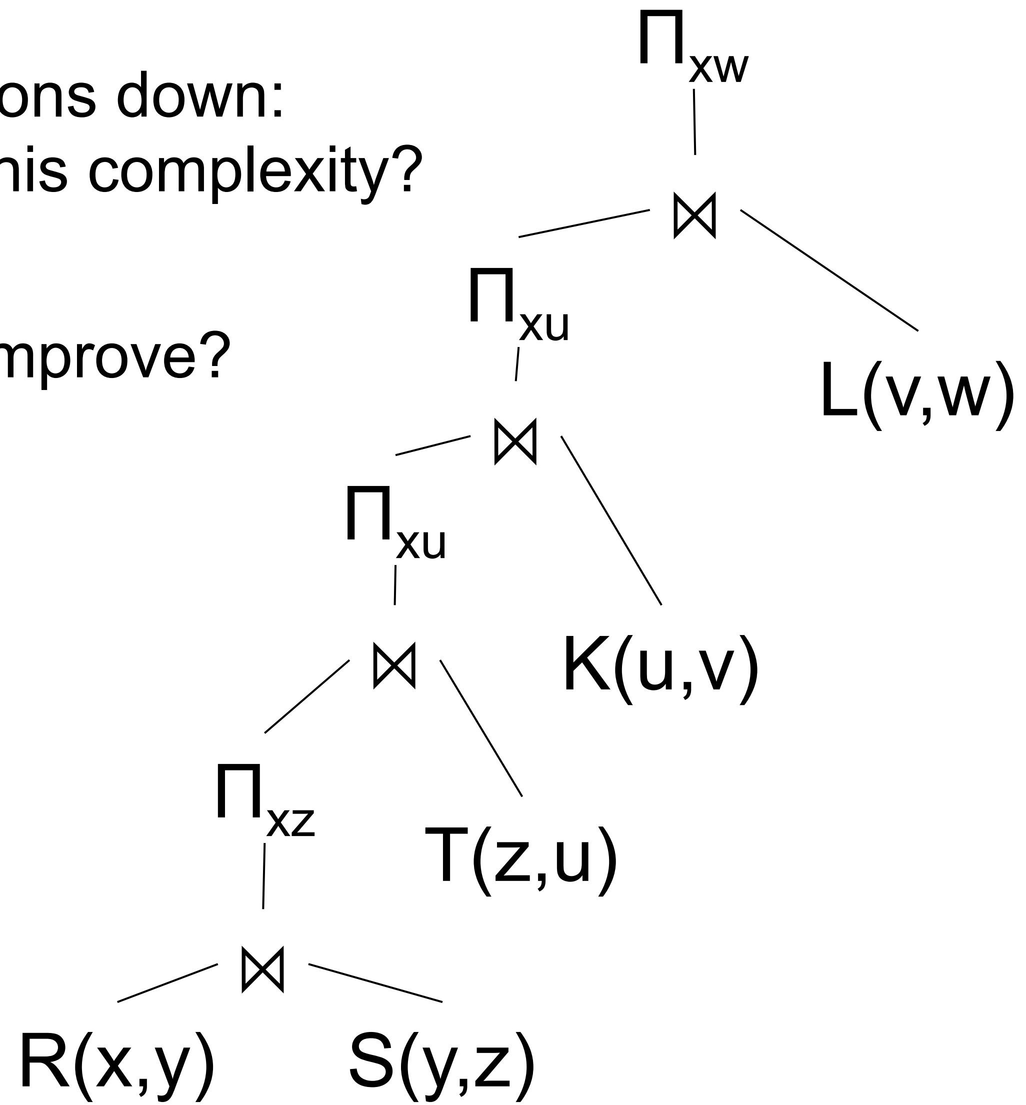
Question in Class

- Q(x,w) :- R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)
- Assume |R|=|S|=|T|=|K|=|L| = N
- What is the complexity of Q?
- What is the complexity of this plan?
- Can you find a more efficient plan?



Question in Class

- Push projections down: What about this complexity?
- Can we still improve?



Semijoin Optimizations **REVIEW**

- In parallel databases: often combined with Bloom Filters (pp. 747 in the textbook)
- Magic sets for datalog were invented after semi-join reductions, and the connection became clear only later
- Some complex semi-join reductions for non-recursive SQL optimizations are sometimes called “magic sets”

Semijoin Reducer

- Given a query:

$$Q = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$$

- A *semijoin reducer* for Q is

$$\begin{aligned} R_{i1} &= R_{i1} \ltimes R_{j1} \\ R_{i2} &= R_{i2} \ltimes R_{j2} \\ &\ldots\ldots \\ R_{ip} &= R_{ip} \ltimes R_{jp} \end{aligned}$$

  such that the query is equivalent to:

$$Q = R_{k1} \bowtie R_{k2} \bowtie \ldots \bowtie R_{kn}$$

- A *full reducer* is such that no dangling tuples remain

Example

- Example:

$$\boxed{Q = R(A,B) \bowtie S(B,C)}$$

- A semijoin reducer is:

$$\boxed{R_1(A,B) = R(A,B) \ltimes S(B,C)}$$

- The rewritten query is:

$$\boxed{Q = R_1(A,B) \bowtie S(B,C)}$$

Semijoin Reducer

- More complex example:

$$Q(y,z,u) = R(\text{'a'}, y), S(y,z), T(z,u), K(u,\text{'b'})$$

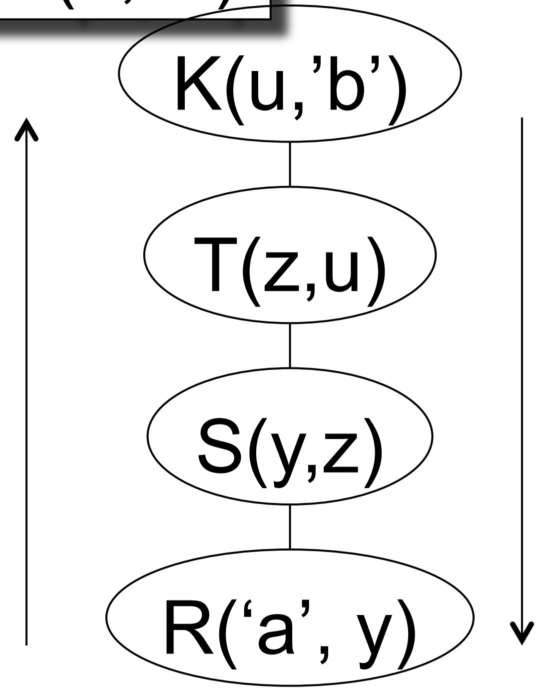
- Find a full reducer

Semijoin Reducer

- More complex example:

  $Q(y,z,u) = R(\text{'a'}, y), S(y,z), T(z,u), K(u,\text{'b'})$

- Find a full reducer



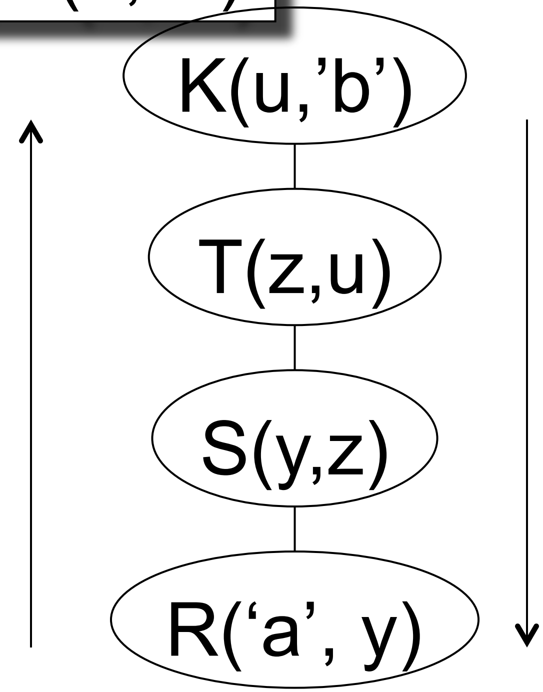
Semijoin Reducer

- More complex example:

$$Q(y,z,u) = R(\text{'a'}, y), S(y,z), T(z,u), K(u,\text{'b'})$$

- Find a full reducer

$$S'(y,z) \text{ :- } S(y,z) \ltimes R(\text{'a'}, y)$$
$$T'(z,u) \text{ :- } T(z,u) \ltimes S'(y,z)$$
$$K'(u) \text{ :- } K(u,\text{'b'}) \ltimes T'(z,u)$$
$$T''(z,u) \text{ :- } T'(z,u) \ltimes K'(u)$$
$$S''(y,z) \text{ :- } S'(y,z) \ltimes T''(z,u)$$
$$R''(y) \text{ :- } R(\text{'a'},y) \ltimes S''(y,z)$$



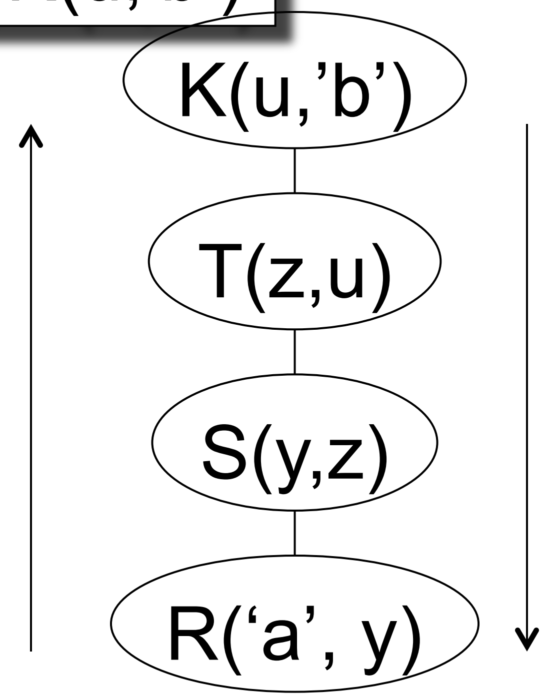
Semijoin Reducer

- More complex example:

$$Q(y,z,u) = R(\text{'a'}, y), S(y,z), T(z,u), K(u,\text{'b'})$$

- Find a full reducer

$S'(y,z)$:- $S(y,z) \ltimes R(\text{'a'}, y)$
$T'(z,u)$:- $T(z,u) \ltimes S'(y,z)$
$K'(u)$:- $K(u,\text{'b'}) \ltimes T'(z,u)$
$T''(z,u)$:- $T'(z,u) \ltimes K'(u)$
$S''(y,z)$:- $S'(y,z) \ltimes T''(z,u)$
$R''(y)$:- $R(\text{'a'},y) \ltimes S''(y,z)$



- Finally, c

$$Q(y,z,u) = R''(y), S''(y,z), T''(z,u), K''(u)$$

Practice at Home…

- Find semi-join reducer for R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)

Not All Queries Have Full Reducers

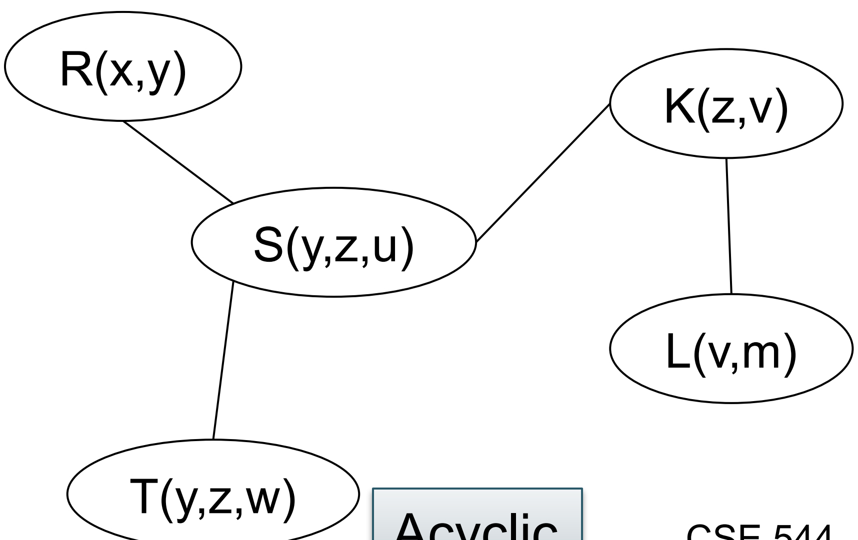
- Example:

$$Q = R(A,B) \bowtie S(B,C) \bowtie T(A,C)$$

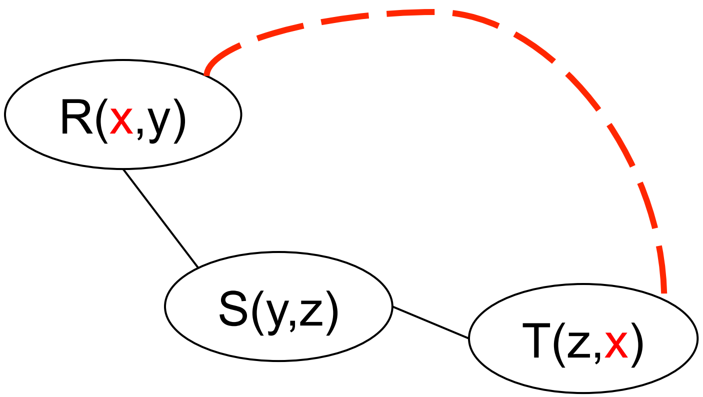
- Can write many different semi-join reducers
- But no full reducer of length O(1) exists

Acyclic Queries

- Fix a Conjunctive Query without self-joins
- Q is *acyclic* if its atoms can be organized in a tree such that for every variable the set of nodes that contain that variable form a connected component



Acyclic



R(x,y),S(y,z),T(z,x) is cyclic

Yannakakis Algorithm

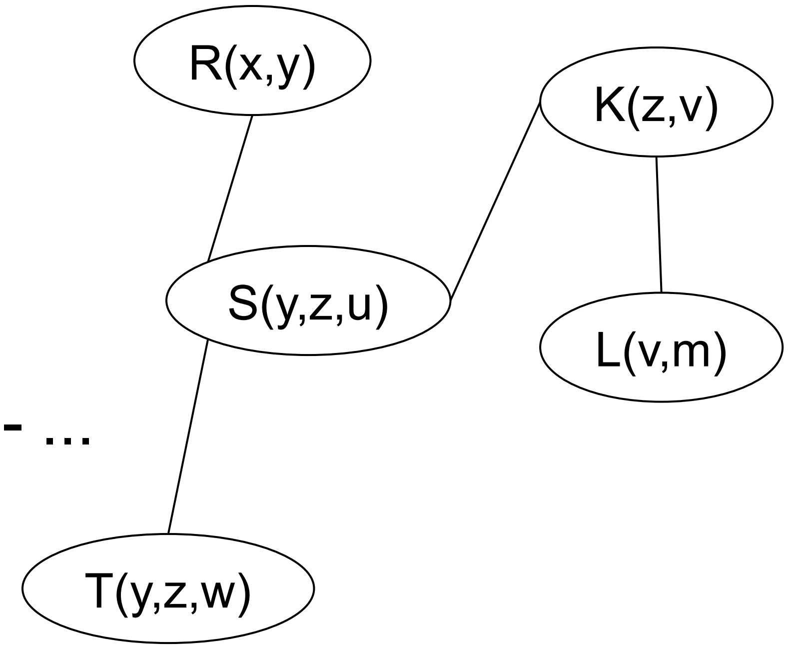
- Given: acyclic query Q
- Compute Q on any database in time O(|Input|+|Output|)

- Step 1: semi-join reduction
  - Pick any root node x in the tree decomposition of Q
  - Do a semi-join reduction sweep from the leaves to x
  - Do a semi-join reduction sweep from x to the leaves
- Step 2: compute the joins bottom up, with early projections

Examples in Class

R(x,y),S(y,z,u),T(y,z,w),K(z,v),L(v,m)

- Boolean query: Q() :- ...
- Non-boolean: Q(x,m) :- ...
- With aggregate: Q(x,sum(m)) :- ...
- And also: Q(x,count(*)) :- ...



In all cases: runtime = O(|R|+|S|+...+|L| + |Output|)

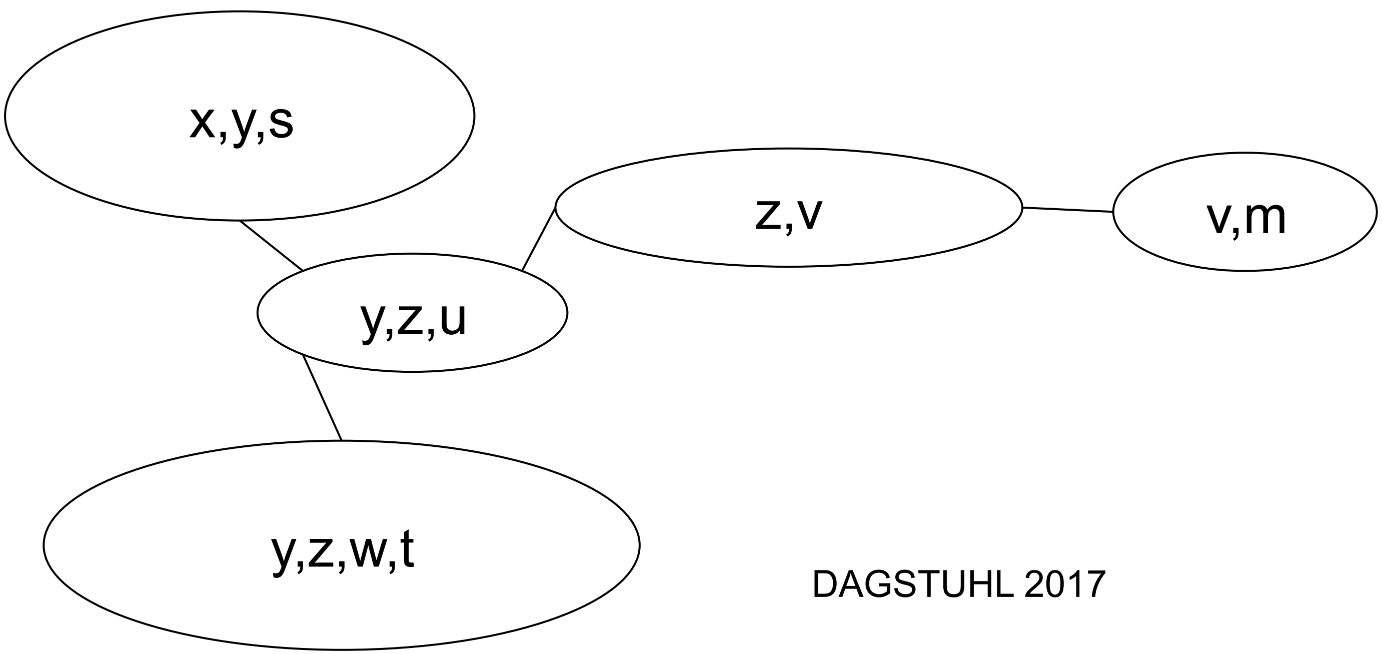
Testing if Q is Acyclic

- An *ear* of Q is an atom R(X) with the following property:
  - Let X' ⊆ X be the set of join variables (meaning: they occur in at least one other atom)
  - There exists some other atom S(Y) such that X' ⊆ Y
- The GYO algorithm (Graham,Yu,Özsoyoğlu) for testing if Q is acyclic:
  - While Q has an ear R(X), remove the atom R(X) from the query
  - If all atoms were removed, then Q is acyclic
  - If atoms remain but there is no ear, then Q is cyclic
- Show example in class

Tree Decomposition

Def *Tree decomposition* is (T, χ), χ:Nodes(T)→$2^{\text{Vars}(Q)}$ s.t.:
(1) $\forall A \in$ Atoms(Q) $\exists t \in$ Nodes(T), Vars(A) $\subseteq \chi(t)$
(2) $\forall x \in$ Vars(Q), $\{t \mid x \in \chi(t)\}$ is connected

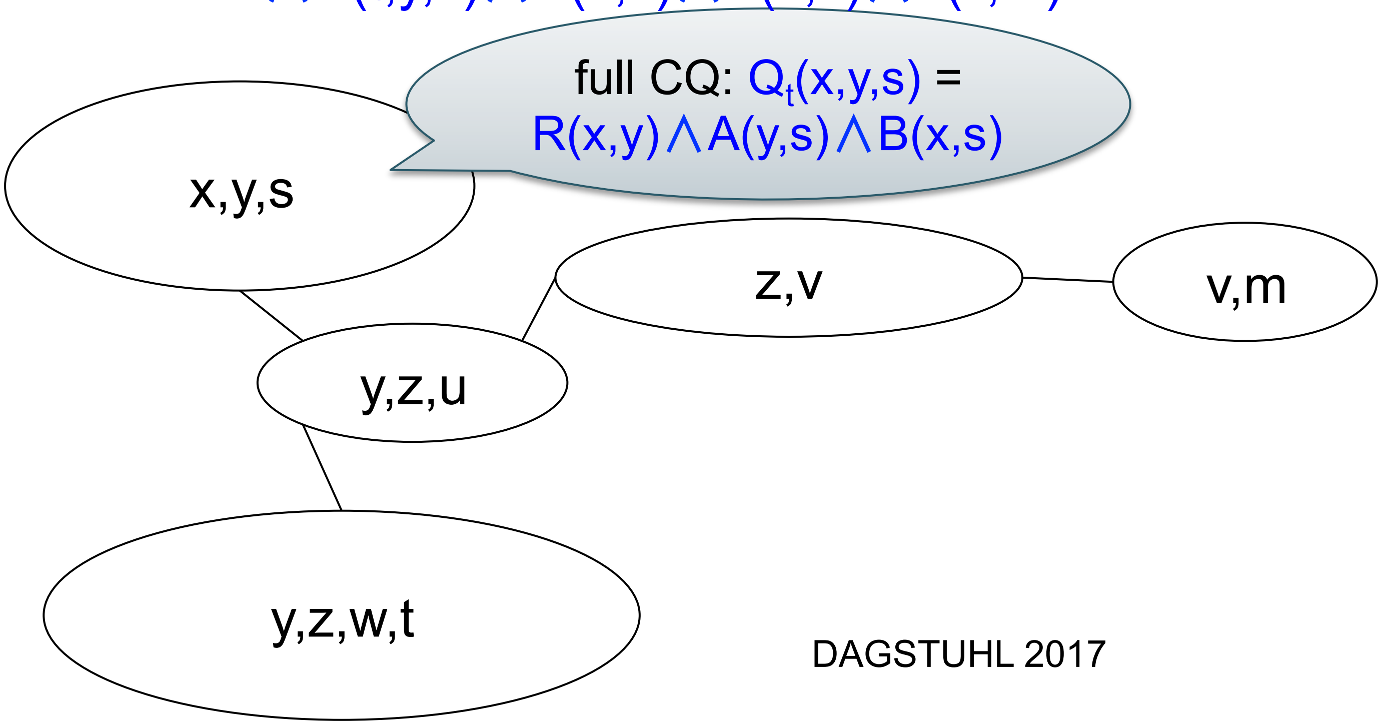
$Q(x,\ldots,m) = R(x,y) \wedge A(y,s) \wedge B(x,s) \wedge S(y,z,u) \wedge T(y,z,w) \wedge C(z,w,t) \wedge D(w,t,y)$
$\wedge E(t,y,z) \wedge K(z,v) \wedge F(z,v) \wedge L(v,m)$



Tree Decomposition

Def *Tree decomposition* is (T, χ), χ:Nodes(T)→$2^{\text{Vars}(Q)}$ s.t.:
(1) $\forall A \in$ Atoms(Q) $\exists t \in$ Nodes(T), Vars(A) $\subseteq \chi(t)$
(2) $\forall x \in$ Vars(Q), $\{t \mid x \in \chi(t)\}$ is connected

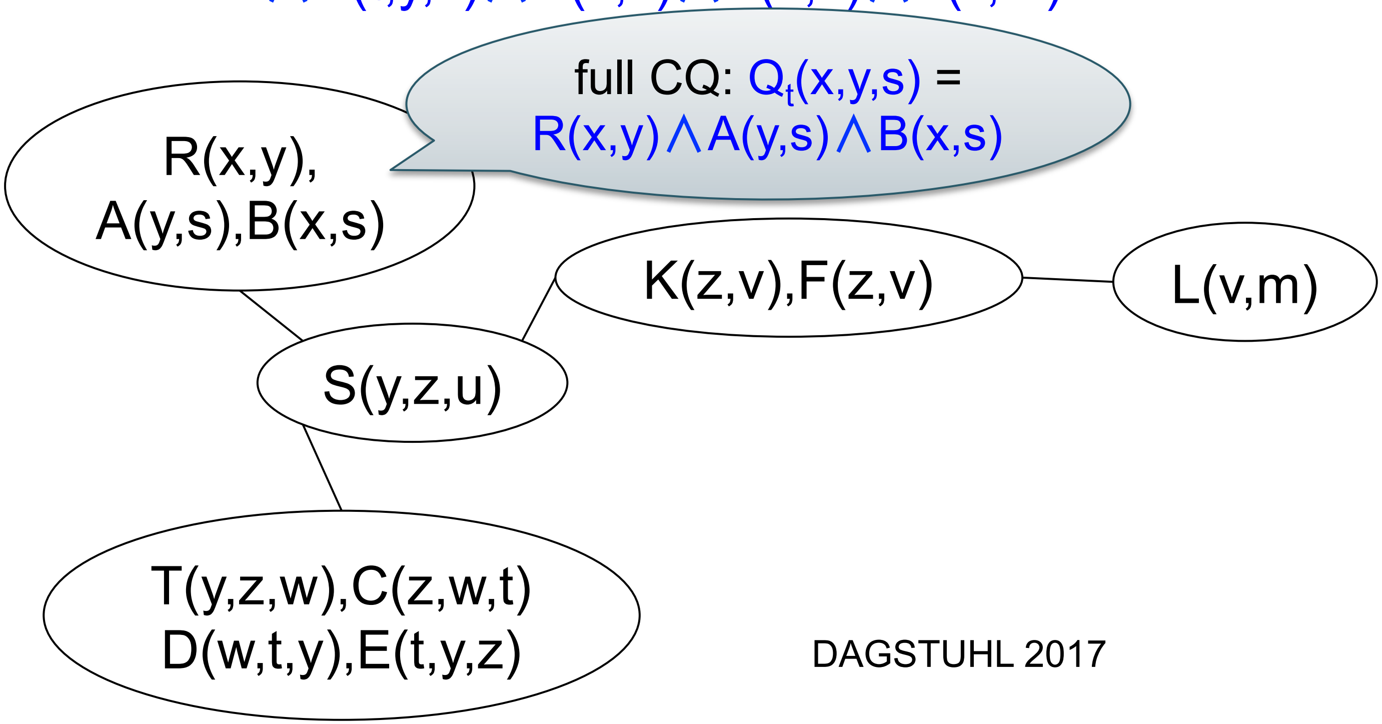
$Q(x,\ldots,m) = R(x,y) \wedge A(y,s) \wedge B(x,s) \wedge S(y,z,u) \wedge T(y,z,w) \wedge C(z,w,t) \wedge D(w,t,y) \wedge E(t,y,z) \wedge K(z,v) \wedge F(z,v) \wedge L(v,m)$



Tree Decomposition

Def *Tree decomposition* is (T, χ), χ:Nodes(T)→$2^{\text{Vars}(Q)}$ s.t.:
(1) $\forall A \in \text{Atoms}(Q)\ \ \exists t \in \text{Nodes}(T),\ \text{Vars}(A) \subseteq \chi(t)$
(2) $\forall x \in \text{Vars}(Q)$, $\{t \mid x \in \chi(t)\}$ is connected

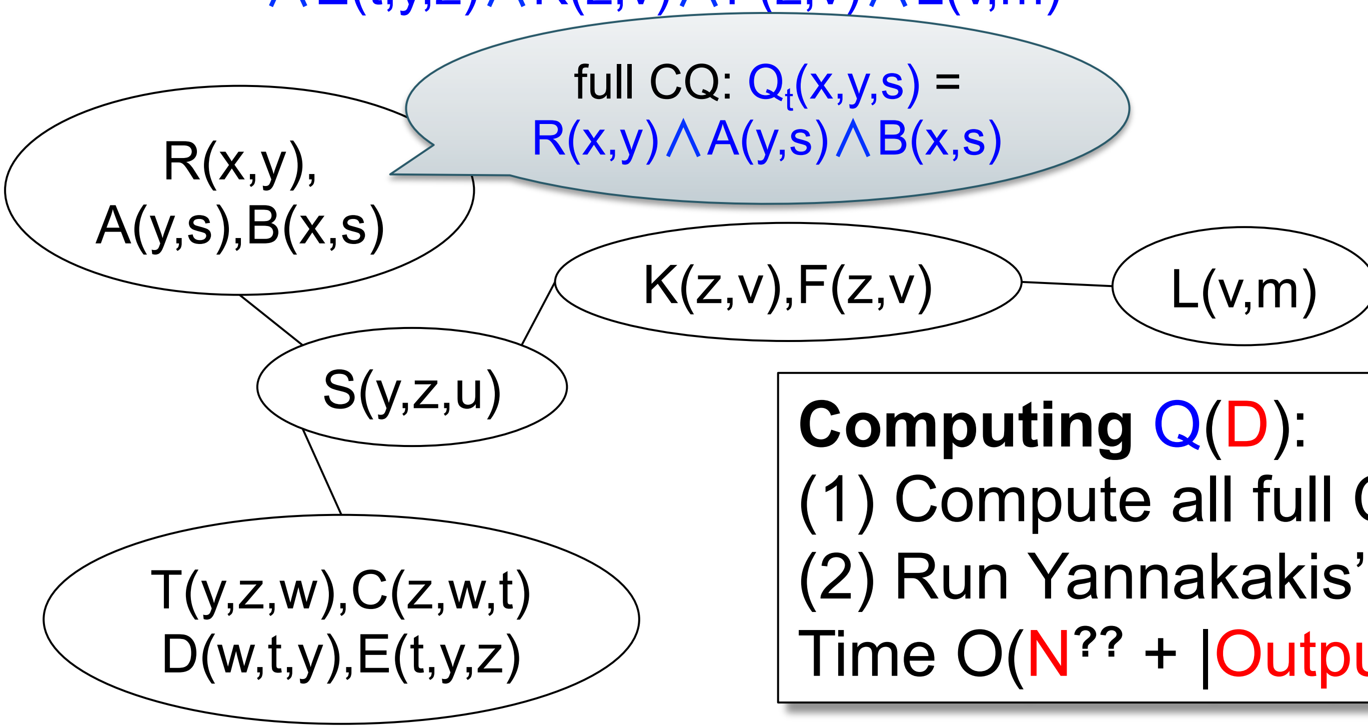
$Q(x,\ldots,m) = R(x,y) \wedge A(y,s) \wedge B(x,s) \wedge S(y,z,u) \wedge T(y,z,w) \wedge C(z,w,t) \wedge D(w,t,y) \wedge E(t,y,z) \wedge K(z,v) \wedge F(z,v) \wedge L(v,m)$



Tree Decomposition

Def *Tree decomposition* is (T, χ), χ:Nodes(T)→$2^{Vars(Q)}$ s.t.:
(1) ∀A∈Atoms(Q) ∃t∈ Nodes(T), Vars(A) ⊆ χ(t)
(2) ∀x∈Vars(Q), {t | x ∈χ(t)} is connected

Q(x,...,m) = R(x,y)∧A(y,s)∧B(x,s)∧S(y,z,u)∧T(y,z,w)∧C(z,w,t)∧D(w,t,y)
∧E(t,y,z)∧K(z,v)∧F(z,v)∧L(v,m)

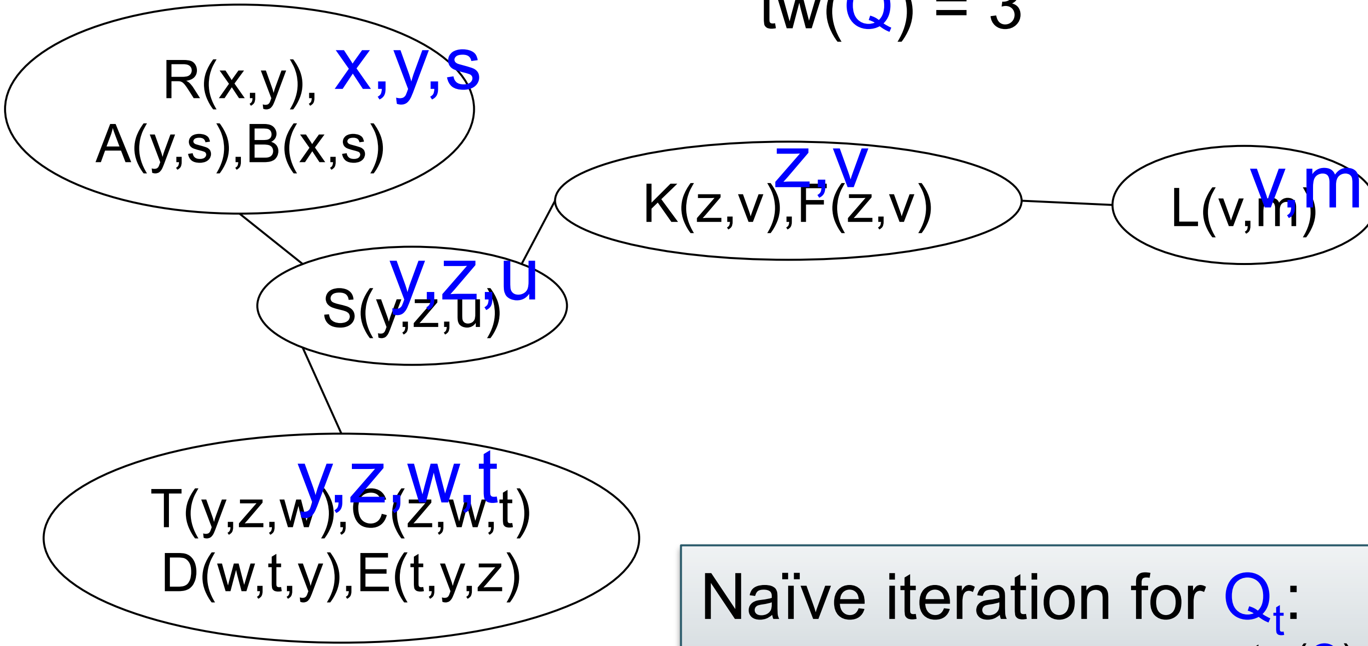


Computing Q(D):
(1) Compute all full CQ's Q_t
(2) Run Yannakakis' on the join tree
Time O($N^{??}$ + |Output|)

Tree-width

Def $tw(Q) = \min_T \max_{t \in Nodes(T)} |Vars(Q_t)| - 1$

tw(Q) = 3



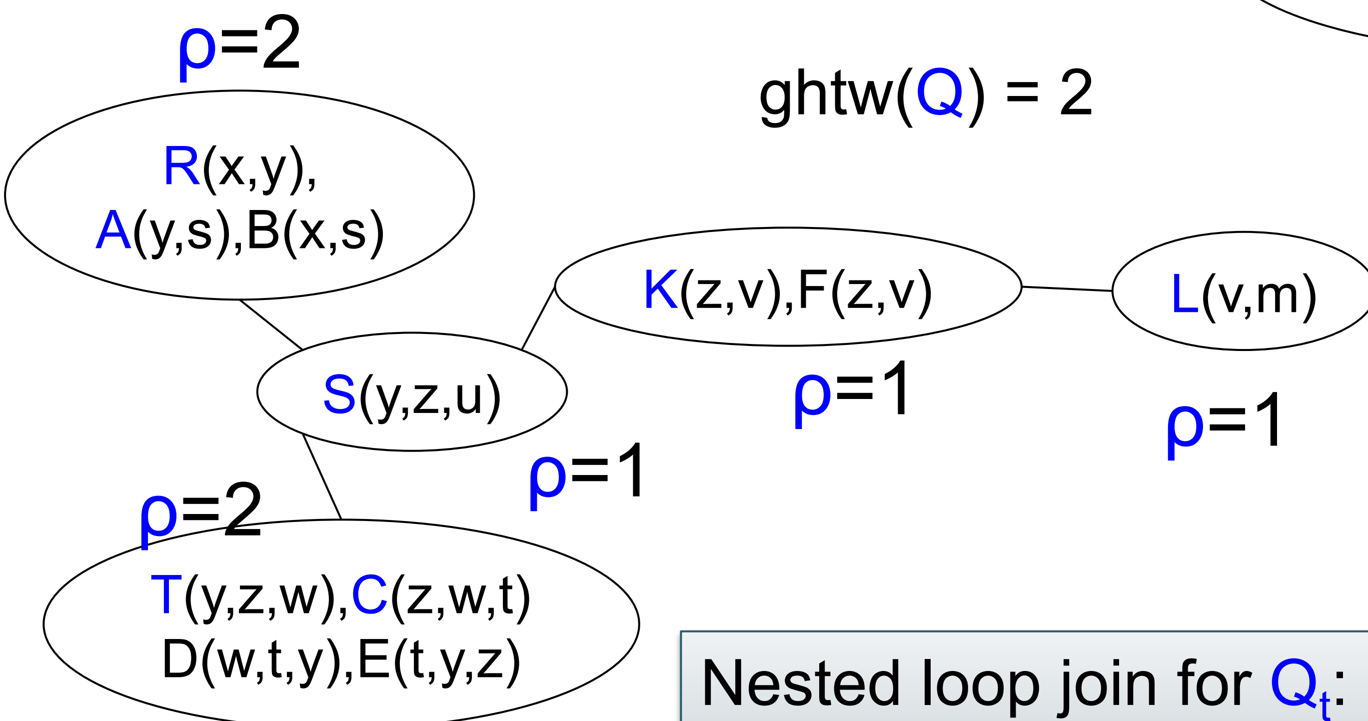
Naïve iteration for Q_t:
Runtime for Q: $O(N^{tw(Q)+1} + |Output|)$

Generalized Hypertree Width

Def $\text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t)$

ρ = edge covering number

$\text{ghtw}(Q) = 2$



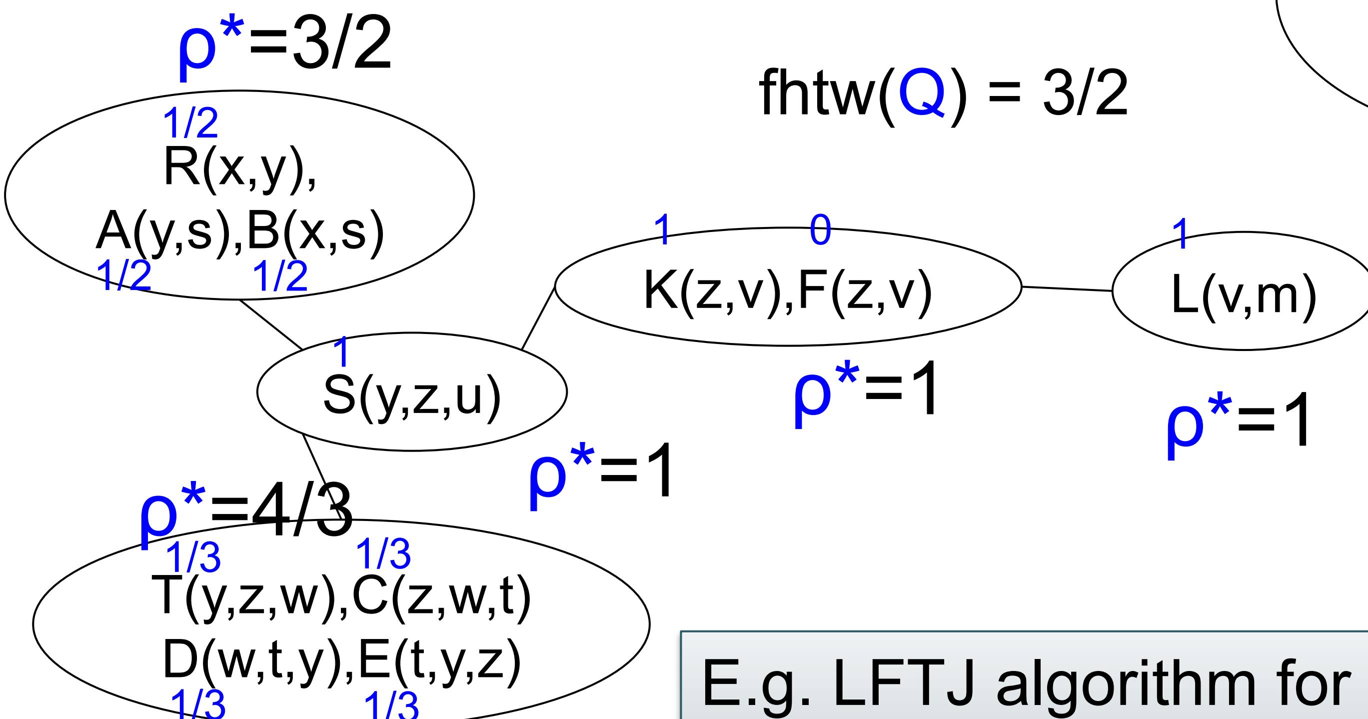
Nested loop join for Q_t:
Runtime for Q: $O(N^{\text{ghtw}(Q)} + |\text{Output}|)$

Fractional Hypertree Width

Def fhtw(Q) = $\min_T \max_{t \in \text{Nodes}(T)} \rho^*(Q_t)$

ρ^* = Fractional edge covering number

fhtw(Q) = 3/2



E.g. LFTJ algorithm for Q_t
Runtime for Q: $O(N^{\text{fhtw}(Q)} + |\text{Output}|)$

Best Algorithm

- Choose *optimal tree* T for Q
- Compute full CQ Q_t for all $t \in \text{Nodes}(T)$
- Run Yannakakis algorithm on the join tree

Total time = $O(N^{fhtw(Q)} + |Output|)$