CS848 Fall 2025: Algorithmic Aspects of Query Processing

Traditional Query Processing

Xiao Hu

Sep 10, 2025

Agenda

- First class: Introduction
- Last and This class: Traditional query processing
  - Relational Algebra
  - Pairwise Framework
  - Yannakakis algorithm
  - Extensions

Semi-Join $R(A,B) \ltimes S(B,C)$

- $R \ltimes S = \pi_{A,B}(R \bowtie S)$: all tuples in R that can be joined with at least one tuple in S
- Law of semi-joins: $R \bowtie S = (R \ltimes S) \bowtie S$
- $O(|R| + |S|)$ ignoring log-factors
- How to use semi-join to remove dangling tuples those won't participate in any join result?

R

A	B
a_1	b_1
a_2	b_1
~~a_1~~	~~b_2~~
a_2	b_3
~~a_1~~	~~b_4~~

S

B	C
b_1	c_3
b_1	c_5
b_3	c_3
b_5	c_5
b_6	c_6

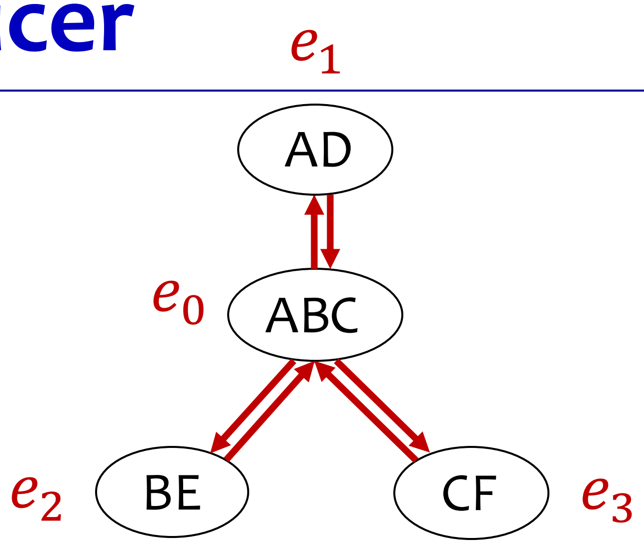
Yannakakis Algorithm: Semi-Join Reducer

Take an arbitrary join tree

- In a bottom-up phase:
  - pick a non-visited node e (with its parent e')
  - update $R_{e'}$ with $R_{e'} \ltimes R_e$

- In a top-down phase:
  - pick a node e
  - For each child e' of e, update $R_{e'}$ with $R_{e'} \ltimes R_e$

- Data Complexity: $O(N)$ ignoring log-factors



$$R_{e_0} \coloneqq R_{e_0} \ltimes R_{e_2}$$
$$R_{e_0} \coloneqq R_{e_0} \ltimes R_{e_3}$$
$$R_{e_1} \coloneqq R_{e_1} \ltimes R_{e_0}$$

$$R_{e_0} \coloneqq R_{e_0} \ltimes R_{e_1}$$
$$R_{e_2} \coloneqq R_{e_2} \ltimes R_{e_0}$$
$$R_{e_3} \coloneqq R_{e_3} \ltimes R_{e_0}$$

Example of Semi-join Reducer

R

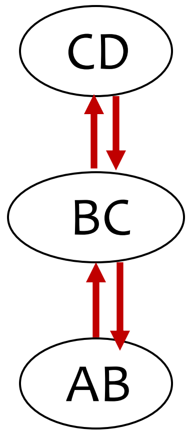
A	B
a_1	b_1
a_2	b_1
a_1	b_2
a_2	b_3
a_1	b_4

S

B	C
b_1	c_3
b_1	c_5
b_3	c_3
b_5	c_5
b_6	c_6

T

C	D
c_1	d_1
c_2	d_2
c_5	d_1
c_6	d_2
c_6	d_3



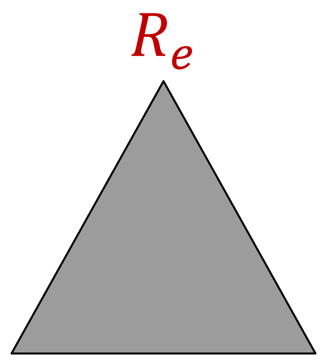
$$Q := R(A,B) \bowtie S(B,C) \bowtie T(C,D)$$

No Dangling tuples

- Every remaining tuple participates in at least one join result.

- In the bottom-up phase:
  - For every node e, once e is reduced, every tuple in R_e participate in at least one result of the sub-join induced by T_e

- In the top-down phase:
  - For every node e, once e is reduced, every tuple in R_e participate in at least one result of the whole join query



T_e: the subtree rooted at node e

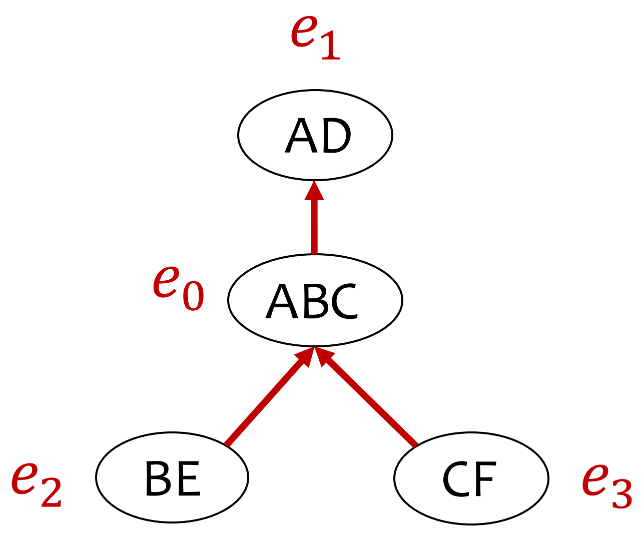
Yannakakis Algorithm: Pairwise Framework

Take an arbitrary join tree

- In a bottom-up phase:
  - pick a non-visited node e (with its parent e')
  - update $R_{e'}$ with $R_{e'} \bowtie R_e$
- Output R_r for the root node r

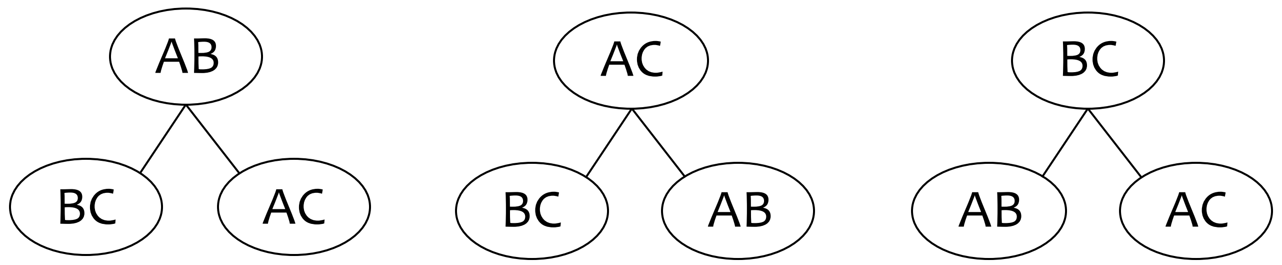
- The intermediate join size is bounded by $O(OUT)$

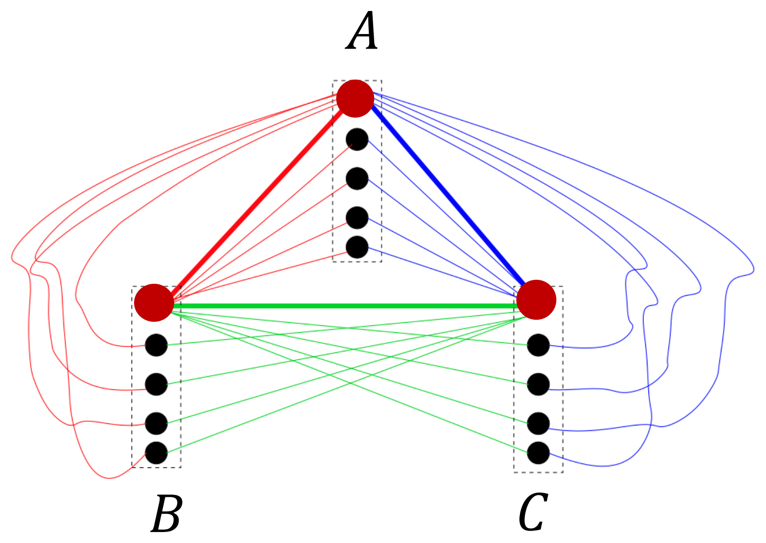
- Data complexity: $O(OUT)$



$$R_{e_0} \coloneqq R_{e_0} \bowtie R_{e_2}$$
$$R_{e_0} \coloneqq R_{e_0} \bowtie R_{e_3}$$
$$R_{e_1} \coloneqq R_{e_1} \bowtie R_{e_0}$$

Discussion: How about Semi-join Reducer on Cyclic Joins?





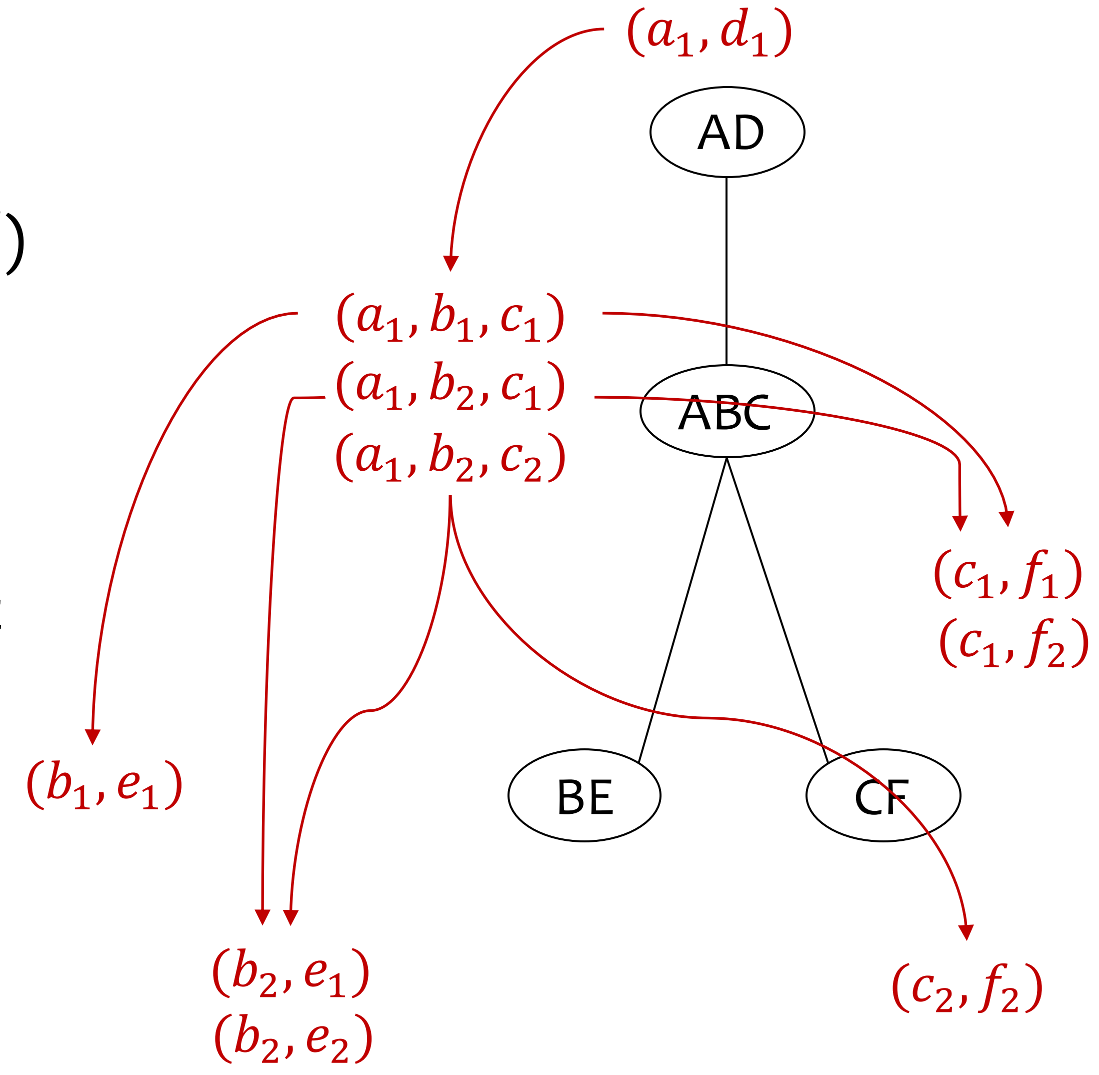
- No more tuples can be removed but some intermediate join result does not participate in any full join result
- How to find non-dangling tuples for cyclic joins? Open questions!

A Practical version of Yannakakis Algorithm

Take an arbitrary join tree

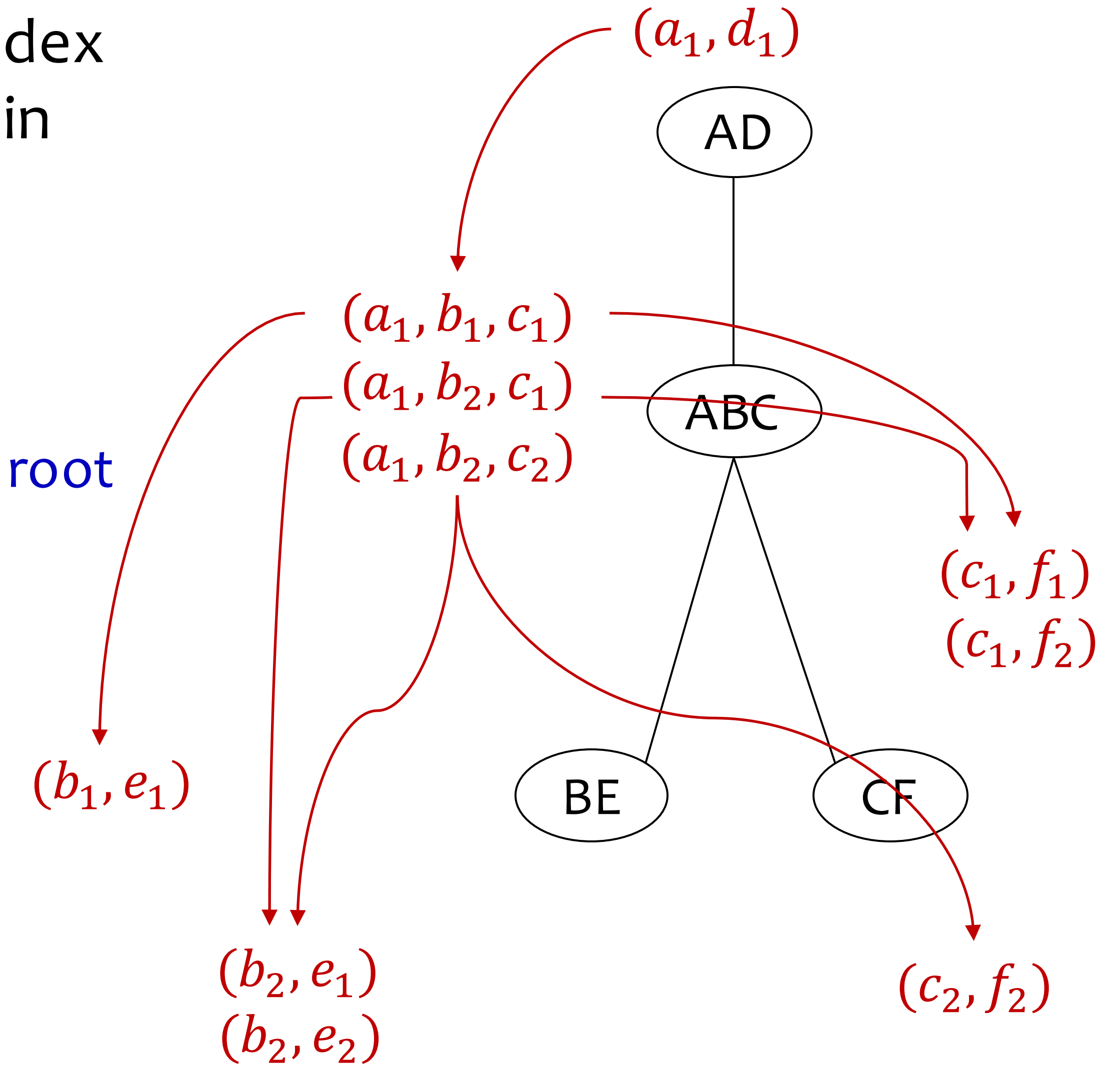
- In a bottom-up phase:
  - pick a non-visited node e (with parent e')
  - update $R_{e'}$ with $R_{e'} \ltimes R_e$

- For each tuple t in the root node R_r:
  - Probe tuples in a top-down manner that can be joined with t



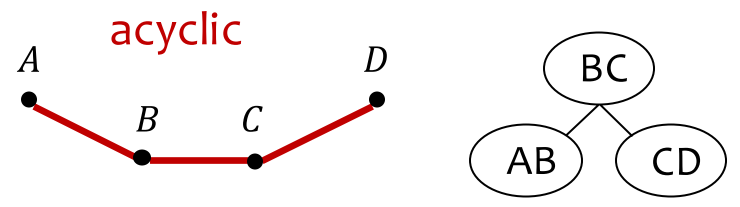
A Practical version of Yannakakis Algorithm

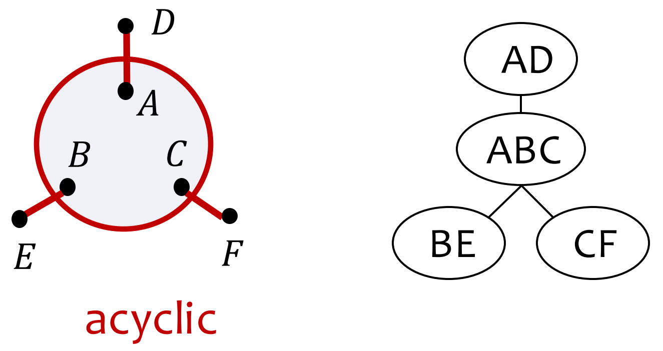
- After $O(N)$ linear preprocessing time, an index of linear size can be built such that every join result can be enumerated with $O(1)$ delay
- Application in dynamic query processing
  - How to maintain non-dangling tuples in the root node and probe indexes?
- Total complexity: $O(N + OUT)$

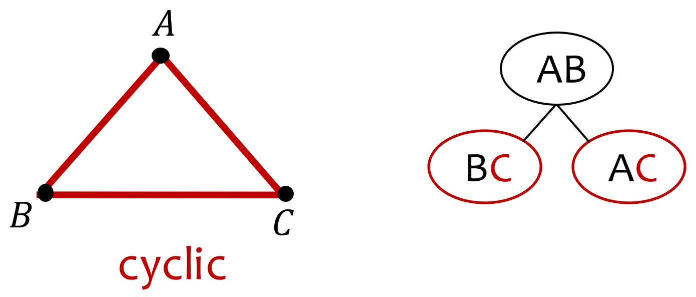


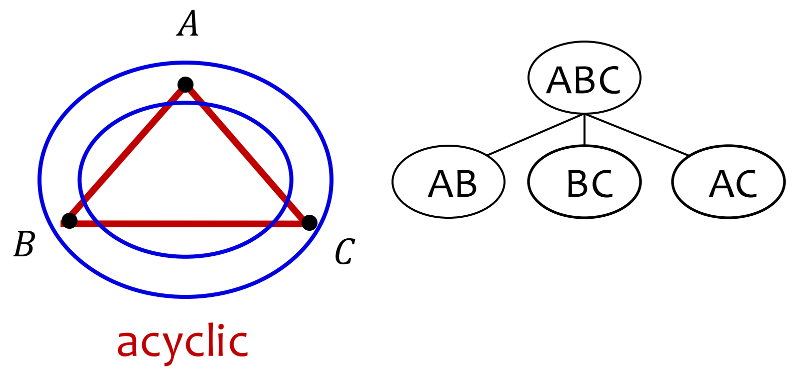
Recap on Acyclicity

- A join query $Q = (V, E)$ is acyclic if it has a join tree T such that
  - one-to-one correspondence between nodes in T with the relations in E;
  - for any attribute $A \in V$, all nodes containing A form a connected subtree.



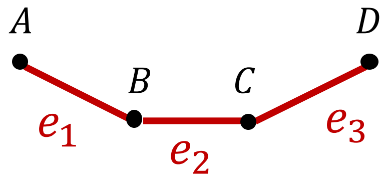




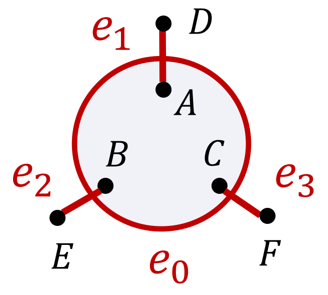


Discussion: How to Build a Join Tree

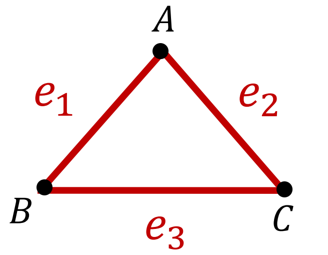
- A query $Q = (V, E)$ is acyclic if GYO reduction results in an empty query:
  - If there is an attribute $A \in V$ that only appears in one relation say e, then remove A from e;
  - If there is a pair of relations $e, e' \in E$ such that $e \subseteq e'$, then remove e from E.

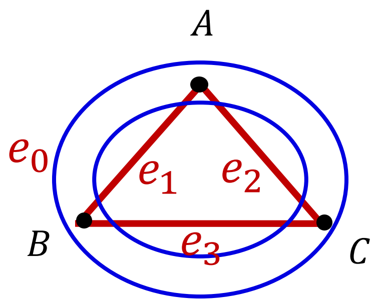


GYO sequence:
$A, e_1, B, e_2, C, D, e_3$



GYO sequence:
$D, e_1, E, e_2, F, e_3, A, B, C, e_0$

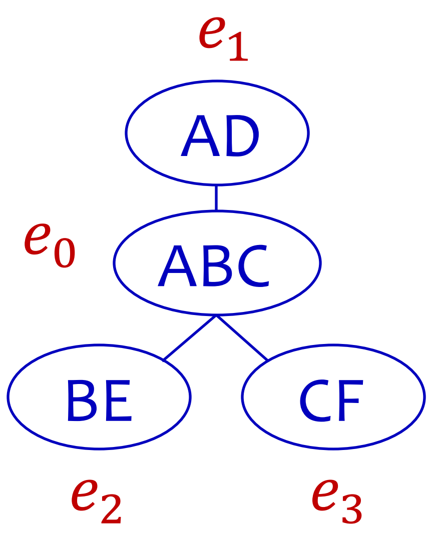




GYO sequence:
$e_1, e_2, e_3, A, B, C, e_0$

Equivalence Between Two Definitions

- Q is acyclic if it has a join tree T.



Always peel off a leaf node

- Q is acyclic if GYO reduction results in an empty query

GYO sequence:

$D, e_1, E, e_2, F, e_3, A, B, C, e_0$

$e_1 \subseteq e_0$ $\quad$ $e_2 \subseteq e_0$ $\quad$ $e_3 \subseteq e_0$

If $e \subseteq e'$ in the GYO sequence, add e as a child node of e'

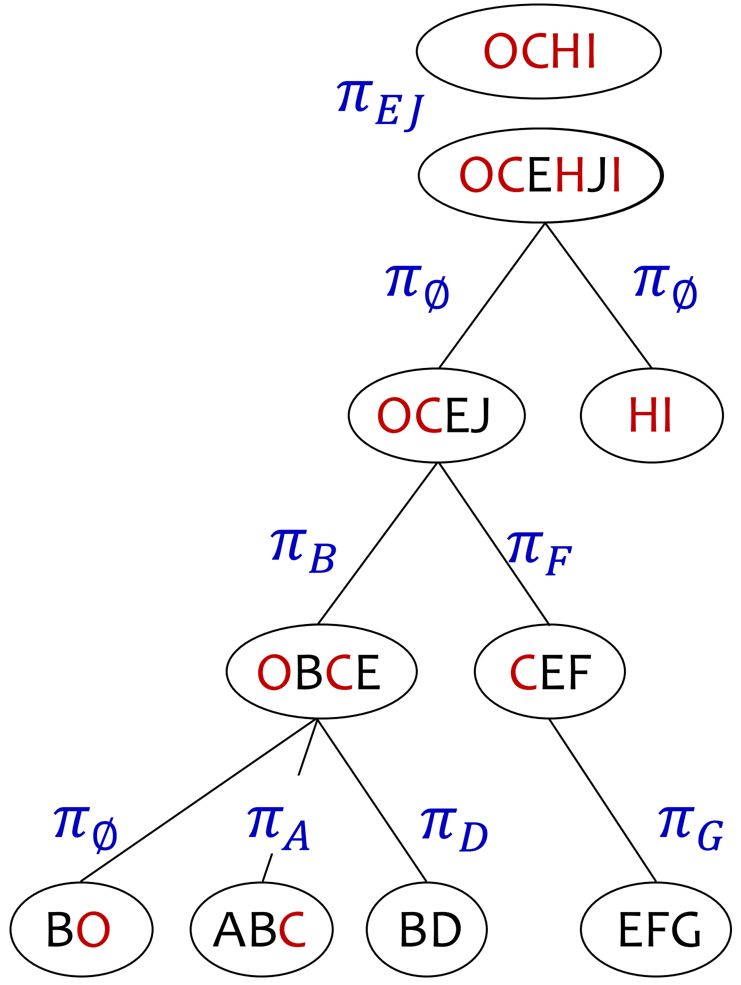
Yannakakis for Acyclic Join-Project Queries

- Chain Matrix Multiplication:
  - $\pi_{AF} R(A, B) \bowtie S(B, C) \bowtie T(C, D) \bowtie W(D, E) \bowtie U(E, F)$

- Parenthesis algorithm

Yannakakis for Acyclic Join-Project Queries

- Let $\boldsymbol{y}$ be the set of output attributes
- Semi-join Reducer
  - If $\boldsymbol{y} = \emptyset$, output true if and only if $R_r \neq \emptyset$
- In a bottom-up phase:
  - Project as early as possible!
  - For a non-output attribute, if it does not appear in any node above, then project it away

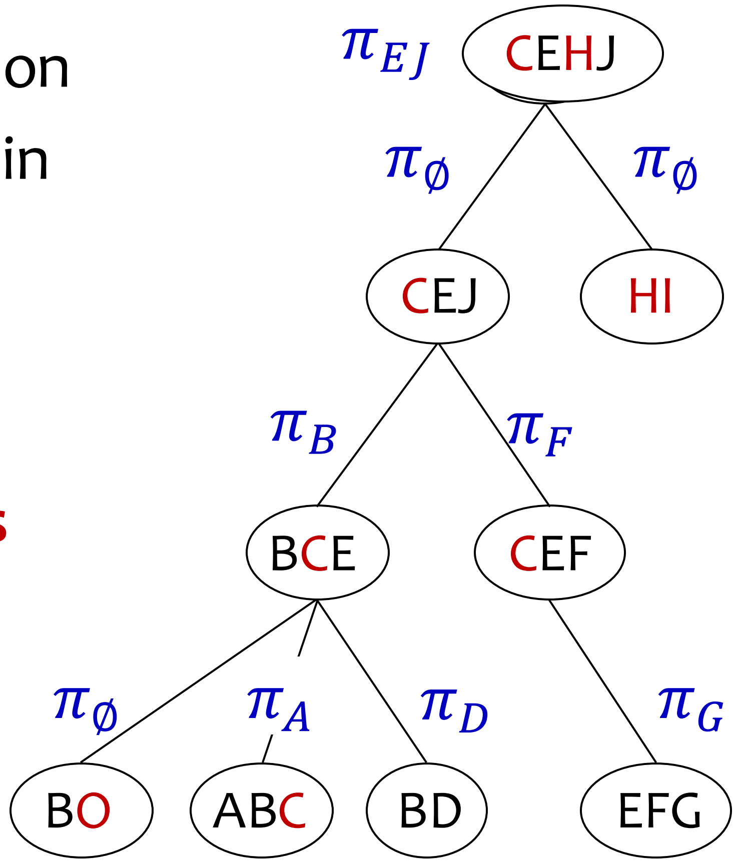


output attributes

Yannakakis for Acyclic Join-Project Query

- Consider an arbitrary node e and its parent u
  - Let v be the attributes at node e after join-projection
  - All attributes in v are either output attributes or join attribute with u
- How to bound the intermediate join size?
  - $R_u \bowtie R_v \subseteq R_u \times (\pi_{v-u} Q)$

    since $v - u$ **are output attributes**
    and no dangling tuples
  - $|R_u \bowtie R_v| \le |R_u| \cdot |Q| \le N \cdot OUT$
- Time complexity: $O(N \cdot OUT)$



output attributes

Yannakakis for Acyclic Join-Aggregate Query

- Annotated Relations are functions mapping tuples to elements from Z

Annotation of a join result t: $w(t) = \prod_e w(\pi_e t)$

$R_1(x_1, x_2)$

x_1	x_2	w
a_1	b_1	2
a_2	b_1	3

$R_2(x_2, x_3)$

x_2	x_3	w
b_1	c_1	2
b_1	C_2	1

$R_3(x_1, x_3)$

x_1	x_3	w
a_1	c_1	1
a_1	c_2	3
a_2	c_2	3

$R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

x_1	x_2	x_3	w
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

- Count the full join size: $w(\cdot) = 1$

$\sum_{x_1, x_2, x_3} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

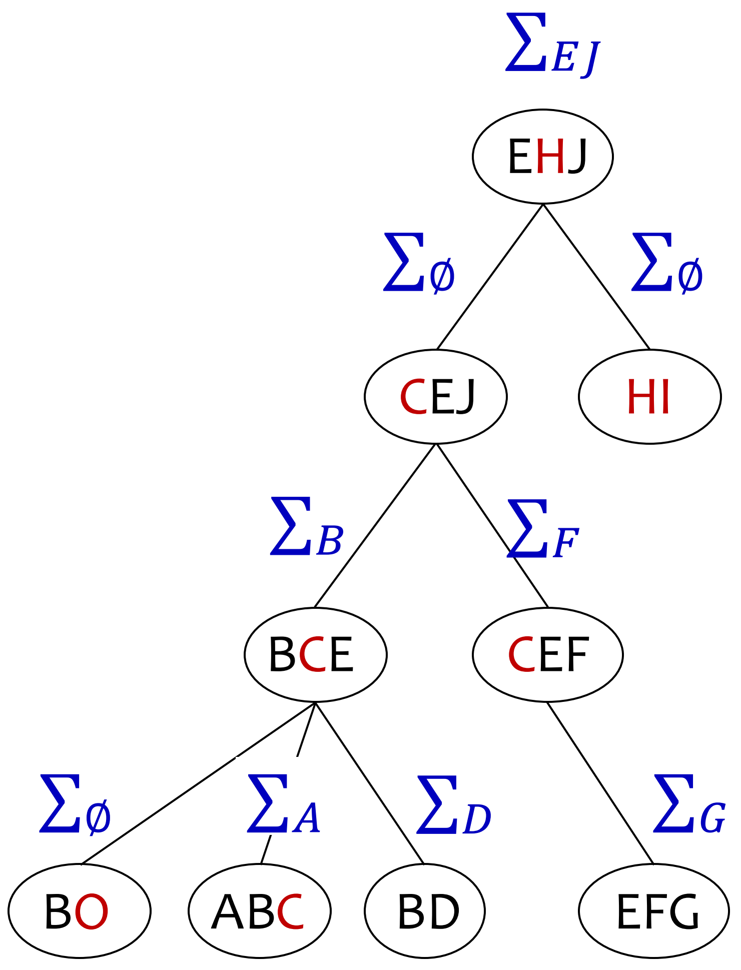
$\emptyset$	w
()	$4 + 6 + 9 = 19$

Annotation of a query result $t \in q(D)$:

$$w(t) = \sum_{t' \in \bowtie_e R_e : \pi_{\mathbf{y}} t' = t} w(t')$$

Yannakakis for Acyclic Join-Aggregate Query

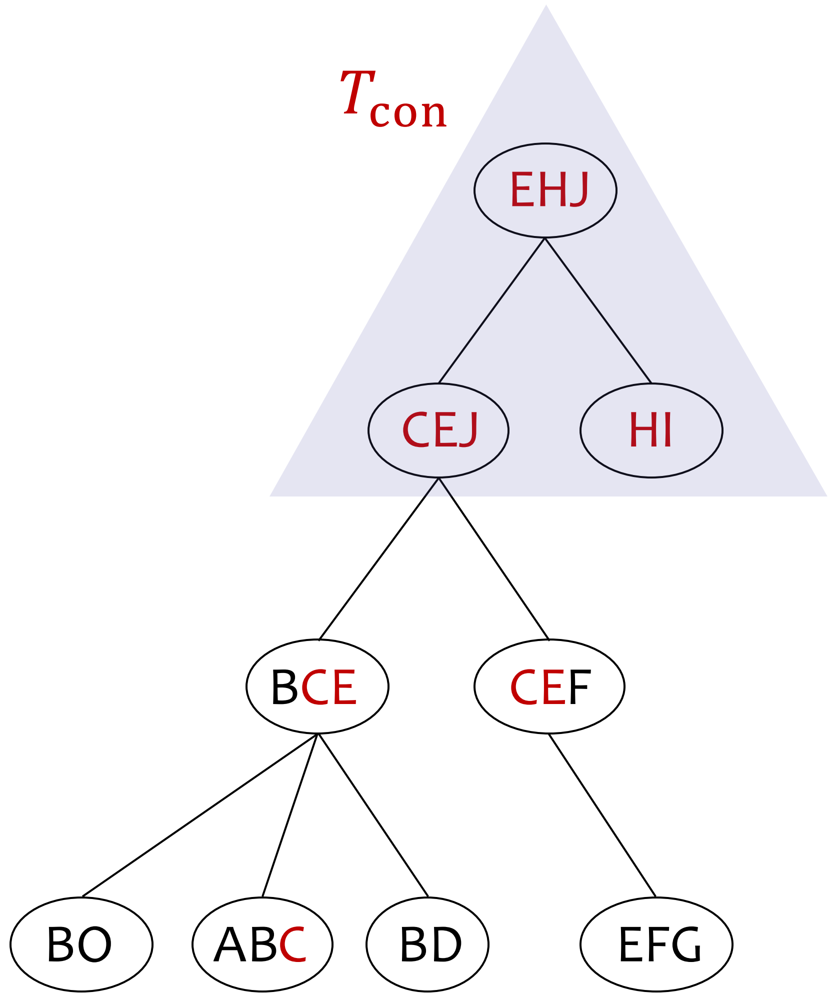
- Semi-join Reducers
- In a bottom-up phase:
  - Aggregate as early as possible!
  - For a non-output attribute, if it does not appear in any node above, then aggregate it away
- Same analysis as join-project queries!
- Semi-ring model usually does not have the inverse of the addition



output attributes

Free-Connex Query

- A join-project query Q is free-connex if it has a join tree T such that
  - There exists a connected subtree $T_{\text{con}} \subseteq T$ containing the root node r of T and the union of nodes in T_{con} is exactly the output attributes



output attributes

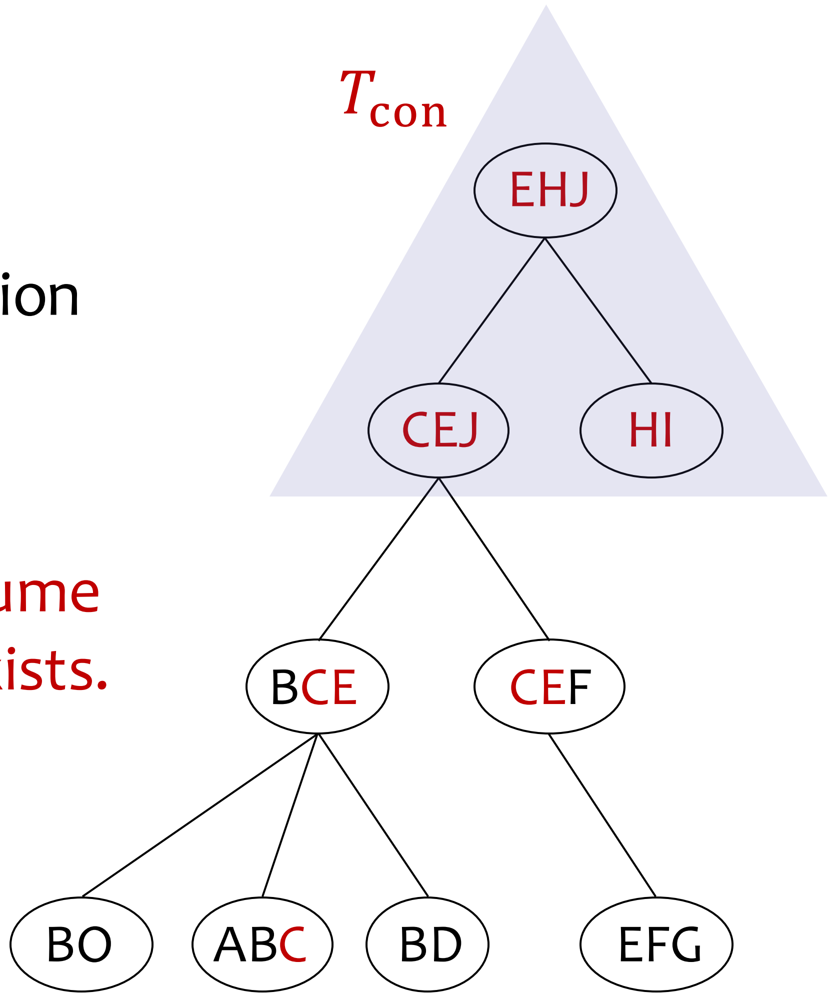
Free-Connex Query

- Let $\boldsymbol{y}$ be the set of output attributes
- Consider an arbitrary node e and its parent u
  - Let v be the attributes at node e after join-projection
  - Either $v \subseteq u$ or $v \subseteq \boldsymbol{y}$!

Case 1: $u \subseteq \boldsymbol{y}$. We have $v \subseteq \boldsymbol{y}$.

Case 2: $u - \boldsymbol{y} \neq \emptyset$. We have $v \subseteq u$. Suppose not, assume $\mathrm{A} \in v - u$. There must be $\mathrm{A} \in e \cap \boldsymbol{y}$. No such T_{con} exists.

- How to bound the intermediate join results?
  - If $v \subseteq u$, $|R_u \bowtie R_v| \leq |R_u| = N$
  - If $v \subseteq \boldsymbol{y}$, $|R_u \bowtie R_v| \leq |Q| = OUT$

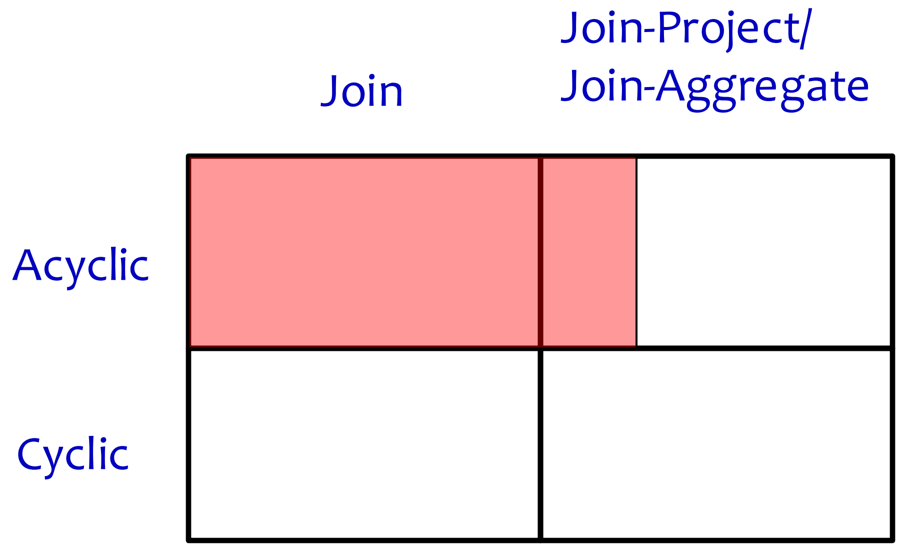


output attributes

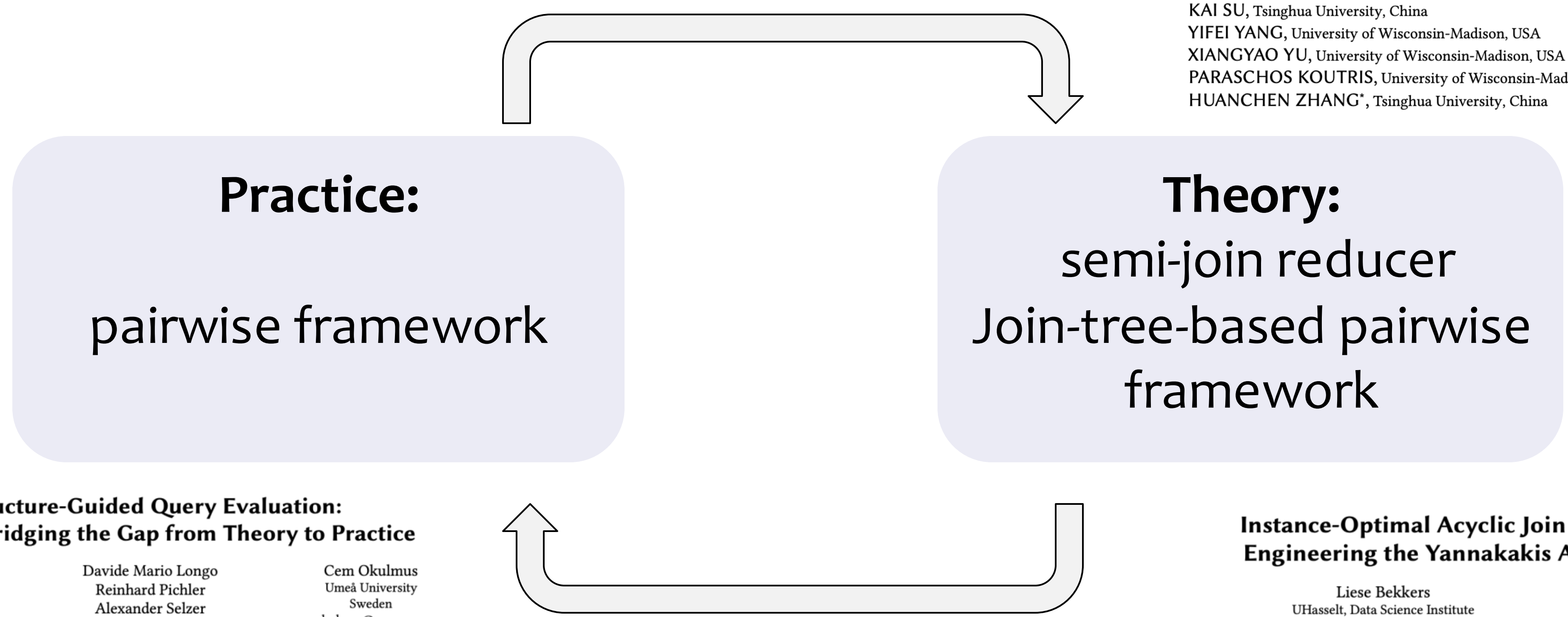
Summary of Yannakakis Algorithm

- Semi-join reducer: remove dangling tuples in $O(N)$ time

- Complexity:
  - Free-connex Join-Project or Join-Aggregate: $O(N + OUT)$
    - Acyclic Joins
  - Acyclic Join-Project or Join-Aggregate: $O(N \cdot OUT)$



Summary of Traditional Query Processing



Practice:

pairwise framework

Theory:

semi-join reducer

Join-tree-based pairwise framework

Debunking the Myth of Join Ordering: Toward Robust SQL Analytics

JUNYI ZHAO, Tsinghua University, China
KAI SU, Tsinghua University, China
YIFEI YANG, University of Wisconsin-Madison, USA
XIANGYAO YU, University of Wisconsin-Madison, USA
PARASCHOS KOUTRIS, University of Wisconsin-Madison, USA
HUANCHEN ZHANG[*], Tsinghua University, China

Structure-Guided Query Evaluation: Towards Bridging the Gap from Theory to Practice

Georg Gottlob
Matthias Lanzinger
University of Oxford
United Kingdom
georg.gottlob@cs.ox.ac.uk
matthias.lanzinger@cs.ox.ac.uk

Davide Mario Longo
Reinhard Pichler
Alexander Selzer
TU Wien
Austria
firstname.lastname@tuwien.ac.at

Cem Okulmus
Umeå University
Sweden
okulmus@cs.umu.se

Predicate Transfer: Efficient Pre-Filtering on Multi-Join Queries

Yifei Yang, Hangdong Zhao, Xiangyao Yu, Paraschos Koutris
University of Wisconsin-Madison
yyang673@wisc.edu,{hangdong,yxy,paris}@cs.wisc.edu

Yannakakis$^+$: Practical Acyclic Query Evaluation with Theoretical Guarantees

QICHEN WANG[*], Hong Kong Baptist University, Hong Kong SAR
BINGNAN CHEN[*], Hong Kong University of Science and Technology, Hong Kong SAR
BINYANG DAI, Hong Kong University of Science and Technology, Hong Kong SAR
KE YI, Hong Kong University of Science and Technology, Hong Kong SAR
FEIFEI LI, Alibaba Group, China
LIANG LIN, Alibaba Group, China

Instance-Optimal Acyclic Join Processing Without Regret: Engineering the Yannakakis Algorithm in Column Stores

Liese Bekkers
UHasselt, Data Science Institute
liese.bekkers@uhasselt.be

Frank Neven
UHasselt, Data Science Institute
frank.neven@uhasselt.be

Stijn Vansummeren
UHasselt, Data Science Institute
stijn.vansummeren@uhasselt.be

Yisu Remy Wang
University of California, Los Angeles
remywang@cs.ucla.edu