ICDE 2022 tutorial



Toward Responsive DBMS: Optimal Join Algorithms, Enumeration, Factorization, Ranking, and Dynamic Programming

Nikolaos Tziavelis, Wolfgang Gatterbauer, Mirek Riedewald

Northeastern University, Boston

Part 2 : Cycles and Tree Decompositions

Northeastern University Khoury College of Computer Sciences

Slides: https://northeastern-datalab.github.io/responsive-dbms-tutorial
DOI: https://doi.org/10.1109/ICDE53745.2022.00299
Data Lab: https://db.khoury.northeastern.edu

DATALAB @Northeastern

Outline tutorial

1: Introduction (Nikos) ~40min

2: Tree Decompositions (Mirek) ~20min

3: Acyclic Queries & Enumeration (Wolfgang) ~25min

BREAK

4: Factorization (Nikos) ~10min

5: Dynamic Programming & Semirings (Wolfgang) ~20min

6: Any-k or Ranked Enumeration (Nikos) ~35min

7. Decomposition of Comparison Predicates (Mirek) ~10min

8. Conclusion (Mirek) ~10min

Overview

- Focus here is on the structure of the join conditions
  - Acyclic join query: “easy”
  - Cyclic join query: hard
- Why are cyclic joins harder?
- How to deal with them: reduce to (union of) acyclic join queries on possibly larger relations

```
SELECT A1, A2, A3, A4        --Projection: all attributes
FROM R1, R2, R3, R4          --Joined relations
WHERE   --Join conditions: Ai = Aj
        R1.A1 = R2.A1 AND R1.A2 = R2.A2
        AND R2.A2 = R3.A2
        AND R2.A1 = R4.A1 AND R2.A2 = R4.A2
        --Selections: A Θ constant
        AND A4 < 1
```

Lower Bound for Any Query

- Need to read entire input at least once: $\Omega(\ell n)$
  - $\Omega(n)$ data complexity

Lower Bound for Any Query

- Need to read entire input at least once: $\Omega(\ell n)$
  - $\Omega(n)$ data complexity
- Need to output every result, each of size ℓ: $\Omega(\ell r)$
  - $\Omega(r)$ data complexity

Lower Bound for Any Query

- Need to read entire input at least once: $\Omega(\ell n)$
  - $\Omega(n)$ data complexity
- Need to output every result, each of size ℓ: $\Omega(\ell r)$
  - $\Omega(r)$ data complexity

- Together: $\Omega(n + r)$ time complexity to compute any CQ

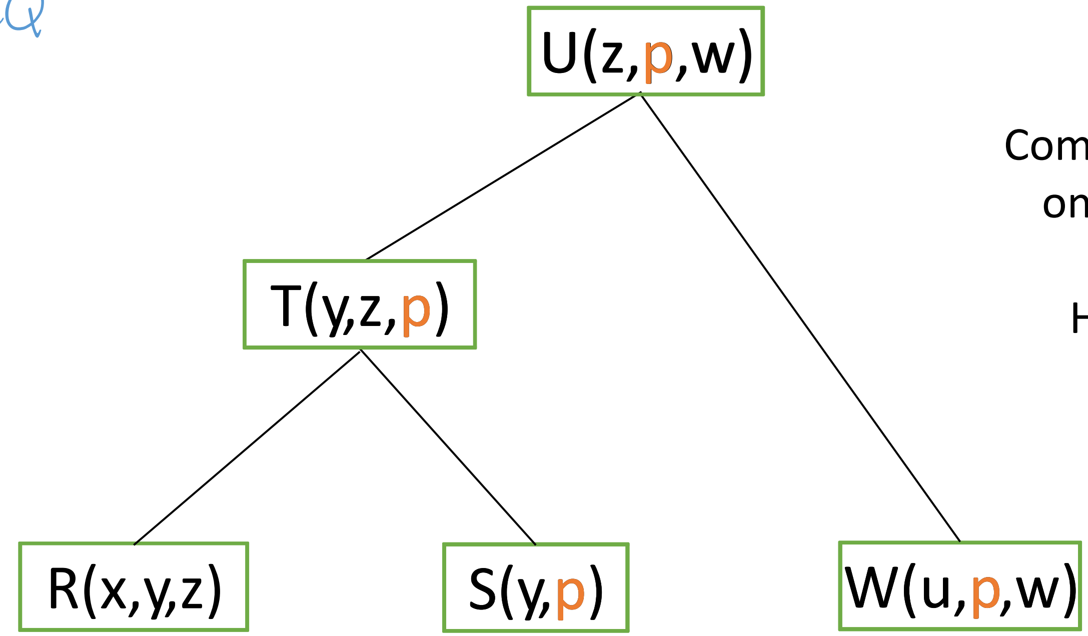
Acyclic queries and the Yannakakis Algorithm

- What is the key idea?
  - For acyclic queries (that do not require cyclic joins), we can remove in linear time all dangling tuples: those that are not part of any answer
  - This allows us to evaluate them very efficiently
  - The Yannakakis algorithm answers acyclic CQs in $O(n + r)$, which is optimal

How do we know whether a CQ does not require cyclic joins?

<u>Join Tree</u>

- Nodes: relations
- the nodes containing the same variable are connected

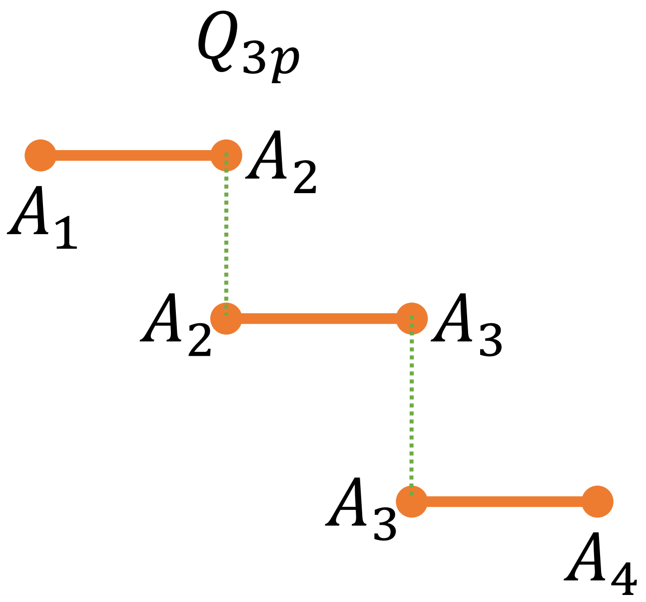


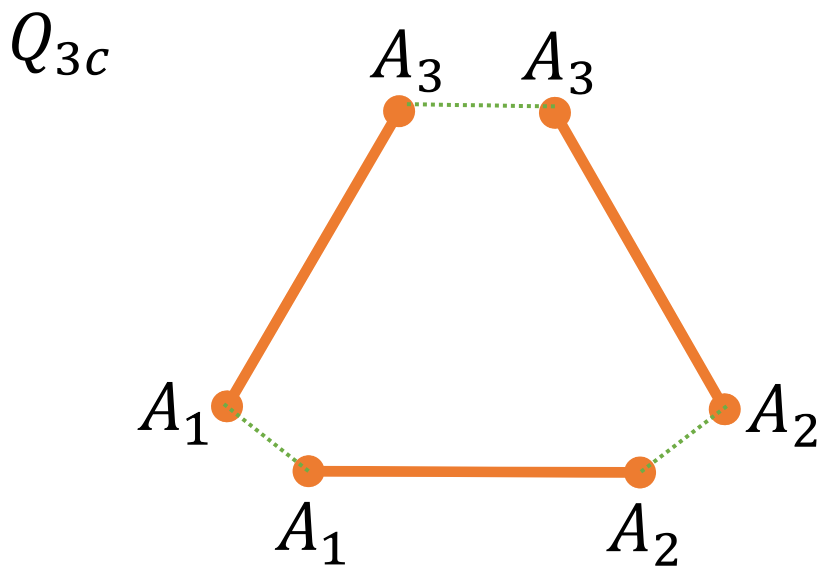
Compared to query plans: only partial join order.

Here T⋈R and T⋈S before T⋈U.

CQs with Cycles

- 3-path: $Q_{3p} = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4)$
- 3-cycle: $Q_{3c} = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_1)$

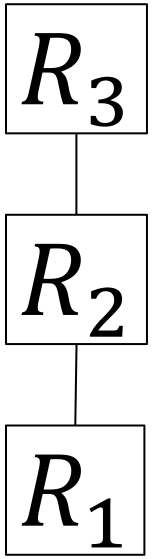




CQs with Cycles

- 3-path: $Q_{3p} = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4)$
- 3-cycle: $Q_{3c} = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_1)$
- Already semi-join reduced in the example

Join tree



R_1

A_1	A_2
1	1
2	1
…	…
n	1

R_2

A_2	A_3
1	1
1	2
…	…
1	n

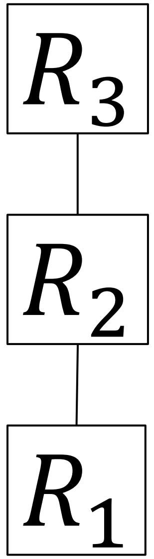
R_3

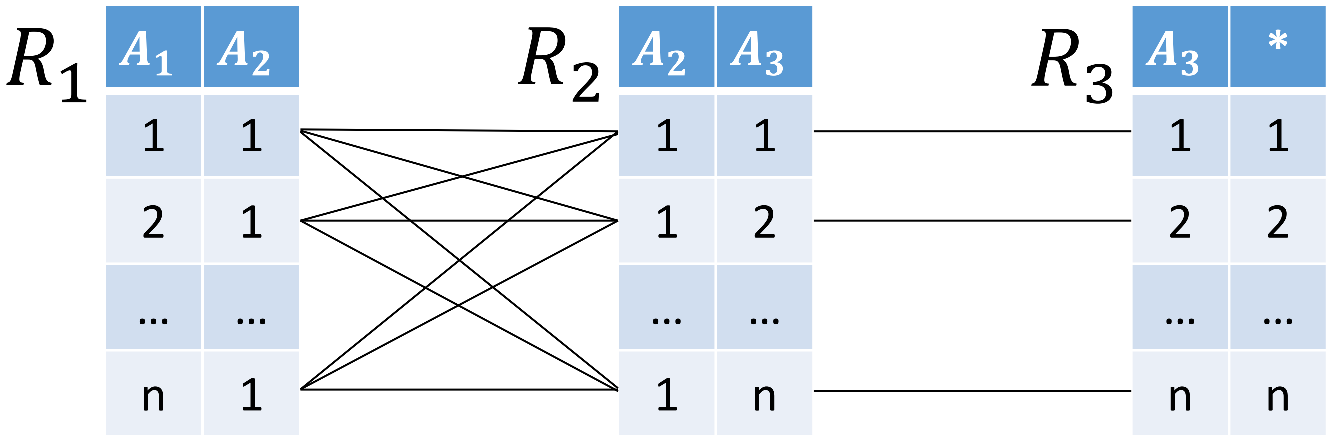
A_3	*
1	1
2	2
…	…
n	n

CQs with Cycles

- 3-path: $Q_{3p} = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4)$
- 3-cycle: $Q_{3c} = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_1)$
- For Q_{3p}, $\mathrm{r} = n^2$ and hence $O(n + r) = O(n^2)$
- For Q_{3c}, $\mathrm{r} = \mathrm{n}$ and hence $O(n + r) = O(n)$
- $R_1 \bowtie R_2$ produces n^2 intermediate results

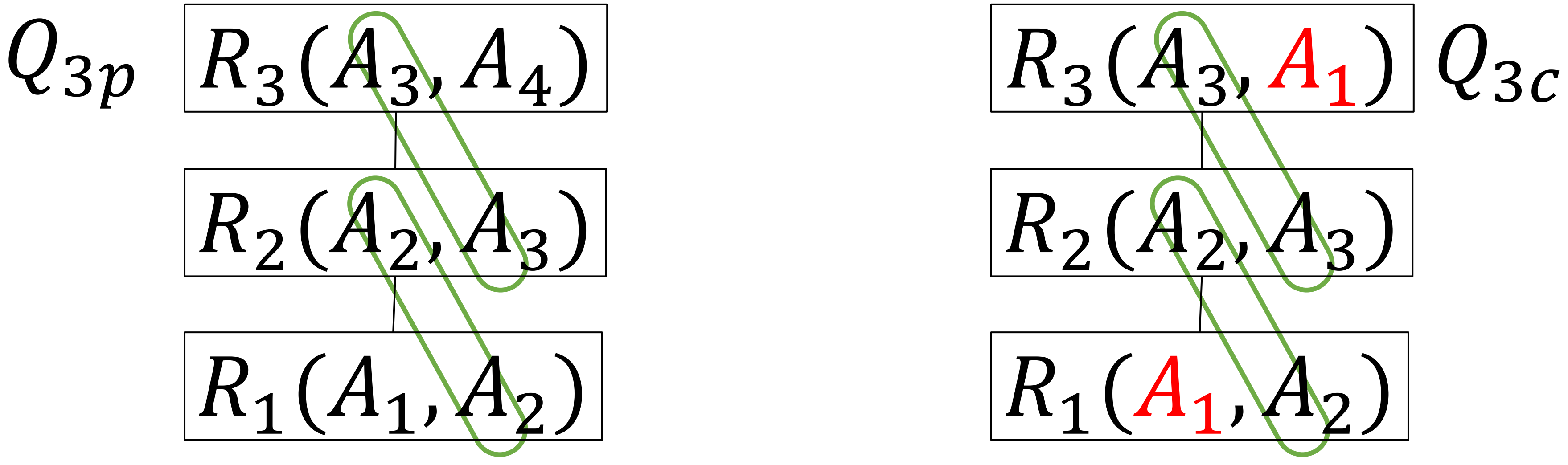
Join tree





What Went Wrong?

- The tree for the 3-cycle is not attribute-connected!
  - In the right tree, A_1 violates this property



Solutions for Cycles? Some Bad News

- Maybe we just need an algorithm that is better suited for cyclic CQs?

- Yes, but…

- … [Ngo+ 18]:
  - $\widetilde{O}(n + r)$ unattainable based on well-accepted complexity-theoretic assumptions

[Ngo+ 18] Ngo, Porat, Ré, Rudra. Worst-case optimal join algorithms. J. ACM'18 https://doi.org/10.1145/3180143

What Can Be Done?

- Worst-case-optimal (WCO) join algorithms [Veldhuizen 14, Ngo+ 14, Ngo+ 18]
- Instead of $\widetilde{O}(n + r)$, get
  $$\widetilde{O}(n + r_{WC}) = \widetilde{O}(r_{WC})$$
- r_{WC} = largest output of Q over any possible DB instance
  - Determined by the AGM bound[4]
  - Based on fractional edge cover of the join hypergraph
    - 3-cycle: $n^{1.5}$ vs naive upper bound n^3

[Veldhuizen 14] Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. ICDT'14 https://doi.org/10.5441/002/icdt.2014.13
[Ngo+ 14] Ngo, Re, Rudra. Skew strikes back: New developments in the theory of join algorithms. SIGMOD Rec.'14 https://doi.org/10.1145/2590989.2590991
[Ngo+ 18] Ngo, Porat, Ré, Rudra. Worst-case optimal join algorithms. J. ACM'18 https://doi.org/10.1145/3180143
[Atserias+ 13] Atserias, Grohe, Marx. Size bounds and query plans for relational joins. . SIAM J. Comput.'13 https://doi.org/10.1137/110859440

What Can Be Done?

- Worst-case-optimal (WCO) join algorithms [Veldhuizen 14, Ngo+ 14, Ngo+ 18]
- Instead of $\widetilde{O}(n + r)$, get
  $$\widetilde{O}(n + r_{\mathrm{WC}}) = \widetilde{O}(r_{\mathrm{WC}})$$
- r_{WC} = largest output of Q over any possible DB instance
  - Determined by the AGM bound[4]
  - Based on fractional edge cover of the join hypergraph
    - 3-cycle: $n^{1.5}$ vs naive upper bound n^3

- Hyper-tree decompositions
- Put more effort into pre-processing to avoid large intermediate results
  - Use WCO joins as sub-routine
- Goal: $\widetilde{O}(n^d + r)$ for smallest d possible
  - $\widetilde{O}(n^d)$ captures pre-processing cost
  - $d = 1$ for acyclic CQ

[Veldhuizen 14] Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. ICDT'14 https://doi.org/10.5441/002/icdt.2014.13
[Ngo+ 14] Ngo, Re, Rudra. Skew strikes back: New developments in the theory of join algorithms. SIGMOD Rec.'14 https://doi.org/10.1145/2590989.2590991
[Ngo+ 18] Ngo, Porat, Ré, Rudra. Worst-case optimal join algorithms. J. ACM'18 https://doi.org/10.1145/3180143
[Atserias+ 13] Atserias, Grohe, Marx. Size bounds and query plans for relational joins. . SIAM J. Comput.'13 https://doi.org/10.1137/110859440

WCO vs Hyper-tree Decompositions

Query	Output size r	WCO complexity	HT decomposition complexity
3-cycle	Small: $O(1), O(n)$	$\mathbf{O(n^{1.5})}$	$O(n^{1.5} + 1 \dots n) = \mathbf{O(n^{1.5})}$
3-cycle	$O(n^{1.5})$	$\mathbf{O(n^{1.5})}$	$O(n^{1.5} + n^{1.5}) = \mathbf{O(n^{1.5})}$
4-cycle	Small: $O(1), O(n)$	$\mathbf{O(n^{2})}$	$O(n^{1.5} + 1 \dots n) = \mathbf{O(n^{1.5})}$
4-cycle	$O(n^{2})$	$\mathbf{O(n^{2})}$	$O(n^{1.5} + n^{2}) = \mathbf{O(n^{2})}$
6-cycle	Small: $O(1), O(n)$	$\mathbf{O(n^{3})}$	$O\left(n^{5/3} + 1 \dots n\right) = \mathbf{O(n^{5/3})}$
6-cycle	$O(n^{3})$	$\mathbf{O(n^{3})}$	$O\left(n^{5/3} + n^{3}\right) = \mathbf{O(n^{3})}$
$2l$-cycle	Small: $O(1), O(n)$	$\mathbf{O(n^{\ell})}$	$O\left(n^{2-1/\ell} + 1 \dots n\right) = O\left(n^{2-1/\ell}\right) = \boldsymbol{o(n^{2})}$

WCO vs Hyper-tree Decompositions

Query	Output size r	WCO complexity	HT decomposition complexity
3-cycle	Small: $O(1), O(n)$	$\mathbf{O(n^{1.5})}$	$O(n^{1.5} + 1 \ldots n) = \mathbf{O(n^{1.5})}$
3-cycle	$O(n^{1.5})$	$\mathbf{O(n^{1.5})}$	$O(n^{1.5} + n^{1.5}) = \mathbf{O(n^{1.5})}$
4-cycle	Small: $O(1), O(n)$	$O(n^2)$	$O(n^{1.5} + 1 \ldots n) = \mathbf{O(n^{1.5})}$
4-cycle	$O(n^2)$	$\mathbf{O(n^2)}$	$O(n^{1.5} + n^2) = \mathbf{O(n^2)}$
6-cycle	Small: $O(1), O(n)$	$O(n^3)$	$O\left(n^{5/3} + 1 \ldots n\right) = \mathbf{O(n^{5/3})}$
6-cycle	$O(n^3)$	$\mathbf{O(n^3)}$	$O\left(n^{5/3} + n^3\right) = \mathbf{O(n^3)}$
$2l$-cycle	Small: $O(1), O(n)$	$O(n^{\ell})$	$O\left(n^{2-1/\ell} + 1 \ldots n\right) = O\left(n^{2-1/\ell}\right) = \boldsymbol{o(n^2)}$

Hyper-tree decompositions never lose. This is true in general. Does that mean we do not need WCO joins at all?

WCO vs Hyper-tree Decompositions

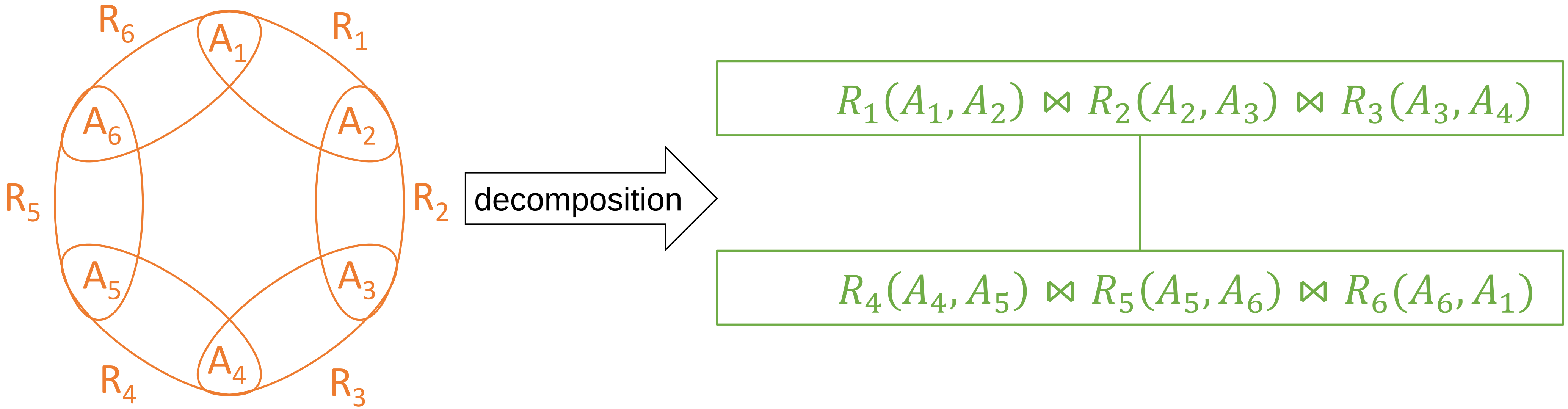
Query	Output size r	WCO complexity	HT decomposition complexity
3-cycle	Small: $O(1), O(n)$	$\mathbf{O(n^{1.5})}$	$O(n^{1.5} + 1 \ldots n) = \mathbf{O(n^{1.5})}$
3-cycle	$O(n^{1.5})$	$\mathbf{O(n^{1.5})}$	$O(n^{1.5} + n^{1.5}) = \mathbf{O(n^{1.5})}$
4-cycle	Small: $O(1), O(n)$	$O(n^2)$	$O(n^{1.5} + 1 \ldots n) = \mathbf{O(n^{1.5})}$
4-cycle	$O(n^2)$	$\mathbf{O(n^2)}$	$O(n^{1.5} + n^2) = \mathbf{O(n^2)}$
6-cycle	Small: $O(1), O(n)$	$O(n^3)$	$O\left(n^{5/3} + 1 \ldots n\right) = \mathbf{O(n^{5/3})}$
6-cycle	$O(n^3)$	$\mathbf{O(n^3)}$	$O\left(n^{5/3} + n^3\right) = \mathbf{O(n^3)}$
$2l$-cycle	Small: $O(1), O(n)$	$O(n^{\ell})$	$O\left(n^{2-1/\ell} + 1 \ldots n\right) = O\left(n^{2-1/\ell}\right) = \boldsymbol{o(n^2)}$

Hyper-tree decompositions never lose. This is true in general. Does that mean we do not need WCO joins at all?

No. WCO joins are used as a subroutine by the HT decomposition approach!

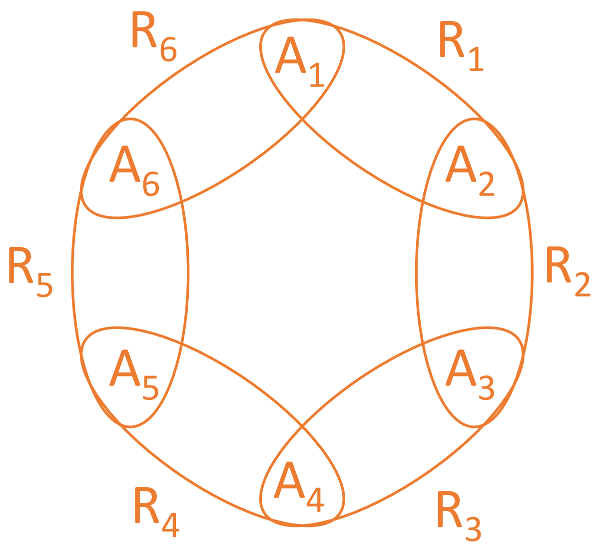
Main Idea of Tree Decompositions

1. Convert cyclic CQ to a rooted tree-shaped CQ



Main Idea of Tree Decompositions

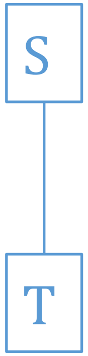
1. Convert cyclic CQ to a rooted tree-shaped CQ
2. Materialize all tree nodes ("bags") using a WCO join algorithm



decomposition

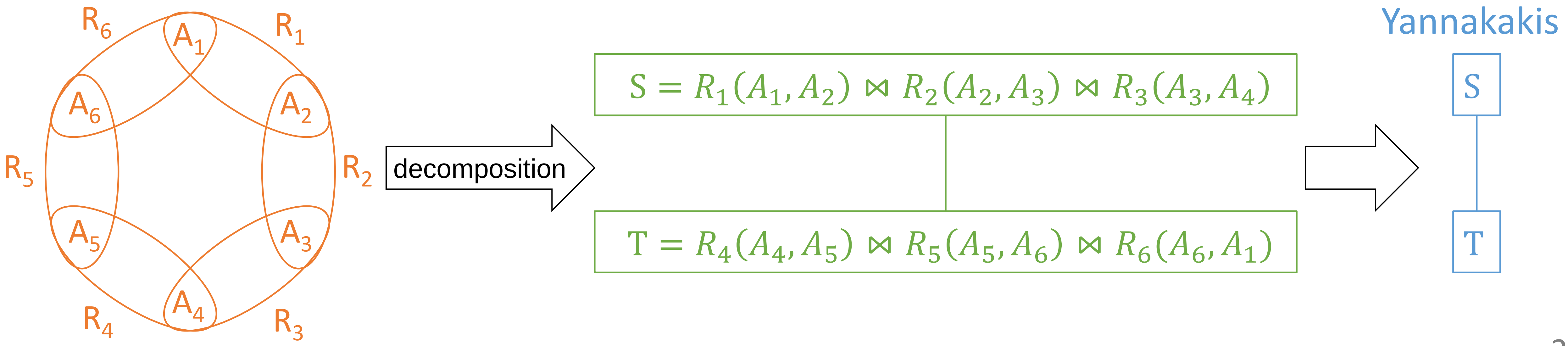
$$S = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4)$$

$$T = R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$



Main Idea of Tree Decompositions

1. Convert cyclic CQ to a rooted tree-shaped CQ
2. Materialize all tree nodes (“bags”) using a WCO join algorithm
3. Apply Yannakakis algorithm on the tree
   - Achieves $O(x + r)$ where x is the size of the largest bag



Tree Decomposition Intuition

$$Q_{6c}(A_1, \ldots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

Tree Decomposition Intuition

$$Q_{6c}(A_1, \dots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Tree Decomposition Intuition

$$Q_{6c}(A_1, \ldots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \\ \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \\ \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

What is the simplest tree with these properties?

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Tree Decomposition Intuition

$$Q_{6c}(A_1, \ldots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

$\mathcal{T}_1$

$R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$ $R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Bag materialization costs $O(n^3)$ (AGM bound)

Tree Decomposition Intuition

$$Q_{6c}(A_1, \ldots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

$\mathcal{T}_1$

$R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$ $R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Can we do better?

Bag materialization costs $O(n^3)$ (AGM bound)

Tree Decomposition Intuition

$$Q_{6c}(A_1,\ldots,A_6) = R_1(A_1,A_2) \bowtie R_2(A_2,A_3) \bowtie R_3(A_3,A_4) \bowtie R_4(A_4,A_5) \bowtie R_5(A_5,A_6) \bowtie R_6(A_6,A_1)$$

$\mathcal{T}_2$

$R_1(A_1,A_2), R_2(A_2,A_3), R_3(A_3,A_4)$

$R_4(A_4,A_5), R_5(A_5,A_6), R_6(A_6,A_1)$

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Bag materialization costs $O(n^2)$ (AGM bound)

Tree Decomposition Intuition

$$Q_{6c}(A_1, \ldots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

$\mathcal{T}_2$

$R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$

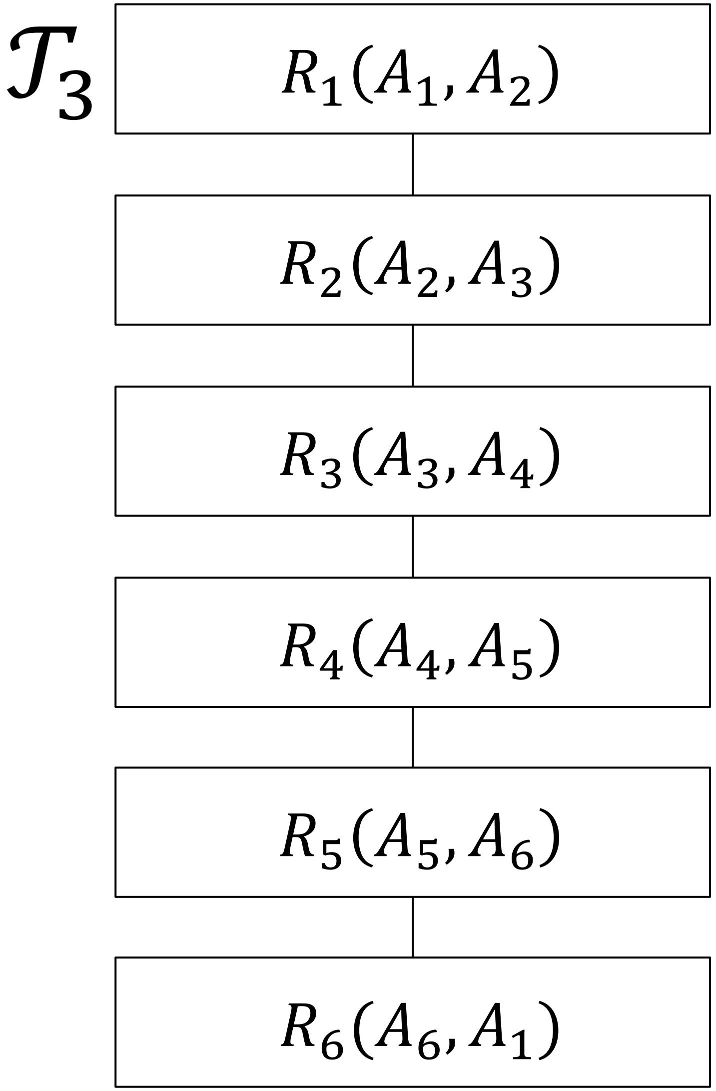
Can we "slim down" the bags even more?

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Bag materialization costs $O(n^2)$ (AGM bound)

Tree Decomposition Intuition



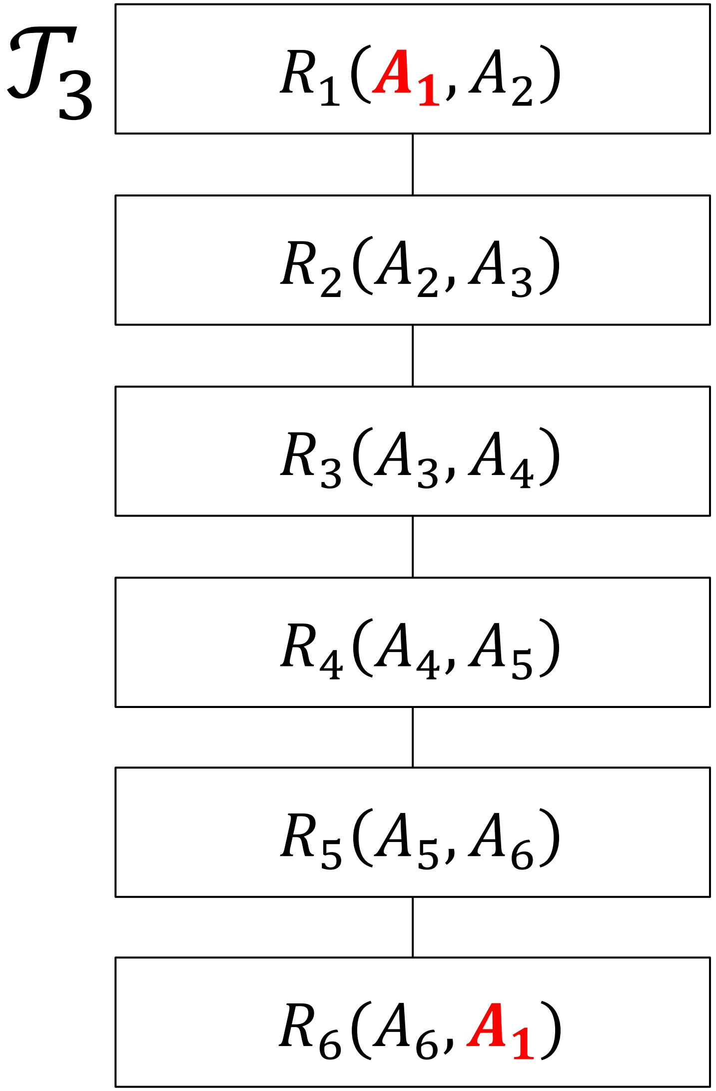
$O(n)$ bag materialization…?

$$Q_{6c}(A_1, \dots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Tree Decomposition Intuition

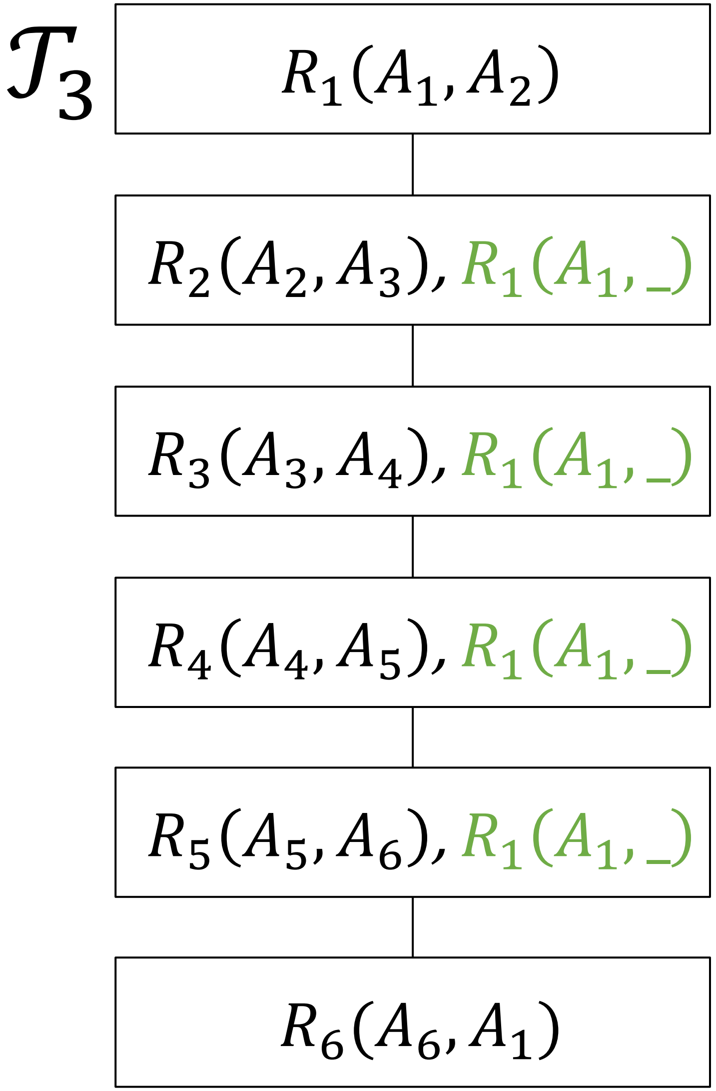


$$Q_{6c}(A_1, \ldots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

Tree Decomposition Intuition



$$Q_{6c}(A_1, \ldots, A_6) = R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4) \bowtie R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$

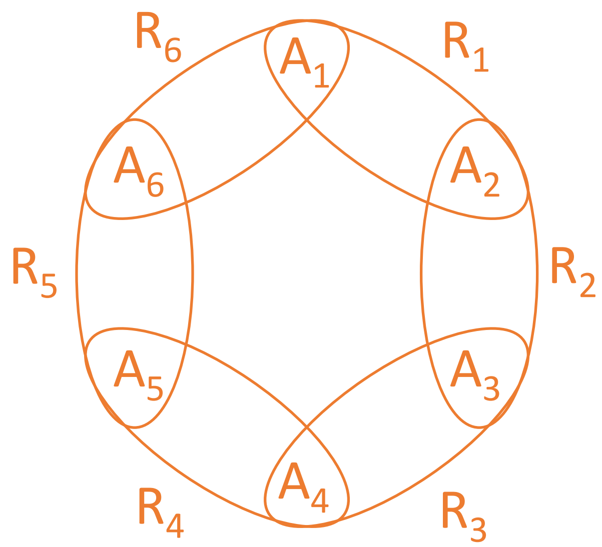
Every relation appearing in the query is covered by a bag (tree node)

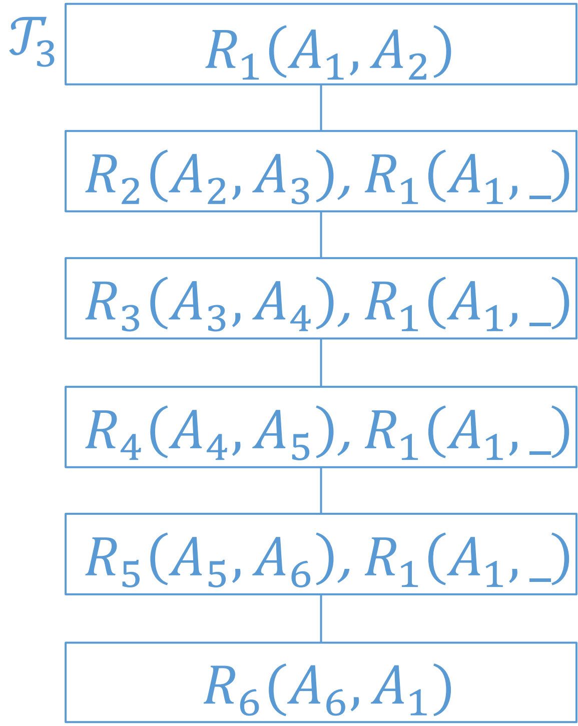
For each attribute, the bags containing it are connected

$O(n \cdot |\pi_{A_1}(R_1)|)$ bag materialization: still $O(n^2)$

Tree Decomposition: Formal Definition

- Given: hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$
  - $\mathcal{V}$: attributes
    - E.g., $\{A_1, A_2, A_3, A_4, A_5, A_6\}$
  - $\mathcal{E}$: relations
    - E.g., R_3 is hyperedge (A_3, A_4)





- A tree decomposition of $\mathcal{H}$ is a pair $(\mathcal{T}, \chi)$ where
  - $\mathcal{T} = (V(\mathcal{T}), E(\mathcal{T}))$ is a tree
  - $\chi: V(\mathcal{T}) \to 2^{\mathcal{V}}$ assigns a bag $\chi(v)$ to each tree node v such that
    - Each hyperedge $F \in \mathcal{E}$ is covered, i.e., $\forall F \in \mathcal{E}: \exists v \in V(\mathcal{T}):\ F \subseteq \chi(v)$
    - For each $u \in \mathcal{V}$, the bags containing u are connected

[Khamis, Ngo, Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? PODS'17]
https://doi.org/10.1145/3034786.3056105

Tree-Decomposition Properties

- Query has multiple decompositions—which is best?

Tree-Decomposition Properties

- Query has multiple decompositions—which is best?

- Consider a tree with $\mathrm{O}(\ell)$ nodes, each materialized using *WCO join*
  - Size of bag i is $\mathrm{O}(n^{d_i})$ for some $d_i \geq 1$ (AGM bound)
  - Fractional hypertree width (fhw) $d = \max_i d_i$ [Grohe+ 14]
  - Total bag-materialization cost: $\mathrm{O}(n^d)$
  - Size of a materialized bag: $\mathrm{O}(n^d)$
  - Resulting cost for Yannakakis algorithm on materialized tree: $\mathrm{O}(n^d + r)$

[Grohe+ 14] Grohe and Marx. Constraint solving via fractional edge covers. ACM TALG'14. https://doi.org/10.1145/2636918

Who Wins?

$\mathcal{T}_3$

$R_1(A_1, A_2)$

$R_2(A_2, A_3), R_1(A_1, \_)$ $O(n^2)$

$R_3(A_3, A_4), R_1(A_1, \_)$

$R_4(A_4, A_5), R_1(A_1, \_)$

$R_5(A_5, A_6), R_1(A_1, \_)$

$R_6(A_6, A_1)$

$\mathcal{T}_1$

$R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$
$R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$ $O(n^3)$

$\mathcal{T}_2$

$R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$ $O(n^2)$

$R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$

A Closer Look

- $\mathcal{T}_1$ loses, because it does not decompose the query

A Closer Look

- $\mathcal{T}_1$ loses, because it does not decompose the query

- Are $\mathcal{T}_2$ and $\mathcal{T}_3$ really equally good?
  - In $\mathcal{T}_2$, bag computation requires joining 3 relations
  - In $\mathcal{T}_3$, bag computation requires joining 2 relations
    - One of them is just the set of distinct A_1-values in R_1

A Closer Look

- $\mathcal{T}_1$ loses, because it does not decompose the query

- Are $\mathcal{T}_2$ and $\mathcal{T}_3$ really equally good?
  - In $\mathcal{T}_2$, bag computation requires joining 3 relations
  - In $\mathcal{T}_3$, bag computation requires joining 2 relations
    - One of them is just the set of distinct A_1-values in R_1

- What if there are “few” distinct A_1-values in R_1, e.g., $O(n^{2/3})$ instead of $O(n)$?

Who Wins?

Relations	Size
$R_1(A_1, A_2)$	$O(n)$
$R_2(A_2, A_3), R_1(A_1, \_)$	$O(n^{5/3})$
$R_3(A_3, A_4), R_1(A_1, \_)$	$O(n^{5/3})$
$R_4(A_4, A_5), R_1(A_1, \_)$	$O(n^{5/3})$
$R_5(A_5, A_6), R_1(A_1, \_)$	$O(n^{5/3})$
$R_6(A_6, A_1)$	$O(n)$

$\mathcal{T}_3$

Degree constraint: $|\pi_{A_1}(R_1)| \leq n^{2/3}$

"The number of distinct A_1 values in R_1 is at most $n^{2/3}$"

Who Wins?

Degree constraint: $|\pi_{A_1}(R_1)| \leq n^{2/3}$

$R_1(A_1, A_2)$ $O(n)$

$R_2(A_2, A_3), R_1(A_1, \_)$ $O(n^{5/3})$

$R_3(A_3, A_4), R_1(A_1, \_)$ $O(n^{5/3})$

$R_4(A_4, A_5), R_1(A_1, \_)$ $O(n^{5/3})$

$R_5(A_5, A_6), R_1(A_1, \_)$ $O(n^{5/3})$

$R_6(A_6, A_1)$ $O(n)$

$\mathcal{T}_3$

$O(n^2)$ $R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$

$O(n^2)$ $R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$

$\mathcal{T}_2$

Could $\mathcal{T}_2$ Win?

- Consider bag $R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4)$ in $\mathcal{T}_2$

- What if each R_1-tuple joins with only “a few” R_2-tuples?
- What if each R_2-tuple joins with only “a few” R_3-tuples?

- What if “a few” was at most $n^{1/3}$?

Who Wins Now?

Degree constraint: $\forall i \in \{2,3,5,6\}$:

$$\forall j\colon \left|\pi_{A_{(i+1) \bmod 6}} \sigma_{A_i=j}(R_i)\right| \le n^{1/3}$$

"Each tuple from R_1 joins with at most $n^{1/3}$ tuples from R_2 and each tuple from R_2 joins with at most $n^{1/3}$ tuples from R_3. The same holds analogously for R_4, R_5, and R_6."

Who Wins Now?

Degree constraint: $\forall i \in \{2,3,5,6\}$:

$\forall j$: $\left|\pi_{A_{i+1}} \sigma_{A_i = j}(R_i)\right| \leq n^{1/3}$

$O(n^{5/3})$ $R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$

$O(n^{5/3})$ $R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$

$\mathcal{T}_2$

Who Wins Now?

$R_1(A_1, A_2)$ $O(n)$

$R_2(A_2, A_3), R_1(A_1, \_)$ $O(n^2)$

$R_3(A_3, A_4), R_1(A_1, \_)$ $O(n^2)$

$R_4(A_4, A_5), R_1(A_1, \_)$ $O(n^2)$

$R_5(A_5, A_6), R_1(A_1, \_)$ $O(n^2)$

$R_6(A_6, A_1)$ $O(n)$

$\mathcal{T}_3$

Degree constraint: $\forall i \in \{2,3,5,6\}$: $\forall j$: $\left|\pi_{A_{i+1}} \sigma_{A_i = j}(R_i)\right| \leq n^{1/3}$

$O(n^{5/3})$ $R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$

$O(n^{5/3})$ $R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$

$\mathcal{T}_2$

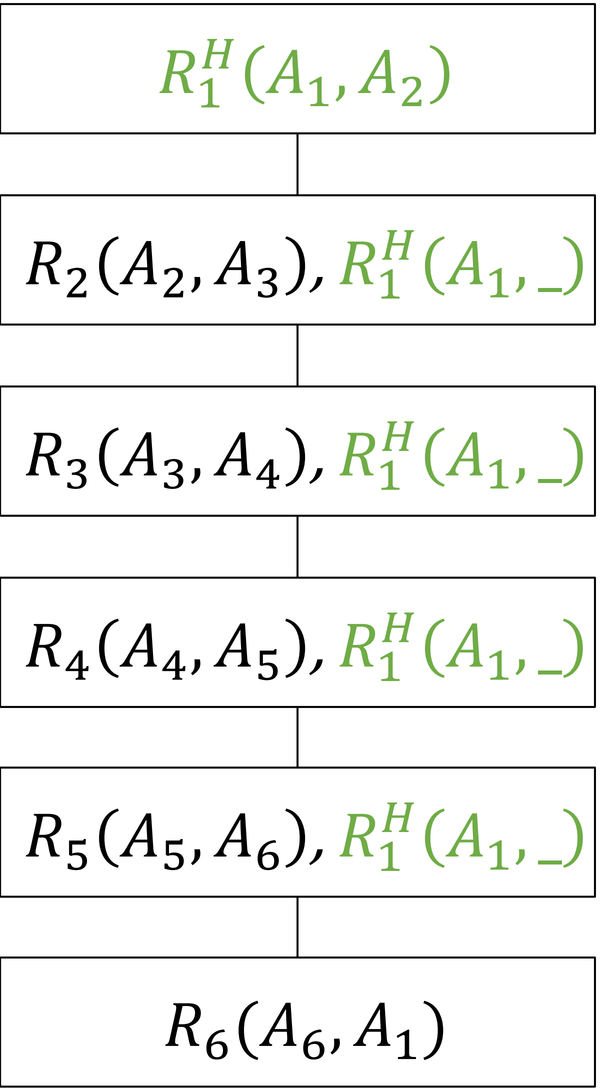
Best of Both Worlds

- Depending on the degree constraints that hold for a DB instance, we may sometimes prefer $\mathcal{T}_2$ and sometimes $\mathcal{T}_3$

- What if we used *both*? [Alon+ 97, Marx 13]
  - Intuition: each decomposition is a different query "plan"
    - Query output = union of individual plans' results
  - Decide for each input tuple to which plan(s) to send it
  - Main idea: split each input relation into heavy and light
    - Goal: enforce desirable degree constraints for each tree

[Alon+ 97] Alon, Yuster, Zwick. Finding and counting given length cycles. Algorithmica'97 https://doi.org/10.1007/BF02523189
[Marx 13] Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. J.ACM'13 https://doi.org/10.1145/2535926

Multiple Plans: Plan 1



$\mathcal{T}_3$

R_1^H: contains all tuples whose A_1-values occur more than $n^{1/3}$ times (fewer than $n^{2/3}$ such A_1-values exist)

Multiple Plans: Plan 1

Degree constraint: $|\pi_{A_1}(R_1^H)| \leq n^{2/3}$

Bag	Cost
$R_1^H(A_1, A_2)$	$O(n)$
$R_2(A_2, A_3), R_1^H(A_1, \_)$	$O(n^{5/3})$
$R_3(A_3, A_4), R_1^H(A_1, \_)$	$O(n^{5/3})$
$R_4(A_4, A_5), R_1^H(A_1, \_)$	$O(n^{5/3})$
$R_5(A_5, A_6), R_1^H(A_1, \_)$	$O(n^{5/3})$
$R_6(A_6, A_1)$	$O(n)$

R_1^H: contains all tuples whose A_1-values occur more than $n^{1/3}$ times (fewer than $n^{2/3}$ such A_1-values exist)

$\mathcal{T}_3$: computes $R_1^H \bowtie R_2 \bowtie \cdots \bowtie R_6$

More Plans

- Note that
  - $Q_{6c} = R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6$ together with
  - $R_1^L = R_1 \setminus R_1^H$
- implies that Q_{6c} is the union of
  - $R_1^H \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6$ and
  - $R_1^L \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6$

- To compute the latter, apply the same trick to R_2

Multiple Plans: Plan 2

Degree constraint: $\left|\pi_{A_2}(R_2^H)\right| \leq n^{2/3}$

$R_2^H(A_2, A_3)$

$R_3(A_3, A_4), R_2^H(A_2, \_)$

$R_4(A_4, A_5), R_2^H(A_2, \_)$

$R_5(A_5, A_6), R_2^H(A_2, \_)$

$R_6(A_6, A_1), R_2^H(A_2, \_)$

$R_1^L(A_1, A_2)$

R_2^H: contains all tuples whose A_2-values occur more than $n^{1/3}$ times (fewer than $n^{2/3}$ such A_2-values exist)

$R_2^L = R_2 \setminus R_2^H$

$\mathcal{T}_3$: computes $R_1^L \bowtie R_2^H \bowtie R_3 \bowtie \cdots \bowtie R_6$

Plans 3 to 6

- Plans discussed so far
  - $R_1^H \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6$
  - $R_1^L \bowtie R_2^H \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6$
- Continue analogously to compute
  - $R_1^L \bowtie R_2^L \bowtie R_3^H \bowtie R_4 \bowtie R_5 \bowtie R_6$
  - $R_1^L \bowtie R_2^L \bowtie R_3^L \bowtie R_4^H \bowtie R_5 \bowtie R_6$
  - $R_1^L \bowtie R_2^L \bowtie R_3^L \bowtie R_4^L \bowtie R_5^H \bowtie R_6$
  - $R_1^L \bowtie R_2^L \bowtie R_3^L \bowtie R_4^L \bowtie R_5^L \bowtie R_6^H$

- What is missing?

The 7-th Plan

- Join all light-only partitions with each other:
  - $R_1^L \bowtie R_2^L \bowtie R_3^L \bowtie R_4^L \bowtie R_5^L \bowtie R_6^L$

- Input now satisfies the other degree constraint:
  - $\forall i \in \{2,3,5,6\}: \forall j: \left|\pi_{A_{i+1}} \sigma_{A_i=j}(R_i)\right| \leq n^{1/3}$

- Use decomposition $\mathcal{T}_2$ for it!

Analysis and Discussion

- Rewrite 6-cycle into 7 sub-queries
  - Six of them use $\mathcal{T}_3$, copying the heavy attribute to intermediate bags
  - One uses $\mathcal{T}_2$ on the all-light case
- Analysis
  - Assigning input tuples to subqueries: $O(n)$
  - Bag materialization: $O(n^{5/3})$
  - Bag size: $O(n^{5/3})$
- Running Yannakakis on each of the 7 trees takes $O\left(n^{5/3} + r\right)$
  - Beats single-tree complexity $O(n^2 + r)$ and WCO-join complexity $O(n^3)$

Tree Decompositions: The Big Picture

- Reduce hard cyclic join to (union of) acyclic join(s)
  - Cyclic join on input of size $O(n)$ becomes acyclic join on "bags"
  - Bags are of size $O(n^d)$, each materialized using WCO join algorithm
  - Width d depends on AGM bound and "how close to a tree" the cyclic query is, e.g., $d = 1$ for acyclic join
  - Finding the optimal width and achieving it are research challenges

- Remainder of the tutorial: focus on acyclic joins
  - Next: Yannakakis algorithm