Part 2: Optimal Join Algorithms
Outline tutorial

• Part 1: Top-\(k\) (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
  – Lower Bound and the Yannakakis Algorithm
  – Problems Caused by Cycles
  – Tree Decompositions
  – Summary and Further Reading
• Part 3: Ranked enumeration over joins (Nikolaos): ~40min
Basic Terminology and Assumptions

• Terminology
  - Full conjunctive query (CQ)
    • Natural join of $l$ relations with $O(n)$ tuples each
    • E.g.: $Q(A_1, A_2, A_3, A_4) = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4)$
    • Any selections comparing attributes to constants, e.g., $A_4 < 1$
  - Query size: $O(l)$
  - Output cardinality: $r$

• Assumptions
  - No pre-computed data structures such as indexes, sorted representation, materialized views
Complexity Notation

- Standard $O$ and $\Omega$ notation for time and memory complexity in the RAM model of computation
- Common practice: focus on data complexity
  - We care about scalability in data size
    - Treat query size $l$ as a constant
    - E.g., $O(f(l) \cdot n^{f(l)} + (\log n)^{f(l)} \cdot r)$ simplifies to $O(n^{f(l)} + (\log n)^{f(l)} \cdot r)$
Complexity Notation

- Standard O and Ω notation for time and memory complexity in the RAM model of computation
- Common practice: focus on data complexity
  - We care about scalability in data size
    - Treat query size \( l \) as a constant
    - E.g., \( O\left( f(l) \cdot n^{f(l)} + (\log n)^{f(l)} \cdot r \right) \) simplifies to \( O\left( n^{f(l)} + (\log n)^{f(l)} \cdot r \right) \)
- We mostly use \( \tilde{O} \)-notation (soft-O) data complexity
  - Abstracts away polylog factors in input size that clutter formulas
  - E.g., \( O\left( n^{f(l)} + (\log n)^{f(l)} \cdot r \right) \) further simplifies to \( \tilde{O}\left( n^{f(l)} + r \right) \)
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
  – Lower Bound and the Yannakakis Algorithm
  – Problems Caused by Cycles
  – Tree Decompositions
  – Summary and Further Reading
• Part 3: Ranked enumeration over joins (Nikolaos): ~40min
Lower Bound for Any Query

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

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

- Amazingly, the Yannakakis algorithm essentially matches the lower bound for acyclic CQs
  - Time complexity $\tilde{O}(n + r)$
Yannakakis Algorithm

- Given: acyclic conjunctive query Q as a rooted join tree

- Step 1: **semi-join reduction** (two sweeps)
  - Semi-join reduction sweep from the leaves to root
  - Semi-join reduction sweep from root to the leaves

- Step 2: use the **join tree as the query plan**
  - Compute the joins bottom up, with early projections

[Mihalis Yannakakis. Algorithms for acyclic database schemes. VLDB’81]  [https://dl.acm.org/doi/10.5555/1286831.1286840]
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4) \]
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \Join R_2(A_1, A_2, A_3) \Join R_3(A_2) \Join R_4(A_1, A_2, A_4) \]

1. Bottom-up traversal (semi-joins)
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4) \]

1. Bottom-up traversal (semi-joins)
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4) \]

1. Bottom-up traversal (semi-joins)
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4) \]

1. Bottom-up traversal (semi-joins)
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4) \]

1. Bottom-up traversal (semi-joins)
Yannakakis Algorithm – Example

$$Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4)$$

1. Bottom-up traversal (semi-joins)
2. Top-down traversal (semi-joins)
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4) \]

1. Bottom-up traversal (semi-joins)
2. Top-down traversal (semi-joins)
3. Join bottom-up
Yannakakis Algorithm – Example

\[ Q = R_1(A_1, A_2) \bowtie R_2(A_1, A_2, A_3) \bowtie R_3(A_2) \bowtie R_4(A_1, A_2, A_4) \]

1. Bottom-up traversal (semi-joins)
2. Top-down traversal (semi-joins)
3. Join bottom-up

In each join step, each left input tuple joins with at least 1 right input tuple, and vice versa!
Yannakakis Algorithm – Properties

• Semi-join sweeps take $\tilde{O}(n)$

• A join step can never shrink intermediate result size
  - This does not hold for all trees
  - Tree must be attribute-connected (more on this soon)

• Hence all intermediate results are of size $O(r)$

• Each join step therefore has $O(n + r)$ input and $O(r)$ output

• Easy to compute a binary join with $O(n + r)$ input and $O(r)$ output in time $\tilde{O}(n + r)$, e.g., using sort-merge join
Outline tutorial

• Part 1: Top-\(k\) (Wolfgang): \(~20\text{min}\)
• Part 2: Optimal Join Algorithms (Mirek): \(~30\text{min}\)
  – Lower Bound and the Yannakakis Algorithm
  – Problems Caused by Cycles
  – Tree Decompositions
  – Summary and Further Reading
• Part 3: Ranked enumeration over joins (Nikolaos): \(~40\text{min}\)
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 input

Join tree

\[ R_3 \]
\[ R_2 \]
\[ R_1 \]
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
- \( R_1 \bowtie R_2 \) produces \( n^2 \) intermediate results
  - Final output size: \( n^2 \) for \( Q_{3p} \), but only \( n \) for \( Q_{3c} \)
What Went Wrong?

- The tree for the 3-cycle is not attribute-connected!
- Attribute-connectedness:
  - For each attribute, the nodes containing it form a connected sub-graph
- 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]:
  - $\tilde{O}(n + r)$ unattainable based on well-accepted complexity-theoretic assumptions

What Can Be Done?

- Worst-case-optimal (WCO) join algorithms
  [Veldhuizen 14, Ngo+ 14, Ngo+ 18]
- Instead of $\tilde{O}(n + r)$, get
  $\tilde{O}(n + r_{WC}) = \tilde{O}(r_{WC})$
- $r_{WC}$ = 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 $n^3$

[Ng0+ 14] Ngo, Re, Rudra. Skew strikes back: New developments in the theory of join algorithms. SIGMOD Rec.’14  
[Ng0+ 18] Ngo, Porat, Ré, Rudra. Worst-case optimal join algorithms. J. ACM’18  
What Can Be Done?

- Worst-case-optimal (WCO) join algorithms
  [Veldhuizen 14, Ngo+ 14, Ngo+ 18]
- Instead of $\tilde{O}(n + r)$, get $\tilde{O}(n + r_{WC}) = \tilde{O}(r_{WC})$
- $r_{WC}$ = 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 $n^3$
- Tree decompositions
- Put more effort into pre-processing to avoid large intermediate results
  - Use WCO joins as sub-routine
- Goal: $\tilde{O}(n^d + r)$ for smallest $d$ possible
  - $\tilde{O}(n^d)$ captures pre-processing cost
  - $d = 1$ for acyclic CQ

\[^1\]Ngo+ 18] Ngo, Porat, Ré, Rudra. Worst-case optimal join algorithms. J. ACM’18 https://doi.org/10.1145/3180143
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
  – Lower Bound and the Yannakakis Algorithm
  – Problems Caused by Cycles
  – Tree Decompositions
  – Summary and Further Reading
• Part 3: Ranked enumeration over joins (Nikolaos): ~40min
Main Idea of Tree Decompositions

• Convert cyclic CQ to a rooted tree-shaped CQ

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

$$R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)$$
Main Idea of Tree Decompositions

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

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

\[
T = R_4(A_4, A_5) \bowtie R_5(A_5, A_6) \bowtie R_6(A_6, A_1)
\]
Main Idea of Tree Decompositions

- Convert cyclic CQ to a rooted tree-shaped CQ
- Materialize all tree nodes ("bags") using a WCO join algorithm
- Apply Yannakakis algorithm on the tree
  - Acyclic CQ whose input relations are the bags
  - Achieves $\tilde{O}(x + r)$ where $x$ is the size of the largest bag

![Diagram showing tree decomposition and Yannakakis algorithm]

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

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

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

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

What is the simplest tree with these properties?

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

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

Bag materialization costs \( O(n^3) \) (AGM bound)
Tree Decomposition Intuition

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

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

For each attribute, the bags containing it are connected

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

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

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

For each attribute, the bags containing it are connected

\[ Q_{6c}(A_1, ..., A_6) = R_1(A_1, A_2) \Join R_2(A_2, A_3) \]
\[ \Join R_3(A_3, A_4) \Join R_4(A_4, A_5) \]
\[ \Join R_5(A_5, A_6) \Join R_6(A_6, A_1) \]

Can we “slim down” the bags even more?

Bag materialization costs \( O(n^2) \) (AGM bound)
Tree Decomposition Intuition

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

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

For each attribute, the bags containing it are connected

O(n) bag materialization...?
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

\[ Q_{6c}(A_1, ..., A_6) = R_1(A_1, A_2) \Join R_2(A_2, A_3) \Join R_3(A_3, A_4) \Join R_4(A_4, A_5) \Join R_5(A_5, A_6) \Join R_6(A_6, A_1) \]

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
  - $\chi: V(\mathcal{T}) \to 2^V$ 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 $O(l)$ nodes, each materialized using \textit{WCO join}
  - Worst-case output of bag $i$ is of size $O(n^{d_i})$ for some $d_i \geq 1$ (AGM bound)
  - Fractional hypertree width (fhw) $d = \max_i d_i$ [Grohe+ 14]
  - Total bag-materialization cost: $O(n^d)$
  - Size of a materialized bag: $O(n^d)$
  - Resulting cost for Yannakakis algorithm on materialized tree: $\tilde{O}(n^d + r)$

Who Wins?

$\mathcal{T}_3$ 

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

$\mathcal{T}_1$

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

$\mathcal{T}_2$

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

\[ \mathcal{T}_1 > \mathcal{T}_2 = \mathcal{T}_3 \]
A Closer Look

• $T_1$ 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 $\mathcal{T}_3$, at most 2 relations are joined
    • 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 $\mathcal{T}_3$, at most 2 relations are joined
    • One of them is just the set of distinct $A_1$-values in $R_1$

• $\mathcal{T}_3$ is better when the number of distinct $A_1$-values in $R_1$ is low, e.g., $O(n^{2/3})$ instead of $O(n)$
Who Wins?

\[ R_1(A_1,A_2) \]
\[ 0(n) \]

\[ R_2(A_2,A_3), R_1(A_1,\_ ) \]
\[ 0(n^{5/3}) \]

\[ R_3(A_3,A_4), R_1(A_1,\_ ) \]
\[ 0(n^{5/3}) \]

\[ R_4(A_4,A_5), R_1(A_1,\_ ) \]
\[ 0(n^{5/3}) \]

\[ R_5(A_5,A_6), R_1(A_1,\_ ) \]
\[ 0(n^{5/3}) \]

\[ R_6(A_6,A_1) \]
\[ 0(n) \]

\[ T_3 \]

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?

$R_1(A_1, A_2)$

$0(n)$

$R_2(A_2, A_3), R_1(A_1, \_)$

$0(n^{5/3})$

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

$0(n^{5/3})$

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

$0(n^{5/3})$

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

$0(n^{5/3})$

$R_6(A_6, A_1)$

$0(n)$

$\mathcal{T}_3$

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

$0(n^2)$

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

$0(n^2)$

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

$\mathcal{T}_2$
Could $\mathcal{T}_2$ Win?

- Consider bag $\mathcal{R}_1(A_1, A_2), \mathcal{R}_2(A_2, A_3), \mathcal{R}_3(A_3, A_4)$ in $\mathcal{T}_2$

- What if each $\mathcal{R}_1$-tuple joins with only "a few" $\mathcal{R}_2$-tuples?
- What if each $\mathcal{R}_2$-tuple joins with only "a few" $\mathcal{R}_3$-tuples?

- What if "a few" was $n^{1/3}$?
Who Wins Now?

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

\[ O(n^{5/3}) \]

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

\[ O(n^{5/3}) \]

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

\( \mathcal{T}_2 \)
Who Wins Now?

\( R_1(A_1, A_2) \)  
\[ O(n) \]

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

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

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

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

\( R_6(A_6, A_1) \)  
\[ O(n) \]

\( T_3 \)

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

\[ O(n^{5/3}) \]

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

\[ O(n^{5/3}) \]

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

\( T_2 \)
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

Multiple Plans: Plan 1

\[ R_1^H(A_1, A_2) \]

\[ R_2(A_2, A_3), R_1^H(A_1, _) \]

\[ R_3(A_3, A_4), R_1^H(A_1, _) \]

\[ R_4(A_4, A_5), R_1^H(A_1, _) \]

\[ R_5(A_5, A_6), R_1^H(A_1, _) \]

\[ R_6(A_6, A_1) \]

\[ \mathcal{T}_3 \]

\[ R_1^H \]: 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

\[ \mathcal{T}_3: \text{computes} \ R_1^H \bowtie R_2 \bowtie \cdots \bowtie R_6 \]

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

- \( R_1^H(A_1, A_2) \): contains all tuples whose \( A_1 \)-values occur more than \( n^{1/3} \) times (fewer than \( n^{2/3} \) such \( A_1 \)-values exist)
- \( R_2(A_2, A_3), R_1^H(A_1, \_): 0(n^{5/3}) \)
- \( R_3(A_3, A_4), R_1^H(A_1, \_): 0(n^{5/3}) \)
- \( R_4(A_4, A_5), R_1^H(A_1, \_): 0(n^{5/3}) \)
- \( R_5(A_5, A_6), R_1^H(A_1, \_): 0(n^{5/3}) \)
- \( R_6(A_6, A_1) \): \( 0(n) \)
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: $|\pi_{A_2}(R_2^H)| \leq n^{2/3}$

$R_2^H$: 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
  - $R_1^L \bowtie R_2^L \bowtie R_3^H \bowtie R_4 \bowtie R_5 \bowtie R_6$
  - $R_1^L \bowtie R_2^L \bowtie R_3^L \bowtie R_4^H \bowtie R_5 \bowtie R_6$

• 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 $\tilde{O}(n^{5/3} + r)$
  - Beats single-tree complexity $\tilde{O}(n^2 + r)$ and WCO-join complexity $\tilde{O}(n^3)$
Tree Decompositions: The Big Picture

- WCO join algorithms applied to a query $Q$: complexity determined by the AGM bound for $Q$
- Decomposing $Q$ into a tree creates bags that each may have a lower AGM bound value (fractional hypertree width)
- This reduces time complexity
  - Materialize each bag using a WCO join algorithm
  - Run Yannakakis algorithm on the tree with materialized bags as input

- Design goal: minimize width of tree, but ensure that each input relation is covered by a bag and the tree is attribute-connected!
Tree Decompositions: Degree Constraints

- Degree constraints generalize cardinality constraints and functional dependencies
- Enable improved complexity guarantees
- Application to general input (with only cardinality constraints):
  - Partition input so that each *partition* satisfies stronger degree constraints
  - Use appropriate tree for each partition
  - Current frontier: submodular width
- Application to input DB with degree constraints:
  - Degree-aware submodular width

[Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. J.ACM’13] [Khamis, Ngo, Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? PODS’17] [Joglekar, Ré. It's all a matter of degree. Theory of Computing Systems’18]
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
  – Lower Bound and the Yannakakis Algorithm
  – Problems Caused by Cycles
  – Tree Decompositions
  – Summary and Further Reading
• Part 3: Ranked enumeration over joins (Nikolaos): ~40min
Optimal-Joins Summary

- On acyclic CQs, the Yannakakis algorithm’s complexity $\tilde{O}(n + r)$ matches the lower bound
  - Simple join-at-a-time query plan after semi-join reduction sweeps
- Lower bound cannot be matched for general cyclic CQs
- WCO joins achieve $\tilde{O}(n + r_{WC})$
  - “Holistic” multi-way join approach
- Tree decomposition approaches achieve $\tilde{O}(n^d + r)$
  - Best $d$ is submodular width $\text{subw}(Q)$ using multi-tree decomposition
  - $n^{\text{subw}(Q)} = O(r_{WC})$, but usually better
    - 6-cycle: $r_{WC} = n^3$, but multi-tree approach achieves $\tilde{O}(n^{5/3} + r)$
Further Reading

• Extensions of the query model
  - Projections and aggregation

  [Khamis, Ngo, Rudra. FAQ: questions asked frequently. PODS’16] https://doi.org/10.1145/2902251.2902280

• Factorized DB

  [Bakibayev, Kočiský, Olteanu, Závodný. Aggregation and Ordering in Factorised Databases. PVLDB’13] https://doi.org/10.14778/2556549.2556579
  [Olteanu, Schleich. Factorized databases. SIGMOD Record’16] https://doi.org/10.1145/3003665.3003667

• Stronger notions of optimality

  [Ngo, Nguyen, Re, Rudra. Beyond worst-case analysis for joins with minesweeper. PODS’14] https://doi.org/10.1145/2594538.2594547

• ...and many more (see our tutorial paper)