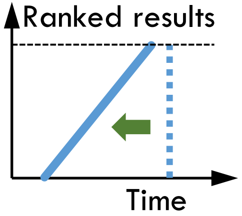
SIGMOD 2020 tutorial

Optimal Join Algorithms meet Top-k

Nikolaos Tziavelis, Wolfgang Gatterbauer, Mirek Riedewald

Northeastern University, Boston



Part 2 : Optimal Join Algorithms

Northeastern University Khoury College of Computer Sciences

Slides: https://northeastern-datalab.github.io/topk-join-tutorial/
DOI: https://doi.org/10.1145/3318464.3383132
Data Lab: https://db.khoury.northeastern.edu

DATALAB @Northeastern

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 $\mathrm{O}(n)$ 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: $\mathrm{O}(l)$
  - Output cardinality: r

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

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., $\mathrm{O}\left(f(l) \cdot n^{f(l)} + (\log n)^{f(l)} \cdot r\right)$ simplifies to $\mathrm{O}\left(n^{f(l)} + (\log n)^{f(l)} \cdot r\right)$

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., $\mathrm{O}\left(f(l) \cdot n^{f(l)} + (\log n)^{f(l)} \cdot r\right)$ simplifies to $\mathrm{O}\left(n^{f(l)} + (\log n)^{f(l)} \cdot r\right)$
- We mostly use $\widetilde{\mathrm{O}}$-notation (soft-O) data complexity
  - Abstracts away polylog factors in input size that clutter formulas
  - E.g., $\mathrm{O}\left(n^{f(l)} + (\log n)^{f(l)} \cdot r\right)$ further simplifies to $\widetilde{\mathrm{O}}\left(n^{f(l)} + r\right)$

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 $\widetilde{O}(n+r)$

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

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

R_1

A_1	A_2
1	20
1	10
4	60

A_1, A_2

R_2

A_1	A_2	A_3
1	10	100
1	20	100
3	10	300
1	40	300
2	30	200

A_2

R_3

A_2
10
20
30

A_1, A_2

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000
2	20	2000

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)

R_1

A_1	A_2
1	20
1	10
4	60

A_1, A_2

R_2

A_1	A_2	A_3
1	10	100
1	20	100
3	10	300
1	40	300
2	30	200

$\boldsymbol{R_2 \ltimes R_4}$

A_2

R_3

A_2
10
20
30

A_1, A_2

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000
2	20	2000

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)

R_1

A_1	A_2
1	20
1	10
4	60

A_1, A_2

R_2

A_1	A_2	A_3
1	10	100
1	20	100
3	10	300
1	40	300
2	30	200

$\boldsymbol{R_2 \ltimes R_4}$

A_1	A_2
1	10
1	20
2	20

A_2

R_3

A_2
10
20
30

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000
2	20	2000

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)

R_1

A_1	A_2
1	20
1	10
4	60

A_1, A_2

R_2

A_1	A_2	A_3
1	10	100
1	20	100
~~3~~	~~10~~	~~300~~
~~1~~	~~40~~	~~300~~
~~2~~	~~30~~	~~200~~

$R_2 \ltimes R_4$

A_1	A_2
1	**10**
1	**20**
2	**20**

A_2

R_3

A_2
10
20
30

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000
2	20	2000

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)

R_1

A_1	A_2
1	20
1	10
4	60

A_1, A_2

R_2

A_1	A_2	A_3
1	10	100
1	20	100
~~3~~	~~10~~	~~300~~
~~1~~	~~40~~	~~300~~
~~2~~	~~30~~	~~200~~

$R_2 \ltimes R_3$

A_2
10
20
30

$R_2 \ltimes R_4$

A_1	A_2
1	10
1	20
2	20

R_3

A_2
10
20
30

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000
2	20	2000

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)

R_1

A_1	A_2
1	20
1	10
~~4~~	~~60~~

$R_1 \ltimes R_2$

A_1	A_2
1	10
1	20

R_2

A_1	A_2	A_3
1	10	100
1	20	100
~~3~~	~~10~~	~~300~~
~~1~~	~~40~~	~~300~~
~~2~~	~~30~~	~~200~~

$R_2 \ltimes R_3$

A_2
10
20
30

$R_2 \ltimes R_4$

A_1	A_2
1	10
1	20
2	20

R_3

A_2
10
20
30

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000
2	20	2000

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)

R_1

A_1	A_2
1	20
1	10
~~4~~	~~60~~

$R_2 \ltimes R_1$

R_2

A_1	A_2	A_3
1	10	100
1	20	100
~~3~~	~~10~~	~~300~~
~~1~~	~~40~~	~~300~~
~~2~~	~~30~~	~~200~~

$R_3 \ltimes R_2$

$R_4 \ltimes R_2$

R_3

A_2
10
20
~~30~~

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000
~~2~~	~~20~~	~~2000~~

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

R_1

A_1	A_2
1	20
1	10

R_2

A_1	A_2	A_3
1	10	100
1	20	100

R_3

A_2
10
20

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000

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!

R_1

A_1	A_2
1	20
1	10

R_2

A_1	A_2	A_3
1	10	100
1	20	100

R_3

A_2
10
20

R_4

A_1	A_2	A_4
1	10	1000
1	20	1000
1	20	2000

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

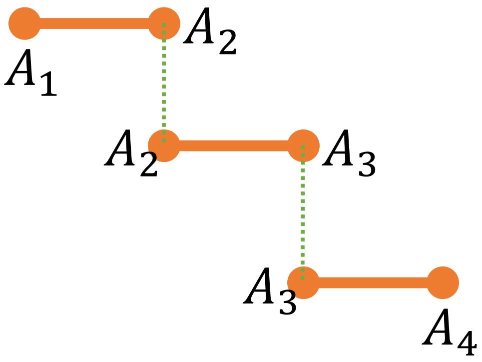
Outline tutorial

- Part 1: Top-k (Wolfgang): ~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

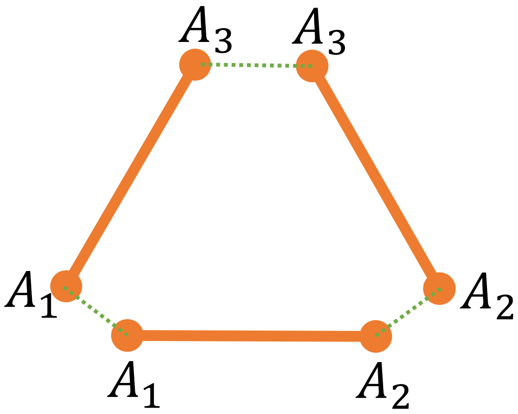
CQs with Cycles

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

Q_{3p}



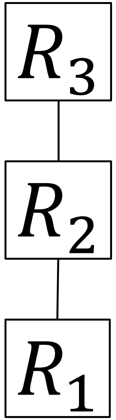
Q_{3c}



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_1

A_1	A_2
1	1
2	1
…	…
n	1

R_2

A_2	A_3
1	1
1	2
…	…
1	n

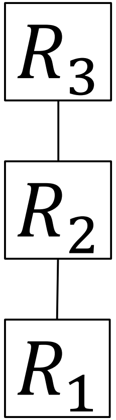
R_3

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

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

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

Q_{3p}

$R_3(A_3, A_4)$

$R_2(A_2, A_3)$

$R_1(A_1, A_2)$

Q_{3c}

$R_3(A_3, A_1)$

$R_2(A_2, A_3)$

$R_1(A_1, A_2)$

Solutions for Cycles? Some Bad News

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

- Yes, but…

- … [Ngo+ 18]:
  - $\widetilde{\mathrm{O}}(n + r)$ 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 $\widetilde{\mathrm{O}}(n + r)$, get
  $$\widetilde{\mathrm{O}}(n + r_{\mathrm{WC}}) = \widetilde{\mathrm{O}}(r_{\mathrm{WC}})$$
- 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

[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 $\widetilde{\mathrm{O}}(n+r)$, get
  $$\widetilde{\mathrm{O}}(n+r_{\mathrm{WC}}) = \widetilde{\mathrm{O}}(r_{\mathrm{WC}})$$
- 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

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

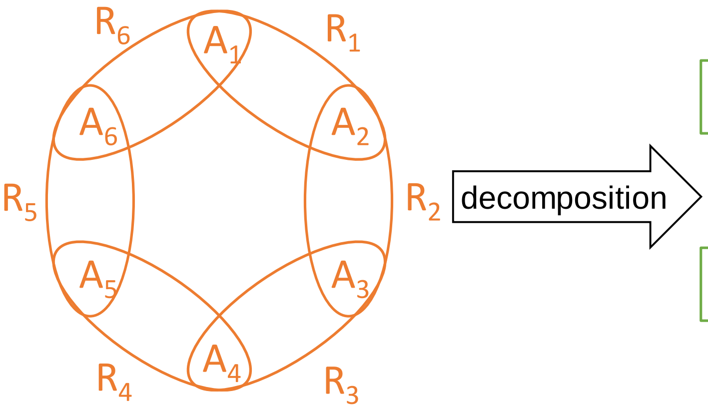
[Veldhuizen 14] Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. ICDT'14 https://doi.org/10.5441/002/icdt.2014.13
[Ngo+ 14] Ngo, Re, Rudra. Skew strikes back: New developments in the theory of join algorithms. SIGMOD Rec.'14 https://doi.org/10.1145/2590989.2590991
[Ngo+ 18] Ngo, Porat, 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

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

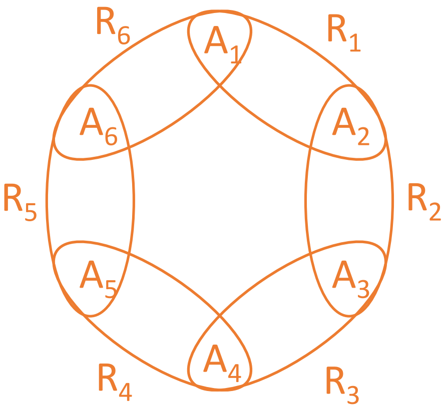


$$R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie R_3(A_3, A_4)$$

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

Main Idea of Tree Decompositions

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



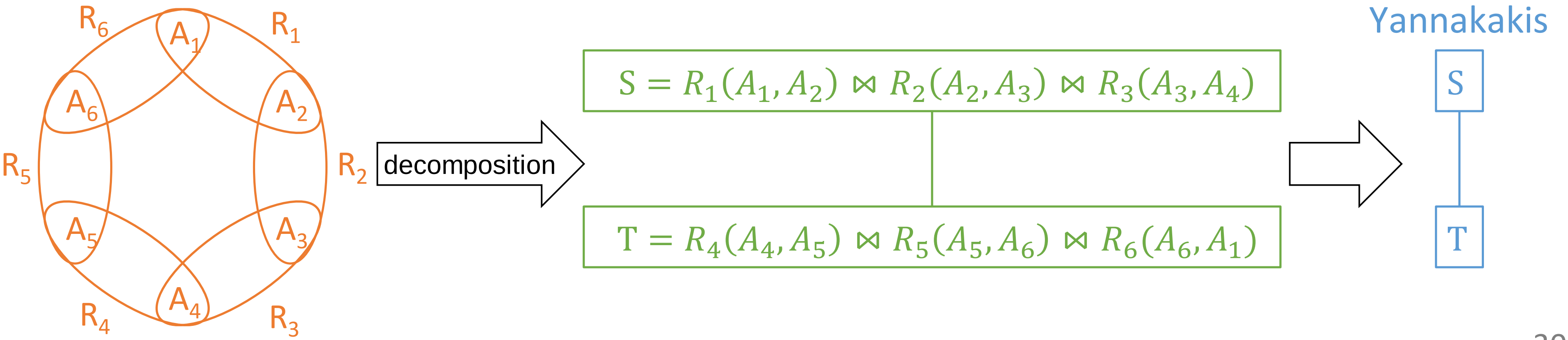
decomposition

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

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

Main Idea of Tree Decompositions

- 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 $\widetilde{O}(x + r)$ 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

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

$\mathcal{T}_1$ $\boxed{\begin{array}{l} R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4) \\ R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1) \end{array}}$

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

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

$\mathcal{T}_1$ $\boxed{\begin{matrix} R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4) \\ R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1) \end{matrix}}$

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

Can we do better?

For each attribute, the bags containing it are connected

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

Tree Decomposition Intuition

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

$\mathcal{T}_2$

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

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

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?

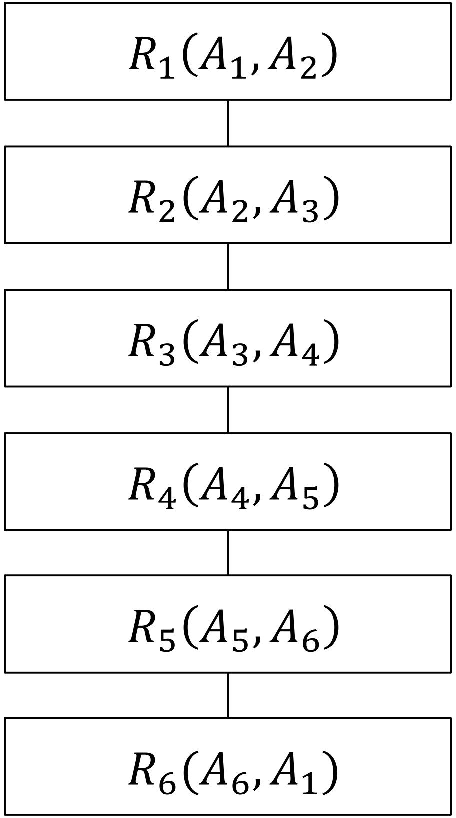
Every relation appearing in the query is covered by a bag (tree node)

For each attribute, the bags containing it are connected

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

Tree Decomposition Intuition

$\mathcal{T}_3$



$O(n)$ bag materialization…?

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

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

$\mathcal{T}_3$

$R_1(\boldsymbol{A_1}, A_2)$

$R_2(A_2, A_3)$

$R_3(A_3, A_4)$

$R_4(A_4, A_5)$

$R_5(A_5, A_6)$

$R_6(A_6, \boldsymbol{A_1})$

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

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

$\mathcal{T}_3$

$R_1(A_1, A_2)$

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

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

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

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

$R_6(A_6, A_1)$

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

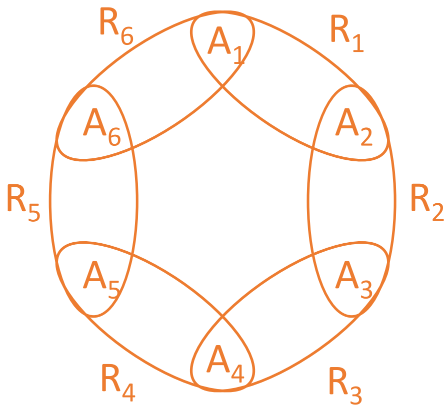
Every relation appearing in the query is covered by a bag (tree node)

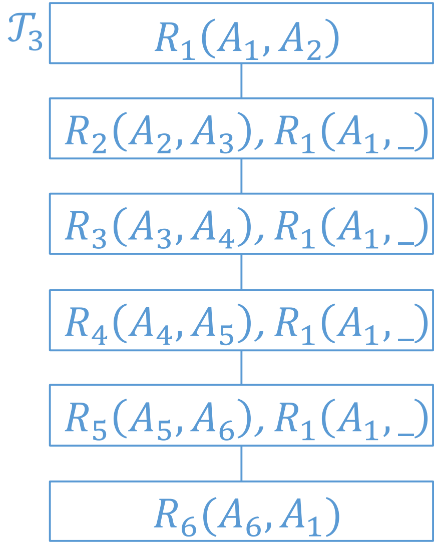
For each attribute, the bags containing it are connected

$O\left(n \cdot \left|\pi_{A_1}(R_1)\right|\right)$ 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 $O(l)$ nodes, each materialized using *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: $\tilde{O}(n^d + r)$

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

Who Wins?

$\mathcal{T}_3$

$R_1(A_1, A_2)$

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

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

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

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

$R_6(A_6, A_1)$

$\mathcal{T}_1$

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

$\mathcal{T}_2$

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

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

$$\mathcal{T}_1 > \mathcal{T}_2 = \mathcal{T}_3$$

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 $\mathcal{T}_3$, 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?

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

$\mathcal{T}_3$

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

$\mathcal{T}_3$:

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

$\mathcal{T}_2$:

Size	Bag
$O(n^2)$	$R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$
$O(n^2)$	$R_4(A_4, A_5), R_5(A_5, A_6), R_6(A_6, A_1)$

Could $\mathcal{T}_2$ Win?

- Consider bag $R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_4)$ 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 $n^{1/3}$?

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

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

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

$\mathcal{T}_2$

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

$R_1^H(A_1, A_2)$

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

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

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

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

$R_6(A_6, A_1)$

$\mathcal{T}_3$

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

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

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

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

$R_2^H(A_2, A_3)$

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

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

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

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

$R_1^L(A_1, A_2)$

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)

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

$\mathcal{J}_3$: 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] https://doi.org/10.1145/2535926
[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
[Joglekar, Ré. It's all a matter of degree. Theory of Computing Systems'18] https://doi.org/10.1007/s00224-017-9811-8

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 $\tilde{O}(n + r_{\mathrm{WC}})$
  - "Holistic" multi-way join approach
- Tree decomposition approaches achieve $\tilde{O}(n^d + r)$
  - Best d is submodular width subw(Q) using multi-tree decomposition
  - $n^{\mathrm{subw}(Q)} = O(r_{\mathrm{WC}})$, but usually better
    - 6-cycle: $r_{\mathrm{WC}} = n^3$, 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
  [Bakibayev, Olteanu, Závodný. FDB: A Query Engine for Factorised Relational Databases. PVLDB'12] https://doi.org/10.14778/2350229.2350242
  [Olteanu, Schleich. Factorized databases. SIGMOD Record'16] https://doi.org/10.1145/3003665.3003667
  [Olteanu, Závodny. Size bounds for factorised representations of query results. TODS'15] https://doi.org/10.1145/2656335

- Stronger notions of optimality

  [Khamis, Ré, Rudra. Joins via Geometric Resolutions: Worst Case and Beyond. TODS'16] https://doi.org/10.1145/2967101
  [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)