Database Theory

Worst-case Optimal Join Algorithms

Matthias Lanzinger, WS2025/26

TU WIEN Informatics

What is the best possible running time we can hope for when evaluating a query?

Two parts are unavoidable:

1. Reading the input relations.
2. Writing the full output.

Ideally, we want an algorithm that does not do significantly more than that.
Formally, for query q with relations $R_1, \ldots, R_n$ and database D we want time

$$O\left(f(q) \cdot \sum_{i=1}^{n} |R_i^D| + g(q) \cdot |q(D)|\right)$$

Input relations

Output size

Call such an algorithm "instance-optimal"

Is such an algorithm possible?

No 😢

(Unless **NP**=**RP**)

What if we relax this a bit to a bound
depending on the largest answer *for some relation sizes?*

Formally, let q be a query with relations $R_1, \ldots, R_k$ and let $\overline{N} = (N_1, \ldots, N_k)$.

Let $\mathscr{D}(\overline{N})$ be the set of all databases (over $R_1, \ldots, R_k$), such that R_i has exactly N_i tuples.

Is there an algorithm that answers q for some given input $D' \in \mathscr{D}(\overline{N})$ in the following time:

$$O\left(f(q) \cdot \sum_{i=1}^{k} N_i + g(q) \cdot \sup_{D \in \mathscr{D}(\overline{N})} |q(D)|\right)$$

Output size of the "worst-case instance"

Algorithms that evaluate conjunctive queries in such time are called "Worst-case Optimal Join Algorithms"

$$O\left(f(q)\cdot\sum_{i=1}^{k}N_i+g(q)\cdot\sup_{D\in\mathscr{D}(\overline{N})}|q(D)|\right)$$

Where f, g are polynomial.

Worst-Case Output Size

Let's first try and understand $\sup_{D \in \mathscr{D}(\overline{N})} |q(D)|$ better.

Consider the query $q := \{(x, z) \mid R_1(x, y) \wedge R_2(y, z)\}$ and fix $\overline{N} = (1,2)$, that is R_1 has 1 tuple, R_2 has 2.

What is $\sup_{D \in \mathscr{D}(\overline{N})} |q(D)|$?

R_1

X	Y
1	1

R_2

Y	Z
1	2
1	3

$|q(D)|$ here equals 2.

Here it is easy to see that other databases will not create larger results.

Worst-Case Output Size

Let's first try and understand $\sup_{D \in \mathscr{D}(\overline{N})} |q(D)|$ better.

$$q := \{(x, z) \mid R_1(x, y) \wedge R_2(y, y') \wedge R_3(y', z)\}$$

with $\overline{N} = (2,1,2)$

$\sup_{D \in \mathscr{D}(\overline{N})} |q(D)| = 4$

with $\overline{N} = (2,2,2)$

$\sup_{D \in \mathscr{D}(\overline{N})} |q(D)| = 4$

with $\overline{N} = (N_1, N_2, N_3)$

?

Worst-case Output Size

The question of worst-case output size is interesting in itself.

Let $q := \{(x, y, z) \mid E(x, y) \wedge E(y, z) \wedge E(x, z)\}$ (the triangle query).

Finding the worst-case output size for q is the same as finding the *maximum number of triangles in a graph* with a fixed number of edges!

Bounding the Number of Triangles

Let us analyse $q := \{(x, y, z) \mid E(x, y) \wedge E(y, z) \wedge E(x, z)\}$ with $|E| = m$.

Naive bound:

This is a join of three relations, so it cannot be larger than the product of their sizes. We get $|q(D)| \leq m^3$.

Bit more thought:

Consider $q' := \{(x, y, z) \mid E(x, y) \wedge E(y, z)\}$, clearly $q'(D) \supseteq q(D)$.

But this is only one join, so we see $|q(D)| \leq |q'(D)| \leq m^2$.

Bounding the Number of Triangles

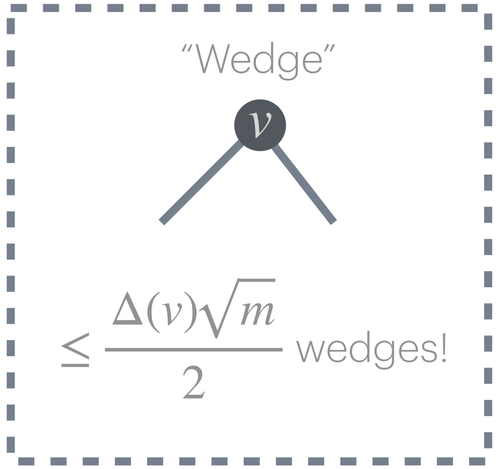
The best possible asymptotic bound is in fact $O(m^{3/2})$!

Bounding the Number of Triangles

We write $\Delta(v)$ for the degree of v

Split the vertices into heavy vertices H and light vertices L,

$$H = \{v \in V \mid \Delta(v) > \sqrt{m}\} \text{ and , } L = V \backslash H.$$



How many triangles have at least 1 light vertex?

Such a triangle has an edge connecting a wedge of a light vertex v.

Because v is light, there are at most $\Delta(v)\frac{\sqrt{m}}{2}$ such wedges.

Hence there are at most $\frac{\sqrt{m}}{2}\sum_{v \in L}\Delta(v) \leq \frac{\sqrt{m}}{2} \cdot 2m = m^{3/2}$ "light triangles" in the whole graph.

Recall, $\sum \Delta(v) \leq 2|E|$ in any graph.

Bounding the Number of Triangles

We write $\Delta(v)$ for the degree of v

Split the vertices into heavy vertices H and light vertices L,

$$H = \{v \in V \mid \Delta(v) > \sqrt{m}\} \text{ and , } L = V \backslash H.$$

How many triangles have only heavy vertices?

There are at most $\binom{|H|}{3}$ combinations of three heavy vertices, so clearly there cannot be more "heavy" triangles.

Furthermore, observe that $|H| \leq 2\sqrt{m}$: else $\sum_{v \in H} \Delta(v) > 2\sqrt{m}\sqrt{m} = 2m$

We use the known inequality $\binom{n}{c} \leq n^c$

So there are at most $\binom{|H|}{3} \leq |H|^3 = 8\sqrt{m}^3 = 8m^{3/2}$ heavy triangles.

Bounding the Number of Triangles

To summarise:

Every vertex is either heavy or light.

There are at most $m^{3/2}$ triangles that contain a light vertex.

There are at most $8m^{3/2}$ triangles that contain only heavy vertices.

$\Rightarrow$ *There are at most* $O(m^{3/2})$ *triangles in a graph with* m *edges.*

Back to Queries

We now know the triangle query outputs at most $O(|E|^{3/2})$ tuples.

So a worst-case optimal join algorithm should take $O(f(q) \cdot |E|^{3/2})$ time.

How?

Back to Queries

Let us switch to a more general version of the triangle query:

$$q := R \bowtie S \bowtie T$$

with schema $R(A, B) \quad S(B, C) \quad T(C, A)$

We can use the same argument as before to show that this query has maximum output size $O(\max\{|R|, |S|, |T|\}^{3/2})$.

We can actually strengthen this to the following (we will see how later)

$$q(D) \leq \sqrt{|R| \cdot |S| \cdot |T|}$$

WCOJ for $\triangle$ — Alg 1

$$\theta \leftarrow \sqrt{\frac{|R| \cdot |S|}{|T|}}$$

$$R^h \leftarrow \{(a,b) \in R : \ |\sigma_{A=a} R| > \theta\}$$

$$R^l \leftarrow \{(a,b) \in R : \ |\sigma_{A=a} R| \leq \theta\}$$

$$\triangle_1 = (R^h \bowtie S) \ltimes T$$

$$\triangle_2 = (R^l \bowtie T) \ltimes S$$

Return $\triangle_1 \cup \triangle_2$

Note that $|R^h| \leq |R|/\theta$

$$|R^h \bowtie S| \leq |R^h| \cdot |S| \leq |S| \cdot |R|/\theta = \sqrt{|R| \cdot |S| \cdot |T|}$$

$$|R^l \bowtie T| \leq |T| \cdot \theta = \sqrt{|R| \cdot |S| \cdot |T|}$$

WCOJ for $\triangle$ — Alg 2

For $a \in \pi_A R \cap \pi_A T$

For $b \in \pi_B \sigma_{A=a} R \cap \pi_B S$

For $c \in \pi_C \sigma_{A=a} T \cap \pi_C \sigma_{B=b} S$

Output (a, b, c)

Note that we can compute $X \cap Y$ in time $\tilde{O}(\min(|X|, |Y|))$.

The tricky part is to bound how often the inner-most loop runs.

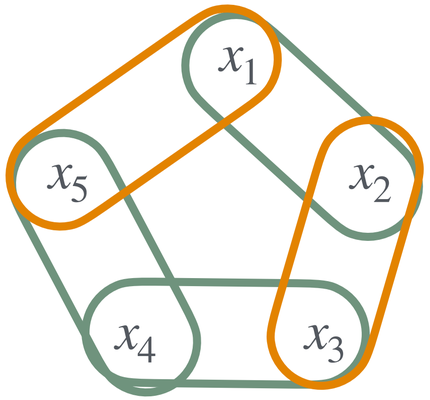
(See more in a *Theory Exercise*)

General Size Bounds

Worst-Case Output Size

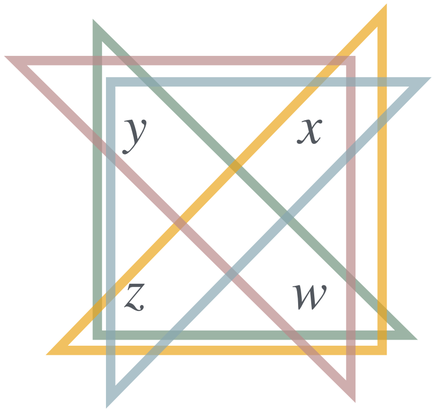
Analysing every query like we did for the triangle
is not feasible.

$$R(x_1, x_2) \wedge S(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_5) \wedge S(x_5, x_1)$$



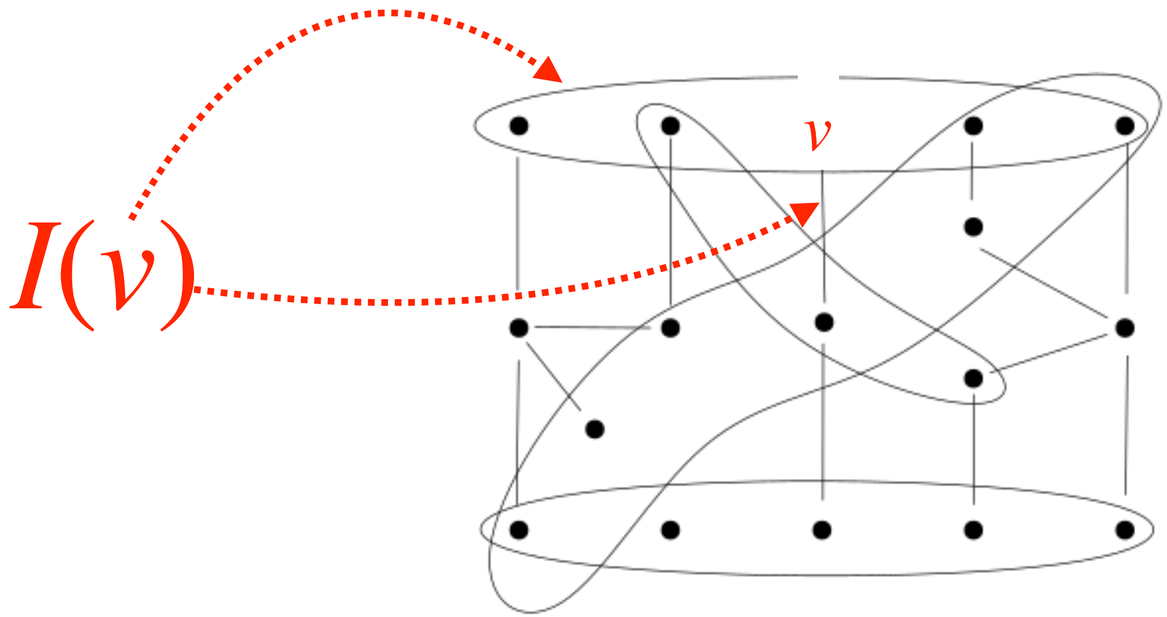
?

$$R(w, x, y) \wedge S(x, y, z) \wedge T(w, y, z) \wedge U(w, x, z)$$



Edge Covers

Let H be a hypergraph. We write $I(v)$ for the set of incident edges to vertex v. That is, $I(v) := \{e \in E(H) \mid v \in e\}$

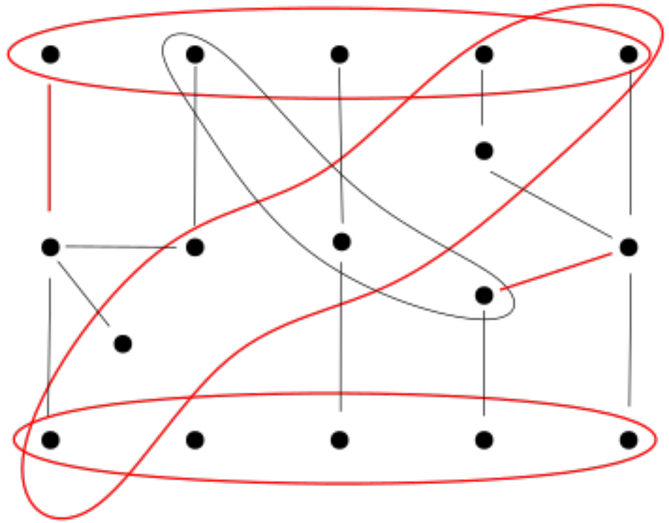


Edge Covers

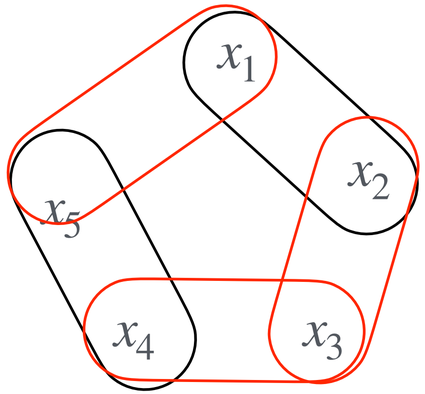
An *edge cover* is a function

$$w : E(H) \to \{0,1\} \text{ such that } \sum_{e \in I(v)} w(e) \geq 1$$

for every vertex v



weight 5
edge cover



weight 3
edge cover

The total weight assigned to edges:

$$\sum_{e \in E(H)} w(e)$$

is called the weight of the cover w

Why Edge Covers?

Recall our initial argument for the worst-case size bound of the triangle query.

Bit more thought:

Consider $q' := \{(x, y, z) \mid E(x, y) \wedge E(y, z)\}$, clearly $q'(D) \supseteq q(D)$.

But this is only one join, so we see $|q(D)| \leq |q'(D)| \leq m^2$.

Why Edge Covers?

What we did was find the smallest subquery that contained all the variables and observed that the output cannot be larger than that join.

Bit more thought:

Consider $q' := \{(x, y, z) \mid E(x, y) \wedge E(y, z)\}$, clearly $q'(D) \supseteq q(D)$.

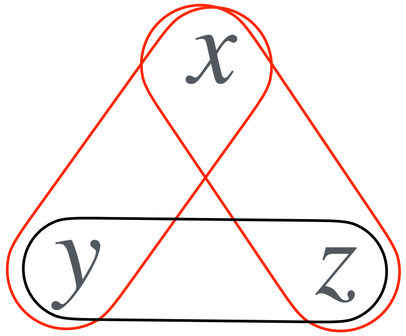
But this is only one join, so we see $|q(D)| \leq |q'(D)| \leq m^2$.

Why Edge Covers?

find the smallest subquery that contained all the variables

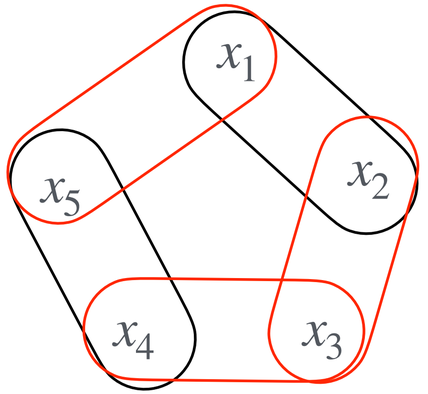
$\Rightarrow$

find a minimal weight edge cover in the query hypergraph



Cover with 2 atoms $\Rightarrow$

$|q(D)| \leq |D|^2$



Cover with 3 atoms $\Rightarrow$

$|q(D)| \leq |D|^3$

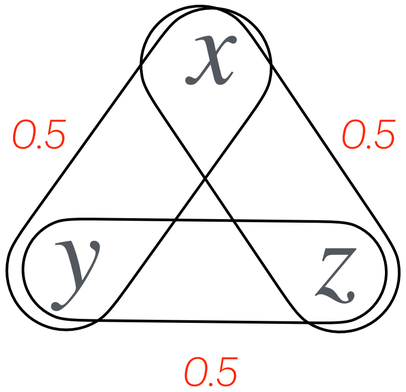
Fractional Edge Covers

A ***fractional*** *edge cover* is a function

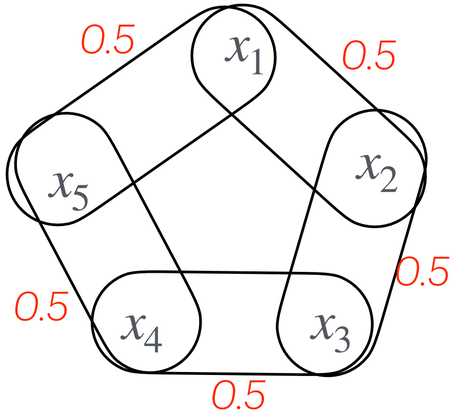
$$w : E(H) \to [0,1] \text{ such that } \sum_{e \in I(v)} w(e) \geq 1$$

(the unit interval)

for every vertex v



weight 1.5
fractional edge cover



weight 2.5
fractional edge cover

(Fractional) Edge Covers

The minimal weight of an edge cover of hypergraph H is called the *edge cover number* of H. Denoted by $\rho(H)$.

The minimal weight of a fractional edge cover of hypergraph H is called the *fractional edge cover number* of H. Denoted by $\rho^*(H)$.

Easy standard observation: $\rho^*(H) \leq \rho(H)$

The AGM Bound

Theorem (Grohe & Marx; 2006)

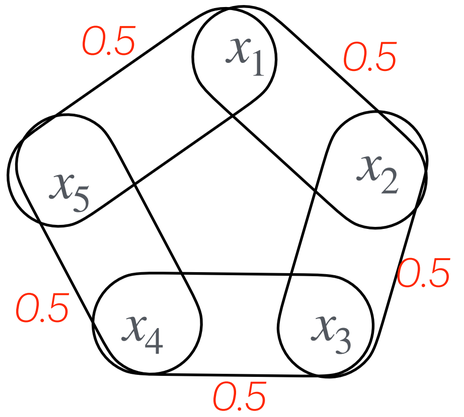
Let w be a *fractional edge cover* of CQ q over relations $R_1, \ldots, R_k$.

Then $$|q(D)| \leq \prod_{i=1}^{k} |R_i|^{w(R_i)}$$

For simplicity we assume that q is self-join free. That is, every relation symbol R_i occurs only once in q. This is just to simplify notation, the theorem holds analogously with self-joins.

AGM Bound – Example

$$q_5 := R(x_1, x_2) \wedge S(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_5) \wedge S(x_5, x_1)$$



weight 2.5
fractional edge cover

The AGM bound then specifically tells us that

$$|q_5(G)| \leq |R|^{0.5} \cdot |S|^{0.5} \cdot |R|^{0.5} \cdot |R|^{0.5} \cdot |S|^{0.5} = |R|^{1.5} |S|$$

The AGM Bound

Let's simplify by using $|R_i| \leq |D|$. Then the upper bound in the theorem becomes

$$|q(D)| \leq \prod_{i=1}^{k} |D|^{w(R_i)} = |D|^{\sum_i w(R_i)} = |D|^{\rho^*(q)}$$

Hence we also commonly simplify the runtime we want from a WCOJ algorithm to

$$O\left(f(q) \cdot |D|^{\rho^*(q)}\right)$$

How do common RDBMS engines perform relative to the AGM bound?

Relational Algebra as Execution Plans

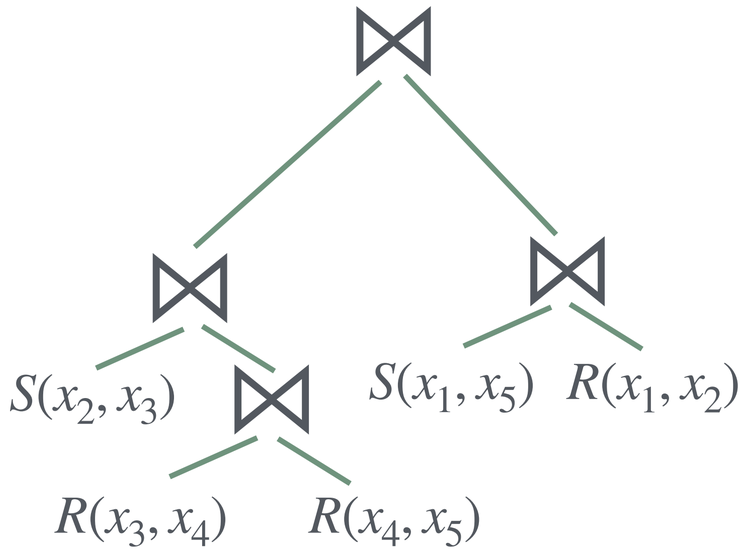
We express execution strategies of database engines in terms of relation algebras. Complexity of the execution strategy will follow from two simple basic facts:

1. The join $R \bowtie S$ can be computed in $\tilde{O}(|R| + |S| + |R \bowtie S|)$ time.
2. A projection $\pi_A(R)$ can be computed in $\tilde{O}(|R|)$ time.

Join Plans and RDBMS

A typical RDBMS processes joins as a tree of joins

$$q_5 := R(x_1, x_2) \wedge S(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_5) \wedge S(x_5, x_1)$$



Formally we say they are evaluated in terms of *join-plans*.

A join plan is a relational algebra expression consisting only of joins (and renaming).

$$(S_{2,3} \bowtie (R_{3,4} \bowtie R_{4,5})) \bowtie (S_{1,5} \bowtie R_{1,2})$$

Join Plans are Suboptimal

Theorem (Atserias, Grohe & Marx; 2008)

For every $m, N \in \mathbb{N}$ there are CQs q with $|q| \geq m$ and databases D with $|D| \geq N$ such that:

1. $\rho^*(q) \leq 2$ and therefore $|q(D)| \leq |D|^2$
2. Every join plan for q has a subplan ψ with $|\psi(D)| \geq |D|^{\frac{1}{3}\log|q|}$

Intuitively, this means that join plans are super-polynomially worse than a worst-case optimal join algorithm. In these queries they have to compute large intermediate results that are much more expensive than the actual output size.

Join-Project Plans

A *join-project plan* for evaluating a query is a relational algebra expression that allows joins as well as unary (to one variable) projections.

It turns out that with a simple join-project plan we can already get very close to worst-case optimality!

Theorem (Atserias, Grohe & Marx; 2008)

Let q be a CQ. There exists a join-project plan for q that can be evaluated in time $O\left(|q|^2 \, |D|^{\rho^*(q)+1} \right)$.

The Join-Project Plan

Suppose a CQ $R_1(\bar{x}_1) \wedge \cdots \wedge R_m(\bar{x}_m)$. For simplicity say all relation names are different and attributes are renamed to match the variables in the CQ. That is, the query is equivalent to the join query $q := R_1 \bowtie \cdots \bowtie R_m$.

Let $A = \{a_1, \ldots, a_n\}$ be the attributes in q and define $B_i = \{a_1, \ldots, a_i\}$.

$$\text{Let } \phi_1 := \left(\cdots(\pi_{B_1}(R_1) \bowtie \pi_{B_1}(R_2)) \bowtie \cdots \bowtie \pi_{B_1}(R_m)\right)$$

$$\phi_{i+1} := \left(\cdots((\phi_i \bowtie \pi_{B_{i+1}}(R_1)) \bowtie \pi_{B_{i+1}}(R_2)) \bowtie \cdots \bowtie \pi_{B_{i+1}}(R_m)\right)$$

Observe that $\phi_i(D) = \pi_i(q(D))$ and thus $q(D) = \phi_m(D)$

The Join-Project Plan

Let $\phi_1 := \left(\cdots(\pi_{B_1}(R_1) \bowtie \pi_{B_1}(R_2)) \bowtie \cdots \bowtie \pi_{B_1}(R_m)\right)$

$\phi_{i+1} := \left(\cdots((\phi_i \bowtie \pi_{B_{i+1}}(R_1)) \bowtie \pi_{B_{i+1}}(R_2)) \bowtie \cdots \bowtie \pi_{B_{i+1}}(R_m)\right)$

Observation 1: For every $i \leq n$, we have $|\phi_i(D)| \leq |D|^{\rho^*(q)}$

Consider $q_i = \pi_{B_i}(R_1) \bowtie \cdots \bowtie \pi_{B_i}(R_m)$, then $\rho^*(q_i) \leq \rho^*(q)$.

And then $|\phi_i(D)| = |q_i(D)| \leq |D|^{\rho^*(q_i)} \leq |D|^{\rho^*(q)}$.

The Join-Project Plan

$$\text{Let } \phi_1 := \left(\cdots(\pi_{B_1}(R_1) \bowtie \pi_{B_1}(R_2)) \bowtie \cdots \bowtie \pi_{B_1}(R_m)\right)$$

$$\phi_{i+1} := \left(\cdots((\phi_i \bowtie \pi_{B_{i+1}}(R_1)) \bowtie \pi_{B_{i+1}}(R_2)) \bowtie \cdots \bowtie \pi_{B_{i+1}}(R_m)\right)$$

Observation 2: All intermediate results in computing ϕ_{i+1} are contained in $\phi_i(D) \times \mathsf{dom}(\mathsf{D})$.

So we can compute each join in $\tilde{O}(|\phi_i(D)| \cdot |D|)$ time.

The Join-Project Plan

$$\text{Let } \phi_1 := \left(\cdots(\pi_{B_1}(R_1) \bowtie \pi_{B_1}(R_2)) \bowtie \cdots \bowtie \pi_{B_1}(R_m)\right)$$

$$\phi_{i+1} := \left(\cdots((\phi_i \bowtie \pi_{B_{i+1}}(R_1)) \bowtie \pi_{B_{i+1}}(R_2)) \bowtie \cdots \bowtie \pi_{B_{i+1}}(R_m)\right)$$

Overall computation required:

$n \cdot m$ projections – each requiring $\tilde{O}(|D|)$ time.

$n \cdot m$ joins – each requiring $\tilde{O}(|D|^{\rho^*(q)+1})$ time.

Conclusion

Practical Implementations

There are more practical algorithms than the join-project plans that we have seen. They leverage more elaborate data structures to get rid of the $+1$ in the exponent and other factors.

This year we are unfortunately out of time to go through the algorithm in detail. If you are interested, the corresponding paper is very readable

> *Veldhuizen, Todd L. "Leapfrog triejoin: A simple, worst-case optimal join algorithm." Proc. International Conference on Database Theory. 2014.*

Exercise Sheets

There is one software project exercise where you implement the 3 WCOJ algorithms we saw today for the triangle query.

There is a theory exercise to dive into the proof of why the second algorithm is indeed worst-case optimal.