A Simple Algorithm for Worst Case Optimal Join and Sampling

***Florent Capelli*, Oliver Irwin, Sylvain Salvati**

CRIL, Université d'Artois

Journées du GT DAAL 2025

13 May 2025

Joining relations

Joining like a pro

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

- Devise a query plan: $(R \bowtie S) \bowtie T$
- Materialize the intermediate joins.

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2

Joining like a pro

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

- Devise a query plan: $(R \bowtie S) \bowtie T$
- Materialize the intermediate joins.

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

$R \bowtie S$	x_1	x_2	x_3
	0	0	0
	0	0	2
	0	1	0
	0	1	2
	2	1	3

Joining like a pro

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

- Devise a query plan: $(R \bowtie S) \bowtie T$
- Materialize the intermediate joins.

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

$R \bowtie S \bowtie T$	x_1	x_2	x_3
	~~0~~	~~0~~	~~0~~
	0	0	2
	0	1	0
	0	1	2
	~~2~~	~~1~~	~~3~~

Joining like a pro

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

- Devise a query plan: $(R \bowtie S) \bowtie T$
- Materialize the intermediate joins.

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

$R \bowtie S \bowtie T$	x_1	x_2	x_3
	0	0	2
	0	1	0
	0	1	2

Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

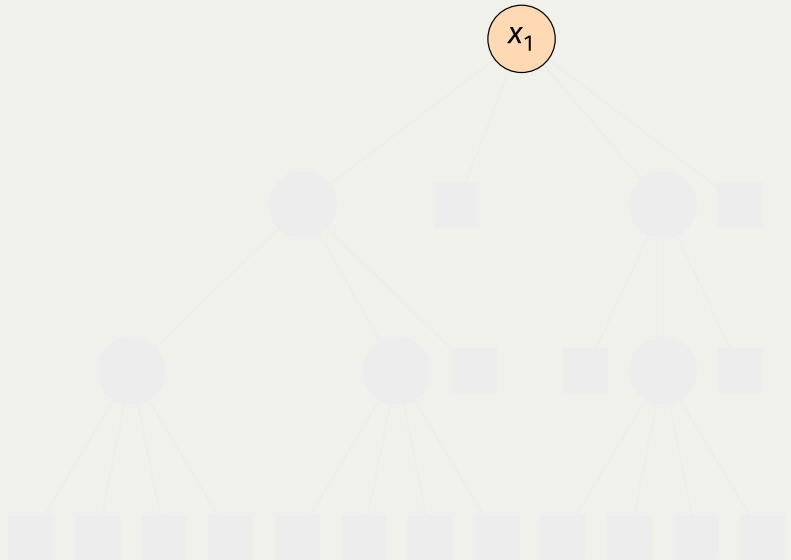
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



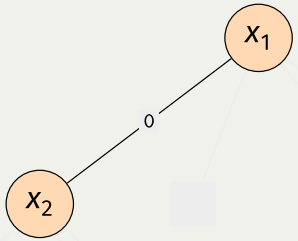
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2



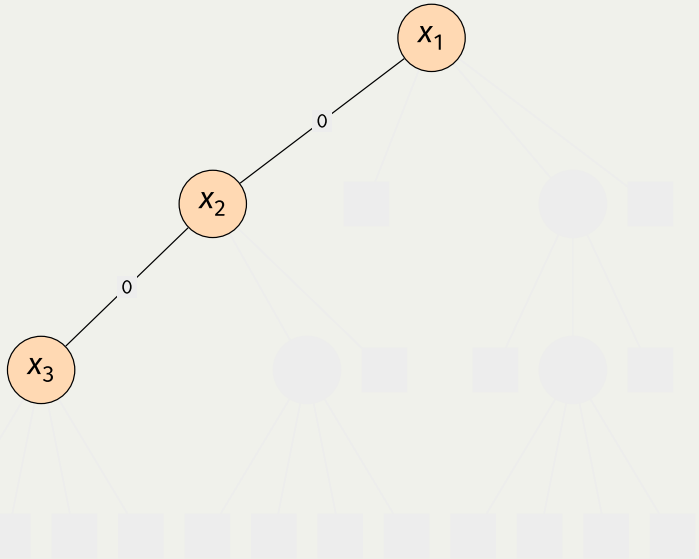
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2



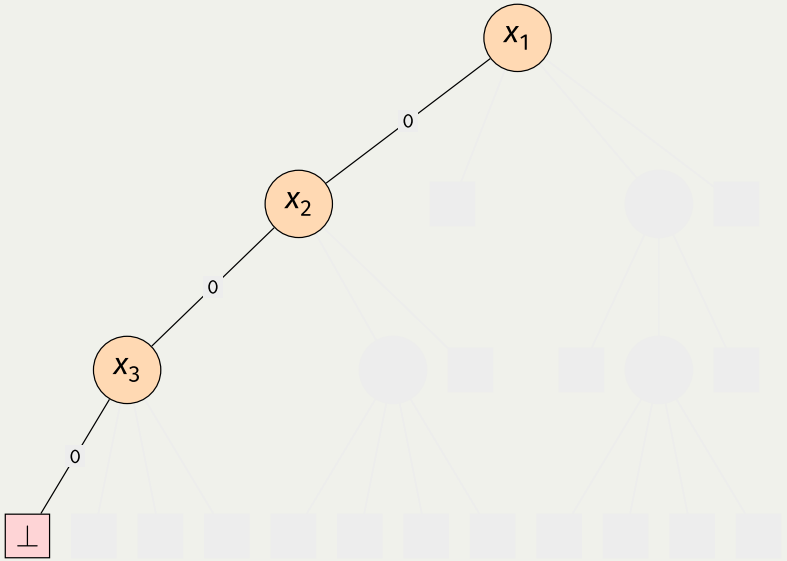
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



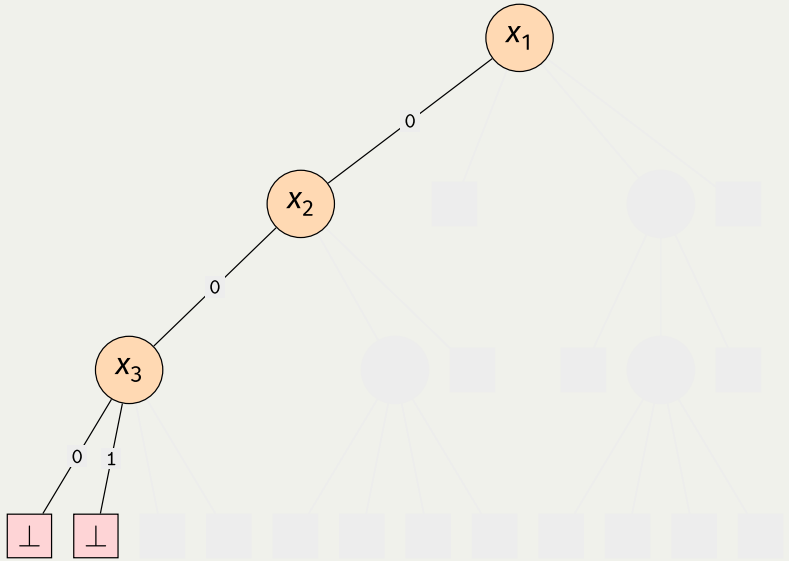
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2



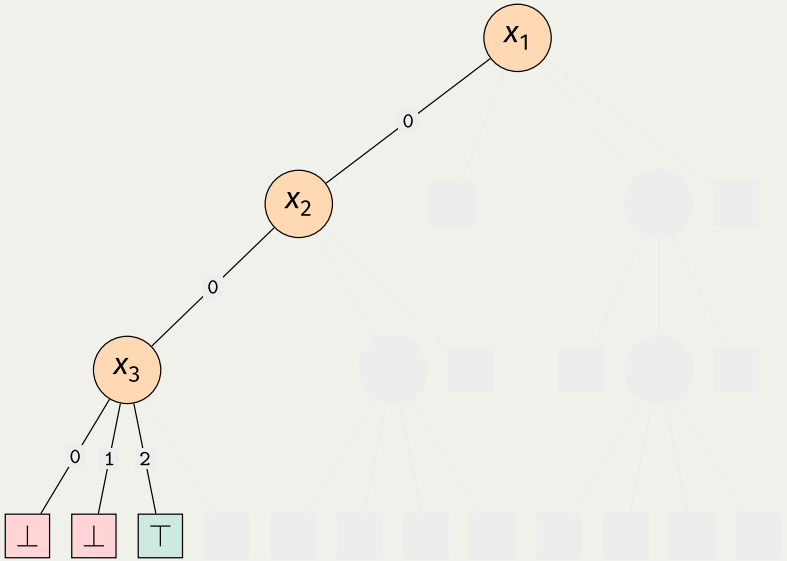
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



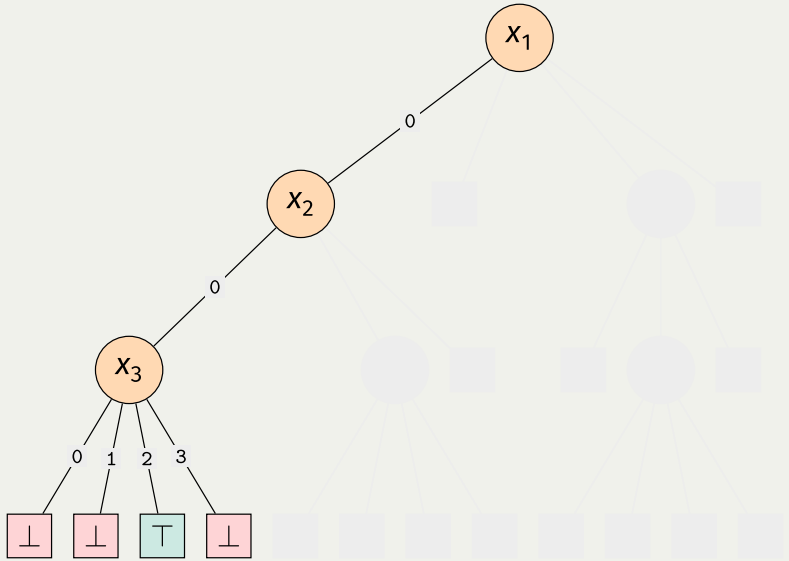
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2



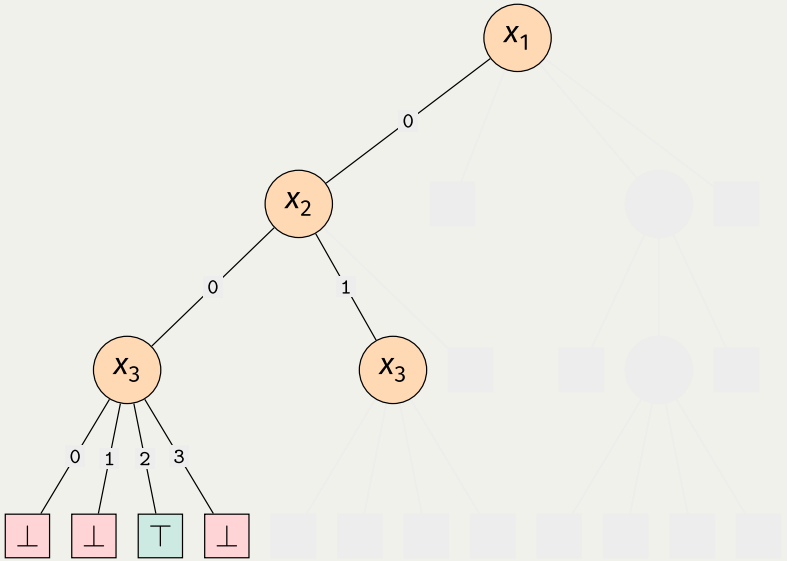
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



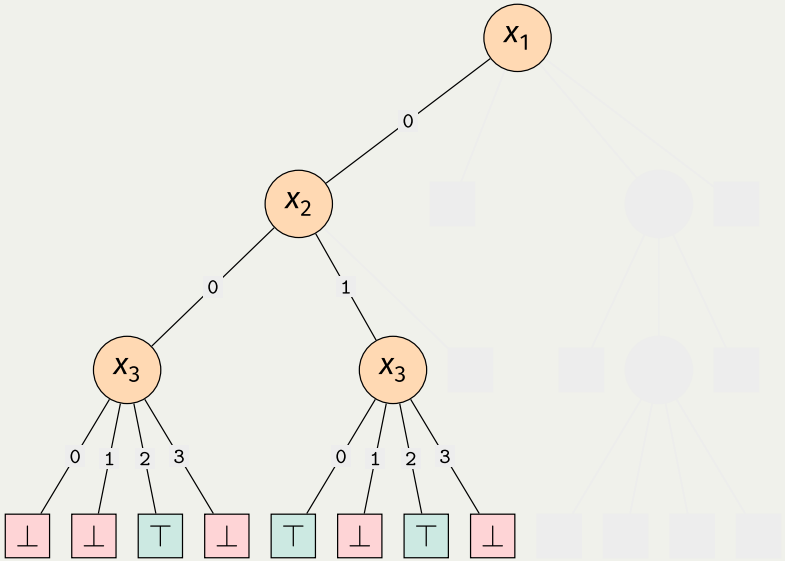
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2



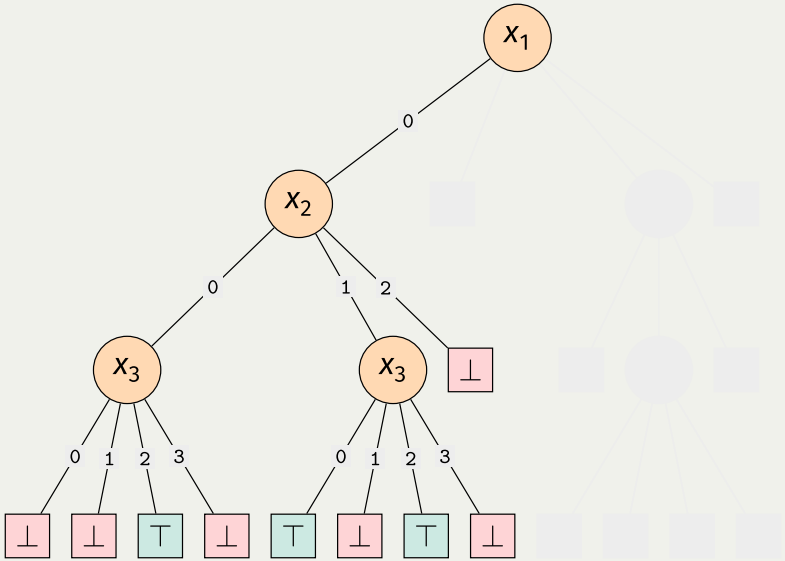
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2



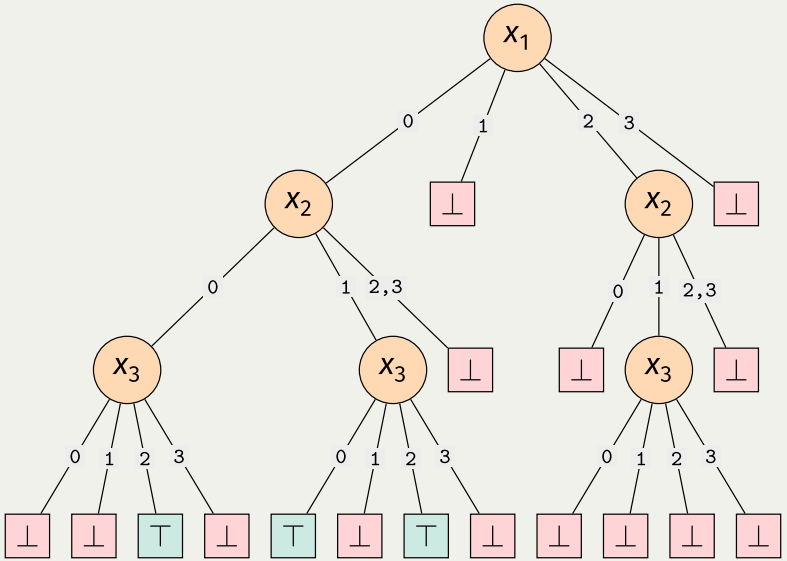
Joining like a brute

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2



Disruptive poll

In theory, is it better to join like:

a. A pro
b. A brute

Disruptive poll

In theory, is it better to join like:

a. A pro
b. A brute

No confidence vote

a. Go home, you are not qualified to talk about joins after saying dumb things like that.
b. We are all theorist here, please tell us the whole story

No confidence vote

a. Go home, you are not qualified to talk about joins after saying dumb things like that.
b. We are all theorist here, please tell us the whole story

What is wrong with joining like a pro

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

- It is known that if $|R^{\mathbb{D}}|, |S^{\mathbb{D}}|, |T^{\mathbb{D}}| \leq N$, then $|Q(\mathbb{D})| \leq N^{1.5}$.
- $R \bowtie S$ may have N^2 answers!

Worst scenario for query plans

Consider $\mathbb{D}$ on domain $D = D_1 \uplus D_2 \uplus D_3$ with:

- $0 \notin D$
- $|D_1| = |D_2| = |D_3| = N$.

R	x_1	x_2
	0	D_2
	D_1	0

S	x_1	x_3
	0	D_3
	D_1	0

T	x_2	x_3
	0	D_3
	D_2	0

- $|R^{\mathbb{D}} \bowtie S^{\mathbb{D}}| \geq N^2$, $|R^{\mathbb{D}} \bowtie T^{\mathbb{D}}| \geq N^2$, $|S^{\mathbb{D}} \bowtie T^{\mathbb{D}}| \geq N^2$
- $|Q(\mathbb{D})| = 0$.

Every query plan will materialize a table of size $O(N^2)$ but the answer table will never be of size greater than $(2N)^{1.5}$.

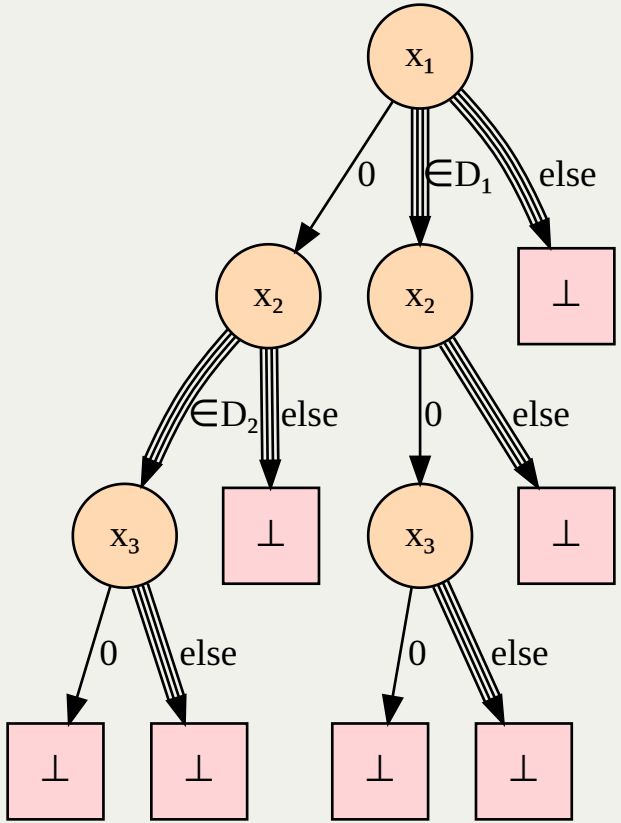
And the brute?

Domain $D = D_1 \uplus D_2 \uplus D_3$ with $0 \notin D$ and $|D_1| = |D_2| = |D_3| = N$.

R	x_1	x_2
	0	D_2
	D_1	0

S	x_1	x_3
	0	D_3
	D_1	0

T	x_2	x_3
	0	D_3
	D_2	0



If we do the "else" branches efficiently (e.g. by reading values from one table), the algorithm makes $O(N)$ recursive calls.

Worst-case optimality

Worst-case optimal join

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

Ideal complexity: output $Q(\mathbb{D})$ in time $O(f(|Q|) \cdot |Q(\mathbb{D})|)$…

… unlikely to be possible.

$f(|Q|)$: data complexity, ie, Q is considered constant. Ideally, f is a reasonable polynomial though.

Worst-case optimal join

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

Ideal complexity: output $Q(\mathbb{D})$ in time $O(f(|Q|) \cdot |Q(\mathbb{D})|)$…

… unlikely to be possible.

Worst case optimal: output $Q(\mathbb{D})$ in time $\tilde{O}(f(|Q|) \cdot N^{1.5})$.

N is the size of the largest input relation and $\tilde{O}(\cdot)$ ignores polylog factors.

$f(|Q|)$: data complexity, ie, Q is considered constant. Ideally, f is a reasonable polynomial though.

Worst case value

Consider a join query Q and all databases for Q with a bound N on the table size:

$$\mathcal{D}_Q^{\leq N} = \{\mathbb{D} \mid \forall R \in Q, |R^{\mathbb{D}}| \leq N\}$$

and let:

$$\mathsf{wc}(Q, N) = \mathsf{sup}_{\mathbb{D} \in \mathcal{D}_Q^{\leq N}} |Q(\mathbb{D})|$$

$\mathsf{wc}(Q, N)$ **is the worst case: the size of the biggest answer set possible with query Q and databases where each table are bounded by N.**

Worst case examples

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$.

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$.
- **Similarly**: $Q_k = R_1(x_1) \wedge ... \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$.

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$.
- **Similarly**: $Q_k = R_1(x_1) \wedge ... \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$.
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$ has $\mathsf{wc}(Q_\square, N) = N^2$.

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$.
- **Similarly**: $Q_k = R_1(x_1) \wedge ... \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$.
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$ has $\mathsf{wc}(Q_\square, N) = N^2$.
- **Triangle query**: $Q_\Delta = R(x, y) \wedge S(x, z) \wedge T(y, z)$, $\mathsf{wc}(Q_\Delta, N) = N^{1.5}$.

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$.
- **Similarly**: $Q_k = R_1(x_1) \wedge ... \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$.
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$ has $\mathsf{wc}(Q_\square, N) = N^2$.
- **Triangle query**: $Q_\Delta = R(x, y) \wedge S(x, z) \wedge T(y, z)$, $\mathsf{wc}(Q_\Delta, N) = N^{1.5}$.
- **The n-cycle**: $Q_{C_n}(x_1, ..., x_n) = R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge ... \wedge R_n(x_n, x_1)$: $\mathsf{wc}(Q_{C_n}) = N^{\frac{n}{2}}$.

Worst case examples

- **Cartesian product**: $Q_2 = R_1(x_1) \wedge R_2(x_2)$ has $\mathsf{wc}(Q_2, N) = N^2$.
- **Similarly**: $Q_k = R_1(x_1) \wedge ... \wedge R_k(x_k)$ has $\mathsf{wc}(Q_2, N) = N^k$.
- **Square query**: $Q_\square = R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_3, x_4) \wedge R(x_4, x_1)$ has $\mathsf{wc}(Q_\square, N) = N^2$.
- **Triangle query**: $Q_\Delta = R(x, y) \wedge S(x, z) \wedge T(y, z)$, $\mathsf{wc}(Q_\Delta, N) = N^{1.5}$.
- **The n-cycle**: $Q_{C_n}(x_1, ..., x_n) = R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge ... \wedge R_n(x_n, x_1)$: $\mathsf{wc}(Q_{C_n}) = N^{\frac{n}{2}}$.

We know how to compute $\rho(Q)$ such that $\mathsf{wc}(Q, N) = \tilde{O}(N^{\rho(Q)})$ but we do not need it!

This is known as the AGM-bound

Worst case optimal join (WCOJ) algorithms

A join algorithm is **worst case optimal** (wrt $\mathcal{D}_Q^{\leq N}$) if for every Q, $N \in \mathbb{N}$ and $\mathbb{D} \in \mathcal{D}_Q^{\leq N}$, it computes $Q(\mathbb{D})$ in time

$$\tilde{O}(f(|Q|) \cdot \mathsf{wc}(Q, N))$$

- Data complexity model: Q considered constant hence $f(|Q|)$ also.
- In this talk, f will be a reasonable polynomial!

The DBMS approach is not worst case optimal (triangle example from before).

Existing WCOJ Algorithm

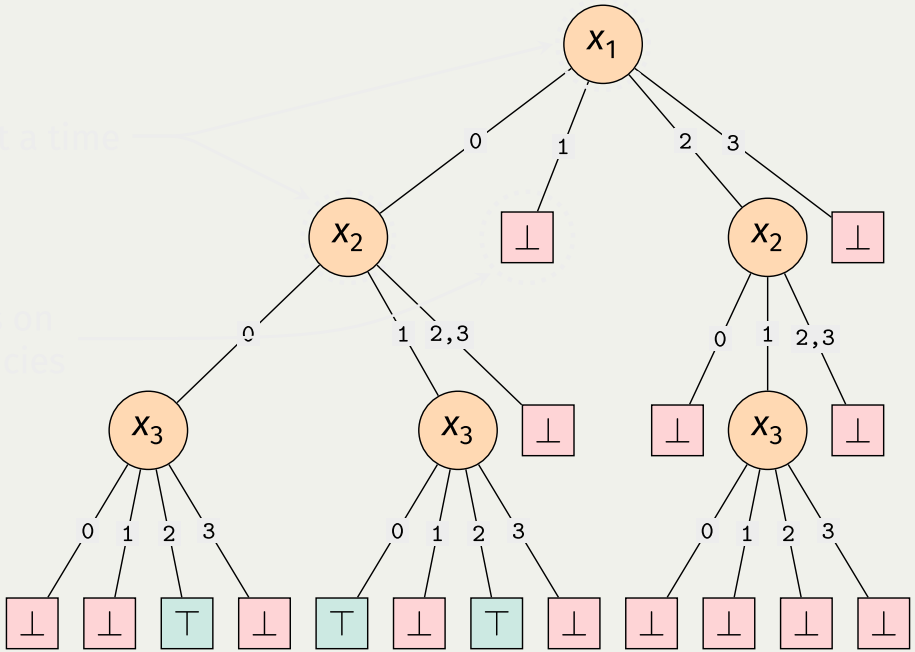
Rich literature:

- **NPRR join** (Ngo, Porat, Ré, Rudra, PODS12): usual join plans but with relations partitionned into high/low degree tuples.
- **Leapfrog Triejoin** (Veldhuizen, ICDT14)
- **Generic Join** (Ngo, PODS18): both branch and bound algorithms as ours but more complex analysis/data structures.
- **PANDA** (PODS17): handle complex database constraints, very complex, long analysis.

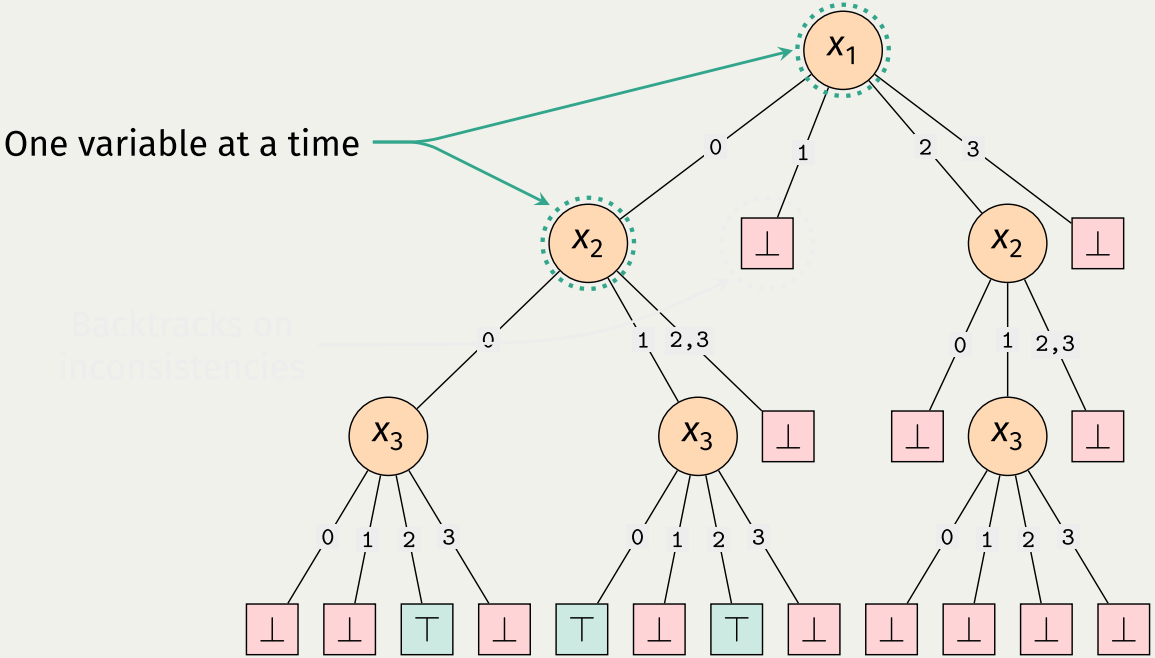
We prove the worst case optimality of the branch and bound algorithm in an elementary way.

Analysing the brute

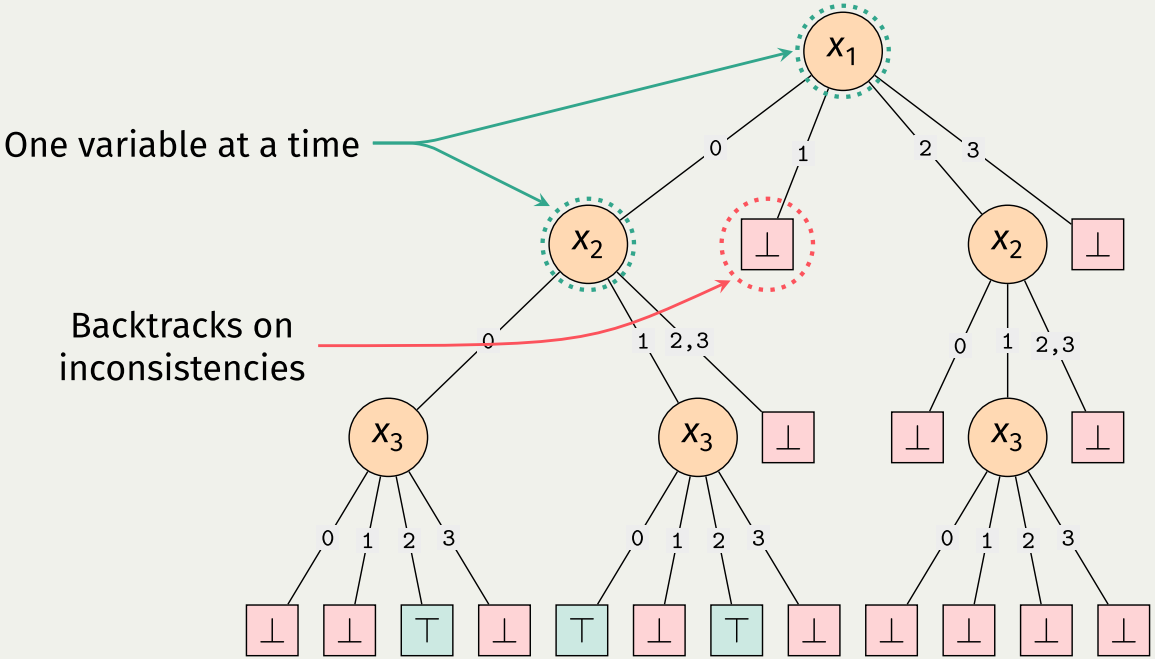
Algorithm reminder



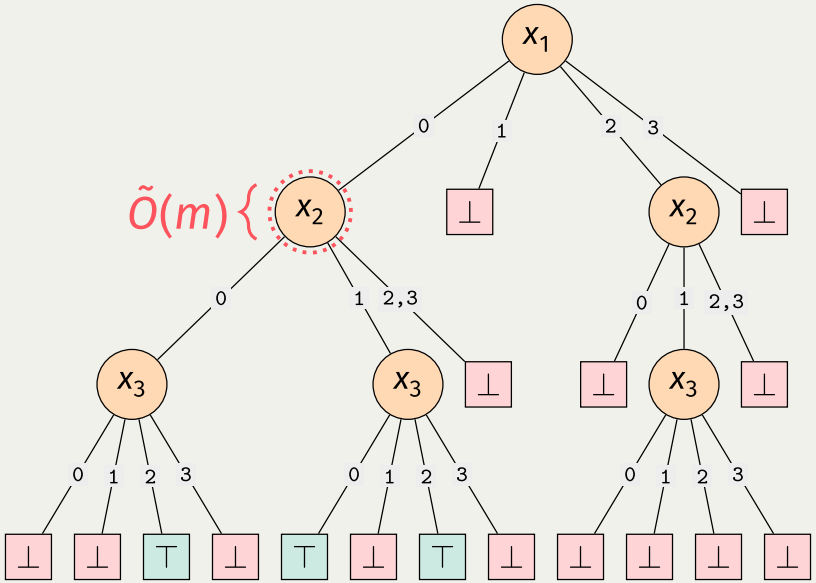
Algorithm reminder



Algorithm reminder



Complexity analysis



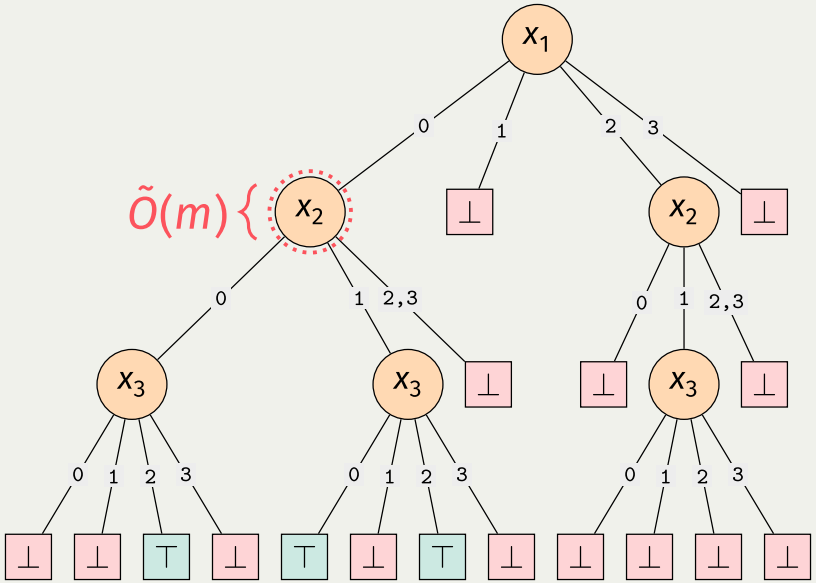
One recursive call:

- branch variable x_i on value $d \in \text{dom}$
- filter/project relations with x_i
- Binary search in $O(\log|R|)$ if R ordered ($O(1)$ possible using tries).

R	x_1	x_2	
▶	0	0	
	0	2	
	1	0	
	1	1	
	2	0	
	2	1	◀

Total complexity: number of recursive calls times $\tilde{O}(m)$ where m is the number of atoms.

Complexity analysis



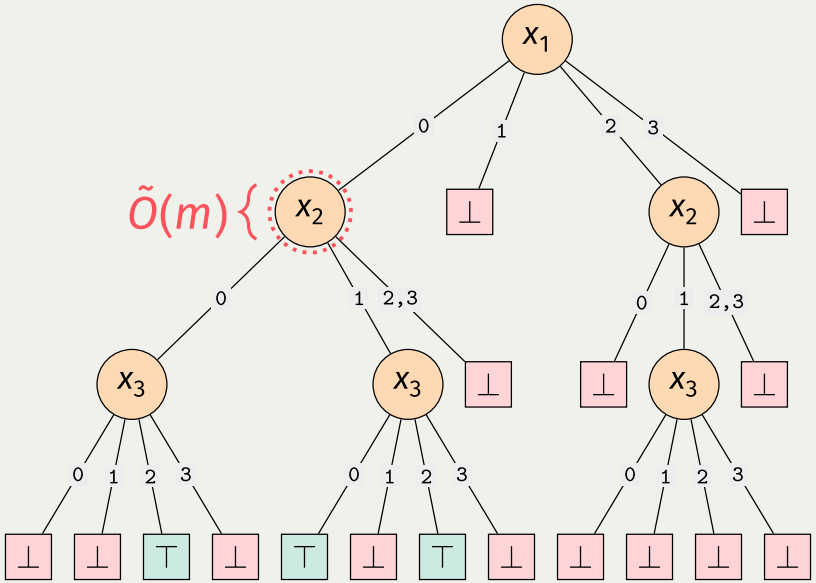
One recursive call:

- branch variable x_i on value $d \in \text{dom}$
- filter/project relations with x_i
- Binary search in $O(\log|R|)$ if R ordered ($O(1)$ possible using tries).

R	x_1	x_2	
	0	0	
	0	2	
▶	1	0	
	1	1	
	2	0	
	2	1	◀

Total complexity: number of recursive calls times $\tilde{O}(m)$ where m is the number of atoms.

Complexity analysis



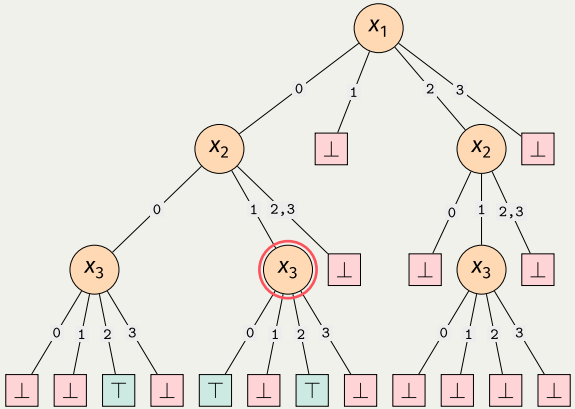
One recursive call:

- branch variable x_i on value $d \in \text{dom}$
- filter/project relations with x_i
- Binary search in $O(\log|R|)$ if R ordered ($O(1)$ possible using tries).

R	x_1	x_2	
	0	0	
	0	2	
▶	1	0	
	1	1	◀
	2	0	
	2	1	

Total complexity: number of recursive calls times $\tilde{O}(m)$ where m is the number of atoms.

Number of calls: example



- Nodes: partial assignment τ
- Here: $\tau := \{x_1 = 0, x_2 = 1\}$
- **Not** $\boxed{\bot}$ node: partial assignment **compatible with every relation**
- τ solution of $Q_2(\mathbb{D}_2)$: project on x_1, x_2.
- At most: $|Q_2(\mathbb{D}_2)|$ such nodes at level 3

$$Q := R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_3)$$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2

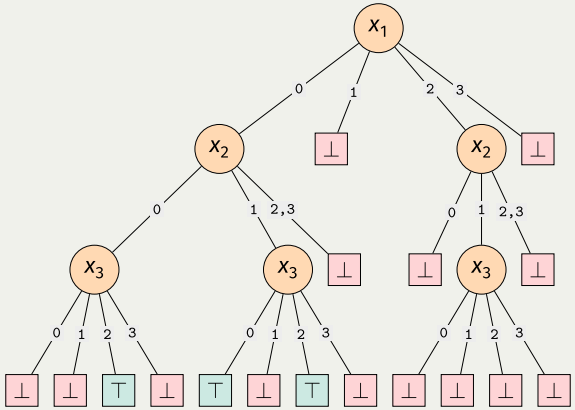
$$Q_2 := R_2(x_1, x_2) \wedge S_2(x_1) \wedge T_2(x_2)$$

R_2 x_1	x_2
0	0
0	1
2	1

S_2 x_1
0
2

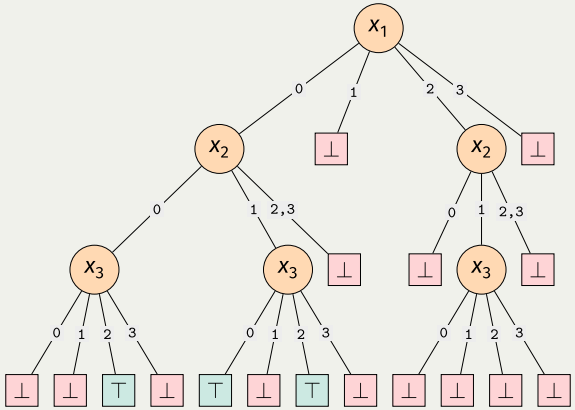
T_2 x_2
0
1

Number of calls in general



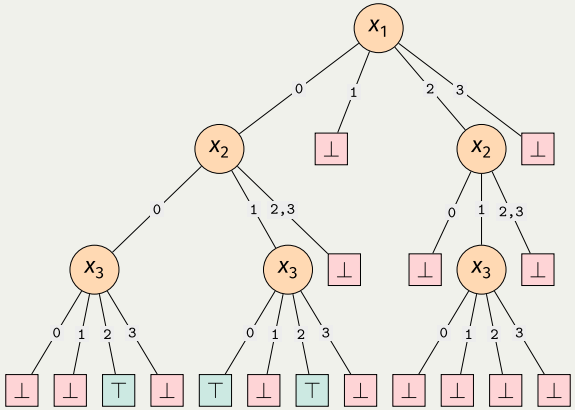
- a call = a node = a partial assignment.
- $\tau := x_1 = d_1, ..., x_i = d_i$ current call, not $\boxed{\bot}$:
  - No inconsistency.
  - $R^{\mathbb{D}}[\tau]$ not empty for each $R \in Q$
  - $\tau \in Q_i(\mathbb{D})$ for $Q_i = \bigwedge_{R \in Q} \prod_{x_1 ... x_i} R$
  - $\leq \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ such nodes!

Number of calls in general



- a call = a node = a partial assignment.
- $\tau := x_1 = d_1, ..., x_i = d_i$ current call, not $\boxed{\bot}$:
  - No inconsistency.
  - $R^{\mathbb{D}}[\tau]$ not empty for each $R \in Q$
  - $\tau \in Q_i(\mathbb{D})$ for $Q_i = \bigwedge_{R \in Q} \prod_{x_1 ... x_i} R$
  - $\leq \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ such nodes!
- $\tau := x_1 = d_1, ..., x_{i+1} = d_{i+1}$ current call is $\boxed{\bot}$:
  - $x_1 = d_1, ..., x_i = d_i$ is **not** $\bot$.
  - $\leq |\mathsf{dom}| \cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ $\bot$-nodes!

Number of calls in general



- a call = a node = a partial assignment.
- $\tau := x_1 = d_1, ..., x_i = d_i$ current call, not $\boxed{\bot}$:
  - No inconsistency.
  - $R^{\mathbb{D}}[\tau]$ not empty for each $R \in Q$
  - $\tau \in Q_i(\mathbb{D})$ for $Q_i = \bigwedge_{R \in Q} \prod_{x_1 ... x_i} R$
  - $\leq \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ such nodes!
- $\tau := x_1 = d_1, ..., x_{i+1} = d_{i+1}$ current call is $\boxed{\bot}$:
  - $x_1 = d_1, ..., x_i = d_i$ is **not** $\bot$.
  - $\leq |\mathsf{dom}| \cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ $\bot$-nodes!

At most $(|\mathsf{dom}|+1) \sum_{i=1}^{n} |Q_i(\mathbb{D})|$ **calls.**

Complexity: $\tilde{O}(m|\mathsf{dom}| \cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|)$.

Toward worst case optimality

$$|Q_i(\mathbb{D})| = |\bigwedge_{R \in Q} \prod_{x_1 \ldots x_i} R^{\mathbb{D}}|$$
$$= |\bigwedge_{R \in Q} R^{\mathbb{D}'}|$$
$$= |Q(\mathbb{D}')|$$

where $R^{\mathbb{D}'} = \prod_{x_1 \ldots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \ldots, x_i}$

$\mathbb{D}$

R x_1	x_2
0	0
0	1
2	1

S x_1	x_3
0	0
0	2
2	3

T x_2	x_3
0	2
1	0
1	2

$\mathbb{D}_2$

R_2 x_1	x_2
0	0
0	1
2	1

S_2 x_1
0
2

T_2 x_2
0
1

$\mathbb{D}'$

R' x_1	x_2
0	0
0	1
2	1

S' x_1	x_3
0	**0**
2	**0**

T' x_2	x_3
0	**0**
1	**0**

Toward worst case optimality

$$|Q_i(\mathbb{D})| = |\bigwedge_{R \in Q} \prod_{x_1 \ldots x_i} R^{\mathbb{D}}|$$
$$= |\bigwedge_{R \in Q} R^{\mathbb{D}'}|$$
$$= |Q(\mathbb{D}')|$$

where $R^{\mathbb{D}'} = \prod_{x_1 \ldots x_i} R^{\mathbb{D}} \times \{0\}^{X_R \setminus x_1, \ldots, x_i}$

Crucial observation:

- $|R^{\mathbb{D}'}| = |\prod_{x_1 \ldots x_i} R^{\mathbb{D}}| \leq |R^{\mathbb{D}}| \leq N$
- Hence $\mathbb{D}' \in \mathcal{D}_Q^{\leq N}$.
- $|Q_i(\mathbb{D})| = |Q(\mathbb{D}')| \leq \mathsf{wc}(Q, N)$

$\mathbb{D}$

R	x_1	x_2
	0	0
	0	1
	2	1

S	x_1	x_3
	0	0
	0	2
	2	3

T	x_2	x_3
	0	2
	1	0
	1	2

$\mathbb{D}_2$

R_2	x_1	x_2
	0	0
	0	1
	2	1

S_2	x_1
	0
	2

T_2	x_2
	0
	1

$\mathbb{D}'$

R'	x_1	x_2
	0	0
	0	1
	2	1

S'	x_1	x_3
	0	**0**
	2	**0**

T'	x_2	x_3
	0	**0**
	1	**0**

Branch and bound complexity

$$|Q_i(\mathbb{D})| \leq wc(Q, N)$$

The complexity of the branch and bound algorithm is

Branch and bound complexity

$$|Q_i(\mathbb{D})| \leq wc(Q, N)$$

The complexity of the branch and bound algorithm is

$$\tilde{O}(m|\mathsf{dom}|\cdot \sum_{i=1}^{n} |Q_i(\mathbb{D})|)$$

Branch and bound complexity

$$|Q_i(\mathbb{D})| \leq wc(Q, N)$$

The complexity of the branch and bound algorithm is

$$\tilde{O}(m|\mathsf{dom}|\cdot n\mathsf{wc}(Q, N))$$

Branch and bound complexity

$$|Q_i(\mathbb{D})| \leq wc(Q, N)$$

The complexity of the branch and bound algorithm is

$$\tilde{O}(mn \cdot |\mathsf{dom}| \cdot \mathsf{wc}(Q, N))$$

Branch and bound complexity

$$|Q_i(\mathbb{D})| \leq wc(Q, N)$$

The complexity of the branch and bound algorithm is

$$\tilde{O}(mn \cdot |\mathsf{dom}| \cdot \mathsf{wc}(Q, N))$$

We do not even need to know $\mathsf{wc}(Q, N)$ to prove it!

Branch and bound complexity

$$|Q_i(\mathbb{D})| \leq wc(Q, N)$$

The complexity of the branch and bound algorithm is

$$\tilde{O}(mn \cdot |\mathsf{dom}| \cdot \mathsf{wc}(Q, N))$$

We do not even need to know $\mathsf{wc}(Q, N)$ to prove it!

Make the domain binary!

R	x	y
	1	2
	2	1
	3	0

$\rightsquigarrow$

$\tilde{R}^b$	x^2	x^1	x^0	y^2	y^1	y^0
	0	0	1	0	1	0
	0	1	0	0	0	1
	0	1	1	0	0	0

- $Q \rightsquigarrow \tilde{Q}^b$ has bn variables
- $\mathbb{D} \rightsquigarrow \tilde{\mathbb{D}}^b$ for $b = \log|\mathsf{dom}|$. Database has roughly the same bitsize but size 2 domain!

WCOJ finally

- To compute $Q(\mathbb{D})$ run simple branch and bound algorithm on $(\tilde{Q}^b, \tilde{\mathbb{D}}^b)$:
  - runs in time $\tilde{O}(m \cdot (n\log|\mathsf{dom}|) \cdot 2\mathsf{wc}(\tilde{Q}^b, N, 2))$
  - where $\mathsf{wc}(\tilde{Q}^b, N, 2)$ is the worst case for $\tilde{Q}^b$ on relations of size $\leq N$ and domain 2.
  - $\mathsf{wc}(\tilde{Q}^b, N, 2) \leq \mathsf{wc}(Q, N)$ by reconverting back to larger domain.

We hence compute $Q(\mathbb{D})$ in time $\tilde{O}(mn \cdot \mathsf{wc}(Q, N))$!

More on Binarization

- Explicit binarization is not necessary: fix one bit of the variables at a time and filter $\mathbb{D}$.
- Using bigger bases (bytes, words etc.):
  - depth/width tradeoff
  - may be interesting in practice
  - Interesting base: $\log|\mathsf{dom}|$.

Sampling answers uniformly

Problem statement

Given Q and $\mathbb{D}$, sample $\tau \in Q(\mathbb{D})$ with probability $\frac{1}{|Q(\mathbb{D})|}$ or fail if $Q(\mathbb{D}) = \emptyset$.

Naive algorithm:

- materialize $Q(\mathbb{D})$ in a table
- sample $i \leq |Q(\mathbb{D})|$ uniformly
- output $Q(\mathbb{D})[i]$.

Complexity using WCOJ: $\tilde{O}(\mathsf{wc}(Q, N)poly(|Q|))$.

Problem statement

Given Q and $\mathbb{D}$, sample $\tau \in Q(\mathbb{D})$ with probability $\frac{1}{|Q(\mathbb{D})|}$ or fail if $Q(\mathbb{D}) = \emptyset$.

Naive algorithm:

- materialize $Q(\mathbb{D})$ in a table
- sample $i \leq |Q(\mathbb{D})|$ uniformly
- output $Q(\mathbb{D})[i]$.

Complexity using WCOJ: $\tilde{O}(\mathsf{wc}(Q, N)poly(|Q|))$.

We can do better: (expected) time $\tilde{O}(\frac{\mathsf{wc}(Q,N)}{|Q(\mathbb{D})|+1}poly(|Q|))$

PODS 23: [Deng, Lu, Tao] and [Kim, Ha, Fletcher, Han]

Problem statement

Given Q and $\mathbb{D}$, sample $\tau \in Q(\mathbb{D})$ with probability $\frac{1}{|Q(\mathbb{D})|}$ or fail if $Q(\mathbb{D}) = \emptyset$.

Naive algorithm:

- materialize $Q(\mathbb{D})$ in a table
- sample $i \leq |Q(\mathbb{D})|$ uniformly
- output $Q(\mathbb{D})[i]$.

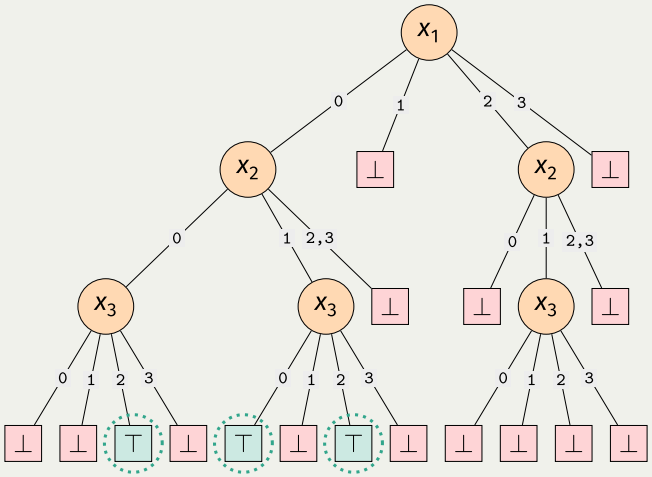
Complexity using WCOJ: $\tilde{O}(\mathsf{wc}(Q, N)poly(|Q|))$.

We can do better: (expected) time $\tilde{O}(\frac{\mathsf{wc}(Q,N)}{|Q(\mathbb{D})|+1}poly(|Q|))$

PODS 23: [Deng, Lu, Tao] and [Kim, Ha, Fletcher, Han]

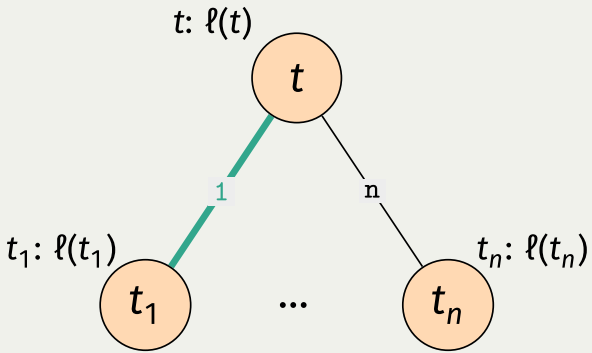
Let's do a modular proof of this fact!

Revisiting the problem



Sampling answers reduces to sampling $\boxed{\top}$-leaves in a tree with ($\boxed{\top}$,$\boxed{\bot}$)-labeled leaves.

Sampling leaves, the easy way

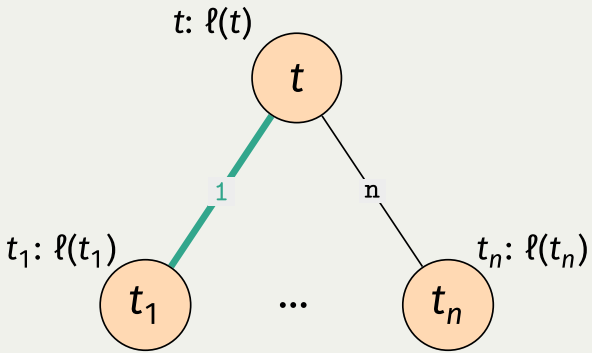


- $\ell(t)$: number of $\boxed{\top}$-leaves below t is known
- Recursively sample uniformly a $\top$-leaf in t_i with probability $\frac{\ell(t_i)}{\ell(t)}$.
- A leaf in $\ell(t_i)$ will hence be sampled with probability

$$\frac{1}{\ell(t_i)} \times \frac{\ell(t_i)}{\ell(t)} = \frac{1}{\ell(t)}$$

Uniform!

Sampling leaves, the easy way



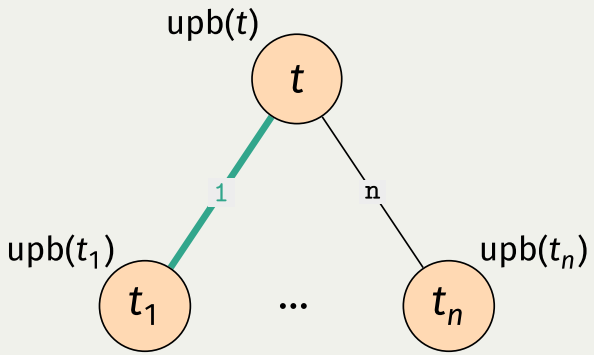
In our case, we do not know $\ell(t)$…

- $\ell(t)$: number of $\boxed{\top}$-leaves below t is known
- Recursively sample uniformly a $\top$-leaf in t_i with probability $\frac{\ell(t_i)}{\ell(t)}$.
- A leaf in $\ell(t_i)$ will hence be sampled with probability

$$\frac{1}{\ell(t_i)} \times \frac{\ell(t_i)}{\ell(t)} = \frac{1}{\ell(t)}$$

Uniform!

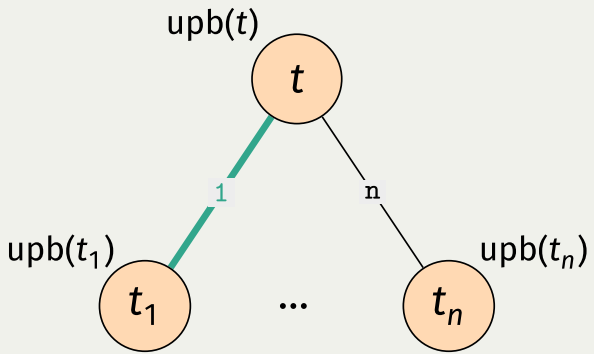
Sampling leaves with a nice oracle



- $upb(t)$: **upperbound** on the number of $\boxed{\top}$-leaves below t is known
- Recursively sample uniformly a $\top$-leaf in t_i with probability $\frac{upb(t_i)}{upb(t)}$.
- **Fail** with probability $1 - \sum_i \frac{upb(t_i)}{upb(t)}$ or upon encountering $\boxed{\bot}$.

Only makes sense if $\sum_i upb(t_i) \leq upb(t)$**.**

Sampling leaves with a nice oracle



- $upb(t)$: **upperbound** on the number of $\boxed{\top}$-leaves below t is known
- Recursively sample uniformly a $\top$-leaf in t_i with probability $\frac{upb(t_i)}{upb(t)}$.
- **Fail** with probability $1 - \sum_i \frac{upb(t_i)}{upb(t)}$ or upon encountering $\boxed{\bot}$.

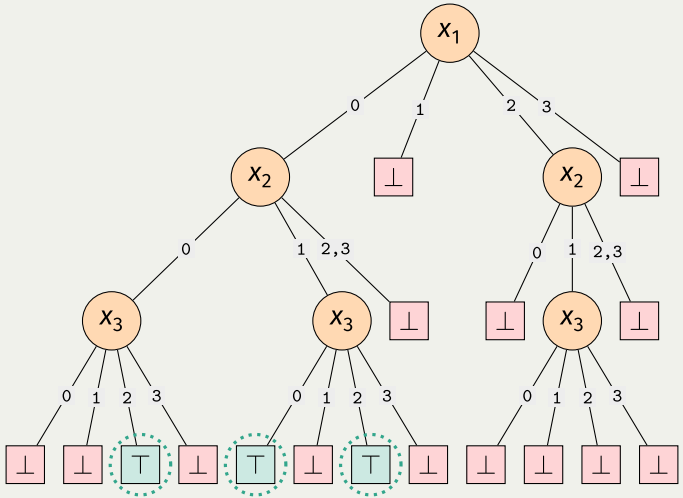
Only makes sense if $\sum_i upb(t_i) \leq upb(t)$**.**

Las Vegas uniform sampling algorithm:

- each leaf is output with probability $\frac{1}{ubp(t)}$,
- fails with proba $1 - \frac{\ell(t)}{upb(t)}$ where $\ell(t)$ is the number of $\boxed{\top}$-leaves under t.

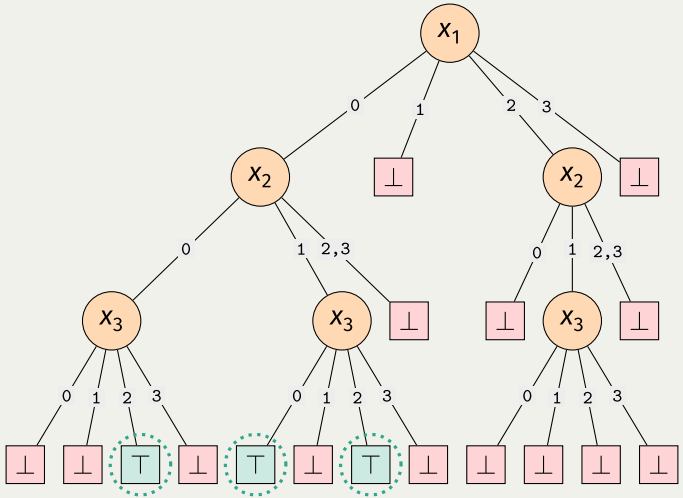
Repeat until output: $O(\frac{upb(r)}{\ell(r)})$ expected calls, where r is the root.

Upper bound oracles for conjunctive queries



- Node t: partial assignment $\tau_t := (x_1 = d_1, ..., x_i = d_i)$
- Number of $\boxed{\top}$ leaves below t: $|Q(\mathbb{D})[\tau_t]|$.
- $upb(t)$???: **look for worst case bounds**!

Upper bound oracles for conjunctive queries

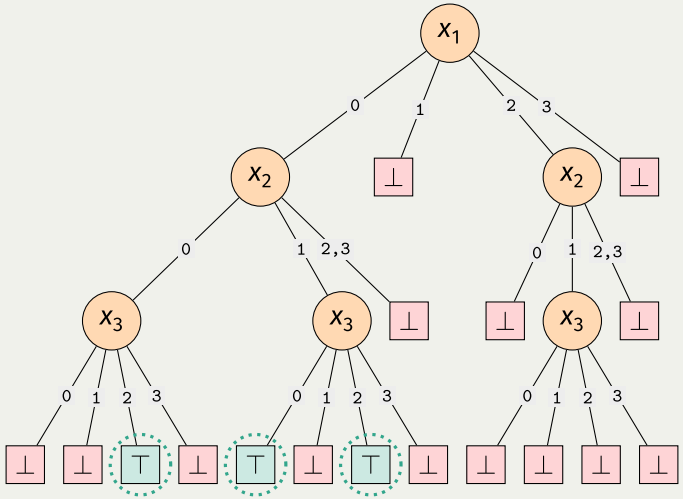


- Node t: partial assignment $\tau_t := (x_1 = d_1, ..., x_i = d_i)$
- Number of $\boxed{\top}$ leaves below t: $|Q(\mathbb{D})[\tau_t]|$.
- $upb(t)$???: **look for worst case bounds**!

AGM bound: there exists positive rational numbers $(\lambda_R)_{R \in Q}$ such that

$$|Q(\mathbb{D})| \leq \prod_{R \in Q} |R^{\mathbb{D}}|^{\lambda_R} \leq \mathsf{wc}(Q, N)$$

Upper bound oracles for conjunctive queries



- Node t: partial assignment $\tau_t := (x_1 = d_1, ..., x_i = d_i)$
- Number of $\boxed{\top}$ leaves below t: $|Q(\mathbb{D})[\tau_t]|$.
- $upb(t)$???: **look for worst case bounds**!

AGM bound: there exists positive rational numbers $(\lambda_R)_{R \in Q}$ such that

$$|Q(\mathbb{D})| \leq \prod_{R \in Q} |R^{\mathbb{D}}|^{\lambda_R} \leq \mathsf{wc}(Q, N)$$

Define $upb(t) = \prod_{R \in Q} |R^{\mathbb{D}}[\tau_t]|^{\lambda_R} \leq \mathsf{wc}(Q, N)$:

- it is an upper bound on $|Q(\mathbb{D})[\tau_t]|$,
- it is supperadditive: $upb(t) \geq \sum_{d \in \mathsf{dom}} upb(t_d)$
- value of upb at the root of the tree: $\mathsf{wc}(Q, N)$!

Wrapping up sampling

Given a super-additive function upperbounding the number of $\boxed{\top}$-leaves in a tree at each node, we have:

Las Vegas uniform sampling algorithm:

- each leaf is output with probability $\frac{1}{ubp(t)}$
- fails with proba $1 - \frac{\ell(t)}{upb(t)}$

Repeat until output: $O(\frac{upb(r)}{\ell(r)})$ expected calls.

Wrapping up sampling

Given a super-additive function upperbounding the number of $\boxed{\top}$-leaves in a tree at each node, we have:

Las Vegas uniform sampling algorithm:

- each leaf **/answer** is output with probability $\frac{1}{ubp(t)} = \frac{1}{wc(Q,N)}$
- fails with proba $1 - \frac{\ell(t)}{upb(t)} = 1 - \frac{|Q(\mathbb{D})|}{wc(Q,N)}$

Repeat until output: $O(\frac{upb(r)}{\ell(r)}) = \frac{\mathsf{wc}(Q,N)}{1+|Q(\mathbb{D})|}$ expected calls.

Wrapping up sampling

Given a super-additive function upperbounding the number of $\boxed{\top}$-leaves in a tree at each node, we have:

Las Vegas uniform sampling algorithm:

- each leaf **/answer** is output with probability $\frac{1}{ubp(t)} = \frac{1}{wc(Q,N)}$
- fails with proba $1 - \frac{\ell(t)}{upb(t)} = 1 - \frac{|Q(\mathbb{D})|}{wc(Q,N)}$

Repeat until output: $O(\frac{upb(r)}{\ell(r)}) = \frac{\mathsf{wc}(Q,N)}{1+|Q(\mathbb{D})|}$ expected calls.

Final complexity: binarize to navigate the tree in $\tilde{O}(nm)$: $\tilde{O}(nm \cdot \frac{\mathsf{wc}(Q,N)}{1+|Q(\mathbb{D})|})$

Matches existing results, proof more modular.

Beyond Cardinality Constraints

Worst case and constraints

So far we have considered worst case wrt this class:

- $\mathcal{D}_Q^{\leq N} = \{\mathbb{D} \mid \forall R \in Q, |R^{\mathbb{D}}| \leq N\}$
- $\mathsf{wc}(Q, N) = \sup_{\mathbb{D} \in \mathcal{D}_Q^{\leq N}} |Q(\mathbb{D})|$

Each relation is subject to a cardinality constraint of size N.

What if we know that our instance has some extra properties (e.g., a *functional dependency*)

- We know $\mathbb{D} \in \mathcal{C} \subseteq \mathcal{D}_Q^{\leq N}$
- We want the join to run in $\tilde{O}(f(|Q|) \cdot \mathsf{wc}(Q, \mathcal{C}))$ where $\mathsf{wc}(Q, \mathcal{C}) := \sup_{\mathbb{D} \in \mathcal{C}} |Q(\mathbb{D})|$.

In this case, we say that our algorithm is worst case optimal wrt $\mathcal{C}$.

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

- Let $\mathcal{C}$ be the class of databases where $|R| \leq N, |S| \leq N$ and R respect functional dependency $x_2 \to x_1$.
- $\mathsf{wc}(Q, \mathcal{C}) \leq N$ because each tuple of $S^{\mathbb{D}}$ can be extended to *at most one solution*.

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

- Let $\mathcal{C}$ be the class of databases where $|R| \leq N, |S| \leq N$ and R respect functional dependency $x_2 \to x_1$.
- $\mathsf{wc}(Q, \mathcal{C}) \leq N$ because each tuple of $S^{\mathbb{D}}$ can be extended to *at most one solution*.

Is our simple join worst case optimal for this class?

Finer constraints can help

$Q = R(x_1, x_2) \wedge S(x_2, x_3)$.

We have: $\mathsf{wc}(Q, N) = N^2$.

- Let $\mathcal{C}$ be the class of databases where $|R| \leq N, |S| \leq N$ and R respect functional dependency $x_2 \to x_1$.
- $\mathsf{wc}(Q, \mathcal{C}) \leq N$ because each tuple of $S^{\mathbb{D}}$ can be extended to *at most one solution*.

Is our simple join worst case optimal for this class?

Short answer: yes if x_2 is set before x_1.

Prefix closed classes

Recall the complexity of our algorithm: $\tilde{O}(m|\mathsf{dom}|\sum_{i=1}^{n}|Q_i(\mathbb{D})|))$ where $Q_i = \bigwedge_{R \in Q} \prod_{x_1,\ldots,x_i} R$

A class of database $\mathcal{C}$ for Q is **prefix closed for order** $\pi = (x_1, \ldots, x_n)$ if for each i and $\mathbb{D} \in \mathcal{C}$:

$$|Q_i(\mathbb{D})| \leq \mathsf{wc}(\mathcal{C})$$

$\mathcal{D}_Q^{\leq N}$ **is prefix closed (for any order)!**

Our algorithm is (almost) worst case optimal as long as we use an order for which $\mathcal{C}$ is prefix closed!

Acyclic functional dependencies

$F = (X_1 \to Y_1, ..., X_k \to Y_k)$ is a set of functional dependencies:

- $G(F)$: vertices are the variables and $x \to y$ if $x \in X_i$ and $y \in Y_i$ for some i.
- If $G(F)$ is acyclic, then let $\pi = x_1, ..., x_n$ be a topological sort of $G(F)$. Then

$$\mathcal{C}_F^N = \{\mathbb{D} \mid \mathbb{D} \text{ respects } F\} \cap \mathcal{D}_Q^{\leq N}$$

is **prefix closed for order** π (exactly the same proof as for cardinality constraints).

Hence our algorithm is worst case optimal wrt $\mathcal{C}_F^N$ (as long as we follow π).

We need to show that this functional dependencies transfer in the binarised setting but it is almost immediate.

Degree constraints

A **degree constraint** is a constraint $(X, Y, N_{Y|X})$ where $X \subseteq Y$. A relation R verifies the constraint if

$$\max\{|\prod_{Y} R[\tau]|, \tau \in \mathsf{dom}^X\} \leq N_{Y|X}$$

- Cardinality constraint = degree constraint with $X = \emptyset$.
- Functional dependency = degree constraint with $N_{Y|X} = 1$.

Acyclic degree constraints

$\Delta = \{(X_1, Y_1, N_1)..., (X_k, Y_k, N_k)\}$ set of degree constraints.

- $G(\Delta)$: vertices are the variables and $x \to y$ if $x \in X_i$ and $y \in Y_i$ for some i.
- If $G(\Delta)$ is acyclic, then let $\pi = x_1, ..., x_n$ be a topological sort of $G(\Delta)$. Then

$$\mathcal{C}_\Delta^N = \{\mathbb{D} \mid \mathbb{D} \text{ respects } \Delta\} \cap \mathcal{D}_Q^{\leq N}$$

is **prefix closed for order** π (exactly the same proof as for cardinality constraints).

Hence our algorithm is worst case optimal wrt $\mathcal{C}_\Delta^N$ (as long as we follow π).

We need to show that this functional dependencies transfer in the binarised setting but it is almost immediate.

Bonus: sampling acyclic degree constraints

We can find (λ_R) such that $\prod_{R \in Q} |R^{\mathbb{D}}|^{\lambda_R} \leq \tilde{O}(\mathsf{wc}(Q, \mathcal{C}^N_\Delta))$ for any $\mathbb{D} \in \mathcal{C}^N_\Delta$ (**polymatroid bound**).

Define $upb(t) := \prod_{R \in Q} |R^{\mathbb{D}}[\tau_t]|^{\lambda_R}$:

- upperbound of $Q(\mathbb{D})[\tau_t]$ for any $\mathbb{D} \in \mathcal{C}^N_\Delta$,
- superadditive.

We have sampling with complexity $\tilde{O}(nm \cdot \frac{\mathsf{wc}(Q, \mathcal{C}^N_\Delta)}{1 + |Q(\mathbb{D})|})$

Conclusion

- Simple algorithms and analysis
- Modular:
  - join is worst-case optimal as soon as the class is prefix closed
  - sampling is in $\frac{\mathsf{wc}(Q,\mathcal{C})}{|Q(\mathbb{D})|}$ as long as one can provide a super additive upper bound

Future work:

- Other classes such as:
  - cyclic FD,
  - general system of degree constraints (as PANDA)
- Explore dynamic ordering: can we capture more classes?