DPconv: Super-Polynomially Faster Join Ordering

Mihail Stoian, Andreas Kipf

Data Systems Lab @UTN

@SIGMOD'25, June 25, 2025

UTN Technische Universität Nürnberg

Join Ordering Recap

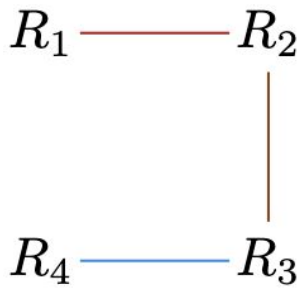
SQL Query

```
SELECT *
FROM R1, R2, R3, R4
WHERE R1.a = R2.b
AND R2.c = R3.d
AND R3.e = R4.f
```

Join Ordering Recap

SQL Query ⇨ Query Graph

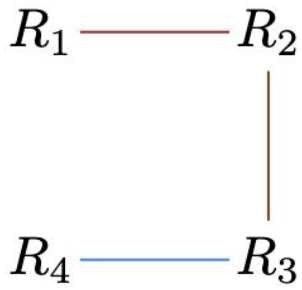
```
SELECT *
FROM R1, R2, R3, R4
WHERE R1.a = R2.b
AND R2.c = R3.d
AND R3.e = R4.f
```

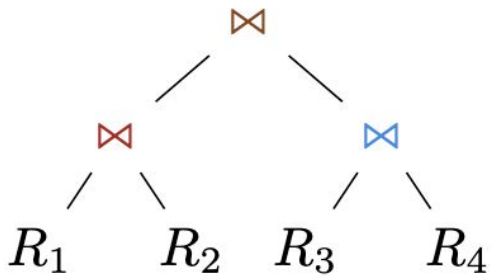


Join Ordering Recap

SQL Query ⇨ Query Graph ⇨ Query Plan

```
SELECT *
FROM R1, R2, R3, R4
WHERE R1.a = R2.b
AND R2.c = R3.d
AND R3.e = R4.f
```





A Long Quest

SIGMOD'79 ⇒ DPsize

Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one which minimizes "total access cost" for performing the entire statement.

This paper will address the issues of access path selection for queries. Retrieval for data manipulation (UPDATE, DELETE) is treated similarly. Section 2 will describe the place of the optimizer in the processing of a SQL statement, and section 3 will describe the storage component access paths that are available on a single physically stored table. In section 4 the

A Long Quest

SIGMOD'96 ⇒ DPsub

SIGMOD'79

in a Relat

IBM Res

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one which minimizes "total access cost" for performing the entire statement.

This paper will address the issues of access path selection for queries. Retrieval for data manipulation (UPDATE, DELETE) is treated similarly. Section 2 will describe the place of the optimizer in the processing of a SQL statement, and section 3 will describe the storage component access paths that are available on a single physically stored table. In section 4 the

Rapid Bushy Join-order Optimization with Cartesian Products

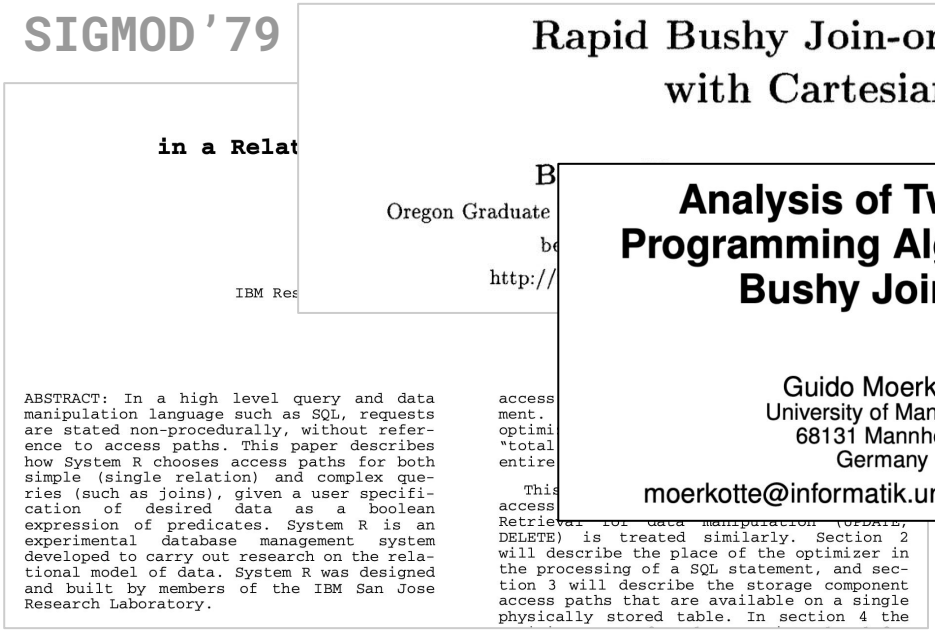
Bennet Vance
Oregon Graduate Institute of Science & Technology
bennet@cse.ogi.edu
http://www.cse.ogi.edu/DISC

David Maier
Oregon Graduate Institute of Science & Technology
maier@cse.ogi.edu
http://www.cse.ogi.edu/DISC

A Long Quest

SIGMOD'96 ⇒ DPsub

SIGMOD'79


in a Relat

IBM Res

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

Rapid Bushy Join-order Optimization
with Cartesian Products

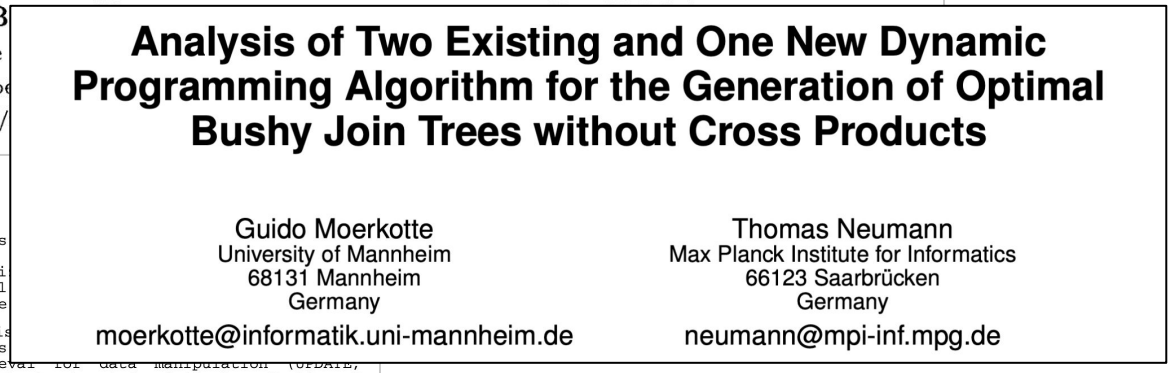
B

Oregon Graduate

b

http://

DPccp


Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products

Guido Moerkotte
University of Mannheim
68131 Mannheim
Germany
moerkotte@informatik.uni-mannheim.de

Thomas Neumann
Max Planck Institute for Informatics
66123 Saarbrücken
Germany
neumann@mpi-inf.mpg.de

VLDB'06

A Long Quest

SIGMOD'96 ⇒ DPsub

SIGMOD'79

in a Relat

IBM Res

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

access
ment.
optimi
"total
entire

This
access
Retrieval for data manipulation (UPDATE, DELETE) is treated similarly. Section 2 will describe the place of the optimizer in the processing of a SQL statement, and section 3 will describe the storage component access paths that are available on a single physically stored table. In section 4 the

Rapid Bushy Join-order Optimization with Cartesian Products

B

Oregon Graduate

be

http://

DPccp

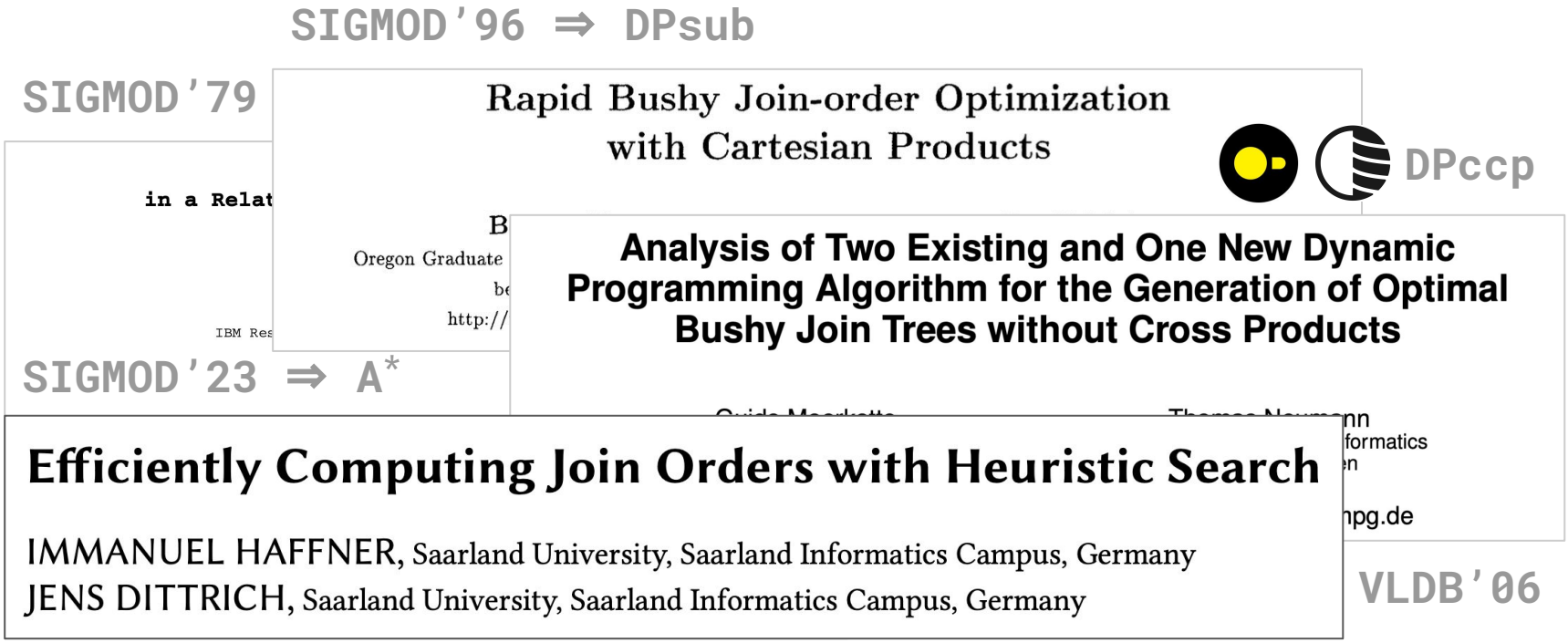
Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products

Guido Moerkotte
University of Mannheim
68131 Mannheim
Germany
moerkotte@informatik.uni-mannheim.de

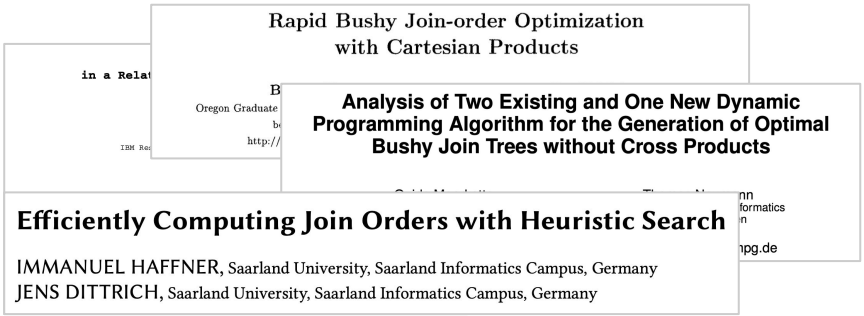
Thomas Neumann
Max Planck Institute for Informatics
66123 Saarbrücken
Germany
neumann@mpi-inf.mpg.de

VLDB'06

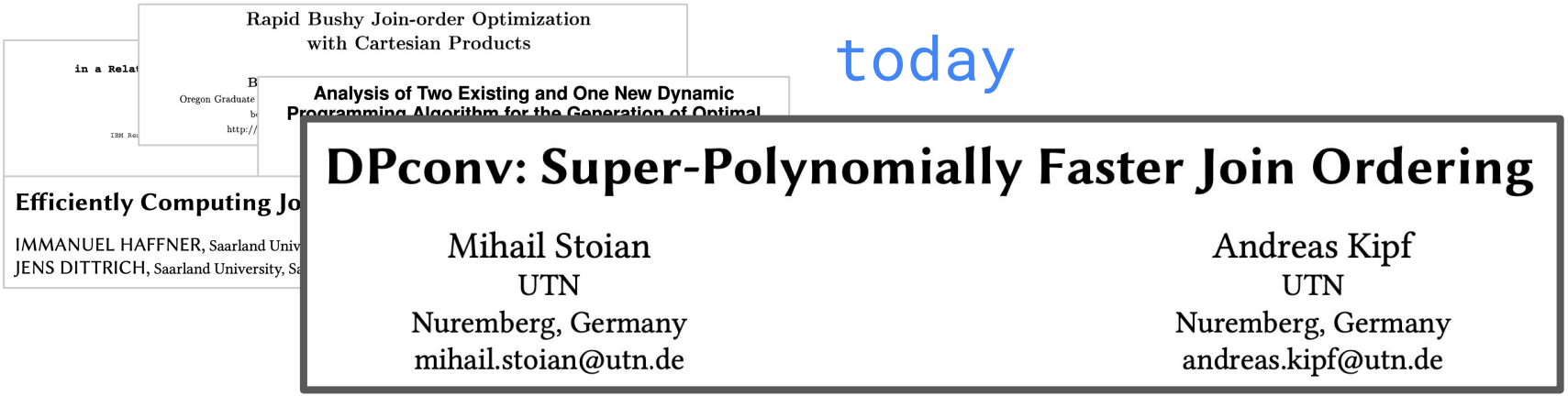
A Long Quest


SIGMOD'96 ⇒ DPsub
SIGMOD'79
Rapid Bushy Join-order Optimization with Cartesian Products
DPccp
Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products
SIGMOD'23 ⇒ A*
Efficiently Computing Join Orders with Heuristic Search
IMMANUEL HAFFNER, Saarland University, Saarland Informatics Campus, Germany
JENS DITTRICH, Saarland University, Saarland Informatics Campus, Germany
VLDB'06

A Long Quest

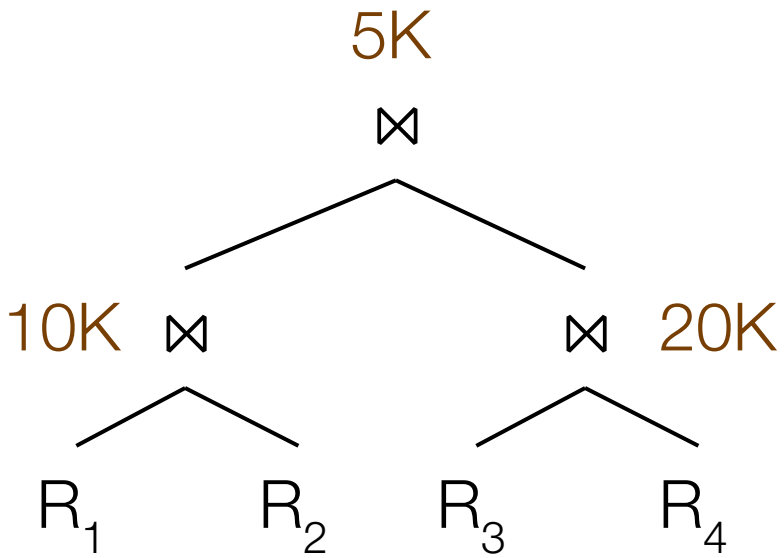

Rapid Bushy Join-order Optimization
with Cartesian Products
Oregon Graduate
Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products
Efficiently Computing Join Orders with Heuristic Search
IMMANUEL HAFFNER, Saarland University, Saarland Informatics Campus, Germany
JENS DITTRICH, Saarland University, Saarland Informatics Campus, Germany

A Long Quest

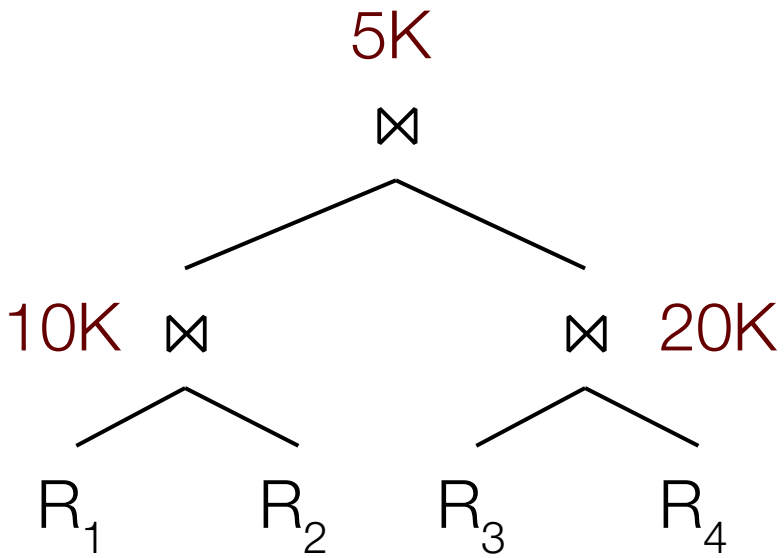

Rapid Bushy Join-order Optimization
with Cartesian Products
Analysis of Two Existing and One New Dynamic
Efficiently Computing Jo
IMMANUEL HAFFNER, Saarland Univ
JENS DITTRICH, Saarland University, Sa
today
DPconv: Super-Polynomially Faster Join Ordering
Mihail Stoian
UTN
Nuremberg, Germany
mihail.stoian@utn.de
Andreas Kipf
UTN
Nuremberg, Germany
andreas.kipf@utn.de

DPconv

Cost Functions

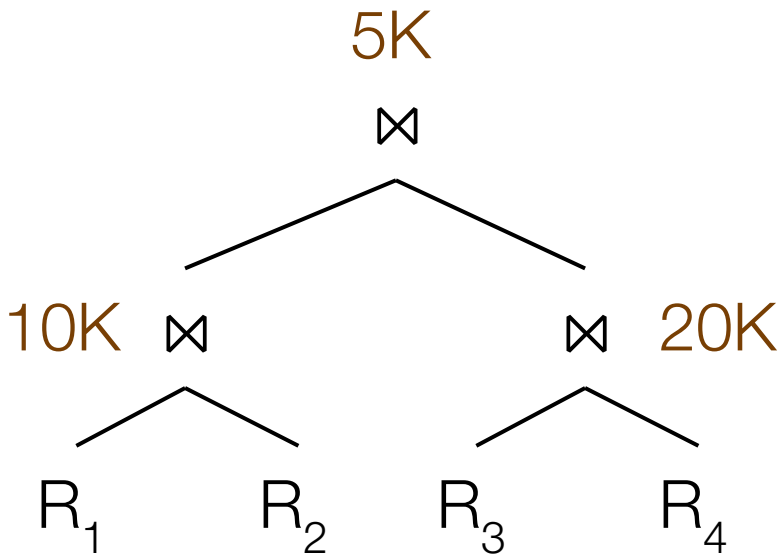


Cost Functions



C_{out}: 10K + 20K + 5K

Cost Functions



C_{out}: 10K + 20K + 5K

C_{max}: **max** {10K, 20K, 5K}

DPconv

Exact C_{max}

$O(3^n)$

DPconv

Exact C_{max}

$O(3^n)$

$O(2^n n^3)$

DPconv

Exact C_{max}

$O(3^n)$

super-polynomial

$O(2^n n^3)$

DPconv

Exact C_{max}

$O(3^n)$

$O(2^n n^3)$

Approximate C_{out}

$O(3^n)$

$O^*(2^n \log W / \varepsilon)$

DPconv

Exact C_{max}

$O(3^n)$

⇩

$O(2^n n^3)$

Approximate C_{out}

$O(3^n)$

super-polynomial ⇩

$O^*(2^n \log W / \varepsilon)$

DPconv

Exact C_{max}

$O(3^n)$

⇩

$O(2^n n^3)$

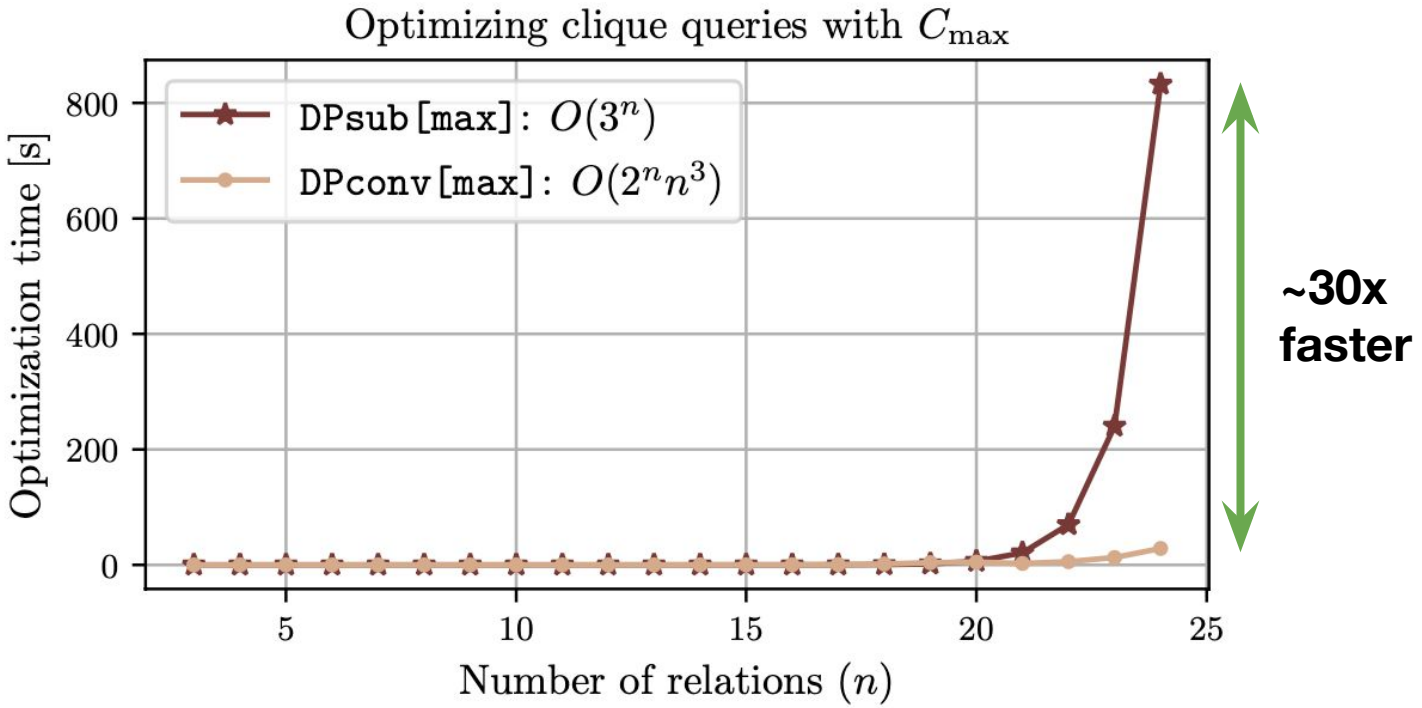
Approximate C_{out}

$O(3^n)$

super-polynomial ⇩

$O^*(2^n \log W / \varepsilon)$

Beyond Theory

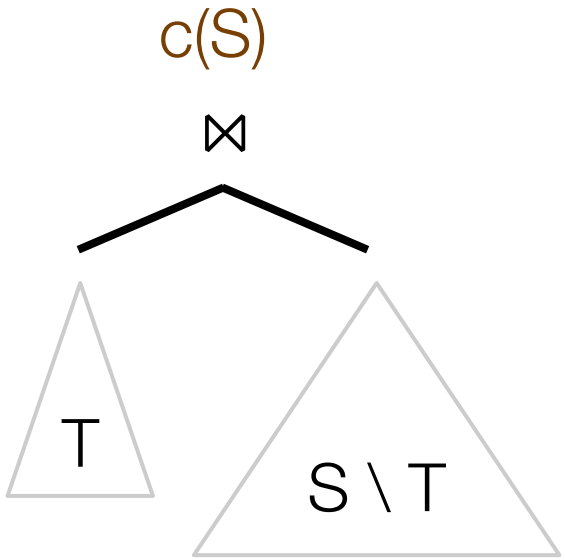


Dynamic Programming

- Bellman's optimality principle.
- Given set $S \subseteq \{1, \ldots, n\}$:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- $\Rightarrow$ Solution: DP[{1, …, n}].

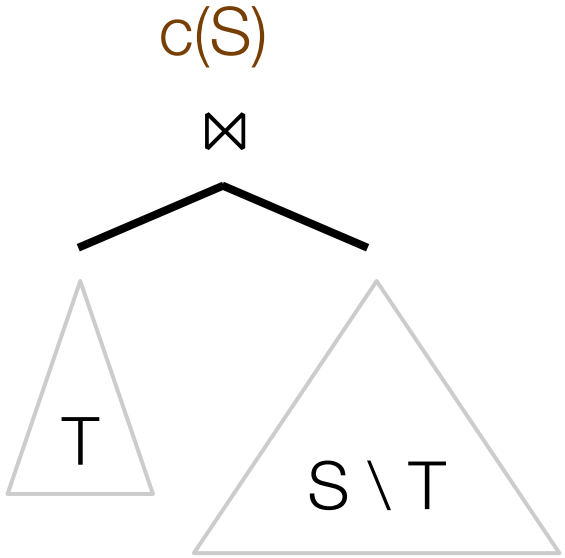
Dynamic Programming

- Bellman's optimality principle.
- Given set S ⊆ {1, …, n}:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- ⇒ Solution: DP[{1, …, n}].



Dynamic Programming

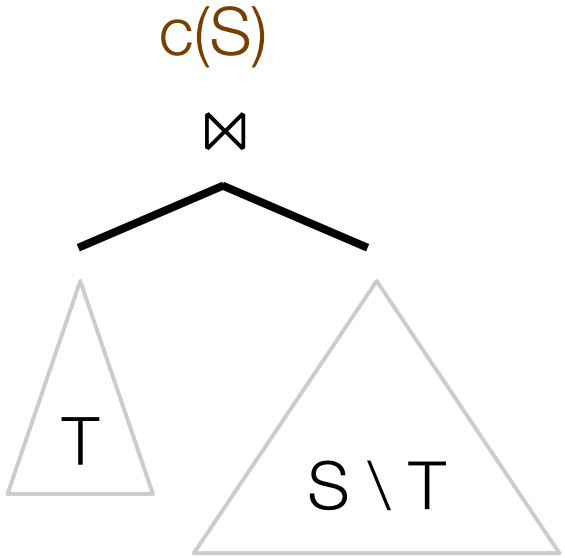
- Bellman's optimality principle.
- Given set S ⊆ {1, …, n}:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- ⇒ Solution: DP[{1, …, n}].
- Recursion for C_{out}:



$$\mathrm{DP}[S] = \qquad (\mathrm{DP}[T] + \mathrm{DP}[S \setminus T])$$

Dynamic Programming

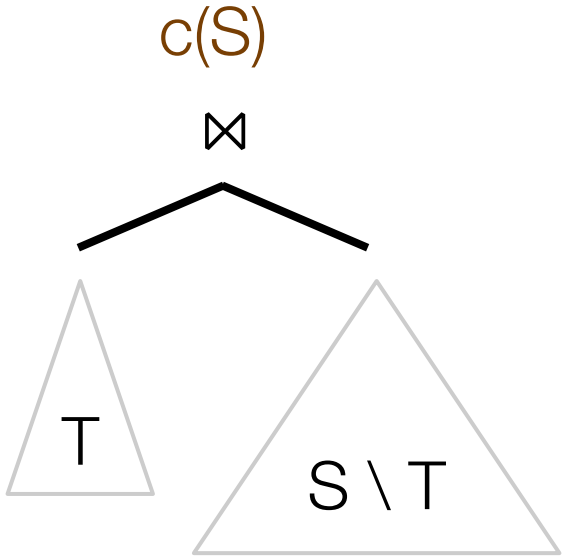
- Bellman's optimality principle.
- Given set S ⊆ {1, …, n}:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- ⇒ Solution: DP[{1, …, n}].
- Recursion for C_{out}:



$$\mathrm{DP}[S] = \min_{T \subseteq S} \left(\mathrm{DP}[T] + \mathrm{DP}[S \setminus T]\right)$$

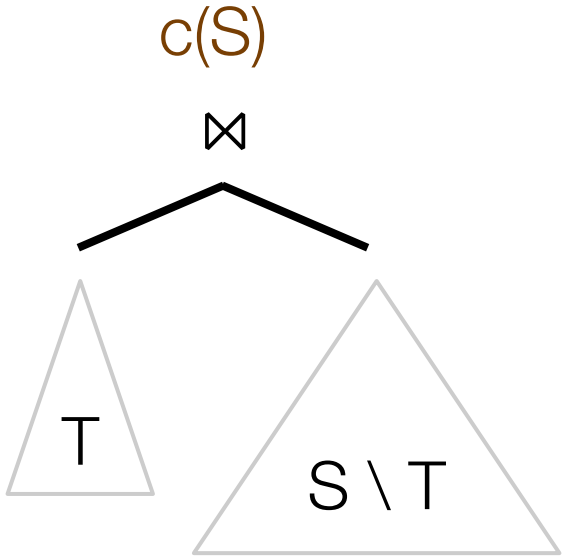
Dynamic Programming

- Bellman's optimality principle.
- Given set S ⊆ {1, …, n}:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- ⇒ Solution: DP[{1, …, n}].
- Recursion for C_{out}:



$$\mathrm{DP}[S] = c(S) + \min_{T \subseteq S} \left(\mathrm{DP}[T] + \mathrm{DP}[S \setminus T]\right)$$

Dynamic Programming



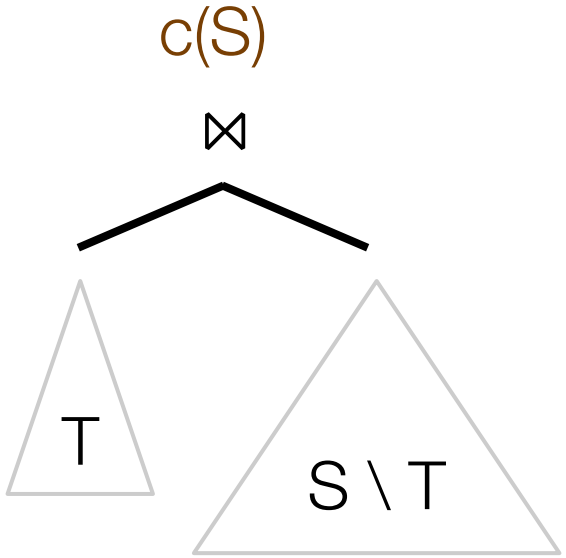
- Bellman's optimality principle.
- Given set S ⊆ {1, …, n}:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- ⇒ Solution: DP[{1, …, n}].
- Recursion for C_{out}:

$$\mathrm{DP}[S] = c(S) + \min_{T \subseteq S} \left(\mathrm{DP}[T] + \mathrm{DP}[S \setminus T]\right)$$

- Running time:

$$\sum_{S \subseteq [n]} 2^{|S|}$$

Dynamic Programming



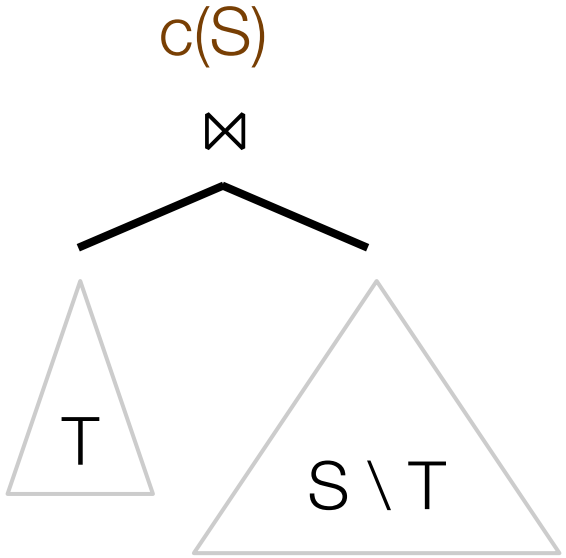
- Bellman's optimality principle.
- Given set $S \subseteq \{1, \ldots, n\}$:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- ⇒ Solution: DP[{1, …, n}].
- Recursion for C_{out}:

$$\mathrm{DP}[S] = c(S) + \min_{T \subseteq S} \left(\mathrm{DP}[T] + \mathrm{DP}[S \setminus T]\right)$$

- Running time:

$$\sum_{S \subseteq [n]} 2^{|S|} = \sum_{k=0}^{n} \binom{n}{k} 2^k$$

Dynamic Programming



- Bellman's optimality principle.
- Given set S ⊆ {1, …, n}:
  - c(S) = Join cardinality of S.
  - DP[S] = Optimal cost to join relations in S.
- ⇒ Solution: DP[{1, …, n}].
- Recursion for C_{out}:

$$\mathrm{DP}[S] = c(S) + \min_{T \subseteq S} \left(\mathrm{DP}[T] + \mathrm{DP}[S \setminus T]\right)$$

- Running time:

$$\sum_{S \subseteq [n]} 2^{|S|} = \sum_{k=0}^{n} \binom{n}{k} 2^k = (1+2)^n = 3^n$$

STOC'06

FOURIER MEETS MÖBIUS: FAST SUBSET CONVOLUTION

ANDREAS BJÖRKLUND, THORE HUSFELDT, PETTERI KASKI, AND MIKKO KOIVISTO

ABSTRACT. We present a fast algorithm for the *subset convolution problem*: given functions f and g defined on the lattice of subsets of an n-element set N, compute their *subset convolution* $f * g$, defined for all $S \subseteq N$ by

$$(f * g)(S) = \sum_{T \subseteq S} f(T) g(S \setminus T),$$

STOC'06

FOURIER MEETS MÖBIUS: FAST SUBSET CONVOLUTION

ANDREAS BJÖRKLUND, THORE HUSFELDT, PETTERI KASKI, AND MIKKO KOIVISTO

ABSTRACT. We present a fast algorithm for the *subset convolution problem*: given functions f and g defined on the lattice of subsets of an n-element set N, compute their *subset convolution* $f * g$, defined for all $S \subseteq N$ by

$$(f * g)(S) = \sum_{T \subseteq S} f(T)g(S \setminus T),$$

STOC'06

FOURIER MEETS MÖBIUS: FAST SUBSET CONVOLUTION

ANDREAS BJÖRKLUND, THORE HUSFELDT, PETTERI KASKI, AND MIKKO KOIVISTO

ABSTRACT. We present a fast algorithm for the *subset convolution problem*: given functions f and g defined on the lattice of subsets of an n-element set N, compute their *subset convolution* $f * g$, defined for all $S \subseteq N$ by

$$(f * g)(S) = \sum_{T \subseteq S} f(T)g(S \setminus T),$$

(+, ×)

STOC'06

FOURIER MEETS MÖBIUS: FAST SUBSET CONVOLUTION

ANDREAS BJÖRKLUND, THORE HUSFELDT, PETTERI KASKI, AND MIKKO KOIVISTO

ABSTRACT. We present a fast algorithm for the *subset convolution problem*: given functions f and g defined on the lattice of subsets of an n-element set N, compute their *subset convolution* $f * g$, defined for all $S \subseteq N$ by

$$(f * g)(S) = \sum_{T \subseteq S} f(T)g(S \setminus T),$$

(+, ×)

STOC'06

FOURIER MEETS MÖBIUS: FAST SUBSET CONVOLUTION

ANDREAS BJÖRKLUND, THORE HUSFELDT, PETTERI KASKI, AND MIKKO KOIVISTO

ABSTRACT. We present a fast algorithm for the *subset convolution problem*: given functions f and g defined on the lattice of subsets of an n-element set N, compute their *subset convolution* $f * g$, defined for all $S \subseteq N$ by

$$(f * g)(S) = \sum_{T \subseteq S} f(T)g(S \setminus T),$$

(+, ×)

STOC'06

FOURIER MEETS MÖBIUS: FAST SUBSET CONVOLUTION

ANDREAS BJÖRKLUND, THORE HUSFELDT, PETTERI KASKI, AND MIKKO KOIVISTO

ABSTRACT. We present a fast algorithm for the *subset convolution problem*: given functions f and g defined on the lattice of subsets of an n-element set N, compute their *subset convolution* $f * g$, defined for all $S \subseteq N$ by

$$(f * g)(S) = \sum_{T \subseteq S} f(T)g(S \setminus T),$$

where addition and multiplication is carried out in an arbitrary ring. Via Möbius transform and inversion, our algorithm evaluates the subset convolution in $O(n^2 2^n)$ additions and multiplications, substantially improving upon the straightforward $O(3^n)$ algorithm. Specifically, if the input func-

○ Ring vs. ⊂ Semi-Ring: Intuition

- Ring example: (**+**, ×).
  - 2 **+** 3 = 5.
  - 5 **+** (-2) = 3.

○ Ring vs. ☾ Semi-Ring: Intuition

- Ring example: (**+**, ×).
  - 2 **+** 3 = 5.
  - 5 **+** (-2) = 3.
- Semi-ring example: (**min**, +), (min, max).
  - **min**(2, 3) = 3.
  - *How to reverse?*

How to Map?

- Find a mapping:

$$(+, \times) \Rightarrow (\min, +)$$

How to Map?

- Find a mapping:

$$(+, \times) \Rightarrow (\min, +)$$

- 💡 Represent a value **v** as $x^{\mathbf{v}}$

How to Map?

- Find a mapping:

$$(+, \times) \Rightarrow (\min, +)$$

- 💡 Represent a value **v** as $x^{\mathbf{v}}$ → e.g., **5** is represented as $x^{\mathbf{5}}$.

How to Map?

- Motivation: Find a mapping:

$$(+, \times) \Rightarrow (\min, +)$$

- 💡 Represent a value **v** as $x^{\mathbf{v}}$ → e.g., **5** is represented as $x^{\mathbf{5}}$.

"+" becomes product: $x^{\mathbf{a}+\mathbf{b}} = x^{\mathbf{a}}x^{\mathbf{b}}$

"min" = extracting the lowest monomial

How to Map?

- Motivation: Find a mapping:

$$(+, \times) \Rightarrow (\min, +)$$

- 💡 Represent a value **v** as $x^{\mathbf{v}}$ → e.g., **5** is represented as $x^{\mathbf{5}}$.

$$\text{“+” becomes product: } x^{\mathbf{a}+\mathbf{b}} = x^{\mathbf{a}}x^{\mathbf{b}}$$

$$\text{“min” = extracting the lowest monomial}$$

- **Drawback**: Running time becomes *pseudo-polynomial*.
  - O(W) overhead, where W is the largest value.

Takeaway

- It is possible to map a semi-ring to a ring. **Drawback**:

$$O(2^n n^2) \Longrightarrow O^*(2^n W).$$

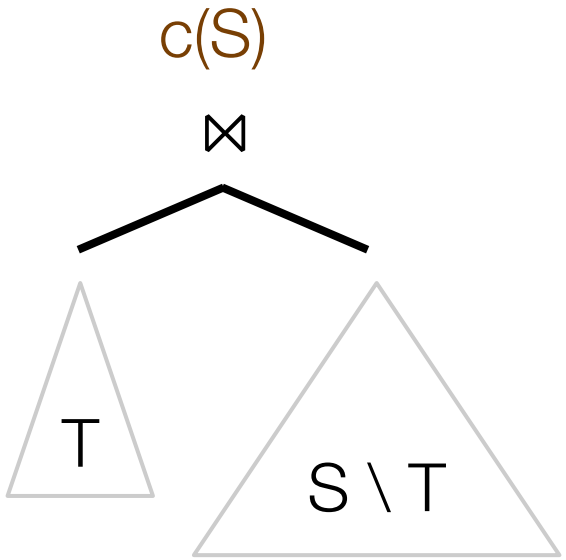
- There is a case where we can avoid the O(W) factor.

Avoiding pseudo-polynomiality

Optimizing C_{max}

- Ad-hoc dynamic programming:

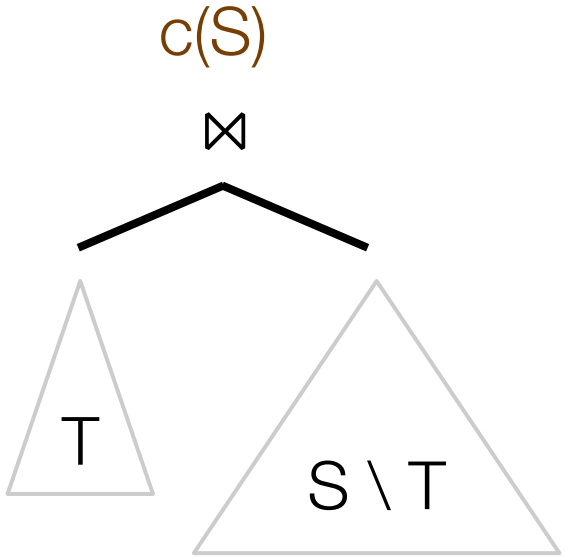
$$\mathrm{DP}[S] = \max\{\mathrm{DP}[T], \mathrm{DP}[S \setminus T]\}$$



Optimizing C_{max}

- Ad-hoc dynamic programming:

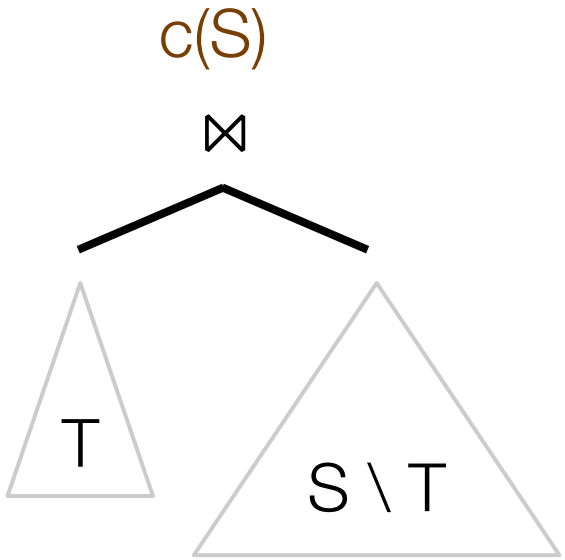
$$\mathrm{DP}[S] = \min_{T \subset S} \max\{\mathrm{DP}[T], \mathrm{DP}[S \setminus T]\}$$



Optimizing C_{max}

- Ad-hoc dynamic programming:

$$\mathrm{DP}[S] = \max\left\{c(S), \min_{T \subset S} \max\{\mathrm{DP}[T], \mathrm{DP}[S \setminus T]\}\right\}$$

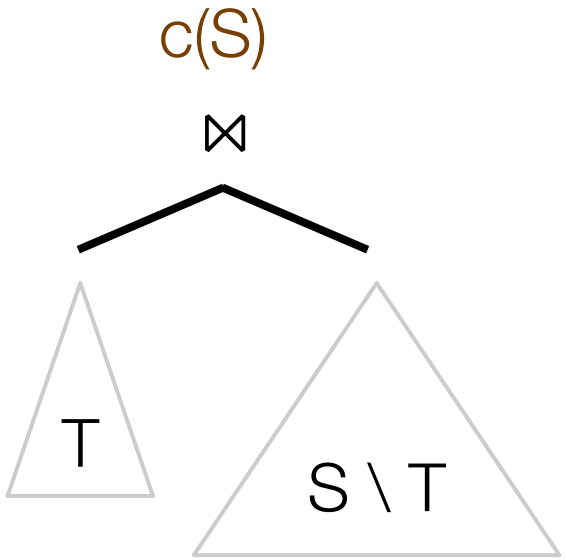


Optimizing C_{max}

- Ad-hoc dynamic programming:

$$\mathrm{DP}[S] = \max\left\{c(S), \min_{T \subset S} \max\{\mathrm{DP}[T], \mathrm{DP}[S \setminus T]\}\right\}$$

- 💡 The DP does not create new values!



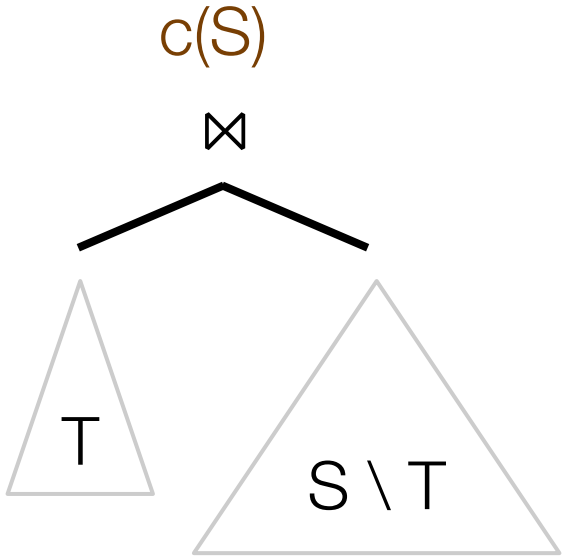
Optimizing C_{max}

- Ad-hoc dynamic programming:

$$\mathrm{DP}[S] = \max\left\{c(S), \min_{T \subset S} \max\{\mathrm{DP}[T], \mathrm{DP}[S \setminus T]\}\right\}$$

- 💡 The DP does not create new values!

  ⇒ We can binary search DP[{1, …, n}].

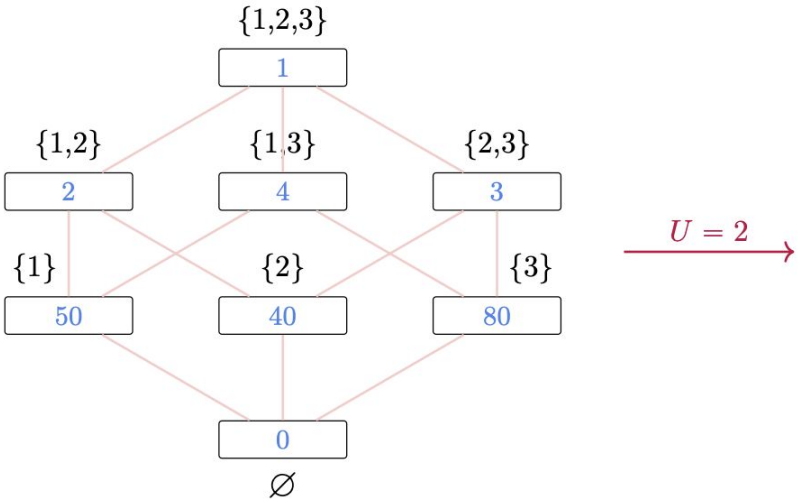


Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq \mathbf{U} \rightarrow 1$.
  - $> \mathbf{U} \rightarrow 0$.

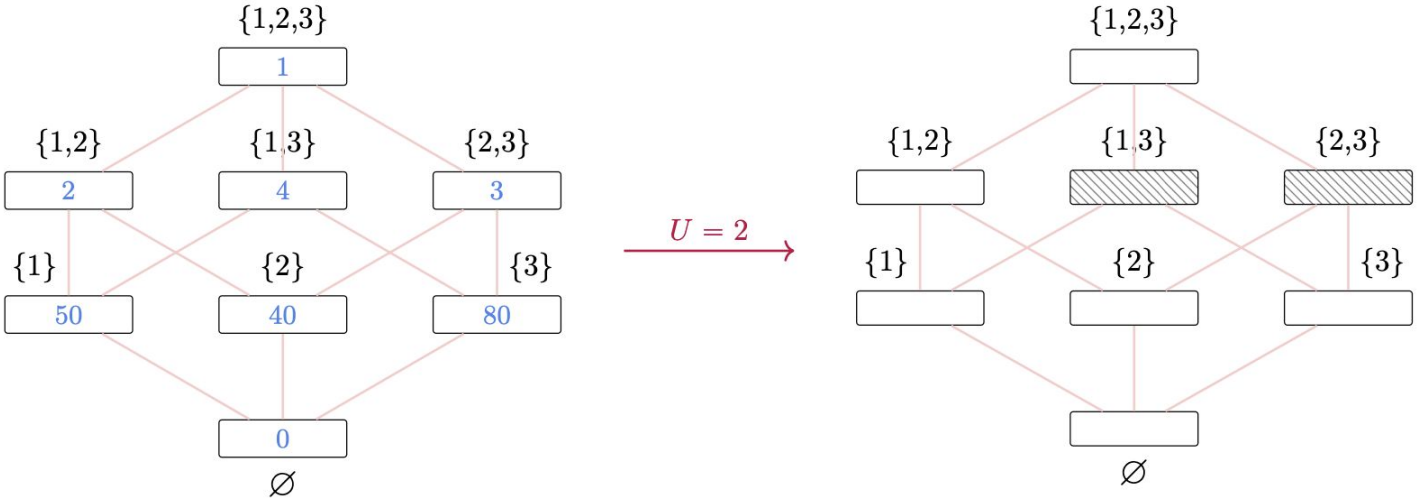
Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq$ **U** $\rightarrow$ 1.
  - $>$ **U** $\rightarrow$ 0.



Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq$ **U** $\rightarrow$ 1.
  - $>$ **U** $\rightarrow$ 0.



Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq \mathbf{U} \rightarrow 1$.
  - $> \mathbf{U} \rightarrow 0$.
- Run the DP in the (+, ×) ring:

$$\mathrm{DP}[S] = \qquad\qquad \mathrm{DP}[T] \cdot \mathrm{DP}[S \setminus T]$$

Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq \mathbf{U} \rightarrow 1$.
  - $> \mathbf{U} \rightarrow 0$.
- Run the DP in the (+, ×) ring:

$$\mathrm{DP}[S] = \sum_{T \subset S} \mathrm{DP}[T] \cdot \mathrm{DP}[S \setminus T]$$

Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq \mathbf{U} \rightarrow 1$.
  - $> \mathbf{U} \rightarrow 0$.
- Run the DP in the (+, ×) ring:

$$\mathrm{DP}[S] = c(S) + \sum_{T \subset S} \mathrm{DP}[T] \cdot \mathrm{DP}[S \setminus T]$$

Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq$ **U** $\rightarrow$ 1.
  - $>$ **U** $\rightarrow$ 0.
- Run the DP in the (+, ×) ring:

$$\mathrm{DP}[S] = c(S) + \sum_{T \subset S} \mathrm{DP}[T] \cdot \mathrm{DP}[S \setminus T]$$

- If DP[{1, …, n}] > 0 ⇒ U is feasible.

Optimizing C_{max}: Algorithm

- Fix a cardinality **U**.
- Rescale the join cardinalities:
  - $\leq$ **U** $\rightarrow$ 1.
  - $>$ **U** $\rightarrow$ 0.
- Run the DP in the (+, ×) ring:

$$\mathrm{DP}[S] = c(S) + \sum_{T \subset S} \mathrm{DP}[T] \cdot \mathrm{DP}[S \setminus T]$$

- If DP[{1, …, n}] > 0 ⇒ U is feasible.
- Running time: $O(\log(2^n) \cdot 2^n n^2) = O(2^n n^3)$.

The End?

- DPconv does not take into account the sparsity of the graph.

  ⇒ We need a sparse subset convolution.

- *Polynomial-space* join ordering?

  ⇒ Preliminary results for acyclic query graphs.