DPconv:

Super-Polynomially Faster Join Ordering

Mihail Stoian, Andreas Kipf

UTN

@Gray Systems Lab

January 28, 2025

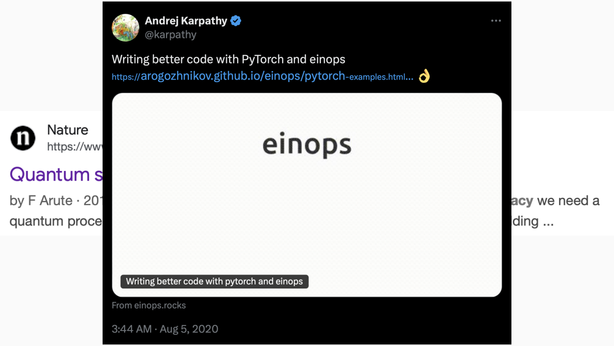
It's not just about join ordering..

Nature
https://www.nature.com › articles

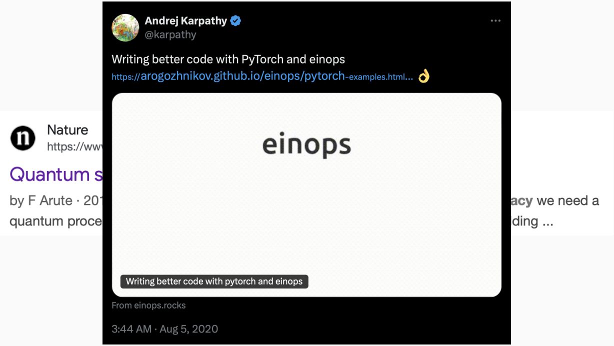
Quantum supremacy using a programmable ...

by F Arute · 2019 · Cited by 9186 — Therefore, in order to claim **quantum supremacy** we need a quantum processor that executes the program with sufficiently low error rates. Building ...

It’s not just about join ordering..



It's not just about join ordering..



First super-polynomial speedups for `einsum` optimization.

Join Ordering

Problem Statement

- Fundamental problem in query optimization.
- Statement: Given a `SQL` query, find the order in which to join the relations.
- Goal: Minimize query time or memory usage.

Join Ordering

Problem Statement

- Fundamental problem in query optimization.
- Statement: Given a `SQL` query, find the order in which to join the relations.
- Goal: Minimize query time or memory usage.
- Cost functions – **minimize**:
  - C_{out}: Sum of the intermediate sizes $\approx$ "I want fast queries".

Join Ordering

Problem Statement

- Fundamental problem in query optimization.
- Statement: Given a `SQL` query, find the order in which to join the relations.
- Goal: Minimize query time or memory usage.
- Cost functions – **minimize**:
  - C_{out}: Sum of the intermediate sizes $\approx$ "I want fast queries".
  - C_{max}: Maximum intermediate size $\approx$ "Please avoid disk spilling".

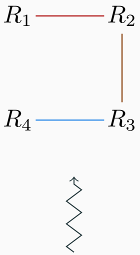
Join Ordering

Problem Statement

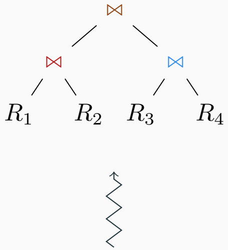
- Fundamental problem in query optimization.
- Statement: Given a `SQL` query, find the order in which to join the relations.
- Goal: Minimize query time or memory usage.
- Cost functions – **minimize**:
  - C_{out}: Sum of the intermediate sizes $\approx$ "I want fast queries".
  - C_{max}: Maximum intermediate size $\approx$ "Please avoid disk spilling".
- Research question since $\sim$50 years: *How fast can we get?*

Join Ordering: Notation

SELECT *
FROM R_1, R_2, R_3, R_4
WHERE R_1.a = R_2.b
AND R_2.c = R_3.d
AND R_3.e = R_4.f



Query graph



Join tree

Join Ordering: Dynamic Programming

Dynamic Programming

- Use Bellman's optimality principle.
- Dynamic programming table $\mathrm{DP}[S]$: The optimal cost to join the relations in the set S.

[1] For $C_{\max}$, replace both $+$ by max.

Join Ordering: Dynamic Programming

Dynamic Programming

- Use Bellman's optimality principle.
- Dynamic programming table $\mathrm{DP}[S]$: The optimal cost to join the relations in the set S.
  $\Rightarrow$ Solution value: $\mathrm{DP}[\{1, \ldots, n\}]$.
- Optimize $\mathrm{DP}[S]$ as

$$\mathrm{DP}[S] = c(S) + \min_{T \subseteq S} (\mathrm{DP}[T] + \mathrm{DP}[S \setminus T]).^{1}$$

[1] For $C_{\max}$, replace both $+$ by max.

Join Ordering: Dynamic Programming

Dynamic Programming

- Use Bellman's optimality principle.
- Dynamic programming table $\mathrm{DP}[S]$: The optimal cost to join the relations in the set S.
  $\Rightarrow$ Solution value: $\mathrm{DP}[\{1, \ldots, n\}]$.
- Optimize $\mathrm{DP}[S]$ as

$$\mathrm{DP}[S] = c(S) + \min_{T \subseteq S} \left(\mathrm{DP}[T] + \mathrm{DP}[S \setminus T]\right).^{1}$$

Running Time Analysis

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

[1] For $C_{\max}$, replace both $+$ by max.

Join Ordering: Status Quo

Exact Algorithms

- Selinger et al. (1979): `DPsizeLinear`[2] $\rightarrow \Theta(2^n n)$.

[2] Left-deep only.

Join Ordering: Status Quo

Exact Algorithms

- Selinger et al. (1979): `DPsizeLinear`[2] $\rightarrow \Theta(2^n n)$.
- Vance and Maier (1996): `DPsub` $\rightarrow \Theta(3^n)$.

[2] Left-deep only.

Join Ordering: Status Quo

Exact Algorithms

- Selinger et al. (1979): `DPsizeLinear`[2] $\rightarrow \Theta(2^n n)$.
- Vance and Maier (1996): `DPsub` $\rightarrow \Theta(3^n)$.
- Moerkotte and Neumann (2006): `DPccp` $\rightarrow \Theta(\#\texttt{ccp})$.

[2] Left-deep only.

Join Ordering: Status Quo

Exact Algorithms

- Selinger et al. (1979): `DPsizeLinear`[2] $\rightarrow \Theta(2^n n)$.
- Vance and Maier (1996): `DPsub` $\rightarrow \Theta(3^n)$.
- Moerkotte and Neumann (2006): `DPccp` $\rightarrow \Theta(\#\texttt{ccp})$.
- Haffner and Dietrich (2023): $A^* \rightarrow O(\#\texttt{ccp})$.

[2] Left-deep only.

Join Ordering: Status Quo

Exact Algorithms

- Selinger et al. (1979): $\texttt{DPsizeLinear}^2 \rightarrow \Theta(2^n n)$.
- Vance and Maier (1996): $\texttt{DPsub} \rightarrow \Theta(3^n)$.
- Moerkotte and Neumann (2006): $\texttt{DPccp} \rightarrow \Theta(\#\texttt{ccp})$.
- Haffner and Dietrich (2023): $A^* \rightarrow O(\#\texttt{ccp})$.

Approximation Algorithms

- Many good heuristics, yet no theoretical guarantees.

[2] Left-deep only.

Join Ordering: Status Quo

Exact Algorithms

- Selinger et al. (1979): $\texttt{DPsizeLinear}^2 \rightarrow \Theta(2^n n)$.
- Vance and Maier (1996): $\texttt{DPsub} \rightarrow \Theta(3^n)$.
- Moerkotte and Neumann (2006): $\texttt{DPccp} \rightarrow \Theta(\#\texttt{ccp})$.
- Haffner and Dietrich (2023): $A^* \rightarrow O(\#\texttt{ccp})$.

Approximation Algorithms

- Many good heuristics, yet no theoretical guarantees.
- For a good reason: it's *hard* [1]:
  - NP-hard to approximate the optimal cost K within $2^{\Theta(\log^{\delta-1} K)}$ for any $\delta > 0$.

[2] Left-deep only.

Our Results

`TL;DR`

Join Ordering DP = Subset Convolution

Our Results: Formally

Let W be the largest join cardinality. Then,

Our Results: Formally

Let W be the largest join cardinality. Then,

- C_{out}: Can be solved in $O^*(2^n W)$ time.[3]

[3] O^* hides polynomial factors in n.

Our Results: Formally

Let W be the largest join cardinality. Then,

- C_{out}: Can be solved in $O^*(2^n W)$ time.[3]
- C_{out}: Can be $(1+\varepsilon)$-approximated in $O^*(2^n \log W/\varepsilon)$ time.

[3] O^* hides polynomial factors in n.

Our Results: Formally

Let W be the largest join cardinality. Then,

- C_{out}: Can be solved in $O^*(2^n W)$ time.[3]
- C_{out}: Can be $(1+\varepsilon)$-approximated in $O^*(2^n \log W/\varepsilon)$ time.
- C_{max}: Can be solved in $O(2^n n^3)$ time (does not depend on W).

[3] O^* hides polynomial factors in n.

Our Results: Formally

Let W be the largest join cardinality. Then,

- C_{out}: Can be solved in $O^*(2^n W)$ time.[3]
- C_{out}: Can be $(1+\varepsilon)$-approximated in $O^*(2^n \log W/\varepsilon)$ time.
- $C_{\max}$: Can be solved in $O(2^n n^3)$ time (does not depend on W).

Implications

- $C_{\max}$: Super-polynomial speedup.

[3] O^* hides polynomial factors in n.

Our Results: Formally

Let W be the largest join cardinality. Then,

- C_{out}: Can be solved in $O^*(2^n W)$ time.[3]
- C_{out}: Can be $(1+\varepsilon)$-approximated in $O^*(2^n \log W/\varepsilon)$ time.
- C_{max}: Can be solved in $O(2^n n^3)$ time (does not depend on W).

Implications

- C_{max}: Super-polynomial speedup.
- C_{out}: First $(1+\varepsilon)$-approximation algorithm.

[3] O^* hides polynomial factors in n.

Our Results: Formally

Let W be the largest join cardinality. Then,

- C_{out}: Can be solved in $O^*(2^n W)$ time.[3]
- C_{out}: Can be $(1+\varepsilon)$-approximated in $O^*(2^n \log W/\varepsilon)$ time.
- C_{max}: Can be solved in $O(2^n n^3)$ time (does not depend on W).

Implications

- C_{max}: Super-polynomial speedup.
- C_{out}: First $(1+\varepsilon)$-approximation algorithm.
- Beyond databases: `einsum` optimization.

[3] O^* hides polynomial factors in n.

Our Results: Formally

Let W be the largest join cardinality. Then,

- C_{out}: Can be solved in $O^*(2^n W)$ time.[3]
- C_{out}: Can be $(1+\varepsilon)$-approximated in $O^*(2^n \log W/\varepsilon)$ time.
- C_{max}: Can be solved in $O(2^n n^3)$ time (does not depend on W).

Implications

- C_{max}: Super-polynomial speedup.
- C_{out}: First $(1+\varepsilon)$-approximation algorithm.
- Beyond databases: `einsum` optimization.
  - Used in quantum circuit simulation.
  - Speedup over the state-of-the-art algorithm in `opt_einsum`.

[3] O^* hides polynomial factors in n.

Overview: Part I

General Framework

- Join ordering DP:

$$\mathrm{DP}[S] = c(S) \otimes \min_{T \subseteq S} (\mathrm{DP}[T] \otimes \mathrm{DP}[S \setminus T]).$$

Overview: Part I

General Framework

- Join ordering DP:

$$\mathrm{DP}[S] = c(S) \otimes \min_{T \subseteq S} (\mathrm{DP}[T] \otimes \mathrm{DP}[S \setminus T]).$$

- Subset convolution in the $(\min, \otimes)$ semi-ring:

$$(f \star g)(S) = \min_{T \subseteq S} (f(T) \otimes g(S \setminus T)).$$

Overview: Part I

General Framework

- Join ordering DP:

$$\mathrm{DP}[S] = c(S) \otimes \min_{T \subseteq S} (\mathrm{DP}[T] \otimes \mathrm{DP}[S \setminus T]).$$

- Subset convolution in the $(\min, \otimes)$ semi-ring:

$$(f \star g)(S) = \min_{T \subseteq S} (f(T) \otimes g(S \setminus T)).$$

- In other words:

$$\mathrm{DP}[S] = c(S) \otimes (\mathrm{DP} \star \mathrm{DP})(S).$$

Overview: Part I

General Framework

- Join ordering DP:

$$\mathrm{DP}[S] = c(S) \otimes \min_{T \subseteq S} (\mathrm{DP}[T] \otimes \mathrm{DP}[S \setminus T]).$$

- Subset convolution in the $(\min, \otimes)$ semi-ring:

$$(f \star g)(S) = \min_{T \subseteq S} (f(T) \otimes g(S \setminus T)).$$

- In other words:

$$\mathrm{DP}[S] = c(S) \otimes (\mathrm{DP} \star \mathrm{DP})(S).$$

- Instantiations:

Overview: Part I

General Framework

- Join ordering DP:

$$\mathrm{DP}[S] = c(S) \otimes \min_{T \subseteq S} (\mathrm{DP}[T] \otimes \mathrm{DP}[S \setminus T]).$$

- Subset convolution in the $(\min, \otimes)$ semi-ring:

$$(f \star g)(S) = \min_{T \subseteq S} (f(T) \otimes g(S \setminus T)).$$

- In other words:

$$\mathrm{DP}[S] = c(S) \otimes (\mathrm{DP} \star \mathrm{DP})(S).$$

- Instantiations:
  - C_{out}: $(\min, +)$.

Overview: Part I

General Framework

- Join ordering DP:

$$\mathrm{DP}[S] = c(S) \otimes \min_{T \subseteq S} (\mathrm{DP}[T] \otimes \mathrm{DP}[S \setminus T]).$$

- Subset convolution in the $(\min, \otimes)$ semi-ring:

$$(f \star g)(S) = \min_{T \subseteq S} (f(T) \otimes g(S \setminus T)).$$

- In other words:

$$\mathrm{DP}[S] = c(S) \otimes (\mathrm{DP} \star \mathrm{DP})(S).$$

- Instantiations:
  - C_{out}: $(\min, +)$.
  - $C_{\max}$: $(\min, \max)$.

Overview: Part II

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

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

Overview: Part II

Fast Subset Convolution

- Evaluating convolutions in semi-rings is hard – $O(3^n)$.

Overview: Part II

Fast Subset Convolution

- Evaluating convolutions in semi-rings is hard – $O(3^n)$.
- Convolution in the $(+, \times)$ ring is much faster – $O(2^n n^2)$:

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

Overview: Part II

Fast Subset Convolution

- Evaluating convolutions in semi-rings is hard – $O(3^n)$.
- Convolution in the $(+, \times)$ ring is much faster – $O(2^n n^2)$:

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

- Good news: We can map a semi-ring into a ring via *polynomials*.
  - Represent a value v as x^v.
  - **Intuition**:
    - "$+$" becomes product: $x^{a+b} = x^a x^b$.
    - "min" is just taking the lowest monomial.

Overview: Part II

Fast Subset Convolution

- Evaluating convolutions in semi-rings is hard – $O(3^n)$.
- Convolution in the $(+, \times)$ ring is much faster – $O(2^n n^2)$:

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

- Good news: We can map a semi-ring into a ring via *polynomials*.
  - Represent a value v as x^v.
  - **Intuition**:
    - "$+$" becomes product: $x^{a+b} = x^a x^b$.
    - "min" is just taking the lowest monomial.
- Bad news: Running time becomes pseudopolynomial: $O^*(2^n W)$, where W is the largest value in f and g.

Overview: Part II

Fast Subset Convolution

- Evaluating convolutions in semi-rings is hard – $O(3^n)$.
- Convolution in the $(+, \times)$ ring is much faster – $O(2^n n^2)$:

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

- Good news: We can map a semi-ring into a ring via *polynomials*.
  - Represent a value v as x^v.
  - **Intuition**:
    - "$+$" becomes product: $x^{a+b} = x^a x^b$.
    - "min" is just taking the lowest monomial.
- Bad news: Running time becomes pseudopolynomial: $O^*(2^n W)$, where W is the largest value in f and g.

Where to go?

Overview: Part III

`DPconv`: **Takeaway**

- *No* pseudopolynomial factor when we optimize for $C_{\max}$.

[4] Or in $O^*(2^{\frac{3n}{2}}/\sqrt{\varepsilon})$-time, however this is not yet practical.

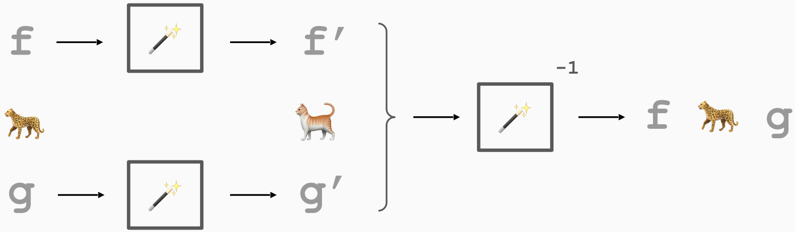
Overview: Part III

DPconv: Takeaway

- *No* pseudopolynomial factor when we optimize for $C_{\max}$.
- *No* pseudopolynomial factor when we move to $(1+\varepsilon)$-approximation $\rightarrow$ C_{out} can be $(1+\varepsilon)$-approximated in $O^*(2^n \log W/\varepsilon)$.[4]

[4] Or in $O^*(2^{\frac{3n}{2}}/\sqrt{\varepsilon})$-time, however this is not yet practical.

Fast Convolutions: Intuition



Fast Subset Convolution

- Same principle: Transform the functions by a magic box.
- Our magic box – **zeta transform**:

$$\zeta f(S) = \sum_{T \subseteq S} f(T),$$

for a subset $S \subseteq [n]$.

Fast Subset Convolution

- Same principle: Transform the functions by a magic box.
- Our magic box – **zeta transform**:

$$\zeta f(S) = \sum_{T \subseteq S} f(T),$$

  for a subset $S \subseteq [n]$.
- Naive evaluation: $O(3^n)$.

Fast Subset Convolution

- Same principle: Transform the functions by a magic box.
- Our magic box – **zeta transform**:

$$\zeta f(S) = \sum_{T \subseteq S} f(T),$$

  for a subset $S \subseteq [n]$.
- Naive evaluation: $O(3^n)$.
- Clever evaluation: $O(2^n n)$.

Zeta Transform: Hands-On

- You have 1 minute to fill up:

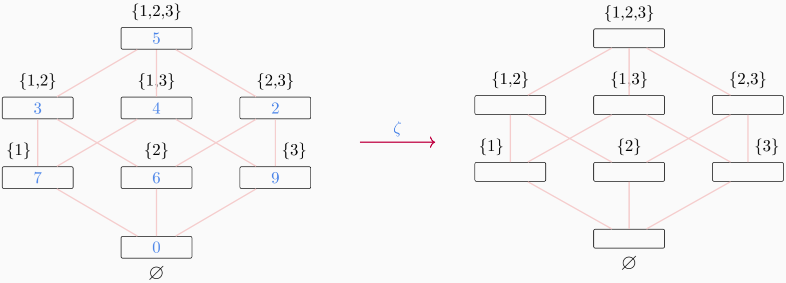


Figure 1: How to compute the zeta transform of a set function.

Zeta Transform: Hands-On

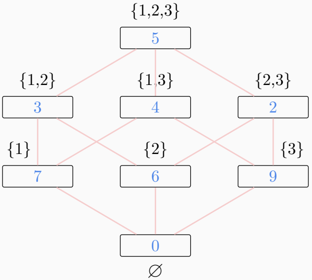
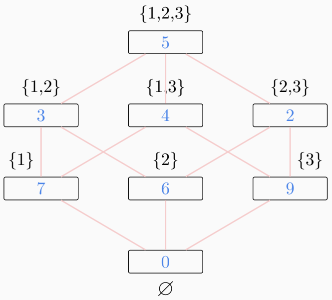


Figure 2: How to compute the zeta transform of a set function.

Fast Zeta Transform: Pseudocode

```
zeta(f):
  for d in range(n):
    for S in range(2**n):
      if S & 2**d:
        f[S] += f[S ^ 2**d]
```



Inverse Zeta Transform

- "Wait, you also need the inverted magic box, right?"

Inverse Zeta Transform

- "Wait, you also need the inverted magic box, right?"
- The inverse of the zeta transform:

$$\mu f(S) = \sum_{T \subseteq S} (-1)^{|S \setminus T|} f(T).$$

Inverse Zeta Transform

- "Wait, you also need the inverted magic box, right?"
- The inverse of the zeta transform:

$$\mu f(S) = \sum_{T \subseteq S} (-1)^{|S \setminus T|} f(T).$$

- In other words: $f = \mu\zeta f = \zeta\mu f$.

Inverse Zeta Transform: Example

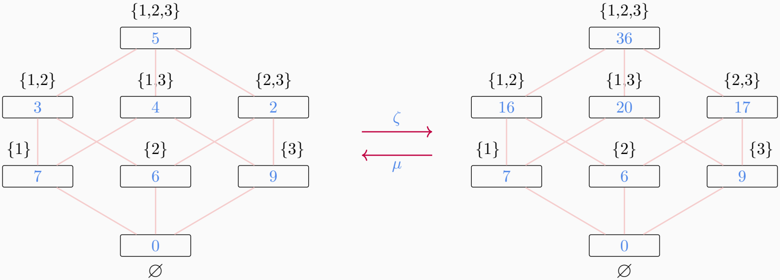


Figure 3: Inverting the zeta transform.

Fast Subset Convolution

First attempt

1. Magic box: ζf, ζg.

Fast Subset Convolution

First attempt

1. Magic box: ζf, ζg.
2. Compute $\zeta f \cdot \zeta g$.

Fast Subset Convolution

First attempt

1. Magic box: ζf, ζg.
2. Compute $\zeta f \cdot \zeta g$.
3. Inverted magic box: $\mu(\zeta f \cdot \zeta g)$.

First attempt

1. Magic box: ζf, ζg.
2. Compute $\zeta f \cdot \zeta g$.
3. Inverted magic box: $\mu(\zeta f \cdot \zeta g)$.

Unfortunately, this only computes

$$\sum_{\substack{U,V \subseteq S \\ U \cup V = S}} f(U)g(V).$$

Fast Subset Convolution

First attempt

1. Magic box: ζf, ζg.
2. Compute $\zeta f \cdot \zeta g$.
3. Inverted magic box: $\mu(\zeta f \cdot \zeta g)$.

Unfortunately, this only computes

$$\sum_{\substack{U,V \subseteq S \\ U \cup V = S}} f(U)g(V).$$

How can we fix it?

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:

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

- Note: The DP does not create any new values!
  $\rightarrow$ We can binary search the optimal value $\mathrm{DP}[\{1, \ldots, n\}]$.

Optimizing $C_{\max}$

- Ad-hoc dynamic programming:

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

- Note: The DP does not create any new values!
  $\rightarrow$ We can binary search the optimal value $\mathrm{DP}[\{1, \ldots, n\}]$.
- Fix a join cardinality γ and define a new cardinality function:

$$c'(S) = \texttt{if } c(S) \leq \gamma \texttt{ then } 1 \texttt{ else } 0.$$

Optimizing C_{max}

- Ad-hoc dynamic programming:

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

- Note: The DP does not create any new values!
  $\rightarrow$ We can binary search the optimal value $\mathrm{DP}[\{1, \ldots, n\}]$.
- Fix a join cardinality γ and define a new cardinality function:

$$c'(S) = \texttt{if } c(S) \leq \gamma \texttt{ then } 1 \texttt{ else } 0.$$

- Now just run the $(+, \times)$ dynamic programming with c':

$$\mathrm{DP}[S] = c'(S) + \sum_{T \subset S} \mathrm{DP}[T] \cdot \mathrm{DP}[S \setminus T].$$

Optimizing C_{max}

- Ad-hoc dynamic programming:

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

- Note: The DP does not create any new values!
  $\rightarrow$ We can binary search the optimal value $\mathrm{DP}[\{1, \ldots, n\}]$.
- Fix a join cardinality γ and define a new cardinality function:

$$c'(S) = \texttt{if } c(S) \leq \gamma \texttt{ then } 1 \texttt{ else } 0.$$

- Now just run the $(+, \times)$ dynamic programming with c':

$$\mathrm{DP}[S] = c'(S) + \sum_{T \subset S} \mathrm{DP}[T] \cdot \mathrm{DP}[S \setminus T].$$

- If $\mathrm{DP}[\{1, \ldots, n\}] > 0$, then γ is feasible.

Optimizing $C_{\max}$

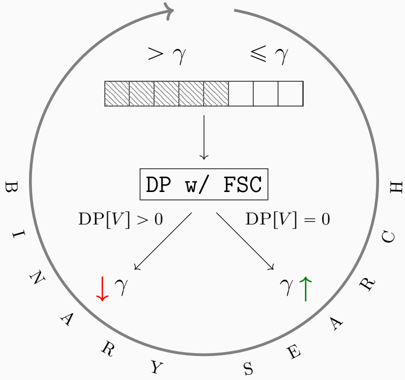


Figure 4: How `DPconv` optimizes $C_{\max}$.

Optimizing C_{max}: Example

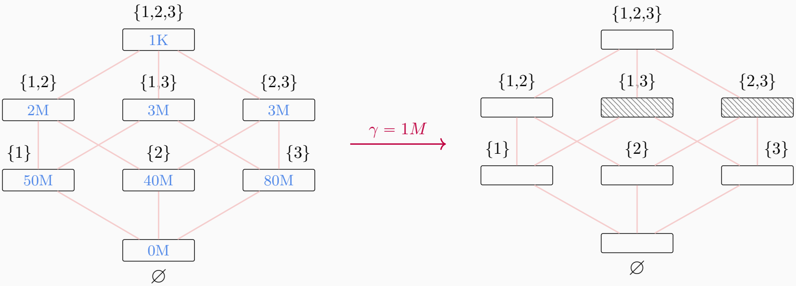


Figure 5: The cardinalities $> \gamma$ are ignored.

Optimizing C_{out}

- Ad-hoc DP: $O(3^n)$.
- C_{out} is working in the $(\min, +)$ semi-ring.
  $\rightarrow$ Using the FSC framework, we obtain a $O^*(2^n W)$-time algorithm.

Optimizing C_{out}

- Ad-hoc DP: $O(3^n)$.
- C_{out} is working in the $(\min, +)$ semi-ring.
  $\rightarrow$ Using the FSC framework, we obtain a $O^*(2^n W)$-time algorithm.
- However, this is not so practical in the context of join ordering.
  $\rightarrow$ Join cardinalities can be *very* large.

Optimizing C_{out}

- Ad-hoc DP: $O(3^n)$.
- C_{out} is working in the $(\min, +)$ semi-ring.
  $\rightarrow$ Using the FSC framework, we obtain a $O^*(2^n W)$-time algorithm.
- However, this is not so practical in the context of join ordering.
  $\rightarrow$ Join cardinalities can be *very* large.
- Two open questions in algorithm design research:

Optimizing C_{out}

- Ad-hoc DP: $O(3^n)$.
- C_{out} is working in the $(\min, +)$ semi-ring.
  $\rightarrow$ Using the FSC framework, we obtain a $O^*(2^n W)$-time algorithm.
- However, this is not so practical in the context of join ordering.
  $\rightarrow$ Join cardinalities can be *very* large.
- Two open questions in algorithm design research:
  - Is $O(3^n)$ the best we can do for $(\min, +)$ subset convolution?

Optimizing C_{out}

- Ad-hoc DP: $O(3^n)$.
- C_{out} is working in the $(\min, +)$ semi-ring.
  $\rightarrow$ Using the FSC framework, we obtain a $O^*(2^n W)$-time algorithm.
- However, this is not so practical in the context of join ordering.
  $\rightarrow$ Join cardinalities can be *very* large.
- Two open questions in algorithm design research:
  - Is $O(3^n)$ the best we can do for $(\min, +)$ subset convolution?
  - Is the pseudopolynomial $O(W)$ factor actually needed?

Joint Optimization: $C_{\mathrm{cap}} = C_{\mathsf{max}} + C_{\mathrm{out}}$

Motivation C_{cap}

- Optimizing for C_{max} alone may lead to slow plans.
  $\rightarrow$ Also enforce that we have an optimal C_{out} plan.

Joint Optimization: $C_{\text{cap}} = C_{\mathsf{max}} + C_{\text{out}}$

Motivation C_{cap}

- Optimizing for C_{max} alone may lead to slow plans.
  $\rightarrow$ Also enforce that we have an optimal C_{out} plan.
- First run `DPccp` with C_{max}, then C_{out} with a threshold.

Motivation C_{cap}

- Optimizing for $C_{\max}$ alone may lead to slow plans.
  $\rightarrow$ Also enforce that we have an optimal C_{out} plan.
- First run `DPccp` with $C_{\max}$, then C_{out} with a threshold.
- However, this is slower:

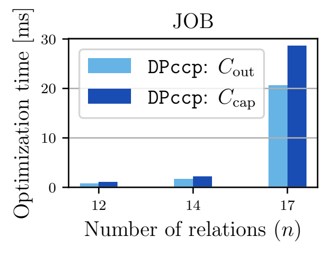


Figure 6: Initial overhead in optimizing C_{cap} on JOB

Joint Optimization: $C_{\mathrm{cap}} = C_{\mathsf{max}} + C_{\mathrm{out}}$

(Very) Simple Solution

- Replace `DPccp` in C_{max} optimization by `DPconv`.

Approximating C_{out}?

Recall: C_{out} takes $O^*(2^n W)$-time with our framework.

How to dissolve the pseudopolynomial factor?

Approximating C_{out}?

Recall: C_{out} takes $O^*(2^n W)$-time with our framework.

How to dissolve the pseudopolynomial factor?

- We will approximate the optimal value of C_{out} within $1 + \varepsilon$.

Approximating C_{out}?

Recall: C_{out} takes $O^*(2^n W)$-time with our framework.

How to dissolve the pseudopolynomial factor?

- We will approximate the optimal value of C_{out} within $1+\varepsilon$.
- Example: The optimal plan has $C_{\mathrm{out}} = 400K$ and $\varepsilon = 1\%$.
  $\rightarrow$ Then we obtain a plan of cost $\leq 404K$.

Approximating C_{out}?

Recall: C_{out} takes $O^*(2^n W)$-time with our framework.

How to dissolve the pseudopolynomial factor?

- We will approximate the optimal value of C_{out} within $1+\varepsilon$.
- Example: The optimal plan has $C_{\mathrm{out}} = 400K$ and $\varepsilon = 1\%$.
  $\rightarrow$ Then we obtain a plan of cost $\leq 404K$.
- Running time: $O^*(2^n \log W/\varepsilon)$.

Approximate $(\min, +)$ Subset Convolution

Definition

Given two set functions f, g, approximate their $(\min, +)$ subset convolution:

$$(f \star g)(S) \leq \widetilde{h}(S) \leq (1 + \varepsilon)(f \star g)(S)$$

for all $S \subseteq [n]$, with $\varepsilon > 0$.

Approximate $(\min, +)$ Subset Convolution

Definition

Given two set functions f, g, approximate their $(\min, +)$ subset convolution:

$$(f \star g)(S) \leq \widetilde{h}(S) \leq (1 + \varepsilon)(f \star g)(S)$$

for all $S \subseteq [n]$, with $\varepsilon > 0$.

How to employ it

- Fix a relative error $\delta > 0$.

Approximate $(\min, +)$ Subset Convolution

Definition

Given two set functions f, g, approximate their $(\min, +)$ subset convolution:

$$(f \star g)(S) \leq \widetilde{h}(S) \leq (1 + \varepsilon)(f \star g)(S)$$

for all $S \subseteq [n]$, with $\varepsilon > 0$.

How to employ it

- Fix a relative error $\delta > 0$.
- Hence, the total error is $(1 + \delta)^{n-1}$.

Definition

Given two set functions f, g, approximate their $(\min, +)$ subset convolution:

$$(f \star g)(S) \leq \widetilde{h}(S) \leq (1 + \varepsilon)(f \star g)(S)$$

for all $S \subseteq [n]$, with $\varepsilon > 0$.

How to employ it

- Fix a relative error $\delta > 0$.
- Hence, the total error is $(1 + \delta)^{n-1}$.
- To obtain $(1 + \varepsilon)$-approximation, simply set $\delta := \Theta(\varepsilon/(n-1))$.

Evaluation

Benchmarks

- Clique queries with cardinalities $c(S)$ s.t. $c(S) \leq c(S_1) \cdot c(S_1)$.

Evaluation

Benchmarks

- Clique queries with cardinalities $c(S)$ s.t. $c(S) \leq c(S_1) \cdot c(S_1)$.
- When comparing to the A^*-based optimizer (SIGMOD'23), we take their original clique queries since their algorithm performance is sensitive to the cardinalities.

Competitors

- `DPsub[max, out]`: $\Theta(3^n)$ – independent of the query shape.

Evaluation

Benchmarks

- Clique queries with cardinalities $c(S)$ s.t. $c(S) \leq c(S_1) \cdot c(S_1)$.
- When comparing to the A^*-based optimizer (SIGMOD'23), we take their original clique queries since their algorithm performance is sensitive to the cardinalities.

Competitors

- `DPsub[max, out]`: $\Theta(3^n)$ – independent of the query shape.
- `DPccp` $=$ `DPsub` on clique queries.

Evaluation

Benchmarks

- Clique queries with cardinalities $c(S)$ s.t. $c(S) \leq c(S_1) \cdot c(S_1)$.
- When comparing to the A^*-based optimizer (SIGMOD'23), we take their original clique queries since their algorithm performance is sensitive to the cardinalities.

Competitors

- `DPsub[max, out]`: $\Theta(3^n)$ – independent of the query shape.
- `DPccp` = `DPsub` on clique queries.
- A^*`[out]`: $O(\#\texttt{ccp})$ – adapts to the cardinalities.

Evaluation

Benchmarks

- Clique queries with cardinalities $c(S)$ s.t. $c(S) \leq c(S_1) \cdot c(S_1)$.
- When comparing to the A^*-based optimizer (SIGMOD'23), we take their original clique queries since their algorithm performance is sensitive to the cardinalities.

Competitors

- `DPsub[max, out]`: $\Theta(3^n)$ – independent of the query shape.
- `DPccp` $=$ `DPsub` on clique queries.
- A^*`[out]`: $O(\#\texttt{ccp})$ – adapts to the cardinalities.
- `DPconv[max]`: $\Theta(2^n n^3)$ – independent of the query shape.

Evaluation: C_{max}

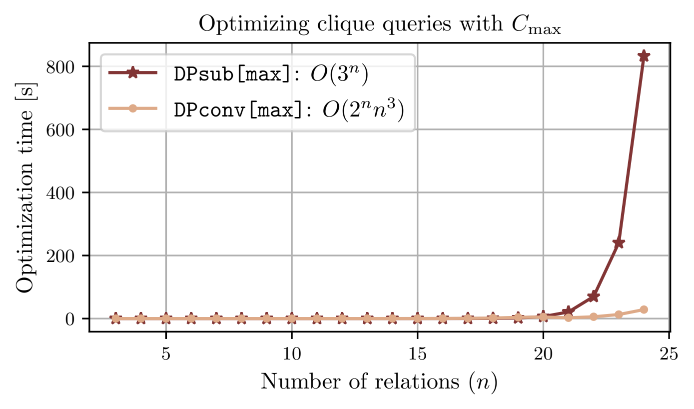


Figure 7: Optimizing for C_{max} on clique queries

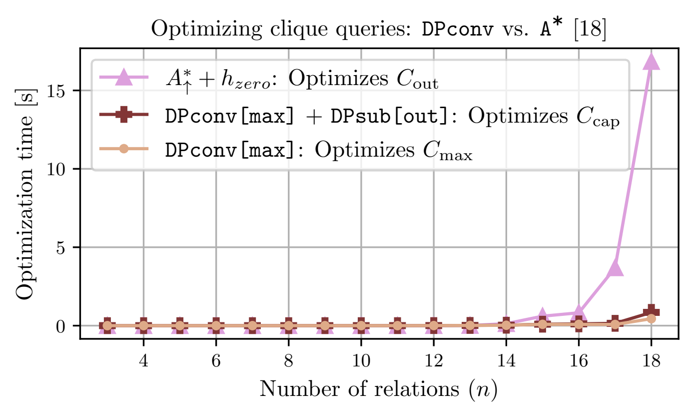


Figure 8: Optimizing for C_{out} and C_{cap} on clique queries

Evaluation: $C_{\text{cap}} = C_{\mathsf{max}} + C_{\text{out}}$

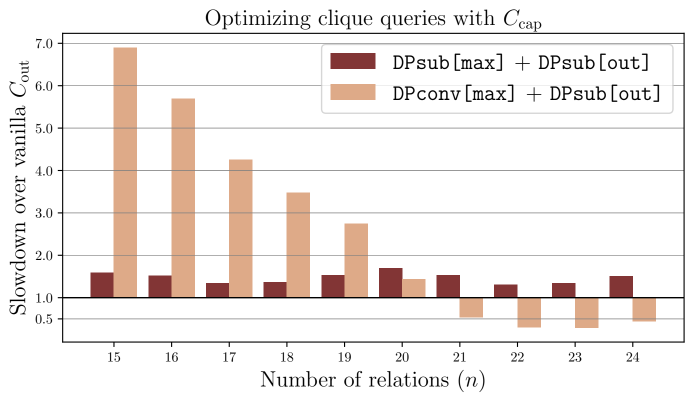


Figure 9: Optimizing for C_{cap} on clique queries

Open Problems

- Currently, `DPconv` is agnostic to the query shape.
  - For $C_{\max}$, it *always* runs in $O(2^n n^3)$-time.
  - We would need a sparse subset convolution.
  - Sparse = Few connected subgraphs.

Open Problems

- Currently, `DPconv` is agnostic to the query shape.
  - For $C_{\max}$, it *always* runs in $O(2^n n^3)$-time.
  - We would need a sparse subset convolution.
  - Sparse = Few connected subgraphs.
- Polynomial-*space* join ordering.
  - All exact join order optimizers take exponential *space*.
  - Example: Steiner tree can be solved in polynomial space.

Open Problems

- Currently, `DPconv` is agnostic to the query shape.
  - For $C_{\max}$, it *always* runs in $O(2^n n^3)$-time.
  - We would need a sparse subset convolution.
  - Sparse = Few connected subgraphs.
- Polynomial-*space* join ordering.
  - All exact join order optimizers take exponential *space*.
  - Example: Steiner tree can be solved in polynomial space.
- Jointly optimize the memory of concurrent queries using $C_{\max}$.
  - AutoWLM [2] can predict query's memory requirements.

Outlook

- This summer: Internship @GSL in Barcelona with Tiemo Bang.

S. Chatterji, S. S. K. Evani, S. Ganguly, and M. D. Yemmanuru.
On the Complexity of Approximate Query Optimization.
In L. Popa, S. Abiteboul, and P. G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 282–292. ACM, 2002.

G. Saxena, M. Rahman, N. Chainani, C. Lin, G. Caragea, F. Chowdhury, R. Marcus, T. Kraska, I. Pandis, and B. M. Narayanaswamy.
Auto-WLM: Machine Learning Enhanced Workload Management in Amazon Redshift.
In S. Das, I. Pandis, K. S. Candan, and S. Amer-Yahia, editors, *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 225–237. ACM, 2023.

Evaluation: Costs

- CEB benchmark (13,644 queries).
- 2,873 queries:
  - C_{out} has 6.8% larger C_{max}.
  - C_{max} looses 22.8% in C_{out}.
  - C_{cap} looses only 9.5% in C_{out} while maintaining optimal C_{max}.