Approximate Min-Sum Subset Convolution

Mihail Stoian

UTN

Data Systems Lab

September 6th, 2024

How Do *You* Deal With NP-Hardness?

UTN

Possible Answers

- Greedy algorithms.

How Do *You* Deal With NP-Hardness?

UTN

Possible Answers

- Greedy algorithms.
- (Polynomial-time) approximation algorithms.

How Do *You* Deal With NP-Hardness?

UTN

Possible Answers

- Greedy algorithms.
- (Polynomial-time) approximation algorithms.
- Parameterized algorithms.

How Do *You* Deal With NP-Hardness?

UTN

Possible Answers

- Greedy algorithms.
- (Polynomial-time) approximation algorithms.
- Parameterized algorithms.
- *Exponential-time* approximation algorithms.

How Do *You* Deal With NP-Hardness?

UTN

Possible Answers

- Greedy algorithms.
- (Polynomial-time) approximation algorithms.
- Parameterized algorithms.
- *Exponential-time* approximation algorithms.

Our Goal

Exact algorithms $\rightarrow$ *Exp-time approx.*
Out of the box.

Our Motivation

Database Systems

Not so known in TCS: Database join ordering.

Our Motivation

UTN

Database Systems

Not so known in TCS: Database join ordering.

```
SELECT count(*)
FROM customer c
JOIN orders o ON c.c_custkey = o.o_custkey
JOIN lineitem l ON o.o_orderkey = l.l_orderkey
JOIN supplier s ON l.l_suppkey = s.s_suppkey
```

Our Motivation

Database Systems

Not so known in TCS: Database join ordering.

- Exact algorithms known since 1970s.
- *Present in any database system you've heard of.*

Our Motivation

UTN

Database Systems

Not so known in TCS: Database join ordering.

- Exact algorithms known since 1970s.
- *Present in any database system you've heard of.*
- Hard to approximate [1].

Our Motivation

UTN

Database Systems

Not so known in TCS: Database join ordering.

- Exact algorithms known since 1970s.
- *Present in any database system you've heard of.*
- Hard to approximate [1].
- Hence, a long suite of greedy algorithms [2, 3, 4, 5].

Our Motivation

UTN

Database Systems

Not so known in TCS: Database join ordering.

- Exact algorithms known since 1970s.
- *Present in any database system you've heard of.*
- Hard to approximate [1].
- Hence, a long suite of greedy algorithms [2, 3, 4, 5].
- *No exp-time approximation algorithm*.

Our Motivation

UTN

Database Systems

Not so known in TCS: Database join ordering.

- Exact algorithms known since 1970s.
- *Present in any database system you've heard of.*
- Hard to approximate [1].
- Hence, a long suite of greedy algorithms [2, 3, 4, 5].
- *No exp-time approximation algorithm*.

Even beyond database systems:
Tensor contraction ordering — used in quantum circuit simulation.

Min-Sum Subset Convolution: Intro

The solution? The following “innocent” looking expression:

Definition

Given two set functions f, g, their min-sum subset convolution is:

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T) \right),$$

for all $S \subseteq [n]$.

The solution? The following "innocent" looking expression:

Definition

Given two set functions f, g, their min-sum subset convolution is:

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T)\right),$$

for all $S \subseteq [n]$.

Min-Sum Subset Convolution: Intro

UTN

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T)\right)$$

	f		g		$f * g$
000:	1		4		5
001:	6		0		1
010:	7		2		3
011:	3		0		1
100:	3	*	1	=	2
101:	5		5		3
110:	4		8		5
111:	1		3		3

Min-Sum Subset Convolution: Intro

UTN

Ubiquity

- Minimum Steiner Tree [6], Prize-Collecting Steiner Tree [7],
- Min-Cost k-Coloring [8],
- Computational Biology [9],
- and many others [10, 11].

Min-Sum Subset Convolution: Hands-On

Let's see what it (roughly) looks like for minimum Steiner tree:

Min-Sum Subset Convolution: Hands-On

Let's see what it (roughly) looks like for minimum Steiner tree:

$$\mathrm{DP}[X, v] = \min_{X' \subseteq X,\, u \in V \setminus T} \mathrm{DP}[X', v] + \mathrm{DP}[X \setminus X', u] + w(uv).$$

Min-Sum Subset Convolution: Hands-On

UTN

Let's see what it (roughly) looks like for minimum Steiner tree:

$$\mathrm{DP}[X, v] = \min_{X' \subseteq X,\, u \in V \setminus T} \mathrm{DP}[X', v] + \mathrm{DP}[X \setminus X', u] + w(uv).$$

Min-Sum Subset Convolution: Hands-On

UTN

Let's see what it (roughly) looks like for minimum Steiner tree:

$$\mathrm{DP}[X, v] = \min_{X' \subseteq X,\, u \in V \setminus T} \mathrm{DP}[X', v] + \mathrm{DP}[X \setminus X', u] + w(uv).$$

How expensive is it to compute?

Min-Sum Subset Convolution: Computation

UTN

Definition

Given two set functions f, g, their min-sum subset convolution is:

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T) \right),$$

for all $S \subseteq [n]$.

Runtime

Two flavors so far:

Min-Sum Subset Convolution: Computation

Definition

Given two set functions f, g, their min-sum subset convolution is:

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T) \right),$$

for all $S \subseteq [n]$.

Runtime

Two flavors so far:

- Naïve: $O(\sum_k \binom{n}{k} 2^k) = O((1+2)^n) = O(3^n)$ [ad-hoc]

Min-Sum Subset Convolution: Computation

UTN

Definition

Given two set functions f, g, their min-sum subset convolution is:

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T) \right),$$

for all $S \subseteq [n]$.

Runtime

Two flavors so far:

- Naïve: $O(\sum_k \binom{n}{k} 2^k) = O((1+2)^n) = O(3^n)$ [ad-hoc]
- Via embedding: $\widetilde{O}(2^n M)$, where M is the max in f, g [12].

Min-Sum Subset Convolution: Computation

UTN

Definition

Given two set functions f, g, their min-sum subset convolution is:

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T) \right),$$

for all $S \subseteq [n]$.

Runtime

Two flavors so far:

- Naïve: $O(\sum_k \binom{n}{k} 2^k) = O((1+2)^n) = O(3^n)$ [ad-hoc]
- Via embedding: $\widetilde{O}(2^n M)$, where M is the max in f, g [12].
  - Known as the bounded-input algorithm.

UTN

Definition

Given two set functions f, g, their min-sum subset convolution is:

$$(f * g)(S) = \min_{T \subseteq S} \left(f(T) + g(S \setminus T) \right),$$

for all $S \subseteq [n]$.

Runtime

Two flavors so far:

- Naïve: $O(\sum_k \binom{n}{k} 2^k) = O((1+2)^n) = O(3^n)$ [ad-hoc]
- Via embedding: $\widetilde{O}(2^n M)$, where M is the max in f, g [12].
  - Known as the bounded-input algorithm.

The applications inherit these runtimes.*

*Unless specialized algorithms exist, e.g., minimum Steiner tree [13].

How To Enable Approximation

UTN

Out-of-the-box $(1 + \varepsilon)$-Approximation.

Replace min-sum subset convolution with $(1 + \varepsilon)$-approximation.

How To Enable Approximation

UTN

Out-of-the-box $(1+\varepsilon)$-Approximation.

Replace min-sum subset convolution with $(1+\varepsilon)$-approximation.

Would solve database join ordering faster.
Would target the NP-hard combinatorial problems from before.

How To Enable Approximation

UTN

Out-of-the-box $(1+\varepsilon)$-Approximation.

Replace min-sum subset convolution with $(1+\varepsilon)$-approximation.

Would solve database join ordering faster.
Would target the NP-hard combinatorial problems from before.

But wait..

No $(1+\varepsilon)$-aproximate min-sum subset convolution so far.

Approximate Min-Sum Subset Convolution

UTN

Definition

Given two set functions f, g, approximate their min-sum subset convolution:

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

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

Approximate Min-Sum Subset Convolution

UTN

Definition

Given two set functions f, g, approximate their min-sum subset convolution:

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

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

Technical Results

Approximate Min-Sum Subset Convolution

UTN

Definition

Given two set functions f, g, approximate their min-sum subset convolution:

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

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

Technical Results

Theorem

*We can have an $(1+\varepsilon)$-approximation in time $\widetilde{O}(2^n \log M/\varepsilon)$.**

* $\widetilde{O}(\cdot)$ hides $n^{O(1)}$ factors.

Approximate Min-Sum Subset Convolution

UTN

Definition

Given two set functions f, g, approximate their min-sum subset convolution:

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

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

Technical Results

Theorem

*We can have an $(1+\varepsilon)$-approximation in time $\widetilde{O}(2^n \log M/\varepsilon)$.**

Theorem

We can have an $(1+\varepsilon)$-approximation in time $\widetilde{O}(2^{\frac{3n}{2}}/\sqrt{\varepsilon})$.

*$\widetilde{O}(\cdot)$ hides $n^{O(1)}$ factors.

More Results

UTN

Remember, this is now a generic tool. Some examples:

More Results

UTN

Remember, this is now a generic tool. Some examples:

Theorem

We can find an $(1+\varepsilon)$*-approximation for prize-collecting Steiner tree in time* $\widetilde{O}(2^{s^+}\log M/\varepsilon)$*.**

*$s^+ = \#$proper potential terminals.

More Results

UTN

Remember, this is now a generic tool. Some examples:

Theorem

*We can find an $(1+\varepsilon)$-approximation for prize-collecting Steiner tree in time $\widetilde{O}(2^{s^+} \log M/\varepsilon)$.**

Theorem

We can find an $(1+\varepsilon)$-approximation for prize-collecting Steiner tree in time $\widetilde{O}(2^{\frac{3s^+}{2}}/\sqrt{\varepsilon})$.

---

* $s^+ = \#$proper potential terminals.

More Results

UTN

Remember, this is now a generic tool. Some examples:

Theorem

We can find an $(1+\varepsilon)$*-approximation for prize-collecting Steiner tree in time* $\widetilde{O}(2^{s^+} \log M/\varepsilon)$.*

Theorem

We can find an $(1+\varepsilon)$*-approximation for prize-collecting Steiner tree in time* $\widetilde{O}(2^{\frac{3s^+}{2}}/\sqrt{\varepsilon})$.

Theorem

We can find an $(1+\varepsilon)$*-approximation for min-cost* k*-coloring in time* $\widetilde{O}(2^{\frac{3n}{2}}/\sqrt{\varepsilon})$.

---

* $s^+ = \#$proper potential terminals.

Since the work of Björklund et al. [12], no other algorithms for tropical semi-rings. We revive this line of research after ≈20 years:

Since the work of Björklund et al. [12], no other algorithms for tropical semi-rings. We revive this line of research after ≈20 years:

Reference	Type	(Semi-)Ring	Running Time
ad-hoc	exact	any	$O(3^n)$
Björklund et al.	exact	$(+, \times)$	$O(2^n n^2)$
Björklund et al.	exact	$(\min, +)$	$\widetilde{O}(2^n M)$

Subset Convs in Tropical Semi-Rings

UTN

Since the work of Björklund et al. [12], no other algorithms for tropical semi-rings. We revive this line of research after ≈20 years:

Reference	Type	(Semi-)Ring	Running Time
ad-hoc	exact	any	$O(3^n)$
Björklund et al.	exact	$(+, \times)$	$O(2^n n^2)$
Björklund et al.	exact	$(\min, +)$	$\widetilde{O}(2^n M)$
this work	exact	$(\min, \max)$	$\widetilde{O}(2^{\frac{3n}{2}})$
this work	$(1+\varepsilon)$-apx	$(\min, +)$	$\widetilde{O}(2^n \log M/\varepsilon)$
this work	$(1+\varepsilon)$-apx	$(\min, +)$	$\widetilde{O}(2^{\frac{3n}{2}}/\sqrt{\varepsilon})$

Table: Reviving research on tropical subset convolutions.

Technical Overview

Sequence Convs $\bowtie$ Subset Convs

UTN

Sequence convs and subset convs have been considered separately.
We initiate their study as a common object.

Sequence Convs $\bowtie$ Subset Convs

UTN

Sequence convs and subset convs have been considered separately. We initiate their study as a common object.

Min-Plus Sequence Convolution

Given two sequences $(a_i)_{i\in[n]}, (b_i)_{i\in[n]}$, their min-plus sequence conv is:

$$(a * b)_k = \min_{i+j=k} (a_i + b_j)\,,$$

for all $k \in [n]$.

Sequence Convs $\bowtie$ Subset Convs

UTN

Sequence convs and subset convs have been considered separately.
We initiate their study as a common object.

Min-Plus Sequence Convolution

Given two sequences $(a_i)_{i\in[n]}, (b_i)_{i\in[n]}$, their min-plus sequence conv is:

$$(a * b)_k = \min_{i+j=k} (a_i + b_j)\,,$$

for all $k \in [n]$.

Sequence Convs: Rich Literature

UTN

Sequence convs and subset convs have been considered separately. We initiate their study as a common object.

Min-Plus Sequence Convolution

Given two sequences $(a_i)_{i\in[n]}, (b_i)_{i\in[n]}$, their min-plus sequence conv is:

$$(a * b)_k = \min_{i+j=k} (a_i + b_j)\,,$$

for all $k \in [n]$.

Sequence Convs: Rich Literature

- $(\min, +)$-conv widely used as hardness [14, 15].

Sequence Convs $\bowtie$ Subset Convs

UTN

Sequence convs and subset convs have been considered separately. We initiate their study as a common object.

Min-Plus Sequence Convolution

Given two sequences $(a_i)_{i\in[n]}, (b_i)_{i\in[n]}$, their min-plus sequence conv is:

$$(a * b)_k = \min_{i+j=k} (a_i + b_j),$$

for all $k \in [n]$.

Sequence Convs: Rich Literature

- $(\min, +)$-conv widely used as hardness [14, 15].
- (Many) approximation algorithms [16, 17, 18].
  - Main application: Tree sparsity [16].

Sequence Convs $\bowtie$ Subset Convs

UTN

Sequence convs and subset convs have been considered separately. We initiate their study as a common object.

Min-Plus Sequence Convolution

Given two sequences $(a_i)_{i\in[n]}, (b_i)_{i\in[n]}$, their min-plus sequence conv is:

$$(a * b)_k = \min_{i+j=k} (a_i + b_j)\,,$$

for all $k \in [n]$.

Sequence Convs: Rich Literature

- $(\min, +)$-conv widely used as hardness [14, 15].
- (Many) approximation algorithms [16, 17, 18].
  - Main application: Tree sparsity [16].
- *However*, no connection to min-sum *subset* conv so far.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

*Note: Input size is $O(2^n)$.

Weakly-Polynomial Approximation Algorithm

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

*Note: Input size is $O(2^n)$.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

1. Consider powers of two in decreasing order:
   $2^{\lceil \log 2M \rceil}, \ldots, 4, 2, 1.$

*Note: Input size is $O(2^n)$.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

1. Consider powers of two in decreasing order: $2^{\lceil \log 2M \rceil}, \ldots, 4, 2, 1$.
2. Scale the values $\rightarrow$ input becomes bounded.

*Note: Input size is $O(2^n)$.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

1. Consider powers of two in decreasing order: $2^{\lceil \log 2M \rceil}, \ldots, 4, 2, 1.$
2. Scale the values $\rightarrow$ input becomes bounded.
3. Run the bounded-input algorithm.

*Note: Input size is $O(2^n)$.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

1. Consider powers of two in decreasing order: $2^{\lceil \log 2M \rceil}, \ldots, 4, 2, 1$.
2. Scale the values $\rightarrow$ input becomes bounded.
3. Run the bounded-input algorithm.

This only works because:

- Min-plus sequence conv runs in time $\widetilde{O}(nM)$ via FFT [19].†

*Note: Input size is $O(2^n)$.

†This hides polylog(nM) factors.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

1. Consider powers of two in decreasing order: $2^{\lceil \log 2M \rceil}, \ldots, 4, 2, 1$.
2. Scale the values $\rightarrow$ input becomes bounded.
3. Run the bounded-input algorithm.

This only works because:

- Min-plus sequence conv runs in time $\widetilde{O}(nM)$ via FFT [19].†
- Min-sum subset conv runs in time $\widetilde{O}(2^n M)$ [12].

*Note: Input size is $O(2^n)$.

†This hides polylog(nM) factors.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

1. Consider powers of two in decreasing order: $2^{\lceil \log 2M \rceil}, \ldots, 4, 2, 1$.
2. Scale the values $\rightarrow$ input becomes bounded.
3. Run the bounded-input algorithm.

This only works because:

- Min-plus sequence conv runs in time $\widetilde{O}(nM)$ via FFT [19].†
- Min-sum subset conv runs in time $\widetilde{O}(2^n M)$ [12].

Theorem

We can have $(1+\varepsilon)$-approximation in time $\widetilde{O}(2^n \log M/\varepsilon)$.

---

*Note: Input size is $O(2^n)$.

†This hides polylog(nM) factors.

Weakly-Polynomial Approximation Algorithm

UTN

Let us start with the weakly-poly approx algorithm (easy).*

Key & Standard Idea: Scaling

1. Consider powers of two in decreasing order: $2^{\lceil \log 2M \rceil}, \ldots, 4, 2, 1$.
2. Scale the values $\rightarrow$ input becomes bounded.
3. Run the bounded-input algorithm.

This only works because:

- Min-plus sequence conv runs in time $\widetilde{O}(nM)$ via FFT [19].†
- Min-sum subset conv runs in time $\widetilde{O}(2^n M)$ [12].

Theorem

We can have $(1+\varepsilon)$*-approximation in time* $\widetilde{O}(2^n \boxed{\log M} / \varepsilon)$.

*Note: Input size is $O(2^n)$.

†This hides polylog(nM) factors.

Can we remove the depedency on M?

Strongly-Polynomial Approximation Algorithm

Can we remove the depedency on M?

Overview

- We can use [BKW, STOC'19]'s framework for strongly-polynomial approx min-plus sequence conv.
- Initially, only developed for sequence convolutions. They asked for more applications.

Strongly-Polynomial Approximation Algorithm

UTN

Can we remove the depedency on M?

Overview

- We can use [BKW, STOC'19]'s framework for strongly-polynomial approx min-plus sequence conv.
- Initially, only developed for sequence convolutions. They asked for more applications.

$\rightarrow$ We would need a *min-max* subset convolution for this. Not present in literature before.

UTN

Definition

Given two set functions f, g, their min-max subset convolution is:

$$(f \otimes g)(S) = \min_{T \subseteq S} \max\{f(T), g(S \setminus T)\},$$

for all $S \subseteq [n]$.

A Detour: Min-Max Subset Convolution

UTN

Definition

Given two set functions f, g, their min-max subset convolution is:

$$(f \oslash g)(S) = \min_{T \subseteq S} \max\{f(T), g(S \setminus T)\},$$

for all $S \subseteq [n]$.

Theorem

Min-max subset convolution can be solved in time $\widetilde{O}(2^{\frac{3n}{2}})$.

Definition

Given two set functions f, g, their min-max subset convolution is:

$$(f \otimes g)(S) = \min_{T \subseteq S} \max\{f(T), g(S \setminus T)\},$$

for all $S \subseteq [n]$.

Theorem

Min-max subset convolution can be solved in time $\widetilde{O}(2^{\frac{3n}{2}})$.

Let's see how.

A Detour: Min-Max Subset Convolution

We adapt Kosaraju's algorithm for min-max sequence conv [20]:

A Detour: Min-Max Subset Convolution

UTN

We adapt Kosaraju's algorithm for min-max sequence conv [20]:

Overview

1. Collect values of f, g into a common list $\mathcal{L}$.

A Detour: Min-Max Subset Convolution

UTN

We adapt Kosaraju's algorithm for min-max sequence conv [20]:

Overview

1. Collect values of f, g into a common list $\mathcal{L}$.
2. Sort $\mathcal{L}$.

A Detour: Min-Max Subset Convolution

We adapt Kosaraju's algorithm for min-max sequence conv [20]:

Overview

1. Collect values of f, g into a common list $\mathcal{L}$.
2. Sort $\mathcal{L}$.
3. Divide $\mathcal{L}$ in chunks of size $O(\sqrt{2^n})$.

A Detour: Min-Max Subset Convolution

UTN

We adapt Kosaraju's algorithm for min-max sequence conv [20]:

Overview

1. Collect values of f, g into a common list $\mathcal{L}$.
2. Sort $\mathcal{L}$.
3. Divide $\mathcal{L}$ in chunks of size $O(\sqrt{2^n})$.
4. Use fast boolean subset convolution on $[f \leq \max \mathcal{C}], [g \leq \max \mathcal{C}]$, where $\mathcal{C} =$ the current chunk.

A Detour: Min-Max Subset Convolution

UTN

We adapt Kosaraju's algorithm for min-max sequence conv [20]:

Overview

1. Collect values of f, g into a common list $\mathcal{L}$.
2. Sort $\mathcal{L}$.
3. Divide $\mathcal{L}$ in chunks of size $O(\sqrt{2^n})$.
4. Use fast boolean subset convolution on $[f \leq \max \mathcal{C}], [g \leq \max \mathcal{C}]$, where $\mathcal{C} =$ the current chunk.
5. Naïvely solve for the sets that got *activated* at the current chunk.

Strongly-Polynomial Approximation Algorithm

UTN

Back to our initial goal: Strongly-polynomial approx.

Back to our initial goal: Strongly-polynomial approx.

BKW's Framework Meets Subset Convolutions

- Applying BKW's framework [18] would only yield $\widetilde{O}(2^{\frac{3n}{2}}/\varepsilon)$.
- However, this can be improved.
- We adapt their refined analysis to the subset setting.

Strongly-Polynomial Approximation Algorithm

UTN

Back to our initial goal: Strongly-polynomial approx.

BKW's Framework Meets Subset Convolutions

- Applying BKW's framework [18] would only yield $\widetilde{O}(2^{\frac{3n}{2}}/\varepsilon)$.
- However, this can be improved.
- We adapt their refined analysis to the subset setting.
- Final runtime: $\widetilde{O}(2^{\frac{3n}{2}}/\sqrt{\varepsilon})$.

How To Enable Approximation: Example

UTN

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \dots, M\}$ be the cost function.

How To Enable Approximation: Example

UTN

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \ldots, M\}$ be the cost function.

Mimimize: $\displaystyle\sum_{v \in V(G)} c(v, \chi(v))$.

How To Enable Approximation: Example

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \ldots, M\}$ be the cost function.

Mimimize: $\sum_{v \in V(G)} c(v, \chi(v))$.

Define:

$$s_i(X) = \begin{cases} \sum_{x \in X} c(x, i), & \text{if } X \text{ is an independent set,} \\ +\infty, & \text{otherwise.} \end{cases}$$

How To Enable Approximation: Example

UTN

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \ldots, M\}$ be the cost function.

Mimimize: $\sum_{v \in V(G)} c(v, \chi(v))$.

Define:

$$s_i(X) = \begin{cases} \sum_{x \in X} c(x, i), & \text{if } X \text{ is an independent set,} \\ +\infty, & \text{otherwise.} \end{cases}$$

Optimal cost: $(s_1 \star \ldots \star s_k)(X)$, for $X \in 2^{V(G)}$.

How To Enable Approximation: Example

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \ldots, M\}$ be the cost function.

Mimimize: $\sum_{v \in V(G)} c(v, \chi(v))$.

Define:

$$s_i(X) = \begin{cases} \sum_{x \in X} c(x, i), & \text{if } X \text{ is an independent set,} \\ +\infty, & \text{otherwise.} \end{cases}$$

Optimal cost: $(s_1 \star \ldots \star s_k)(X)$, for $X \in 2^{V(G)}$.

Approach

How To Enable Approximation: Example

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \dots, M\}$ be the cost function.

Mimimize: $\sum_{v \in V(G)} c(v, \chi(v))$.

Define:

$$s_i(X) = \begin{cases} \sum_{x \in X} c(x, i), & \text{if } X \text{ is an independent set,} \\ +\infty, & \text{otherwise.} \end{cases}$$

Optimal cost: $(s_1 \star \dots \star s_k)(X)$, for $X \in 2^{V(G)}$.

Approach

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

How To Enable Approximation: Example

UTN

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \dots, M\}$ be the cost function.

Mimimize: $\sum_{v \in V(G)} c(v, \chi(v))$.

Define:

$$s_i(X) = \begin{cases} \sum_{x \in X} c(x, i), & \text{if } X \text{ is an independent set,} \\ +\infty, & \text{otherwise.} \end{cases}$$

Optimal cost: $(s_1 \star \dots \star s_k)(X)$, for $X \in 2^{V(G)}$.

Approach

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

How To Enable Approximation: Example

UTN

Min-Cost k-Coloring

Let $c : V(G) \times [k] \to \{-M, \ldots, M\}$ be the cost function.

Mimimize: $\sum_{v \in V(G)} c(v, \chi(v))$.

Define:

$$s_i(X) = \begin{cases} \sum_{x \in X} c(x, i), & \text{if } X \text{ is an independent set,} \\ +\infty, & \text{otherwise.} \end{cases}$$

Optimal cost: $(s_1 \star \ldots \star s_k)(X)$, for $X \in 2^{V(G)}$.

Approach

- Fix a relative error $\delta > 0$.
- Hence, the total error is $(1 + \delta)^{k-1}$.
- Set $\delta := \Theta(\varepsilon/(k-1))$.

Discussion

UTN

Summary

- Out-of-the-box exp-time $(1 + \varepsilon)$-approximations.

Discussion

Summary

- Out-of-the-box exp-time $(1+\varepsilon)$-approximations.
- Swiss-army knife: $(1+\varepsilon)$-approximate min-sum subset conv.

Discussion

UTN

Summary

- Out-of-the-box exp-time $(1+\varepsilon)$-approximations.
- Swiss-army knife: $(1+\varepsilon)$-approximate min-sum subset conv.
- Refreshed subset convs in tropical semi-rings after $\approx$20 years.

Discussion

UTN

Summary

- Out-of-the-box exp-time $(1+\varepsilon)$-approximations.
- Swiss-army knife: $(1+\varepsilon)$-approximate min-sum subset conv.
- Refreshed subset convs in tropical semi-rings after ≈20 years.

(Major) Open Problems

- Polynomial speedups for min-sum subset convolution.
  $\rightarrow (\min, +)$ sequence convolution has a rich literature.

Discussion

UTN

Summary

- Out-of-the-box exp-time $(1+\varepsilon)$-approximations.
- Swiss-army knife: $(1+\varepsilon)$-approximate min-sum subset conv.
- Refreshed subset convs in tropical semi-rings after ≈20 years.

(Major) Open Problems

- Polynomial speedups for min-sum subset convolution.
  $\rightarrow (\min,+)$ sequence convolution has a rich literature.
- Conjecture similar to that in the sequence setting [14, 15]:

Conjecture

There is no $O((3-\delta)^n \text{polylog}(M))$-time exact algorithm for min-sum subset convolution, with $\delta > 0$.

References I

Sourav Chatterji, Sai Surya Kiran Evani, Sumit Ganguly, and Mahesh Datt Yemmanuru.
On the complexity of approximate query optimization.
In Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 282–292, 2002.

Donald Kossmann and Konrad Stocker.
Iterative dynamic programming: A new class of query optimization algorithms.
ACM Trans. Database Syst., 25(1):43–82, mar 2000.

References II

UTN

- Leonidas Fegaras.
  A new heuristic for optimizing large queries.
  In Proceedings of the 9th International Conference on Database and Expert Systems Applications, DEXA '98, page 726–735, Berlin, Heidelberg, 1998. Springer-Verlag.

- Chiang Lee, Chi-Sheng Shih, and Yaw-Huei Chen.
  Optimizing large join queries using a graph-based approach.
  IEEE Transactions on Knowledge and Data Engineering, 13(2):298–315, 2001.

- Thomas Neumann and Bernhard Radke.
  Adaptive optimization of very large join queries.
  In Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, page 677–692, New York, NY, USA, 2018. Association for Computing Machinery.

References III

Stuart E Dreyfus and Robert A Wagner.
The steiner problem in graphs.
Networks, 1(3):195–207, 1971.

Daniel Rehfeldt and Thorsten Koch.
On the exact solution of prize-collecting steiner tree problems.
INFORMS Journal on Computing, 34(2):872–889, 2022.

Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh, Marek Cygan, Fedor V Fomin, et al.
Algebraic techniques: sieves, convolutions, and polynomials.
Parameterized Algorithms, pages 321–355, 2015.

UTN

- Sebastian Böcker and Florian Rasche.
  Towards de novo identification of metabolites by analyzing tandem mass spectra.
  In Christian Huber, Oliver Kohlbacher, Michal Linial, Katrin Marcus, and Knut Reinert, editors, Computational Proteomics, volume 8101 of Dagstuhl Seminar Proceedings (DagSemProc), pages 1–5, Dagstuhl, Germany, 2008. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- Juha Harviainen and Mikko Koivisto.
  Revisiting bayesian network learning with small vertex cover.
  In Robin J. Evans and Ilya Shpitser, editors, Uncertainty in Artificial Intelligence, UAI 2023, July 31 - 4 August 2023, Pittsburgh, PA, USA, volume 216 of Proceedings of Machine Learning Research, pages 819–828. PMLR, 2023.

UTN

Oriana Ponta, Falk Hüffner, and Rolf Niedermeier.
Speeding up dynamic programming for some np-hard graph recoloring problems.
In Theory and Applications of Models of Computation: 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings 5, pages 490–501. Springer, 2008.

Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto.
Fourier meets möbius: fast subset convolution.
In Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, pages 67–74, 2007.

UTN

Bernhard Fuchs, Walter Kern, D Molle, Stefan Richter, Peter Rossmanith, and Xinhui Wang.
Dynamic programming for minimum steiner trees.
Theory of Computing Systems, 41:493–500, 2007.

Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michał Włodarczyk.
On problems equivalent to (min,+)-convolution.
ACM Transactions on Algorithms (TALG), 15(1):1–25, 2019.

Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider.
On the fine-grained complexity of one-dimensional dynamic programming.
arXiv preprint arXiv:1703.00941, 2017.

UTN

- Arturs Backurs, Piotr Indyk, and Ludwig Schmidt.
  Better approximations for tree sparsity in nearly-linear time.
  In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 2215–2229. SIAM, 2017.
- Marcin Mucha, Karol Wegrzycki, and Michał Włodarczyk.
  A subquadratic approximation scheme for partition.
  In Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 70–88. SIAM, 2019.
- Karl Bringmann, Marvin Künnemann, and Karol Wegrzycki.
  Approximating apsp without scaling: equivalence of approximate min-plus and exact min-max.
  In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pages 943–954, 2019.

K. Wegrzycki.
Provably Optimal Dynamic Programming.
2019.

S.R. Kosaraju.
Efficient tree pattern matching.
In 30th Annual Symposium on Foundations of Computer Science, pages 178–183, 1989.