Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner Tree and related problems

Jesper Nederlof

7 July 2009

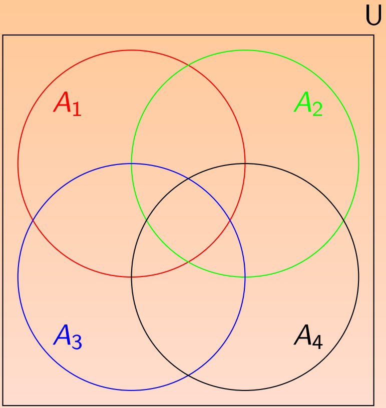
Outline

Exact exponential time algorithms

- We study the worst-case running time of algorithms for $\mathcal{NP}$-complete problems.
- The running times we achieve can typically be written as $c^n p(n)$, for some constant c, polynomial function p and input measure n.
- We will denote such a running time with $\mathcal{O}^*(c^n)$, ignoring the polynomial factor.

Inclusion-Exclusion: An example

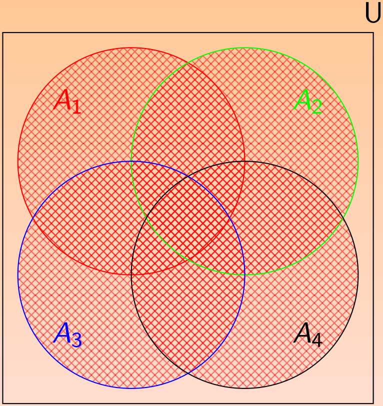
- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^{4} A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

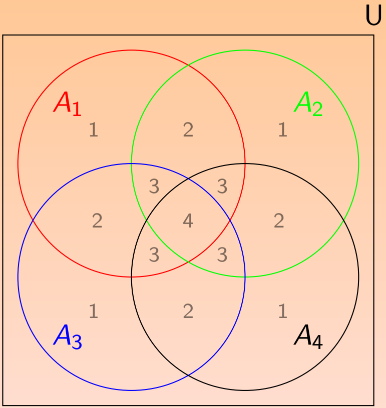
- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^{4} A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

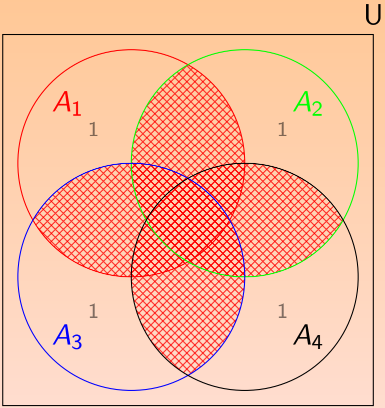
- Suppose we are given a family of subsets $A_1, \dots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^4 A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

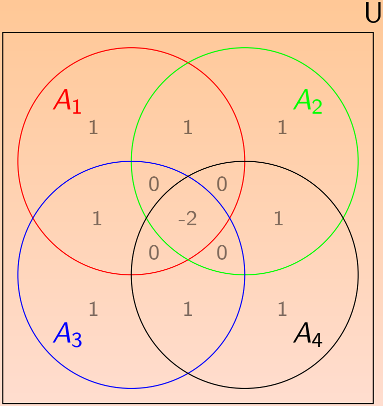
- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^{4} A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_{i} |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

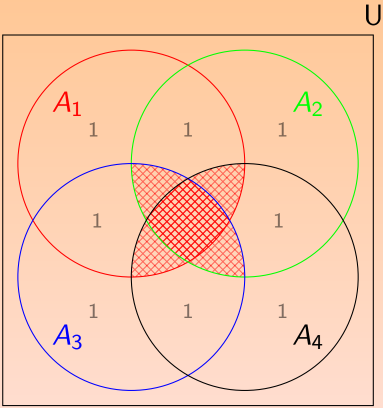
- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^{4} A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

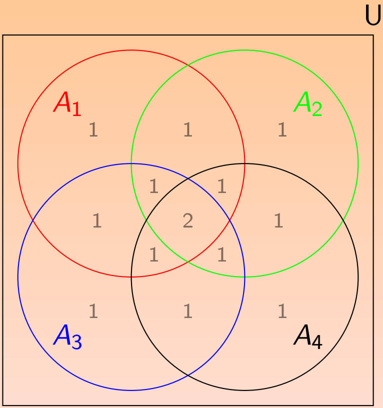
- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^{4} A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

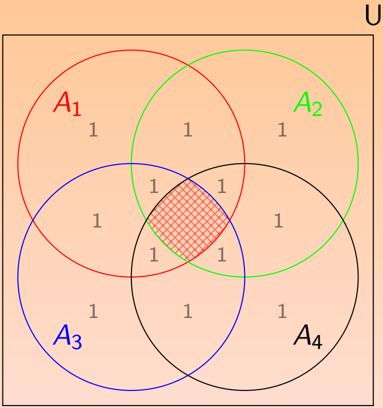
- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^4 A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

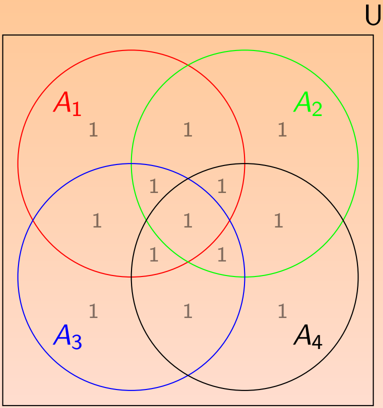
- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^{4} A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_{i} |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

Inclusion-Exclusion: An example

- Suppose we are given a family of subsets $A_1, \ldots, A_4 \subseteq U$
- We will compute $|\bigcup_{i=1}^4 A_i|$, by just using intersections.
- For notational ease, we assume all other sets A_i to be empty.



$$\sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|$$

The IE-formula

- More general, if $A_1, \ldots, A_n \subseteq U$ then:

$$|\bigcap_{i=1}^{n} \overline{A_i}| = \sum_{X \subseteq \{1,\ldots,n\}} (-1)^{|X|} |\bigcap_{i \in X} A_i|$$

where we define $\bigcap_{i \in \emptyset} A_i = U$ and $\overline{A_i} = U \setminus A_i$

- This equality is called the IE-formula, and each application of it is said to be an IE-formulation.

Finding and using IE-formulations

If we want to solve a counting problem with IE, then we have to:

- Think about an universe U, which at least contains everything we want to count. One could obtain an useful universe by relaxing constraints that are imposed on solutions.
- Define subsets $A_1, \dots, A_n$ such that $|\bigcap_{i=1}^{n} \overline{A_i}|$ is what we want to compute.
- Solve the problem of computing $|\bigcap_{i \in X} A_i|$, for a given $X \subseteq \{1, \dots, n\}$. We will call this the simplified problem.
- Notice that if the simplified problem can be solved in polynomial time, we can obtain an $\mathcal{O}^*(2^n)$-time polynomial space algorithm.

Hamiltonian path (Karp, 1982)

Definition (Hamiltonian path)

An Hamiltonian path in a graph G with nodes $v_1, \ldots, v_n$ is a walk of $n-1$ edges that visits all nodes.

- We relax the constraint that all nodes have to be visited and choose as universe U:
  - all walks of $n-1$ edges in G.
- Define A_i as all walks with $n-1$ edges that avoid node v_i. Now, we have that $|\bigcap_{i=1}^{n} \overline{A_i}|$ is the number of Hamiltonian paths of G.
- The simplified problem is to compute $|\bigcap_{i \in X} A_i|$, which is the number of walks of $n-1$ edges in the subgraph of G induced by nodes $\{v_1, \ldots, v_n\} \setminus X$.

The simplified problem

Define $w_X(s,k)$ as the number of walks with k edges from s that avoid the nodeset X.

$$w_X(s,0) = 1$$
$$w_X(s,k) = \sum_{t \in N(s) \setminus X} w_X(t, k-1)$$

where $N(s)$ are all neighbors of s.

- The number of walks with $n-1$ edges avoiding nodeset X, $|\bigcap_{i \in X} A_i|$, is equal to

$$\sum_{s \in V \setminus X} w_X(s, n-1)$$

- Hence, the simplified problem can be solved in polynomial time.

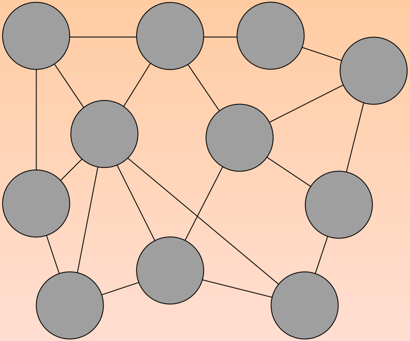
Hamiltonian path

Theorem (Karp, 1982)

Counting the number of hamiltonian paths can be done in $\mathcal{O}^*(2^n)$ time using polynomial space.

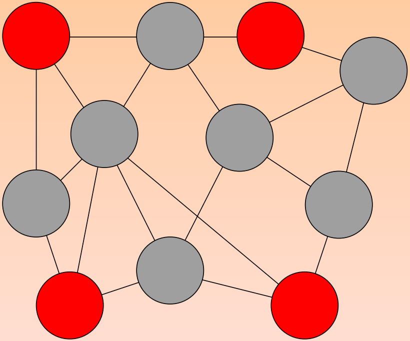
Unit weight Steiner tree

- Given a graph $G = (V, E)$ and a set of terminals $\{t_1, \dots, t_k\} \subseteq V$.
- A Steiner tree is a subtree $S \subseteq E$ connecting all terminals.
- We solve the decision variant: does there exist a Steiner tree with at most l edges?



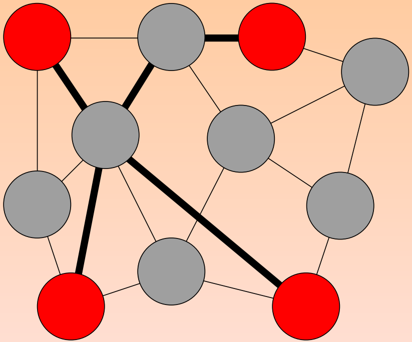
Unit weight Steiner tree

- Given a graph $G = (V, E)$ and a set of terminals $\{t_1, \dots, t_k\} \subseteq V$.
- A Steiner tree is a subtree $S \subseteq E$ connecting all terminals.
- We solve the decision variant: does there exist a Steiner tree with at most l edges?



Unit weight Steiner tree

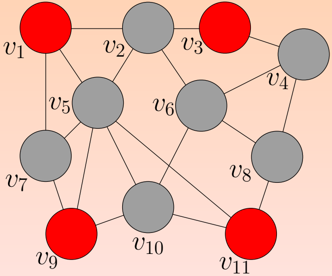
- Given a graph $G = (V, E)$ and a set of terminals $\{t_1, \ldots, t_k\} \subseteq V$.
- A Steiner tree is a subtree $S \subseteq E$ connecting all terminals.
- We solve the decision variant: does there exist a Steiner tree with at most l edges?

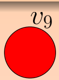


Steiner tree: Branching walk

Definition (Branching walk)

A branching walk in $G=(V,E)$ is a pair $B=(T_B,\phi)$ where $T_B=(V_B,E_B)$ is a rooted, ordered tree and $\phi: V_B \to V$ is a homomorphism from T_B to G. The length of B is $|E_B|$, and B is from s if the root of T_B is mapped to s by ϕ.

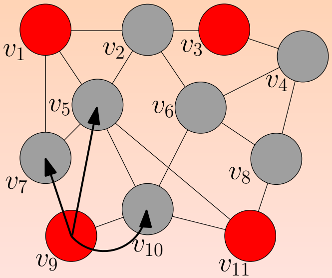


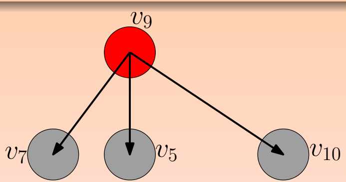


Steiner tree: Branching walk

Definition (Branching walk)

A branching walk in $G = (V, E)$ is a pair $B = (T_B, \phi)$ where $T_B = (V_B, E_B)$ is a rooted, ordered tree and $\phi : V_B \rightarrow V$ is a homomorphism from T_B to G. The length of B is $|E_B|$, and B is from s if the root of T_B is mapped to s by ϕ.

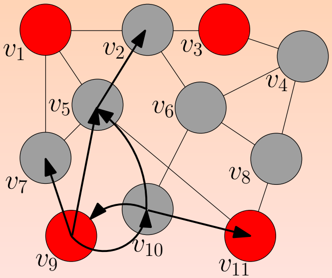


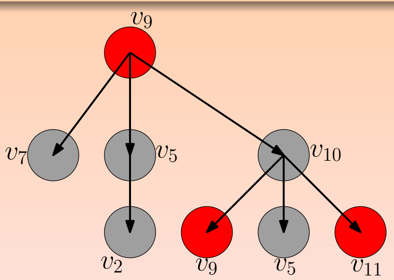


Steiner tree: Branching walk

Definition (Branching walk)

A branching walk in $G = (V, E)$ is a pair $B = (T_B, \phi)$ where $T_B = (V_B, E_B)$ is a rooted, ordered tree and $\phi : V_B \to V$ is a homomorphism from T_B to G. The length of B is $|E_B|$, and B is from s if the root of T_B is mapped to s by ϕ.

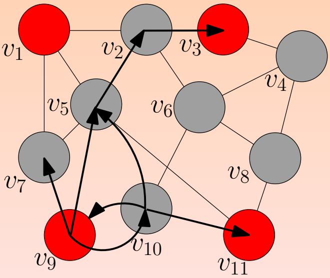


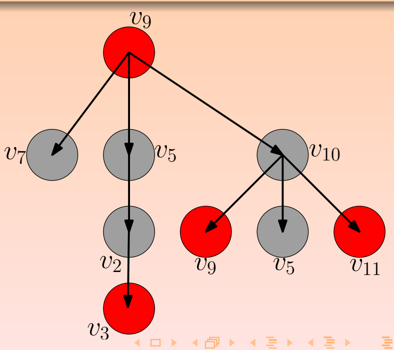


Steiner tree: Branching walk

Definition (Branching walk)

A branching walk in $G = (V, E)$ is a pair $B = (T_B, \phi)$ where $T_B = (V_B, E_B)$ is a rooted, ordered tree and $\phi : V_B \to V$ is a homomorphism from T_B to G. The length of B is $|E_B|$, and B is from s if the root of T_B is mapped to s by ϕ.





Steiner tree: Reformulating

Lemma

Let s be a terminal. There exists a branching walk from s of length at most l that visits all terminals if and only if there exists a Steiner tree T with at most l edges.

Proof.

$(\Leftarrow)$: Choose T_B to be T with root s and ϕ to be the identity function.
$(\Rightarrow)$: Consider the minimal subgraph of G in which B is still a branching walk. Choose T as a spanning tree of this graph. □

Steiner tree: An IE-formulation

- Define U as all branching walks from s of length l.
- Define A_i as all branching walks avoiding terminal t_i.
- The input is a YES-instance iff

$$|\bigcap_{i=1}^{k} \overline{A_i}| > 0$$

- It remains to solve the simplified problem.

Counting branching walks

- The simplified problem is to count the number of branching walks from s in $G[V \setminus X]$.
- Let $b_X(s,j)$ be the number of branching walks from s of length l in $G[V \setminus X]$.
- $b_X(s,0) = 1$, and for $j > 0$:

$$b_X(s,j) = \sum_{t \in N(s) \setminus X} \sum_{i=0}^{j-1} b_X(s,i)\, b_X(t,j-1-i)$$

Steiner tree

- This gives us an $\mathcal{O}^*(2^k)$-time poly-space algorithm.
- Using the concept of branching walks, we can also obtain $\mathcal{O}^*(2^n)$-time poly-space algorithms for finding spanning trees that
  - minimize the maximum degree (Degree Constrained Spanning Tree).
  - maximize the number of internal nodes (Max Internal Spanning Tree).
- (Both are known to be NP-complete).

Möbius inversion

Definition (Zeta and Möbius transform)

Given a function $f : 2^V \to \mathbb{Z}^+$, the zeta-transform ζf and the Möbius-transform μf are defined as follows:

$$\zeta f(V) = \sum_{X \subseteq V} f(X) \qquad \mu f(V) = \sum_{X \subseteq V} (-1)^{|V \setminus X|} f(X)$$

Theorem (Möbius inversion)

The Möbius-transform is the inverse of the zeta-transform, i.e. for each $f : 2^V \to \mathbb{Z}^+$:

$$f(V) = \sum_{X \subseteq V} (-1)^{|V \setminus X|} \sum_{A \subseteq X} f(A)$$

Möbius inversion

- Essentially, Möbius inversion and Inclusion-Exclusion are exactly the same.
- Computing the zeta-transform can be viewed as solving the simplified problem.
- So we should also be able to obtain the algorithms in a more structural (algebraical) way.

Hamiltonian path revisited

Definition (Hamiltonian path)

An Hamiltonian path in a graph G with nodes $v_1, \ldots, v_n$ is a walk of $n-1$ edges that visits all nodes.

We use an Held & Karp-style DP: let $h_l(s, R)$ be the number of walks from s of length l containing exactly the nodeset R.

$$h_l(s,R) = \begin{cases} [R = \emptyset] & \text{if } l = 0 \\ \sum_{t \in N(s) \cap R} h_{l-1}(t, R \setminus t) + h_{l-1}(t, R) & \text{otherwise} \end{cases}$$

Note that $h_{n-1}(s, V \setminus s)$ is the number of Hamiltonian paths from s.

$$h_l(s,R) = \begin{cases} [R=\emptyset] & \text{if } l=0 \\ \displaystyle\sum_{t \in N(s)\cap R} h_{l-1}(t, R\setminus t) + h_{l-1}(t,R) & \text{otherwise} \end{cases}$$

$$\zeta h_0(s,R) = \sum_{X\subseteq R} [X=\emptyset] = 1$$

$$\begin{aligned} \zeta h_l(s,R) &= \sum_{X\subseteq R} \sum_{t\in N(s)\cap X} h_{l-1}(t, X\setminus t) + h_{l-1}(t,X) \\ &= \sum_{t\in N(s)\cap R} \sum_{t\in X\subseteq R} h_{l-1}(t, X\setminus t) + h_{l-1}(t,X) \\ &= \sum_{t\in N(s)\cap R} \zeta h_{l-1}(t,R) \end{aligned}$$

Conclusion: $\zeta h_k(s,R) = w_{V\setminus R}(s,k)$

$$h_l(s,R) = \begin{cases} [R=\emptyset] & \text{if } l=0 \\ \sum_{t \in N(s)\cap R} h_{l-1}(t, R \setminus t) + h_{l-1}(t,R) & \text{otherwise} \end{cases}$$

$$\zeta h_0(s,R) = \sum_{X \subseteq R} [X=\emptyset] = 1$$

$$\begin{aligned} \zeta h_l(s,R) &= \sum_{X \subseteq R} \sum_{t \in N(s)\cap X} h_{l-1}(t, X \setminus t) + h_{l-1}(t,X) \\ &= \sum_{t \in N(s)\cap R} \sum_{t \in X \subseteq R} h_{l-1}(t, X \setminus t) + h_{l-1}(t,X) \\ &= \sum_{t \in N(s)\cap R} \zeta h_{l-1}(t,R) \end{aligned}$$

Conclusion: $\zeta h_k(s,R) = w_{V\setminus R}(s,k)$

$$h_l(s,R) = \begin{cases} [R=\emptyset] & \text{if } l=0 \\ \sum_{t \in N(s)\cap R} h_{l-1}(t, R\setminus t) + h_{l-1}(t,R) & \text{otherwise} \end{cases}$$

$$\zeta h_0(s,R) = \sum_{X \subseteq R} [X=\emptyset] = 1$$

$$\begin{aligned} \zeta h_l(s,R) &= \sum_{X\subseteq R} \sum_{t \in N(s)\cap X} h_{l-1}(t, X\setminus t) + h_{l-1}(t,X) \\ &= \sum_{t\in N(s)\cap R} \sum_{t \in X \subseteq R} h_{l-1}(t, X\setminus t) + h_{l-1}(t,X) \\ &= \sum_{t\in N(s)\cap R} \zeta h_{l-1}(t,R) \end{aligned}$$

Conclusion: $\zeta h_k(s,R) = w_{V\setminus R}(s,k)$

$$h_l(s,R) = \begin{cases} [R = \emptyset] & \text{if } l = 0 \\ \displaystyle\sum_{t \in N(s) \cap R} h_{l-1}(t, R \setminus t) + h_{l-1}(t, R) & \text{otherwise} \end{cases}$$

$$\zeta h_0(s,R) = \sum_{X \subseteq R} [X = \emptyset] = 1$$

$$\begin{aligned} \zeta h_l(s,R) &= \sum_{X \subseteq R} \sum_{t \in N(s) \cap X} h_{l-1}(t, X \setminus t) + h_{l-1}(t, X) \\ &= \sum_{t \in N(s) \cap R} \sum_{t \in X \subseteq R} h_{l-1}(t, X \setminus t) + h_{l-1}(t, X) \\ &= \sum_{t \in N(s) \cap R} \zeta h_{l-1}(t, R) \end{aligned}$$

Conclusion: $\zeta h_k(s,R) = w_{V \setminus R}(s,k)$

Subset convolution

Definition (Björklund et al., 2007)

Given two functions $f, g : 2^V \to \mathbb{Z}^+$, the cover product $(f *_c g)$ is defined as follows:

$$(f *_c g)(V) = \sum_{\substack{A,B \subseteq V \\ A \cup B = V}} f(A)g(B)$$

Subset convolution

Theorem (Björklund et al., 2007)

$\zeta((f *_c g)(V)) = \zeta f(V) * \zeta g(V)$

Proof.

$$
\begin{aligned}
\zeta(f *_c g) &= \sum_{X \subseteq V} \sum_{\substack{A,B \subseteq X \\ A \cup B = X}} f(A)g(B) \\
&= \Big(\sum_{A \subseteq V} f(A)\Big)\Big(\sum_{B \subseteq V} g(B)\Big) \\
&= \zeta f(V) * \zeta g(V)
\end{aligned}
$$

□

Subset convolution

- Equipped with subset convolution, the considered IE-formulation for Steiner Tree can be obtained by applying Möbius inversion to the Dreyfus-Wagner recurrence (1972).
- Using the same setup, one can also obtain $\mathcal{O}^*(2^n)$-time poly-space algorithms for computing the number of c-component spanning forests and the cover polynomial of a graph on n nodes.

Conclusions

- Möbius inversion is a powerful tool to improve the space requirement of some dynamic programming algorithms.
- All dynamic programming algorithms admitting the 'subset convolution shape' can be improved to polynomial-space algorithms.

The end

- Thank you all for your attention!