Fast Polynomial-Space Algorithms Using Möbius Inversion: Improving on Steiner Tree and Related Problems

Conference Paper *in* Algorithmica · July 2009

DOI: 10.1007/978-3-642-02927-1_59 · Source: DBLP

1 author:

Jesper Nederlof
Eindhoven University of Technology
77 PUBLICATIONS **1,425** CITATIONS

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

Jesper Nederlof

Department of Informatics, University of Bergen, N-5020 Bergen, Norway.
jesper.nederlof@ii.uib.no

Abstract. Given a graph with n vertices, k terminals and bounded integer weights on the edges, we compute the minimum Steiner Tree in $\mathcal{O}^*(2^k)$ time and polynomial space, where the $\mathcal{O}^*$ notation omits $\texttt{poly}(n,k)$ factors. Among our results are also polynomial-space $\mathcal{O}^*(2^n)$ algorithms for several $\mathcal{NP}$-complete spanning tree and partition problems.
The previous fastest known algorithms for these problems use the technique of dynamic programming among subsets, and require exponential space. We introduce the concept of branching walks and extend the Inclusion-Exclusion algorithm of Karp for counting Hamiltonian paths. Moreover, we show that our algorithms can also be obtained by applying Möbius inversion on the recurrences used for the dynamic programming algorithms.

1 Introduction

One of the most widely used techniques for achieving moderately exponential time algorithms for $\mathcal{NP}$-hard problems is dynamic programming among subsets, but unfortunately an exponential storage requirement seems to be inherent to this technique. As mentioned by Woeginger [20] this requirement makes them useless in practice. Therefore polynomial-space exact algorithms have already been studied for several $\mathcal{NP}$-hard problems [6, 7, 11, 16, 17, 20]. Hence, from both a theoretical and a practical perspective it is desirable to identify those dynamic programming algorithms that can be improved to require polynomial space, preferably maintaining the best known upper bound on the running time. In this paper we improve several algorithms in this way.

In 2006, Björklund et al. [6] drew new attention to the principle of Inclusion-Exclusion: they gave $\mathcal{O}^*(2^n)$-time algorithms[1] for several set partition problems, the most prominent one being k-Coloring. They also mention a simple adjustment to their algorithm to achieve an $\mathcal{O}^*(2.24^n)$-time algorithm with polynomial space for k-Coloring. Also related to this are the $\mathcal{O}^*(2^n)$-time polynomial-space algorithms for #Hamiltonian path by Karp [16] and (implicitly) Kohn et al. [17], and for #Perfect Matching by Björklund and Husfeldt [2].

* This work is supported by the Research Council of Norway.
[1] The $\mathcal{O}^*$ notation omits polynomial factors, and n denotes the number of nodes of the graph.

	Problem	References
$\mathcal{O}^*(2^k)$	Steiner tree*[2]	[3, 9, 11, 12]
$\mathcal{O}^*(2^n)$	Degree constrained spanning tree*	[13]
	Max internal spanning tree*	[10]
	#c-Spanning forests*	[4, 15]
	Cover polynomial*	[4]
	#Hamiltonian path	[16, 17]
	#Perfect matching	[2]
$\mathcal{O}^*(2.24^n)$	k-Coloring	[6, 7]

Table 1. An overview of problems that can be solved using polynomial space and with the given time bound using Inclusion-Exclusion. For the problems indicated with a *, we provide the first polynomial-space algorithm with the given running time.

We show that some dynamic programming algorithms can be improved to obtain polynomial-space algorithms with the same worst-case running time. Our algorithms heavily rely on the work of Björklund et al. [2–6]. The results can be read from Table 1: we add five problems to the list of polynomial space Inclusion-Exclusion algorithms (note that this list is not exhaustive).

Steiner Tree is one of the most well-studied $\mathcal{NP}$-complete problems. The Dreyfus-Wagner [9] dynamic programming algorithm has been the fastest exact algorithm for over 30 years. However, recently Björklund et al. [3] gave an $\mathcal{O}^*(2^k)$-time algorithm[2] for the variant with bounded integer weights, and Fuchs et al. [12] gave an $\mathcal{O}^*(c^k)$-time algorithm for the general case, for any $c > 2$. Both algorithms use $\Omega(2^k)$ space. In [11], Fomin et al. initiated the study of polynomial space algorithms for Steiner Tree. They gave polynomial space algorithms with running time bounded by $\mathcal{O}(5.96^k n^{\mathcal{O}(\log k)})$ and $\mathcal{O}(1.60^n)$ where n is the number of nodes in the graph. They pose the question whether Steiner Tree is fixed parameter tractable with respect to k when there is a polynomial space restriction. We answer this question affirmatively by providing an algorithm that runs in $\mathcal{O}^*(2^k)$ time and meets the restriction. Using the techniques of [11], this also leads to a polynomial-space $\mathcal{O}^*(1.3533^n)$-time algorithm.

The Max Internal Spanning Tree (MIST) and Degree Constrained Spanning Tree (DCST, also called Min-Max Degree Spanning Tree) problems are natural generalizations of Hamiltonian path. In [10], Fernau et al. ask if there exists an $\mathcal{O}^*(2^n)$-time algorithm to solve MIST. In [13], Gaspers et al. ask if there exists an $\mathcal{O}^*(2^n)$-time algorithm solving DCST. We answer both questions by giving polynomial-space algorithms with this running time.

The Cover polynomial of a directed graph introduced by Graham and Chung [8] generalizes all problems that can be solved using two operations named deletion and contraction of edges, and is designed to be the directed analogue of the Tutte polynomial. We improve the $\mathcal{O}^*(3^n)$-time polynomial-space algorithm of Björklund et al. [4] to an $\mathcal{O}^*(2^n)$-time polynomial-space algorithm. We also give the same improvement for #c-Spanning forests, which is one particular case of the Tutte polynomial. For more information about the Tutte polynomial we refer to [4].

The paper is organised as follows: in Section 2 we recall the principle of Inclusion-Exclusion and the well-known Hamiltonian path algorithm. After this we provide a natural extension by introducing the concept of *branching walks*

[2] k stands for the number of terminals.

and give the resulting algorithms. In the next section we show how the Inclusion-Exclusion algorithms can be obtained from dynamic programming algorithms by taking the *zeta transform* of the associated recurrences. After this, we give a more structural approach to the subset products introduced in [3] and provide applications.

We use the following definitions: for any set S, 2^S is the *power set* of S, i.e. the set consisting of all subsets of S. For a boolean expression b, $[b]$ stands for 1 if b is `true`, and for 0 if b is `false`. Let $G = (V, E)$ be a (directed) graph. Throughout the paper, we use $|V| = n$. The graph *induced by* X, where $X \subseteq V$, is the graph $G[X]$ with nodeset X and all edges in E only adjacent to nodes in X. For $v \in V$, $N(V)$ are all nodes adjacent to v. A *walk of length* k in G is a tuple $W = (v_1, \ldots, v_{k+1}) \in V^{k+1}$ such that $(v_i, v_{i+1}) \in E$ for each $0 \leq i \leq k$. W is from v if $v_1 = v$, and W is *cyclic* if $v_1 = v_{k+1}$. Let $G' = (V', E')$ also be a graph, a *homomorphism from* G *to* G' is a function $\phi : V \to V'$ such that $(u, v) \in E$ implies $(\phi(u), \phi(v)) \in E'$.

2 Inclusion-Exclusion formulations

Let us start this section by stating the principle of Inclusion-Exclusion. The following theorem can be found in many textbooks on discrete mathematics. For a proof see for example Bax [1].

Theorem 1 (Folklore). *Let U be a set and $A_1, \ldots, A_n \subseteq U$. With the convention $\bigcap_{i \in \emptyset} \overline{A_i} = U$, the following holds:*

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

In this paper, we call any application of the above theorem an *IE-formulation*. In this context we will refer to the set U as the *universe*, and to $A_1, \ldots, A_n$ as the *requirements*. Moreover, we call the task of computing $|\bigcap_{i \in X} \overline{A_i}|$ for an arbitrary $X \subseteq \{1, \ldots, n\}$ the *simplified problem*. Note that if the simplified problem can be computed in polynomial time, there exists an $\mathcal{O}^*(2^n)$-time polynomial-space algorithm that evaluates Equation 1.

We mention that it is also possible to break Theorem 1 down into smaller steps, i.e. we choose a subset of $\{1, \ldots, n\}$ and apply the theorem. This has recently been used for a faster exact algorithm for Dominating Set in van Rooij et al. [19]; see also Bax [1]. Other methods to speed up IE-formulations are given by Björklund et al. [2, 5] and are surveyed in [18].

We continue this section by giving some IE-formulations. The first one is well-known, but illustrative and it will be extended in the next subsections.

2.1 Hamiltonian Paths

Given a graph $G = (V, E)$, a Hamiltonian path is a walk that contains each node exactly once[3]. The #Hamiltonian Path problem is to count the number

[3] This is slightly different from the usual definition, since a path corresponds to two walks in both directions.

of Hamiltonian paths. The following IE-formulation is due to Karp [16]: define the universe U as all walks of length $n-1$ in G, and define A_v as all walks of length $n-1$ that contain node v, for each $v \in V$. With these definitions, the left-hand side of Equation 1, $|\bigcap_{v \in V} A_v|$, is the number of Hamiltonian paths in G. Now it remains to show how to solve the simplified problem: given $R \subseteq V$ and $s \in R$, let $w_k(s, R)$ be the number of walks from s of length k in $G[R]$. Then $w_k(s, R)$ admits the following recurrence:

$$w_k(s,R) = \begin{cases} 1 & \text{if } k = 0 \\ \displaystyle\sum_{t \in N(s) \cap R} w_{k-1}(t,R) & \text{otherwise} \end{cases} \tag{2}$$

Notice that $w_k(s, R)$ is $\mathcal{O}(n^n)$, hence the number of bits needed to represent this value is polynomially bounded, and that

$$|\bigcap_{v \in X} \overline{A_v}| = \sum_{s \in V \setminus X} w_{n-1}(s, V \setminus X)$$

Hence, the simplified problem can be solved in polynomial time using dynamic programming on Equation 2 (the parameter R is fixed but is added for clearness). Thus it takes $\mathcal{O}^*(2^n)$ time and polynomial space to evaluate Equation 1.

2.2 Steiner tree

Now we are ready for our first new result. Assume a graph $G = (V, E)$ and weight function $w : E \to \mathbb{Z}_+$ are given. The Steiner Tree problem is the following: given a set of *terminals* $K \subseteq V$ and an integer c, does there exist a subtree $T = (V', E')$ of G such that $K \subseteq V'$ and $\sum_{e \in E'} w(e) \leq c$. In this section we will give an extension of the results in the previous section to obtain a new IE-formulation for Steiner Tree with *unit weights*, meaning $w(e) = 1$ for every edge $e \in E$. We introduce the following definition:

Definition 2. *A* branching walk *B in $G = (V, E)$ is a pair (T_B, ϕ) where $T_B = (V_B, E_B)$ is an ordered tree and $\phi : V_B \to V$ is a homomorphism from T_B to G. The* length *of B, denoted with $|B|$, is $|E_B|$. For a node $s \in V$, B is* from *s if the root of T_B is mapped to s by ϕ. If a branching walk is said to be* unordered*, T_B is an unordered tree.*

We will use $\phi(V_B) = \{\phi(u) | u \in V_B\}$ and $\phi(E_B) = \{(\phi(u), \phi(v)) | (u, v) \in E_B\}$, hence $\phi(V_B) \subseteq V$ and $\phi(E_B) \subseteq E_B$. A branching walk is a natural generalization of a walk: notice that a branching walk is a walk if T_B is a path. The definition is particularly useful in combination with the following lemma:

Lemma 3. *Let $s \in K$. There exists a subtree $T = (V', E')$ of G such that $K \subseteq V'$ and $|E'| \leq c$ if and only if there exists a branching walk $B = (T_B = (V_B, E_B), \phi)$ from s such that $K \subseteq \phi(V_B)$ and $|B| \leq c$.*

Proof. For the first part, choose $T_B = T$ and $\phi : V_B \to V' = V_B$ to be the identity function. For the second part, we can take T to be a spanning tree of the graph $(\phi(V_B), \phi(E_B))$, and it has the required properties. □

Consider the following IE-formulation: let $s \in K$ be an arbitrarily chosen terminal, and define the universe U as all branching walks from s of length c. For each $v \in K$, define a requirement A_v that consists of all elements of U that contain terminal v (i.e. $v \in \phi(V_B)$). It follows that the left-hand side of Equation 1, $|\bigcap_{v \in X} A_v|$, is the number of branching walks that contain all terminals. Using Lemma 3 this is larger than 0 if and only if the instance of Steiner Tree is a `yes`-instance.

It remains to show how the simplified problem can be solved. For $R \subseteq K$, let $R' = (V \setminus K) \cup R$, and define $b_j^R(s)$ as the number of branching walks from s of length j in $G[R']$, where $s \in R'$. Note that the simplified problem is to compute

$$|\bigcap_{v \in X} \overline{A_v}| = b_c^{K \setminus X}(s)$$

for a given set $X \subseteq K$ of terminals. Now $b_c^R(s)$ can be computed in polynomial time using the following lemma:

Lemma 4. *Let $R \subseteq K$ and $s \in R'$, then*

$$b_j^R(s) = \begin{cases} 1 & \text{if } j = 0 \quad (3a) \\ \sum_{t \in N(s) \cap R'} \sum_{j_1 + j_2 = j-1} b_{j_1}^R(t)\, b_{j_2}^R(s) & \text{otherwise} \quad (3b) \end{cases}$$

Proof. There is one branching walk of length 0, $B = (T_B, \phi)$, from s with T_B being a single node and ϕ mapping this single node to s, hence Case 3a. If the length $j = |E_B|$ is larger than 0, take the first child c_1 of the root of T_B. Notice that $(s, \phi(c_1))$ has to be in E; therefore, $t \in N(s) \cap R'$. Now any tree T_B consists of an edge from the root r to c_1, and two trees rooted at r and c_1. Hence, B also consists a two branching walks, one from t and one from s. The lengths of these branching walks have to sum up to $j - 1$ since the edge (r, c_1) already contributes 1 to the length of B. Now it remains to sum over all possibilities of distributions of the length, and hence Case 3b also holds. □

From Equation 3 it follows that for each $j > 0$ and $s \in R'$, $b_j^R(s)$ is $\mathcal{O}((nj)^j)$. Hence, the number of bits needed to represent $b_j^R(s)$ is polynomially bounded.

Theorem 5. *The* Steiner tree *problem with unit weights can be solved in $\mathcal{O}^*(2^k)$ time and polynomial space, where k is the number of terminals.*

Proof. Due to Lemma 3 the considered IE-formulation solves Steiner Tree, and we can use dynamic programming on Equation 3b to compute the simplified problem in polynomial time. □

The following result is a direct consequence of Theorem 5 and the considerations of Section 4 in [11]:

Corollary 6. Steiner Tree *with unit weights can be solved in $\mathcal{O}^*(1.3533^n)$ time using polynomial space.*

2.3 Further IE-formulations

In this section we give IE-formulations for two problems that lead to algorithms running in $\mathcal{O}^*(2^n)$-time and using polynomial space. In the following assume we are given a graph $G = (V, E)$ and an integer c. In both IE-formulations we define A_v, for each $v \in V$, to be all elements of U that contain node v. The universe U itself will be more tailor-made for both problems.

Degree constrained spanning tree The Degree Constrained Spanning Tree problem, also called Min-Max Degree Spanning Tree, asks whether G has a spanning tree with maximum degree at most c. Define U as all branching walks (T_B, ϕ) of length $n-1$ such that T_B has maximum degree at most c. For $R \subseteq V$, define $d_j^R(g, s)$ as the number of branching walks (T_B, ϕ) from s of length j in $G[R]$ such that the degree of the root of T_B is at most g. The simplified problem is to compute

$$|\bigcap_{v \in X} \overline{A_v}| = \sum_{s \in V \setminus X} d_{n-1}^{V \setminus X}(c, s)$$

and it can be computed in polynomial time with dynamic programming using:

$$d_j^R(g, s) = \begin{cases} [g \geq 0] & \text{if } j = 0 \\ \sum_{t \in N(s) \cap R} \sum_{j_1 + j_2 = j-1} d_{j_1}^R(c-1, t)\, d_{j_2}^R(g-1, s) & \text{otherwise} \end{cases}$$

To see that the equation holds, notice that $d_0^R(g, s) = [g \geq 0]$ by definition and if $j > 0$, we count combinations of two branching walks: in the branching walk from t we are allowed to choose $c-1$ neighbors, and in the one from s we are allowed to use one neighbor less than before.

Max internal spanning tree The Max Internal Spanning Tree asks whether G has a spanning tree with at least c internal nodes (i.e. nodes with degree at least 2). Define the universe U as all branching walks (T_B, ϕ) of length $n-1$ such that T_B has at most $n-(c+1)$ leaves. For $R \subseteq V$, define $m_{g,j}^R(s)$ as the number of branching walks in $G[R]$ of length j from s having at most g leaves.

$$m_{g,j}^R(s) = \begin{cases} [g \leq 1] & \text{if } j = 0 \\ \sum_{t \in N(s) \cap R} \sum_{g_1 + g_2 = g} \sum_{j_1 + j_2 = j-1} m_{g_1, j_1}^R(t)\, m_{g_2, j_2}^R(s) & \text{otherwise} \end{cases}$$

In the equation we count all possible distributions of the length and the number of leaves in the branching walks from s and t. Now the simplified problem is to compute $\sum_{s \in V \setminus X} m_{n-(c+1), j}^{V \setminus X}(s)$. To see that the IE-formulation solves the problem, note there exists $B \in \bigcap_{v \in V} A_v$ if and only if there exists $(T_B, \phi) \in \bigcap_{v \in V} A_v$ such that the root of T_B has degree 1. And in T_B, the number of internal nodes is the number of non-leaves minus 1.

Theorem 7. Degree Constrained Spanning Tree *and* Max Internal Spanning Tree *can be solved in* $\mathcal{O}^*(2^n)$ *time and polynomial space.*

Proof. The discussed IE-formulations solve the problems due to Lemma 3 and the above considerations, and the simplified problems can be solved in polynomial time with dynamic programming on the stated recurrences. □

3 Möbius inversion

In this section we study an algebraic equivalent of Inclusion-Exclusion, called Möbius inversion. Basically, it consists of the following two transforms:

Definition 8. *Given a function* $f : 2^V \to \mathbb{Z}_+$ *and* $Y \subseteq V$*, the* zeta transform $\zeta f(Y)$ *and the* Möbius transform $\mu f(Y)$*, are defined as:*

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

Now the principle of Möbius inversion can be formulated as the following theorem (this is already folklore, but we make the equivalence relation more clear with a proof):

Theorem 9 (Folklore). *The Möbius transform is the inverse of the zeta transform; that is, for every* $Y \subseteq V$*,* $f(Y) = \mu\zeta f(Y)$*.*

Proof. Define U and $A_v \subseteq U$ for $v \in V$ such that for $X \subseteq V$ we have:

$$f(X) = |\bigcap_{v \in X} A_v \setminus \bigcup_{v \in V \setminus X} A_v|$$

This can be done by defining $U = \{e_i^X | X \subseteq V, 1 \leq i \leq f(X)\}$ such that $e_i^X \in A_v$ if and only if $v \in X$. Now $\zeta f(Y) = |\bigcap_{i \in V \setminus Y} \overline{A_i}|$, and hence $\mu\zeta f(Y)$ is equal to the right-hand of Equation 1. Since $f(Y)$ is equal to the left-hand side of Equation 1, the result follows from Theorem 1. □

So intuitively, using the terminology of the previous section, any IE-formulation is equivalent to applying Möbius inversion and the simplified problem is to compute $\zeta f(V \setminus X)$. Now we will reobtain the IE-formulation of Karp [16] discussed in Subsection 2.1 by applying Möbius inversion to the classical dynamic programming approach.

Hamiltonian path revisited Let us again consider #Hamiltonian Path. Let $h(s, R)$ be the number of Hamiltonian paths from s in $G[R \cup s]$. Recall the dynamic programming algorithm of Held and Karp [14]:

$$h(s, R) = \begin{cases} 1 & \text{if } R = \emptyset \\ \sum_{t \in N(s) \cap R} h(t, R \setminus t) & \text{otherwise} \end{cases}$$

We start by adding a parameter k, which is the length of the Hamiltonian paths we are counting. Although this seems superfluous because we know that each Hamiltonian path in $G[R \cup s]$ has length $|R|$, it gives us some needed flexibility.

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

Now $h_{n-1}(s, V \setminus s)$ is the number of Hamiltonian paths from s. Consider the following slightly different function

$$h'_k(s,R) = \begin{cases} [R = \emptyset] & \text{if k=0} \quad (4a) \\ \sum_{t \in N(s) \cap R} h'_{k-1}(t, R \setminus t) + h'_{k-1}(t, R) & \text{otherwise} \quad (4b) \end{cases}$$

Notice that $h'_{|R|}(s,R) = h_{|R|}(s, |R|)$, since the term $h'_{k-1}(t,R)$ added in Case 4b is 0 if $k \leq |R|$. As a next step, we take the zeta transform on both sides of Equation 4. For Case 4a, we have $\zeta h'_0(s,R) = 1$, and for Case 4b:

$$\begin{aligned} \zeta h'_k(s,R) &= \sum_{X \subseteq R} \sum_{t \in N(s) \cap X} h'_{k-1}(t, X \setminus t) + h'_{k-1}(t,X) \\ &= \sum_{t \in N(s) \cap R} \sum_{t \in X \subseteq R} h'_{k-1}(t, X \setminus t) + h'_{k-1}(t,X) \\ &= \sum_{t \in N(s) \cap R} \zeta h'_{k-1}(t,R) \end{aligned}$$

It is immediate that $\zeta h'_k(t,R) = w_k(s,R)$, and we obtained the IE-formulation of Subsection 2.1.

3.1 Subset products

An application for which Möbius inversion is particularly suited is the computation of subset products, introduced by Björklund et al. We will use the following:

Definition 10 ([3]). *Given two functions $f, g : 2^V \to \mathbb{Z}_+$, the* cover product *$(f *_c g)(Y)$, for $Y \subseteq V$ is defined as:*

$$(f *_c g)(Y) = \sum_{A \cup B = Y} f(A) g(B)$$

Assuming f and g can be evaluated in polynomial time, the naive way to compute $(f *_c g)(Y)$ would take $\mathcal{O}^*(3^n)$ time. In [3], Björklund et al. implicitly use the following theorem in order to obtain an $\mathcal{O}^*(2^n)$ algorithm:

Theorem 11 ([3]). *Given two functions $f, g : 2^V \to \mathbb{Z}_+$, the following holds for $Y \subseteq V$:*

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

Proof. Consider the following rewriting:

$$\zeta(f *_c g)(Y) = \sum_{X \subseteq Y} \sum_{A \cup B = X} f(A)g(B) = \Big(\sum_{A \subseteq Y} f(A) \Big) \Big(\sum_{B \subseteq Y} g(B) \Big)$$

The first equality follows by definition. For the second equality notice that for each $A, B \subseteq Y$, there exists exactly one $X \subseteq Y$ such that $A \cup B = X$, hence we can sum over each combination of two subsets A and B. Now the theorem follows from the definition of zeta transform. □

Steiner Tree revisited Let us again consider Steiner Tree. Recall we denote R' for $(V \setminus K) \cup R$. Our starting point is an adjusted version of the famous Dreyfus-Wagner recurrence [3, 9]: for $R \subseteq K$, integer c and $t \in V$ we are going to define $s_c(t, R)$ such that it will be larger than 0 if and only if there exists a subtree $T = (V', E')$ of the graph $G[R' \cup t]$ such that $R \cup t \subseteq V'$ and $\sum_{e \in E'} w(e) \leq c$. For $c \leq 0$, we have $s_c(t, R) = [c = 0 \wedge R = \emptyset]$, and for $c > 0$ define:

$$s_c(t, R) = \sum_{u \in N(t) \cap R'} g_{c-w(t,u)}(t, u, R \setminus u)$$

$$g_c(t, u, R) = \sum_{c_1 + c_2 = c} \sum_{A \cup B = R} s_{c_1}(u, A)\, s_{c_2}(t, B)$$

We use a slightly different variant s'_c of s_c. Define $s'_0(t, R)$ to be $s_0(t, R)$, and for $c > 0$:

$$s'_c(t, R) = \sum_{u \in N(t) \setminus K} g(R) + \sum_{u \in N(t) \cap R} g(R) + g(R \setminus u)$$

where we shorthand $g'_{c-w(t,u)}(t, u, R)$ with $g(R)$, and the definition of g' is obtained by replacing s with s' in the definition of g (hence s' does not depend on s). Note that $s'_c(t, R) > 0$ if and only if $s_c(t, R) > 0$, since $0 \leq g(R) \leq g(R \setminus u)$. Now we take the zeta transform of both s'_c and g'_c:

$$\begin{aligned}
\zeta s'_c(t, R) &= \sum_{X \subseteq R} \Big(\sum_{u \in N(t) \setminus K} g(X) + \sum_{u \in N(t) \cap X} g(X) + g(X \setminus u) \Big) \\
&= \sum_{u \in N(t) \setminus K} \zeta g(R) \; + \sum_{u \in N(t) \cap R} \sum_{u \in X \subseteq R} g(X) + g(X \setminus u) \\
&= \sum_{u \in N(t) \setminus K} \zeta g(R) \; + \sum_{u \in N(t) \cap R} \zeta g(R) \\
&= \sum_{u \in N(t) \cap R'} \zeta g(R)
\end{aligned}$$

$$\begin{aligned}
\zeta g'_c(t, u, R) &= \sum_{X \subseteq R} \sum_{c_1 + c_2 = c} \sum_{A \cup B = X} s'_{c_1}(u, A)\, s'_{c_2}(t, B) \\
&= \sum_{c_1 + c_2 = c} \zeta(s'_{c_1}(u) *_c s'_{c_2}(t))(R) \\
&= \sum_{c_1 + c_2 = c} \zeta s'_{c_1}(u, R)\; \zeta s'_{c_2}(t, R)
\end{aligned}$$

Combining both derivations gives us

$$\zeta s'_c(t,R) = \sum_{u \in N(t) \cap R'} \sum_{c_1+c_2=c-w(t,u)} \zeta s'_{c_1}(u,R)\ \zeta s'_{c_2}(t,R)$$

comparing this with Equation 3, we see that $\zeta s'_c(t,R) = b^R_c(t)$ in the special case of unit weights. And the following result also follows:

Theorem 12. *The* Steiner Tree *problem with bounded integer weights can be solved in $\mathcal{O}^*(2^k)$ and polynomial space.*

3.2 Further applications

In this subsection we give some other applications of the methods considered in the previous subsection, continuing the work of Björklund et al. [3, 4].

Cover Polynomial We use $x^{\underline{i}}$ for the falling factorial $\frac{x!}{(x-i)!}$. A Hamiltonian cycle of a graph is a cyclic walk that contains all nodes exactly once. The cover polynomial of a directed graph $D = (V, A)$ can be defined as (see also [4, 8]):

$$\sum_{i,j} C_V(i,j) x^{\underline{i}} y^{\underline{j}}$$

where $C_V(i,j)$ can be interpreted as the number of ways to partition V into i directed paths and j directed cycles of D. Since paths and cycles with l edges contain $l+1$ and l nodes respectively, the sum of the lengths of the paths and cycles in such a partition will be $n-i$. Moreover, if V is covered, the path and cycles are disjoint because of this size restriction. This allows us to define C_Y using the cover product, such that C_V matches the above interpretation:

$$C_Y(i,j) = \frac{1}{i!j!} \sum_{l_1+\ldots l_{i+j}=n-i} (h_{l_1} *_c \ldots *_c h_{l_i}\ *_c\ c_{l_{i+1}} *_c \ldots *_c c_{l_{i+j}})(Y)$$

where $h_l(Y)$ and $c_l(Y)$ are the number of Hamiltonian paths and Hamiltonian cycles of length l in $D[Y]$, respectively (note that like before we use the redudant parameter l, for obtaining the efficient computable zeta transform). Recall Equation 4 and note we can replace $h_l(Y)$ with $h'_l(Y) = \sum_{s \in V} h'_l(s,Y)$, and $\zeta h'_l(Y)$ is the number of walks of length l in $D[Y]$. We mention that one can in a analogue way replace $c_t(Y)$ with $c'_t(Y)$ such that $\zeta c'_l(Y)$ is the number of cyclic walks of length l in $D[Y]$. Now we apply Theorem 11 on the cover products and obtain:

$$\zeta C_Y(i,j) = \frac{1}{i!j!} \sum_{l_1+\ldots l_{i+j}=n-i} \Big(\prod_{t=1}^{i} \zeta h'_{l_t}(Y)\Big)\ \Big(\prod_{t=i+1}^{i+j} \zeta c'_{l_t}(Y)\Big)$$

which can be computed in polynomial using standard dynamic programming, since $\zeta h'_l(Y) = \sum_{s \in Y} w_k(s,Y)$ and $\zeta c'_l(Y)$ also can.

#c-Spanning forests A c-spanning forest of $G = (V, E)$ is an acyclic subgraph of G with exactly c connected components. Denote $\tau(c)$ for the number of c-spanning forests of G. Assume an ordering $\prec$ on the nodeset V is given. For $Y \subseteq V$, define $\hat{b}_l(Y)$ as the number of unordered branching walks $(T_B = (V_B, E_B), \phi)$ in G such that $Y \subseteq \phi(V_B)$ and $\phi(r)$ is minimum among $\phi(V_B)$, where r is the root of T_B (recall from Subsection 2.2 that $\phi(V_B) = \{\phi(u) | u \in V_B\}$). Now we can write $\tau(c)$ as follows:

Lemma 13.

$$\tau(c) = \frac{1}{c!} \sum_{l_1+\ldots+l_c=n-c} (\hat{b}_{l_1} *_c \ldots *_c \hat{b}_{l_c})(V) \tag{5}$$

Proof. A set of c branching walks of total length $n - c$ can only cover V if it induces a c-spanning forest. Every tree in this spanning forest corresponds to one unordered branching walk from the minimum node it contains. Hence obtain the equality. □

Now we can use Möbius inversion and Theorem 11 on Equation 5 to obtain

$$\tau(c) = \mu \left(\frac{1}{c!} \sum_{l_1+\ldots+l_c=n-c} \prod_{i=1}^{c} (\zeta \hat{b}_{l_1})(V) \right)$$

and it remains to show how to compute $\zeta \hat{b}_{l_1}(R)$ for $R \subseteq V$. For $s \in R$, define $\hat{b}^R_{j,q}(s)$ as the number of unordered branching walks (T_B, ϕ) from s of length j in $G[R]$ such that no child of the root of T_B is mapped to one of the first $q - 1$ neighbors of s in $G[R]$ with respect to the ordering $\prec$. Notice that:

$$\hat{b}^R_{j,q}(s) = \begin{cases} 0 & \text{if } q > |N(s) \cap R| \\ 1 & \text{else if } j = 0 \\ b^R_{j,q+1}(s) + \sum_{j_1+j_2=j-1} \hat{b}^R_{j_1,1}(N^s_q)\ \hat{b}^R_{j_2,q+1}(s) & \text{otherwise} \end{cases}$$

where N^s_q is the q^{th}-first element of the set $N(s) \cap R$ with respect to the ordering $\prec$. Now $\hat{b}_l(R) = \sum_{s \in R} b^{R_s}_{l,1}(s)$, where R_s stands for the set of all elements e in R such that $s \prec e$.

Theorem 14. Cover Polynomial *and* #c-Spanning forests *can be solved in* $\mathcal{O}^*(2^n)$ *time and polynomial space.*

Conclusion

We studied applications where the zeta transform is computable in polynomial time. As mentioned in the introduction, our algorithms considerably improve on dynamic programming in practice: in addition to improving the space requirement, our algorithms can potentially be made faster in practice when combined with techniques from [1, 19]. We want to mention that applying Möbius inversion to a problem is not straightforward: first one has to come up with a function with the wanted properties, in order to succesfully apply Möbius inversion.

To support finding more applications, it is interesting whether more subset products with similar nice properties can be found, for some examples, we also refer to [3].

Acknowledgements The author wants to thank his advisor Pinar Heggernes, Daniel Lokshtanov and the anonymous referees for their valuable support and insightful remarks on this paper.

References

1. E. T. Bax. Recurrence-Based Reductions for Inclusion and Exclusion Algorithms Applied to #P Problems. 1996.
2. A. Björklund and T. Husfeldt. Exact algorithms for exact satisfiability and number of perfect matchings. *Lecture Notes in Computer Science*, 4051:548–559, 2006.
3. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: fast subset convolution. In *STOC*, pages 67–74, 2007.
4. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Computing the tutte polynomial in vertex-exponential time. In *FOCS*, pages 677–686, 2008.
5. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Trimmed moebius inversion and graphs of bounded degree. In *STACS*, pages 85–96, 2008.
6. A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion–exclusion. In *SIAM Journal on Computing, special issue dedicated to selected papers from FOCS 2006*, pages 575–582, 2006.
7. H. L. Bodlaender and D. Kratsch. An exact algorithm for graph coloring with polynomial memory. Technical Report UU-CS-2006-015, Department of Information and Computing Sciences, Utrecht University, 2006.
8. F. R. K. Chung and R. L. Graham. On the cover polynomial of a digraph. *J. Comb. Theory, Ser. B*, 65(2):273–290, 1995.
9. S. Dreyfus and R. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.
10. H. Fernau, D. Raible, S. Gaspers, and A. A. Stepanov. Exact exponential time algorithms for max internal spanning tree. *CoRR*, abs/0811.1875, 2008.
11. F. V. Fomin, F. Grandoni, and D. Kratsch. Faster steiner tree computation in polynomial-space. In *ESA*, pages 430–441, 2008.
12. B. Fuchs, W. Kern, D. Molle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007.
13. S. Gaspers, S. Saurabh, and A. A. Stepanov. A moderately exponential time algorithm for full degree spanning tree. In *TAMC*, pages 479–489, 2008.
14. M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
15. F. Jaeger., D. L. Vertigan, and D. J. A. Welsh. On the computational complexity of the jones and tutte polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society*, 108(01):35–53, 1990.
16. R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1:49–51, 1982.
17. S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the traveling salesman problem. In *ACM '77: Proceedings of the 1977 annual conference*, pages 294–300, New York, NY, USA, 1977. ACM.
18. J. Nederlof. Inclusion exclusion for hard problems. Master's thesis, Utrecht University, August 2008.
19. J. M. M. van Rooij, J. Nederlof, and T. C. van Dijk. Inclusion/exclusion meets measure and conquer: Exact algorithms for counting dominating sets. Technical Report UU-CS-2008-043, Utrecht, The Netherlands, 2008.
20. G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In *IWPEC*, pages 281–290, 2004.