Faster and Space Efficient Exact Exponential Algorithms: Combinatorial and Algebraic Approaches

Dongxiao Yu[1], Yuexuan Wang[2], Qiang-Sheng Hua[2] and Francis C.M. Lau[1]⋆

[1] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong, P.R. China
dxyu@cs.hku.hk, fcmlau@cs.hku.hk

[2] Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, 100084, P.R. China
amywang@mail.tsinghua.edu.cn, qshua@mail.tsinghua.edu.cn

Abstract. Exponential algorithms, whose time complexity is $O(c^n)$ for some constant $c > 1$, are inevitable when exactly solving NP-complete problems unless $\mathbf{P} = \mathbf{NP}$. This chapter presents recently emerged combinatorial and algebraic techniques for designing exact exponential time algorithms. The discussed techniques can be used either to derive faster exact exponential algorithms, or to significantly reduce the space requirements while without increasing the running time. For illustration, exact algorithms arising from the use of these techniques for some optimization and counting problems are given.

1 Introduction

It is a folklore that any NP-complete problem can be exactly solved in exponential time via exhaustive search. Whether there is a faster way than this kind of brute-force approach to solve any such problem is still unclear. Nonetheless, many researchers have found exact exponential time algorithms that are faster than trivial exhaustive search for quite many NP-hard optimization problems such as the traveling salesman problem (TSP) [39]. The study of exact exponential algorithms for NP-hard problems has received increasing attention since the seminal survey by Woeginger [55] in 2001. As a result, some classical techniques such as inclusion-exclusion (e.g. [33, 13, 11]) have been resurrected, and some new techniques, such as fast subset convolution [14] and the algebraic methods (e.g. [10]) have been developed. Unfortunately, for some optimization problems, although it is possible to obtain faster exponential algorithms, exponential space is required, which renders these algorithms not practical for real-life use. This chapter presents some important recently resurrected or emerged combinatorial and algebraic approaches for designing faster exact exponential time algorithms, and discusses how these techniques may be used to reduce the space complexity while not sacrificing the time complexity.

⋆ Corresponding author (fcmlau@cs.hku.hk).

Notation. Throughout this chapter, a modified big-O notation is used to hide a polylogarithmic factor: for a positive real constant d and a function τ, $O^*(\tau)$ denotes a time complexity of the form $O(\tau \log^d \tau)$.

Structure of the chapter. The main theme of this chapter is to introduce two classes of methods for designing either faster or space efficient exact exponential time algorithms. Section 2 discusses the combinatorial methods, which covers fast subset convolution and inclusion-exclusion based algorithms. Section 3 presents the recently emerged algebraic methods, which covers the algebraic sieving method and the polynomial circuit based algebraic method. Each of these two sections will first explain how to use these techniques to design faster exact exponential time algorithms, and then discuss how they may be used to reduce the space complexity while not increasing the running time. Various examples to illustrate the techniques will also be given. Section 4 concludes the chapter and Section 5 highlights the related work on these two classes of methods.

2 Combinatorial Methods

There are two well developed and commonly used combinatorial techniques—fast subset convolution and inclusion-exclusion. The fast subset convolution can compute the convolution of any two given functions over some subset lattice in $O^*(2^n)$ time, whereas a direct evaluation needs $\Omega(3^n)$ time. This fast algorithm is based on fast algorithms for the zeta transform and other related transforms which are actually accelerated dynamic programming algorithms. Thus it needs exponential space since dynamic programming approaches have to store all the useful auxiliary information. Surprisingly, for certain special type of input functions called singletons this exponential space could be circumvented. Furthermore, several variants of subset convolution, such as the covering product and the packing product [14], also play an important role in the design of faster exponential time exact algorithms.

The inclusion-exclusion principle is a classical sieving method: to determine the size of a set, the set is first overcounted, then subtract from this count, add again, subtract again, until finally arriving at the exact number of elements. This allows to count combinatorial objects indirectly, which is better than direct counting which could be inefficient or even impossible. Furthermore, although it needs to go through all subsets, the inclusion-exclusion technique is powerful in providing polynomial space exact algorithms. Combining it with some dynamic programming based algorithms, such as the fast zeta transform, gives more surprising results. In fact, the inclusion-exclusion technique solves optimization problems via solving the corresponding counting problems, and hence it is an important alternative for designing exact algorithms for counting problems.

Before the detailed descriptions of these two techniques, some essential tools needs to be introduced—the fast zeta transform and the fast Möbius transform.

2.1 Zeta Transform and Möbius Transform

Denote by R an arbitrary (possibly noncommutative) ring and by N a set of n elements, $n \geq 0$. Let f be a function that associates with every subset $S \subseteq N$ an element $f(S)$ of the ring R.

The zeta transform of f is the function $f\zeta$ that associates every $S \subseteq N$ with an element in R

$$f\zeta(S) = \sum_{X \subseteq S} f(X). \tag{1}$$

The Möbius transform $f\mu$ of f is defined as:

$$f\mu(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} f(X). \tag{2}$$

Given the zeta transform $f\zeta$, the original function f can be obtained by the following formula which is often called Möbius inversion:

$$f(X) = \sum_{S \subseteq X} (-1)^{|X \setminus S|} f\zeta(S). \tag{3}$$

For more details of these two transforms, please refer to [27].

2.1.1 Fast Zeta Transform and Fast Möbius Transform

The straightforward way to compute the zeta transform evaluates $f\zeta(S)$ at every $S \subseteq N$, using $O(3^n)$ ring additions in total. This can be improved to use only $O(n2^n)$ ring operations by applying the Yates's method [37, 58], and the resulting algorithm is commonly called the fast zeta transform. Without loss of generality, assume that $N = \{1, 2, \ldots, n\}$. To compute $f\zeta$, let initially $f\zeta_0(S) = f(S)$ for every $S \subseteq N$. Then iterate for $j = 1, 2, \ldots, n$ and $S \subseteq N$ with the following recurrence:

$$f\zeta_j(S) = \begin{cases} f\zeta_{j-1}(S), & \textit{if } j \notin X \\ f\zeta_{j-1}(S \setminus \{j\}) + f\zeta_{j-1}(S), & \textit{otherwise.} \end{cases} \tag{4}$$

It can be verified by induction on j that the above recurrence gives $f\zeta_n(S) = f\zeta(S)$ for any $S \subseteq N$ in $O(n2^n)$ ring operations. The Möbius Transform can be computed in a similar manner. Initially, let $f\mu_0(S) = f(S)$ for every $S \subseteq N$. Then iterate for $j = 1, 2, \ldots, n$ and $S \subseteq N$ as follows.

$$f\mu_j(S) = \begin{cases} f\mu_{j-1}(S), & \textit{if } j \notin X \\ -f\mu_{j-1}(S \setminus \{j\}) + f\mu_{j-1}(S), & \textit{otherwise.} \end{cases} \tag{5}$$

Similarly, it can be proved that $f\mu_n(S) = f\mu(S)$ for any $S \subseteq N$.

Theorem 1. *Let N be a set of n elements. Then the zeta transform and the Möbius transform on the subset lattice of N can be computed in $O(n2^n)$ ring operations.*

2.2 Subset Convolution

Definition 1 (Subset convolution). *Given a universe set N of n elements, f and g are functions defined on subsets of N, which associate every $S \subseteq N$ with an element of a ring R, respectively. The convolution $f * g$ for all $S \subseteq N$ is defined as follows.*

$$f * g(S) = \sum_{X \subseteq S} f(X) g(S \setminus X) \tag{6}$$

The subset convolution can yield solutions to many hard computational problems. In particular, by embedding the integer max-sum or min-sum semiring into the sum-product ring, the technique can be used to implement many dynamic programming based algorithms.

2.2.1 Fast Subset Convolution

In principle, the fast subset convolution consists of several efficient dynamic programming instances. In the fast subset convolution, the evaluation of (6) can be achieved via the product of "ranked" extensions of the classical zeta transforms of f and g on the subset lattice, followed by a "ranked" Möbius transform. The fast subset convolution is summarized in Algorithm 1.

For every $k = 0, 1, \ldots, n$ and $S \subseteq N$, the ranked zeta transform of f is defined as

$$f\zeta(k, S) = \sum_{X \subseteq S, |X| = k} f(X). \tag{7}$$

Based on the above defined ranked zeta transform, the inversion can be achieved by

$$f(X) = \sum_{S \subseteq X} (-1)^{|S \setminus X|} f\zeta(|X|, S). \tag{8}$$

Although the above formula seems redundant, it will provide a key for fast evaluation of the subset convolution. Note that the ranked zeta transform can be computed in $O(n^2 2^n)$ ring operations by carrying out the fast zeta transform (4) independently for every $k = 0, 1, \ldots, n$. Similarly, the ranked inversion (8) can be computed in $O(n^2 2^n)$ ring operations by carrying out the fast Möbius transform (5) independently for each $k = 0, 1, \ldots, n$.

For the two ranked zeta transforms, $f\zeta$ and $g\zeta$, define a new convolution $f\zeta \circledast g\zeta$ for all $k = 0, 1, \ldots, n$ and $S \subseteq N$ by

$$f\zeta \circledast g\zeta(k, S) = \sum_{j=0}^{k} f\zeta(j, S) g\zeta(k - j, S). \tag{9}$$

Lemma 1. $f * g(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} f\zeta \circledast g\zeta(|S|, S)$

Proof.

$$\begin{aligned}\sum_{X\subseteq S}(-1)^{|S\setminus X|}f\zeta\circledast g\zeta(|S|,S) &= \sum_{X\subseteq S}(-1)^{|S\setminus X|}\sum_{j=0}^{|S|}f\zeta(j,S)g\zeta(|S|-j,S)\\ &= \sum_{X\subseteq S}(-1)^{|S\setminus X|}\sum_{j=0}^{|S|}\sum_{\substack{U,V\subseteq X\\ |U|=j,|V|=|S|-j}}f(U)g(V).\end{aligned} \tag{10}$$

Because X ranges over all subsets of S, for any ordered pair (U,V) of subsets of S satisfying the condition that $|U|+|V|=|S|$, the term $f(U)g(V)$ occurs exactly once in the sum with sign $(-1)^{|S\setminus X|}$ for subsets X of S iff $U\cup V\subseteq X$. Then putting the terms associated with each pair (U,V) together, by the Binomial Theorem, the coefficient of $f(U)g(V)$ is

$$\sum_{x=|U\cup V|}^{|S|}\binom{|S|-|U\cup V|}{x-|U\cup V|}(-1)^{|S|-x}=\begin{cases}1, & \textit{if } |U\cup V|=|S|\\ 0, & \textit{otherwise.}\end{cases} \tag{11}$$

The conditions that $|U|+|V|=|S|$ and $|U\cup V|=|S|$ imply that $U\cup V=S$ and $U\cap V=\emptyset$; and it follows that

$$\begin{aligned}\sum_{X\subseteq S}(-1)^{|S\setminus X|}f\zeta\circledast g\zeta(|S|,S) &= \sum_{U\cup V=S,U\cap V=\emptyset}f(U)g(V)\\ &= f*g(S).\end{aligned} \tag{12}$$

□

Theorem 2. *The subset convolution over an arbitrary ring can be evaluated in $O(n^2 2^n)$ ring operations using the fast subset convolution as shown in Algorithm 1.*

Proof. The correctness is established by Lemma 1. In Algorithm 1, the time is mainly consumed in lines 2, 6 and 9. By Theorem 1, the time consumed in lines 2 and 9 is $O(n^2 2^n)$, and the time needed in line 6 is $O(n^2 2^n)$ according to Formula (9). Thus the time complexity of Algorithm 1 is $O(n^2 2^n)$. □

The following theorem is implied by Theorem 2.

Theorem 3. *The subset convolution over the integer sum-product ring can be computed in $O^*(2^n \log M)$ time, provided that the range of the input functions is $\{-M,-M+1,\dots,M\}$.*

Proof. By Theorem 2, the subset convolution can be computed in $O(n^2 2^n)$ ring operations. Thus it suffices to note that any intermediate results, for which ring operations are performed, are $O(n\log M)$-bit integers. This is the case because the ranked zeta transform of an input function can be computed with integers

Algorithm 1 Fast Subset Convolution

```
Input: A universe set $N$ of $n$ elements and functions $f, g$ defined on $2^U$
Output: The subset convolution $f * g$ for every set $S \subseteq N$
1: For $i = 0$ to $n$ do
2: Compute the ranked zeta transforms $f\zeta(k, X), g\zeta(k, X)$ for every $X \subseteq N$ using the
   fast zeta transform.
3: End For
4: For $k = 0$ to $n$ do
5:    For every $X \subseteq N$ do
6:    Compute the convolution $f\zeta \circledast g\zeta(k, X)$
7:    End For
8: End For
9: Compute the subset convolution $f * g$ for every set $S \subseteq N$ based on Formula (12)
   using the fast Möbius transform.
```

between $-M2^n$ and $M2^n$, and it follows that the convolution of ranked transforms can be computed with $O(n \log M)$-bit integers. Furthermore, the ranked Möbius tranform is computed by adding and subtracting $O(n \log M)$-bit integers for $O(2^n)$ times. □

An obvious disadvantage of the fast subset convolution algorithm is its inapplicability to semirings where additive inverses need not exist. For some special semirings usually appearing in combinatorial optimization problems, e.g., the integer max-sum and integer min-sum semirings, one can embed these semirings into the integer sum-product ring.

Theorem 4. *The subset convolution over the integer max-sum (min-sum) semiring can be computed in $O^*(2^n M)$ time, provided that the range of the input functions is $\{-M, -M+1, \ldots, M\}$.*

Proof. Here we only provide the proof for the max-sum semiring, as that for the min-sum semiring is similar. Without loss of generality, assume that the range of the input functions is $\{0, 1, \ldots, M\}$; otherwise, the correct output can be obtained by first adding M to each value of both input functions, then computing the convolution, and finally subtracting $2M$ from the computed values.

Let f, g be the two input functions. Let $\beta = 2^n + 1$ and $M^{'} = \beta^M$. Define new functions $f^{'} = \beta^f$ and $g^{'} = \beta^f$ which map the subsets of N to $\{0, 1, \ldots, M^{'}\}$. By Theorem 3, the subset convolution $f^{'} * g^{'}$ over the integer sum-product ring can be computed in $O^*(2^n \log M^{'}) = O^*(2^n M)$ time. It remains to show that for all $S \subseteq N$, the value for $\max_{T \subseteq S}\{f(T) + g(S \setminus T)\}$ can be efficiently deduced given the value of $f^{'} * g^{'}(S)$. Observe that $f^{'} * g^{'}(S)$ can be expressed as

$$\begin{aligned} f^{'} * g^{'}(S) &= \sum_{T \subseteq S} \beta^{f(T)+g(S\setminus T)} \\ &= \alpha_0(S) + \alpha_1(S)\beta + \cdots + \alpha_{2M}(S)\beta^{2M}. \end{aligned} \tag{13}$$

Due to the choice of β, each coefficient $\alpha_r(S)$ is uniquely determined and equals the number of subsets T of S for which $f(T) + g(S \setminus T) = r$. Thus, for each $S \subseteq N$, the largest r for which $\alpha_r(S) > 0$ corresponds to the value of $f * g(S)$. Clearly, this can be found in $O(M)$ time. □

Example 1 (Partitioning Problem). The generic problem of partitioning is to divide an n-element set N into k disjoint subsets such that each of which satisfies some desired property specified by an indicator function f on the subsets of N. Given N, k, and f as input, the task is to decide whether there exists a partition $\{S_1, S_2, \dots, S_k\}$ of N such that $f(S_c) = 1$ for each $c = 1, 2, \dots, k$. Many classical graph partitioning problems are of this form. For example, in graph coloring, $f(S) = 1$ iff S is an independent set in the input graph with the vertices N. Likewise, in domatic partitioning, f is the indicator of dominating sets. Note that the number of valid partitions of N is given by $f^{*k}(N)$, where f^{*k} is defined as

$$f^{*k} = \underbrace{f * \dots * f}_{k\ times}$$

Thus the valid partitions can be counted by $O(\log k)$ subset convolutions using the doubling trick—computing only convolutions f^{*2^i}. □

Example 2 (Steiner Tree Problem).

Given an undirected graph $G = (V, E)$, a weight $w(e) > 0$ for each edge $e \in E$, and a set of vertices $K \subseteq V$, the Steiner Tree Problem is to find a subgraph H of G that connects the vertices in K and has the minimum total weight $\sum_{e \in E(H)} w(e)$ among all such subgraphs of G. Here we only consider the case where the weight of each edge is bounded by a constant. Obviously, H is necessarily a tree. A Steiner tree always refers to such an optimal subgraph. The fast subset convolution can be used to accelerate the famous Dreyfus-Wagner algorithm [25].

For a vertex subset $Y \subseteq V$, denote the total weight of a Steiner tree connecting Y in G by $W(Y)$. The Dreyfus-Wagner algorithm is based on the following dynamic programming called Dreyfus-Wagner recurrence: for $|Y| \geq 3$, and all $q \in Y$, $X = Y \setminus \{q\}$.

$$W(\{q\} \cup X) = \min\{W(\{p, q\}) + g_p(X) : p \in V\}, \tag{14}$$

$$g_p(X) = \min\{W(\{p\} \cup D) + W(\{p\} \cup (X \setminus D)) : \emptyset \subset D \subset X\}. \tag{15}$$

The Steiner tree problem can be solved by computing the weight $W(K)$ via the above recursion. For the base case, observe that for $|Y| = 1$ the weight $W(Y) = 0$ and for $|Y| = 2$ the weight $W(Y)$ can be determined by a shortest-path computation based on the edge weight $w(e)$, e.g., Johnson's algorithm [35]. A bottom-up evaluation of $W(K)$ takes $O^*(3^k n + 2^k n^2 + nm)$ time, where $n = |V|$, $m = |E|$ and $k = |K|$. Once the values $W(\{p\} \cup Y)$ and $g_p(Y)$ for all $Y \subset K$ and $p \in V$ are computed and stored, an actual Steiner tree is easy to construct by tracing backwards a path of optimal choices in (14) and (15).

The fast subset convolution over the min-sum semiring can expedite the evaluation of the Dreyfus-Wagner recursion in (15). Here the fast subset convolution needs to be implemented in a level-wise manner. For each level $l = 2, 3, \dots, k-1$, assume the value $W(\{q\} \cup X)$ has been computed and stored for all $X \subset K$ and $q \in V \setminus X$. Define the function f_p for all $X \subseteq K$ and $p \in V$ as

$$f_p(X) = \begin{cases} W(\{p\} \cup X), & if\ 1 \leq |X| \leq l-1 \\ \infty, & otherwise. \end{cases} \tag{16}$$

Obviously, $g_p(X) = f_p * f_p(\mathrm{X})$ for all $X \subseteq K$ and $|X| = l$. Then applying the fast subset convolution over the min-sum semiring, by Theorem 4, the Steiner tree problem can be solved in $O^*(2^k n^2 + nm)$ time. □

2.3 Variants of Subset Convolution

2.3.1 Covering Product and Intersecting Covering Product

Definition 2. *Given two functions f, g defined on 2^N which associate each subset of N to an element of a ring R, the covering product $f *_c g(S)$ for each $S \subseteq N$ is defined as:*

$$f *_c g(S) = \sum_{U,V \subseteq S, U \cup V = S} f(U)g(V) \tag{17}$$

Theorem 5. *The covering product over an arbitrary ring can be evaluated in $O(n^2 2^n)$ ring operations.*

Proof. The proof is mainly based on the following Lemma.

Lemma 2. *For any $S \subseteq N$, the following holds:*

$$(f *_c g)\zeta(S) = f\zeta(S) \cdot g\zeta(S) \tag{18}$$

Proof.

$$\begin{aligned} (f *_c g)\zeta(S) &= \sum_{X \subseteq S} (f *_c g)(S) \\ &= \sum_{X \subseteq S} \sum_{U \cup V = X} f(U)g(V) \\ &= \sum_{U \subseteq S} f(U) \sum_{V \subseteq V} g(V) \\ &= f\zeta(S) \cdot g\zeta(S). \end{aligned} \tag{19}$$

The third equality comes from the fact that for each $U, V \subseteq X$, there exists exactly one $X \subseteq S$ such that $U \cup V = X$. □

By Lemma 2, $f *_c g$ can be obtained by computing the Möbius transform of $f\zeta(S) \cdot g\zeta(S)$. Then an $O^*(2^n)$ time algorithm for the covering product is as follows: first compute $f\zeta$ and $g\zeta$ using the fast zeta transform (4), then multiply $f\zeta$ by $g\zeta$ for all $S \subseteq N$, and finally compute $f *_c g$ using the fast Möbius transform (5). □

Definition 3. *Given two functions f, g defined on 2^N which associate each subset of N to an element of a ring R, the intersecting covering product $f *_{ic} g(S)$ for each $S \subseteq N$ is defined as:*

$$f *_{ic} g(S) = \sum_{\substack{U,V \subseteq S \\ U \cup V = S, U \cap V \neq \emptyset}} f(U)g(V) \tag{20}$$

An $O^*(2^n)$ time algorithm can be immediately derived from the fact that $f *_{ic} g = f *_c g - f * g$.

Theorem 6. *The intersecting covering product over an arbitrary ring can be evaluated in $O(n^2 2^n)$ ring operations.*

Furthermore, more precise control over the allowed intersection cardinalities $|U \cap V| = l$ besides the $l = 0$ $(f * g)$ and $l > 0$ $(f *_{ic} g)$ cases can be obtained by modifying (9).

Example 3 (Minimum Connected Spanning SubHypergraph (MCSH)). A hypergraph is a pair $H = (V, \mathcal{E})$, where V is a finite set and $\mathcal{E}$ is a set consisting of subsets of V. A hypergraph $J = (W, F)$ is a subhypergraph of H if $W \subseteq V$ and $F \subseteq \mathcal{E}$. A subhypergraph is spanning if $V = W$. A path in a hypergraph H is a sequence $(x_1, E_1, x_2, E_2, \ldots, E_l, x_{l+1})$ such that (a) $x_1, x_2, \ldots, x_{l+1} \in V$ are all distinct, (b) $E_1, E_2, \ldots, E_l \in \mathcal{E}$ are all distinct, and (c) $x_i, x_{i+1} \in E_i$ for all $i = 1, 2, \ldots, l$. Such a path joins x_1 to x_{l+1}. A hypergraph is connected if for all distinct $x, y \in V$ there exists a path joining x to y.

Given a hypergraph $H = (V, \mathcal{E})$ and a weight $w(E) > 0$ for each hyperedge $E \in \mathcal{E}$, the minimum connected spanning subhypergraph problem is to find a connected spanning subhypergraph of H that has the minimum total weight, or to assert that none exists. Here the weight of every edge is assumed to be bounded by a constant. The MCSH problem can be solved in $O^*(2^n)$ time using the intersecting covering product over the min-sum semiring, where $n = |V|$.

First define the k-th power of the intersecting covering product for all $k = 2, 3, \ldots$ by

$$f^{*_{ic}^k} = f *_{ic} (f^{*_{ic}^{k-1}}). \tag{21}$$

Here the order in which the products are evaluated matters because the intersecting covering product is not associative. Define the function f for all $E \subseteq V$ by

$$f(E) = \begin{cases} w(E), & if\ E \in \mathcal{E} \\ \infty, & otherwise. \end{cases} \tag{22}$$

Now observe that (a) an MCSH can be constructed by augmenting a connected subhypergraph of H one hyperedge at a time, and (b) at most $n-1$ hyperedges occur in an MCSH of H. Thus $f{*}_{ic}^{k}(V) < \infty$ is the minimum weight of a connected spanning subhypergraph of H consisting of k hyperedges. By storing the function values $f{*}_{ic}^{k}$ for each $k = 1, 2, \ldots, n-1$, the actual MCSH can be determined by tracing back the computation one edge at a time.

2.3.2 Other Variants

Another important variant of the subset convolution is the packing product which is defined for all $S \subseteq N$ as

$$f *_p g(S) = \sum_{U,V \subseteq S, U \cap V = \emptyset} f(U)g(V). \tag{23}$$

Given f and g, the packing product can be evaluated in $O(n^2 2^n)$ ring operations by first computing the subset convolution $f * g$ and then convolving the result with vector $\overrightarrow{1}$ with all entries being equal to 1 based on the following fact:

$$f *_p g = f * g * \overrightarrow{1}. \tag{24}$$

Some further variations are possible by restricting the domain, e.g., to any hereditary family of subsets of N. For more details, please refer to [14].

2.4 Inclusion-Exclusion

The inclusion-exclusion technique is based on a fundamental counting principle—the inclusion-exclusion principle. Clearly, it is a natural approach for solving counting problems. Here we mainly focus on demonstrating its power in solving decision versions of optimization problems when direct computation is inefficient or even impossible. Another important feature of the inclusion-exclusion technique is that it does not need exponential space in principle. Hence, this technique is also a good choice for deriving polynomial space algorithms.

Lemma 3 (Inclusion-exclusion principle). *Let B be a finite set with subsets $A_1, \ldots, A_n \subseteq B$. With the convention that $\cap_{i \in \emptyset} A_i = B$, the following holds:*

1. The number of elements of B which lie in none of the A_i is

$$|\bigcap_{i=1}^{n} \overline{A_i}| = \sum_{X \subseteq \{1,\ldots,n\}} (-1)^{|X|} |\bigcap_{i \in X} A_i|. \tag{25}$$

2. Let $w : B \to R$ be a real-valued weight function extended to the domain 2^B by setting $w(A) = \sum_{e \in A} w(e)$. Then

$$w(|\bigcap_{i=1}^{n} \overline{A_i}|) = \sum_{X \subseteq \{1,\ldots,n\}} (-1)^{|X|} w(|\bigcap_{i \in X} A_i|). \tag{26}$$

Proof. The Lemma is proved by analyzing the contribution of every element $e \in B$ to both sides of the expression. If e is not contained by any A_i, it contributes $w(e)$ to the left hand side. To the right hand side it contributes $w(e)$ exactly once, namely in the term corresponding to $X = \emptyset$. Conversely, assume that e lies in A_i for all $i \in I \neq \emptyset$. Its contribution to the left hand side is 0. On the right hand side, since e lies in the intersection $\cap_{i \in X} A_i$ for every $X \subseteq I$, by the Binomial Theorem, its total contribution is

$$\sum_{X \subseteq I} (-1)^{|X|} w(e) = w(e) \sum_{i=0}^{|I|} (-1)^i \binom{|I|}{i} = 0. \tag{27}$$

□

Example 4 (Set Partition). Given a set system $(N, \mathcal{F})$ and an integer k, where $\mathcal{F} \subseteq 2^N$, a k-partition of the given set system is a tuple $(S_1, \ldots, S_k)$ over $\mathcal{F}$ such that $\cup_{i=1}^k S_i = N$ and $S_i \cap S_j = \emptyset$ for any $1 \leq i \neq j \leq k$. Denote $P_k(\mathcal{F})$ as the number of such k-partitions. The set partition problem is to determine the minimum k such that there is a k-partition of $(N, \mathcal{F})$, i.e., the minimum k such that $P_k(\mathcal{F}) > 0$. Clearly, this problem can be solved by computing $P_k(\mathcal{F})$ for at most n different k values. The following lemma shows how to compute $P_k(\mathcal{F})$ by making use of the inclusion-exclusion principle.

Lemma 4. *Let $a_k(X)$ denote the number of k-tuples $(S_1, \ldots, S_k)$ over $\mathcal{F}$ for which $S_i \cap X = \emptyset (1 \leq i \leq k)$ and $\sum_{i=1}^k |S_i| = n$. Then*

$$p_k(\mathcal{F}) = \sum_{X \subseteq N} (-1)^{|X|} a_k(X). \tag{28}$$

Proof. This is a direct application of Lemma 3, where B is the set of k-tuples $(S_1, \ldots, S_k)$ over $\mathcal{F}$ satisfying $\sum_{i=1}^k |S_i| = n$, and $A_i \subseteq B$ are those k-tuples that avoid $\{i\}$. Then $p_k(F)$ is exactly the number of k-tuples that lie in none of the A_i, and $|\cap_{i \in X} A_i| = a_k(X)$. □

Write $f\zeta^{(l)}(Y)$ for the number of sets $S \in F$ with $|S| = l$ and $S \subseteq Y$, and recall that it is the zeta transform of the indicator function

$$f^{(l)}(S) = \begin{cases} 1, & if\ S \in \mathcal{F}, |S| = l \\ 0, & otherwise. \end{cases} \tag{29}$$

Thus, the fast zeta transform (4) computes a table containing $f\zeta^{(l)}(Y)$ for all l and Y in $O^*(2^n)$ time.

Once these values have been computed, $a_k(X)$ can be obtained for any fixed $X \subseteq N$ by dynamic programming in time polynomial in k and n as follows. Define $g(j, m)$ to be the number of j-tuples $(S_1, \ldots, S_j)$ for which $S_c \cap X = \emptyset (1 \leq c \leq j)$ and $\sum_{c=1}^j |S_j| = m$, formally

$$g(j, m) = \sum_{l_1 + \cdots + l_j = m} \prod_{c=1}^j f\zeta^{(l_c)}(N \setminus X). \tag{30}$$

Then $a_k(X) = g(k, n)$, and it can be computed by the following recursion:

$$g(j, m) = \sum_{l=0}^{m} g(j-1, m-l) f\zeta^{(l)}(N \setminus X), \tag{31}$$

observing that $g(1, m) = f\zeta^{(m)}(N \setminus X)$. Finally, summing up $a_k(X)$ according to (28) gives an $O^*(2^n)$ time algorithm for computing $P_k(\mathcal{F})$. □

Example 5. (Cover Polynomial) Denote $x^{\underline{i}} = \frac{x!}{(x-i)!}$. A Hamiltonian path of a graph is a walk that contains all the nodes exactly once. If the walk is cyclic, it is called a Hamiltonian cycle. The cover polynomial of a directed graph $D = (V, A)$ is defined as (c.f. [15][22])

$$\sum_{i,j} C_V(i,j) x^{\underline{i}} y^{\underline{i}}. \tag{32}$$

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. The so called computing cover polynomial is to compute the coefficient $C_V(i, j)$.

For $X \subseteq V$, denote by $h(X)$ the number of Hamiltonian paths in $D[X]$, and denote by $c(X)$ the number of Hamiltonian cycles in $D[X]$. Define $h(\emptyset) = c(\emptyset) = 0$. Note that for all $x \in V$, $h(\{x\}) = 1$ and $c(\{x\})$ is the number of loops incident on x. Then $C_V(i, j)$ can be expressed as:

$$C_V(i,j) = \frac{1}{i!j!} \sum_{X_1,\ldots,X_i,Y_1,\ldots,Y_j} h(X_1)\cdots h(X_i)c(Y_1)\cdots c(Y_j), \tag{33}$$

where the sum is over all $(i + j)$-tuples $(X_1, \ldots, X_i, Y_1, \ldots, Y_j)$ such that it is a partition of V.

The expression for $C_V(i, j)$ can be reformulated using the principle of inclusion-exclusion. Let z be a polynomial indeterminate. Define for every $U \subseteq V$ the polynomials

$$H_U(z) = \sum_{X \subseteq U} h(X) z^{|X|}, C_U(z) = \sum_{Y \subseteq U} c(Y) z^{|Y|}. \tag{34}$$

Viewed as set functions, $H_U(z)$ and $C_U(z)$ are zeta transforms of the set functions $h(X)z^{|X|}$ and $c(Y)z^{|Y|}$, respectively. By inclusion-exclusion,

$$C_V(i,j) = \frac{1}{i!j!} \sum_{U \subseteq V} (-1)^{|V \setminus U|} \{z^n\} H_U(z)^i C_U(z)^j. \tag{35}$$

Based on the above formula, an $O^*(2^n)$ time algorithm can be obtained. For $S \subseteq V$, let $w_S(s, t, l)$ denote the number of directed walks of length l from vertex s to vertex t in $D[S]$. Define $w_S(s, t, l) = 0$ if $s \notin S$ or $t \notin S$. By inclusion-exclusion again,

$$\begin{aligned} h(X) &= \sum_{s,t \in V} \sum_{S \subseteq X} (-1)^{|X \setminus S|} w_S(s,t,|X|-1), \\ c(Y) &= \frac{1}{|Y|} \sum_{s \in V} \sum_{S \subseteq Y} (-1)^{|Y \setminus S|} w_S(s,s,|X|). \end{aligned} \tag{36}$$

Observing $w_S(s,t,l)$ can be computed in polynomial time, the fast Möbius transform (5) can compute $h(X)$ and $c(Y)$ for every $X, Y \subseteq V$ in $O^*(2^n)$ time. Then H and C can be evaluated using the fast zeta transform according to (34). Finally, given H and C, (35) can be evaluated in $O^*(2^n)$ time. □

2.5 Space Efficient Exact Algorithms

2.5.1 Polynomial Space Algorithm Based On Subset Convolution

For some special input functions whose zeta transforms can be determined easily, polynomial space algorithms for some combinatorial optimization problems can be derived via the subset convolution and the covering product. A particular type of input functions called singletons is used here to demonstrate how to efficiently compute the subset convolution and the covering product values for the element set N using only polynomial space. A function $f : U \to R$ is called a singleton if there exists an element $e \in U$ such that $f(x) = 0$ unless $x = e$.

Theorem 7. *Given two functions f, g defined on 2^N which associate each subset of N to an element of a ring R, if f, g are singletons, then (1) $f *_c g(N)$ can be computed in $O^*(2^n)$ time and polynomial space, and (2) $f * g(N)$ can be computed in $O^*(2^n)$ time and polynomial space.*

Proof. (1) Assume that $f(X_f) = e_f$ and $g(X_g) = e_g$, where $X_f, X_g \subseteq N$ and $e_f, e_g \in R$. Since, f, g are singletons, $f(X) = 0$ for any $X \subseteq N$ other than X_f. The same result is also true for g and any subsets other than X_g. The following Lemma is proved in Section 2.3.1.

Lemma 5. *For any $S \subseteq N$, the following holds:*

$$(f *_c g)\zeta(S) = f\zeta(S) \cdot g\zeta(S) \tag{37}$$

Note that for singletons f, g, the zeta transforms for every $Y \subseteq N$ can be easily given as

$$f\zeta(Y) = \begin{cases} e_f, & if\ X_f \subseteq Y \\ 0, & if\ otherwise. \end{cases} \tag{38}$$

$$g\zeta(Y) = \begin{cases} e_g, & if\ X_g \subseteq Y \\ 0, & if\ otherwise. \end{cases} \tag{39}$$

Thus the zeta transform of $f *_c g$ for every subset $Y \subseteq N$ can be computed in polynomial time after getting the zeta transforms $f\zeta$ and $g\zeta$. Then $f *_c g(N)$ can be obtained in $O^*(2^n)$ time by computing the Möbius inversion (3). The space used is polynomial since it is not necessary to store $(f *_c g)\zeta$ values for every $Y \subseteq N$.

(2) Denote $h = f * g$. Define a relaxation of a function $f : 2^N \to R$ as a sequence of functions $f_i : 2^N \to R$, for $0 \leq i \leq |N|$, such that for every $0 \leq i \leq |N|, Y \subseteq N$:

$$f_i(Y) = \begin{cases} f(Y), & if\ i = |Y| \\ 0, & if\ i < |Y|. \end{cases} \tag{40}$$

Observe that a function f can have many different relaxations, since there are no restrictions on what $f_i(X)$ should be when $i > |X|$. The basic idea of the proof is that for the input functions f and g, there exists a relaxation $\{h_i\}$ of h such that the zeta transform of $h_{|N|}$ can be computed efficiently. Since $h(N) = h_{|N|}(N)$, $h(N)$ can be obtained by computing the Möbius inversion (3) of $h\zeta_{|N|}$.

For input functions f and g, define $f_i = f$ and $g_i = g$ for all i. It is easy to verify that $\{f_i\}$ and $\{g_i\}$ are relaxations of f and g, respectively. Then define for all i, $h_i = \sum_{j=0}^{i} f_j *_c g_{i-j}$.

Lemma 6. *As defined above, $\{h_i\}$ is a relaxation of h.*

Proof. By the definition of the covering product, for every $X \subseteq N$

$$h_i(X) = \sum_{j=0}^{i} \sum_{Y \cup Z = X} f_j(Y) g_{i-j}(Z). \tag{41}$$

Consider an arbitrary summand $f_j(Y)g_{i-j}(Z)$. There are two cases:

Case 1 ($|X| > i$). Since $Y \cup Z = X$, $|Y| + |Z| \geq |X| > i$. It concludes that either $|Y| > j$ and $f_j(Y) = 0$, or $|Z| > i - j$ and $g_{i-j}(Z) = 0$, because $\{f_i\}$ and $\{g_i\}$ both are relaxations of f and g respectively. Thus $f_j(Y)g_{i-j}(Z) = 0$. Since $f_j(Y)g_{i-j}(Z)$ is an arbitrary term in the summation for $h_i(X)$, it follows that $h_i(X) = 0$.

Case 2 ($|X| = i$). If $|Y| + |Z| > i$, either $|Y| > j$ or $|Z| > i-j$ and $f_j(Y)g_{i-j}(Z) = 0$ which is analogous to the first case. If $|Y| + |Z| = i$, then Y and Z are disjoint. Since $\{f_i\}$ and $\{g_{i-j}\}$ are relaxations of f and g respectively, only $f_{|Y|}(Y)g_{|Z|}(Z)$

will contribute to the sum. Thus,

$$\begin{aligned} h_i(X) &= \sum_{j=0}^{i} \sum_{Y \cup Z = X} f_j(Y) g_{i-j}(Z) \\ &= \sum_{Y \cup Z = X, Y \cap Z = \emptyset} f_{|Y|}(Y) g_{|Z|}(Z) \\ &= \sum_{Y \cup Z = X, Y \cap Z = \emptyset} f(Y) g(Z) \\ &= h(X). \end{aligned} \tag{42}$$

□

By Lemma 5 and the definition of h_i in (41), it follows that $h\zeta_i = \sum_{j=0}^{i} f\zeta_j g\zeta_{i-j}$. Thus the zeta transform of $h_{|N|}(X)$ for every $X \subseteq N$ can be computed in polynomial time given the zeta transforms for $\{f_i\}$ and $\{g_i\}$. From (1), the zeta transforms of $\{f_i\}$ and $\{g_i\}$ can be obtained efficiently. Similar to what is done in (1), $h(N) = h_{|N|}(N)$ can be obtained in $O^*(2^n)$ time and polynomial space by computing the Möbius inversion

$$h_{|N|}(N) = \sum_{X \subseteq N} (-1)^{|N \setminus X|} h\zeta_{|N|}(X). \tag{43}$$

Example 6 (Cover Polynomial Revisited). Define

$$C_Y(i,j) = \frac{1}{i!j!} \sum_{l_1 + \cdots + l_{i+j} = n-i} h_{l_1} *_c \cdots h_{l_i} *_c c_{l_{i+1}} \cdots c_{l_{i+j}}(Y), \tag{44}$$

where $h_l(Y)$ and $c_l(Y)$ are the number of Hamiltonian paths and Hamiltonian cycles of length l in $D[Y]$, respectively (here the parameter l is introduced in order to obtain efficient computable zeta transforms). Note that paths and cycles with l edges contain $l+1$ and l nodes respectively, which follows that the sum of the lengths of the paths and cycles in such a partition will be $n-i$. Moreover, if V is covered, the paths and cycles are disjoint because of the size restriction. The above definition for $C_V(i,j)$ is equivalent to that in Example 5. Note that $h_l(Y)$ can be replaced by $h_l^{'}(Y)$ which denotes the number of walks of length l in $D[Y]$. Similarly, $c_l(Y)$ can be replaced by $c_l^{'}(Y)$ which is the number of cyclic walks of length l. Using the technique introduced in Theorem 7, $C\zeta_Y(i,j)$ can be computed using the following formula given $h\zeta_l^{'}(Y)$ and $c\zeta_l^{'}(Y)$,

$$C\zeta_Y(i,j) = \frac{1}{i!j!} \sum_{l_1 + \cdots + l_{i+j} = n-i} \prod_{t=1}^{i} h\zeta_{l_t}^{'}(Y) \prod_{p=1}^{j} c\zeta_{l_{i+p}}^{'}(Y). \tag{45}$$

Finally $h\zeta_l^{'}(Y)$ and $c\zeta_l^{'}(Y)$ can be computed in polynomial time using standard dynamic programming (c.f. [44]). Then $C_V(i,j)$ can be obtained by computing the Möbius inversion as described in Theorem 7 (by a similar argument as used in proving Lemma 1, it can be proved that all extra items introduced in the process of relaxing h_l and c_l are canceled through computing the Möbius inversion). Putting everything together will provide an $O^*(2^n)$ time and polynomial space algorithm for counting cover polynomial. □

Combining with the algebraic method to be introduced in Section 3.2, the technique introduced in Theorem 7 can be used to give polynomial space algorithms for the traveling salesman problem, the Steiner tree problem and the weighted set cover problem etc. without increasing the running times as have been derived in the fastest solutions to these problems [36][14][28][13].

2.5.2 Polynomial Space Exact Algorithms based on Inclusion-Exclusion

By the inclusion-exclusion principle (25), if $|\bigcap_{i\in X} A_i|$ can be computed in polynomial time for each $X \subseteq \{1,\dots,n\}$, there exists a polynomial-space algorithm evaluating Equation (25) in $O^*(2^n)$ time.

Example 7 (Counting Hamiltonian Paths). Given a graph $G = (V, E)$, a Hamiltonian path is a walk that contains each node exactly once. The Counting Hamiltonian Path problem is to count the number of Hamiltonian paths. The following inclusion-exclusion based algorithm is due to Karp [36]: define the universe B as all walks of length $n-1$ in G, and define A_v as all walks of length $n-1$ containing v for each $v \in V$. By the inclusion-exclusion principle, the number of Hamiltonian paths is

$$h = \sum_{X\subseteq V} (-1)^{|X|} |\bigcap_{v\in X} \overline{A_v}|. \tag{46}$$

For $X \subseteq V$ and $s \in X$, let $w_k(s, X)$ be the number of walks from s of length k in $G[X]$. Then

$$|\bigcap_{v\in X} \overline{A_v}| = \sum_{s\in V\setminus X} w_{n-1}(s, V \setminus X). \tag{47}$$

For fixed $X \subseteq V$, $w_k(s, X)$ can be computed in polynomial time using the following recurrence.

$$w_k(s, X) = \begin{cases} 1, & if\ k = 0 \\ \sum_{t\in N(s)\cap X} w_{k-1}(t, X), & otherwise. \end{cases} \tag{48}$$

Notice that $w_k(s, X)$ is at most $O(n^n)$ which needs polynomial bits to represent. Thus $|\bigcap_{v\in X} \overline{A_v}|$ can be computed in polynomial space and time, which provides an $O^*(2^n)$ time and polynomial space algorithm to evaluate Equation (46).

Example 8 (Counting Perfect Matchings). Given two undirected graphs F and G, let $sub(F, G)$ denote the number of distinct copies of a graph F contained in a graph G. Clearly, if F is a matching of $n/2$ edges, where n is the vertex number of G, then $sub(F, G)$ is the number of perfect matchings in G.

A homomorphism from F to G is a mapping from the vertex set of F to that of G such that the image of every edge of F is an edge of G. Let $hom(F,G)$ and $inj(F,G)$ be the numbers of homomorphisms and injective homomorphisms from F to G respectively. Furthermore, let $aut(F,F)$ denote the number of automorphisms, i.e., bijective homomorphisms, from F to itself. The following equation switches the focus to computing $inj(F,G)$, since $aut(F,F)$ for a graph F on n_F vertices can be computed in subexponential time [2].

$$sub(F,G) = \frac{inj(F,G)}{aut(F,F)} \tag{49}$$

Let S be a given subset of $V(G)$, then a homomorphism f from F to G is called S-saturating if (a) $S \subseteq f(V(F))$ and (b) for all $v \in S$, $|f^{-1}(v)| = 1$. Let $S - hom(F,G)$ denote the number of S-saturating homomorphisms. By the inclusion-exclusion principle,

$$inj(F,G) = \sum_{W \in V(G)\setminus S} (-1)^{|W|} S - hom(F, G[V(G) \setminus W]). \tag{50}$$

Note that if F is a matching of $n/2$ edges, $S - hom(F, G[V(G) \setminus W])$ can be computed in polynomial time and polynomial space using the following equation, which gives rise to an $O^*(2^n)$ time polynomial space algorithm for counting the number of perfect matchings in G. Let $S = \{v_1, \dots, v_a\}$, then

$$S{-}hom(F, G[V(G)\setminus W]) = \binom{\frac{n}{2}}{a} a!(\prod_{i=1}^{a}(2deg_{V(G)\setminus W}(v_i))|2E(G[V(G)\setminus(W \cup S)])|^{\frac{n}{2}-a}, \tag{51}$$

where $deg_{V(G)\setminus W}(v_i)$ is the number of vertices adjacent to v_i in $V(G) \setminus W$.

2.5.3 Linear Space Exact Algorithms based on Linear Space Fast Zeta Transform

As shown before, inclusion-exclusion together with the fast zeta transform can provide efficient exact algorithms for many hard problems, e.g., the covering problem, the partitioning problem and the packing problem (also c.f. [13, 11, 40]). However, all these algorithms need $O^*(2^n)$ space to carry out the fast zeta transform. The following linear space fast zeta transform (Algorithm 2) can help these algorithms achieve a linear space bound without increasing the running time.

Theorem 8 (Linear Space Zeta Transform). *Suppose f vanishes outside a family $\mathcal{F}$ of subsets of N and the member of $\mathcal{F}$ can be listed in time $O^*(2^n)$ and space $O^*(|\mathcal{F}|)$. Then the values $f\zeta(X)$, for every $X \subseteq U$, can be listed in time $O^*(2^n)$ and space $O^*(|\mathcal{F}|)$ using Algorithm 2.*

Proof. Partition U into U_1 and U_2 such that $|U_1| = n_1$, $|U_2| = n_2$, where $n_2 = \lceil \log |\mathcal{F}| \rceil$ and $n_1 = n - n_2$. The correctness of Algorithm 2 is established by the following Lemma.

Lemma 7.

$$f\zeta(X_1 \cup X_2) = h(X_2) \tag{52}$$

Proof.

$$\begin{aligned} h(X_2) &= \sum_{Y_2 \subseteq X_2} g(Y_2) \\ &= \sum_{Y_2 \subseteq X_2} \sum_{Y \in \mathcal{F}} [Y \cap U_1 \subseteq X_1][Y \cap U_2 = Y_2] f(Y) \\ &= \sum_{Y \in \mathcal{F}} [Y \cap U_1 \subseteq X_1][Y \cap U_2 \subseteq X_2] f(Y) \\ &= \sum_{Y \in \mathcal{F}, Y \subseteq X_1 \cup X_2} f(Y) \\ &= \sum_{Y \subseteq X_1 \cup X_2} f(Y) \\ &= f\zeta(X_1 \cup X_2). \end{aligned} \tag{53}$$

□

Observe that the algorithm runs in $O^*(2^{n_1}(2^{n_2} + |\mathcal{F}|))$ time and $O^*(2^{n_2})$ space. By the assigned values of n_1 and n_2, the algorithm thus runs in $O^*(2^n)$ time and $O^*(|\mathcal{F}|)$ space.

□

Algorithm 2 Linear Space Algorithm for Zeta Transform

```
Input: A universe set $N$ of $n$ elements and a functions $f$ defined on a family $\mathcal{F}$ of
        subsets of $N$
Output: The zeta transform $f\zeta$ for all sets in $2^N$
 1: For each $X_1 \subseteq U_1$ do
 2:     For each $Y_2 \subseteq U_2$ do
 3:         set $g(Y_2) \leftarrow 0$
 4:     End For
 5:     For each $Y \in \mathcal{F}$ do
 6:         If $Y \cap U_1 \subseteq X_1$, then
 7:             set $g(Y \cap U_2) \leftarrow g(Y \cap U_2) + f(Y)$
 8:     End For
 9:     Compute $h \leftarrow g\zeta$ using the fast zeta transform on $2^{U_2}$
10:     For each $X_2 \subseteq U_2$ do
11:         output the value $h(X_2)$ as the value $f\zeta(X_1 \cup X_2)$
12:     End For
13: End For
```

In Algorithm 2, because the computations are independent for each $X_1 \subseteq U_1$, they can be executed in parallel on $O^*(2^n/|\mathcal{F}|)$ processors.

Example 9 (Counting k-Partitions). The problem and the notations are defined in Example 4. It is convenient to express the k-partitions in terms of generating polynomials. Based on the principle of inclusion-exclusion, (28) can be reformulated as follows in which $a_k(X)$ is replaced by a polynomial over an intermediate z:

$$d_k(\mathcal{F}) = \sum_{X \subseteq N} (-1)^{|N \setminus X|} (\sum_{i=0}^{n} a_i(X) z^i)^k, \tag{54}$$

where the coefficient $a_j(X)$ is the number of subsets $Y \subseteq X$ that belong to F and are of size j. Now $d_k(\mathcal{F})$ is a polynomial whose coefficient of the monomial z^n is the number of k-partitions. To evaluate this expression, note that the polynomial $a(X) = \sum_{i=0}^{n} a_i(X) z^i$ is equal to the zeta transform $g\zeta(X)$, where $g(Y) = [Y \in \mathcal{F}] z^{|Y|}$. Now the linear space fast zeta transform operating in a ring of polynomials lists the polynomials $a(X)$ for all $X \subseteq N$ in $O^*(2^n)$ time and $O^*(|\mathcal{F}|)$ space. □

Similar ideas to that used in the linear space fast zeta transform can give space efficient algorithms for other elementary transforms and problems, e.g., the intersection transform, the disjoint sum and the chromatic polynomial. An important feature of this type of algorithms is that they admit efficient parallelization as discussed before. In particular, some algorithms [19] of this type can achieve a trade-off between time and space complexities by adjusting the number of processors executing the algorithm in parallel.

2.6 Faster Exact Algorithms in Bounded Degree Graphs

For bounded-degree graphs, faster exact algorithms can be derived for certain hard problems. The basic idea is to exploit some special properties so that only a special set of subsets of the vertex set, but not all subsets, need to be considered; then based on a projection theorem presented by Chung et al. [23], which can be used to bound the size of the special set, the running time can be reduced for bounded-degree graphs. In this section, an inclusion-exclusion based algorithm will exemplify how to derive faster exact algorithms for bounded-degree graphs. This method is not only applicable to inclusion-exclusion based algorithms, but also to some other types of algorithms. The following lemma is the key to employ this method.

Lemma 8. *([23]) Let V be a finite set with subsets $A_1, A_2, \ldots, A_r$ such that every $v \in V$ is contained in at least δ subsets. Let $\mathcal{F}$ be a family of subsets of V. For each $1 \le i \le r$ define the projections $\mathcal{F}_i = \{F \cap A_i : F \in \mathcal{F}\}$. Then,*

$$|\mathcal{F}|^\delta \le \prod_{i=1}^{r} |\mathcal{F}_i|. \tag{55}$$

Example 10 (TSP in bounded-degree graphs). Given a graph G with nodes V and distant function $d : V \times V \to N$, the Traveling Salesman Problem (TSP) is to find a minimum weighted Hamiltonian cycle based on the distance function that minimizes the total distance. Assume that the maximum degree of G is $\Delta = O(1)$. A set W of vertices is a connected dominating set if every vertex $v \in V$ is either in W or adjacent to a vertex in W and the induced subgraph $G[W]$ is connected. Denote by $\mathcal{CD}$ the family of connected dominating sets of G.

The starting point is the algorithm by Karp [36], and, independently, by Kohn et al. [39]. Assume that the distance function is bounded by a constant, i.e., $d(u,v) \in \{0,1,\dots,B\} \cup \{\infty\}$ for any pair of vertices u, v, where $B = O(1)$.

The algorithm can be conveniently described in terms of generating polynomials. Select an arbitrary reference vertex $s \in V$, and let $U = V \setminus \{s\}$. For each $X \subseteq U$, denote by $q(X)$ the polynomial over the indeterminate z for which the coefficient of each monomial z^w counts the directed closed walks (in the complete directed graph with vertex set V and edge weights given by d) through s that (i) avoid the vertices in X, (ii) have length n, and (iii) have finite weight w. Then $q(X)$ can be computed in time polynomial in n by the following recurrence and setting $q(X) = p(n,s)$ for a fixed $X \subseteq U$. Initialize the recurrence for each vertex $u \in V \setminus X$ with

$$p(0,u) = \begin{cases} 1, & if\ u = s \\ 0, & otherwise. \end{cases} \tag{56}$$

For convenience, define $z^{\infty} = 0$. For each length $l = 1,2,\dots,n$ and each vertex $u \in V \setminus X$, let

$$p(l,u) = \sum_{v \in V \setminus X} p(l-1,v) z^{d(v,u)}. \tag{57}$$

Here note that due to the assumption on bounded weights, each $p(l,u)$ has at most a polynomial number of monomials with nonzero coefficients.

By the principle of inclusion-exclusion, the coefficients of the monomials in the polynomial

$$Q = \sum_{X \subseteq U} (-1)^{|X|} q(X) \tag{58}$$

count, by weight, the number of directed Hamiltonian cycles. It follows immediately that the traveling salesman problem can be solved in time $O^*(2^n)$.

For bounded-degree graphs, faster algorithms can be derived by analyzing (58) in more details. It will be convenient to work with a complemented form of (58). For each $S \subseteq U$, let $r(S) = q(U \setminus S)$. Then (58) can be rewritten as

$$Q = (-1)^n \sum_{S \subseteq U} (-1)^{|S|} r(S). \tag{59}$$

Observe that the induced subgraph $G[\{s\} \cup S]$ need not be connected. Associate with each $S \subseteq U$ the unique $f(S) \subseteq U$ such that $G[\{s\} \cup f(S)]$ is the connected component of $G[\{s\} \cup S]$ that contains s. It follows that $r(S) = r(f(S))$

for all $S \subseteq U$. This observation enables the following partition of the subsets of U into f-preimages of constant r-value. For each $T \subseteq U$, let $f^{-1}(T) = \{S \subseteq U : f(S) = T\}$. Then rewrite (59) in the partition form

$$Q = (-1)^n \sum_{T \subseteq U} r(T) \sum_{S \in f^{-1}(T)} (-1)^{|S|}. \tag{60}$$

Lemma 9.

$$\sum_{S \in f^{-1}(T)} (-1)^{|S|} = \begin{cases} (-1)^{|T|}, & \textit{if } \{s\} \cup T \in \mathcal{CD} \\ 0, & \textit{otherwise.} \end{cases} \tag{61}$$

Proof. Consider an arbitrary $T \subseteq U$. The preimage $f^{-1}(T)$ is clearly empty if $G[\{s\} \cup T]$ is not connected. Thus in the following assume that $G[\{s\} \cup T]$ is connected. For a set $W \subseteq V$, denote $\overline{N}(W)$ the set of vertices in W or adjacent to at least one vertex in W. Observe that $f(S) = T$ holds for an $S \subseteq U$ iff $T \subseteq S$ and $S \cap \overline{N}(\{s\} \cup T) = T$. In particular, if $V \setminus \overline{N}(\{s\} \cup T)$ is non-empty, then $f^{-1}(T)$ contains equally many even- and odd-sized subsets. Conversely, if $V \setminus \overline{N}(\{s\} \cup T)$ is empty (which means that $\{s\} \cup T$ is a dominating set of G), then $f^{-1}(T) = \{T\}$. $\square$

Using the above lemma to simplify (60), we have

$$Q = (-1)^n \sum_{T \subseteq U, \{s\} \cup T \in \mathcal{CD}} (-1)^{|T|} r(T). \tag{62}$$

To arrive at an algorithm with running time $|\mathcal{CD}| n^{O(1)}$, it suffices to list the elements of $\mathcal{CD}$ with delay bounded by $n^{O(1)}$. Observe that $\mathcal{CD}$ is an up-closed family of subsets of V, that is, if a set is in the family, then so are all of its supersets. Furthermore, whether a given $W \subseteq V$ is in $\mathcal{CD}$ can be decided in polynomial time. These observations enable listing the sets in $\mathcal{CD}$ in a top-down, depth-first manner. Thus the only remaining task is to bound the size of $\mathcal{CD}$.

Lemma 10. *Let V be a finite set with r elements and with subsets $A_1, A_2, \ldots, A_r$ such that every $v \in V$ is contained in exactly δ subsets. Let $\mathcal{F}$ be a family of subsets of V and assume that there is a log-concave function $f \geq 1$ and $0 \leq s \leq r$ such that the projections $\mathcal{F}_i = \{F \cap A_i : F \in \mathcal{F}\}$ satisfy $|\mathcal{F}_i| \leq f(|A_i|)$ for each $s+1 \leq i \leq r$. Then,*

$$|\mathcal{F}| \leq f(\delta)^{r/\delta} \prod_{i=1}^{s} 2^{|A_i|/\delta}. \tag{63}$$

Proof. Let $a_i = |A_i|$ and note that $\sum_{i=1}^{r} a_i = \delta r$. By Lemma 8,

$$|\mathcal{F}|^\delta \leq \prod_{i=1}^{s} 2^{a_i} \prod_{i=s+1}^{r} f(a_i) \leq \prod_{i=1}^{s} 2^{a_i} \prod_{i=1}^{r} f(a_i). \tag{64}$$

Since f is log-concave, Jensen's inequality gives

$$\frac{1}{r}\sum_{i=1}^{r}\log f(a_i) \leq \log f((\sum_{i=1}^{r} a_i)/r) = \log f(\delta). \tag{65}$$

Taking exponential and combining with (64) gives $|\mathcal{F}|^{\delta} \leq f(\delta)^r \prod_{i=1}^{s} 2^{|A_i|}$, which yields the claimed bound. □

In order to bound the size of $\mathcal{CD}$, we need only to consider the special case where A_i are defined in terms of neighborhoods of the vertices of G. For each $v \in V$, define the closed neighborhood $N(v)$ by $N(v) = \{v\} \cup \{u \in V : u \text{ and } v \text{ are adjacent in } G\}$. Begin by defining the subsets A_v for $v \in V$ as $A_v = N(v)$. Then, for each $u \in V$ with degree $d(u) < \Delta$, add u to $\Delta - d(u)$ of the sets A_v not already containing it (it does not matter which ones). This ensures that every $u \in V$ is contained in exactly $\Delta + 1$ sets A_v. Combining with the following given bound on the size of $\mathcal{CD}$, an $O^*(\beta_{\Delta}^{n})$ time algorithm can be obtained as described above, where $\beta_{\Delta} = (2^{\Delta+1} - 2)^{1/(\Delta+1)}$.

Lemma 11. *An n-vertex graphs with maximum vertex degree Δ has at most $\beta_{\Delta}^{n} + n$ connected dominating sets, where $\beta_{\Delta} = (2^{\Delta+1} - 2)^{1/(\Delta+1)}$.*

Proof. Let $\mathcal{CD}^{'} = \mathcal{CD} \setminus \{\{v\} : v \in V\}$. Then for every $C^{'} \in \mathcal{CD}^{'}$ and every A_v, $C^{'} \cap A_v \neq \{v\}$. Thus the number of sets in the projection $\mathcal{CD}_{v}^{'} = \{F \cap A_v : F \in \mathcal{CD}^{'}\}$ is at most $2^{|A_v|} - 2$, since $F \cap A_v \neq \emptyset$ for any $F \in \mathcal{CD}^{'}$. To obtain the bound, apply Lemma 10 with the log-concave function $f(x) = 2^x - 1$ and $s = 0$. □

3 Algebraic Methods

The application of algebraic methods in designing exact algorithms can be dated back to the famous paper by Kohn et al. [39] on solving the traveling salesman problem using the generating polynomials (for details, please refer to Example 10), in which by involving an intermediate x, a polynomial $G(x) = \sum a_i x^{e_i}$ is produced, where e_i denotes the cost of a possible tour and a_i denotes the number of tours having this cost. Then the minimal tour cost, i.e., the smallest exponent, can be extracted by evaluating $G(x)$ at a specific value close to 0 and using a logarithmic technique. In this section, two recently developed algebraic methods are introduced—algebraic sieving method and polynomial circuit based algebraic method.

3.1 Algebraic Sieving

Sieving methods are commonly used in designing exact and parameterized algorithms, and have a long history. For example, the inclusion-exclusion principle is a typical sieving method by which all possibilities are first counted, and then false contributions are cancelled out. Here a new algebraic sieving method which makes use of determinants over a field of characteristic two to achieve sieving is introduced. The basic idea is to construct a polynomial in the underlying variables over a finite field in which at least one unique monomial exists for each required structure (e.g., Hamiltonian cycle) and no monomials results from unwanted structures (e.g., non-Hamiltonian cycle covers). The existence of structures being looked for ultimately emerges as a polynomial testing problem. Then the old fingerprint idea, often attributed to Freivalds (c.f. [43]), is applied in which the polynomial is evaluated in a randomly chosen point in a finite field. The fingerprint result is used to judge whether the constructed polynomial is zero polynomial or not. The Schwartz-Zippel Lemma (c.f. [43]) ensures that such judgement will be correct with high probability. The key point of this method is how to sieve the unwanted monomials to obtain the final required polynomial. So far only determinants over a finite field is taken advantage of to achieve sieving. Whether there exists some other more powerful tools that can be used for sieving under the algebraic framework needs further studying. Before going into the details, some notations need to be defined.

The determinant of an $n \times n$ matrix $\mathbf{A}$ over an arbitrary ring R can be defined by the Leibniz formula:

$$det(\mathbf{A}) = \sum_{\sigma:[n]\to[n]} sgn(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}, \tag{66}$$

where the summation is over all permutations of n elements, and sgn is a function called the sign of the permutation which assigns either 1 or -1 to a permutation. The permanent of $\mathbf{A}$ is defined by

$$per(\mathbf{A}) = \sum_{\sigma:[n]\to[n]} \prod_{i=1}^{n} A_{i,\sigma(i)}. \tag{67}$$

Denote $GF(2^k)$ as a finite field of characteristic two. It is well known that the determinant of $\mathbf{A}$ over $GF(2^k)$ coincides with the permanent, since in such a field every element serves as its own additive inverse, in particular the element 1; the sgn function identically maps 1 to every permutation. Moreover, although the determinant is a sum of an exponential number of terms, it can be computed in polynomial time using for instance LU-factorization of the matrix which can be found in any textbook on linear algebra. In fact, to compute the determinant is not harder than square matrix multiplication (c.f. [21]). Hence the determinant can be computed in $O(n^\omega)$ field operations where ω is the Coppersmith-Winograd exponent [24]. The following properties and lemma will be used in deriving algebraic sieving algorithms.

Property 1. $\mathbf{A}$ is a matrix over $GF(2^k)$, then $per(\mathbf{A}) = det(\mathbf{A})$.

Property 2. For a given $n \times n$ matrix $\mathbf{A}$, $det(\mathbf{A})$ can be computed in $O(n^\omega)$ time.

Lemma 12 (Schwartz-Zippel). *Let $P(x_1, \dots, x_n)$ be a non-zero n-variate polynomial of total degree d over a field F. Pick $r_1, \dots, r_n \in F$ uniformly at random, then*

$$P_r(P(r_1, \dots, r_n) = 0) \leq \frac{d}{|F|}. \tag{68}$$

3.1.1 k-Dimensional Matching

A hypergraph $H = (V, E)$ consists of a set V of n vertices and a multiset E of hyperedges which are subsets of V. In particular, hyperedges may include only one (or even no) vertex and may appear more than once. In a k-uniform hypergraph each edge $e \in E$ has size $|e| = k$. Given a vertex subset U of V, the projected hypergraph of H on U, denoted as $H[U] = (U, E[U])$, is a hypergraph on U where there is one edge $e_U \in E[U]$ for every $e \in E$, defined by $e_U = e \cap U$.

Definition 4 (k-Dimensional Matching). *Given a k-uniform hypergraph $H = (V_1 \cup V_2 \cdots \cup V_k, E)$, with $E \subseteq V_1 \times V_2 \times \cdots \times V_k$, where $|V_i| = n$ for $1 \leq i \leq k$, the k-dimensional matching problem asks whether there exists a k-dimensional matching $S \subseteq E$ such that $\cup_{e \in S} e = V_1 \cup V_2 \cdots \cup V_k$ and $\forall e_1 \neq e_2 \in S$: $e_1 \cap e_2 = \emptyset$.*

In the following, each hyperedge e is associated with a variable x_e over $GF(2^m)$ for some $m \geq \log n + 1$. The next lemma shows how to construct a polynomial only consisting of monomials representing k-dimensional matchings by sieving unwanted monomials.

Lemma 13. *Denote $\mathcal{M}$ as the family of all k-dimensional matchings. Let $V = \cup_{i=1}^{k} V_i$ and $U = V_1 \cup V_2$, then*

$$\sum_{X \subseteq V \setminus U} P(H, U, X) = \sum_{M \in \mathcal{M}} \prod_{e \in M} x_e, \tag{69}$$

where the computation is over a multivariate polynomial ring over $GF(2^m)$, and

$$P(H, U, X) = \sum_{M' \in \mathcal{M}'} \prod_{e \in M'} x_e, \tag{70}$$

where $\mathcal{M}'$ is the family of all perfect matchings in $H[U]$ avoiding X, i.e., for every $e \in M'$, $e \cap X = \emptyset$.

Proof. For every $M \in \mathcal{M}$, M only avoids $\emptyset$. Thus the monomial $\prod_{e \in M} x_e$ for every k-dimensional matching M emerges in the constructed polynomial precisely once. For a non-k-dimensional matching M', it avoids all subsets of $V \setminus (\cup_{e \in M'} e)$. Then the monomial $\prod_{e \in M'} x_e$ for M' will be considered in $P(H, U, X')$ for every $X' \subseteq V \setminus (\cup_{e \in M'} e)$. It is well known that a non-empty set has even number of subsets. Hence, all of these $\prod_{e \in M'} x_e$ cancel each other since the operations are in a field of characteristic two. $\square$

¿From Lemma 13, $\sum_{X \subseteq V \setminus U} P(H, U, X)$ consists of monomials representing k-dimensional matchings of H. Thus, by evaluating $\sum_{X \subseteq V \setminus U} P(H, U, X)$ on a randomly chosen point $r_1, \ldots, r_{|E|} \in GF(2^m)$, the result is non-zero with probability at least $\frac{1}{2}$ if there exists at least one k-dimensional matching in H by Lemma 12. If repeating the evaluation for polynomial times, the error probability can be exponentially small. The remaining work is how to efficiently compute $P(H, U, X)$ on a randomly chosen point $r_1, \ldots, r_{|E|} \in GF(2^m)$ for every $X \subseteq V \setminus U$.

Clearly, $H[U]$ is a bipartite multigraph. Define its Edmonds matrix $\mathbf{A}(H, V_1, V_2)$ as

$$\mathbf{A}(H, V_1, V_2)_{ij} = \sum_{e=(i,j) \in E[U], i \in V_1, j \in V_2} x_e. \tag{71}$$

Denote $\mathcal{M}$ as the family of all perfect matchings in $H[U]$. Then the following lemma generalizes Edmonds' work [26] for the case of bipartite multigraphs.

Lemma 14. *$det(\boldsymbol{A}(H, V_1, V_2)) = \sum_{M \in \hat{\mathcal{M}}} \prod_{e \in M} x_e$, where the computation is over a multivariate polynomial ring over $GF(2^m)$ and the summation is over all perfect matchings $\hat{\mathcal{M}}$ in $H[V_1 \cup V_2]$.*

Proof. By the definition of determinant, the summation is over all products of n of the matrix elements in which every row or column is used exactly once. In terms of the bipartite graph, this corresponds to a perfect matching in the graph since rows and columns represent the two vertex sets respectively. Furthermore, the converse is also true. For every perfect matching there is a permutation describing it. Hence the mapping is one-to-one. The inner product counts all choices of edges producing a matching described by a permutation σ since

$$\prod_{i=1}^{n} A_{i,\sigma(i)} = \prod_{i=1}^{n} \sum_{e=(i,\sigma(i))} x_e = \sum_{M \in \mathcal{M}(\sigma)} \prod_{e \in M} x_e, \tag{72}$$

where $\mathcal{M}(\sigma)$ is the set of all perfect matchings $\{e_1, \ldots, e_n\}$ such that $e_i = (i, \sigma(i))$.

Let $H_X[U]$ denote the projected hypergraph of H on U by only projecting edges disjoint to X in H on U. Based on the above Lemma, $P(H, U, X)$ on a randomly chosen point $r_1, \ldots, r_{|E|} \in GF(2^m)$ for every $X \subseteq V \setminus U$ can be computed by first constructing the Edmonds matrix of $H_X[U]$, with the variables replaced by the random sample points $r_1, \ldots, r_{|E|}$, and then computing its determinant. Putting everything together generates a Monte Carlo algorithm with exponentially small error probability for the k-dimensional matching problem. The runtime bound is easily seen to be $O^*(2^{(k-2)n})$ since $|U| = |V_1| + |V_2| = 2n$ and $P(H, U, X)$ for every $X \subseteq V \setminus U$ can be evaluated in $O(n^\omega)$ time by Lemma 14 and Property 2.

Theorem 9. *The k-dimensional matching problem on kn vertices can be solved by a Monte Carlo algorithm with an exponentially small failure probability in n and $O^*(2^{(k-2)n})$ runtime.*

3.1.2 Hamiltonicity

An undirected graph $G = (V, E)$ on n vertices is said to be Hamiltonian if it has a Hamiltonian cycle, a vertex order $(v_0, v_1, \ldots, v_{n-1})$ such that $v_i v_{i+1} \in E$ for all i. The indices are enumerated modulo n, i.e., $v_{n-1}v_0$ is also an edge. The problem of detecting if a graph is Hamiltonian is called the Hamiltonicity problem. Clearly, this problem is a special case of the traveling salesman problem.

Before the algebraic sieving algorithm for the Hamiltonicity problem was derived, the dynamic programming recurrence that solves the general TSP in $O(n^2 2^n)$ time introduced by Bellman [7][5] and independently by Held and karp [30] in the early 1960's was the strongest result for this special problem. The algebraic sieving for the Hamiltonicity problem is obtained by reducing the problem to a variant of the cycle cover counting problem called Labeled Cycle Cover Sum, and then making use of the sieving algorithm to solve this variant. The algorithm will be described after introducing some useful notations.

In a directed graph $D = (V, A)$, a cycle cover is a subset $C \subseteq A$ such that for every vertex $v \in V$, there is exactly one arc $a_{v1} \in C$ starting from v and exactly one arc $a_{v2} \in C$ ending in v. The graphs considered have no loops. Denote $cc(D)$ as the family of all cycle covers of D, and $hc(D) \subseteq cc(D)$ as the set of Hamiltonian cycle covers. A Hamiltonian cycle cover consists of one big cycle passing through all vertices. The remaining cycle covers (which have more than one cycle in $cc(D) \setminus hc(D)$) are called non-Hamiltonian cycle covers. For undirected graphs, $hc(G)$ includes the Hamiltonian cycles with orientation, i.e., traversed in both directions. For a Hamiltonian cycle $H \in hc(G)$ for an undirected graph G, an arcs $uv \in G$ infers that the cycle is oriented from u to v along the edge uv. Denote $g : A \to B$ as a surjective function from the domain A to the codomain B and denote the preimage of every $b \in B$ as $g^{-1}(b)$, i.e., $g^{-1}(b) = \{a \in A : g(a) = b\}$. For a matrix A, denote by $A_{i,j}$ the element at row i and column j. The labeled cycle cover sum problem is defined as follows:

Definition 5. *Given a directed graph $D = (V, A)$, a finite set L of labels, and a function $f : A \times 2^L \to R$ on some codomain ring R, the labeled cycle cover sum problem is to compute the following defined term over R.*

$$\Lambda(D, L, f) = \sum_{C \in cc(D)} \sum_{g : L \to C} \prod_{a \in C} f(a, g^{-1}(a)), \tag{73}$$

where the inner sum is over all surjective functions g.

In the above definition, f is introduced to make all non-Hamiltonian cycle covers cancel out in the sum. To do this, as before, the computations will be done over $GF(2^k)$ denoting a finite field of characteristic two. In what follows, a directed graph is bidirected if for every arc uv it has an arc in the opposite direction, i.e., vu. In order to make sure that the Hamiltonian cycle cover will not be canceled as well, an asymmetry around an arbitrarily chosen special vertex s is defined. A function $f : A \times 2^L \to GF(2^k)$ is an s-oriented mirror function if $f(uv, Z) = f(vu, Z)$ for all Z and all $u \neq s, v \neq s$. The following lemma captures how the non-Hamiltonian cycle covers vanish, which also implies the non-existence of false positives in the resulting algorithms.

Lemma 15. *Given a bidirected graph $D = (V, A)$, a finite set L, and a special vertex $s \in V$, let f be an s-oriented mirror function with codomain $GF(2^k)$. Then*

$$\Lambda(D, L, f) = \sum_{H \in hc(D)} \sum_{g:L \to H} \prod_{a \in H} f(a, g^{-1}(a)). \tag{74}$$

Proof. By the definition of the labeled cycle cover sum, a labeled cycle cover is a tuple (C, g) with $C \in cc(D)$ and $g : L \to C$. The labeled non-Hamiltonian cycle covers can be partitioned into dual pairs as follows such that both cycle covers in every pair contribute the same term to the sum. For a labeled non-Hamiltonian cycle cover (C, g), let $\mathbf{C}$ be the first cycle of C not passing through s (note that such cycle must exist since the cycle cover consists of at least two cycles and all cycles are vertex disjoint). Here "first" is w.r.t. any fixed order of the cycles. Then the dual cycle cover of (C, g) is defined as $R(C, g) = (C^{'}, g^{'})$, where $C^{'} = C$ except for the cycle $\mathbf{C}$ which is reversed in $C^{'}$, i.e., every arc $uv \in \mathbf{C}$ is replaced by the arc in the opposite direction vu in $C^{'}$, and $g^{'-1}$ is identical to g^{-1} on $C \backslash \mathbf{C}$, and is defined by $g^{'-1}(uv) = g^{-1}(vu)$ for all arcs $uv \in \mathbf{C}$. In other words, the reversed arcs preserve their original labeling. Note that such a dual cycle cover always exists since D is bidirected and $(C, g) \neq R(C, g)$,$(C, g) = R(R(C, g))$. Hence the mapping uniquely pairs up the labeled non-Hamiltonian cycle covers.

Since f is an s-oriented mirror function and has $f(uv, Z) = f(vu, Z)$ for all arcs uv not incident on s, (C, g) and $R(C, g)$ contribute the same product term to the sum for computing $\Lambda(D, L, f)$. Finally, since the computations are done in a field of characteristic two, all these terms will be canceled. □

When limiting the computations over $GF(2^k)$, the labeled cycle cover sum can be computed efficiently via determinant. Permanent has a natural interpretation as the sum of weighted cycle covers in a directed graph. Formally, let $D = (V, A)$ be a directed graph with weights $w : A \to GF(2^k)$, and define a $|V| \times |V|$ matrix with rows and columns representing the vertices V

$$A_{i,j} = \begin{cases} w(ij), & if\ ij \in A \\ 0, & otherwise. \end{cases} \tag{75}$$

Then

$$per(A) = \sum_{C \in cc(D)} \prod_{a \in C} w(a). \tag{76}$$

By Property 1 and Property 2, $per(A)$ is identical with $det(A)$ and can be computed in $O(n^\omega)$ time. Define a polynomial in an indeterminate r as follows, with r aiming at controlling the total rank of the subsets used as labels in the labeled cycle covers:

$$p(f, r) = \sum_{Y \subseteq L} det(\sum_{Z \subseteq Y} r^{|Z|} M_f(Z)), \tag{77}$$

where

$$M_f(Z)_{i,j} = \begin{cases} f(ij, Z), & if\ ij \in A \\ 0, & otherwise. \end{cases} \tag{78}$$

Lemma 16. *For a directed graph D, a set L of labels, and any function $f : A \times L \to GF(2^k)$,*

$$[r^{|L|}]p(f,r) = \Lambda(D,L,f), \tag{79}$$

where $[r^{|L|}]p(f,r)$ denotes the coefficient of $r^{|L|}$ in $p(f,r)$.

Proof. $p(f,r)$ can be rewritten as

$$\begin{aligned} p(f,r) &= \sum_{Y\subseteq L}\sum_{C\in cc(D)}\sum_{q:C\to 2^Y\setminus\{\emptyset\}}\prod_{a\in C} r^{|q(a)|}f(a,q(a)) \\ &= \sum_{C\in cc(D)}\sum_{q:C\to 2^L\setminus\{\emptyset\}}\sum_{\cup_{a\in C}q(a)\subseteq Y\subseteq L}\prod_{a\in C} r^{|q(a)|}f(a,q(a)). \end{aligned} \tag{80}$$

For functions $q : C \to 2^L \setminus \{\emptyset\}$ such that $\cup_{a\in C}q(a) \subset L$, i.e., whose union over the elements do not cover all of L, the innermost summation is executed an even number of times with the same term (there are $2^{|L-\cup_{a\in C}q(a)|}$ equal terms). Since the computations are over $GF(2^k)$, these cancel. Then

$$p(f,r) = \sum_{C\in cc(D)}\sum_{\substack{q:C\to 2^L\setminus\{\emptyset\}\\ \cup_{a\in C}q(a)=L}} r^{\sum_{a\in C}|q(a)|}\prod_{a\in C} f(a,q(a)), \tag{81}$$

in which the coefficient of $r^{|L|}$

$$[r^{|L|}]p(f,r) = \sum_{C\in cc(D)}\sum_{\substack{q:C\to 2^L\setminus\{\emptyset\}\\ \cup_{a\in C}q(a)=L\\ \forall a\neq b: q(a)\cap q(b)=\emptyset}}\prod_{a\in C} f(a,q(a)), \tag{82}$$

since $\cup_{a\in C}q(a) = L, \cup_{a\in C}|q(a)| = |L|$ implies $\forall a \neq b : q(a)\cap q(b) = \emptyset$. Inverting the function q will give the labeled cycle cover sum as defined in Definition 5. □

The above lemma is the base identity that enables a relatively efficient algorithm for computing the labeled cycle cover sum.

Lemma 17. *The labeled cycle cover sum $\Lambda(D,L,f)$ over a field $GF(2^k)$ on a directed graph D on n vertices, and with $2^k > |L|n$, can be computed in $O((|L|^2 n + |L|n^{1+\omega})2^{|L|})$ arithmetic operations over $GF(2^k)$, where ω is the Coppersmith-Winograd matrix multiplication coefficient.*

Proof. The labeled cycle cover sum is evaluated via the identity in Lemma 16. Observing that $p(f,r)$ as a polynomial in r has maximum degree $|L|n$, to recover one of its coefficients (the one for $r^{|L|}$), one needs to evaluate the polynomial for $|L|n$ choices of r and to use interpolation to solve for the coefficient being sought. This can be done for instance by using a generator g of the multiplicative group in $GF(2^k)$ and evaluating the polynomial in the points $r = g^0, g^1, \ldots, g^{|L|n-1}$. The requirement $2^k > |L|n$ assures the distinctness of these points, and hence the interpolation is possible. For every fixed r, the algorithm begins by tabulating $T(Y) = \sum_{Z \subseteq Y} r^{|Z|} M_f(Z)$ for all $Y \subseteq L$ through the fast zeta transform (4) in $O(|L|2^{|L|})$ field operations. Then $p(f, r) = \sum_{Y \subseteq L} det(T(Y))$ can be evaluated in $O(n^{\omega} 2^{|L|})$ operations by Property 2. Once all values are computed, the polynomial time Lagrange interpolation can be used to compute the coefficient. Summing up the number of field operations required over all $|L|n$ values of r, the lemma follows. □

Next, f will be defined to associate the argument arc and label set with a non-constant multivariate polynomial such that $f(su, X)$ and $f(us, X)$ will not share variables for any $su, us \in A$ to ensure that the Hamiltonian cycles oriented in opposite directions will contribute different terms to the sum. Based on the definition of f, the labeled cycle cover sum is represented by a polynomial in the underlying variables with one unique monomial per oriented Hamiltonian cycle and without monomials resulting from non-Hamiltonian cycle covers, which is a consequence of Lemma 15. Then the fingerprint idea is employed to evaluate the polynomial in a randomly chosen point and the Schwarz-Zippel Lemma ensures that with high probability, it can be detected whether the polynomial resulting from the labeled cycle cover sum is identically zero (i.e., no Hamiltonian cycles) or not (there is at least one Hamiltonian cycle) by identifying the evaluating result with the polynomial. For simplicity of analysis, assume that n is even.

For a uniformly randomly chosen partition $V = V_1 \cup V_2$ with $|V_1| = |V_2| = n/2$ and a fixed Hamiltonian cycle $H = (v_0, \ldots, v_{n-1})$ in G. An arc $v_i v_{i+1}$ is called unlabeled by V_2 if both v_i and v_{i+1} belong to V_1, and the set of all unlabeled arcs is denoted as $U(H)$. The remaining arcs are referred to as labeled by V_2, and denote the set of all labeled arcs as $L(H)$. Define $hc(G, V_2, m)$ as the subset of $hc(G)$ of Hamiltonian cycles which have precisely m unlabeled arcs by V_2. Assign every edge $uv \in E$ two variables x_{uv} and x_{vu}, and x_{uv} and x_{vu} are identified except when $u = s$ or $v = s$.

Consider a complete bidirected graph $D = (V_1, F)$ and use V_2 as some of the labels. In addition to V_2, a set L_m of size m of extra labels aiming at handling arcs unlabeled by V_2. For each edge uv in $G[V_1]$ and every element $d \in L_m$, new variables $x_{uv,d}$ and $x_{vu,d}$ are introduced. And $x_{uv,d}$ coincides with $x_{vu,d}$ except when $u = s$ or $v = s$. For two vertices $u, v \in V$, and a subset $X \subseteq V$, define $\mathcal{P}_{u,v}(X)$ as the family of all simple paths in G from u to v passing through exactly the vertices in X. For $uv \in F$ and $X \subseteq V_2$, define

$$f(uv, X) = \sum_{P \in \mathcal{P}_{u,v}(X)} \prod_{wz \in P} x_{wz}. \tag{83}$$

For every arc $uv \in F$ such that uv is an edge in $G[V_1]$, and $d \in L_m$, define $f(uv, \{d\}) = x_{uv,d}$. In other points f is set to zero.

Lemma 18. *With $G, D, V_2, U(\cdot), L(\cdot), m, L_m$ and f defined as above,*
(i) $\Lambda(D, V_2 \cup L_m, f) = \sum_{H \in hc(G,V_2,m)} (\prod_{uv \in L(H)} x_{uv})(\sum_{\sigma: U(H) \to L_m} \prod_{uv \in U(H)} x_{uv,\sigma(uv)})$;
(ii) $\Lambda(D, V_2 \cup L_m, f)$ is a zero polynomial iff $hc(G, V_2, m) = \emptyset$.

Proof. (i) Since D is bidirected and f is s-oriented mirror function, by Lemma 15,

$$\Lambda(D, V_2 \cup L_m, f) = \sum_{H \in hc(D)} \sum_{g: V_2 \cup L_m \to H} \prod_{a \in H} f(a, g^{-1}(a)), \tag{84}$$

where the inner sum is over all surjective functions g. In order to prove the claim, it needs to expand the Hamiltonian cycles in D into Hamiltonian cycles of G. Note that the arcs of a Hamiltonian cycle in D in the sum above are labeled either by an element of L_m or by a non-empty subset of V_2, since only in such cases f is non-zero. Next the definition of labeled and unlabeled arcs are extended (which were defined previously for Hamiltonian cycles in G only). For a Hamiltonian cycle $H \in hc(D)$ labeled by the surjective function $g : V_2 \cup L_m \to H$, an arc $uv \in H$ is called labeled by V_2 if $g^{-1}(uv) \subseteq V_2$, and unlabeled by V_2 if $g^{-1}(uv) \in L_m$.

Since every arc uv unlabeled by V_2, $g^{-1}(uv)$ is an element of L_m, only the Hamiltonian cycles in D with exactly m arcs unlabeled by V_2 leave a non-zero contribution. Then

$$\Lambda(D, V_2 \cup L_m, f) = \sum_{H \in hc(D)} \sum_{\substack{H_L \cup H_U = H \\ H_L \cap H_U = \emptyset \\ |H_U| = m}} \Big(\sum_{g: V_2 \to H_L} \prod_{a \in H_L} f(a, g^{-1}(a)) \Big) \cdot \Big(\sum_{\sigma: H_U \to L_m} \prod_{a \in H_U} f(a, \sigma(a)) \Big), \tag{85}$$

where the summation is over all surjective functions g and one-to-one functions σ. Replace f in the above expression, then

$$\begin{aligned}\Lambda(D, V_2 \cup L_m, f) = &\sum_{H \in hc(D)} \sum_{\substack{H_L \cup H_U = H \\ H_L \cap H_U = \emptyset \\ |H_U| = m \\ \forall a \in H_U : a \in E}} \Big(\sum_{g: V_2 \to H_L} \prod_{a \in H_L} \sum_{P \in \mathcal{P}_{u,v}(g^{-1}(uv))} \prod_{wz \in P} x_{wz} \Big) \\ &\cdot \Big(\sum_{\sigma: H_U \to L_m} \prod_{uv \in H_U} x_{uv, \sigma(uv)} \Big),\end{aligned} \tag{86}$$

Every vertex in V_2 is mapped to precisely one arc in F on a Hamiltonian cycle H in D by g. Moreover, such a cycle H leaves a non-zero result iff there are precisely m arcs in the cycle not being mapped to by g (i.e. unlabeled by V_2) and they are also edges in G. Rewriting the expression as a sum of Hamiltonian cycles in G,

$$\Lambda(D, V_2 \cup L_m, f) = \sum_{H \in hc(G, V_2, m)} \Big(\prod_{uv \in L(H)} x_{uv} \Big) \Big(\sum_{\sigma: U(H) \to L_m} \prod_{uv \in U(H)} x_{uv, \sigma(uv)} \Big). \tag{87}$$

(ii) If the graph G has no Hamiltonian cycles, the sum is clearly zero. For the other direction, each Hamiltonian cycle contributes a set of $m!$ different monomials per orientation of the cycle (one for each permutation σ), in which there is one variable per edge along the cycle. Monomials resulting from different Hamiltonian cycles are different since for each of two different Hamiltonian cycles, one has an edge that the other does not have. Every pair of opposite directed Hamiltonian cycles along the same undirected Hamiltonian cycle also traverse s through opposite directed arcs. Since the variables tied to the opposite directed arcs incident on s are different, these are also unique monomials in the sum. □

The above Lemma describes how to transform the input graph G given m, V_1, V_2 into a symbolic labeled cycle cover sum $\Lambda(D, V_2 \cup L_m, f)$ on a bidirected graph D on $n/2$ vertices and $n/2 + m$ labels $V_2 \cup L_m$. The next lemma shows that only the cases $m \leq n/4$ need to be considered.

Lemma 19. *Let $G = (V, E)$ be a Hamiltonian undirected graph. For a uniformly randomly chosen equal partition $V_1 \cup V_2 = V$, $|V_1| = |V_2| = n/2$,*

$$Pr(\sum_{i=0}^{n/4} |hc(G, V_2, i)| > 0) \geq \frac{1}{n+1}. \tag{88}$$

Proof. Consider one Hamiltonian cycle $(v_0, \ldots, v_{n-1})$ in G. Let X denote the random variable representing the number of unlabeled arcs by V_2. Clearly, $Pr(v_i) = \frac{1}{2}$ for any i, and $Pr(v_i, v_{i+1} \in V_1) < \frac{1}{4}$ since the events $v_i \in V_1$ and $v_{i+1} \in V_1$ are almost independent. Hence the expected number of arcs unlabeled by V_2, $E(X) < n/4$. By Markov's inequality, $Pr(X > (1 + 1/n)E(X)) < \frac{1}{1+1/n}$, and the lemma follows.

Before discussing the algorithm, the only remaining problem is how to calculate f for all subsets of V_2. This can be achieved by running a variant of the Bellman-Held-Karp recursion. Formally, let $\hat{f} : (V \times V) \times 2^V \to GF(2^k)$ be defined by

$$\hat{f}(uv, X) = \sum_{P \in \mathcal{P}_{u,v}(X)} \prod_{wz \in P} x_{wz}. \tag{89}$$

Then $f(uv, X) = \hat{f}(uv, X)$ for $u, v \in V_1, X \subseteq V_2$, and the recursion

$$\hat{f}(uv, X) = \begin{cases} x_{uv}, & if\ |X| = 0 \\ \sum_{w \in X, uw \in E} x_{uw} \hat{f}(wv, X \setminus \{w\}), & otherwise. \end{cases} \tag{90}$$

can be used to tabulate $f(\cdot, X)$ on $X \subseteq V_2$. The running time is $O^*(2^{|V_2|})$.

Next the algorithm is described in detail. First, pick a partition $V_1 \cup V_2 = V$ uniformly at random with $|V_1| = |V_2| = n/2$. Then the algorithm loops over m, the number of edges unlabeled by V_2 along a Hamiltonian cycle from 0 to $n/4$. For each value of m, by Lemma 18, the problem is transformed to determine whether the symbolic labeled cycle cover sum $\Lambda(D, V_2 \cup L_m, f)$ is a non-zero polynomial. As described in Lemma 17, $\Lambda(D, V_2 \cup L_m, f)$ will be evaluated in a randomly chosen point over the field $GF(2^k)$ with $2^k > cn$ for some $c > 1$. By Lemma 12, with probability at least $1 - 1/c$ it will result in a non-zero answer iff $\Lambda(D, V_2 \cup L_m, f)$ was a non-zero polynomial (i.e., G has a Hamiltonian cycle with m edges unlabeled by V_2). Then repeat the algorithm for every $m \leq n/4$, and by Lemma 19, with probability at least $(1 - 1/c)/(n + 1)$, the presence of a Hamiltonian cycle can be detected if it exists. Running the algorithm a polynomial number of times in n, the probability of failure will be reduced to exponentially small. The runtime is bounded by a polynomial factor in m and n of the runtime in Lemma 17. The worst case occurs for $m = 4/n$ in which case the time bound is $O^*(2^{3n/4})$.

Theorem 10. *There is a Monte Carlo algorithm detecting whether an undirected graph with n vertices is Hamiltonian or not in $O^*(2^{3n/4})$ time, with no false positives and no false negatives with probability exponentially small in n.*

3.2 Polynomial Circuit based Algorithms

This method only needs polynomial space. The problems to be solved will be translated into polynomials represented by polynomial circuits, where the translation is done in such a way that the instance considered is a "yes"-instance iff the coefficient of a special monomial in the polynomial is non-zero; this allows us only to compute the coefficient of the special monomial. A discrete Fourier transform based technique provides an efficient way to achieve the goal.

3.2.1 Coefficient Interpolation

Given a set R and a set of variables X, an arithmetic circuit C over $(R, +, *)$ is a directed acyclic graph in which each source gate is labeled by a variable $x \in X$ or an element of R and other gates are either an addition gate or a product gate. The size $|C|$ of a circuit C is the number of gates in C. The depth $\Delta(C)$ of a circuit C is the length of a longest directed path in C. Denote by $\mathbb{N}$ and $\mathbb{C}$ the naturals and the complex numbers, respectively.

Computing the coefficient of some special monomial can be abstracted as the following Coefficient Interpolation problem which can be solved efficiently based on the discrete Fourier transform.

Definition 6 (Coefficient Interpolation). *Given an arithmetic circuit C over $(\mathbb{N}, +, *)$ and variables $x_1, x_2, \cdots, x_\beta$ over $\mathbb{N}$ that computes a polynomial $P(x_1, \ldots, x_\beta)$, together with β integers $t_1, t_2, \ldots, t_\beta$, the problem is to compute the coefficient of the term $x_1^{t_1} x_2^{t_2} \cdots x_\beta^{t_\beta}$.*

Denote the coefficients of P as a β-dimensional array $\{c_{n_1,n_2,\ldots,n_\beta}\}$. Furthermore, assume that the coefficients of any polynomial outputted by a gate of C are bounded by m and any exponent of x_i in the terms of P is bounded by $N_i - 1$ for $i \leq \beta$. Note that P can also be interpreted as a polynomial over the complex numbers. The discrete Fourier transform of the β-dimensional array $\{c_{n_1,n_2,\ldots,n_\beta}\}$ is the array $\{C_{n_1,n_2,\ldots,n_\beta}\}$ given by the following formula

$$\begin{aligned} C_{k_1,k_2,\ldots,k_\beta} &= \frac{1}{N_1 N_2 \cdots N_\beta} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_\beta=0}^{N_\beta-1} c_{n_1,n_2,\ldots,n_\beta} \prod_{i=1}^{\beta} \omega_{N_i}^{k_i n_i} \\ &= P(\omega_{N_1}^{k_1}, \omega_{N_2}^{k_2}, \ldots, \omega_{N_\beta}^{k_\beta}), \end{aligned} \tag{91}$$

where ω_{N_i} is the i-th complex root of unity, given by $e^{-2\pi i/N_i}$. Then the inverse discrete Fourier transform can give the coefficient $c_{n_1,n_2,\ldots,n_\beta}$, as follows.

$$\begin{aligned} c_{t_1,t_2,\ldots,t_\beta} &= \frac{1}{N_1N_2\cdots N_\beta}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\cdots\sum_{n_\beta=0}^{N_\beta-1} C_{n_1,n_2,\ldots,n_\beta}\prod_{i=1}^{\beta}\omega_{N_i}^{-t_in_i} \\ &= \frac{1}{N_1N_2\cdots N_\beta}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\cdots\sum_{n_\beta=0}^{N_\beta-1} P(\omega_{N_1}^{n_1},\omega_{N_2}^{n_2},\ldots,\omega_{N_\beta}^{n_\beta})\prod_{i=1}^{\beta}\omega_{N_i}^{(N_i-t_i)n_i} \end{aligned} \tag{92}$$

Thus by evaluating the polynomial P for $N_1N_2\cdots N_\beta$ different arrays of values, which can be done using the arithmetic circuit C, $c_{t_1,t_2,\ldots,t_\beta}$ can be computed based on the above formula. Hence $c_{t_1,t_2,\ldots,t_\beta}$ can be computed with $O(N_1N_2\cdots N_\beta|C|)$ real multiplications and additions and only $O(\beta+|C|)$ real numbers need to be stored at any point of the calculation.

In order to overcome the problem that the real number can not be exactly stored and worked with, the binary representations of the numbers with the bits after the l most significant bits truncated will be used instead during the execution of the algorithm. Here the number l is chosen such that if rounding the resulting estimate $c^{'}$ of $c_{t_1,t_2,\ldots,t_\beta}$ to the nearest integer, $c_{t_1,t_2,\ldots,t_\beta}$ can be correctly obtained. Observe that addition of these estimates can be done in time $O(l)$, and that multiplication can be carried out in $O^*(l)$ time using a fast algorithm for integer multiplication, such as the Fürer's algorithm [29]. Hence the total time spent by the algorithm is $O^*((N_1N_2\cdots N_\beta)l)$ and the total space is $O((\beta+|C|)l)$. The following lemma makes sure the existence of such a choice for l.

Lemma 20 ([42]). *Let C be an arithmetic circuit over $(\mathbb{C},+,*)$ computing a polynomial $P(x_1,x_2,\ldots,x_\beta)$ such that (a) all constants of C are positive integers, (b) all coefficients of P are at most m, and (c) P has degree d (d) C has depth α. Let $x_1,x_2,\ldots,x_\beta\in\mathbb{C}, x_1^{'},x_2^{'},\ldots,x_\beta^{'}\in\mathbb{C}$ be such that for every i, $\|x_i\|=1$ and $\|x_i-x_i^{'}\|\le\epsilon$. Let $p^{'}$ be the result of applying the circuit C to $x_1^{'},x_2^{'},\ldots,x_\beta^{'}$ using the floating point arithmetic, truncating intermediate results after l bits. Suppose $(\epsilon+2\cdot2^{-l})(4md)^\alpha\le1$. Then $\|p^{'}-P(x_1,\ldots,x_\beta)\|\le(\epsilon+2\cdot2^{-l})(4md)^\alpha$.*

For a fixed value of l one can compute for every $i\le\beta$ an estimate of ω_{N_i} whose distance ϵ to ω_{N_i} is at most $2{\cdot}2^{-l}$. Now, given $n_1,\ldots,n_\beta,t_1,\ldots t_\beta,N_1,\ldots,N_\beta$, based on the circuit C for computing P, a circuit $C^{'}$ can be constructed for computing

$$P^{'}=P(x_1^{n_1},x_2^{n_2},\ldots,x_\beta^{n_\beta})\prod_{i=1}^{\beta}x_i^{(N_i-t_i)n_i}. \tag{93}$$

Clearly, the depth of $C^{'}$ is $\alpha+\log N+\log\beta$, where $N=\max_i N_i$. Furthermore the degree of $C^{'}$ is at most $d+\beta N$ and the largest coefficient is at most md.

Choose l such that $10 \cdot 2^{-l}(4(d+\beta N)md)^{\alpha+\log N+\log\beta} \leq 1$. Applying Lemma 20 on $C^{'}$ yields an estimation error for

$$P^{'}(\omega_{N_1}^{n_1}, \omega_{N_2}^{n_2}, \ldots, \omega_{N_\beta}^{n_\beta}) = P(\omega_{N_1}^{n_1}, \omega_{N_2}^{n_2}, \ldots, \omega_{N_\beta}^{n_\beta}) \prod_{i=1}^{\beta} \omega_{N_i}^{(N_i-t_i)n_i}, \tag{94}$$

which is less than $\frac{1}{2}$. The total estimation error for $c_{t_1,t_2,\ldots,t_\beta}$ is less than $\frac{N_1 N_2 \cdots N_\beta}{2N_1 N_2 \cdots N_\beta} = \frac{1}{2}$. Thus $c^{'}$ rounded to the nearest integer is $c_{t_1,t_2,\ldots,t_\beta}$. Finally, observe that $l = ((\alpha + \log d)(\log m + \log d))$, which yields the following concluding theorem.

Theorem 11. *The coefficient interpolation problem can be solved in* $O^*(N_1 \cdots N_\beta(\Delta(C) + \log d)(\log m + \log d))$ *time and* $O(\beta + |C|(\Delta(C) + \log d)(\log m + \log d))$ *space.*

Remark 1. Here it is necessary to point out that the indegree of the constructed circuit C is implicitly required to be bounded by a constant.

By transforming the problems to polynomials, the above technique can provide pseudo-polynomial time polynomial space algorithms for several hard problems, e.g., the knapsack problem.

Example 11 (Knapsack Problem). Given a set $S = \{s_1, \ldots, s_n\}$ of n items each of which has positive integer weight w_i and value v_i, and two positive integers w and v, the Knapsack problem asks whether there exists a subset $S^{'}$ of items whose total weight is at most w and total value is at most v. It may further assume that all items have weight at most w and value less than v, since otherwise this problem is trivial. The Knapsack problem can be reduced to Coefficient Interpolation as follows.

Define the polynomial $P_S(x,y) = \sum_{i=0}^{nw} \sum_{j=0}^{nv} c_{ij} x^i y^j$, where c_{ij} is the number of item sets $S^{'} \subseteq S$ whose total weight is exactly i and the total value of the items not in $S^{'}$ is exactly j. The Knapsack problem is equivalent to checking whether there exists a weight $w^{'} \leq w$ and value $v^{'} \leq (\sum_{s_i \in S} v_i) - v$ such that $c_{w^{'}v^{'}}$ is nonzero. Clearly, this can be done by solving coefficient interpolation for each pair of possible values for $w^{'}$ and $v^{'}$, but this would incur an unnecessary overhead of vw in the running time. Instead, define the polynomial $P_S^{'}(x,y) = p_S(x,y)(\sum_{i=0}^{nw} x^i)(\sum_{j=0}^{nv} y^j)$ and set $c_{ij}^{'}$ to be the coefficient of the term $x_i y_j$ in $P_S^{'}$. Then it suffices to check whether $c_{wv^*}^{'}$ is nonzero, where $v^* = \sum_{s_i \in S} v_i - v$. The remaining task is to construct a polynomial circuit $C^{'}$ for $P_S^{'}$ such that faster polynomial space algorithms can be obtained by applying Theorem 11.

Note that $P_S(x,y)$ can be fractorized into $P_S(x,y) = \prod_{i=1}^{n}(x^{w_i} + y^{v_i})$ which yields a circuit of size $O(n\log w + n\log v)$ and depth $O(\log w + \log v + \log n)$ for computing P_S. Based on the following given circuit of size and depth $O(\log n + \log w)$ for computing $\sum_{s_i \in S} x^{w_i}$ and circuit of size and depth $O(\log n + \log v)$ for computing $\sum_{s_i \in S} y^{v_i}$, a circuit $C^{'}$ of size $O(n\log w + n\log v)$ and depth $O(\log w + \log v + \log n)$ for computing $P_S^{'}$ can be constructed.

Lemma 21. *There exists a circuit of size and depth $O(\log m)$ that computes $\sum_{i=0}^{m} x^i$.*

Proof. The following recurrence can be turned into a circuit for computing $\sum_{i=0}^{m} x^i$.

$$\sum_{j=0}^{p} x^j = \begin{cases} (x^{p/2}+1)\sum_{j=0}^{p/2} x^j, & \textit{if } p \textit{ is even} \\ x(x^{\lfloor p/2 \rfloor}+1)\sum_{j=0}^{\lfloor p/2 \rfloor} x^j + 1, & \textit{otherwise.} \end{cases} \tag{95}$$

To make the circuit it needs to construct gates evaluating $x^{\lfloor p/2 \rfloor}, x^{\lfloor p/4 \rfloor}, x^{\lfloor p/8 \rfloor}$, etc. This can be done using an identical trick to the one above by observing that $x^p = (x^{p/2})^2$ if p is even and $x^p = x(x^{\lfloor p/2 \rfloor})^2$ if p is odd. □

Finally, note that the coefficients of p_S are at most 2^n and hence the coefficients of $P_S^{'}$ are at most $2^n wvn^2$. Also note that the degree of $P_S^{'}$ is at most $2nv + 2nw$. Then applying Theorem 11, one can get an algorithm for the Knapsack problem with time complexity $O^*(n^4 vw(\log v + \log w)^2)$ and space complexity $O^*(n^2(\log v + \log w)^2)$. □

3.2.2 Combination with Subset Convolution

By introducing some special gates (e.g., the subset convolution gate and the covering product gate introduced in Section 2.2) in the constructed polynomial circuits, some polynomial space algorithms for many other problems can be derived, e.g., the TSP problem, the weighted Steiner tree problem and the weighted set cover problem. Based on the polynomial space result for the subset convolution and the covering product (Theorem 7 in Section 2.5.1) and using the embedding technique (introduced in Section 2.2), the following result can be obtained by Theorem 11. Whether some other operations and a wider range of input functions can be involved in the polynomial circuit framework to give polynomial space algorithms or faster exponential space algorithms for hard problems, as well as whether this approach can be applied to a wider range of problems, need further studies.

Theorem 12 ([42]). *Let C be a circuit over $\{f : 2^V \to \mathcal{M}\}$ with operation gates including subset convolution, covering product, addition and product gates. Let $\hat{C}$ be the circuit over $\{\hat{g} : 2^V \to \mathbb{N}\}$ obtained by interpreting C over $(\mathbb{N}, +, *)$, and let u be such that $u \geq h(Y)$ where h is the output of any gate of $\hat{C}$ and $Y \subseteq V$. If the input function is a singleton and is bounded by W, then it can be decided whether $f_{out}(V) \leq k$ in $O^*(2^{|V|} W \log u)$ time and $O^*(\log u \log W)$ space.*

Example 12 (Weighted Set Cover). Given a set U, a family $\mathcal{F} = \{S_1, \ldots, S_l\} \subseteq 2^U$, a weight function $w : \mathcal{F} \to \mathbb{N}$, and an integer t. A set cover of U is a family $\mathcal{C} \subseteq \mathcal{F}$ such that $U \subseteq \sum_{S \in \mathcal{C}} S$. The weight $w(\mathcal{C}) = \sum_{S \in \mathcal{C}} w(S)$. The weighted set cover problem is to decide whether there exists a set cover of U with weight at most t. Define $m_i(Y)$ as the minimum weight of a set cover $\mathcal{C}$ of Y such that

$|\mathcal{C}| = i$. Then the problem is to determine whether $m_n(U) \leq t$. $m_i(Y)$ can be computed using the following recurrence:

$$m_i(Y) = \begin{cases} 0, & if\ i = 0, Y = \emptyset \\ \infty, & i = 0, Y \neq \emptyset \\ \min_{1\leq j\leq l} m_{\lceil i/2\rceil}(Y) * m_{\lfloor i/2\rfloor}(Y) * [Y \subseteq S_j], & otherwise, \end{cases} \tag{96}$$

where $*$ denotes the subset convolution operation. By replacing the min operation by max and making use of the embedding technique (in Section 1) to turn the operations over the max-sum semiring to be done over the product-sum ring, it can be proved that the input function is a singleton and the above recurrence can be turned into a circuit of depth $O(\log |U|)$. Applying Theorem 12, a polynomial space algorithm with running time $O^*(2^{|U|}W)$ can be obtained, where W is the maximum weight assigned to sets in $\mathcal{F}$.

4 Conclusion

This chapter summarizes some recently resurrected or newly developed combinatorial and algebraic techniques for designing either faster or space efficient exact exponential time algorithms for NP-hard problems. Detailed examples to illustrate these techniques have been given. Despite significant progresses in the past few years, the area is still largely unexplored with many open problems, and new techniques for solving these problems are very much in need. For example, can the graph coloring problem be exactly solved in time faster than $O^*(2^n)$? Can a deterministic $O^*(c^n)$ time exact algorithm be derived for TSP where $c < 2$? It appears that using the inclusion-exclusion technique alone can not break the $O^*(2^n)$ barrier of the graph coloring problem since it needs to go through all the subsets. As Example 12 shows, one promising avenue along which to design either faster or space efficient exact algorithms is to combine multiple techniques. For graph coloring, by combining the combinatorial methods and the algebraic methods, an $O^*(c^n)$ $(c < 2)$ time exact exponential time algorithm might eventually be obtained. More specifically, it is worthwhile to try applying some efficient combinatorial computing tools such as fast zeta transform in an algebraic framework. Furthermore, as Alan Perlis once said [41], "*for every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run*", exponential time exact algorithms with a small constant base would be preferable to polynomial time algorithms for some problem instances of moderate or reasonably large sizes. For example, an $O^*(1.01^n)$ exact algorithm may run much faster than a polynomial time algorithm with running time $O(n^3)$ for relatively large values of n, such as 2000. Besides designing faster exact algorithms, designing space efficient (in particular polynomial or even linear space) exact algorithms is also very important since compared to exponential time consumption, exponential space consumption could be more costly from the standpoint of computing resource usages. All in all, developing practical polynomial space exact algorithms for real-life use will need more devoted effort in the future. Although some of the open problems listed in Woeginger's survey [55] have been solved, many other ones remain open, some of which can be found in [56, 57].

5 Notes

The fast zeta transform comes from Yates's work [58]. The general theory of Möbius inversions on posets is developed by Rota [51]. Björklund et al. introduced the linear space fast zeta transform in [19]. A variant of fast zeta transform—trimmed zeta transform is introduced in [16], which is used to give faster exact algorithms for the important algorithmic tool Disjoint Sum in the design of parameterized algorithms [18]. For details on Yates's algorithm and Möbius inversion, please refer to Knuths's book [38] and Fedor and Kratsch's book [27].

The fast algorithm for subset convolution comes from Björklund et al. [14]. There is also an FFT based implementation for the fast subset convolution [27]. The fast algorithms for covering product and intersecting covering product are introduced by Björklund et al. [14], whereas the proof of Lemma 2 comes from Nederlof's paper [44]. The exact algorithms for Examples 1, 2, and 3 are due to Björklund et al. [14]. The fast subset convolution has also been used to derive faster exact algorithms for counting spanning forests [44], computing Tutte polynomial [15], computing f-width and rank-width [46]. For more information on applications of the fast subset convolution, please refer to [27]. The polynomial space algorithms for the covering product and the subset convolution are based on Nederlof's work [44] and Lokshtanov and Nederlof's work [42]. The exact algorithm for computing cover polynomial in Example 6 is due to [44]. Subset convolution can also be used to derive faster parameterized algorithms. For example, van Rooij, Bodlaender, and Rossmanith [49] gave an $O(2^t n)$ time parameterized algorithm for counting perfect matchings in graphs of treewidth at most t based on a generalized fast subset convolution algorithm. Vassilevska and Williams [54] introduced a new approach for computing permanent of rectangular matrix based on the fast subset convolution.

The application of inclusion-exclusion to combinatorial optimization goes back to Ryser's formula for the permanent [52], which remains the most effective way to count the number of matchings in a bipartite graph. The first explicit reference to combinatorial optimization is for TSP by Kohn, Gottlieb, and Kohn [39], and a concise paper by Karp [36] applying the idea to a number of hard problems. Karp's paper is very compact and has probably gone slightly unnoticed, as it focuses on cases where the technique offers reduction in space as compared to a dynamic programming algorithm, but at the expense of a possible slight increase in running time. Later, Bax and Franklin [3, 4, 6] used similar methods to count paths and cycles in general graphs. These techniques provide an $O^*(2^n)$ time algorithm for counting Hamiltonian paths [36, 4].

The proof of the inclusion-exclusion principle comes from Björklund et al.'s paper [13]. Exact algorithms for the set partition problem in Example 4 and for computing cover polynomial in Example 6 are due to Björklund et al. [13, 15]. In the conference versions [11, 40] of [13] and some other papers [12, 32, 33, 31], exact algorithms using inclusion-exclusion for other partitioning and covering problems are given. The inclusion-exclusion technique is also used to give a faster exact algorithm for computing the permanent of a matrix over rings and finite commutative semirings [20]. The polynomial space exact algorithm for counting Hamiltonian paths in Example 7 is due to Karp [36]. The inclusion-exclusion approach for counting subgraph isomorphisms using homomorphism as demonstrated in Example 8 is developed by Amini, Fomin and Saurabh [1]. Nederlof [44] further developed inclusion-exclusion techniques to derive a number of polynomial space algorithms. Additionally, the approach based on a combination of branching and inclusion-exclusion is developed in [45, 50, 48, 47]. For the recent progress on the algorithmic uses of inclusion-exclusion, please refer to the survey by Husfeldt [34].

The technique used for deriving faster exact algorithms in bounded graphs is introduced by Björklund et al. in [16, 17]. The exact algorithm for TSP in bounded graphs in Example 10 comes from [17].

The application of algebraic methods in designing exact algorithms can be traced back to the famous paper by Kohn et al. [39] on solving the traveling salesman problem using generating polynomials. Björklund introduced the algebraic sieving method in some recent papers [9, 10]. The exact algorithms for k-dimensional matching and Hamiltonicity come from [9] and [10], respectively. The polynomial circuit based approach for deriving polynomial space exact algorithms is developed by Lokshtanov and Nederlof in [42]. Faster polynomial space exact algorithms for subset sum were also proposed in the same paper, and by combining with subset convolution, they gave polynomial space exact algorithms for the Steiner tree problem as well as the TSP problem. Very recently, by using a new algebraic method based on computing the hafnian over an arbitrary ring, Björklund [8] gave a faster polynomial space algorithm for counting the number of perfect matchings.

This chapter focuses only on the recently resurrected or newly developed combinatorial and algebraic approaches for designing either faster or space efficient exact algorithms. Many recent successes in using these approaches suggest that these approaches could play a significant role in the design of exponential exact algorithms for many other NP-hard problems. Other than these approaches, there are also other classical methods for tackling NP-hard problems, such as branching and local search methods. For detailed illustrations of these classical methods, please refer to Woeginger's survey [55]. Schöning in his concise survey [53] investigated how randomization can be used to design randomized exponential time algorithms. Most recently, perhaps in response to the rapid development of exact algorithms research, Fomin and Kratsch have written an excellent text [27] on exact exponential algorithms. Both classical methods and newly developed techniques such as measure and conquer are covered by this book. The book discusses also combinatorial methods for which our chapter tries to fill in some of missing details. On top of that, this chapter demonstrates how to employ these techniques to design space efficient exact algorithms without increasing the running time, as well as how the newly developed algebraic approaches can be utilized to design either faster or space efficient exact algorithms.

6 Cross-References

For more details on dynamic programming, please refer to Chapter [Advanced Techniques in Dynamic Programming]. Additional information about combinatorial problems on coloring, dominating set and hypergraph Matching can be found in Chapters [On Coloring Problems,Combinatorial Optimization Problems on Hypergraph Matchings: A Probabilistic Analysis, Algorithmic Aspects of Domination in Graphs,Variations of Dominating Set Problem].

7 Acknowledgements

This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00302, the National Natural Science Foundation of China Grant 61103186, 61073174, 61033001, 61061130540, the Hi-Tech research and Development Program of China Grant 2006AA10Z216, and Hong Kong RGC-GRF grants 714009E and 714311.

Recommended Readings

1. O. Amini, F. V. Fomin, S. Saurabh. Counting Subgraphs via Homomorphisms. *ICALP* (1):71-82, 2009.
2. L. Babai, W. M. Kantor, and E. M. Luks, Computational complexity and the classification of finite simple groups, *FOCS*, pages 162-171, 1983.
3. E. T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Information Processing Letters*, 47(4):203-207, 1993.
4. E. T. Bax. Algorithms to count paths and cycles. *Information Processing Letters*, 52(5):249-252, 1994.
5. R. Bellman. Bottleneck problems and dynamic programming. *Proc. Natl. Acad. Sci.* 39, 947-951, 1953.
6. E. T. Bax and J. Franklin. A finite-difference sieve to count paths and cycles by length. *Information Processing Letters*, 60(4):171-176, 1996.
7. R. Bellman. *Dynamic Programming*, Princeton University Press, Princeton, 1957.
8. A. Björklund. Counting perfect matchings as fast as Ryser. *SODA*, pages 914-921, 2012.
9. A. Björklund. Exact covers via determinants. *STACS*, pages 95-106, 2010.
10. A. Björklund. Determinant sums for undirected Hamiltonicity. *FOCS*, pages 23-26, 2010.
11. A. Bjorklund, T. Husfeldt. Inclusion-exclusion algorithms for counting set partitions. *FOCS*, pages 575-582, 2006.
12. A. Björklund, T. Husfeldt. Exact Algorithms for Exact Satisfiability and Number of Perfect Matchings. *Algorithmica* 52(2): 226-249, 2008.

13. A. Bjorklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546-563, 2009.
14. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto. Fourier meets möbius: fast subset convolution. *STOC*, pages 67-74, 2007.
15. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto. Computing the tutte polynomial in vertex-exponential time. *FOCS*, pages 677-686, 2008.
16. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto. Trimmed moebius inversion and graphs of bounded degree. *Theory Comput Syst.* 47:637-654, 2010.
17. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto. The traveling salesman problem in bounded degree graphs. *ICALP*, pages 198-209, 2008.
18. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto. Counting Paths and Packings in Halves. *ESA* pages 578-586, 2009.
19. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto. Covering and Packing in Linear Space. *ICALP* (1):727-737, 2010.
20. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto. Evaluation of permanents in rings and semirings. *Information Processing Letter*, 110(20):867-870, 2010.
21. J. R. Bunch and J. E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28: 231-236, 1974.
22. F. R. K. Chung, R. L. Graham. On the cover polynomial of a digraph. *J. Comb. Theory*, Ser. B 65(2):273-290, 1995.
23. F. R. K. Chung, P. Frankl, R. L. Graham, J. B. Shearer. Some intersection theorems for ordered sets and graphs. *J. Comb. Theory*, Ser. A 43:23-37, 1986.
24. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251-280, 1990
25. S. E. Dreyfus, R. A. Wagner, The Steiner problem in graphs, *Networks* 1:195-207, 1971/72.
26. J. Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the Natural Bureau of Standards*, 71B, 4:241-245, 1967.
27. F. V. Fomin, D. Kratsch. *Exact Exponential Algorithms.* Springer, 2010
28. B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. *Theory Comput. Syst.*, 41(3):493-500, 2007.
29. M. Fürer. Faster integer multiplication. *STOC*, pages 57-66, 2007.
30. 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.
31. Q.-S. Hua, Y. Wang, D. Yu, F.C.M. Lau, Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theoretical Computer Science*, 411(26-28):2467-2474, 2010.
32. Q.-S. Hua, D. Yu, Y. Wang, F.C.M. Lau, Exact algorithms for set multicover and multiset multicover problems. *ISAAC*, 34-44, 2009.
33. Q.-S. Hua, Y. Wang, D. Yu, F.C.M. Lau, Set multi-covering via inclusion-exclusion. *Theoretical Computer Science*, 410(38-40):3882-3892, 2009.
34. T. Husfeldt. Invitation to Algorithmic Uses of Inclusion-Exclusion. *ICALP* (2), pages 42-59, 2011.
35. D. B. Johnson, Efficient algorithms for shortest paths in sparse networks, *J. ACM*, 24:1-13, 1977.
36. R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1:49-51, 1982.
37. R. Kennes. Computational aspects of the Moebius transform of a graph. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:201-223, 1991.

38. D. E. Knuth. *The art of computer programming, Vol. 3: Seminumerical algorithms*, third ednm Addison-Wesley, 1998.
39. 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*, ACM Press, pages 294-300, 1977.
40. M. Koivisto. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion, *FOCS*, pages 583-590, 2006.
41. R. Lipton. Fast exponential algorithms, `http://rjlipton.wordpress.com/2009/02/13/polynomial-vs-exponential-time/`.
42. D. Lokshtanov, J. Nederlof. Saving Space by Algebraization. *STOC*, pages 321-330, 2010.
43. R. Motwani, P. Raghavan. *Randomized algorithms*, Cambridge University Press, 1995.
44. J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: improving on Steiner tree and related problems. *ICALP*, (1):713-725, 2009.
45. J. Nederlof, J. M. M. van Rooij. Inclusion/exclusion branching for partial dominating set and set splitting. *IPEC*, pages 204-215, 2010.
46. S.i. Oum. Computing rank-width exactly. *Inf. Process. Lett.* 109(13):745-748, 2009.
47. D. Paulusma, J.M.M. van Rooij. On Partitioning a Graph into Two Connected Subgraphs. *ISAAC* pages 1215-1224, 2009.
48. J.M.M. van Rooij. Polynomial Space Algorithms for Counting Dominating Sets and the Domatic Number. *CIAC* pages 73-84, 2010.
49. J.M.M. van Rooij, H.L. Bodlaender, P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. *ESA*, pages 566-577, 2009.
50. J.M.M. van Rooij, J. Nederlof, T. C. van Dijk. Inclusion/Exclusion Meets Measure and Conquer: Exact Algorithms for Counting Dominating Sets. *ESA*, pages 554-565, 2009.
51. G.C. Rota. On the foundations of combinatorial theory. I. Theory of Möbius functions. Z. Wahrscheinlichkeitstheorie und Verw. Gebiete 2, 340-368, 1964.
52. H. J. Ryser. *Combinatorial Mathematics.* Number 14 in Carus Math. Monographs. Math. Assoc. America, 1963.
53. U. Schöning. Algorithmics in exponential time. *STACS*, pp. 36-43, 2005.
54. V. Vassilevska, R. Williams. Finding, minimizing, and counting weighted subgraphs. *STOC*, pages 455-464, 2009.
55. G. J. Woeginger. Exact Algorithms for NP-Hard Problems: A Survey. *Combinatorial Optimization*, pages 185-208, 2001.
56. G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems. *IWPEC*, pages 281-290, 2004.
57. G. J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics* 156(3): 397-405, 2008.
58. F. Yates. The design and analysis of factorial experiments. *Technical Communication No. 35, Commonwealth Bureau of Soil Science*, Harpenden, UK, 1937.