Saving Space by Algebraization[*]

Daniel Lokshtanov
University of Bergen
PB 7803, N-5020
Bergen, Norway
daniello@ii.uib.no

Jesper Nederlof
University of Bergen
PB 7803, N-5020
Bergen, Norway
jesper.nederlof@ii.uib.no

ABSTRACT

The Subset Sum and Knapsack problems are fundamental $\mathcal{NP}$-complete problems and the pseudo-polynomial time dynamic programming algorithms for them appear in every algorithms textbook. The algorithms require pseudo-polynomial time and space. Since we do not expect polynomial time algorithms for Subset Sum and Knapsack to exist, a very natural question is whether they can be solved in *pseudo-polynomial time* and *polynomial space.* In this paper we answer this question affirmatively, and give the first pseudo-polynomial time, polynomial space algorithms for these problems.

Our approach is based on algebraic methods and turns out to be useful for several other problems as well. Then we show how the framework yields polynomial space exact algorithms for the classical Traveling Salesman, Weighted Set Cover and Weighted Steiner Tree problems as well. Our algorithms match the time bound of the best known pseudo-polynomial space algorithms for these problems.

Categories and Subject Descriptors

F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics; G.2.2 [**Discrete Mathematics**]: Graph Theory

General Terms

Algorithms, Theory

Keywords

Dynamic Programming, Space Efficient, Fourier, Möbius

[*]Supported by the Norwegian Research Council

1. INTRODUCTION

In the 1950's, Richard Bellman published his book "Bottleneck Problems and Dynamic Programming" [1], initiating the systematic study of *Dynamic Programming* (DP). Nowadays, DP is considered to be one of the most prominent techniques for designing algorithms, and many algorithmic results obtained by DP are not known to be obtainable by any other technique. Conceptually, a DP algorithm can be seen as a combination of a table and an algorithm computing table entries, usually formalized as a recurrence. An inherent property of DP algorithms is that they require a relatively big amount of working memory. This is because often, many previously computed table entries are required for efficient computation of new entries. For an elementary introduction to DP, see for example [7].

In this paper we identify sufficient conditions for being able to turn DP algorithms into algorithms with roughly the same running time and significantly lower space usage. In particular, we show that if a DP algorithm can be formalized as an expression using specific operators, then one can use algebraic transforms to evaluate the expression in a space-efficient manner.

Two of the canonical applications of DP are the Subset Sum and Knapsack problems. In the Subset Sum problem one is given a set of integers $S = \{w_1, \dots, w_n\}$ and an integer w. The question is whether there exist of subset $S' \subseteq S$ such that $\sum_{w_i \in S'} w_i = w$. In the Knapsack problem one is given an additional set of values $v_1, \dots, v_n$ and an integer v. Here the objective is to find a subset S' of items whose total weight is at most w and total value is at least v. The Subset Sum problem is the special case of Knapsack where $w = v$ and all items have weight equal to value. The classical $O(nw)$ time[1] for Subset Sum of Bellman [2] uses $O(w)$ space. Hence, the algorithm uses *pseudo-polynomial* time and space. A natural and fundamental question is whether it is possible to have a pseudo-polynomial time algorithm for Subset Sum using only polynomial space. We answer this question affirmatively by showing that Bellman's DP algorithm can be encoded as an expression with a restricted set of operators. In particular, we obtain an algorithm with running time $\tilde{\mathcal{O}}(n^3 t \log t)$ time and $\tilde{\mathcal{O}}(n^2)$ space. For the more general Knapsack problem we give an algorithm running in $\tilde{\mathcal{O}}(n^4 \log^2(vw)vw)$ time and $\tilde{\mathcal{O}}(n^2 \log(vw))$ space.

Our methodolody enables us to make many exponential-time and space DP algorithms run in polynomial space in-

[1]We make use of the $\mathcal{O}$, $\tilde{\mathcal{O}}$ and $\mathcal{O}^*$ notation which suppress factors constant, polylogarithmic in instance size, and polynomial in instance size respectively.

stead. This is useful because algorithms using both exponential time and space tend to run out of space long before they run out of time. Notable results in the direction of polynomial space exponential-time algorithms for $\mathcal{NP}$-hard problems include Karp's $\mathcal{O}^*(2^n)$ time algorithm [16] for Hamiltonian Path, Björklund, Husfeldt and Koivisto's Set Cover[2] algorithm [4] running in time $\mathcal{O}^*(2^n)$ and Nederlof's $\mathcal{O}^*(2^k)$ time algorithm for Unweighted Steiner Tree [19]. Our method unifies these resukls, and gives improved algorithms for the weighted variants of these problems. An important open problem is whether Traveling Salesman can be solved in $O^*(2^n)$ time and polynomial space [21]. We make progress towards resolving this problem by giving a polynomial space algorithm for Traveling Salesman running in $O^*(2^n w)$ time. For the two other problems, Weighted Set Cover and Weighted Steiner Tree we make similar improvements. In particular our algorithms match the time bound of the best known pseudo-polynomial space algorithms for these problems.

The paper is organized as follows: In Section 2 we introduce the necessary notation. Then we briefly explain our approach in Section 3. In 4 we give sufficient conditions to be able to save space for DP on tables indexed by integers. In 5 we give sufficient conditions to be able to save space for DP on tables indexed by subsets. Finally, in Section 6 we combine the results from the two previous sections. All the latter three sections will be concluded by a few applications.

2. PRELIMINARIES AND NOTATION

We denote the naturals, the integers and the complex numbers by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{C}$ respectively. For an integer i denote $\mathbb{N}_i$ for the set of integers $\{0, \ldots, i-1\}$, and $\mathbb{Z}_i$ for the ring of integers modulo i. For sets A and B, the set $A[B]$ is the set of all functions $f : B \rightarrow A$. One can think of $A[B]$ as a table with elements of B as keys and elements of A as values. Thus vectors of d integers are elements of $\mathbb{Z}[\mathbb{N}_d]$. For $\mathbf{v} \in A[B]$ we will use boldface for $\mathbf{v}$ if we think of $\mathbf{v}$ as a vector or table, and for $i \in B$, $\mathbf{v}[i]$ is the element of A associated with i - the i'th entry of $\mathbf{v}$. If we think of $\mathbf{f} \in A[B]$ as a function we will not use boldface for f and use $f(x)$ for f evaluated at $x \in B$. For a vector $v \in \mathbb{N}^d$, the set $\mathbb{N}_{\mathbf{v}} = \mathbb{N}_{\mathbf{v[0]}} \times \mathbb{N}_{\mathbf{v[1]}} \ldots \times \mathbb{N}_{\mathbf{v[d]}}$ where $\times$ is the cartesian product. For a set S, 2^S is the set of all subsets of S and $|S|$ is the cardinality of S.

We will instantiate vectors using $\langle$ and $\rangle$, that is, $\langle 3, 2\rangle$ is a vector with two entries whose first entry is 3 and second is 2. The $=$, $\leq$ and $<$ relations for vectors are pointwise so $\mathbf{u} \leq \mathbf{v}$ means $\mathbf{u}[i] \leq \mathbf{v}[i]$ for all i. The *absolute value* of a (complex) number a and the length of a vector $\mathbf{v}$ will be denoted by $\|a\|$ and $\|\mathbf{v}\|$ respectively.

We will be working with rings, semi-rings and generally sets that come with some operations on the elements. We let $+$ denote addition and $\cdot$ denote multiplication. For vectors, $\cdot$ means the dot product. We will often use the shorthand ab for $a \cdot b$. When working with elements of $A[B]$ where A comes with addition $(+)$ and a multiplication $(\cdot)$ operator, we define the *pointwise addition* operator $\oplus$ and the *pointwise multiplication* operator $\odot$ as follows. For $\mathbf{x}, \mathbf{y} \in A[B]$ and every $b \in B$,

$$(\mathbf{x} \oplus \mathbf{y})[b] = \mathbf{x}[b] + \mathbf{y}[b] \qquad (\mathbf{x} \odot \mathbf{y})[b] = \mathbf{x}[b] \cdot \mathbf{y}[b].$$

[2]with a polynomial number of sets

We will employ *Iverson's bracket notation* which works as follows. For a predicate b, $[b] = 1$ if b is true and 0 otherwise. When Iverson bracket notation is used in the context of a (semi-) ring $\mathcal{R}$, 1 will be $\mathcal{R}$'s identity element under multiplication and 0 will be $\mathcal{R}$'s identity under addition. We will sometimes use Iverson bracket notation to define functions. In that case x, $\mathbf{x}$ or X is implicitely assumed to be the parameter of the function. That is, for a fixed subset Y of a ground set V, $[X = Y]10$ is a function from 2^V to $\mathbb{N}$ which is 0 whenever $X \neq Y$ and 10 if $X = Y$. For sets A and B such that A has a 0 element, a *singleton* is an element $\mathbf{f}$ of $A[B]$ such that there is a $b \in B$ with $\mathbf{f}[x] = 0$ unless $x = b$.

For a set S and binary operators O_1, O_2 on S, a *circuit* C over $(S; O_1, O_2)$ is a directed acyclic graph $D = (N, A)$ with parallel arcs, such that every node of D is either a *constant* gate, O_1 gate or O_2 gate. Every node of D has indegree 0 or 2, the indegree 0 nodes are constants, and are labelled with elements of S. The indegree 2 nodes are either O_1 or O_2 gates. For an indegree 2 node, its two in-neighbours are its children. The *output* of a constant gate is the element it is labeled with. The output of an O_i gate is the result of performing O_i on the output of its two children. One gate c of C is additionally marked as the *output* gate. The output of C is the output of c. The *depth* $\Delta(C)$ of C is the size of the longest path, and the *size* of C is the size of the underlying graph. In this paper we will abuse notation, and refer by the same symbol both to a gate of C and to the output of that gate.

3. METHODOLOGY

A typical dynamic programming algorithm computes some basic tables, and then combines them using some operations in order to get an output table. Often we are only interested in a single element of the output table. In this paper we will formalize a dynamic programming algorithm as circuit C over a ring $(\mathcal{R}[S]; \oplus, O_2)$, where O_2 is a "complicated" convolution operator. Since S is very big, we are not allowed to store a single table. Thus, we cannot explicitely store the constants - the inputs to our circuit. Instead we will require the constants of our circuits to be singletons, and represent them implicitely - as the only key whose value is non-zero, together with the value associated to that key.

The main idea is to apply a linear transform on the circuit, such that the "complicated" operator O_2 is transformed into the much simpler $\odot$. This allows us to use C over $(\mathcal{R}, +, \cdot)$ to compute an entry of the transformed output table fast. Now we can obtain an entry of the original output of C by applying the inverse transformation.

4. SAVING SPACE BY THE DFT

4.1 The Discrete Fourier Transform

We recap Discrete Fourier Transform (DFT) and a few basic results about it. For every $\mathbf{N} \in \mathbb{N}^d$, the Discrete Fourier Transform is a linear transform $\mathcal{F} : \mathbb{C}[\mathbb{N}_{\mathbf{N}}] \rightarrow \mathbb{C}[\mathbb{N}_{\mathbf{N}}]$. Given $\mathbf{N}$ we let $\mathbf{t} \in \mathbb{C}^d$ so that $\mathbf{t}[j] = 2\pi\iota/\mathbf{N}[j]$, and $N = \Pi_{i=0}^{d-1}\mathbf{N}$. For a matrix $\mathbf{a} \in \mathbb{C}[\mathbb{N}_{\mathbf{N}}]$ the DFT of $\mathbf{a}$, $\mathcal{F}(\mathbf{a})$ is defined as follows.

$$(\mathcal{F}(\mathbf{a}))[\mathbf{x}] = \sum_{\mathbf{j} \in \mathbb{N}_{\mathbf{N}}} e^{(\mathbf{x} \odot \mathbf{t}) \cdot \mathbf{j}} \mathbf{a}[\mathbf{j}]$$

The inverse fourier transform $\mathcal{F}^{-1}(\mathbf{a})$ of $\mathbf{a}$ is defined as follows.

$$(\mathcal{F}^{-1}(\mathbf{a}))[\mathbf{x}] = \frac{1}{N}\sum_{\mathbf{j}\in\mathbb{N}_{\mathbf{N}}} e^{-(\mathbf{x}\odot\mathbf{t})\cdot\mathbf{j}}\mathbf{a}[\mathbf{j}]$$

The definitions of $\mathcal{F}$ and $\mathcal{F}^{-1}$ show that they both are linear transforms. That $\mathcal{F}^{-1}$ indeed is the inverse of $\mathcal{F}$ is stated in the next theorem.

THEOREM 4.1 ([7]). *For* $\mathbf{a}\in\mathbb{C}[\mathbb{N}_{\mathbf{N}}]$, $\mathcal{F}^{-1}(\mathcal{F}(\mathbf{a}))=\mathbf{a}$.

We now define the *convolution* operator which is central to our results. The convolution operator $\otimes$ takes two matrices $\mathbf{a}$ and $\mathbf{b}$ in $\mathbb{Z}[\mathbb{N}_{\mathbf{N}}]$ and produces a matrix $\mathbf{a}\otimes\mathbf{b}$ in $\mathbb{Z}[\mathbb{N}_{\mathbf{N}}]$, such that for any $\mathbf{x}$

$$(\mathbf{a}\otimes\mathbf{b})[\mathbf{x}] = \sum_{\mathbf{0}\le\mathbf{j}\le\mathbf{x}} \mathbf{a}[\mathbf{j}]\mathbf{b}[\mathbf{x}-\mathbf{j}].$$

For $\mathbf{a}$ and $\mathbf{b}\in\mathbb{Z}[\mathbb{N}_{\mathbf{N}}]$ we say that the convolution $(\mathbf{a}\otimes\mathbf{b}$ *overflows* if there are vectors $\mathbf{x}$, $\mathbf{y}\in\mathbb{N}_{\mathbf{N}}$ such that $\mathbf{a}[\mathbf{x}]\neq 0$, $\mathbf{b}[\mathbf{y}]\neq 0$ and $x+y\not<\mathbf{N}$. If the elements of $\mathbb{Z}[\mathbb{N}_{\mathbf{N}}]$ are interpreted as d-variate polynomials with integer coefficients, convolution is equivalent to polynomial multiplication in coefficient form. From that viewpoint an overflow happens when the multiplication of two polynomials creates monomials of too high degree. It should be noted that the $\otimes$ operator generalizes to infinite tables like $\mathbb{Z}[\mathbb{N}]$ and to tables containing complex numbers like $\mathbb{C}[\mathbb{N}_{\mathbf{N}}]$ in the natural way. The next theorem is the reason why the DFT is useful for our purposes. The theorem states that the DFT translates the "complicated" $\otimes$ operator into $\odot$ which is much easier to work with. This comes in handy, as the dynamic programming algorithms for SUBSET SUM and KNAPSACK can be encoded using the $\oplus$ and the $\otimes$ operators.

THEOREM 4.2 (CONVOLUTION THEOREM [7]). *For* $\mathbf{a}$, $\mathbf{b}$ *in* $\mathbb{C}[\mathbb{N}_{\mathbf{N}}]$ *such that* $\mathbf{a}\otimes\mathbf{b}$ *does not overflow,* $\mathcal{F}(\mathbf{a}\otimes\mathbf{b}) = \mathcal{F}(\mathbf{a})\odot\mathcal{F}(\mathbf{b})$.

4.2 Numeric DP Without Tables

In this section we set up a general framework for turning pseudo-polynomial time and space algorithms that do a certain kind of dynamic programming on large d-dimensional tables into pseudo-polynomial time and space algorithms. We then show how our framework applies to the well-known algorithms for SUBSET SUM and KNAPSACK.

THEOREM 4.3. *Let* $\mathbf{N}\in\mathbb{N}^d$, $N=|\mathbb{N}_{\mathbf{N}}|$ *and* C *be a circuit over* $(\mathbb{Z}[\mathbb{N}_{\mathbf{N}}];\oplus,\otimes)$ *with only singleton constants. Suppose that for any gate* $\mathbf{c}\in C$ *and any* $\mathbf{x}\in\mathbb{N}_{\mathbf{N}}$ *that* $|\mathbf{c}[\mathbf{x}]|\le m$ *and that no convolution gate overflows. Let* $\mathbf{f}$ *be the output of* C*. Then, given* $\mathbf{N}$*,* m*, and* $\mathbf{g}\in\mathbb{N}_{\mathbf{N}}$ *we can compute* $\mathbf{f}[\mathbf{g}]$ *in time* $\tilde{\mathcal{O}}((|C|N\log(N))\log(Nm)\Delta(C))$ *and space* $\mathcal{O}((|C|+\log(N))\log(Nm)\Delta(C))$.

PROOF. Since integers are also complex numbers, we can reinterpret the circuit C as a circuit over $(\mathbb{C}[\mathbb{N}_{\mathbf{N}}];\oplus,\otimes)$. From the circuit C we construct another circuit C^1 over $\mathbb{C}[\mathbb{N}_{\mathbf{N}}]$ with the same directed graph as C, but with different gates. In particular, every constant gate $\mathbf{a}\in C$ is replaced with the constant gate $\mathbf{a}^1=\mathcal{F}(\mathbf{a})$. Every convolution gate $(\otimes)$ is replaced by a pointwise multiplication gate $(\odot)$.

We prove that for any gate $\mathbf{a}\in C$ and its corresponding gate $\mathbf{a}^1$, we have $\mathbf{a}^1=\mathcal{F}(\mathbf{a})$. The proof is by induction along a topological ordering of C. For the base case, if $\mathbf{a}$ is a constant gate the equality holds by construction. For the inductive step, let $\mathbf{a}$ be a gate with children $\mathbf{b}$ and $\mathbf{c}$, and let $\mathbf{a}^1$, $\mathbf{b}^1$ and $\mathbf{c}^1$ be the corresponding gates of C^1. By the inductive hypothesis $\mathbf{b}^1=\mathcal{F}(\mathbf{b})$ and $\mathbf{c}^1=\mathcal{F}(\mathbf{c})$. If $\mathbf{a}$ is an addition gate then $\mathbf{a}^1=\mathcal{F}(\mathbf{b})\oplus\mathcal{F}(\mathbf{c})=\mathcal{F}(\mathbf{b}\oplus\mathbf{c})=\mathcal{F}(\mathbf{a})$. Finally, if $\mathbf{a}$ is a convolution gate then Theorem 4.2 implies that $\mathbf{a}^1=\mathcal{F}(\mathbf{b})\odot\mathcal{F}(\mathbf{c})=\mathcal{F}(\mathbf{b}\otimes\mathbf{c})=\mathcal{F}(\mathbf{a})$.

We will use C^1 together with the fact that $\mathbf{f}=\mathcal{F}^{-1}(\mathcal{F}(\mathbf{f}))$ in order to compute $\mathbf{f}[\mathbf{g}]$. Writing out the definition of $\mathcal{F}^{-1}$ in $\mathbf{f}=\mathcal{F}^{-1}(\mathcal{F}(\mathbf{f}))$ we obtain the following equation for $\mathbf{f}[\mathbf{g}]$.

$$\mathbf{f}[\mathbf{g}] = \frac{1}{N}\sum_{\mathbf{j}\in\mathbb{N}_{\mathbf{N}}} e^{-(\mathbf{g}\odot\mathbf{t})\cdot\mathbf{j}}\cdot(\mathcal{F}(\mathbf{f}))[\mathbf{j}] \qquad (1)$$

The algorithm computes $\mathbf{f}[\mathbf{g}]$ using Equation 1, using the circuit C^1 to compute the value of $(\mathcal{F}(\mathbf{f}))[\mathbf{j}]$ for each $\mathbf{j}$. We will now explain in detail how Equation 1 is used to compute $\mathbf{f}[\mathbf{g}]$. Before evaluating the sum in Equation 1 we compute $e^{\mathbf{t}[i]}$ and obtain $e^{-\mathbf{t}[i]}$ for every $i<d$ by taking complex conjugates.

The sum in Equation 1 is computed in the natural way - with d nested loops over the coordinates of $\mathbf{j}$. In each iteration we compute $e^{-(\mathbf{g}\odot\mathbf{t})\cdot\mathbf{j}}$ using Claim 1 below by noting that $e^{-(\mathbf{g}\odot\mathbf{t})\cdot\mathbf{j}}=\prod_{i\in\mathbb{N}_d}(e^{-\mathbf{t}[i]})^{\mathbf{g}[i]\cdot\mathbf{j}[i]}$. Thus, for each $\mathbf{j}$ we can compute $e^{-(\mathbf{g}\odot\mathbf{t})\cdot\mathbf{j}}$ by a circuit of size $\mathcal{O}(\sum_{i\in\mathbb{N}_d}\log(\mathbf{N}[i]))=\mathcal{O}(\log(N))$.

CLAIM 1. *Given* $a\in\mathbb{C}$ *and* $b\in\mathbb{N}$ *as input, we can construct a circuit* C^* *over* $(\mathbb{C};+,\cdot)$ *computing* a^b *of size* $\mathcal{O}(\log b)$ *in* $\mathcal{O}(\log b)$ *time.*

PROOF. If b is even then $a^b=a^{b/2}\cdot a^{b/2}$ while if b is odd then $a^b=a\cdot a^{(b-1)/2}\cdot a^{(b-1)/2}$. Turning these recurrences into a circuit of size $\mathcal{O}(\log b)$ with a as its only constant is routine. □

Now we turn our attention to computing $(\mathcal{F}(\mathbf{f}))[\mathbf{j}]$ using the circuit C^1. Given C^1 and $\mathbf{j}$, define $C^1_{\mathbf{j}}$ to be a circuit over $(\mathbb{C},+,\cdot)$ with the same directed graph as C^1. For every constant gate $\mathcal{F}(\mathbf{a})$ of C^1 the corresponding gate of $C^1_{\mathbf{j}}$ is $(\mathcal{F}(\mathbf{a}))[\mathbf{j}]$. Pointwise addition $(\oplus)$ gates of C^1 are replaced with $+$ gates and pointwise multiplication $(\odot)$ gates of C^1 are replaced with $\cdot$ gates in $C^1_{\mathbf{j}}$. It follows directly from the construction of $C^1_{\mathbf{j}}$ that it outputs $(\mathcal{F}(\mathbf{f}))[\mathbf{j}]$ using $|C^1|=|C|$ arithmetic operations, given that the constants of $C^1_{\mathbf{j}}$ are provided. Observe that for any singleton $\mathbf{a}=[\mathbf{x}=\mathbf{p}]v$ we have that

$$\begin{aligned}(\mathcal{F}(\mathbf{a}))[\mathbf{x}] &= \sum_{\mathbf{j}\in\mathbb{N}_{\mathbf{N}}} e^{(\mathbf{x}\odot\mathbf{t})\cdot\mathbf{j}}\cdot[\mathbf{j}=\mathbf{p}]v \\ &= v\prod_{i\in\mathbb{N}_d}(e^{\mathbf{t}[i]})^{\mathbf{p}[i]\cdot\mathbf{x}[i]}\end{aligned} \qquad (2)$$

For a constant $\mathbf{a}=[\mathbf{x}=\mathbf{p}]v$ of C we compute the corresponding constant $(\mathcal{F}(\mathbf{a}))[\mathbf{j}]$ of $C^1_{\mathbf{j}}$ using Equation 2. By Claim 1 this can be done with a circuit of size $\mathcal{O}(\log(N))$.

To summarize the algorithm - we compute $\mathbf{f}[\mathbf{g}]$ using Equation 1, using the circuit $C^1_{\mathbf{j}}$ to compute the value of $(\mathcal{F}(\mathbf{f}))[\mathbf{j}]$ for each $\mathbf{j}$. The constants of $C^1_{\mathbf{j}}$ are computed using Equation 2. If we were able to store and manipulate complex numbers exactly, correctness would follow immediately from Equations 1 and 2. Of course, the algorithm cannot store and work with complex numbers. Instead, it will work with a binary representation of the numbers, both the real and the imaginary part, truncated ℓ bits after the decimal point for a properly chosen ℓ. It can easily be verified that the absolute value of any number in an intermediate step is bounded from above by $N^2 \cdot m$. Thus an estimation of each used number in $\mathbb{C}$ will be stored using $\mathcal{O}(\log(N) + \log(m) + \ell)$ bits.

The way the algorithm handles ℓ-bit precision is that some base numbers, the integers given in the input and $e^{\mathbf{t}[i]}$ for every $i < d$ are stored with *estimation error* $2^{-\ell}$, where estimation error is the distance in the complex plane between the number represented by the bitstring, and the number the bitstring is supposed to represent. Computing $e^{\mathbf{t}[i]}$ with estimation error $2^{-\ell}$ can be done in $\mathcal{O}(\ell^3)$ time and $\mathcal{O}(\ell)$ space using the Taylor series for sin and cos and a reasonable series expansion for π, such as the one provided by Chudnovsky and Chudnovsky [6]. All other numbers are obtained by performing addition and multiplication on the base numbers, truncating intermediate results ℓ bits after the decimal point. As more and more operations are required to obtain a number, the estimation error for that number increases. Since we know that $\mathbf{f}[\mathbf{g}]$ is an integer, the algorithm simply rounds the estimation of $\mathbf{f}[\mathbf{g}]$ to the nearest integer and outputs that as the answer. This answer is correct if the estimation error of $\mathbf{f}[\mathbf{g}]$ is less than $1/2$. What remains is to show that chosing $\ell = \mathcal{O}(|C| + \log(N) + \log(m)\Delta(C))$ will yield estimation error less than $1/2$ for $\mathbf{f}[\mathbf{g}]$. Let a' and b' be estimations of a and b respectively, let $c_1' = a' + b'$ truncated ℓ bits after the decimal point and $c_1 = a + b$. Then, we have that

$$\|c_1' - c_1\| \le \|a' - a\| + \|b' - b\| + 2^{-\ell}. \tag{3}$$

Here, the $2^{-\ell}$ term comes from truncating c' ℓ bits after the decimal point. Now, suppose $c_2' = a' \cdot b'$ truncated ℓ bits after the decimal point and $c_2 = a \cdot b$. This yields the following error bound:

$$\begin{aligned} \|c_2' - c_2\| \le \quad & \|a' - a\| \cdot \|b\| + \|b' - b\| \cdot \|a\| \qquad (4) \\ + \quad & \|a' - a\| \cdot \|b' - b\| + 2^{-\ell} \end{aligned}$$

Equipped with Equations 3 and 4 we are ready to bound the error of the output of a circuit over $(\mathbb{C}, +, \cdot)$. The proof lends ideas from Knuth's analysis of the Schönhage-Strassen algorithm for integer multiplication[17].

Claim 2. *Let $\hat{C}$ be a circuit over $(\mathbb{C}; +, \cdot)$, and m and ℓ be positive integers such that for any gate $v \in \hat{C}$, $\|v\| \le m$. Suppose estimations of the constants of $\hat{C}$ are given with error at most ϵ for each constant, such that $2^{-\ell} \le \epsilon \cdot (4m)^{\Delta(\hat{C})} \le 1$. Let c be the output of $\hat{C}$ and let c' be the estimate of c obtained by evaluating $\hat{C}$ on the estimations of its constants, truncating intermediate results ℓ bits after the decimal point. Then $\|c' - c\| \le \epsilon \cdot (4m)^{\Delta(\hat{C})}$.*

Proof. We prove the statement by induction on $\Delta(\hat{C})$. For the base case, if $\Delta(\hat{C}) = 0$ then v is a constant and the statement of the claim holds. Otherwise, suppose the statement holds for all circuits of depth at most $k - 1$ and let $\hat{C}$ have depth k. Let u and v be the children of the output gate c and let $\hat{C}_u$ and $\hat{C}_v$ be the subcircuits of $\hat{C}$ rooted at u and v respectively. Let u' and v' be estimations of u and v obtained by evaluating $\hat{C}_u$ and $\hat{C}_v$ respectively on the estimations of their constants, truncating intermediate results ℓ bits after the decimal point. As $\Delta(\hat{C}_u) \le k - 1$ and $\Delta(\hat{C}_v) \le k - 1$, by the induction hypothesis we have $\|u' - u\| \le \epsilon \cdot (4m)^{\Delta(\hat{C})-1}$ and $\|v' - v\| \le \epsilon \cdot (4m)^{\Delta(\hat{C})-1}$. If c is an addition gate, Equation 3 yields

$$\|c' - c\| \le \epsilon \cdot 2(4m)^{\Delta(\hat{C})-1} + 2^{-\ell}$$

which is at most $\epsilon \cdot (4m)^{\Delta(\hat{C})}$. On the other hand, if c is a multiplication gate, Equation 4 yields

$$\|c' - c\| \le \epsilon \cdot 2m(4m)^{\Delta(\hat{C})-1} + (\epsilon \cdot (4m)^{\Delta(\hat{C})-1})^2 + 2^{-\ell}$$

which is at most $\epsilon \cdot (2m + 2)(4m)^{\Delta(\hat{C})-1}$ since

$$(\epsilon \cdot (4m)^{\Delta(\hat{C})-1})^2 \le \epsilon \cdot (4m)^{\Delta(\hat{C})-1}$$

and $2^{-\ell} \le \epsilon$. Since $m \ge 1$ this concludes the proof of the claim. $\square$

For a fixed ℓ the estimation error of $e^{\mathbf{t}[i]}$ and $e^{-\mathbf{t}[i]}$ is $2^{-\ell}$ for every $i < d$. For every $\mathbf{j}$, $e^{-(\mathbf{g} \odot \mathbf{t}) \cdot \mathbf{j}}$ is computed by a circuit of size $\mathcal{O}(\log(N))$ and absolute value 1 in every step. Hence, by Claim 2 the estimation error of $e^{-(\mathbf{g} \odot \mathbf{t}) \cdot \mathbf{j}}$ is $2^{-\ell} 4^{\mathcal{O}(\log(N))}$. Similarly, for every $\mathbf{j}$ the constants of $C^1_{\mathbf{j}}$ are computed by circuits of size $\mathcal{O}(\log(N))$ and absolute value 1 in every step, the multiplication by v can be performed at the very end of the computation of Equation 2. Thus by Claim 2 the estimation error of the constants of $C^1_{\mathbf{j}}$ is bounded from above by

$$2^{-\ell} 4^{\mathcal{O}(\log(N))} 4v \le 2^{-\ell + \mathcal{O}(\log(N) + \log(m))}.$$

Since C^1 computes $\mathcal{F}(\mathbf{a})$ for every gate $\mathbf{a}$ of C, and each of the N entries of $\mathbf{a}$ have absolute value at most m, the absolute value of the number computed by any gate of $C^1_{\mathbf{j}}$ is bounded by $m \cdot N$. Hence by Claim 2 the estimation error of $(\mathcal{F}(\mathbf{f}))[\mathbf{j}]$ is

$$2^{-\ell + \mathcal{O}(\log(N) + \log(m))} (4Nm)^{\Delta(C)} \le 2^{-\ell + \mathcal{O}(\log(Nm)\Delta(C))}.$$

By Equation 3 the estimation error of $e^{-(\mathbf{g} \odot \mathbf{t}) \cdot \mathbf{j}} \cdot (\mathcal{F}(\mathbf{f}))[\mathbf{j}]$ is $2^{-\ell + \mathcal{O}(\log(Nm)\Delta(C))}$. Since the sum in Equation 1 runs over N elements and at the end we divide by N, the total estimation error becomes $2^{-\ell + \mathcal{O}(\log(Nm)\Delta(C))}$. Choosing $\ell = \mathcal{O}(\log(Nm)\Delta(C))$ yields estimation error less than $1/2$ for $\mathbf{f}[\mathbf{g}]$.

Finally, we analyze the running time of the algorithm and bound the space used. The outer loop has N iterations, in each we evaluate at most $|C|$ circuits of size $\mathcal{O}(\log(N))$ to compute the constants of $C^1_{\mathbf{j}}$. Evaluating $C^1_{\mathbf{j}}$ takes $|C|$ arithmetic operations. Thus, the total number of arithmetic operations is $\mathcal{O}(|C|N\log(N))$ while the total number of comlpex numbers stored at any time is $\mathcal{O}(|C| + \log(N))$. The number of bits used to store each number is $\mathcal{O}(\log(Nm)\Delta(C))$. Thus the space used by the algorithm is $\mathcal{O}((|C| + \log(N))\log(Nm)\Delta(C))$. The time required to add two numbers is $\tilde{\mathcal{O}}(\log(Nm)\Delta(C))$ using a fast integer multiplication algorithm (see [11, 20]). Thus, the total running time becomes $\tilde{\mathcal{O}}((|C|N\log(N)\log(Nm)\Delta(C))$, completing the proof of the theorem. $\square$

It is worth mentioning that a result similar to Theorem 4.3 can be obtained using modular arithmetic instead of complex numbers. Due to space constraints, details are deferred to the full version of the paper.

4.3 Applications

Subset Sum

We now show how to use Theorem 4.3 to give a pseudo-polynomial time, polynomial space algorithm for the SUBSET SUM problem, defined below.

SUBSET SUM

Instance:	Set S of positive integers $w_1, \ldots, w_n$, an integer w.
Question:	Does there exist a subset $S' \subseteq S$ such that $\sum_{w_i \in S'} w_i = w$?

Notice that SUBSET SUM can trivially be solved in $\mathcal{O}(2^n)$ time and polynomial space. This has been improved to $\mathcal{O}^*(2^{n/2})$ time and space by [14] and recently to $\mathcal{O}^*(2^{0.311n})$ time and $\mathcal{O}^*(2^{0.256}n)$ space [15]. Since a $w_i > w$ cannot participate in any solution we will assume that $w_i \leq w$ for every i. Also, if $w > 2^n$ then the trivial $\mathcal{O}(2^n)$ time algorithm runs in $\mathcal{O}(w)$ time and polynomial space. For every $1 \leq i \leq n$ and $x \leq nw$ let $\mathbf{s}_i[x]$ be the number of subsets $S' \subseteq \{1, \ldots, i\}$ such that $\sum_{j \in S'} w_j = x$. The well-known $\mathcal{O}(nw)$ time, $\mathcal{O}(w)$ space dynamic programming algorithm by Bellman (see [8]) can be reformulated as the following recurrence.

$$\begin{aligned} \mathbf{s}_1 &= [x = 0] \oplus [x = w_0] && (5) \\ \mathbf{s}_i &= \mathbf{s}_{i-1} \oplus (\mathbf{s}_{i-1} \otimes [x = w_i]) \qquad \text{for } i > 1. \end{aligned}$$

We are interested to know whether $\mathbf{s}_n[w] \neq 0$. Equation 5 can easily be translated into a circuit C over $(\mathbb{Z}[\mathbb{N}_{nw+1}]; \oplus, \otimes)$ with size and depth $\mathcal{O}(n)$ and the entries of any table computed by C can be bounded by 2^n since each entry of $\mathbf{s}_i$ counts subsets of $\{1, \ldots, i\}$. Thus, applying Theorem 4.3 would already yield a pseudo-polynomial time, polynomial space algorithm for SUBSET SUM. However, rearranging the recurrence before applying Theorem 4.3 gives a slightly better running time for the algorithm. The recurrence in Equation 5 can be rearranged as follows.

$$\mathbf{s}_i = \mathbf{s}_{i-1} \otimes ([x = 0] \oplus [x = w_i]) \qquad \text{for } i > 1.$$

Expanding this equation for $\mathbf{s}_n$ yields

$$\begin{aligned} \mathbf{s}_n = &([x = 0] \oplus [x = w_1]) \otimes ([x = 0] \oplus [x = w_2]) \qquad (6) \\ &\ldots \otimes ([x = 0] \oplus [x = w_n]) \end{aligned}$$

Since the $\otimes$ operator is commutative Equation 6 can be turned into a circuit C over $(\mathbb{Z}[\mathbb{N}_{nw+1}]; \oplus, \otimes)$ of size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log(n))$, with maximum table entry bounded by 2^n. Applying Theorem 4.3 on C yields the following theorem.

THEOREM 4.4. *The* SUBSET SUM *problem can be solved in $\tilde{\mathcal{O}}(n^3 t \log t)$ time and $\tilde{\mathcal{O}}(n^2)$ space.*

Knapsack

Now we show that essentially the same approach as in the previous section can be taken to solve the KNAPSACK problem in pseudo-polynomial time using polynomial space. In KNAPSACK we are given a set of n items, each with a weight and a value. We have to select a set of items with total weight at most w and total value at least v. A formal definition of the problem is given below.

KNAPSACK

Instance:	Set S of n pairs of positive integers $(v_1, w_1), \ldots, (v_n, w_n)$, two positive integers, v and w.
Question:	Is there a subset $S' \subseteq \mathbb{N}_n$ such that $\sum_{i \in S'} v_i \geq v$ and $\sum_{i \in S'} w_i \leq w$?

Similarly to the SUBSET SUM problem, there is a trivial brute force algorithm running in $\mathcal{O}(2^n)$ time and polynomial space. Also, any item with weight more than w can not participate in the solution, and any item with weight at most w and value at least v constitutes a solution by itself. Hence we will assume that $\log w \leq n$, $\log v \leq n$ and that $w_i \leq w$ and $v_i \leq v$ for every i. For every $1 \leq i \leq n$ define $\mathbf{s}_i \in \mathbb{Z}[\mathbb{N}_{\langle 2nv+1, 2nw+1 \rangle}]$, where $\mathbf{s}_i[\langle x, y \rangle]$ is the number of subsets $S' \subseteq \{1, \ldots, i\}$ such that $x + \sum_{j \in S'} v_j = v \cdot i$ and $\sum_{j \in S'} w_j = y$. That is, for every $i \geq 1$, $\mathbf{s}_i[\langle x, y \rangle]$ counts the number of subsets of the first i items with total value $(v \cdot i) - x$ and total weight y. Then, the following recurrence holds for $\mathbf{s}_i$, $1 \leq i \leq n$.

$$\begin{aligned} \mathbf{s}_1 &= [\mathbf{x} = \langle v, 0 \rangle] \oplus [\mathbf{x} = \langle v - v_1, w_1 \rangle] \\ \mathbf{s}_i &= \mathbf{s}_{i-1} \oplus (\mathbf{s}_{i-1} \otimes [\mathbf{x} = \langle v - v_i, w_i \rangle]) \qquad \text{for } i > 1. \end{aligned}$$

Just as for SUBSET SUM, this recurrence can be rearranged to yield the following formula for $\mathbf{s}_n$.

$$\begin{aligned} \mathbf{s}_n = &([\mathbf{x} = \mathbf{0} \oplus [\mathbf{x} = \langle v - v_1, w_1 \rangle]) \\ &\otimes ([\mathbf{x} = \mathbf{0} \oplus [\mathbf{x} = \langle v - v_2, w_2 \rangle]) \qquad (7) \\ &\ldots \otimes ([\mathbf{x} = \mathbf{0} \oplus [\mathbf{x} = \langle v - v_n, w_n \rangle]) \end{aligned}$$

>From Equation 7 we construct a circuit C of size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$ computing $\mathbf{s}_n$. We are interested in whether there exists an $\mathbf{x} \leq \langle nv - v, w \rangle$ such that $\mathbf{s}_n[\mathbf{x}] \neq 0$. We could have used Theorem 4.3 and to find $\mathbf{s}_n[\mathbf{x}] \neq 0$ for each $\mathbf{x} \leq \langle nv - v, w \rangle$, but that creates an unnecessary overhead of $\mathcal{O}(vw)$ in the running time. Instead we will compute

$$(\mathbf{s}_n \otimes [\mathbf{x} \leq \langle nv - v, w \rangle])[\langle nv - v, w \rangle] \qquad (8)$$

since $(\mathbf{s}_n \otimes [\mathbf{x} \leq \langle nv - v, w \rangle])[\langle nv - v, w \rangle]$ is non-zero if and only if there is a $\mathbf{x} \leq \langle nv - v, w \rangle$ such that $\mathbf{s}_n[\mathbf{x}] \neq 0$. We need a circuit $\hat{C}$ to compute $[\mathbf{x} \leq \langle nv - v, w \rangle]$. For that we will use the following observation.

OBSERVATION 4.1. *For every integer p and $p < q$ there is a circuit $\hat{C}$ of size $\mathcal{O}(\log(p))$ over $(\mathbb{Z}[\mathbb{N}_q]; \oplus, \otimes)$ computing $[x \leq p]$.*

PROOF. If p is even then $[x \leq p] = [x \leq p] \oplus [x \leq p/2] \otimes [x \leq p/2]$. If p is odd then

$$[x \leq p] = [x \leq p] \oplus ([x \leq \lfloor p/2 \rfloor] \otimes [x \leq \lfloor p/2 \rfloor]) \oplus [x = p].$$

Transforming this recurrence into a circuit of size $O(\log(p))$ is routine. □

Of course, Observation 4.1 can also be used for circuits over $(\mathbb{Z}[\mathbb{N}_{\langle 2nv+1, 2nw+1 \rangle}]; \oplus, \otimes)$ in order to to make a circuit of size $\mathcal{O}(\log(nv))$ that computes $[\mathbf{x} \leq \langle nv - v, 0 \rangle]$ and another of size $\mathcal{O}(\log(w))$ that computes $[\mathbf{x} \leq \langle 0, w \rangle]$. Now,

$$[\mathbf{x} \leq \langle nv - v, w \rangle] = [\mathbf{x} \leq \langle nv - v, 0 \rangle] \otimes [\mathbf{x} \leq \langle 0, w \rangle].$$

Hence we have a circuit $\hat{C}$ of size $\mathcal{O}(n + \log v + \log w)$ and depth $\mathcal{O}(\log(n) + \log v + \log w)$ computing $(\mathbf{s}_n \otimes [\mathbf{x} \leq \langle nv - v, w \rangle])[\langle nv - v, w \rangle]$. Also for this circuit, the table entries are bounded by 2^n since they count subsets of n-sized sets. Applying Theorem 4.3 on $\hat{C}$ yields the following result.

THEOREM 4.5. *The* KNAPSACK *problem can be solved in* $\tilde{\mathcal{O}}(n^4 \log^2(cw)vw)$ *time and* $\tilde{\mathcal{O}}(n^2 \log(vw))$ *space.*

5. SAVING SPACE BY MÖBIUS INVERSION

In this section we will prove a general theorem that identifies sufficient conditions for turning exponential space dynamic programming algorithms over the subset lattice into polynomial space algorithms based on *Möbius Inversion.* While our theorem does not yield any new polynomial space algorithms, it unifies and generalizes several well-known results, such as the HAMILTONIAN PATH algorithm by Karp [16], the UNWEIGHTED SET COVER algorithm by Björklund et al [4] and the UNWEIGHTED STEINER TREE algorithm by Nederlof [19].

5.1 Möbius Inversion

Let V be a set and let $\mathcal{R}$ be a ring. We will consider circuits over $(\mathcal{R}[2^V];\oplus,\diamond_u)$, where $\diamond_u$ is the *union product* operator [3], defined below:

$$(a \diamond_u b)(Y) = \sum_{A\cup B=Y} a(A)b(B)$$

For $f \in \mathcal{R}[2^V]$, the *zeta-transform* ζf and the *Möbius-transform* μf are defined as:

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

Now the principle of *Möbius Inversion* can be summarized as the following theorem:

LEMMA 5.1 ([5]). *For any function $f \in \mathcal{R}[2^V]$ and $Y \subseteq V$, it holds that $\mu(\zeta f)[Y] = f[Y]$.*

Recall that $\odot$ denotes pointwise multiplication. Similarly to the DFT, Möbius Inversion is particularly useful because it transforms the "hard" to compute operator $\diamond_u$ into $\odot$. The following lemma follows from the discussion in [3]. We give a proof here for completeness.

LEMMA 5.2 ([3]). *For any function $f, g \in \mathcal{R}[2^V]$ and $Y \subseteq V$, $\zeta(f \diamond_u g)[Y] = (\zeta f) \odot (\zeta g)$.*

PROOF. We have to prove that for each $Y \subseteq V$

$$\sum_{X\subseteq Y}\sum_{A\cup B=X} f[A]g[B] = \Big(\sum_{A\subseteq Y} f[A]\Big)\Big(\sum_{B\subseteq Y} f[B]\Big)$$

To see that this equation holds, notice that for each $A, B \subseteq Y$, there exists exactly one $X \subseteq Y$ such that $A \cup B = X$, hence we can sum over each combination of two subsets A and B of Y. $\square$

5.2 DP Over Subsets Without Tables

LEMMA 5.3. *Let C be a circuit over $(\mathcal{R}[2^V];\oplus,\diamond_u)$ and output s. Then, there is a polynomial time algorithm that, given $Y \subseteq V$, creates a circuit C^Y over $(\mathcal{R};+,\cdot)$ with the same underlying graph as C, such that for every gate $a \in C$ the corresponding gate in C^Y outputs $(\zeta s)[Y]$.*

PROOF. >From C we construct a circuit C' over $(\mathcal{R}[2^V];\oplus,\odot)$, by relabelling all $\diamond_u$ gates with $\odot$, and replacing every constant gate $a \in C$ with the constant gate $a' = \zeta a$. We prove that for every gate $a \in C$ and corresponding gate $a' \in C'$, $a' = \zeta a$ by induction along a topological ordering of C. If a is a constant gate, equality follows by construction. Otherwise let b and c be the children of a and b' and c' be the corresponding gates in C'. If a is a $\oplus$ gate then $a' = b' \oplus c' = \zeta b \oplus \zeta c = \zeta(b \oplus c) = \zeta a$ since ζ is a linear transform. On the other hand, if a is a $\diamond_u$ gate then $a' = b' \odot c' = \zeta b \odot \zeta c = b \diamond_u c = \zeta a$, since Lemma 5.2 yields $\zeta b \odot \zeta c = b \diamond_u c$.

Since C' only uses pointwise addition and pointwise multiplication, it can be viewed as $2^{|V|}$ disjoint circuits over $(\mathcal{R},+,\cdot)$, one circuit C_Y with output $(\zeta s)(Y)$ for each subset Y of V. For a fixed Y and constant gate $a = [X = S_a]v_a$ of C, the corresponfing constant gate $a_Y \in C_Y$ should be $(\zeta a)(Y)$ which is v_a if $S_a \subseteq Y$ and 0 otherwise. Given C and Y it is easy to construct C_Y in polynomial time. $\square$

The idea is that the value $s[V]$ in which one is typically interested can be obtained by $\sum_{X\subseteq V}(-1)^{|V\setminus X|}\zeta s[V]$, where $(\zeta s)[V]$ can computed fast using the circuit obtained by the above lemma. However, in the next subsection another operator is introduced that turns out to be more useful for our applications, and there Lemma 5.3 will be used as an intermediate result.

5.3 Subset Convolution

Let V be a ground set. In this section we will study circuits over $(\mathcal{R}[2^V],\oplus,*_\mathcal{R})$, where $*_\mathcal{R}$ is the *subset convolution* operator over $\mathcal{R}$. We proceed to formally define $*_\mathcal{R}$.

DEFINITION 5.1 ([3]). *Let V be a set, $\mathcal{R}$ be a ring and let $f, g \in \mathcal{R}[2^V]$. The operator* subset convolution $*_\mathcal{R}$ *over $\mathcal{R}$ is defined as follows: for each $Y \subseteq V$*

$$(f *_\mathcal{R} g)[Y] = \sum_{X\subseteq Y} f[X]g[Y\setminus X] \qquad (9)$$

where addition and multiplication are in the ring $\mathcal{R}$.

We consider circuits over $(\mathcal{R}[2^V],\oplus,*_\mathcal{R})$. We first embed subset convolution into the union product using the notion of relaxations, defined below.

DEFINITION 5.2. *A* relaxation *of a function $f : \mathcal{R}[2^V]$ is a sequence of functions $\{f_i : f_i \in \mathcal{R}[2^V]\}$, for $0 \le i \le |V|$, such that for every $0 \le i \le |V|$, $Y \subseteq V$:*

$$f_i[Y] = \begin{cases} f[Y] & \text{if } i = |Y| \\ 0 & \text{if } i < |Y| \end{cases}$$

We will denote the most common special case of subset convolution, being $*_\mathbb{Z}$, by $*$. Recall that $f \in \mathbb{Z}[2^V]$ is a singleton if $f = [X = S_f]v_f$ for some set S_f and integer v_f. The following theorem applies to circuits over $\mathbb{Z}[2^V]$ with pointwise addition and subset convolution.

THEOREM 5.1. *Let V be a set, and let C be a circuit over $(\mathbb{Z}[2^V],\oplus,*)$. Suppose C outputs s, all its constants are singletons, and m is an integer such that $s[V] \le m$. Then, given C and m, $s[V]$ can be computed using $\mathcal{O}^*(2^{|V|})$ time and $\mathcal{O}(|V||C|\log m)$ space.*

Before we prove this theorem, we first prove the following Lemma, which is also used in Section 6.

LEMMA 5.4. *Let V be a set and let C be a circuit over $(\mathcal{R}[2^V];\oplus,*_\mathcal{R})$. Suppose C outputs s and all its constants are singletons. Then there exists a relaxation $\{s_i\}$ of s and a circuit C' over $(\mathcal{R}[2^V];\oplus,\diamond_u)$ outputting $s_{|V|}[V]$ with $|C'| = |C||V|$, that can be constructed in polynomial time.*

PROOF. Let $|V| = n$. We construct C' by replacing every gate of $a \in C$ by n gates $a_1, \ldots, a_n \in C'$. For every constant gate $a = [X = S_a]v_a$, we set $a_i = 0$ if $i < |S|$ and $a_i = a$ otherwise. Notice that this means that all constant gates of C' are singletons. If a is not a constant gate, let b and c be the children of a in C. If $a = b \oplus c$, then for every i we set $a_i = b_i \oplus c_i$. Finally, if $a = b *_{\mathcal{R}} c$ we set $a_i = \sum_{j=0}^{i}(b_i \diamond_u c_{i-j})$. Clearly, the above procedure can be executed in polynomial time.

We prove that for every gate a, $\{a_i\}$ is a relaxation of a by induction along a topological order of C. If a is a constant gate this follows by construction. Otherwise, let b and c be the children of a. Then $\{b_i\}$ and $\{c_i\}$ are relaxations of b and c respectively by the induction hypothesis. If a is an addition gate ($a = b \oplus c$), then $\{a_i\}$ is a relaxation of a since

- for $i = |X|$, $a_i[X] = b_i[X] + c_i[X] = b[X] + c[X]$ since $b_i[X] = b[X]$ and $c_i[X] = c[X]$ by the induction hypothesis.
- for $i < |X|$, $a_i[X] = b_i[X] + c_i[X] = b[X] + c[X] = 0$ since both $b_i[X]$ and $c_i[X]$ are 0 by the induction hypothesis.

On the other hand if a is a subset convolution gate ($a = b *_{\mathcal{R}} c$), We have that for each $0 \leq i \leq n$,

$$a_i[X] = \sum_{j=0}^{i} \sum_{Y \cup Z = X} b_j[Y]c_{i-j}[Z].$$

Consider a summand $b_j[Y]c_{i-j}[Z]$. There are two cases:

- ($|X| > i$) Since $Y \cup Z = X$ we have that $|Y| + |Z| \geq |X| > i$. Now either $|Y| > j$ and $b_j[Y] = 0$, or $|Z| > i - j$ and $c_{i-j}[Z] = 0$, because $\{b_i\}$ and $\{c_i\}$ both are relaxations of b and c respectively. Then we have that $a_i[X] = 0$.
- ($|X| = i$) If $|Y| + |Z| > i$, either $|Y| > j$ or $|Z| > i - j$ and $a_j[A]b_{i-j}[B] = 0$ analogously to the first case. If $|Y| + |Z| = i$ then Y and Z are disjoint. Since only these pairs Y, Z will contribute to the sum, we match the definition of subset convolution and hence $a_i[X] = a[X]$.

Thus $\{a_i\}$ is a relaxation of a for all gates a, concluding the proof.

□

PROOF OF THEOREM 5.1. Note that C can as well be considered as a circuit over the ring $\mathbb{Z}_{m+1}$ since $s[V] \leq m+1$. Combining Lemma 5.4 and 5.3 there is a relaxation $\{s_i\}$ of s such that for every $Y \subseteq V$ we can construct a circuit C_Y over $(\mathbb{Z}_{m+1}; +, \cdot)$ of size $|V||C|$ outputting $(\zeta s_n)[Y]$. Möbius Inversion states that

$$s_n[V] = \sum_{Y \subseteq V} (-1)^{|V \setminus Y|}(\zeta s_n)[Y]$$

and now the theorem follows, since for every $Y \subseteq V$, we can evaluate C^Y in order to compute $s_n[Y]$ in $|C|$ operations in $\mathbb{Z}_{m+1}$ and $\mathcal{O}(|V||C| \log m)$ space. Using a fast integer multiplication algorithm (for example [11]), multiplication in $\mathbb{Z}_{m+1}$ takes $\mathcal{O}(\log m \log \log m \log^* m)$ time, which is $\mathcal{O}^*(1)$. This completes the proof of the theorem. □

5.4 An Application of Theorem 5.1

Now we will give an application of Theorem 5.1. In particular we will show that with our framework, the polynomial space algorithm of Nederlof [19] for UNWEIGHTED STEINER TREE follows directly from the Dreyfus-Wagner [9] recurrence. UNWEIGHTED STEINER TREE is defined as follows.

UNWEIGHTED STEINER TREE

Instance:	A graph $G = (V, E)$, $T \subseteq V$ with $|T| = k$ and an integer $t \leq |V|$.
Question:	Does there exist a subtree (V', E') of G such that $V' \leq t$ and $T \subseteq V'$?

A dynamic programming algorithm for this problem was given already in 1972 by Dreyfus and Wagner [9]. This algorithm uses $\mathcal{O}^*(3^k)$ time and $\mathcal{O}^*(2^k)$ space. In 2007, Björklund et al. [3] gave an improved algorithm using $\mathcal{O}^*(2^k)$ time and space. Finally, in 2009, Nederlof [19] gave a $\mathcal{O}^*(2^k)$ time and polynomial space algorithm. We show how the last result can be obtained from the first result using Theorem 5.1. Slightly reformulated, the dynamic programming of [9] can be summarized as follows. For every $1 \leq i \leq n$ and $v \in V$ define $f_i^v \in \mathbb{Z}[2^T]$ as

$$\begin{aligned} f_v^1 &= [X = \emptyset] && \text{for } v \notin T \\ f_v^1 &= [X = \{v\}] + [X = \emptyset] && \text{for } v \in T \qquad (10) \\ f_v^i &= \sum_{j=1}^{i-1} \sum_{w \in N(v)} f_w^j * f_v^{i-j} && \text{for } i > 1 \end{aligned}$$

The crux is that for every i, $f_v^i[X] > 0$ if and only if there exists a subtree (V', E') of G such that $|V'| \leq i$ and $X \cup \{v\} \subseteq V'$. This can be proved by induction on i: For $i = 1$ there exists such a subtree if and only if $X \subseteq \{v\}$ and $v \in T$ or $X = \emptyset$. For $i > 1$, note that there is a subtree (V', E') of the graph G such that $|V'| \leq i$ and $X \cup \{v\} \subseteq V'$ if and only if there are two subtrees $(A \cup \{v\}, E_1)$, $(B \cup \{w\}, E_2)$ such that $A \cup B \cup \{v, w\} = V'$, $|V_1| + |V_2| \leq i$ and $(v, w) \in E'$.

The instance of Unweighted Steiner Tree problem is a yes-instance if and only if $f_v^t[T] > 0$, where $v \in T$. Now we can apply Theorem 5.1 to compute $f_v^t[T]$. It is easy to see that the above recurrence implies a circuit with singleton constants, pairwise addition and convolution operators. Moreover, it can be easily be proved with induction on i that $f_v^i[X] \leq |V|^{3i} 2^{|V|i}$ for every $X \subseteq T$ and $v \in V$, hence $f_v^t[T] \leq |V|^{3t} 2^{|V|t}$. Thus, using $m = |V|^{3t} 2^{|V|r}$ we obtain:

THEOREM 5.2 ([19]). *The Unweighted Steiner Tree problem can be solved in $\mathcal{O}^*(2^k)$ time and polynomial space.*

6. COMBINING DFT AND MÖBIUS TO SAVE SPACE

In this section we will combine the tools introduced in the previous two sections to obtain new polynomial space algorithms for several minimization problems. We obtain an analogue of Theorem 5.1 in the case where $\mathcal{R}$ is the min sum semi-ring $\mathcal{M}$ consisting of the set $\mathbb{N} \cup \infty$ with operators min and $+$. Notice that the identity of min is ∞ and the identity of $+$ is 0.

Evidently, Möbius Inversion is not applicable in this case since the addition operator of $\mathcal{M}$, minimization, does not have the inverse which is required for Möbius inversion. A commonly used solution to this is to embed $\mathcal{M}$ in the ring

$\mathbb{Z}[\mathbb{N}]$ with $\oplus$ and $\otimes$ as operators. Recall that $\oplus$ is pairwise addition $\otimes$ is convolution. We represent $a \in \mathcal{M}$ by $\mathbf{a}' \in \mathbb{Z}[\mathbb{N}]$ such that $\mathbf{a}[i] > 0$ for $i = a$ and $\mathbf{a}[i] = 0$ for $i < a$. Let $\mathbf{b}'$ represent b in a similar manner. Then we have that if

$$\min\{a, b\} = c \quad a + b = d \quad \mathbf{a}' \oplus \mathbf{b}' = \mathbf{c}' \quad \mathbf{a}' \otimes \mathbf{b}' = \mathbf{d}'$$

then c and d are exactly the minimum i such that $\mathbf{c}'[i] > 0$ and the minimum j such that $\mathbf{d}'[j] > 0$, respectively.

6.1 Minimization DP Without Tables

For dynamic programming algorithms, the special case of subset convolution $*_\mathcal{R}$ where $\mathcal{R} = \mathcal{M}$ is particularly useful. We refer to this special case as *min sum subset convolution.* Recalling Definition 5.1, $*_\mathcal{M}$ is defined as

$$(\mathbf{f} *_\mathcal{M} \mathbf{g})[X] = \min_{W \subseteq X} \mathbf{f}[W] + \mathbf{g}[X \setminus W]$$

For $\mathbf{f}$, $\mathbf{g} \in \mathbb{N}[2^V]$, the *pointwise minimization* operator min is defined as $(\mathbf{f} \min \mathbf{g})[X] = \min(\mathbf{f}[X], \mathbf{g}[X])$. The *pointwise maximization* operator max is defined similarly, that is, $(\mathbf{f} \max \mathbf{g})[X] = \max(\mathbf{f}[X], \mathbf{g}[X])$.

Theorem 6.1. *Let V be a set and w be an integer. Let C be a circuit over $(\mathbb{N}[2^V], \min, *_\mathcal{M})$. Let $\hat{C}$ be obtained from C by replacing* min *with* max *gates, $*_\mathcal{M}$ with $\oplus$ gates and all constants with a table containing $w + 1$ in all entries. Suppose all constants of C are singletons, and C and $\hat{C}$ output* $\mathbf{s}$ *and $\hat{\mathbf{s}}$ respectively. Then it can be decided whether $\mathbf{s}[V] \le w$ using $\mathcal{O}^*(2^{|V|}u)$ time and polynomial space, where $\hat{\mathbf{s}}[Y] \le u$ for every $Y \subseteq V$.*

Before proving Theorem 6.1 let us remark that there is an algorithm deciding whether $\hat{\mathbf{s}}[V] \le w$ in $\mathcal{O}^*(3^n)$ time and $\mathcal{O}^*(2^n)$ space, by storing the elements of $\mathcal{M}$ for each subset and each gate and using Dynamic Programming, similarly to for example, [2, 9].

Proof of Theorem 6.1. If $u \le w$ we can return yes immediately because $\mathbf{s}[V] \le \hat{\mathbf{s}}[V] \le u \le w$. Thus we assume that $w < u$. Also, we can assume $|V| \ge 2$ since otherwise we can decide whether $u \ge \hat{\mathbf{s}}[Y]$ in polynomial time using straight-forward dynamic programming techniques as mentioned above.

Let $\mathcal{P}$ be the ring ($\mathbb{Z}[\mathbb{N}]$ with addition operator $\oplus$ and multiplication $\otimes$. Embed the min sum semi-ring $\mathcal{M}$ into $\mathcal{P}$. Specifically, we construct a circuit C^0 over $(\mathcal{P}[2^V]; \oplus, *_\mathcal{P})$ from C by replacing all min gates by $\oplus$ gates, $\otimes$ gates by $*_\mathcal{P}$, and replacing the constant gates of C with different constant gates in the following manner. For a constant gate $\mathbf{a} \in C$, labeled with $z[X = S]$ with $z \in \mathbb{N} \cup \infty$ and $S \subseteq V$, the corresponding gate $\mathbf{a}' \in C^0$ is labeled with $\mathbf{e}[X = S]$, where $\mathbf{e} \in \mathcal{P}$ is defined as the vector such that

1. $\mathbf{e}[w + 1] = 1$ and $\mathbf{e}[f] = 0$ for $f \le w$ if $z = \infty$.
2. $\mathbf{e}[z] = 1$ and $\mathbf{e}[f] = 0$ for $f \ne z$, otherwise.

Let $Y \subseteq V$, $\mathbf{a} \in C$, $\mathbf{a}' \in C'$ be the gate corresponding to $\mathbf{a}$, and i be the minimum integer such that $(\mathbf{a}'[Y])[i] > 0$. By induction on $n(c)$ combined with the discussion of the previous subsection, it follows directly that $\mathbf{a}[Y] \ge w$ if $i = w$ and $\mathbf{a}[Y] < w$ otherwise.

Let $\texttt{leq}^w$ be a circuit over $(\mathcal{P}; \oplus, *_\mathcal{P})$ outputting $\mathbf{d}$, where $(\mathbf{d}[S])[i] = 1$ if $S = \emptyset$ and $i \le t$ and 0 otherwise. Using the circuit with the same underlying graph as in Observation 4.1 (and relabeling constants of labeled with $[x = p]$ with $[x = p][X = \emptyset]$), we can make sure that $|\texttt{leq}^w|$ is $\mathcal{O}(\log w)$. We construct the circuit C^1 from C^0 as follows: add the circuit $\texttt{leq}^w$ to C^0, and let the output of C^1 be the gate $\mathbf{t} = \mathbf{d} *_\mathcal{P} \mathbf{s}$. Now observe that $(\mathbf{t}[V])[w]) > 0$ if and only if $\mathbf{s}[V] \le w$.

By Lemma 5.4 there exists a relaxation $\{\mathbf{t}_i\}$ of $\mathbf{t}$, such that for every Y we can in polynomial time construct a circuit C^2 over $(\mathcal{P}[2^V], \oplus, *_u)$ with $|C^2| = |C^1||V|$, outputting $(\zeta \mathbf{e}_n)[Y]$. This particular circuit C^2 will be related to $\hat{C}$ in the following claim. For this claim we need to introduce some notation.

For $\mathbf{p} \in \mathcal{P}$, denote $\alpha(\mathbf{p})$ for the largest i such that $\mathbf{p}[i] > 0$. Also, let $\beta(\mathbf{p})$ be the maximum value $\mathbf{p}[i]$ among all integers i. Finally for $\mathbf{a} \in C$, denote $n(\mathbf{a})$ for the number of gates $\mathbf{b} \in C$ such that there is a path from $\mathbf{b}$ to $\mathbf{a}$ in C.

Claim 3. *Suppose $|V| \ge 2$. For every gate $\mathbf{a} \in C$ with corresponding gates $\mathbf{a}_i \in C^2$ and $\hat{\mathbf{a}} \in \hat{C}$, it holds that for every $Y \subseteq V$, (i) $\alpha(\mathbf{a}_i) \le \tilde{a}$ and (ii) $\beta(\mathbf{a}_i) \le (|V|\tilde{a})^{n(\mathbf{a})}$, where $\tilde{a} = \max_{X \subseteq V} (\hat{\mathbf{a}})[X]$.*

Proof. We prove the claim by induction on $n(\mathbf{a})$. In particular, we let $\mathbf{b}, \mathbf{c} \in C$ be the in-neighbors of $\mathbf{a}$ and $\mathbf{b}_i, \mathbf{c}_i \in C^2$ and $\hat{\mathbf{b}}, \hat{\mathbf{c}} \in \hat{C}$ be corresponding gates of $\mathbf{b}$ and $\mathbf{c}$ in C^2 and $\hat{C}$ respectively. We also refer to the integers $\tilde{b} = \max_{X \subseteq V} (\hat{\mathbf{b}})[X]$ and $\tilde{c} = \max_{X \subseteq V} (\hat{\mathbf{c}})[X]$.

Suppose $\mathbf{a}$ is a constant gate ($n(\mathbf{a}) = 1$), that is $\mathbf{a} = [X = Y]e$ where $e \in \mathbb{N}$. Then $(\mathbf{a}_i[Y])[k] = [Y = S][|Y| \ge i][k = e]$. Both equations (i) and (ii) are trivially true since $\tilde{a} = e$.

Suppose $\mathbf{a}$ is a min gate, then $\mathbf{a}_i$ is a $\oplus$ gate and (i) follows from the fact that

$$\alpha(\mathbf{a}_i) = \max\{\alpha(\mathbf{b}_i), \alpha(\mathbf{c}_i)\} \le \max\{\tilde{b}, \tilde{c}\} \le \tilde{a}$$

where the inequality is due to the induction hypothesis and the latter equality is due to fact that $\hat{\mathbf{a}}$ is a max gate. For (ii), notice that

$$\beta(\mathbf{a}_i) \le \beta(\mathbf{b}_i) + \beta(\mathbf{c}_i) \le (|V|\tilde{b})^{n(\mathbf{b})} + (|V|\tilde{c})^{n(\mathbf{c})} \le (|V|\tilde{a})^{n(\mathbf{a})}$$

where the second inequality follows from the induction hypothesis and the third inequality follows from the fact that $\tilde{b}$ and $\tilde{c}$ are smaller than $\tilde{a}$, $|V| \ge 2$ and $n(\mathbf{b}) + n(\mathbf{c}) = n(\mathbf{a})$.

If $\mathbf{a}$ is a $*_\mathcal{M}$ gate, we have for every $\alpha(\mathbf{a}_i) = \alpha(\mathbf{b}_i) + \alpha(\mathbf{c}_i) \le \tilde{a}$ by the induction hypothesis and (i) follows. Also, (ii) holds since $\beta(\mathbf{a}_i)$ is at most

$$|V|\tilde{a}\beta(\mathbf{b}_i)\beta(\mathbf{c}_i) \le |V|\tilde{a}(|V|\tilde{b})^{n(b)}(|V|\tilde{c})^{n(c)} \le (|V|\tilde{a})^{n(a)}$$

for the first inequality, recall that each table entry is the result of a sum of at most $|V|\tilde{a}$ products of table entries of $\mathbf{b}_j$ and $\mathbf{c}_{i-j}$ for some $j \le i$. The second inequality is then due to the induction hypothesis, and the third inequality follows from $\tilde{b}, \tilde{c} \le \tilde{a}$ and $n(b) + n(c) + 1 = n(a)$. □

Now Claim 3 implies that for all gates $a \in C$ and $0 \le i \le n$, $\alpha(\mathbf{a}_i) \le \tilde{a}$ and $\beta(\mathbf{a}_i) \le (n\tilde{a})^n$. For the gates $\mathbf{a}$ in C^2 coming from the added circuit $\texttt{leq}^w$, we have that $\alpha(\mathbf{a}) \le w$ and $\beta(\mathbf{a}) = 1$. For every $0 \le i \le n$, $\mathbf{t}_i = \mathbf{d} \Diamond \mathbf{s}_i$, and it it follows that $\alpha(\mathbf{t}_i) \le \tilde{a} + w$ and $\beta(\mathbf{e}_i) \le (n\tilde{a})^n \tilde{a}$.

Now use Lemma 5.3 to obtain the circuit C^Y over the ring $(\mathbb{Z}[N]; \oplus, \otimes)$ such that for each $\mathbf{a}_i \in C^2$, the corresponding gate $\mathbf{a}_i^Y \in \mathbb{Z}[\mathbb{N}]$ outputs $(\zeta \mathbf{a}_i)[Y]$.

Since $\mathbf{a}_i^Y = \sum_{X \subseteq Y} \mathbf{a}_i[Y]$, it follows that $\mathbf{a}_i^Y[j] = 0$ for $j > \alpha(\mathbf{a}_i)$ and $\mathbf{a}_i^Y \le 2^n \beta(\mathbf{a}_i)$. Now Theorem 4.3 can be used on C^Y to compute $((\zeta \mathbf{t}_n)[Y])[T]$. It is easily seen that C^Y only has singleton constants, and that $\Delta(C^Y)$ is polynomial in $|C|$. Moreover, since for all gates $\mathbf{g} \in C^Y$, $\alpha(\mathbf{g}) \leq \tilde{a} + w \leq 2u$ and $\beta(g) \leq 2^n \beta(\mathbf{a_i}) \leq (n\tilde{a})^n \tilde{a} \leq (nu)^{n+1}$, we can apply the theorem with $\mathbf{N}$ being the singleton vector $\langle 2W \rangle$ and m being nu^{n+1}. Hence for any $Y \subseteq V$, $((\zeta \mathbf{t})_n[Y])[T]$ can be computed in $\mathcal{O}^*(W)$ time and polynomial space.

Finally we use Möbius inversion to obtain $(\mathbf{t}_n[V])[w]$ from all values $((\zeta \mathbf{t}_n)[Y])[w]$ for every $Y \subseteq V$. Recall $(\mathbf{t}_n[V])[w] > 0$ if and only if $s[V] \leq w$, hence we can decide whether $\mathbf{s}[V] \leq w$ in the claimed time and space bound. $\square$

6.2 Applications

We will now give a few applications of Theorem 6.1. We consider circuits using pointwise addition and the min-sum subset convolution $*_{\mathcal{M}}$. In the context of the min sum semiring, Iverson's bracket notation works as follows; $[b] = 0$ if $b = \mathit{true}$ and $[b] = \infty$ otherwise. Thus, a constant $[X = S]v$ is v if $X = S$ and $[X = S]v$ is ∞ otherwise. For a constant e we will shorthand $f *_{\mathcal{M}} [X = \emptyset]e$ as $f + e$, since min-sum subset convolution by $[X = \emptyset]e$ just adds e to all entries of f. Finally, we will denote by $\langle\langle x \rangle\rangle$ a table where all elements are x. We will only consider decision variants, but it should be noted that using binary search and standard self-reduction it is possible to extend these algorithms to construct an optimal solution, at the cost of a polynomial factor in the running time.

Traveling Salesman Problem

A Hamiltonian path of a graph is a path visiting all vertices exactly once. We study the following generalization of Hamiltonian path:

Traveling Salesman Problem (TSP)

Instance:	A graph $G = (V, E)$, a vertex s and a function $\phi : V \times V \to \{1, \ldots, d\}$ and an integer $t \leq |V|d$.
Question:	Is there a Hamiltonian path $E' \in E$ of weight at most t?

Denote $n = |V|$. Early $\mathcal{O}^*(2^n)$ time and space DP algorithms are given in [2, 13]. Later, an algorithm running in time $\mathcal{O}^*(2^n d)$ using $O^*(d)$ space was given by Karp [16]. TSP can also be solved in $\mathcal{O}^*(4^n)$ time and polynomial space [12]. Recently, [18] proposed a combination of these two approaches to obtain a space-time trade off. It is an interesting open problem ([22]) whether TSP can be solved in $\mathcal{O}(2^n)$ time and polynomial space.

For $v \in V$ and $X \subseteq V \backslash \{s, v\}$, define $\mathbf{f}_v[X]$ as the minimum weight of a Hamiltonian path in $G[X \cup \{s, v\}]$ starting in s and ending in v. The Bellman-Held-Karp recurrence [2, 13] is:

$$\begin{aligned} \mathbf{f}_v[\emptyset] &= w(s, v) \\ \mathbf{f}_v[X] &= \min_{u \in N(v) \cap X} \mathbf{f}_u[X \setminus \{v\}] + w(u, v) \end{aligned} \tag{11}$$

To see that the above equation holds, note that there only is one Hamiltonian path to consider if $X = \emptyset$ and its weight is $w(s, v)$. If X is not empty, any Hamiltonian path of $G[X \cup \{s, v\}]$ starting in s and ending in v consists of a Hamiltonian path of $G[X \cup \{s, u\}]$ starting in s and ending in u, and the edge (u, v). Hence, we can minimize over all the last edges a Hamiltonian path can have, and find the minimum weight such a Hamiltonian path can have using previously computed values and the value $w(u, v)$.

In order to turn the recurrence 11 into a circuit, we formulate it using min sum subset convolution

$$\begin{aligned} \mathbf{f}_v[\emptyset] &= w(s, v) \\ \mathbf{f}_v[X] &= \min_{u \in N(v)} (\mathbf{f}_u *_{\mathcal{M}} [X = \{u\}])[X] + w(u, v) \end{aligned}$$

but the second case of this recurrence is *cyclic*, hence it can not be translated into a circuit. However, it is easy to see we can instead use the following recurrence:

$$\begin{aligned} \mathbf{f}_v^0 &= [X = \emptyset] *_{\mathcal{M}} w(s, v) \\ \mathbf{f}_v^i &= \min_{u \in N(v)} (\mathbf{f}_u^{i-1} *_{\mathcal{M}} [X = \{v\}]) + w(u, v) \qquad \text{for } 0 < i \end{aligned}$$

since $\mathbf{f}_v^n[V] = \mathbf{f}_v[V]$ for every $v \in V \setminus \{s\}$. Now we can apply Theorem 6.1. To this end, we have to construct the circuit C', which can be formulated as the following recurrence:

$$\begin{aligned} \mathbf{g}_v^0 &= \langle\langle t+1 \rangle\rangle \oplus \langle\langle t+1 \rangle\rangle \\ \mathbf{g}_v^i &= \max_{u \in N(v)} \mathbf{g}_u^{i-1} \oplus \langle\langle t+1 \rangle\rangle \oplus \langle\langle t+1 \rangle\rangle \qquad \text{for } 0 < i \end{aligned}$$

and it is easy to see that for every $X \subseteq V$, $\mathbf{g}_v^i[X] \leq 3(i+1)(t+1)$. Then Theorem 6.1 implies

Theorem 6.2. *The Traveling Salesman problem can be solved in $\mathcal{O}^*(2^n d)$ time and polynomial space.*

Weighted Steiner Tree

In this section, we will extend the example of Subsection 5.4 to a weighted version of the Steiner Tree problem, which is the following:

Weighted Steiner Tree (WST)

Instance:	A graph $G = (V, E)$ with weight function $w : E \to \{1, \ldots, d\}$, terminals $T \subseteq V$ and an integer t such that $d \leq t \leq |V|d$.
Question:	Does there exist a subtree (V', E') of G such that $\sum_{e \in E'} w(e) \leq t$ and $T \subseteq V'$?

The current fastest algorithm for this problem is due to [10], and uses $\mathcal{O}((2+\epsilon)^k n^{h(\epsilon)})$ time and space. It is an interesting question, whether there is a $\mathcal{O}^*(c^k)$ time, polynomial space algorithm for this problem. We again consider the algorithm of Dreyfus and Wagner [9], but now for the weighted case:

$$\begin{aligned} \mathbf{f}_v^1 &= [X = \emptyset] && \text{for } v \notin T \\ \mathbf{f}_v^1 &= [X = \{v\}] \min[X = \emptyset] && \text{for } v \in T \\ \mathbf{f}_v^i &= \min_{j=1}^{i-1} \min_{w \in N(v)} \mathbf{f}_w^j *_{\mathcal{M}} \mathbf{f}_v^{i-j} + w(vw) && \text{for } i > 1 \end{aligned} \tag{12}$$

Now $\mathbf{f}_v^n[T] \leq t$ if and only if there exists a subtree (V', E') of G such that $\sum_{e \in E'} w(e) \leq t$ with $T \subseteq V'$, analogously to the recurrence of subsection 5.4. We apply Theorem 6.1. To this end, we construct the circuit C', which can be formulated as the following recurrence:

$$\begin{aligned} \mathbf{g}_v^1 &= \langle\langle t+1 \rangle\rangle \\ \mathbf{g}_v^i &= \max_{j=1}^{i-1} \max_{w \in N(v)} \mathbf{g}_w^j \oplus \mathbf{g}_v^{i-j} \oplus \langle\langle t+1 \rangle\rangle \qquad \text{for } i > 1 \end{aligned}$$

Hence $\mathbf{g}_v^i[X] \leq 3(t+1)(i+1)$, yielding the following theorem.

THEOREM 6.3. *The Weighted Steiner Tree problem can be solved in $\mathcal{O}^*(2^{|T|}d)$ time and polynomial space.*

Weighted Set Cover

Our last application is for the following problem:

WEIGHTED SET COVER

Instance: A set V, a family of subsets $S_1, \ldots, S_m$, a weight function $w : \{1, \ldots, l\} \to \{1, \ldots, d\}$ and an integer $t \geq d$.

Question: Does there exist a $C \subseteq \{1, \ldots, l\}$ such that $\bigcup_{i \in C} S_i = V$ and $\sum_{i \in C} w(i) \leq t$?

The current fastest algorithm for WEIGHTED SET COVER due to [4] uses $\mathcal{O}^*(2^{|V|}d)$ time and $\mathcal{O}^*(d)$ space. In this variant of the problem, the sets are given explicitely and hence the number m is polynomial in input size. The standard dynamic programming algorithm for our the problem uses the following recurrence. For ease of notation, let $S_0 = \emptyset$, $w(0) = 0$ an for each $0 \leq i \leq n$, define $\mathbf{f}_i[X]$ as the minimum over all subsets C of $\{0, \ldots, i\}$ such that $\cup_{j \in C} S_j \supseteq X$. Then

$$\mathbf{f}^0 = [X = \emptyset]$$
$$\mathbf{f}^i = f^{i-1} *_{\mathcal{M}} \mathbf{s}^{S_i}$$

where $\mathbf{s}$ is defined as

$$\mathbf{s}^{\{e_1, \ldots e_l\}} = ([X = \{e_1\}] + [X = \emptyset]) *_{\mathcal{M}} \ldots *_{\mathcal{M}} ([X = \{e_l\}] + [X = \emptyset])$$

Note that $\mathbf{s}^W$ is 1 if $W \subseteq X$ and ∞ otherwise. Again, in order to apply Theorem 6.1 we have to consider the circuit $\hat{C}$ obtained by replacing min by max and $*_{\mathcal{M}}$ by $\oplus$:

$$\mathbf{g}^0 = \langle\langle t+1 \rangle\rangle$$
$$\mathbf{g}^i = \mathbf{g}^{i-1} \oplus \mathbf{r}^{S_i}$$
$$(\mathbf{r}^{\{e_1, \ldots e_l\}}] = \sum_{i=1}^{l} \langle\langle t+1 \rangle\rangle \max \langle\langle t+1 \rangle\rangle$$

And hence for every X $\mathbf{g}^i[X] \leq t + 1 + m|V|(t+1)$. Hence Theorem 6.1 yields the following result.

THEOREM 6.4. *The Weighted Set Cover problem can be solved in $\mathcal{O}^*(2^{|V|}d)$ time and polynomial space.*

Acknowledgements

We thank Fedor Fomin, Pinar Heggernes, Neeldhara Misra and Saket Saurabh for some useful discussions and help to improve the presentation of the paper.

7. REFERENCES

[1] R. Bellman. Bottleneck problems and dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 39(9):947, 1953.

[2] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962.

[3] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: Fast subset convolution. In *STOC*, pages 67–74, 2007.

[4] A. Bjorklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.

[5] P. J. Cameron. *Combinatorics*. Cambridge University Press, 1998.

[6] D. V. Chudnovsky and G. V. Chudnovsky. Approximations and complex multiplication according to ramanujan. *Ramanujan Revisited*, pages 375–396 and 468–472, 1988.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to algorithms, 2001.

[8] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill, 2006.

[9] S. Dreyfus and R. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[10] 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.

[11] M. Fürer. Faster integer multiplication. In *STOC*, pages 57–66, 2007.

[12] Y. Gurevich and S. Shelah. Expected computation time for hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987.

[13] 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.

[14] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974.

[15] N. Howgrave-Graham and A. Joux. A new generic algorithm for hard knapsacks (to appear in *eurocrypt* 2010).

[16] R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1:49–51, 1982.

[17] D. E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969.

[18] M. Koivisto and P. Parviainen. A Space–Time Tradeoff for Permutation Problems.

[19] J. Nederlof. Fast polynomial-space algorithms using möbius inversion: Improving on steiner tree and related problems. In *ICALP '09*, pages 713–725. Springer-Verlag, 2009.

[20] A. Schönhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.

[21] G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In *IWPEC*, pages 281–290, 2004.

[22] G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In R. G. Downey, M. R. Fellows, and F. K. H. A. Dehne, editors, *IWPEC*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004.