An FPTAS for #Knapsack and Related Counting Problems

Parikshit Gopalan* Adam Klivans† Raghu Meka† Daniel Štefankovič‡ Santosh Vempala§ Eric Vigoda§

Abstract— Given n elements with non-negative integer weights $w_1, \ldots, w_n$ and an integer capacity C, we consider the counting version of the classic knapsack problem: find the number of distinct subsets whose weights add up to at most C. We give the first deterministic, fully polynomial-time approximation scheme (FPTAS) for estimating the number of solutions to any knapsack constraint (our estimate has relative error $1 \pm \varepsilon$). Our algorithm is based on dynamic programming. Previously, randomized polynomial-time approximation schemes (FPRAS) were known first by Morris and Sinclair via Markov chain Monte Carlo techniques, and subsequently by Dyer via dynamic programming and rejection sampling.

In addition, we present a new method for deterministic approximate counting using *read-once branching programs*. Our approach yields an FPTAS for several other counting problems, including counting solutions for the multidimensional knapsack problem with a constant number of constraints, the general integer knapsack problem, and the contingency tables problem with a constant number of rows.

1. Introduction

Randomized algorithms are usually simpler and faster than their deterministic counterparts. In spite of this, it is widely believed that P=BPP (see, e. g., [3]), i.e., at least up to polynomial complexity, randomness is not essential. This conjecture is supported by the fact that there are relatively few problems for which exact randomized polynomial-time algorithms exist but deterministic ones are not known. Notable among them is the problem of testing whether a polynomial is identically zero (a special case of this, primality testing was open for decades but a deterministic algorithm is now known, [1]).

In approximation algorithms, however, there are many more such examples. The entire field of approximate counting is based on Markov chain Monte Carlo (MCMC) sampling [13], a technique that is inherently randomized and has had remarkable success. The problems of counting matchings [11], [14], colorings [10], various tilings, partitions and arrangements [18], estimating partition functions [12], [24], or volumes [7], [17] are all solved by first designing a random sampling method and then reducing counting to repeated sampling. In all of these cases, when the input is presented explicitly, it is conceivable that deterministic polynomial-time algorithms exist.

Our interest is in obtaining a *deterministic* approximation algorithm or FPTAS (fully polynomial approximation scheme) for #P-complete counting problems. We desire an algorithm that for an input instance I and a given approximation factor $\varepsilon > 0$, estimates the number of solutions for I within a relative factor $1 \pm \varepsilon$ in time polynomial in the input size $|I|$ and $1/\varepsilon$.

There are far fewer examples of deterministic approximation schemes for #P-complete problems. One of the first examples is due to Ajtai and Wigderson [2] (see also Luby and Veličković [19, Corollary 13]) who gave an algorithm for approximating the number of solutions to a DNF formula where each term has constant length. A notable recent example of an FPTAS for a #P-complete problem is Weitz's algorithm [25] for counting independent sets in graphs of maximum degree $\Delta \leq 5$. Similar approaches to Weitz's algorithm were later used for counting all matchings of bounded degree graphs [4], and k-colorings of triangle-free graphs with maximum degree Δ when $k > 2.84\Delta$ [9] .

1.1. An FPTAS for #Knapsack

Here we consider one of the most basic counting problems, namely approximately counting the number of $0/1$ knapsack solutions. More precisely, we are given a list of non-negative integer weights $w_1, \ldots, w_n$ and an integer capacity C, and wish to count the number of subsets of the weights that add up to at most C. (The decision version of this problem is NP-hard, but has a well-known pseudo-polynomial algorithm based on dynamic programming.) From a geometric perspective, for the n-dimensional Boolean hypercube, we are given as input an n-dimensional hyperplane, and our goal is to determine the number of vertices of the hypercube that lie on one side of the given hyperplane. We give an FPTAS for the problem, that is, a deterministic algorithm that for any $\varepsilon > 0$ estimates the number of solutions to within relative error $1 \pm \varepsilon$ in time polynomial in n and $1/\varepsilon$.

Motivation: Our result follows a line of work in the literature. Dyer et al. [8] gave a randomized sub-exponential time algorithm for this problem, based on near-uniform sampling of feasible solutions by a random walk. Morris and Sinclair [21] improved this, showing a rapidly mixing Markov chain, and obtained an FPRAS (fully-polynomial *randomized* approximation scheme). In a surprising development, Dyer [6], gave a completely different approach,

*Microsoft Research, Silicon Valley, Mountain View, CA 94043. Email: parik@microsoft.com.

†Department of Computer Science, UT Austin, Austin TX 78712. Email:{klivans,raghu}@cs.utexas.edu.

‡Department of Computer Science, University of Rochester, Rochester, NY 14627. Email: stefanko@cs.rochester.edu. Research supported in part by NSF grant CCF-0910415.

§School of Computer Science, Georgia Institute of Technology, Atlanta GA 30332. Email: {vempala,vigoda}@cc.gatech.edu. Research supported in part by NSF grants CCF-0830298 and CCF-0910584.

combining dynamic programming with simple rejection sampling to also obtain an FPRAS. Although much simpler, randomization still appears to be essential in his approach—without the sampling part, his algorithm only gives a factor $\sqrt{n}$ approximation.

Additionally, there has been much recent work on constructing pseudorandom generators (PRGs) for geometric concept classes, in particular halfspaces (e.g., [5], [20], [22]). It is easy to see that pseudorandom generators for halfspaces imply deterministic approximation schemes for counting solutions to knapsack constraints by enumerating over all input seeds to the generator. The estimate obtained, however, has small additive error, rather than our desired relative error. Further, the seed-lengths of these generators are too large to yield an FPTAS. Still, a clear goal of this line of research has been an FPTAS for counting knapsack solutions, which we obtain here for the first time.

Techniques: Our algorithm, like Dyer's algorithm mentioned above, is based on dynamic programming and is inspired by the pseudo-polynomial algorithm for the decision/optimization version of the knapsack problem. The complexity of the pseudo-polynomial algorithm is $O(nC)$, where C is the capacity bound. A pseudo-polynomial algorithm for the counting problem can be achieved as well using the following recurrence:

$$S(i,j) = S(i-1,j) + S(i-1,j-w_i),$$

with appropriate initial conditions. Here $S(i,j)$ is the number of knapsack solutions that use a subset of the items $\{1,\ldots,i\}$ and their weights sum to at most j.

Roughly speaking, since we are only interested in approximate counting, Dyer's idea was the following. He scales down the capacity to a polynomial in n, and scales down the weights by the same factor where the new weights are rounded down if necessary. He then counts the solutions to the new problem efficiently using the pseudo-polynomial time dynamic programming algorithm. The new problem could have more solutions (since we rounded down) but Dyer showed it has at most a factor of $O(n)$ more for a suitable choice of scaling. Further, given the exact counting algorithm for the new problem, one gets an efficient sampler, then uses rejection sampling to only sample solutions to the original problem. The sampler leads to a counting algorithm using standard techniques. Dyer's algorithm has running time $O(n^3 + \varepsilon^{-2}n^2)$ using the above approach, and $O(n^{2.5}\sqrt{\log(\varepsilon^{-1})} + n^2\varepsilon^{-2})$ using a more sophisticated approach that also utilizes randomized rounding.

To remove the use of randomness, one might attempt to use a more coarse-grained dynamic program, namely rather than consider all integer capacities $1,2,\ldots,C$, what if we only consider weights that go up in some geometric series? This would allow us to reduce the table size to $n\log C$ rather than nC. The problem is that varying the capacity even by an exponentially small factor $(1 + n/2^n)$ can change the number of solutions by a constant factor! Instead, we index the table by the prefix of items allowed and the number of solutions with the entry in the table being the minimum capacity that allows these indices to be feasible. We can now consider approximate numbers of solutions and obtain a small table. Our first result is the following:

Theorem 1.1. *Let $w_1,\ldots,w_n$ and C be an instance of a knapsack problem. Let Z be the number of solutions of the knapsack problem. There is a deterministic algorithm which for any $\varepsilon \in (0,1)$ outputs Z' such that $Z \le Z' \le Z(1+\varepsilon)$. The algorithm runs in time $O(n^3\varepsilon^{-1}\log(n/\varepsilon))$.*

The running time of our algorithm is competitive with that of Dyer. One interesting improvement is the dependence on ϵ. Our algorithm has a linear dependence on ϵ^{-1} (ignoring the logarithm term), whereas Monte Carlo approaches, including Dyer's algorithm [6] and earlier algorithms for this problem [21], [8], have running time which depends on ε^{-2}.

1.2. Counting using Branching Programs

Given our counting algorithm for the knapsack problem, a natural next step is to count solutions to multidimensional knapsack instances and other related extensions of the knapsack problem. Unfortunately, the dynamic programming based approach for knapsack does not generalize for the multidimensional case. We overcome this hurdle by presenting a different, more general FPTAS for counting knapsack solutions under a much wider class of weighted distributions. In doing so, we present a general framework for deterministic approximate counting using read-once branching programs (for a definition of the branching programs we use see Section 3.1) that we believe to be of independent interest.

It is not difficult to see that a read-once branching program of possibly exponential width can compute the set of feasible solutions of a knapsack instance. Meka and Zuckerman [20] observed that there exist small-width, read-once branching programs that approximate the set of feasible solutions for any knapsack instance to within a small additive error.

We build on this observation and show that there exist small-width, read-once branching programs for computing knapsack-type constraints to within a small relative error. Further, the approximations hold with respect to any small-space source, which is a large class of (not necessarily uniform) distributions on $\{0,1\}^n$. We then combine these ideas with the dynamic programming results of Dyer [6] to obtain an FPTAS for several other related counting problems, including counting solutions to the multidimensional knapsack problem, and counting solutions to the contingency tables problem.

In the multi-dimensional knapsack problem, we are given k knapsack instances $\left\{\left(w_1^{(j)}, w_2^{(j)}, \ldots, w_n^{(j)}, C^{(j)}\right)\right\}_{j=1}^{k}$, and the goal is to compute the number of solutions satisfying

all constraints; i.e., compute the cardinality of the set of solutions:

$$\{S \subset \{1,\ldots,n\} : \text{ for all } 1 \le j \le k,\ \sum_{i\in S} w_i^{(j)} \le C^{(j)}\}.$$

Morris and Sinclair [21] and Dyer [6] showed that their approaches to #Knapsack extend to the multidimensional problem, yielding an FPRAS when k is constant. We obtain an FPTAS for the multi-dimensional knapsack problem also when k is constant:

Theorem 1.2. *Let $\left\{\left(w_1^{(j)}, w_2^{(j)}, \ldots, w_n^{(j)}, C^{(j)}\right)\right\}_{j=1}^{k}$ be an instance of a multidimensional knapsack problem. Let $W = \sum_{i,j} w_i^{(j)} + \sum_j C^{(j)}$ and $\varepsilon > 0$. There is an* FPTAS *which for any $\varepsilon > 0$ computes a $1 \pm \varepsilon$ relative approximation of the number of solutions to the multidimensional knapsack instance in time $O((n/\epsilon)^{O(k^2)} \log W)$.*

Our algorithm works via a reduction from counting multi-dimensional knapsack solutions under the uniform distribution to solving k (one-dimensional) knapsack counting problems, but under a carefully chosen small-space distribution constructed using Dyer's results. Thus the fact that our second algorithm works for all small-space sources is crucially used.

A yet more sophisticated application of our approach yields an FPTAS for counting the number of integer-valued contingency tables. Given row sums $\mathbf{r} = (r_1,\ldots,r_m) \in \mathbb{Z}_+^m$ and column sums $\mathbf{c} = (c_1,\ldots,c_n) \in \mathbb{Z}_+^n$, let $CT(\mathbf{r},\mathbf{c}) \subseteq \mathbb{Z}_+^{m\times n}$ denote the set of integer-valued contingency tables with row and column sums given by $\mathbf{r},\mathbf{c}$:

$$CT(\mathbf{r},\mathbf{c}) = \{\, X \in \mathbb{Z}_+^{m\times n} \ :\ \sum_j X_{ij} = r_i, i \in [m],\ \sum_i X_{ij} = c_j, j \in [n] \,\}.$$

Note that, as in the case of knapsack, the magnitude of the row and column sums could be exponential in n. Dyer [6] gave an FPRAS for counting solutions to contingency tables (with a constant number of rows) based on dynamic programming. We give an FPTAS for this problem.

Theorem 1.3. *Given row sums $\mathbf{r} = (r_1,\ldots,r_m) \in \mathbb{Z}_+^m$ and column sums $\mathbf{c} = (c_1,\ldots,c_n) \in \mathbb{Z}_+^n$ with $R = \max_i r_i$ and $\epsilon > 0$, there is an* FPTAS *that computes a $(1\pm\epsilon)$-relative error approximation for $|CT(\mathbf{r},\mathbf{c})|$ in time $(n^{O(m)}(\log R)/\epsilon)^m$.*

In Section 2 we present the dynamic programming FPTAS for #Knapsack, proving Theorem 1.1. In Section 3 we define read-once branching programs and present the FPTAS for the multidimensional knapsack problem (Theorem 1.2). We also present an FPTAS for counting solutions to the general integer knapsack problem in Section 4. The proof of Theorem 1.3 is deferred to the full version of the paper.

2. Simple Dynamic Programming Algorithm for #Knapsack

In this section we present our dynamic programming algorithm. Fix a knapsack instance and fix an ordering on the elements and their weights.

We begin by defining the function $\tau : \{0,\ldots,n\}\times\mathbb{R}_{\ge 0} \to \mathbb{R}\cup\{\pm\infty\}$ where $\tau(i,a)$ is the smallest C such that there exist at least a solutions to the knapsack problem with weights $w_1,\ldots,w_i$ and capacity C. It is not clear how to compute the function τ efficiently since the second argument has exponentially many possible values (the number of solutions is an integer between 0 and 2^n). Nevertheless, τ will be used in the analysis and it is useful for motivating the definition of our algorithm.

Note that, by definition, $\tau(i,a)$ is monotone in a, that is, $a \le a' \implies \tau(i,a) \le \tau(i,a')$. For $i = 0$, using standard conventions, the value of τ is given by:

$$\tau(0,a) = \begin{cases} -\infty & \text{if } a = 0, \\ 0 & \text{if } 0 < a \le 1, \\ \infty & \text{otherwise.} \end{cases} \tag{1}$$

Note that the number of knapsack solutions satisfies: $Z = \max\{a : \tau(n,a) \le C\}$. We will show that $\tau(i,a)$ satisfies the following recurrence (we explain the recurrence below).

Lemma 2.1. *For any $i \in [n]$ and any $a \in \mathbb{R}_{\ge 0}$ we have*

$$\tau(i,a) = \min_{\alpha\in[0,1]} \max \begin{cases} \tau\big(i-1,\alpha a\big), \\ \tau\big(i-1,(1-\alpha)a\big) + w_i. \end{cases} \tag{2}$$

Intuitively, to obtain a solutions that consider the first i items, we need to have, for some $\alpha \in [0,1]$, αa solutions that consider the first $i-1$ items, and $(1-\alpha)a$ solutions that contain the i-th item and consider the first $i-1$ items. We try all possible values of α and take the one that yields the smallest (optimal) value for $\tau(i,a)$.

Proof of Lemma 2.1: Fix any $\alpha \in [0,1]$. Let $B = \max\{\tau\big(i-1,\alpha a\big), \tau\big(i-1,(1-\alpha)a\big) + w_i\}$. Since $B \ge \tau\big(i-1,\alpha a\big)$, there are at least αa solutions with weights $w_1,\ldots,w_{i-1}$ and capacity B. Similarly, since $B - w_i \ge \tau\big(i-1,(1-\alpha)a\big)$, there are at least $(1-\alpha)a$ solutions with weights $w_1,\ldots,w_{i-1}$ and capacity $B-w_i$. Hence, there are at least a solutions with weights $w_1,\ldots,w_i$ and capacity B, and thus $\tau(i,a) \le B$. To see that we did not double count, note that the first type of solutions (of which there are at least αa) has $x_i = 0$ and the second type of solutions (of which there are at least $(1-\alpha)a$) has $x_i = 1$.

We established

$$\tau(i,a) \le \min_{\alpha\in[0,1]} \max \begin{cases} \tau\big(i-1,\alpha a\big), \\ \tau\big(i-1,(1-\alpha)a\big) + w_i. \end{cases} \tag{3}$$

Consider the solution of the knapsack problem with weights $w_1,\ldots,w_i$ and capacity $C = \tau(i,a)$ that has at least a solutions. Let β be the fraction of the solutions

that do not include item i. Then $\tau(i-1,\beta a) \leq C$, $\tau(i-1,(1-\beta)a) \leq C - w_i$, and hence

$$\max\{\tau(i-1,\beta a), \tau(i-1,(1-\beta)a) + w_i\} \leq C = \tau(i,a).$$

We established

$$\tau(i,a) \geq \min_{\alpha\in[0,1]} \max \begin{cases} \tau\big(i-1,\alpha a\big), \\ \tau\big(i-1,(1-\alpha)a\big) + w_i. \end{cases} \quad (4)$$

Equations (3) and (4) yield (2). ∎

Now we move to an approximation of τ that we can compute efficiently. We define a function T which only considers a small set of values a for the second argument in the function τ; these values will form a geometric progression.

Let $Q := 1 + \frac{\varepsilon}{n+1}$ and let $s := \lceil n \log_Q 2 \rceil = O(n^2/\varepsilon)$. The function $T : \{0,\ldots,n\} \times \{0,\ldots,s\} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is defined using the recurrence (2) that the function τ satisfies. Namely, T is defined by the following recurrence:

$$T[i,j] = \min_{\alpha\in[0,1]} \max \begin{cases} T\big[i-1, \lfloor j + \ln_Q \alpha \rfloor\big], \\ T\big[i-1, \lfloor j + \ln_Q(1-\alpha) \rfloor\big] + w_i. \end{cases}$$

The second argument of T is, approximately, the logarithm (with base Q) of the second argument of τ. (Note that $\ln_Q \alpha$ and $\ln_Q(1-\alpha)$ are negative.)

More precisely, T is defined by the following algorithm **CountKnapsack**.

CountKnapsack
Input: Integers $w_1, w_2, \ldots, w_n, C$ and $\varepsilon > 0$.

1. Set $T[0,0] = 0$ and $T[0,j] = \infty$ for $j > 0$.
2. Set $Q = (1 + \varepsilon/(n+1))$ and $s = \lceil n \log_Q 2 \rceil$.
3. For $i = 1 \to n$, for $j = 0 \to s$, set

$$T[i,j] = \min_{\alpha\in[0,1]} \max \begin{cases} T\big[i-1, \lfloor j + \ln_Q \alpha \rfloor\big], \\ T\big[i-1, \lfloor j + \ln_Q(1-\alpha) \rfloor\big] + w_i, \end{cases} \quad (5)$$

where, by convention, $T[i-1,k] = 0$ for $k < 0$.

4. Let $j' := \max\{j : T[n,j] \leq C\}$. Output $Z' := Q^{j'+1}$.

Note, the recurrence (5) is over all $\alpha \in [0,1]$. However, since the second arguments (namely, $\lfloor j + \ln_Q \alpha \rfloor$ and $\lfloor j + \ln_Q(1-\alpha) \rfloor$) are step functions of α, it suffices to consider a discrete set S of α which yields all possible values of the second arguments. For $j \in \{0,1,\ldots,s\}$, the set S is $S = S_1 \cup S_2$ where: $S_1 = \{Q^{-j}, \ldots, Q^0\}$ and $S_2 = \{1 - Q^0, \ldots, 1 - Q^{-j}\}$. The set S_1 captures those α pertaining to the argument $\lfloor j + \ln_Q \alpha \rfloor$ in (5), and S_2 captures those for $\lfloor j + \ln_Q(1-\alpha) \rfloor$. By only considering this subset S of possible α we will be able to compute T efficiently.

The key fact is that T approximates τ in the following sense.

Lemma 2.2. *Let $i \geq 1$. Assume that for all $j \in \{0,\ldots,s\}$ we have that $T[i-1,j]$ satisfy (6). Then for all $j \in \{0,\ldots,s\}$ we have that $T[i,j]$ computed using (5) satisfies:*

$$\tau(i, Q^{j-i}) \leq T[i,j] \leq \tau(i, Q^j). \quad (6)$$

Proof: By the assumption of the lemma and the monotonicity of τ we have

$$T\big[i-1, \lfloor j + \ln_Q \alpha \rfloor\big] \geq \tau(i-1, Q^{\lfloor j + \ln_Q \alpha \rfloor - (i-1)}) \geq \tau(i-1, \alpha Q^{j-i}), \quad (7)$$

and

$$T\big[i-1, \lfloor j + \ln_Q(1-\alpha) \rfloor\big] \geq \tau\left(i-1, Q^{\lfloor j + \ln_Q(1-\alpha) \rfloor - (i-1)}\right) \geq \tau(i-1, (1-\alpha)Q^{j-i}). \quad (8)$$

Combining (7) and (8) with min and max operators we obtain

$$\left(\min_{\alpha\in[0,1]} \max \begin{cases} T\big[i-1, \lfloor j + \ln_Q \alpha \rfloor\big], \\ T\big[i-1, \lfloor j + \ln_Q(1-\alpha) \rfloor\big] + w_i \end{cases} \right) \geq \left(\min_{\alpha\in[0,1]} \max \begin{cases} \tau(i-1, \alpha Q^{j-i}), \\ \tau(i-1, (1-\alpha)Q^{j-i}) + w_i \end{cases} \right) = \tau(i, Q^{j-i}),$$

establishing that $T[i,j]$ computed using (5) satisfy the lower bound in (6).

By the assumption of the lemma and the monotonicity of τ we have

$$T\big[i-1, \lfloor j + \ln_Q \alpha \rfloor\big] \leq \tau(i-1, Q^{\lfloor j + \ln_Q \alpha \rfloor}) \leq \tau(i-1, \alpha Q^j), \quad (9)$$

$$T\big[i-1, \lfloor j + \ln_Q(1-\alpha) \rfloor\big] \leq \tau(i-1, Q^{\lfloor j + \ln_Q(1-\alpha) \rfloor}) \leq \tau(i-1, (1-\alpha)Q^j). \quad (10)$$

Combining (9) and (10) with min, max operators we obtain

$$\left(\min_{\alpha\in[0,1]} \max \begin{cases} T\big[i-1, \lfloor j + \ln_Q \alpha \rfloor\big], \\ T\big[i-1, \lfloor j + \ln_Q(1-\alpha) \rfloor\big] + w_i \end{cases} \right) \leq \left(\min_{\alpha\in[0,1]} \max \begin{cases} \tau(i-1, \alpha Q^j), \\ \tau(i-1, (1-\alpha)Q^j) + w_i \end{cases} \right) = \tau(i, Q^j),$$

establishing that $T[i,j]$ computed using (5) satisfy the upper bound in (6). ∎

We can now prove that the output Z' of the algorithm **CountKnapsack** is a $(1 \pm \varepsilon)$ multiplicative approximation of Z. Note that Z' is never an underestimate of Z, since,

$$C < T[n, j'+1] \leq \tau(n, Q^{j'+1}),$$

that is, there are at most $Q^{j'+1}$ solutions. We also have

$$\tau(n, Q^{j'-n}) \leq T[n,j'] \leq C,$$

that is, there are at least $Q^{j'-n}$ solutions. Hence

$$\frac{Z'}{Z} \leq \frac{Q^{j'+1}}{Q^{j'-n}} = Q^{n+1} \leq \mathrm{e}^{\varepsilon}.$$

This proves that the output Z' of **CountKnapsack** satisfies the conclusion of Theorem 1.1. It remains to show that the algorithm can be modified to achieve the claimed running time.

Running Time: As noted earlier, the minimum over α in the recurrence (5) only needs to be evaluated at the discrete subset $S = S_1 \cup S_2$ defined earlier. Since $|S| = O(s)$, $T[i,j]$ can be computed in $O(s)$ time. Since there are $O(ns)$ entries of the table and $s = O(n^2/\varepsilon)$ the algorithm **CountKnapsack** can be implemented in $O(ns^2) = O(n^5/\varepsilon^2)$ time.

The running time can further be improved to $O(n^3\varepsilon^{-1}\log(n/\varepsilon))$ by observing that we can do binary search over both S_1 and S_2 to find the optimal α, see the full version of the paper for details.

3. Approximate Counting using Branching Programs

In this section we give an FPTAS for the multidimensional knapsack problem, thereby proving Theorem 1.2. To solve the multidimensional case, it will be convenient for us to represent knapsack instances as read-once branching programs (ROBPs). Here we describe read-once branching programs and explain their relevance for deterministic, approximate counting. Using this machinery we also give an FPTAS for all the problems considered in Dyer's paper [6].

3.1. Preliminaries

Definition 3.1 (ROBP)**.** *An (S,T)-branching program M is a layered multi-graph with a layer for each $0 \leq i \leq T$ and at most S vertices (states) in each layer. The first layer has a single vertex v_0 and each vertex in the last layer is labeled with 0 (rejecting) or 1 (accepting). For $0 \leq i \leq T$, a vertex v in layer i has two outgoing edges labeled $0,1$ and ending at vertices in layer $i+1$.*

Note that by definition, an (S,T)-branching program is read-once and oblivious in the sense that all the vertices in a layer are labeled by the same variable (these are also known as *Ordered Binary Decision Diagrams* in the literature). We also use the following notation: Let M be an (S,T)-branching program and v a vertex in layer i of M.

1. For a string z, $M(v,z)$ denotes the state reached by starting from v and following edges labeled with z.
2. For $z \in \{0,1\}^n$, let $M(z) = 1$ if $M(v_0,z)$ is an accepting state, and $M(z) = 0$ otherwise.
3. $A_M(v) = \{z : M(v,z) \text{ is accepting in } M\}$ and $P_M(v)$ is the probability that $M(v,z)$ is an accepting state for z chosen uniformly at random.
4. $L(M,i)$ denotes the vertices in layer i of M.
5. For a set U, $x \in_u U$ denotes a uniformly random element of U.

Given an instance of the knapsack problem $w_1,\ldots,w_n$ and C, let $W = \sum_i |w_i|$. It is easy to see that the set of accepting strings for this knapsack instance are computed exactly by a (W,n)-branching program. Intuitively, given an input x, the jth layer of this branching program keeps track of the partial sum after reading j bits of input, namely $\sum_{i=1}^{j} w_i x_i$. Since the partial sum cannot exceed W, a width-W ROBP suffices to carry out the computation.

Monotone ROBPs: We will also require that the branching programs we work with satisfy a monotonicity condition and can be described implicitly. We discuss these two notions below.

Definition 3.2 (Monotone ROBP)**.** *A (W,n)-branching program M is monotone if for all $i \leq n$, there exists a total ordering $\prec$ on the vertices in $L(M,i)$ such that if $u \prec v$, then $A_M(u) \subseteq A_M(v)$.*

Monotone ROBPs were introduced by Meka and Zuckerman [20] in their work on pseudorandom generators for halfspaces. It is easy to see that the branching program for knapsack satisfies the above monotonicity condition: Given partial sums v_j, v_k we say $v_j \prec v_k$ if $v_j > v_k$, since a larger partial sum means that fewer suffix strings will be accepted.

Since we deal with monotone ROBPs that potentially have width exponential in n, we require that the monotone ROBP M be described implicitly in the following sense (we assume that the following two operations are of unit cost):

1. Ordering: given two states u, v we can efficiently check if $u \prec v$ and if so find a w that is *halfway* between u, v, i.e., $\big||\{x : u \prec x \prec w\}| - |\{x : w \prec x \prec v\}|\big| \leq 1$.
2. Transitions: Given any vertex of M we can compute the two neighbors of the vertex.

Small-Space Sources: Let M be a ROBP computing an instance of knapsack. We are interested in computing the quantity $\widehat{M} = \Pr_{x\in\{0,1\}^n}[M(x) = 1]$ deterministically, as $2^n \cdot \widehat{M}$ equals the number of solutions to the knapsack instance. As we will see in the next section, we will also need to estimate $\Pr_{x\sim D}[M(x) = 1]$ where D is a non-uniform distribution over $\{0,1\}^n$ that corresponds to conditioning on certain events. As such, we will need to use the notion of a *small-space* source, which is a small-width branching program that encodes a distribution on $\{0,1\}^n$ (we will often abuse notation and denote the distribution generated by a branching program and the branching program itself by D).

Small-space sources were introduced by Kamp et al. [15] in their work on randomness extractors as a generalization of many commonly studied distributions such as Markov-chain sources and bit-fixing sources.

Definition 3.3 (small-space sources, Kamp et al. [15])**.** *A width w small-space source is described by a (w,n)-*

branching program D and additional probability distributions p_v on the outgoing edges associated with vertices $v \in D$. Samples from the source are generated by taking a random walk on D according to the distributions p_v and outputting the labels of the edges traversed.

Several natural distributions such as all symmetric distributions and product distributions can be generated by small-space sources. We will use the following straightforward claims:

Claim 3.4. *Given a ROBP M of width at most W and a small-space source D of width at most S, $\Pr_{x\sim D}[M(x)=1]$ can be computed exactly via dynamic programming in time $O(n \cdot S \cdot W)$.*

Claim 3.5. *Given a (W,n)-ROBP M, the uniform distribution over M's accepting inputs, $\{x : M(x)=1\}$ is a width W small-space source.*

3.2. Main Structural Result and Applications

As mentioned earlier, the set of accepting strings for a knapsack instance of total weight W can be computed exactly by a (W,n)-branching program. Unfortunately, this observation does not help in counting solutions to knapsack as W can be exponentially large. To handle the large width, we exploit the fact that the natural ROBP for computing knapsack solutions is monotone to efficiently compute a small-width ROBP that approximates the knapsack ROBP with small relative error. Moreover, we are able to compute such an approximation small-width ROBP with small relative error under arbitrary small space sources, which is critical for the multi-dimensional case.

Our main counting results are obtained by proving the following key structural theorem for monotone ROBPs that we believe is of independent interest:

Theorem 3.6. *Given a (W,n)-monotone ROBP M, $\delta > 0$, and a small-space distribution D over $\{0,1\}^n$ of width at most S, there exists an $(O(n^2S/\delta),n)$-monotone ROBP M^0 such that for all z, $M(z) \le M^0(z)$ and*

$$\Pr_{x\sim D}[M(z)=1] \le \Pr_{x\sim D}[M^0(z)=1] \le (1+\delta)\Pr_{x\sim D}[M(z)=1].$$

Moreover, given an implicit description of M and an explicit description of D, M^0 can be constructed in deterministic time $O(n^3S(S+\log(W))\log(n/\delta)/\delta)$.

We first show how to use Theorem 3.6 to obtain an FPTAS for the multidimensional knapsack problem. We then give a proof of Theorem 3.6 in Section 3.3. Notice that an FPTAS for the (one dimensional) knapsack problem follows immediately from Theorem 3.6 and the earlier observation that a weight W knapsack instance is a (W,n)-monotone ROBP. In fact, we can approximately count knapsack solutions with respect to all symmetric and product distributions, as each of these can generated by a small-space source.

There are two subtle issues in applying Theorem 3.6 directly for the multidimensional knapsack problem. Firstly, the natural ROBP for multidimensional knapsack obtained by taking the intersection of ROBPs for the individual knapsack constraints need not be monotone. Secondly, even starting with small-width, low relative-error approximating ROBPs for each individual knapsack constraint, it is not clear how to obtain a low relative-error approximating ROBP for the multidimensional case. This is because taking the intersection of the approximating ROBPs only preserves approximation in *additive* error and not in relative error which is what we want. We handle these issues by using the following elegant result due to Dyer, which essentially helps us reduce the problem of obtaining relative-error approximations to obtaining additive-error approximations with respect to small space sources.

Theorem 3.7 (Dyer, [6])**.** *Given knapsack instances $\left\{\left(w_1^{(i)},\ldots,w_n^{(i)},C^{(i)}\right)\right\}_{i=1}^{k}$ we can deterministically, in time $O(n^3)$, construct a new set of knapsack instances $\left\{\left(u_1^{(i)},\ldots,u_n^{(i)},B^{(i)}\right)\right\}_{i=1}^{k}$, each with a total weight of at most $O(n^3)$ such that the following holds. For $1 \le i \le k$, let S_i, S_i' denote the feasible solutions for the knapsack instances $\left(w_1^{(i)},\ldots,w_n^{(i)},C^{(i)}\right)$, $\left(u_1^{(i)},\ldots,u_n^{(i)},B^{(i)}\right)$ respectively. Then, $S_i \subseteq S_i'$ and*

$$\Big|\bigcap_{i\in[k]} S_i'\Big| \le (n+1)^k \Big|\bigcap_{i\in[k]} S_i\Big|.$$

Proof of Theorem 1.2: We first use Theorem 3.7 to obtain low-weight knapsack instances. As each instance has weight at most $O(n^3)$, they are each computable exactly by a $(O(n^3),n)$-ROBP. It is straightforward to check that the intersection of several ROBPs is a computable by a ROBP with width at most the product of the widths of the original ROBPs. Therefore, the intersection $U = \cap_i S_i'$ is computable by a $(O(n^{3k}),n)$-ROBP. Thus, if D is the uniform distribution over U, then by Claim 3.5, D can be generated by an explicit $O(n^{3k})$ space source.

For $i \in [k]$, let M^i be a (W,n)-ROBP exactly computing the feasible set of the i'th original knapsack instance. Recall that we wish to approximate the quantity $\Pr_{x\in_u\{0,1\}^n}[\wedge_i M^i(x)=1]$. We can rewrite this as follows:

$$\Pr_{x\in_u\{0,1\}^n}[\wedge_i M^i(x)=1] = \Pr_{x\in_u\{0,1\}^n}[x\in U]\cdot\Pr_{x\sim D}[\wedge_i M^i(x)=1].$$

Since the multidimensional knapsack instance output by Dyer's algorithm has low-weight, we can apply Claim 3.4 to compute $\Pr_{x\in_u\{0,1\}^n}[x\in U]$ exactly. Theorem 3.6 will give us a relative-error approximation to $\Pr_{x\sim D}[\wedge_i M^i(x)=1]$ and complete the proof. We now proceed more formally.

Let $\delta = O(\epsilon/k(n+1)^k)$ to be chosen later. For every $i \in [k]$, by Theorem 3.6 we can explicitly in time $n^{O(k)}(\log W)/\delta$ construct a $(n^{O(k)}/\delta, n)$-ROBP N^i such that,

$$\Pr_{x\sim D}[N^i(x) \neq M^i(x)] \leq \delta.$$

Let M be the $(n^{O(k^2)}/\delta^k, n)$-ROBP computing the intersection of N^i for $i \in [k]$, i.e., $M(x) = \wedge_i N^i(x)$. Then by a union bound we have $\Pr_{x\sim D}[M(x) \neq \wedge_i M^i(x)] \leq k\delta$. On the other hand, by Theorem 3.7, $\Pr_{x\sim D}[\wedge_i M^i(x) = 1] \geq 1/(n+1)^k$. Therefore, from the above two equations and setting $\delta = \epsilon/2k(n+1)^k$, we get that

$$\begin{aligned}\Pr_{x\sim D}[M(x)=1] &\leq \Pr_{x\sim D}[\wedge_i M^i(x)=1] \\ &\leq (1+\epsilon)\Pr_{x\sim D}[M(x)=1].\end{aligned}$$

Thus, $p = \Pr_{x\in_u\{0,1\}^n}[x \in U] \cdot \Pr_{x\sim D}[M(x)=1]$ is an ϵ-relative error approximation to $\Pr_{x\in_u\{0,1\}^n}[x \in U] \cdot \Pr_{x\sim D}[\wedge_i M^i(x)=1] = \Pr_{x\in_u\{0,1\}^n}[\wedge_i M^i(x)=1]$.

The theorem follows as we can compute p in time $(n/\delta)^{O(k^2)}$ using Claim 3.4, as D is a small-space source of width at most $O(n^{3k})$ and M has width at most $(n/\delta)^{O(k^2)}$. ∎

3.3. Proof of Theorem 3.6

We start with some notation. Let D denote the small space generator of width at most S. For $A \subseteq \{0,1\}^n$ we use $D(A)$ to denote the measure of A under D. Let $U^1, \ldots, U^n$ be the vertices in D with U^i being the i'th layer of D. For a vertex $u \in U^i$, let D^u be the distribution over $\{0,1\}^{n-i}$ induced by taking a random walk in D starting from u. Given a vertex $v \in L(M,i)$ and $u \in U^i$, let $P_{M,u}(v)$ denote the probability of accepting if we start from v and make transitions in M according to a suffix sampled from distribution D^u.

We will start from the exact branching program M and work backwards, constructing a sequence of programs $M^n = M, \ldots, M^0$, where M^i is obtained from M^{i+1} by "rounding" the $(i+1)^{st}$ layer. The construction is similar in spirit to the work of Meka and Zuckerman [20], who showed how to round a halfspace constraint with respect to the uniform distribution and obtain small additive error: partial sums can be grouped together if they result in similar "suffix acceptance probability." Here we work with relative error and small-space sources; we do the rounding in such a way as to ensure the acceptance probabilities are well approximated under each of the possible distributions on suffixes D^u. Further, after layer i is rounded, it will be of small width. Thus, the branching program M^0 will be a small-width program.

The rounding process for creating M^i has two steps. First, we need to create "breakpoints" for the $(i+1)$'st layer of M^{i+1}. Then, we have to round the edges going from layer i to layer $i+1$. We now create our breakpoints. Let $L(M^{i+1}, i+1) = \{v_1 \prec v_2 \cdots \prec v_W\}$. Fix a vertex $u \in U^{i+1}$. We define a set $B^{i+1}(u) = \{v_{u(j)}\} \subseteq L(M^{i+1}, i+1)$ of breakpoints for u as follows. We start with $v_{u(1)} = v_W$ and given $v_{u(j)}$ define $v_{u(j+1)}$ by

$$\begin{aligned}v_{u(j+1)} &= \max v \ s.t. v \prec v_{u(j)} \text{ and} \\ 0 < P_{M^{i+1},u}(v) &< P_{M^{i+1},u}(v_{u(j)})/(1+\varepsilon) \qquad (11)\end{aligned}$$

Let $B^{i+1} = \cup_{u\in U^{i+1}} B^{i+1}(u) = \{b_1 \prec \cdots \prec b_N\}$ be the union of breakpoints for all u. We set $L(M^i, i+1) = B^{i+1}$. The vertices in all other layers stay the same as in M^{i+1}. Figure 1(a) (page 8) shows the breakpoints (in blue) for a single vertex u of the small space source, and Figure 1(b) (page 8) shows the complete set of breakpoints.

Now we describe the edges of M^i. All the edges except those from layer i to $i+1$ stay the same as in M^{i+1}, and we round these edges *upward* as follows: let $v' \in L(M^{i+1}, i)$ and for $z \in \{0,1\}$, let $M^{i+1}(v', z) = v \in L(M^{i+1}, i+1)$. Find two consecutive vertices $b_k, b_{k+1} \in L(M^i, i+1)$ such that $b_k \prec v \preceq b_{k+1}$. We set $M^i(v', z) = b_{k+1}$. Note that this only increases the number of accepting suffixes for v'. Figure 1(b) shows the rounding of edges. This completes the construction of the M^is. The following claim is straightforward:

Claim 3.8. *The branching program M^i is monotone where the ordering of vertices in each layer is the same as M.*

We also obtain the following simple lemma:

Lemma 3.9. *For $v \in M^i$, $A_{M^{i+1}}(v) \subseteq A_{M^i}(v)$. Thus $P_{M,u}(v) \leq P_{M^i,u}(v)$ for all $u \in U^i$.*

Proof: It suffices to prove the claim when $v \in L(M^i, i)$. Fix $z \in \{0,1\}$ and let $v' = M^{i+1}(v,z)$. We have $M^i(v,z) = b_{k+1}$ where $b_k \prec v' \preceq b_{k+1}$. By Claim 3.8, we have $A_{M^i}(v') \subseteq A_{M^i}(b_{k+1})$. Thus the set of accepting suffixes only increases for either value of z. ∎

Further, we claim that M^0 can be computed efficiently.We now show that the number of accepting solutions does not increase too much.

Lemma 3.10. *For $v \in L(M^i, j)$ and $u \in U^j$, we have $P_{M^i,u}(v) \leq P_{M,u}(v)(1+\varepsilon)^{n-i}$.*

Proof: It suffices to show that $P_{M^i,u}(v) \leq P_{M^{i+1},u}(v)(1+\varepsilon)$. This claim is trivial for $j \geq i+1$ since for such vertices, $P_{M^i,u}(v) = P_{M^{i+1},u}(v)$. The crux of the argument is when $j = i$. Since M^{i+1} and M^i are identical up to layer i, the claim for $j < i$ will follow.

Fix $v \in L(M^i, i)$ and $u \in U^i$. Let u_0, u_1 denote the neighbors of u in D. Then we have

$$\begin{aligned}P_{M^i,u}(v) = p_u(0)P_{M^i,u_0}(M^i(v,0)) + \\ p_u(1)P_{M^i,u_1}(M^i(v,1)). \qquad (12)\end{aligned}$$

We first bound $P_{M^i,u_0}(M^i(v,0))$. Let b', b'''' be the breakpoints in $B^{i+1}(u_0)$ such that $b' \prec M^{i+1}(v,0) \preceq b''''$ and let a_2, a_3 be the breakpoints in B^{i+1} such that $b'' \prec M^{i+1}(v,0) \preceq b'''$. Note that $M^i(v,0) = b'''$, by the

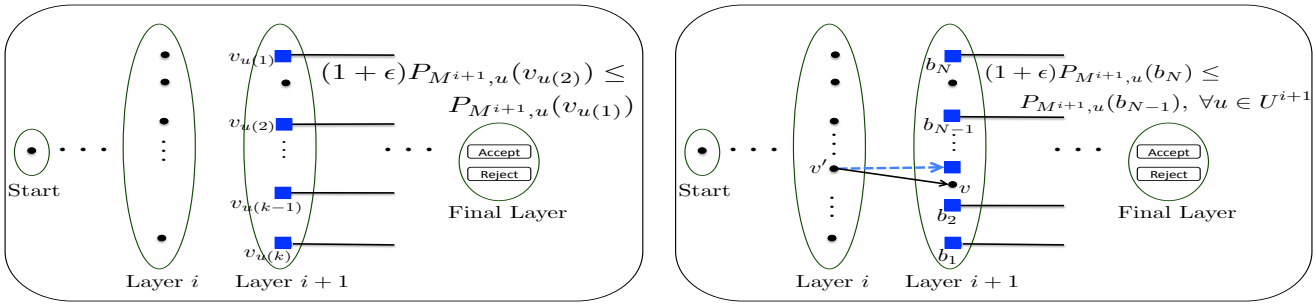

Figure 1. Rounding M^{i+1} to M^i. **(a)**: Breaking points (blue squares) $B^{i+1}(u)$ for a fixed u. **(b)**: Edge rounding: the original edge (black solid) from v' to v is rounded up (blue dashed) to b_3 - the next "higher" vertex to v in B^{i+1}.

construction of M^i. Since $B^{i+1}(u_0) \subseteq B^{i+1}$, we get $b' \preceq b'' \prec M^{i+1}(v,0) \preceq b''' \preceq b''''$. By the definition of breakpoints, we have $P_{M^{i+1},u_0}(b'''') \leq (1+\varepsilon)P_{M^{i+1},u_0}(M^{i+1}(v,0))$ and by the monotonicity of M^{i+1} $P_{M^{i+1},u_0}(b''') \leq P_{M^{i+1},u_0}(b'''')$, which together show that

$$P_{M^{i+1},u_0}(b''') \leq (1+\varepsilon)P_{M^{i+1},u_0}(M^{i+1}(v,0)).$$

Since $b''' \in L(M^i, i+1)$, we have $P_{M^i,u_0}(b''') = P_{M^{i+1},u_0}(b''')$. Thus

$$P_{M^i,u_0}(M^i(v,0)) \leq (1+\varepsilon)P_{M^{i+1},u_0}(M^{i+1}(v,0)).$$

Similarly, we can show

$$P_{M^i,u_1}(M^i(v,1)) \leq (1+\varepsilon)P_{M^{i+1},u_1}(M^{i+1}(v,1)).$$

Plugging these into Equation 12 gives

$$\begin{aligned} P_{M^i,u}(v) \ &\leq (1+\varepsilon)(p_u(0)P_{M^{i+1},u_0}(M^{i+1}(v,0)) + \\ &p_u(1)P_{M^{i+1},u_1}(M^{i+1}(v,1))) \ = (1+\varepsilon)P_{M^{i+1},u}(v) \end{aligned}$$

which is what we set out to prove. ∎

We now analyze the complexity of constructing M^0 from M:

Claim 3.11. *The branching program M^0 can be constructed in time $O(n^2 S(S+\log(W))\log(nS/\varepsilon)/\varepsilon)$.*

Proof: Observe that for every i and $u \in U^i$, $|B^i(u)| \leq \frac{2n}{\varepsilon}$ and hence $|B^i| \leq \frac{2nS}{\varepsilon}$. Let us analyze the complexity of constructing M^i from M^{i+1}. We will assume inductively that the set B^{i+2} is known and stored in a binary tree along with the values $P_{M^{i+1},u}(b)$, for every $b \in B^{i+2}$ and $u \in U^{i+2}$. Hence, given $v \in L(M, i+1)$, we can find $b_k, b_{k+1} \in B^{i+2}$ such that $b_k \prec v \preceq b_{k+1}$ in time $\log(nS/\varepsilon)$. This ensures that if we are given a vertex $v' \in L(M^{i+1}, i+1)$ and $u \in U^{i+1}$, we can compute $P_{M^{i+1},u}(v')$ in time $\log(nS/\varepsilon)$. To see this, note that

$$P_{M^{i+1},u}(v') = \sum_{z \in \{0,1\}} p_u(z) P_{M^{i+1},u_z}(M^{i+1}(v',z))$$

where $u_z \in U^{i+2}$ denotes the vertex reached in D when taking the edge labeled z from u. To compute $M^{i+1}(v',z)$ we first compute $v = M(v',z)$ using the fact that M is described implicitly. We then find $b_k \prec v \preceq b_{k+1}$ in B^{i+2} and set $M^{i+1}(v',z) = b_{k+1}$. Since we have the values of $P_{M^{i+1},u_z}(b)$ precomputed, we can use them to compute $P_{M^{i+1},u}(v')$. The time required is dominated by the $O(\log(nS/\epsilon))$ time needed to find b_{k+1}.

Now, for each $u \in U^{i+1}$, by using binary search on the set of vertices, each new breakpoint in $B^{i+1}(u)$ can be found in time $O(\log(W)\log(nS/\varepsilon))$. Thus finding the set B^{i+1} takes time $O(nS\log(W)\log(nS/\varepsilon)/\varepsilon)$.

Once we find the set B^{i+1}, we store it as a binary tree. We compute and store the values of $P_{M^i,u}(b) = P_{M^{i+1},u}(b)$ for each $b \in B^{i+1}$ and $u \in U^{i+1}$ in time $O(nS^2\log(nS/\varepsilon)/\varepsilon)$.

Thus overall, the time required to construct M^0 from M is $O(n^2 S(S+\log(W))\log(nS/\varepsilon)/\varepsilon)$. ∎

Proof of Theorem 3.6: Choose $\varepsilon = \Omega(\delta/n)$ so that $(1+\varepsilon)^n \leq (1+\delta)$. We construct the program M^0 from M and output $P_{M^0,u}(s)$ where s is the start state of M and u is the start state of S. By Claim 3.11, this takes time $O(n^3 S(S+\log(W))\log(nS/\delta)/\delta)$. Applying Lemmas 3.10 and 3.9, we conclude that

$$P_{M,u}(s) \leq P_{M^0,u}(s) \leq P_{M,u}(s)(1+\delta).$$

Note that $P_{M,u}(s)$ and $P_{M^0,u}(s)$ are respectively the probabilities that M and M^0 accept a string sampled from the distribution D. This completes the proof. ∎

4. An FPTAS for General Integer Knapsack

In this section, we address the problem of counting solutions to knapsack where the feasible solutions can take integer values instead of being restricted to be 0/1 valued. Given, non-negative integer weights $w = (w_1, \ldots, w_n)$, a capacity C and non-negative integer ranges $u = (u_1, \ldots, u_n)$, the goal here is to estimate the size of the set of solutions $\mathsf{KNAP}(w, C, u) = \{x : \sum_{i \leq n} w_i x_i \leq C,\ 0 \leq x_i \leq u_i\}$. Note that the range sizes $u_1, \ldots, u_n$ could be exponential in n. Dyer [6] gave an FPRAS for

integer-valued knapsack as well. We obtain a FPTAS for the problem.

Theorem 4.1 (integer knapsack)**.** *Given a knapsack instance* $\mathsf{KNAP}(w, C, u)$ *with weight* $W = \sum_i w_i u_i + C$, $U = \max_i u_i$ *and* $\epsilon > 0$, *there is a deterministic* $O(n^5(\log U)^2(\log W)/\epsilon^2)$ *algorithm that computes an* ϵ*-relative error approximation for* $|\mathsf{KNAP}(w, C, u)|$.

As in the case of $\{0,1\}$-knapsack we start with the exact branching program M for $\mathsf{KNAP}(w, C, u)$, where each state in $L(M, j)$ corresponds to a partial sum $v_j = \sum_{i \le j} w_i x_i$ and has $(u_{j+1} + 1)$ outgoing edges corresponding to the possible values of variable x_{j+1}. We then approximate this program with a small-width branching program as was done for $\{0,1\}$-knapsack. However, unlike the previous case, where we only had to worry about the width being large, the program M can have both exponentially large width and degree. To handle this, we observe that the branching program M is an *interval ROBP* in the sense defined below, which allows us to shrink the state space as well as obtain succinct descriptions of the edges of the new branching programs we construct.

Definition 4.2 (Interval ROBPs)**.** *For* $u = (u_1, \ldots, u_n) \in \mathbb{Z}_+^n$, $S, T \in \mathbb{Z}_+$, *an* (S, u, T)*-interval ROBP* M *is a layered multi-graph with a layer for each* $0 \le i \le T$, *at most* S *states in each layer. The first layer has a single (start) vertex, each vertex in the last layer is labeled accepting or rejecting. A vertex* v *in layer* $i-1$ *has exactly* $u_i + 1$ *edges labeled* $\{0, 1, \ldots, u_i\}$ *to vertices in layer* i. *Further, there exists a total order* $\prec$ *on the vertices of layer* i *for* $0 \le i \le T$ *such that the edge labelings respect the ordering in the following sense: for a vertex* v *in layer* $i-1$, *if* $M(v, k)$ *denotes the* k*'th neighbor of* v *for* $0 \le k \le u_i$, *then* $M(v, u_i) \preceq M(v, u_i - 1) \preceq \cdots \preceq M(v, 0)$.

An interval ROBP defines a natural Boolean function $M : \{0, \ldots, u_1\} \times \{0, \ldots, u_2\} \times \cdots \times \{0, \ldots, u_n\} \to \{0, 1\}$ *where on input* $x = (x_1, \ldots, x_n)$, *we begin at the start vertex and output the label of the final vertex reached when traversing* M *according to* x.

Note that the case $u = (1, 1, \ldots, 1)$ corresponds to a special class of monotone ROBPs. The intuition behind the definition of interval ROBPs is that even if an interval ROBP M has large degree, the edges of M can be represented succinctly: Given an (S, u, T)-interval ROBP M, and a vertex v in layer $i-1$, the edges out of v can be described exactly by a subset of at most $2S$ edges irrespective of how large the degree of v u_i is. For, if v' is a vertex in i'th layer, and $E(v, v') = \{0 \le k \le u_i : M(v, k) = v'\}$ is the set of edge labels going from v to v', then $E(v, v')$ is an interval, meaning $E(v, v') = \{l_{v,v'}, l_{v,v'} + 1, l_{v,v'} + 2, \ldots, r_{v,v'}\}$ for some integers $l_{v,v'}, r_{v,v'}$. Thus, the set of edges $E(v, v')$ is completely described by $l_{v,v'}$ and $r_{v,v'}$. We exploit this observation critically when computing the small-width approximating branching program.

In analogy to the case of $\{0,1\}$-knapsack, given an instance $\mathsf{KNAP}(w, C, u)$ of integer knapsack, there is an interval ROBP that exactly computes the set $\mathsf{KNAP}(w, C, u)$. Let M denote this interval ROBP with edges between layer $i-1$ and layer i labeled by $x_i \in \{0, \ldots, u_i\}$ and for $v \in L(M, i-1)$, $0 \le x_i \le u_i$ we have $M(v, x_i) = v + w_i x_i \in L(M, i)$.

Given a vertex $v \in L(M, i)$ we use $P_M(v)$ to denote the probability that $M(v, z)$ accepts, for z chosen uniformly form $\{0, \ldots, u_{j+1}\} \times \cdots \times \{0, \ldots, u_n\}$. As in the proof of Theorem 3.6, we construct a series of progressively simpler interval ROBPs $M^n = M, M^{n-1}, \ldots, M^0$ with a similar rounding procedure.

We next describe how to obtain M^i from M^{i+1}. This involves two steps: we first create "breakpoints" to sparsify the $i+1$'th layer $L(M^{i+1}, i+1)$ of M^{i+1} and then round the edges going from layer i to layer $i+1$. We set $L(M^i, i+1) = \{v_1, \ldots, v_\ell\} \subseteq L(M^{i+1}, i+1)$ where the v_j's are defined as follows: Let $v_1 = 0$. Given v_j, let

$$v_{j+1} = \min v \text{ such that } v > v_j \text{ and } 0 < P_{M^{i+1}}(v) < P_{M^{i+1}}(v_j)/(1+\eta). \tag{13}$$

Let $I_1 = \{v_1, \ldots, v_2 - 1\}, \ldots, I_\ell = \{v_\ell, \ldots\}$, where $\ell \le n(\log U)/\eta$ as $P_{M^i}(v_1) \le 1$ and $P_{M^i}(v_\ell) \ge U^{-n}$. Next we redirect the transitions going from level i to level $i+1$. If we have an edge labeled $z \in \{0, \ldots, u_{i+1}\}$ entering a vertex $v \in I_j$, then we redirect the edge to vertex v_j. The redirection will be done implicitly in the sense that for any vertex v in level i and a vertex v_j, we only compute and store the end points $\{l_{v,v_j}, r_{v,v_j}\}$ of the interval $E(v, v_j) = \{0 \le k \le u_{i+1} : M^i(v, k) = v_j\}$.

Our branching programs have the following approximating properties analogous to Lemmas 3.8, 3.9, 3.10 and Claim 3.11. The proofs are deferred to the full version.

Lemma 4.3. *For any* $v \in L(M^i, j)$ *and* $0 \le k \le l \le u_{j+1}$, $M^i(v, k) \le M^i(v, l)$. *Let* $v, v' \in L(M^i, j)$ *and* $v \le v'$. *For any suffix* z, $M^i(v, z) \le M^i(v', z)$.

Lemma 4.4. *For* $v \in M^i$, *we have* $A_{M^{i+1}}(v) \subseteq A_{M^i}(v)$. *Further, for any* $v \in L(M^i, j)$ *where* $j \le i$, *we have* $P_M(v) \le P_{M^i}(v) \le P_M(v)(1+\eta)^{n-i}$.

Lemma 4.5. *Each vertex* $v_j \in L(M^i, i+1)$ *can be computed in time* $O(n(\log U)(\log W)/\eta)$.

Proof of Theorem 4.1: We set $\eta = \delta/2n$ and use the above arguments to construct the branching program M^0 and compute the value of $P_{M^0}(s)$ where s is the start state. By Lemma 4.4

$$P_M(s) \le P_{M^0}(s) \le P_M(s)(1+\eta)^n \le (1+\delta)P_M(s),$$

where the last inequality holds for small enough δ. Finally, note that the number of solutions $|\mathsf{KNAP}(w, C, u)|$ is pre-

cisely $P_M(s)\prod_i(u_i+1)$. Hence we output $P_{M^0}(s)\prod_i(u_i+1)$. ■

REFERENCES

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Ann. of Math.*, 160(2):781–793, 2004.

[2] M. Ajtai and A. Wigderson. Deterministic simulation of probabilistic constant depth circuits. *Advances in Computing Research - Randomness and Computation*, 5:199–223, 1989. A preliminary version appears in *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, 11-19, 1985.

[3] S. Arora and B. Barak. *Computational complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009.

[4] M. Bayati, D. Gamarnik, D. Katz, C. Nair, and P. Tetali. Simple deterministic approximation algorithms for counting matchings. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 122–127, 2007.

[5] I. Diakonikolas, P. Gopalan, R. Jaiswal, R. A. Servedio, and E. Viola. Bounded independence fools halfspaces. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 171–180, 2009.

[6] M. Dyer. Approximate counting by dynamic programming. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 693–699, 2003.

[7] M. Dyer, A. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

[8] M. Dyer, A. Frieze, R. Kannan, A. Kapoor, L. Perkovic, and U. Vazirani. A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem. *Combin. Probab. Comput.*, 2(3):271–284, 1993.

[9] D. Gamarnik and D. Katz. Correlation decay and deterministic FPTAS for counting list-colorings of a graph. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1245–1254, 2007.

[10] M. Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Struct. Algorithms*, 7(2):157–165, 1995.

[11] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

[12] M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993.

[13] M. Jerrum and A. Sinclair. The Markov Chain Monte Carlo Method: An Approach To Approximate Counting and Integration. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 482–520. PWS Publishing, 1996.

[14] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

[15] J. Kamp, A. Rao, S. P. Vadhan, and D. Zuckerman. Deterministic extractors for small-space sources. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 691–700, 2006.

[16] R. M. Karp and M. Luby. Monte Carlo algorithms for the planar multiterminal network reliability problem. *J. Complexity*, 1(1):45–64, 1985.

[17] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *J. Comput. System Sci.*, 72(2):392–417, 2006.

[18] M. Luby, D. Randall, and A. Sinclair. Markov chain algorithms for planar lattice structures. *SIAM J. Comput.*, 31(1):167–192, 2001.

[19] M. Luby and B. Veličković. On deterministic approximation of DNF. *Algorithmica*, 16(4/5):415–433, 1996.

[20] R. Meka and D. Zuckerman. Pseudorandom generators for polynomial threshold functions. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 427–436, 2010.

[21] B. Morris and A. Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. *SIAM J. Comput.*, 34(1):195–226, 2004.

[22] Y. Rabani and A. Shpilka. Explicit construction of a small epsilon-net for linear threshold functions. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 649–658, 2009.

[23] A. Sly. Computational transition at the uniqueness threshold. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 287–296, 2010.

[24] D. Štefankovič, S. Vempala, and E. Vigoda. Adaptive simulated annealing: a near-optimal connection between sampling and counting. *J. ACM*, 56(3):1–36, 2009.

[25] D. Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 140–149, 2006.