A DETERMINISTIC POLYNOMIAL-TIME APPROXIMATION SCHEME FOR COUNTING KNAPSACK SOLUTIONS*

DANIEL ŠTEFANKOVIČ†, SANTOSH VEMPALA‡, AND ERIC VIGODA‡

Abstract. Given n elements with nonnegative 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 the given capacity. We give a deterministic algorithm that estimates the number of solutions to within relative error $1 \pm \varepsilon$ in time polynomial in n and $1/\varepsilon$ (fully polynomial approximation scheme). More precisely, our algorithm takes time $O(n^3 \varepsilon^{-1} \log(n/\varepsilon))$. Our algorithm is based on dynamic programming. Previously, randomized polynomial-time approximation schemes were known first by Morris and Sinclair via Markov chain Monte Carlo techniques and subsequently by Dyer via dynamic programming and rejection sampling.

Key words. approximate counting, knapsack, dynamic programming

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].)

However, when one moves to approximation algorithms, there are many more such examples. The field of approximate counting is largely based on Markov chain Monte Carlo sampling [15], a technique that is inherently randomized, and has had remarkable success. The problems of counting matchings [13, 16], colorings [12], various tilings, partitions and arrangements [19], estimating partition functions [14, 27], or volumes [6, 18] are all solved by first designing a random sampling method and then reducing counting to repeated sampling. In all these cases, when the input is presented explicitly, it is conceivable that deterministic polynomial-time algorithms exist.[1]

Our interest is in obtaining a fully polynomial approximation scheme (FPAS) for a #P-complete counting problem. Formally, our aim is to construct 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 $1/\varepsilon$. Similarly, a fully-polynomial randomized approximation scheme (FPRAS) takes an additional parameter $1 > \delta > 0$ as input for the failure probability, the

[1] Volume computation has an exponential lower bound for deterministic algorithms, but that is due to the more general oracle model in which the input is presented.

output of the algorithm is correct with probability $\geq 1 - \delta$, and the running time is required to be polynomial in $\log(1/\delta)$.

There is a line of work concerned with approximate counting of satisfying assignments of disjunctive normal form (DNF) formulas, beginning with the FPRAS of Karp and Luby [17]. An FPAS for the #DNF problem when every clause is constant length follows from work of Ajtai and Wigderson [2]; see Luby and Veličković [20, Corollary 13].

A notable recent example of an FPAS for a #P-complete problem is Weitz's algorithm [30] for counting independent sets in graphs of maximum degree $\Delta \leq 5$. More generally, Weitz's FPAS counts independent sets weighted by a parameter $\lambda > 0$ for graphs of maximum degree Δ when Δ is constant and $\lambda < \lambda_c(\Delta)$, where $\lambda_c(\Delta) = (\Delta - 1)^{\Delta-1}/(\Delta - 2)^{\Delta}$ is the so-called uniqueness threshold on the infinite Δ-regular tree. Recently, Sly [26] showed that there is no polynomial-time approximation scheme (unless NP = RP) for estimating this sum of weighted independent sets on graphs of maximum degree Δ for λ satisfying $\lambda_c(\Delta) < \lambda < \lambda_c(\Delta) + \varepsilon_\Delta$ for some $\varepsilon_\Delta > 0$ (in [8] the assumption $\lambda < \lambda_c(\Delta) + \varepsilon_\Delta$ is removed for all $\Delta \neq 3, 4$). Weitz's approach was later extended to counting all matchings of bounded degree graphs [4]. An alternative, related approach of Gamarnik and Katz [9] gives an FPAS for k-colorings of triangle-free graphs with maximum degree Δ when Δ is a sufficiently large constant and $k > 2.84\ldots\Delta$. One limitation of Weitz's approach and related works is that the running time is quite large, in particular, the exponent depends on $\log \Delta$. In contrast, our algorithm has a small polynomial running time.

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 nonnegative 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. From a geometric perspective, for the n-dimensional boolean hypercube, we are given as input an n-dimensional halfspace, and our goal is to determine the number of vertices of the hypercube that intersect the given halfspace. We give an FPAS 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 $1/\varepsilon$. Our algorithm is strongly polynomial (see, e.g., [29, 21]), that is, it is polynomial-time and the number of arithmetic operations is polynomial in the dimension of the problem. Our algorithm also satisfies the more stringent definition of "fully combinatorial" strongly polynomial-time algorithm (see, e.g., [24, 25]), since the only operations on numbers it uses are additions and comparisons.

Our result follows a line of work in the literature. Dyer et al. [7] gave a randomized subexponential time algorithm for this problem, based on near-uniform sampling of feasible solutions by a random walk. Morris and Sinclair [23] improved this, showing a rapidly mixing Markov chain, and obtained an FPRAS. The proof of convergence of the Markov chain is based on the technique of canonical paths and a notion of balanced permutations introduced in their analysis. In a surprising development, Dyer [5] gave a completely different approach, 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 n approximation.

Our algorithm is also based on dynamic programming, and similar to Dyer, it is inspired by the pseudopolynomial algorithm for the decision/optimization version of the knapsack problem. The complexity of the latter algorithm is $O(nC)$, where C is

the capacity bound. A similar complexity can be achieved for the counting problem as well using the 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: scale down the capacity to a polynomial in n, scale down the weights by the same factor and round down the new weights, and then count the solutions to the new problem efficiently using the pseudopolynomial-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 by an arbitrarily small additive factor can change the number of solutions by more than a constant factor (for example, if all the items have the same weight w, n is a square, and $C = w\sqrt{n} - 1$, then increasing C by one changes the number of solutions from $\sum_{i=0}^{\sqrt{n}-1} \binom{n}{i}$ to $\sum_{i=0}^{\sqrt{n}} \binom{n}{i}$).

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 main result is the following theorem.

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 $O(n^3\varepsilon^{-1}\log(n/\varepsilon))$, assuming unit cost additions and comparisons on numbers with $O(\log C)$ bits.*

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 [5] and earlier algorithms for this problem [23, 7], have running time proportional to ε^{-2}.

A preliminary version of this paper was posted on the arXiv at [28]. Independently, Gopalan, Klivans, and Meka [10] posted a paper which presents a weakly polynomial-time algorithm for the #Knapsack problem studied in this paper; in particular, their number of arithmetic operations depends on $\log C$. They use a more sophisticated approach, relying on read-once branching programs and insight from Meka and Zuckerman [22], that extends to several natural variants including the multidimensional knapsack problem. In contrast, our approach is simpler, using only

elementary techniques, and yields a strongly polynomial-time algorithm. The conference version of our paper and the paper of Gopalan, Klivans, and Meka [10] appeared as a merged extended abstract in [11].

2. Algorithm. 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 $\tau : \{0, \ldots, n\} \times \mathbb{R}_{\geq 0} \to \mathbb{R} \cup \{\pm\infty\}$, 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,

$$a \leq a' \implies \tau(i, a) \leq \tau(i, a'). \tag{2.1}$$

For $i = 0$, using standard conventions, the value of τ is given by

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

Note that the number of knapsack solutions satisfies

$$Z = \max\{a \,:\, \tau(n, a) \leq C\}.$$

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 $a \in \mathbb{R}_{\geq 0}$ we have*

$$\tau(i, a) = \min_{\alpha \in [0,1]} \max \begin{cases} \tau\big(i-1, \alpha a\big), \\ \tau\big(i-1, (1-\alpha)a\big) + w_i. \end{cases} \tag{2.2}$$

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 ith 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 $\tau(i, a)$. We defer the formal proof of the above lemma to section 3.

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 (2.2) that the function τ satisfies. Namely, T is defined by the following recurrence:

$$T[i, j] = \min_{\alpha \in [0,1]} \max \begin{cases} T\left[i-1, \lfloor j + \log_Q \alpha \rfloor\right], \\ T\left[i-1, \lfloor j + \log_Q (1-\alpha) \rfloor\right] + w_i. \end{cases} \tag{2.3}$$

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

$$T[i,j] = \min_{\alpha \in [0,1]} \max \begin{cases} T\left[i-1, \lfloor j + \log_Q \alpha \rfloor\right], \\ T\left[i-1, \lfloor j + \log_Q (1-\alpha) \rfloor\right] + w_i, \end{cases} \tag{2.3}$$

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

4. Let

$$j' := \max\{j \,:\, T[n,j] \le C\}.$$

5. Output $Z' := Q^{j'+1}$.

Note that the recurrence (2.3) is over all $\alpha \in [0,1]$. However, since the second arguments (namely, $\lfloor j + \log_Q \alpha \rfloor$ and $\lfloor j + \log_Q(1-\alpha) \rfloor$) 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

$$S_1 = \{Q^{-j}, \ldots, Q^0\} \text{ and } S_2 = \{1 - Q^0, \ldots, 1 - Q^{-j}\}.$$

The set S_1 captures those α pertaining to the argument $\lfloor j + \log_Q \alpha \rfloor$ in (2.3), and S_2 captures those for $\lfloor j + \log_Q(1-\alpha) \rfloor$. By only considering this subset S of possible α we will be able to compute T efficiently.

Remark 1. Strictly speaking, the computational model of strongly polynomial algorithms [29, 21, 24, 25] does not allow for rounding (which we need, for example, to compute $\lfloor \log_Q \alpha \rfloor$). However, we need to round only quantities that depend only on the dimension of the problem and the desired precision (but not on the weights) and hence we can implement them explicitly (operating on bits) without affecting our claim of a strongly polynomial algorithm. The details are technical and are deferred to section 4.

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

Lemma 2.2. *Let $i \ge 1$. Assume that for all $j \in \{0, \ldots, s\}$ we have that $T[i-1, j]$ satisfy* (2.4)*. Then for all $j \in \{0, \ldots, s\}$ we have that $T[i,j]$ computed using* (2.3) *satisfies*

$$\tau(i, Q^{j-i}) \le T[i,j] \le \tau(i, Q^j). \tag{2.4}$$

We defer the proof of Lemma 2.2 to section 3.

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

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

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

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

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

$$\frac{Z'}{Z} \le \frac{Q^{j'+1}}{Q^{j'-n}} = Q^{n+1} \le \mathrm{e}^{\varepsilon}. \tag{2.5}$$

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

2.1. Running time. As noted earlier, the minimum over α in the recurrence (2.3) 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.

To improve the running time, recall from (2.1) that $\tau(i, a)$ is a nondecreasing function in a. Similarly, we will argue that $T[i, j]$ is a nondecreasing function in j. By induction on i, we know that in the recursive definition of $T[i, j]$ in (2.3), the two arguments in the maximum are nondecreasing in j (for fixed α) and hence it follows that for $j \le j'$, $T[i, j] \le T[i, j']$ as claimed.

Now we are going to exploit the monotonicity of $T[i, j]$ to improve the running time. The first argument in the maximum in (2.3) (namely, $T\left[i-1, \lfloor j + \log_Q \alpha \rfloor\right]$) is nondecreasing in α. Similarly, the second argument in the maximum in (2.3) is a nonincreasing function in α. Hence, the minimum of the maximum of the two arguments occurs either at the boundary (that is, for $\alpha \in \{0, 1\}$) or for $\alpha \in (0, 1)$, where the dominant function in (2.3) changes, that is, α such that for $\beta < \alpha$

$$T\left[i-1, \lfloor j + \log_Q \beta \rfloor\right] < T\left[i-1, \lfloor j + \log_Q (1-\beta) \rfloor\right] + w_i,$$

and for $\beta > \alpha$

$$T\left[i-1, \lfloor j + \log_Q \beta \rfloor\right] \ge T\left[i-1, \lfloor j + \log_Q (1-\beta) \rfloor\right] + w_i.$$

Therefore, if we had the set S in sorted order, we can find the α that achieves the minimum in (2.3) using binary search in $O(\log s)$ time. In order to sort S we would need to be able to compare Q^{-j_1} with $1 - Q^{-j_2}$ for integer values of j_1, j_2, which would lead to unnecessary complications (for example, we would need to analyze the precision needed to perform the comparison). Note that S_1 is in sorted order and hence, using binary search, in $O(\log s)$ time we can find the $\alpha_1 \in S_1$ that achieves the minimum (of the maximum of the two arguments). Similarly we can find the $\alpha_2 \in S_2$ that achieves the minimum, and then we take α to be the smaller of α_1 and α_2. Therefore, step 3 of the algorithm CountKnapsack to compute $T[i, j]$ can be implemented in $O(\log s)$ time, and the entire algorithm then takes $O(n^3 \varepsilon^{-1} \log(n/\varepsilon))$ time, assuming unit cost addition and comparison of $O(\log C)$ bit numbers (and modulo the issue of computing $\lfloor j + \log_Q \alpha \rfloor$ and $\lfloor j + \log_Q (1-\alpha) \rfloor$, which we address in section 4). This completes the proof of the running time claimed in Theorem 1.1.

3. Proofs of lemmas. Here we present the proofs of the earlier lemmas.

We begin with the proof of Lemma 2.1, which presents the recurrence for the function $\tau(i, a)$.

Proof of Lemma 2.1. Fix any $\alpha \in [0,1]$. Let $B = \max\{\tau(i-1,\alpha a), \tau(i-1,(1-\alpha)a) + w_i\}$. Since $B \geq \tau(i-1,\alpha a)$, there are at least αa solutions with weights $w_1,\ldots,w_{i-1}$ and capacity B. Similarly, since $B - w_i \geq \tau(i-1,(1-\alpha)a)$, 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 $\tau(i,a) \leq B$. 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

$$\tau(i,a) \leq \min_{\alpha\in[0,1]} \max \begin{cases} \tau\big(i-1,\alpha a\big), \\ \tau\big(i-1,(1-\alpha)a\big) + w_i. \end{cases} \tag{3.1}$$

Consider the set of solutions of the knapsack problem with weights $w_1,\ldots,w_i$ and capacity $C = \tau(i,a)$. By the definition of $\tau(i,a)$ there are at least a solutions in this set. 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

$$\tau(i,a) \geq \min_{\alpha\in[0,1]} \max \begin{cases} \tau\big(i-1,\alpha a\big), \\ \tau\big(i-1,(1-\alpha)a\big) + w_i. \end{cases} \tag{3.2}$$

Equations (3.1) and (3.2) yield (2.2). □

We now prove Lemma 2.2 that the function T approximates τ.

Proof of Lemma 2.2. By the assumption of the lemma and (2.1) we have

$$T\big[i-1,\lfloor j+\log_Q \alpha\rfloor\big] \geq \tau(i-1,Q^{\lfloor j+\log_Q\alpha\rfloor-(i-1)}) \geq \tau(i-1,\alpha Q^{j-i}) \tag{3.3}$$

and

$$\begin{aligned} T\big[i-1,\lfloor j+\log_Q(1-\alpha)\rfloor\big] &\geq \tau\left(i-1,Q^{\lfloor j+\log_Q(1-\alpha)\rfloor-(i-1)}\right) \\ &\geq \tau(i-1,(1-\alpha)Q^{j-i}). \end{aligned} \tag{3.4}$$

Combining (3.3) and (3.4) with min and max operators we obtain

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

establishing that $T[i,j]$ computed using (2.3) satisfy the lower bound in (2.4).

By the assumption of the lemma and (2.1) we have

$$T\big[i-1,\lfloor j+\log_Q\alpha\rfloor\big] \leq \tau(i-1,Q^{\lfloor j+\log_Q\alpha\rfloor}) \leq \tau(i-1,\alpha Q^j) \tag{3.5}$$

and

$$T\big[i-1,\lfloor j+\log_Q(1-\alpha)\rfloor\big] \leq \tau(i-1,Q^{\lfloor j+\log_Q(1-\alpha)\rfloor}) \leq \tau(i-1,(1-\alpha)Q^j). \tag{3.6}$$

Combining (3.5) and (3.6) with min and max operators we obtain

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

establishing that $T[i,j]$ computed using (2.3) satisfy the upper bound in (2.4). □

4. The bit complexity of the algorithm. The only arithmetic operations used on the inputs are in step 3 (the addition of w_i and the comparisons between the T's) and in step 4 (the comparison with C). These can be implemented using $O(\log C)$ bit operations and hence contribute $O((n^3/\varepsilon)\log(n/\varepsilon)\log C)$ to the total number of bit operations of the algorithm. As we mentioned in Remark 1 we also have to implement the computation of $\lfloor \log_Q \alpha \rfloor$.

To analyze the bit complexity of computing $\lfloor \log_Q \alpha \rfloor$ it will be convenient to choose a different value of Q. We will choose Q such that $1/Q = 1 - 2^{-t}$, where $t = 1 + \lceil \log_2(2(n+1)/\varepsilon) \rceil$. Note that

$$Q \leq 1 + \varepsilon/(2(n+1)). \tag{4.1}$$

The problem of computing $\lfloor \log_Q \alpha \rfloor$ in step 3 of the algorithm ((2.3) for $\alpha \in S_2$ in the top part and for $\alpha \in S_1$ in the bottom part) is equivalent to the following problem: given j, find the smallest k such that $(1/Q)^k + (1/Q)^j \leq 1$. (This follows from the fact that for $k = -\lfloor \log_Q(1-(1/Q)^j) \rfloor$ we have $(1/Q)^k \leq 1-(1/Q)^j$ and $(1/Q)^{k-1} > 1-(1/Q)^j$.)

The range of k and j we are interested in is restricted as follows. If $j \geq 2^t t$, then we can take $k = 1$ (since $(1/Q)^j + 1/Q \leq \exp(-t) + 1 - 2^{-t} < 1$). For any $j \geq 1$ we have $k \leq 2^t t$ (this follows from the inequality from the previous sentence). Thus for the relevant j (and k) we have

$$(1/Q)^j \geq (1-2^{-t})^{-2^t t} \geq \exp(-t). \tag{4.2}$$

We will (approximately) compute $(1/Q)^j$ for $j = 0, \ldots, s$ using $B := 5t$ bits of precision. More precisely we will compute integers $Q_0, \ldots, Q_s$ such that

$$Q_j \leq 2^B (1/Q)^j \leq Q_j + j. \tag{4.3}$$

Note that if (4.3) is satisfied, then for

$$Q_{j+1} := \lfloor Q_j - Q_j 2^{-t} \rfloor \tag{4.4}$$

we have

$$Q_{j+1} \leq Q_j - Q_j 2^{-t} \leq 2^B (1/Q)^{j+1} \leq Q_j(1-2^{-t}) + j(1-2^{-t}) \leq Q_{j+1} + j + 1.$$

The computation in (4.4) can be implemented using $O(B)$ bit operations and hence we can compute all $Q_0, \ldots, Q_s$ in $O(Bs) = O((n^2/\varepsilon)\log(n/\varepsilon))$ time.

From (4.3) and (4.2) we have $Q_j + j \geq 2^{5t}\exp(-t)$, and since $j \leq 2^t t \leq 2^{5t-1}\exp(-t)$ we can conclude

$$Q_j \geq 2^{5t-1}\exp(-t) \geq 2^{3t}. \tag{4.5}$$

From (4.4) and (4.5) we have

$$Q_j - Q_{j+1} \geq 2^{-t} Q_j \geq 2^{2t}. \tag{4.6}$$

Now we return to the problem of computing $\lfloor \log_Q(1-(1/Q)^j) \rfloor$ for given j. We will find the smallest k such that

$$Q_j + Q_k + j + k \leq 2^B. \tag{4.7}$$

Then from (4.3) we have

$$(1/Q)^j + (1/Q)^k \leq 1. \tag{4.8}$$

By the choice of k we have

$$Q_j + Q_{k-1} + j + (k-1) > 2^B,$$

which, using (4.6), implies

$$Q_j + Q_{k-2} > 2^B,$$

and hence, using (4.3), we have

$$(1/Q)^j + (1/Q)^{k-2} > 1. \tag{4.9}$$

From (4.8) and (4.9) we have that $\lfloor \log_Q(1-(1/Q)^j) \rfloor$ is either k or $k+1$. Using k instead of $\lfloor \log_Q(1-(1/Q)^j) \rfloor$ in the third step of the algorithm will result in a further (minor) loss in precision in Lemma 2.2—the argument Q^{j-i} in (2.4) will be replaced by argument Q^{j-2i} (see (4.11) below). We obtain the following analogue of Lemma 2.2 with the above modifications.

LEMMA 4.1. *Let $i \geq 1$. Assume that for all $j \in \{0, \ldots, s\}$ we have that $T[i-1, j]$ satisfy* (4.11). *Then for all $j \in \{0, \ldots, s\}$ we have that $T[i, j]$ computed using*

$$T[i,j] = \min_{\alpha \in [0,1]} \max \begin{cases} T\big[i-1, j + F(Q,\alpha)\big], \\ T\big[i-1, j + F(Q, 1-\alpha)\big] + w_i, \end{cases} \tag{4.10}$$

where $\lfloor \log_Q \alpha \rfloor - F(Q,\alpha) \in \{0,1\}$ satisfies

$$\tau(i, Q^{j-2i}) \leq T[i,j] \leq \tau(i, Q^j). \tag{4.11}$$

Proof. The proof is the same as the proof of Lemma 2.2; the only difference is in (3.3) and (3.4) (where we lose additional additive one in the exponent, since now we use inequality $F(Q,\alpha) \geq (\log_Q \alpha) - 2$ instead of $\lfloor \log_Q \alpha \rfloor \geq (\log_Q \alpha) - 1$). □

Proof of Theorem 1.1. The correctness of the modified algorithm outlined in this section follows immediately from Lemma 4.1, as is done using Lemma 2.2 at the end of section 2 (equation (2.5)), using the bound (4.1).

The computation of k for all $j \in \{0, \ldots, s\}$ can be done in $O(Bs)$ time since we do comparisons only on B-bit numbers (the total number of comparisons is $O(s)$, since to find the k for $j+1$ we start from the k for j). Thus the computation of $\lfloor \log_Q \alpha \rfloor$ terms used by the algorithm is within the claimed $O(n^3/\varepsilon \log(n/\varepsilon))$ bit operations. The number of addition and comparison operations on inputs was shown to be $O(n^3/\varepsilon \log(n/\varepsilon))$ in section 2.1 (each of them has only $O(\log C)$ bits and can be done using $O(\log C)$ bit operations). □

REFERENCES

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

[2] M. Ajtai and A. Wigderson, *Deterministic simulation of probabilistic constant depth circuits*, Adv. Comput. Res. Randomness Computation, 5 (1989), pp. 199–223.

[3] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, Cambridge University Press, Cambridge, UK, 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), 2007, pp. 122–127.

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

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

[7] 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 (1993), pp. 271–284.

[8] A. Galanis, Q. Ge, D. Štefankovič, E. Vigoda, and L. Yang, *Improved inapproximability results for counting independent sets in the hard-core model*, in Proceedings of the 15th International Workshop on Randomization and Computation (RANDOM), 2011, pp. 567–578.

[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), 2007, pp. 1245–1254.

[10] P. Gopalan, A. Klivans, and R. Meka, *Polynomial-time approximation schemes for knapsack and related counting problems using branching programs*, arXiv:1008.3187, 2010.

[11] P. Gopalan, A. Klivans, R. Meka, D. Štefankovič, S. Vempala, and E. Vigoda, *An FPTAS for #Knapsack and related counting problems*, Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS), 2011, pp. 817–826.

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

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

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

[15] M. Jerrum and A. Sinclair, *The Markov chain Monte Carlo method: An approach to approximate counting and integration*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS Publishing, Boston, 1996, pp. 482–520.

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

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

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

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

[20] M. Luby and B. Veličković, *On deterministic approximation of DNF*, Algorithmica, 16 (1996), pp. 415–433.

[21] N. Megiddo, *On the complexity of linear programming*, in Advances in Economic Theory, Econom. Soc. Monogr. 12, Cambridge, UK, 1989, pp. 225–268.

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

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

[24] J. B. Orlin, *A faster strongly polynomial minimum cost flow algorithm*, Oper. Res., 41 (1993), pp. 338–350.

[25] A. Schrijver, *A combinatorial algorithm minimizing submodular functions in strongly polynomial time*, J. Combin. Theory Ser. B, 80 (2000), pp. 346–355.

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

[27] D. Štefankovič, S. Vempala, and E. Vigoda, *Adaptive simulated annealing: A near-optimal connection between sampling and counting*, J. ACM, 56 (2009), pp. 1–36.
[28] D. Štefankovič, S. Vempala, and E. Vigoda, *A deterministic polynomial-time approximation scheme for counting knapsack solutions*, arXiv:1008.1687, 2010.
[29] É. Tardos, *A strongly polynomial minimum cost circulation algorithm*, Combinatorica, 5 (1985), pp. 247–255.
[30] D. Weitz, *Counting independent sets up to the tree threshold*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC), 2006, pp. 140–149.