A Deterministic Fully Polynomial Time Approximation Scheme For Counting Integer Knapsack Solutions Made Easy

Nir Halman
Hebrew University of Jerusalem
halman@huji.ac.il

July 3, 2016

Abstract

Given n elements with nonnegative integer weights $w = (w_1, \ldots, w_n)$, an integer capacity C and positive integer ranges $u = (u_1, \ldots, u_n)$, we consider the counting version of the classic integer knapsack problem: find the number of distinct multisets whose weights add up to at most C. We give a deterministic algorithm that estimates the number of solutions to within relative error ϵ in time polynomial in $n, \log U$ and $1/\epsilon$, where $U = \max_i u_i$. More precisely, our algorithm runs in $O(\frac{n^3 \log^2 U}{\epsilon} \log \frac{n \log U}{\epsilon})$ time. This is an improvement of n^2 and $1/\epsilon$ (up to log terms) over the best known deterministic algorithm by Gopalan *et al.* [FOCS, (2011), pp. 817-826]. Our algorithm is relatively simple, and its analysis is rather elementary. Our results are achieved by means of a careful formulation of the problem as a dynamic program, using the notion of *binding constraints*.

Keywords: Approximate counting, integer knapsack, dynamic programming, bounding constraints, K-approximating sets and functions.

1 Introduction

In this paper we target at designing a deterministic fully polynomial time approximation scheme (FPTAS) for one of the most basic #P-complete counting problems – counting the number of integer knapsack solutions. Given n elements with nonnegative integer weights $w = (w_1, \ldots, w_n)$, an integer capacity C, and positive integer ranges $u = (u_1, \ldots, u_n)$, we consider the counting version of the classic integer knapsack problem: find the size of the set of feasible solutions $\text{KNAP}(w, C, u) = \{x \mid \sum_{i \le n} w_i x_i \le C,\ 0 \le x_i \le u_i\}$. (We assume, w.l.o.g., that $w_i u_i \le C$ for all i.) We give a deterministic FPTAS for this problem that for any tolerance $\epsilon > 0$ estimates the number of solutions within relative error ϵ in time polynomial in the (binary) input size and $1/\epsilon$.

Our result. Our main result is the following theorem (the base of the logarithms in this paper are all 2 unless otherwise specified).

Theorem 1.1 *Given a knapsack instance $KNAP(w, C, u)$ with $U = \max_i u_i$ and $\epsilon > 0$, there is a deterministic $O(\frac{n^3 \log^2 U}{\epsilon} \log \frac{n \log U}{\epsilon})$ algorithm that computes an ϵ-relative error approximation for $|KNAP(w, C, u)|$.*

Relevance to existing literature. The field of approximate counting is largely based on Markov Chain Monte Carlo Sampling [1], a technique that is inherently randomized, and has had remarkable success, see [2] and the references therein. The first approximation schemes for counting integer knapsack solutions are fully polynomial *randomized* approximation schemes (FPRASs). Given parameters $\epsilon > 0$ for the error tolerance and $1 > \delta > 0$ for the failure probability, the FPRAS returns a solution which is correct with probability at least $1 - \delta$, and the running time is required to be polynomial in the (binary) input size, $1/\epsilon$ and in $\log(1/\delta)$. To the best of our knowledge, the best FPRAS up to date is given by Dyer [3], and is achieved by combining dynamic programming (DP, to be distinguished from dynamic program by context) with simple rejection sampling. The complexity of the algorithm is $O(n^5 + n^4/\epsilon^2)$, so in fact the algorithm is strongly polynomial (see, e.g., [4]), that is, the number of arithmetic operations is polynomial in n and independent of C, U.

To the best of our knowledge, the currently best (deterministic) FPTAS for this problem is given by Gopalan *et al.* [5], and has complexity $O(\frac{n^5}{\epsilon^2} \log^2 U \log W)$, where $W = \sum_i w_i u_i + C$ (see also [6]). We note that the real achievement of [6] is providing an FPTAS for the *multidimensional* version of the problem. Because of this reason they use a somewhat more sophisticated approach than ours, relying on read-once branching programs and insight from Meka and Zuckerman [7].

We note in passing that the first (deterministic) FPTAS for counting 0/1 knapsack solutions (i.e., our problem restricted to the case where $u = (1, \dots, 1)$) is given by Štefankovič *et al.* [2] and runs in $O(n^3 \epsilon^{-1} \log(n/\epsilon))$ time. The currently best (deterministic) FPTAS runs in $O(n^3 \epsilon^{-1} \log(1/\epsilon)/ \log n)$ time [8].

Technique used. In this paper we give two FPTASs that are based upon formulating the counting problem as a DP. Instead of deciding at once how many copies of item i to put in the knapsack, we split the decision into a sequence of at most $\log u_i$ binary sub-decisions concerning (not necessarily all the) bundles of $1, 2, 4, \dots, 2^{\lfloor \log u_i \rfloor}$ copies of the item. In order to "translate" this into a DP, we use the idea of what we call *binding constraints*, as explained in detail below. The first FPTAS uses a "primal" DP formulation and approximates it via the recent technique of K-approximation sets and functions introduced by [9], which we overview in Section 2.1. The second FPTAS uses a "dual" DP formulation and approximates it in a similar way [2] approximate the 0/1 knapsack problem. We overview their solution in Section 3.1.

Our contribution. While not strongly polynomial, the running time of our solutions are of order n and $1/\epsilon$ (up to log terms) faster than the (randomized, but strongly-polynomial) algorithm of Dyer [3]. The complexity of our solutions is also better by factors of n^2 and $1/\epsilon$ (up to log terms) than the (non strongly-polynomial, but deterministic) algorithm of Gopalan *et al.* [5]. Moreover, our algorithms are relatively simple and their analysis is rather elementary. A second contribution is our new DP technique – "binding constraints", which may be of independent interest.

Organization of the paper. In Section 2 we present an FPTAS which is based upon a primal DP formulation of the problem. Our second FPTAS, based upon a dual DP formulation, is given in Section 3. In this way we showcase that the idea of binding constraints is useful for the primal as well as the dual DP formulation.

2 Algorithm via a primal DP formulation

A pseudo-polynomial algorithm is achieved using the following recurrence:

$$\begin{array}{lll} s_i(j) & = \sum_{k=0}^{m_i(j)} s_{i-1}(j-kw_i) & 2 \leq i \leq n,\ j=1,\dots,C, \\ s_1(j) & = m_1(j)+1 & j=1,\dots,C, \end{array} \quad (1)$$

where function $m_i : [0,\dots,C] \rightarrow \mathbb{Z}^+$ is defined as $m_i(j) := \max\{x \in \mathbb{Z}^+ \mid x \leq u_i,\ xw_i \leq j\}$ and returns the maximum number of copies of item i that can be placed in a knapsack with capacity j. Here $s_i(j)$ is the number of integer knapsack solutions that use a subset of the items $\{1,\dots,i\}$ whose weights sum up to at most j. The solution of the counting problem is therefore $s_n(C)$. The complexity of this pseudo-polynomial algorithm is $O(nUC)$, i.e., exponential in both the (binary) sizes of U and C. We call such formulation *primal* because the range of the functions in (1) is the number of solutions.

In order to get our FPTAS we give in Section 2.2 a more careful DP formulation which is exponential only in the (binary) size of C. Before doing so, we briefly overview the technique of K-approximation sets and functions in Section 2.1. We use this technique in order to get our first FPTAS.

2.1 K-approximation sets and functions

Halman *et al.* [9] have introduced the technique of K-approximation sets and functions, and used it to develop an FPTAS for a certain stochastic inventory control problem. Halman *et al.* [10] have applied this tool to develop a framework for constructing FPTASs for a rather general class of stochastic DPs. This technique has been used to yield FPTASs to various optimization problems, see [10] and the references therein. In this section we provide an overview of the technique of K-approximation sets and functions. In the next section we use this tool to construct FPTASs for counting the number of solutions of the integer knapsack problem. To simplify the discussion, we modify Halman *et al.*'s definition of the K-approximation function by restricting it to integer-valued nondecreasing functions.

Let $K \geq 1$, and let $\varphi : \{0,\dots,B\} \rightarrow Z^+$ be an arbitrary function. We say that $\tilde{\varphi} : \{0,\dots,B\} \rightarrow Z^+$ is a *K-approximation function* of φ if $\varphi(x) \leq \tilde{\varphi}(x) \leq K\varphi(x)$ for all $x = 0,\dots,B$. The following property of K-approximation functions is extracted from Proposition 5.1 of [10], which provides a set of general computational rules of K-approximation functions. Its validity follows directly from the definition of K-approximation functions.

Property 2.1 *For $i = 1,2$ let $K_i \geq 1$, let $\varphi_i : \{0,\dots,B\} \rightarrow Z^+$ and let $\tilde{\varphi}_i : \{0,\dots,B\} \rightarrow Z^+$ be a K_i-approximation of φ_i. The following properties hold:*

Summation of approximation: *$\tilde{\varphi}_1 + \tilde{\varphi}_2$ is a $\max\{K_1,K_2\}$-approximation function of $\varphi_1 + \varphi_2$.*

Approximation of approximation: *If $\varphi_2 = \tilde{\varphi}_1$ then $\tilde{\varphi}_2$ is a K_1K_2-approximation function of φ_1.*

Let $K > 1$. Let $\varphi : \{0, \ldots, B\} \to Z^+$ be a nondecreasing function and $W = \{k_1, k_2, \ldots, k_r\}$ be a subset of $\{0, \ldots, B\}$, where $0 = k_1 < k_2 < \cdots < k_r = B$. We say that W is a K-*approximation set* of φ if $\varphi(k_{j+1}) \leq K\varphi(k_j)$ for each $j = 1, 2, \ldots, r-1$ that satisfies $k_{j+1} - k_j > 1$. This means that the *values* of φ on consecutive points of the approximation set essentially form a geometric progression with ratio of approximately K. (Consecutive points of the approximation set *itself* do not necessarily form a geometric sequence.) It is easy to see that given φ, there exists a K-approximation set of φ with cardinality $O(\log_K M)$, where M is any constant upper bound of $\varphi(\cdot)$. Furthermore, this set can be constructed in $O\big((1 + t_\varphi) \log_K M \log_2 B\big)$ time, where t_φ is the amount of time required to evaluate φ (see [10, Prop. 4.6] for a formal proof).

Given φ and a K-approximation set $W = \{k_1, k_2, \ldots, k_r\}$ of φ, a K-approximation function of φ can be obtained easily as follows [10, Def.4.4]: Define $\hat{\varphi} : \{0, \ldots, B\} \to Z^+$ such that

$$\hat{\varphi}(x) = \varphi(k_j) \qquad k_{j-1} < x \leq k_j \text{ and } j = 2, \ldots, r,$$

and that

$$\hat{\varphi}(k_1) = \varphi(k_1).$$

Note that $\varphi(x) \leq \hat{\varphi}(x) \leq K\varphi(x)$ for $x = 0, \ldots, B$. Therefore, $\hat{\varphi}$ is a nondecreasing K-approximation function of φ. We say that $\hat{\varphi}$ is the *K-approximation function of φ induced by W*.

The procedure for the construction of a K-approximation function $\tilde{\varphi}$ for φ is stated as Algorithm 1[1].

Algorithm 1 FUNCTION COMPRESS(φ, K) returns a step nondecreasing K-approximation of φ

```
1: Function Compress(φ, K)
2: obtain a K-approximation set W of φ
3: return the K-approximation function of φ induced by W
```

By applying approximation of approximation in Property 2.1 and the discussion above we get the following result (see also [10, Prop. 4.5]).

Proposition 2.2 *Let $K_1, K_2 \geq 1$ be real numbers, $M > 1$ be an integer, and let $\varphi : [0, \ldots, B] \to [0, \ldots, M]$ be a nondecreasing function. Let $\bar{\varphi}$ be a nondecreasing K_2-approximation function of φ. Then Function* COMPRESS*($\bar{\varphi}, K_1$) returns in $O((1 + t_{\bar{\varphi}})(\log_K M \log B))$ time a nondecreasing step function $\tilde{\varphi}$ with $O(\log_{K_1} M)$ steps which K_1K_2-approximates φ. The query time of $\tilde{\varphi}$ is $O(\log \log_{K_1} M)$ if it is stored in a sorted array $\{(x, \tilde{\varphi}) \mid x \in W\}$.*

2.2 A more efficient DP formulation

In this section we reformulate (1) as a DP that can be solved in time pseudo-polynomial in the (binary) size of C only. As explained in the Introduction, instead of deciding at once how many copies of item i to put in the knapsack, we break the decision into $\lfloor \log^+ m_i(j) \rfloor + 1$ binary sub-decisions. Sub-decision $\ell = 1, \ldots, \lfloor \log^+ m_i(j) \rfloor + 1$ checks the possibility of putting $2^{\ell-1}$ copies of item i in the knapsack. We do so using, what we call, the idea of *binding constraints*. For $\ell \geq 1$ let $z_{i,\ell,0}(j)$ be the number of solutions for a knapsack of

[1]The author thanks Jim Orlin for suggesting the presentation of this function, as well as the term "Compress".

capacity j that use a subset of the items $\{1,\ldots,i\}$, put no more than $2^\ell - 1$ copies of item i, and no more than u_k copies of item k, for $k=1,\ldots,i-1$. For $\ell \geq 1$ let $z_{i,\ell,1}(j)$ be the number of solutions for a knapsack of capacity j that use a subset of the items $\{1,\ldots,i\}$, put no more than $u_i \mod 2^\ell$ copies of item i, and no more than u_k copies of item k, for $k=1,\ldots,i-1$. In this way, considering the third index of $z_{i,\ell,r}(j)$, if $r=0$ then the constraint $x \leq u_i$ is assumed to be *non binding*. If, on the other hand, $r=1$ then the constraint $x \leq u_i$ may be *binding*. Before giving the formal recurrences we need a few definitions. Let $\log^+ x := \max\{0, \log x\}$. Let $\text{msb}(x,i) := \lfloor \log(x \mod 2^i)\rfloor + 1$. $\text{msb}(x,i)$ is therefore the most significant 1-digit of $(x \mod 2^i)$ if $(x \mod 2^i) > 0$, and is $-\infty$ otherwise. E.g., $\text{msb}(5,2)=1$ and $\text{msb}(4,1)=-\infty$. Our recurrences are as follows:

$$z_{i,\ell,0}(j) = z_{i,\ell-1,0}(j) + z_{i,\ell-1,0}(j - 2^{\ell-1}w_i) \qquad \ell = 2,\ldots,\lfloor \log^+ m_i(j)\rfloor + 1\,, \tag{2a}$$

$$z_{i,\ell,1}(j) = z_{i,\ell-1,0}(j) + z_{i,\text{msb}(u_i,\ell-1),1}(j - 2^{\ell-1}w_i) \qquad \ell = 2,\ldots,\lfloor \log^+ m_i(j)\rfloor + 1\,, \tag{2b}$$

$$z_{i,1,r}(j) = \begin{array}{l} z_{i-1,\lfloor \log^+ m_{i-1}(j)\rfloor+1,1}(j) + \\ + z_{i-1,\lfloor \log^+ m_{i-1}(j-w_i)\rfloor+1,1}(j-w_i)\,, \end{array} \tag{2c}$$

$$z_{i,-\infty,1}(j) = z_{i-1,\lfloor \log^+ m_{i-1}(j)\rfloor+1,1}(j)\,, \tag{2d}$$

$$z_{1,\ell,r}(j) = m_1(j) + 1 \qquad \ell = 1,\ldots,\lfloor \log^+ m_1(j)\rfloor + 1\,, \tag{2e}$$

$$z_{i,\ell,r}(j) = 0 \qquad j < 0, \tag{2f}$$

where $r=0,1$, $i=2,\ldots,n$, and $j=0,\ldots,C$, unless otherwise specified. The solution of the counting problem is therefore $z_{n,\lfloor \log u_n\rfloor+1,1}(C)$. The time needed to solve this program is only $O(nC\log U)$.

We now explain the six equations in formulation (2) in more detail. Equation (2a) deals with the case where the constraint $x \leq u_i$ is non binding, so putting $2^\ell - 1$ more copies of item i in a knapsack of remaining capacity j is a feasible possibility. Clearly, in the following steps the constraint $x \leq u_i$ remains non binding. As for equation (2b), it deals with the case where the constraint $x \leq u_i$ may be binding when putting $2^{\ell-1}$ copies of item i in the knapsack. If we do put this number of copies, the constraint may be binding, otherwise it is assured to be non binding. Equation (2c) deals with the possibility of putting an odd number of copies of item i in the knapsack. Equation (2d) is only called by equation (2b), when *exactly* u_i copies of item i are put in the knapsack. Equation (2e) deals with the initial condition of one element only, and the last equation deals with the boundary condition that there is not enough capacity in the knapsack.

In order to design an FPTAS to our problem, we first extend the DP formulation (2) to any integer positive index ℓ by letting $z_{i,\ell,r}(j) = 0$ for $i=1,\ldots,n, r=0,1$ and $\ell > \lfloor \log^+ m_i(j)\rfloor + 1$. (Note that without this extension $z_{i,\ell,r}(\cdot)$ is not necessarily defined over the entire interval $(-\infty,\ldots,C]$. Moreover, this extended formulation assures that $z_{i,\ell,r}(\cdot)$ is monotone nondecreasing.) We denote this extended set of recurrences by (3). The solution of the counting problem via (3) remains $z_{n,\lfloor \log u_n\rfloor+1,1}(C)$.

2.3 Algorithm statement

The idea behind our approximation algorithm is to compute an approximation for $z_{n,\lfloor \log u_n\rfloor+1,1}(C)$ by using the recurrences in (3). This is done by recursively computing K-approximation functions for the $O(\sum_{i=1}^n \lfloor \log u_i\rfloor)$ different functions in (3). Due to approximation of approximation in Property 2.1 there is a deterioration of at most factor K between the ratio of approximation of $z_{i,\ell,r}$ and that of $z_{i,\ell-1,r}$ (for $\ell > 1$), as well as between the ratio of approximation of $z_{i,1,r}$ and that of $z_{i-1,\lfloor \log u_{i-1} \rfloor+1,r}$. Therefore, by choosing $K = \sqrt[(n-1)(\lfloor \log U \rfloor+1)+1]{1+\epsilon}$ one gets that the total accumulated multiplicative error over the entire algorithm does not exceed $1+\epsilon$. For a given instance (w, C, u) of the integer knapsack problem and a tolerance parameter $\epsilon \in (0, 1]$, our approximation algorithm is formally given as Function CountIntegerKnapsackPrimal(w, C, u, ϵ), see Algorithm 2. (From hereon after we use the notation $z(\cdot)$, where the "$\cdot$" stands for the argument of function z. E.g., the value of $z(\cdot - w)$ for variable value 2 is $z(2-w)$. Put it differently, the function z is shifted by $-w$.)

Algorithm 2 FPTAS for counting integer knapsack.

1: **Function CountIntegerKnapsackPrimal**(w, C, u, ϵ)
2: $K \leftarrow \sqrt[(n-1)(\lfloor \log U \rfloor+1)+1]{1+\epsilon}$
3: **for** $\ell := 1$ **to** $\lfloor \log u_1 \rfloor + 1$ **and** $r = 0, 1$ **do** $\tilde{z}_{1,\ell,r} \leftarrow$ Compress$(z_{1,\ell,r}, K)$ /* $z_{1,\ell,r}$ as defined in (3) */
4: **for** $i := 2$ **to** n **do**
5: $\quad \tilde{z}_{i,-\infty,1}(\cdot) \leftarrow \tilde{z}_{i-1,\lfloor \log^+ m_{i-1}(\cdot) \rfloor+1,1}(\cdot)$
6: $\quad$ **for** $r = 0, 1$ **do** $\tilde{z}_{i,1,r}(\cdot) \leftarrow$ Compress$(\tilde{z}_{i-1,\lfloor \log^+ m_{i-1}(\cdot) \rfloor+1,1}(\cdot) + \tilde{z}_{i-1,\lfloor \log^+ m_{i-1}(\cdot - w_i) \rfloor+1,1}(\cdot - w_i), K)$
7: $\quad$ **for** $\ell := 2$ **to** $\lfloor \log u_i \rfloor + 1$ **do**
8: $\quad\quad \tilde{z}_{i,\ell,0}(\cdot) \leftarrow$ Compress$(\tilde{z}_{i,\ell-1,0}(\cdot) + \tilde{z}_{i,\ell-1,0}(\cdot - 2^{\ell-1} w_i), K)$
9: $\quad\quad \tilde{z}_{i,\ell,1}(\cdot) \leftarrow$ Compress$(\tilde{z}_{i,\ell-1,0}(\cdot) + \tilde{z}_{i,\mathrm{msb}(u_i,\ell-1),1}(\cdot - 2^{\ell-1} w_i), K)$
10: $\quad$ **end for**
11: **end for**
12: **return** $\tilde{z}_{n,\lfloor \log u_n \rfloor+1,1}(C)$

2.4 Algorithm analysis

Note first that all functions in (3) are nondecreasing in j. Hence, all calls to function Compress are valid.

We next prove that $\tilde{z}_{n,\lfloor \log^+ m_n(C) \rfloor+1,1}(C)$ returned by Algorithm 2 is a relative $(1+\epsilon)$-approximation of $|\mathrm{KNAP}(w, C, u)|$. To do so, we first show by double induction over i and ℓ that $\tilde{z}_{i,\ell,r}$ is a $K^{(i-2)(\lfloor \log U \rfloor+1)+\ell+1}$-approximation of $z_{i,\ell,r}$ for $i = 2, \ldots, n$, $\ell = 0, \ldots, \lfloor \log u_i \rfloor + 1$ and $r = 0, 1$. (For the purpose of the induction we treat the case of $\ell = -\infty$ as if $\ell = 0$.) Note that due to step 3, $\tilde{z}_{1,\ell,r}$ are K-approximations of $z_{1,\ell,r}$.

We first treat the base case of $i = 2$ and $\ell < 2$. Considering step 5, due to step 3, we get that $\tilde{z}_{2,-\infty,1}$ is a K-approximation of $z_{2,-\infty,1}$ as needed. Considering step 6, due to summation of approximation applied with parameters set to $\varphi_1(\cdot) = z_{1,\lfloor \log^+ m_1(\cdot) \rfloor+1,1}(\cdot)$, $\varphi_2(\cdot) = z_{1,\lfloor \log^+ m_1(\cdot - w_2) \rfloor+1,1}(\cdot - w_2)$, $\tilde{\varphi}_1(\cdot) = \tilde{z}_{1,\lfloor \log^+ m_1(\cdot) \rfloor+1,1}(\cdot)$, $\tilde{\varphi}_2(\cdot) = \tilde{z}_{1,\lfloor \log^+ m_1(\cdot - w_2) \rfloor+1,1}(\cdot - w_2)$ and $K_1 = K_2 = K$, we get that the function inside the Compress operator is a K-approximation of $z_{2,1,r}$. Applying Proposition 2.2 with parameters set to $\varphi(\cdot) = z_{2,1,r}(\cdot)$, $\bar{\varphi}(\cdot) = \tilde{\varphi}_1(\cdot) + \tilde{\varphi}_2(\cdot)$, $K_1 = K_2 = K$ we get that $\tilde{z}_{2,1,r}$ is a K^2-approximation of $z_{2,1,r}$ as needed.

We now perform an induction on ℓ. For the base case of $\ell = 2$, repeating the same arguments for steps 8-9 we get that $\tilde{z}_{2,2,r}$ is a K^3-approximation of $z_{2,2,r}$, and by induction on ℓ, that $\tilde{z}_{2,\ell,r}$ is a $K^{\ell+1}$-approximation of $z_{2,\ell,r}$ as needed, thus proving the base case of $i = 2$. The induction hypothesis is that $\tilde{z}_{i,\ell,r}$ is a $K^{(i-2)(\lfloor \log U \rfloor+1)+\ell+1}$-approximation of $z_{i,\ell,r}$.

We show next that $\tilde{z}_{i+1,\ell,r}$ is a $K^{(i-1)(\lfloor \log U \rfloor +1)+\ell+1}$-approximation of $z_{i+1,\ell,r}$. Due to summation of approximation applied with parameters set to $\varphi_1(\cdot) = z_{i,\lfloor \log^+ m_i(\cdot) \rfloor+1,1}(\cdot)$, $\varphi_2(\cdot) = z_{i,\lfloor \log^+ m_i(\cdot - w_{i+1}) \rfloor+1,1}(\cdot - w_{i+1})$ and $K_1 = K_2 = K^{(i-1)(\lfloor \log U \rfloor+1)+1}$ (note that the choice of K_1, K_2 is done by applying the induction hypothesis and using the inequalities $\log^+ m_i(\cdot - w_{i+1}) \leq \log^+ m_i(\cdot) \leq \log u_i \leq \log U$) we get that $\tilde{\varphi}_1 + \tilde{\varphi}_2$ is a $K^{(i-1)(\lfloor \log U \rfloor+1)+1}$-approximation of $z_{i+1,1,r}$. Applying once more Proposition 2.2 with parameters set to $\varphi = z_{i+1,1,r}$, $\bar{\varphi} = \tilde{\varphi}_1 + \tilde{\varphi}_2$, $K_1 = K^{(i-1)(\lfloor \log U \rfloor+1)+1}$ and $K_2 = K$ we get that $\tilde{z}_{i+1,1,r}$ is a $K^{(i-1)(\lfloor \log U \rfloor+1)+2}$-approximation of $z_{i+1,1,r}$ as needed. We now perform an induction on ℓ. For the base case of $\ell = 2$, repeating the same arguments for steps 8-9 we get that $\tilde{z}_{i+1,2,r}$ is a $K^{(i-1)(\lfloor \log U \rfloor+1)+3}$-approximation of $z_{i+1,2,r}$, and by induction on ℓ, that $\tilde{z}_{i+1,\ell,r}$ is a $K^{(i-1)(\lfloor \log U \rfloor+1)+\ell+1}$-approximation of $z_{i+1,\ell,r}$. This completes the proof by induction. Recalling the value of K as set in step 2 and taking $i = n$ we get that $\tilde{z}_{n,\lfloor \log u_n \rfloor+1,1}$ is indeed a $(1+\epsilon)$-approximation of $z_{n,\lfloor \log u_n \rfloor+1,1}$, and specifically that $\tilde{z}_{n,\lfloor \log u_n \rfloor+1,1}(C)$ is indeed a $(1+\epsilon)$-approximation of the solution of the counting problem.

It remains to analyze the complexity of the algorithm. Clearly, the running time of the algorithm is dominated by the operations done in the inner "for" loop, i.e., steps 8-9, which are executed $O(n \log U)$ times. We analyze, w.l.o.g., a single execution of step 8. By Proposition 2.2, the query time of each of the $\tilde{z}_{i,\ell-1,0}(\cdot)$ and $\tilde{z}_{i,\ell-1,0}(\cdot)$ is $O(\log \log_K M)$, where M is an upper bound on the counting problem, e.g., $M = U^n$. Therefore, applying again Proposition 2.2, each call to Compress runs in $O(\log_K M \log C \log \log_K M)$ time. Using the inequality $(1 + \frac{x}{n})^n \leq 1 + 2x$ which holds for $0 \leq x \leq 1$ we get that $K \geq 1 + \frac{\epsilon}{2\left((n-1)(\lfloor \log U \rfloor+1)+1\right)}$. Using the inequality $\log(1+y) \geq y$ which holds for $y \in [0,1]$, and changing the bases of the logarithms to two, we get that the overall running time of the algorithm is $O(\frac{n^3}{\epsilon} \log^3 U \log C \log \frac{n \log U}{\epsilon})$.

3 Algorithm via a dual DP formulation

In this section we provide an FPTAS to counting integer knapsack solutions using the analysis of [2] for counting 0/1 knapsack solutions. Our FPTAS will be faster than the one presented in the previous section by a factor of $\log U \log C$.

3.1 The 0/1 knapsack

In this section we present the main ideas used to derive the FPTAS to counting 0/1 knapsack solutions [2, Sec. 2]. Štefankovič *et al.* [2] begin by defining a dual DP formulation as follows. For $i = 1, \ldots, n$ let $\tau_i(a)$ be the smallest capacity C such that there exist at least a solutions to the knapsack problem with items $1, 2, \ldots, i$ and capacity C. Using standard conventions, the value of τ_0 is given by

$$\tau_0(a) = \begin{cases} -\infty & \text{if } a = 0\,, \\ 0 & \text{if } 0 < a \leq 1\,, \\ \infty & \text{otherwise.} \end{cases} \tag{4}$$

It follows that the number of knapsack solutions satisfies $Z = \max\{a \mid \tau_n(a) \leq C\}$. [2, Lem. 2.1] states that $\tau_i(a)$ satisfies the following recurrence:

$$\tau_i(a) = \min_{\alpha \in [0,1]} \max \begin{cases} \tau_{i-1}(\alpha a), \\ \tau_{i-1}((1-\alpha)a) + w_i\,. \end{cases} \tag{5}$$

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. The recursion tries all possible values of α and take the one that yields the smallest (optimal) value for $\tau_i(a)$. We call such formulation *dual* because the range of the functions in (4)-(5) is the capacity of the knapsack.

[2] then move to an approximation of τ that can be computed efficiently and define function $T : \{0,\dots,s\} \rightarrow \mathbb{R}^+ \cup \{\infty\}$ which only considers a small subset of values a for the argument in $\tau(\cdot)$, these values form a geometric progression. Let

$$T_0(a) = \begin{cases} -\infty & \text{if } a = 0\,, \\ 0 & \text{if } 0 < a \leq 1\,, \\ \infty & \text{otherwise,} \end{cases}$$

and let

$$Q := 1 + \frac{\epsilon}{n+1}, \qquad s := \lceil \log_Q 2^n \rceil = O(n^2/\epsilon).$$

The functions $T_i(\cdot)$ are defined via the recurrence (5) that the function τ satisfies. Namely, T is defined by the following recurrence:

$$T_i(j) = \min_{\alpha \in [0,1]} \max \begin{cases} T_{i-1}(j + \log_Q \alpha), \\ T_{i-1}(j + \log_Q (1-\alpha)) + w_i\,. \end{cases} \tag{6}$$

The FPTAS computes all $T_i(\cdot)$ exhaustively and returns $Q^{j'+1}$, where $j' := \max\{j \mid T_n(j)) \leq C\}$, see [2] for the analysis of the FPTAS.

3.2 The integer knapsack

In this section we present our DP algorithm. Instead of deciding at once how many copies of item i to put in the knapsack, we split the decision into $\lfloor \log u_i \rfloor + 1$ binary sub-decisions. If the values of the various u_i are all powers of 2 minus one, then the binary sub-decisions $j = 1,\dots,\lfloor \log u_i \rfloor + 1$ for item i are equivalent to deciding whether to put in the knapsack "bundles" of 2^{j-1} copies of item i. E.g., for $u_1 = 7$ we split the decision concerning item 1 into the 3 binary sub-descisions of whether to put in the knapsack 1,2,4 copies of item 1. In this case there is a simple DP formulation which is equivalent to a 0/1 knapsack problem with exactly $\sum_{i=1}^{n} \log(u_i + 1)$ items. In the next subsection we deal with the more complicated case where not necessarily all the values of the u_i are powers of 2 minus one.

3.2.1 The dual DP formulation

In what follows we show that even if the values of the u_i are not all powers of 2 minus one we can still give a recurrence using, what we call, the idea of *binding constraints*. For $\ell \geq 1$ let $\tau_{i,\ell,0}(a)$ be the minimal knapsack capacity needed so that there are at least a solutions that use a subset of the items $\{1,\dots,i\}$, put no more than $2^\ell - 1$ copies of item i, and no more than u_k copies of item k, for $k = 1,\dots,i-1$. For $\ell \geq 1$ let $\tau_{i,\ell,1}(a)$ be the minimal knapsack capacity needed so that there are at least a solutions that use a subset of the items $\{1,\dots,i\}$, put no more than $u_i \mod 2^\ell$ copies of item i, and no more than u_k copies of item k, for $k = 1,\dots,i-1$. In this way, considering the third index of $\tau_{i,\ell,r}(a)$, if $r = 0$ then the constraint $x \leq u_i$ is assumed to be *non binding*. If, on the other hand, $r = 1$ then the constraint $x \le u_i$ may be *binding*. Our recurrences are as follows (for simplicity we set $u_0 = 1$. Recall that the definition of $\mathrm{msb}(\cdot)$ is given in Section 2.2):

$$\tau_{i,\ell,0}(a) = \min_{\alpha\in[0,1]} \max \begin{cases} \tau_{i,\ell-1,0}(\alpha a), \\ \tau_{i,\ell-1,0}((1-\alpha)a) + 2^{\ell-1} w_i \end{cases} \qquad \ell = 2,\dots,\lfloor \log u_i \rfloor + 1 \tag{7a}$$

$$\tau_{i,\ell,1}(a) = \min_{\alpha\in[0,1]} \max \begin{cases} \tau_{i,\ell-1,0}(\alpha a), \\ \tau_{i,\mathrm{msb}(u_i,\ell-1),1}((1-\alpha)a) + 2^{\ell-1} w_i \end{cases} \qquad \ell = 2,\dots,\lfloor \log u_i \rfloor + 1 \tag{7b}$$

$$\tau_{i,1,r}(a) = \min_{\alpha\in[0,1]} \max \begin{cases} \tau_{i-1,\lfloor \log u_{i-1}\rfloor+1,1}(\alpha a), \\ \tau_{i-1,\lfloor \log u_{i-1}\rfloor+1,1}((1-\alpha)a) + w_i \end{cases} \qquad r = 0,1 \tag{7c}$$

$$\tau_{i,-\infty,1}(a) = \tau_{i-1,\lfloor \log u_{i-1}\rfloor+1,1}(a) \tag{7d}$$

$$\tau_{0,1,1}(a) = \begin{cases} -\infty & \text{if } a = 0\,, \\ 0 & \text{if } 0 < a \le 1\,, \\ \infty & \text{otherwise.} \end{cases} \tag{7e}$$

where $i = 1,\dots,n$. The number of knapsack solutions satisfies

$$Z = \max\{a \mid \tau_{n,\lfloor \log^+ u_n\rfloor+1,1}(a) \le C\}.$$

We now explain the five equations in formulation (7) in more detail. Equation (7a) deals with the case where the constraint $x \le u_i$ is non binding, so placing in the knapsack $2^\ell - 1$ more copies of item i is a feasible possibility. Clearly, in the following steps the constraint $x \le u_i$ remains non binding. As for equation (7b), it deals with the case where the constraint $x \le u_i$ may be binding when putting $2^{\ell-1}$ copies of item i in the knapsack. If we do put this number of copies, the constraint may be binding and at most $u_i \mod 2^{\ell-1}$ more copies can be placed in the knapsack. Otherwise it is assured to be non binding. Equation (7c) deals with the possibility of placing in the knapsack an odd number of copies of item i. As for equation (7d), note that it is called by equation (7b) when *exactly* u_i copies of item i are put in the knapsack. Equation (7e) is a boundary condition similar to (4).

We now define an approximation $T_{i,\ell,r}$ of $\tau_{i,\ell,r}$ similarly to the 0/1-knapsack case, but where

$$Q := 1 + \frac{\epsilon}{(n+1)\log U}, \qquad s := \lceil \log_Q(U^n) \rceil = O(\frac{n^2 \log^2 U}{\epsilon}).$$

The function $T_{i,\ell,r}$ is defined using the recurrence (7). E.g., using (7a) we define:

$$T_{i,\ell,0}(j) = \min_{\alpha\in[0,1]} \max \begin{cases} T_{i,\ell-1,0}(j + \log_Q \alpha), \\ T_{i,\ell-1,0}(j + \log_Q(1-\alpha)) + 2^{\ell-1} w_i\,. \end{cases} \tag{8}$$

3.2.2 Algorithm statement

Similarly to the algorithm given in [2], also our algorithm computes all $T_{i,\ell,r}(\cdot)$ exhaustively and returns $Q^{j'+1}$, where $j' := \max\{j \mid T_{n,\lfloor \log u_n v\rfloor,1}(j)) \le C\}$. For a given instance (w, C, u) of the integer knapsack problem and a tolerance parameter $\epsilon \in (0,1]$, our approximation algorithm is stated as Algorithm 3

Algorithm 3 FPTAS for counting integer knapsack via dual DP formulation.

Function CountIntegerKnapsackDual(w, C, u, ϵ)
$Q \leftarrow 1 + \frac{\epsilon}{(n+1)\log U}$, $s \leftarrow \lceil \log_Q(U^n) \rceil$, $T_{0,1,1}(0) \leftarrow 0$, $T_{0,1,1}(j) \leftarrow \infty$ for $j > 0$
for $i := 1$ **to** n **do**
  By convention, $T_{i,\ell,r}(k) \leftarrow 0$ for $\ell \geq 1, r = 0, 1$ and $k < 0$
  Calculate $T_{i,-\infty,1}(\cdot)$ via the analogue of equation (7d)
  for $r = 0, 1$ **do** Calculate $T_{i,1,r}(\cdot)$ via the analogue of equation (7c)
  for $\ell := 2$ **to** $\lfloor \log u_i \rfloor + 1$ **do**
    Calculate $T_{i,\ell,0}(\cdot)$ via the analogue of equation (7a), i.e., via equation (8)
    Calculate $T_{i,\ell,1}(\cdot)$ via the analogue of equation (7b)
  end for
end for
$j' \leftarrow \max\{j \mid T_{n,\lfloor \log u_n v \rfloor,1}(j) \leq C\}$
return $Q^{j'+1}$

We now outline an analysis of the running time of Algorithm 3. Since the arguments of function $T_{i,\ell,r}$ in (8) and alike are step functions of α, it suffices to consider a discrete set of α which yields all possible values of the arguments. Such set is of cardinality $O(s)$. Because the various $T_{i,\ell,r}$ are nondecreasing functions of α, the minima in (8) and alike can be computed in time $O(\log s)$ via binary search. Note that there are $O(ns)$ entries of the various functions. As explained in the analysis in [2], the algorithm can be implemented in $O(ns \log s)$ time. Since we have $s = O(\frac{n^2 \log^2 U}{\epsilon})$, the algorithm can be implemented in $O(\frac{n^3 \log^2 U}{\epsilon} \log \frac{n \log U}{\epsilon})$ time, as indicated in Theorem 1.1. We note that the running time of the algorithm differs from the one of [2] because in the latter case we have a different value of s, i.e., $s = O(\frac{n^2}{\epsilon})$.

4 Concluding remarks

In this paper we present two deterministic FPTASs for counting integer knapsack solutions, each of which improves upon the best known results. Both FPTASs relay on clever DP formulations and the new DP technique of binding constraints. The only strongly polynomial approximation scheme for this problem is a (randomized) FPRAS [3]. It is an open problem to design an FPTAS that is both strongly-polynomial and deterministic. It is also an open problem to design an FPTAS for the multidimensional knapsack problem that is more efficient than the one of [6].

References

[1] Jerrum, M., Sinclair, A.: The Markov chain Monte Carlo method: An approach to approximate counting and integration. In Hochbaum, D., ed.: Approximation Algorithms for NP-hard Problems. PWS Publishing Company, Boston (1996) 482–520

[2] Štefankovič, D., Vempala, S., Vigoda, E.: A deterministic polynomial-time approximation scheme for counting knapsack solutions. SIAM Journal on Computing **41** (2012) 356–366

[3] Dyer, M.E.: Approximate counting by dynamic programming. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), June 9-11, 2003, San Diego, CA, USA. (2003) 693–699

[4] Megiddo, N.: On the complexity of linear programming. In: Advances in economic theory, Cambridge, UK, Econom. Soc. Monogr. 12 (1989) 225–268

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

[6] Gopalan, P., Klivans, A., Meka, R.: Polynomial-time approximation schemes for Knapsack and related counting problems using branching programs. CoRR **abs/1008.3187** (2010)

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

[8] Rizzi, R., Tomescu, A.: Faster FPTASes for counting and random generation of knapsack solutions. In: Proceedings of the 22nd Annual European Symposium on Algorithms (ESA). (2014) 762–773

[9] Halman, N., Klabjan, D., Mostagir, M., Orlin, J., Simchi-Levi, D.: A fully polynomial time approximation scheme for single-item stochastic inventory control with discrete demand. Mathematics of Operations Research **34** (2009) 674–685

[10] Halman, N., Klabjan, D., Li, C.L., Orlin, J., Simchi-Levi, D.: Fully polynomial time approximation schemes for stochastic dynamic programs. SIAM Journal on Discrete Mathematics **28** (2014) 1725–1796