

An FPTAS for Stochastic Unbounded Min-Knapsack Problem

Zhihao Jiang^[0000-0002-9682-2476] and Haoyu Zhao^[0000-0001-6775-1421]

Institute for Interdisciplinary Information Sciences, Tsinghua University, China
{jzh16,zhaohy16}@mails.tsinghua.edu.cn

Abstract. In this paper, we study the stochastic unbounded min-knapsack problem (**Min-SUKP**). The ordinary unbounded min-knapsack problem states that: There are n types of items, and there is an infinite number of items of each type. The items of the same type have the same cost and weight. We want to choose a set of items such that the total weight is at least W and the total cost is minimized. The **Min-SUKP** generalizes the ordinary unbounded min-knapsack problem to the stochastic setting, where the weight of each item is a random variable following a known distribution and the items of the same type follow the same weight distribution. In **Min-SUKP**, different types of items may have different cost and weight distributions. In this paper, we provide an FPTAS for **Min-SUKP**, i.e., the approximate value our algorithm computes is at most $(1 + \epsilon)$ times the optimum, and our algorithm runs in $\text{poly}(1/\epsilon, n, \log W)$ time.

Keywords: Stochastic Knapsack, Renewal Decision Problem, Approximation Algorithms

1 Introduction

In this paper, we study the stochastic unbounded min-knapsack problem (**Min-SUKP**). The problem is motivated by the following renewal decision problems introduced in [7]. A system (e.g., a motor vehicle) must operate for t units of time. A particular component (e.g., a battery) is essential for its operation and must be replaced each time it fails. There are n different types of replacement components, and every kind of items has infinite supplies. A type i replacement costs C_i and has a random lifetime with distribution depending on i . The problem is to assign the initial component and subsequent replacements from among the types to minimize the total expected cost of providing an operative component for the t units of time. Formally, we would like to solve the following **Min-SUKP** problem, defined as follows:

Problem 1 (stochastic unbounded min-knapsack). There are n types of items a_1, a_2, \dots, a_n . For an item of type a_i , the cost is a deterministic value c_i , and the weight is random value X_i which follows a known distribution D_i with non-negative integer support. Let $D_i(j)$ denote $\Pr\{X_i \leq j\}$. Each type has infinite

supplies, and the weight of each item is independent of the weight of the items of other types and other items of the same type. Besides, there is a knapsack with capacity W . Our objective is to insert items into the knapsack one by one until the total weight of items in the knapsack is at least W . The realized weight of an item is revealed to us as soon as it is inserted into the knapsack. What is the expected cost of the strategy that minimizes the expected total cost of the items we insert?

Remark 1. The above problem is the stochastic version of the ordinary unbounded min-knapsack problem. Comparing to the ordinary knapsack problem, there is an infinite number of items of each type, and the objective is to minimize the total cost (rather than maximize the total profit).

Remark 2. It can be shown that **Min-SUKP** is NP-hard. In [9], the authors mentioned that the unbounded knapsack problem (**UKP**) is NP-hard, and it can be easily shown that the unbounded min-knapsack is NP-hard, since there is a polynomial reduction between these 2 problems. The problem **Min-SUKP** is NP-hard since it is a generalization of unbounded min-knapsack.

Derman et al.[7] discussed **Min-SUKP** when the weight distributions of items are exponential and provided an exact algorithm to compute the optimal policy. Assaf [1] discussed **Min-SUKP** when the weight distributions of items have a common matrix phase type representation.

In this paper, we present a fully polynomial time approximation scheme (FPTAS) for this problem for general discrete distributions.

Roughly speaking, we borrow the idea of the FPTAS for the knapsack problem and the method for computing the distribution of the sum of random variables [16]. However, there are a few technical difficulties we need to handle. The outline of our algorithm is as follows. We first compute a constant factor approximation for the optimal cost (Section 2), and then we apply the discretization and a dynamic program based on the approximation value (Section 3). However, the dynamic program can only solve the problem in a restricted case where the cost for any item is ‘not too small’ (the cost of each item is larger than a specific value). To solve the whole problem, we consider a reduction from the general setting to the restricted setting and show that the error of the reduction is negligible (Section 4).

1.1 Related Work

The knapsack problem is a classical problem in combinatorial optimization. The classical knapsack problem (max-knapsack problem) is the following problem: Given a set of items with sizes and costs, and a knapsack with a capacity, our goal is to select some items and maximize the total cost of selected items with the constraint that the total size of selected items does not exceed the capacity of the knapsack.

The min-knapsack problem (**Min-KP**) [5] is a natural variant of the ordinary knapsack problem. In the min-knapsack problem, the goal is to minimize

the total cost of the selected items such that the total size of the selected items is not less than the capacity of the knapsack. Although the min-knapsack problem is similar to the max-knapsack problem, a polynomial-time approximation scheme (PTAS) for the max-knapsack problem does not directly lead to a PTAS for the min-knapsack problem. For the (deterministic) min-knapsack problem, approximation algorithms with constant factors are given in [5,10,4]. Han and Makino [12] considered an online version of min-knapsack, that is, the items are given one-by-one over time.

There is also a line of work focusing on the FPTAS for unbounded knapsack problem (**UKP**). **UKP** is similar to the original 0-1 knapsack problem, except that there are infinite number of items of each type. The first FPTAS for **UKP** is introduced by [13], and they show an FPTAS by extending their FPTAS for 0-1 knapsack problem. Their algorithm runs in $O(n + \frac{1}{\epsilon^4} \log \frac{1}{\epsilon})$ time and needs $O(n + \frac{1}{\epsilon^3})$ space. Later, [15] showed an FPTAS with time complexity $O(n \log n + \frac{1}{\epsilon^2} (n + \log \frac{1}{\epsilon}))$ and space complexity $O(n + \frac{1}{\epsilon^2})$. In 2018, [14] presented an FPTAS that runs in $O(n \frac{1}{\epsilon^2} \log^3 \frac{1}{\epsilon})$ time and requires $O(n + \frac{1}{\epsilon} \log^2 \frac{1}{\epsilon})$ space.

However, in some applications, precisely knowing the size of each item is not realistic. In many real applications, we can only get the size distribution of a type of item. This problem leads to the stochastic knapsack problem (**SKP** [19]), which is a generalization of **KP**. In **SKP**, the cost of each item is deterministic, but the sizes of items are random variables with known distributions, and we get the realized size of an item as soon as it is inserted into the knapsack. The goal is to compute a solution policy which indicates the item we insert into the knapsack at a given remaining capacity. For the stochastic max-knapsack problem, an approximation with a constant factor was provided in the seminal work [6]. The current best approximation ratio for **SKP** is 2 [3,18]. An $(1 + \epsilon)$ approximation with relaxed capacity (bi-criterion PTAS) is given in [2,17]. Besides, Deshpande et al.[8] gave a constant-factor approximation algorithm for the stochastic min-knapsack.

Gupta et al.[11] considered a generalization of **SKP**, where the cost of items may be correlated, and we can cancel an item during its execution in the policy. Cancelling an item means we can set a bounding size each time we want to insert an item, we cancel the item if the realized size of the item is larger than the bounding size. When we cancel an item, the size of the item is equal to the bounding size, and the cost of the item is zero. This generalization is referred to as Stochastic Knapsack with Correlated Rewards and Cancellations (**SK-CC**). Gupta et al.[11] gave a constant-factor approximation for **SK-CC** based on LP relaxation. A bicriterion PTAS for **SK-CC** is provided in [17].

1.2 Preliminary

Proposition 1. *Without the loss of generality, we can assume that the support of D_i , which is the weight distribution of an item of type i , has positive integer support.*

We skip the proof of Proposition 1. Please see the proof in Appendix B.

From now on, we can suppose that each type of item has weight distribution with positive integer support.

In **Min-SUKP**, the optimal item added can be determined by the remaining capacity. Let OPT_w denote the expected cost of the optimal strategy when the remaining size is w . We can assume that the support of D_i is $\{0, 1, \dots, W\}$. Let $OPT_0 = OPT_{-1} = \dots = OPT_{-W+1} = 0$. Define $d_i(j) = D_i(j) - D_i(j-1) = \Pr\{X_i = j\}$. From the dynamic program, we have pseudo-polynomial time Algorithm 1 that can compute the exact optimal value.

Algorithm 1 Pseudo-polynomial Time Algorithm

```

1:  $OPT_i \leftarrow 0$  for  $-W + 1 \leq i \leq 0$ 
2: for  $i = 1 \rightarrow W$  do
3:    $OPT_i = \min_{j=1}^n \left( c_j + \sum_{k=1}^W d_j(k) \cdot OPT_{i-k} \right)$ 
return  $OPT_W$ 

```

Algorithm 1 runs in $\mathbf{poly}(n, W)$ time.

In this paper, we show an FPTAS to compute OPT_W . Our algorithm runs in $\mathbf{poly}(\frac{1}{\epsilon}, n, \log W)$ time and return OPT'_W , which is an approximation for OPT_W , such that $(1 - \epsilon)OPT_W \leq OPT'_W \leq (1 + \epsilon)OPT_W$. We assume that there is an oracle \mathcal{A} such that we can call \mathcal{A} to get $D_i(j) = \Pr\{X_i \leq j\}$. Since we require that our algorithm runs in $\mathbf{poly}(\frac{1}{\epsilon}, n, \log W)$ time, our algorithm can call the oracle for at most $\mathbf{poly}(\frac{1}{\epsilon}, n, \log W)$ times.

2 A Constant Factor Estimation

In this section, we show that there is a constant factor approximation for the optimal value. This constant factor approximation serves to estimate the optimal value roughly, and our FPTAS uses the standard discretization technique based on this rough estimation.

Define $b_i = \frac{c_i}{E[X_i]}$. When we insert an item of type i , the expected weight is $E[X_i]$, and the cost is $c_i = b_i E[X_i]$. Suppose $m = \arg \min_i b_i$, and we will show that $2b_m W$ is a constant approximation for the optimal value OPT_W . Formally, we have the following lemma,

Lemma 1. *For all $-W + 1 \leq w \leq W$, $b_m w \leq OPT_w \leq b_m(w + W)$, where $m = \arg \min_i b_i$.*

This lemma can be proved by induction, and please see Appendix C for its formal proof.

Specifically, when $w = W$, we get $b_m W \leq OPT_W \leq 2b_m W$ directly from the above lemma. However, when computing b_m , we need to enumerate the support. To avoid expensive enumeration, we can compute $E[X_i]$ approximatively. We

round the realized weight x_i into $2^{\lceil \log_2 x_i \rceil}$. Just let

$$\overline{E}[X_i] = \sum_{j=1}^W d_i(j) 2^{\lceil \log_2 j \rceil} = D_i(1) + \sum_{j=0}^{\lceil \log W \rceil} (2^j \cdot (D_i(2^{j+1}) - D_i(2^j))).$$

We have $\frac{E[X_i]}{2} \leq \overline{E}[X_i] \leq E[X_i]$, since $\frac{x_i}{2} \leq 2^{\lceil \log_2 x_i \rceil} \leq x_i$.

Let $\overline{OPT}_W = 2W \cdot \min_i \frac{c_i}{E[X_i]}$. From the previous argument, we have $2b_m W \leq \overline{OPT}_W \leq 4b_m W$, which means $OPT_W \leq \overline{OPT}_W \leq 4OPT_W$.

Let $T = \frac{1}{4}\overline{OPT}_W$. We have $\frac{1}{4}OPT_W \leq T \leq OPT_W$. T is the estimation of OPT_W .

3 FPTAS Under Certain Assumption

In this section, we discuss **Min-SUKP** under the following assumption.

Definition 1 (Cheap/Expensive type).

Let $\theta = \frac{\epsilon}{10n}$. We call type i is an expensive type if $c_i \geq \theta T$, otherwise we call type i is a cheap type.

Assumption 1 we assume all the types are expensive.

And we give an algorithm with approximation error at most ϵT in this section under Assumption 1.

In general, our algorithm for **Min-SUKP** is inspired from the FPTAS of the ordinary knapsack problem [20]. We define $\hat{f}_c = \max\{w | OPT_w \leq c\}$, and compute the approximation for \hat{f} . However, the support of \hat{f} is the set of real numbers. So we discretize \hat{f} and only compute the approximation for $\hat{f}_{i\delta T}$ for all $i \leq \lceil \frac{1}{\delta} \rceil + 1$, where i is non-negative integer and $\delta = \frac{\epsilon^2}{100n}$. In our algorithm, we use dynamic programming to compute f_i , which is the approximation for $\hat{f}_{i\delta T}$. Then we use f_i to get an approximate value of OPT_W . Since $\hat{f}_{i\delta T}$ is monotonically increasing with respect to i , we can find the smallest i such that $f_i \geq W$ and return the value $i\delta T$ as the approximate value of OPT_W .

Now we show how to compute f_i . First, suppose that $\hat{f}_{i\delta T} = w^*$, and from the dynamic programming, we have

$$OPT_{w^*} = \min_k \left\{ c_k + \sum_{j=1}^W d_k(w^* - j) OPT_j \right\}.$$

Since OPT_w is non-decreasing while w is increasing, recall $\hat{f}_c = \max\{w | OPT_w \leq c\}$, and we get,

$$\begin{aligned} w^* &= \max \left\{ w' \mid \exists k, c_k + \sum_{j=1}^W d_k(w' - j) OPT_j \leq i\delta T \right\} \\ &= \max \left\{ w' \mid \exists k, c_k + \sum_{j=1}^{w'-1} d_k(w' - j) OPT_j \leq i\delta T \right\}. \end{aligned}$$

Algorithm 2 The Dynamic Program for Computing approximate answer for **Min-SUKP** under Assumption 1

```

1: Let  $\delta = \frac{\epsilon^2}{100n}$ .
2:  $f_0 \leftarrow 0$ 
3: for  $i = 1 \rightarrow \lceil \frac{1}{\delta} \rceil + 1$  do
4:   Compute  $f_i$  using binary search according to Algorithm 3
5:   if  $f_i \geq W$  then
6:     return  $\hat{V} := i\delta T$ 

```

Algorithm 3 Given w , judge whether $f_i \geq w$ (whether $g_w \leq i\delta T$)

```

1: for  $j = 1 \rightarrow n$  do
2:   for  $m = 0 \rightarrow i - 2$  do
3:      $P_m = Pr[X_j \in [w - f_{m+1}, w - f_m]]$  ▷ by Binary search from oracle
4:      $P_{i-1} = Pr[X_j \in [1, w - f_{i-1}]]$ 
5:      $g_w \leftarrow c_j + \sum_{m=0}^{i-1} P_m(m+1)\delta T$  ▷ Equation (1)
6:     if  $g_w \leq i\delta T$  then return true
       return false

```

Define $\hat{g}_w := j\delta T$ for all $\hat{f}_{j-1} < w \leq \hat{f}_j$. Then \hat{g}_w is the rounding up discretization value of OPT_w , and we can approximately compute w^* (let \hat{w} denote the approximate value) by

$$\hat{w} = \max \left\{ w' \mid \exists k, \left(c_k + \sum_{j=1}^{w'-1} d_k(w' - j)\hat{g}_j \right) \leq i\delta T \right\}.$$

However, we do not have \hat{g} during the computation. Instead, we use the following quantity to approximate \hat{g} . Given f_0, \dots, f_{i-1} , define $g_w := j\delta T$ for all $f_{j-1} < w \leq f_j$ where $j \leq i - 1$, and define $g_w = i\delta T$ for all $w > f_{i-1}$. Then we have

$$f_i = \max \left\{ w' \mid \exists k, \left(c_k + \sum_{j=1}^{w'-1} d_k(w' - j)g_j \right) \leq i\delta T \right\}. \quad (1)$$

Remark 3. When we compute f_i , we have already gotten f_0, f_1, \dots, f_{i-1} .

To compute f_i , we use binary search to guess $f_i = w'$ and accept the largest w' that satisfies the constraint in (1).

The pseudo-code of our algorithm is shown in Algorithm 2. The detailed version of the pseudo-codes is presented in Appendix A.

In details, we enumerate i and compute f_i until f_i reaches the weight lower limit W . To compute f_i , we use binary search starting with $L = 0, R = W$. In each step of binary search, let $w = (L + R)/2$ and compute g_w , and decide to recur in which half according to the relation between g_w and $i\delta T$, until $L = R$ which means $f_i = L = R$.

To quantify the approximation error by algorithm 2, we have the following theorem.

Theorem 1. *The output \hat{V} of Algorithm 2 satisfies $(1-\delta)(1-\frac{\epsilon}{10})\hat{V} \leq OPT_W \leq \hat{V}$.*

Generally speaking, this results can be shown in 2 steps: First, we will show that the real optimal value is upper bounded by the value computed in our algorithm, and next, we will show that under Assumption 1, the difference between the value computed in Algorithm 2 and the real optimal value is upper bounded by a small value. Given these two results, we can prove Theorem 1. Please see Appendix D for the formal proof of Theorem 1.

From the above theorem, we know that the output \hat{V} of Algorithm 2 is a $(1 + \epsilon)$ -approximation for OPT_W .

4 FPTAS in the General Case

In the previous section, we show that there is an FPTAS of **Min-SUKP** under Assumption 1 (when all the types are expensive). In this section, we remove Assumption 1 and show that there is an FPTAS of **Min-SUKP**. We will first present the general idea of our algorithm.

Our Ideas: If we use the algorithm in the last section to compute in general case, the error will not be bounded. The key reason is that we may insert lots of items of cheap types. One idea is, we can bundle lots of items in the same cheap type p into bigger items (an induced type p'), such that p' is expensive. Then we replace type p by the new type p' . Now, we can use the algorithm in the last section. However, we can only use bundled items even if we only want to use one item of a certain cheap item. Luckily, using some extra items of cheap items does not weaken the policy very much.

The remaining problem is, how to compute the distribution of many items of type p ? For example, we always use $e_p = 2^k$ items of type p each time. We discretize the weight distribution X_p , and use doubling trick to compute the approximate distributions for $X_{p,1}, \sum_{i=1}^2 X_{p,i}, \sum_{i=1}^4 X_{p,i}, \dots$ one by one, where $X_{p,i}$ are independent to each other and follow the same distribution of X_p . We can show that, using the approximation distributions in the computation will not lead to much error.

4.1 Adding Limitations to Strategy

For type p , if $c_p < \theta T$ ($\theta = \frac{\epsilon}{10n}$ as defined in the previous section), then there exists $e_p = 2^{k_p}, k_p \in \mathbb{Z}$ such that $e_p c_p \in [\theta T, 2\theta T]$. For convenience, if $c_p \geq \theta T$, we denote $e_p = 1$. We have the following restriction to the strategy.

Definition 2 (Restricted strategy). *A strategy is called restricted strategy, if for all type p , the total number of items of type p we insert is always a multiple of e_p .*

If we know that for all type p , the total number of items of type p is always a multiple of e_p , we hope that each time we use an item of type p , we will use e_p of them together. This leads to the following definition.

Definition 3 (Block strategy). *A strategy is called block strategy, if we always insert a multiple of e_p number of items of type p together.*

The following theorem shows that, adding limitation to the strategy will not affect the optimal value too much.

Theorem 2. *Suppose the expected cost of the best block strategy is $OPT_W^{(b)}$, then $OPT_W \leq OPT_W^{(b)} \leq OPT_W + \frac{\epsilon T}{5}$.*

Because of the space limitation, we will present the proof sketch below. For the formal proof of Theorem 2, please see Appendix E.

Proof (Proof sketch). The proof Theorem 2 is divided into 2 parts. The first part shows that the optimal value for the original problem does not differ much from the optimal value with restricted strategy (see Definition 2), and the second part shows that the optimal value with restricted strategy is the same as the optimal value with block strategy (see Definition 3). The first part is simple since we can add some item after following the optimal strategy in the original problem. The second part follows from the intuition that if we must use an item in the future, it is good to use it right now.

4.2 Computing the Summation Distribution of Many Items of the Same Type

In the last part, we define block strategy by adding a constraint to the ordinary strategy. And we find the expected cost of the optimal block strategy is close to that of the optimal strategy.

The block strategies conform to Assumption 1 in Section 3. If we know the distribution of the total weight of e_p items of type p , we can compute the approximate optimal expected cost by Algorithm 2. In this part, we give an algorithm which approximately computes the distribution of the total weight of e_p items of type p .

Due to the space limitation, we present our algorithm in this section, and we put the analysis of our algorithm into the appendix (see Appendix F). To present our idea, we need the following definitions.

Definition 4 (Distribution Array). *For a random variable X with positive integer support, we use $X[i]$ to denote the probability that $X \geq i$, i.e. $X[i] = \Pr\{X \geq i\}$, and we use an array $\text{Dist}(X) := (X[1], X[2], \dots, X[W])$ to denote the distribution. We call $\text{Dist}(X)$ the distribution array of variable X .*

Remark 4. From the definition, we know that $\text{Dist}(X)$ is a non-increasing array. Besides, in the definition, $\text{Dist}(X)$ has only W elements since we only care $X[i]$ when $i \leq W$.

Definition 5. *For any non-increasing array $D = (D_1, D_2, \dots, D_W)$ of length W , if $D_1 \leq 1$ and $D_W \geq 0$, there is a random variable X such that $\text{Dist}(X) = D$. We say that X is the variable corresponding to distribution array D , denoted by $\text{Var}(D) := X$.*

Suppose $\{Y_i\}_{i \geq 1}$ are identical independent random variables with distribution array $\text{Dist}(X_p)$. Let S_i denote $\sum_{j=1}^i Y_j$ and $\text{Dist}(S_i)$ denote the corresponding distribution array. We want to compute the distribution array of S_{e_p} and we have the following equations,

$$\Pr\{S_{2i} = w\} = \sum_{j=1}^{w-1} (\Pr\{S_i = j\} \cdot \Pr\{S_i = w - j\}), \forall 1 \leq w \leq W, \quad (2)$$

$$S_{2i}[w] = \Pr\{S_i \geq w\} + \sum_{j=1}^{w-1} (\Pr\{S_i = j\} \cdot \Pr\{S_i \geq w - j\}) \quad (3)$$

$$= S_i[w] + \sum_{j=1}^{w-1} ((S_i[j] - S_i[j+1]) \cdot S_i[w - j]). \quad (4)$$

Note that S_{2i} can be computed from S_i , so we only need to compute $S_1, S_2, S_4, \dots, S_{e_p}$ successively (recall that $e_p = 2_p^k$ where $k_p \in \mathbb{Z}$). Note that S_1 could be got from the oracle.

However, computing the exact distribution of S_{2j} is slow (needs at least $\text{poly}(W)$ time), so we compute an approximate value of S_i . To introduce our method which approximately computes the distribution, we need the following definitions.

Definition 6 (η -Approximate Array). Given a positive real number η , for distribution array $A = (a_1, a_2, \dots, a_m)$, define $A' = (a'_1, a'_2, \dots, a'_m)$ as the η -approximate array of A , where for all $i \in [m]$,

$$a'_i = (1 + \eta)^{\lceil \log_{1+\eta} a_i \rceil}, a_i > (1 + \eta)^{-\zeta}.$$

Definition 7 ((ζ, η) -Approximate Array). Given positive real numbers ζ, η , for distribution array $A = (a_1, a_2, \dots, a_m)$, define $A' = (a'_1, a'_2, \dots, a'_m)$ as the (ζ, η) -approximate array of A , where for all $i \in [m]$,

$$a'_i = \begin{cases} (1 + \eta)^{\lceil \log_{1+\eta} a_i \rceil}, & a_i > (1 + \eta)^{-\zeta} \\ (1 + \eta)^{-\zeta}, & a_i \leq (1 + \eta)^{-\zeta} \end{cases}.$$

Definition 8 ((ζ, η) -Approximation). For random variable X , suppose distribution array D is (ζ, η) -approximate array of $\text{Dist}(X)$. Define $\text{Var}(D)$ as the (ζ, η) -approximation of X .

Remark 5. The (ζ, η) -Approximation of a random variable is still a random variable. And for any random variable X with integer support in $[1, W]$, the (ζ, η) -approximation of X has at most $\lceil \zeta \rceil$ different possible values.

Let $\eta = \frac{\epsilon}{10 \log W}$ and $\zeta = \log_{1+\eta} \frac{W}{\eta}$, and our algorithm is shown as following: We first compute (ζ, η) -approximation of S_1 which is denoted by B_1 . Then for all $i \in \lceil \log_{e_p} \rceil$, we compute the distribution array of B'_{2^i} , which is the

summation of independent $B_{2^{i-1}}$ and $B_{2^{i-1}}$. Then we compute B_{2^i} which is the (ζ, η) -approximation for B'_{2^i} . Finally, we can get B_{e_p} which is an approximate random variable of S_{e_p} .

When we compute the summation of $B_{2^{i-1}}$ and $B_{2^{i-1}}$, as there are at most $O(\zeta)$ different values in $\text{Dist}(B_{2^{i-1}})$, there are at most $O(\zeta)$ values w such that $\Pr\{B_{2^{i-1}} = w\} > 0$. Based on the previous argument, we can enumerate w_1 and w_2 such that $\Pr\{B_{2^{i-1}} = w_1\} > 0$ and $\Pr\{B_{2^{i-1}} = w_2\} > 0$. In the end, we sort each $\Pr\{B_{2^{i-1}} = w_1\} \cdot \Pr\{B_{2^{i-1}} = w_2\}$ by the value $w_1 + w_2$ and arrange them to get the distribution array $\text{Dist}(B'_{2^i})$. This shows that we can compute the approximate distribution in $O(\zeta^2 \log \zeta)$ time.

Formally, we have Algorithm 4 to compute B_{e_p} .

Algorithm 4 Computing Approximate Distribution of S_{e_p}

- 1: Let $\eta = \frac{\epsilon}{10 \log W}$ and $\zeta = \log_{1+\eta} \frac{W}{\eta}$.
 - 2: Let B_1 be the (ζ, η) -approximation for S_1 . Compute $\text{Dist}(B_1)$ according to the oracle
 - 3: **for** $i = 1 \rightarrow \log_2 e_p$ **do**
 - 4: Let B'_{2^i} be the summation of $B_{2^{i-1}}$ and $B_{2^{i-1}}$. Compute $\text{Dist}(B'_{2^i})$.
 - 5: Compute $\text{Dist}(B_{2^i})$, which is the (ζ, η) -approximation for B'_{2^i} .
 - 6: **return** $\text{Dist}(B_{e_p})$
-

Before we state the main theorem that bounds the approximation error of our algorithm, we combine the full procedure and get our final Algorithm 5 for **Min-SUKP**.

Algorithm 5 Algorithm for **Min-SUKP**

- 1: For each type p , let $S_{e_p}^p$ be the summation of e_p i.i.d. variables with distribution X_p . Compute approximate distribution Y_p of $S_{e_p}^p$ by Algorithm 4.
 - 2: For each item type p , construct new type p' with expected cost $c_p e_p$ and weight distribution Y_p .
 - 3: Let W and all the new types be the input, and get return value \hat{V} of Algorithm 2.
 - 4: **return** \hat{V} .
-

Then, we have our main theorem, which discusses the approximation error of Algorithm 5.

Theorem 3. *The output \hat{V} of Algorithm 5 satisfies*

$$(1 - \epsilon)OPT_W \leq \hat{V} \leq (1 + \epsilon)OPT_W.$$

To prove this theorem, we first show $\text{Dist}(B_{e_p})$ in Algorithm 4 is approximation of $\text{Dist}(A_{e_p})$, by constructing another strategy C_{e_p} which is strictly better than B_{e_p} and the expected cost of C_{e_p} is closed to the expected cost of A_{e_p}

(induction is used). Then we combine all the errors in Algorithm 5 and prove that Algorithm 5 is FPTAS of **Min-SUKP**. For details, please see Appendix F.

4.3 Time Complexity

Our algorithm runs in $\text{poly}(n, \log W, \frac{1}{\epsilon})$. Combined with Theorem 3, Algorithm 5 is an FPTAS for **Min-SUKP**. The theorem for the time complexity of Algorithm 5 is stated as follow,

Theorem 4. *Algorithm 5 runs in polynomial time and thus is an FPTAS for **Min-SUKP**. More specifically, Algorithm 5 has time complexity*

$$O\left(\frac{n \log^6 W}{\epsilon^3} + \frac{n^3 \log W}{\epsilon^4}\right).$$

This theorem can be proved by recalling the parameters we have set, counting for the number of each operation, and expanding the parameters as n , $\log W$ and $\frac{1}{\epsilon}$. Please see Appendix G for the formal proof.

5 Conclusions and Further Work

We obtain the first FPTAS for **Min-SUKP** in this paper. We focus on computing approximately the optimal value, but our algorithms and proofs immediately imply how to construct an approximate strategy in polynomial time.

There are some other directions related to **Min-SUKP** which are still open. It would be interesting to design a PTAS (or FPTAS) for the 0/1 stochastic minimization knapsack problem, the 0/1 stochastic (maximization) knapsack problem and the stochastic unbounded (maximization) knapsack problem. Hopefully, our techniques can be helpful in solving these problem.

Acknowledgement

The authors would like to thank Jian Li for several useful discussions and the help with polishing the paper. The research is supported in part by the National Basic Research Program of China Grant 2015CB358700, the National Natural Science Foundation of China Grant 61822203, 61772297, 61632016, 61761146003, and a grant from Microsoft Research Asia

References

1. Assaf, D.: Renewal decisions when category life distributions are of phase-type. *Mathematics of Operations Research* **7**(4), 557–567 (1982)
2. Bhalgat, A., Goel, A., Khanna, S., SIAM, ACM: Improved Approximation Results for Stochastic Knapsack Problems. *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia (2011)

3. Bhalgat, A.: A $(2 + \epsilon)$ -approximation algorithm for the stochastic knapsack problem. Manuscript (2012)
4. Carnes, T., Shmoys, D.: Primal-dual schema for capacitated covering problems, Lecture Notes in Computer Science, vol. 5035, pp. 288–302. Springer-Verlag Berlin, Berlin (2008)
5. Csirik, J., Frenk, J.B.G., Labbe, M., Zhang, S.: Heuristics for the 0-1 min-knapsack problem. Acta Cybernetica **10**(1-2), 15–20 (1991)
6. Dean, B.C., Goemans, M.X., Vondrak, J., iee computer, s.: Approximating the stochastic knapsack problem: The benefit of adaptivity, pp. 208–217. Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Soc, Los Alamitos (2004)
7. Derman, C., Lieberman, G.J., Ross, S.M.: A renewal decision problem. Management Science **24**(5), 554–561 (1978)
8. Deshpande, A., Hellerstein, L., Kletenik, D.: Approximation algorithms for stochastic submodular set cover with applications to boolean function evaluation and min-knapsack. ACM Transactions on Algorithms **12**(3), 28 (2016)
9. Garey, M.R., Johnson, D.S.: Computers and intractability, vol. 29. wh freeman New York (2002)
10. Guntzer, M.M., Jungnickel, D.: Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem. Operations Research Letters **26**(2), 55–66 (2000)
11. Gupta, A., Krishnaswamy, R., Molinaro, M., Ravi, R.: Approximation algorithms for correlated knapsacks and non-martingale bandits. In: IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011. pp. 827–836 (2011)
12. Han, X., Makino, K.: Online Minimization Knapsack Problem, Lecture Notes in Computer Science, vol. 5893, pp. 182–193. Springer-Verlag Berlin, Berlin (2010)
13. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM (JACM) **22**(4), 463–468 (1975)
14. Jansen, K., Kraft, S.E.: A faster fptas for the unbounded knapsack problem. European Journal of Combinatorics **68**, 148–174 (2018)
15. Kellerer, H., Pferschy, U., Pisinger, D.: Multidimensional knapsack problems. In: Knapsack problems, pp. 235–283. Springer (2004)
16. Li, J., Shi, T.L.: A fully polynomial-time approximation scheme for approximating a sum of random variables. Operations Research Letters **42**(3), 197–202 (2014)
17. Li, J., Yuan, W.: Stochastic combinatorial optimization via poisson approximation. In: Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing. pp. 971–980. STOC '13, ACM, New York, NY, USA (2013)
18. Ma, W.: Improvements and generalizations of stochastic knapsack and multi-armed bandit approximation algorithms. In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (2014)
19. Ross, K.W., Tsang, D.H.: The stochastic knapsack problem. IEEE Transactions on communications **37**(7), 740–747 (1989)
20. Sahni, S.: Approximate algorithms for 0/1 knapsack problem. Journal of the ACM **22**(1), 115–124 (1975)

A Detailed Version of Algorithm 2

In this section, we provide the detailed version of Algorithm 2, which is shown below as Algorithm 6.

Algorithm 6 The Dynamic Program for Computing approximate answer for **Min-SUKP** under Assumption 1: Detailed Version

```

1: Let  $\delta = \frac{\epsilon^2}{100n}$ .
2:  $f_0 \leftarrow 0$ 
3: for  $i = 1 \rightarrow \lceil \frac{1}{\delta} \rceil + 1$  do ▷ The DP for computing  $f_i$ 
4:   Let  $L = 0, R = W$ 
5:   while  $R > L$  do ▷ Binary search to guess  $f_i$ 
6:      $w \leftarrow \lfloor (L + R)/2 \rfloor$ 
7:      $e \leftarrow 0$  ▷  $e$  means whether  $g_w$  is less or equal to  $i\delta T$ 
8:     for  $j = 1 \rightarrow n$  do
9:       for  $m = 0 \rightarrow i - 2$  do ▷ Compute the probability after rounding the
         weight
10:         $P_m = Pr[X_j \in [w - f_{m+1}, w - f_m]]$  ▷ by Binary search from oracle
11:         $P_{i-1} = Pr[X_j \in [1, w - f_{i-1}]]$ 
12:         $g \leftarrow c_j + \sum_{m=0}^{i-1} P_m(m+1)\delta T$  ▷ Equation (1)
13:        if  $g \leq i\delta T$  then
14:           $e \leftarrow 1$ 
15:        if  $e = 1$  then
16:           $L = w$ 
17:        else
18:           $R = w - 1$ 
19:       $f_i \leftarrow L$ 
20:      if  $f_i \geq W$  then
21:        return  $\hat{V} := i\delta T$ 

```

B Proof of Proposition 1

Proof (Proof of Proposition 1).

Firstly recall that in the definition of **Min-SUKP**, D_i has non-negative integer support. If we add an item of type i and the realized weight $X_i = 0$, because there are infinite number of items of each type and the state does not change, from the dynamic program, we should also add the item of type i until the realized weight is not 0. We can construct another type i' with distribution D'_i and cost c'_i to replace type i , where using one item of type i' is equivalent to using several items of type i until the realized weight of one item is positive. Then, D'_i has positive integer support, and formally speaking, we have

$$c'_i = \frac{c_i}{1 - d_i(0)}, d'_i(t) = \frac{d_i(t)}{1 - d_i(0)}, \forall t > 0,$$

where $d'_i(t) = D'_i(t) - D'_i(t-1)$ (recall that $d_i(t) = D_i(t) - D_i(t-1)$).

Then we can get

$$D'_i(t) = \sum_{j=1}^t d_i(j) = \frac{D_i(t) - D_i(0)}{1 - d_i(0)}.$$

□

C Proof of Lemma 1

Proof (Proof of Lemma 1). Prove by induction. First, for all $-W + 1 \leq w \leq 0$, $OPT_w = 0$,

$$b_m w \leq OPT_w \leq b_m(w + W).$$

Suppose for all $w \leq k, k \geq 0$, $b_m w \leq OPT_w \leq b_m(w + W)$. Then we have that

$$\begin{aligned} OPT_{k+1} &= \min_i (c_i + \mathbb{E}_{X_i} [OPT_{k+1-X_i}]) \\ &\leq c_m + \mathbb{E}_{X_m} [OPT_{k+1-X_m}] \\ &= c_m + \sum_{w=1}^W d_m(w) OPT_{k+1-w} \\ &\leq c_m + \sum_{w=1}^W d_m(w) b_m(k+1-w+W) \\ &= c_m + b_m(k+1+W) - b_m \mathbb{E}[X_m] \\ &= b_m(k+1+W). \end{aligned}$$

$$\begin{aligned} OPT_{k+1} &= \min_i (c_i + \mathbb{E}_{X_i} [OPT_{k+1-X_i}]) \\ &= \min_i \left(c_i + \sum_{w=1}^W d_i(w) OPT_{k+1-w} \right) \\ &\geq \min_i \left(c_i + \sum_{w=1}^W d_i(w) b_i(k+1-w) \right) \\ &= \min_i (c_i + b_i(k+1) - b_i \mathbb{E}[X_i]) \\ &= \min_i (b_i(k+1)) \\ &\geq b_m(k+1). \end{aligned}$$

Then we arrange the terms, and we get

$$b_m(k+1) \leq OPT_{k+1} \leq b_m(k+1+W).$$

Above, we complete the proof by induction. □

D Proof of Theorem 1

In this section, we will analyze the approximation error of Algorithm 2 and prove Theorem 1. We will rely on two lemmas to prove Theorem 1. Generally speaking, the first lemma shows that the real optimal value is upper bounded by the value computed in our algorithm, and the second lemma shows that under Assumption 1, the error between the real optimal value is lower bounded by the difference between the value computed in our algorithm and another small value.

Before proving the theorem, let's first recall Assumption 1. Assumption 1 says that, for each type i , $c_i \geq \theta T$, where $\theta = \frac{\epsilon}{10n}$ and c_i is the cost of type i . Then, we will also recall the variables and notations defined previously.

We use OPT_W denote the optimal value of **Min-SUKP** and \overline{OPT}_W denote the estimation of the optimal value in Section 2. We define $T = \frac{1}{4}\overline{OPT}_W$.

Similar to the FPTAS of the ordinary knapsack problem, we define $\hat{f}_c = \max\{w | OPT_w \leq c\}$, and compute an approximation for \hat{f} . However, the support of \hat{f} is the set of real numbers. So we discretize \hat{f} and only compute the approximation for $\hat{f}_{i\delta T}$ for all $i \leq \lceil \frac{1}{\delta} \rceil + 1$, where $\delta = \frac{\epsilon^2}{100n}$. In our algorithm, we use dynamic programming to compute f_i , which is the approximation for $\hat{f}_{i\delta T}$.

We also define $\hat{g}_w := j\delta T$, for all $\hat{f}_{j-1} < w \leq \hat{f}_j$. Then \hat{g}_w is the rounding up discretization value of OPT_w . In the algorithm, we use the following quantity to approximate \hat{f} . Given f_0, \dots, f_{i-1} , define $g_w := j\delta T$ for all $f_{j-1} < w \leq f_j$, $j \leq i-1$, and define $g_w = i\delta T$ for all $w > f_{i-1}$. The ideas behind Algorithm 2 and the process to compute f_i, g_w are shown in Section 3.

Then, we will prove Theorem 1. We first have the following lemmas.

Lemma 2. *For all f_i , we have $f_i \leq \hat{f}_{i\delta T}$, which means $OPT_{f_i} \leq i\delta T$.*

Proof (Proof of Lemma 2).

We prove this by induction. The base case is true, which is just $f_0 = 0$ and $\hat{f}_0 = 0$. Now, assume the statement is true for f_j where $j \leq i-1$. We prove the statement is also true for f_i .

For $0 \leq j < i$, $OPT_{f_j} \leq j\delta T$. So for all $1 \leq j \leq i-1$, for all $f_{j-1} < w \leq f_j$,

$$OPT_w \leq OPT_{f_j} \leq j\delta T.$$

We know $g_w = j\delta T$, so $OPT_w \leq g_w$ for all $0 \leq w \leq f_{i-1}$.

When we compute f_i , we define $g_w = i\delta T$ for all $w > f_{i-1}$, so $OPT_w \leq g_w$ for all $w \leq \hat{f}_{i\delta T}$.

We know

$$f_i = \max \left\{ w \mid \exists k, \left(c_k + \sum_{j=1}^{w-1} d_k(w-j)g_j \right) \leq i\delta T \right\},$$

and

$$\hat{f}_{i\delta T} = \max \left\{ w' \mid \exists k, \left(c_k + \sum_{j=1}^{w'-1} d_k(w'-j)OPT_j \right) \leq i\delta T \right\}.$$

So, $f_i \leq \hat{f}_{i\delta T}$, which implies $OPT_{f_i} \leq i\delta T$. \square

Lemma 3. For all f_i , $OPT_{f_{i+1}} > i\delta T - \frac{i\delta^2}{\theta}T$.

Proof (Proof of Lemma 3).

We prove it by induction. The base case is true which is $f(1) > 0$. Now assume the statement is true for f_j where $j \leq i-1$. We show that the statement is also true for f_i .

For $0 \leq j < i$, $OPT_{f_{j+1}} > j\delta T - \frac{j\delta^2}{\theta}T$. So for all $1 \leq j \leq i$, for all $f_{j-1} < w \leq f_j$,

$$OPT_w \geq OPT_{f_{j-1}+1} > (j-1)\delta T - \frac{(j-1)\delta^2}{\theta}T.$$

We know $g_w = j\delta T$, where $f_{j-1} < w \leq f_j$, so

$$OPT_w \geq \left(1 - \frac{\delta}{\theta}\right)g_w - \delta T, \quad \forall f_{j-1} < w \leq f_j. \quad (5)$$

We know

$$f_i = \max \left\{ w' \mid \exists k, \left(c_k + \sum_{j=1}^{w'-1} d_k(w' - j)g_j \right) \leq i\delta T \right\}.$$

Let $w^* = f_i + 1$. Then for all k ,

$$\left(c_k + \sum_{j=1}^{w^*-1} d_k(w^* - j)g_j \right) > i\delta T.$$

From 5, we know that for all k ,

$$\begin{aligned} & \left(c_k + \sum_{j=1}^{w^*-1} d_k(w^* - j)OPT_j \right) \\ & \geq \left(c_k - \delta T + \left(1 - \frac{\delta}{\theta}\right) \sum_{j=1}^{w^*-1} d_k(w^* - j)g_j \right) \\ & \geq \left(c_k - \frac{\delta}{\theta}c_k + \left(1 - \frac{\delta}{\theta}\right) \sum_{j=1}^{w^*-1} d_k(w^* - j)g_j \right) \\ & \geq \left(1 - \frac{\delta}{\theta}\right) \left(c_k + \sum_{j=1}^{w^*-1} d_k(w^* - j)g_j \right) \\ & > \left(1 - \frac{\delta}{\theta}\right) i\delta T \\ & = i\delta T - \frac{i\delta^2}{\theta}T. \end{aligned}$$

It means

$$OPT_{w^*} = \min_k \left\{ c_k + \sum_{j=1}^W d_k(w^* - j)OPT_j \right\} > i\delta T - \frac{i\delta^2}{\theta}T.$$

Then, we complete the proof by induction. \square

Given Lemma 2 and Lemma 3, we can prove the main theorem (Theorem 1) in Section 3.

Proof (Proof of Theorem 1). First, with our algorithm, we have $\hat{V} = i\delta T$, where $f_{i-1} < W \leq f_i$. Then because OPT_w is increasing with respect to w , we know that

$$OPT_{f_{i-1}+1} \leq OPT_W \leq OPT_{f_i}.$$

Combining Lemma 2 and Lemma 3, we have

$$OPT_{f_i} \leq i\delta T, \quad OPT_{f_{i-1}+1} > (i-1)\delta T - \frac{(i-1)\delta^2}{\theta}T = (i-1)\left(1 - \frac{\epsilon}{10}\right)\delta T,$$

which leads to

$$(i-1)\left(1 - \frac{\epsilon}{10}\right)\delta T \leq OPT_W \leq i\delta T.$$

Recalling our definition of T , we have $T \leq OPT_W$. Then we have $(i-1)\left(1 - \frac{\epsilon}{10}\right) \geq \frac{1}{\delta}$, which implies $i \geq \frac{1}{\delta}$. Then from the fact that $\frac{i-1}{i} = 1 - \frac{1}{i} \geq 1 - \delta$, we have

$$(1 - \delta)\left(1 - \frac{\epsilon}{10}\right)i\delta T \leq OPT_W \leq i\delta T,$$

which completes the proof of Theorem 1. \square

E Proof of Theorem 2

Suppose the expected cost of the best restricted strategy is $OPT_W^{(r)}$. Obviously, we have $OPT_W \leq OPT_W^{(r)}$.

Lemma 4. *We have*

$$OPT_W^{(r)} \leq OPT_W + \frac{\epsilon T}{5}.$$

Proof. In the best strategy with expected cost OPT_W , after the total weight of items in the knapsack reaches W , we can insert some extra items such that the total number of items of type p in the knapsack is a multiple of e_p . By inserting extra items, the original strategy becomes a restricted strategy. And the cost of extra items of one type is at most $2\theta T$. So the total extra cost is at most $2n\theta T = \frac{\epsilon T}{5}$. \square

The following lemma shows, in a strategy, if we must insert a multiple of e_p number of items of type p , it is equivalent to a strategy such that at each time we choose to insert an item of type p , we insert e_p of them together.

Lemma 5. *In the process we insert items, if we must insert at least one more item of type p (recall that the number of items of type p we insert in the end must be multiple of e_p), Inserting one item of type p right now is one best strategy.*

Proof. We first insert one item of type p right now and pretend we didn't add this item. Then we insert items under the guidance of the original strategy until the original strategy tells us to insert the item of type p into the knapsack. At present, we do not add it, and we regard the item of type p we inserted before as the item we should insert at present.

It is clear that the new strategy has the same expected cost as the original strategy. \square

The proof of Theorem 2 follows directly from Lemma 4 and Lemma 5.

F Proof of Theorem 3

In this section, we will analyze the approximation error of Algorithm 5 and prove Theorem 3. To begin with, we need some definitions to illustrate our points.

Definition 9 (Sub-strategy). *In the strategy, we usually adaptively insert a series of items with the same type. We call this process sub-strategy.*

Let $Weight(s)$ denotes the total weight of items inserted in a sub-strategy s , noting that $Weight(s)$ is a random variable. To be convenient, we use $DistW(s)$ to denote $Dist(Weight(s))$.

Define the expected cost of sub-strategy s as the expected total cost of items inserted in a sub-strategy s .

Remark 6. We can view the sub-strategy as a series of functions $\{f_{i,p} : \mathbb{Z}^{i-1} \rightarrow [0, 1]\}$, where $f_{i,p}$ denotes the choice function of the i^{th} item in the sub-strategy. The $i-1$ parameters in $f_{i,p}$ denote the realized weights of the previous items of type p , and the output denote the probability to insert a new item of type p , otherwise, we do not insert a new item and the whole sub-strategy is stopped. As for the decision of the i^{th} item, we randomly choose an action based on the output of $f_{i,p}$.

For example, suppose an item of a certain type will weigh 2 or 3 with the same probability 0.5, and the cost of one item is 1. One sub-strategy (denoted by s) is: First insert one item. If it weighs 2, then we insert another and stop, otherwise, it will stop immediately. The expected cost of the sub-strategy is 1.5, and the distribution array of $W(s)$ is $(1, 1, 1, 0.5, 0.25, 0, 0, \dots, 0)$, since the total weight is 3 with probability 0.5, 4 with probability 0.25 and 5 with probability 0.25.

Definition 10. *Given two distribution arrays $Dist(A) = (A[1], A[2], \dots, A[W])$ and $Dist(B) = (B[1], B[2], \dots, B[W])$, if for all i , $A[i] \geq B[i]$, then we use $Dist(A) \geq Dist(B)$ to denote this relation.*

Definition 11. *Given one distribution array $Dist(A) = (A[1], A[2], \dots, A[W])$ and one positive real number λ , define*

$$\lambda \cdot Dist(A) := (\min(1, \lambda A[1]), \min(1, \lambda A[2]), \dots, \min(1, \lambda A[W])).$$

Lemma 6. *Suppose that there is a sub-strategy s with expected cost c_s and distribution array $\text{Dist}W(s) = (s[1], s[2], \dots, s[W])$. Then for any $\lambda > 1$, there exists a sub-strategy t with expected cost λc_s , and $t[w] \geq \min(1, \lambda s[w])$ for all $w \in [1, W]$, i.e., $\text{Dist}W(t) \geq \lambda \cdot \text{Dist}W(s)$.*

Proof. Given a sub-strategy s and the distribution array $\text{Dist}W(s) = (s[1], s[2], \dots, s[W])$, supposing w denote the realized total weight of the sub-strategy s , define random variable r_s as the quality factor of s , where r_s is uniformly distributed in $(s[w+1], s[w])$ given w . (Here, we denote $s[W+1] = 0$.)

So, the quality factor r_s is uniformly distributed in $(0, 1]$ without knowing w . Further more, the larger w is, the smaller r_s is.

We build a sub-strategy t given as follow:

We first execute sub-strategy s . If the quality factor r_s satisfies $r_s \leq \frac{1}{\lambda}$, then stop, otherwise we repeat executing sub-strategy s until the quality factor satisfies $r_s \leq \frac{1}{\lambda}$.

Let c_s and c_t denote the expected cost of s and t respectively. They should satisfy $c_t = c_s + (1 - \frac{1}{\lambda})c_t$, which implies $c_t = \lambda c_s$.

Then we prove that for all w , we have $t[w] \geq \min(1, \lambda s[w])$. If $s[w] > \frac{1}{\lambda}$, because we construct sub-strategy t such that we will get a realized s with the quality factor $\leq \frac{1}{\lambda}$, then in the last time we execute s , total weight in s must reach w , so $t[w] = 1$. If $s[w] \leq \frac{1}{\lambda}$, then we also consider the last time we execute s . The quality factor is at most $\frac{1}{\lambda}$ in this case, so the probability that the total weight is greater than w given the quality factor is at most $\frac{1}{\lambda}$ is $\lambda s[w]$. \square

Lemma 7. *Given a sub-strategy s whose expected cost is c_s , suppose the η -approximate array of $\text{Dist}W(s)$ is D' , and the (ζ, η) -approximate array of $\text{Dist}W(s)$ is D'' . Then there is a sub-strategy t_1 whose expected cost is $(1 + \eta)c_s$ and $\text{Dist}W(t_1) \geq D'$. And there is a sub-strategy t_2 whose expected cost is at most $(1 + \eta)^2 c_s$ and $\text{Dist}W(t_2) \geq D''$.*

Proof. First, $D' \leq (1 + \eta)\text{Dist}W(s)$ from the definition of η -approximation. Then from Lemma 6, sub-strategy t_1 exists.

Then we construct sub-strategy t_2 . The sub-strategy is: Execute sub-strategy t_1 first. If its quality factor is larger than $1 - (1 + \eta)^{-\zeta}$, insert W more items of type p to ensure that the total weight reaches W . Then $\text{Dist}W(t_2) \geq D''$ simply follows from the definition of sub-strategy t_2 . The expected cost of t_2 is

$$c_{t_2} = (1 + \eta)c_s + (1 + \eta)^{-\zeta} W c_s \leq (1 + \eta)^2 c_s.$$

\square

Lemma 8. *Given two variables X and Y which satisfy $\text{Dist}(X) \geq \text{Dist}(Y)$, and another variable Z with distribution array $\text{Dist}(Z)$, then $\text{Dist}(X + Z) \geq \text{Dist}(Y + Z)$.*

Proof. For all w , similar to Equation 4, we have

$$\begin{aligned}
(X + Z)[w] &= Z[w] + \sum_{j=1}^{w-1} ((Z[j] - Z[j+1]) \cdot X[w-j]) \\
&\geq Z[w] + \sum_{j=1}^{w-1} ((Z[j] - Z[j+1]) \cdot Y[w-j]) \\
&= (Y + Z)[w].
\end{aligned}$$

□

Corollary 1. *Given two variables X_1, X_2 with the same distribution $\text{Dist}(X)$ and two variables Y_1, Y_2 with the same distribution $\text{Dist}(Y)$, if $\text{Dist}(X) \geq \text{Dist}(Y)$, then $\text{Dist}(X_1 + X_2) \geq \text{Dist}(X_1 + Y_1) \geq \text{Dist}(Y_1 + Y_2)$.*

Define sub-strategy A_{2^i} which always use 2^i items of type p , and define random variable S_{2^i} as the total weight of 2^i items of type p . Then we have $\text{DistW}(A_{2^i}) = \text{Dist}(S_{2^i})$. Recalling the variable B_{2^i} in our algorithm, when computing B_{2^i} , we always round the value larger (from the definition of (ζ, η) -approximation). So, from the induction argument, $\text{Dist}(B_{2^i}) \geq \text{DistW}(A_{2^i})$ for all i . We restate the argument into the following corollary.

Corollary 2. *For all i such that $2^i \leq e_p$,*

$$\text{Dist}(B_{2^i}) \geq \text{DistW}(A_{2^i}) = \text{Dist}(S_{2^i}).$$

Now we want to claim that B_{2^i} does not deviate too much from A_{2^i} . Let c_p denote the cost of one item of type p . We have the following lemma.

Lemma 9. *For all $i \leq \log_2 e_p$, there is a sub-strategy C_{2^i} , whose expected cost is at most $(1 + \eta)^{2^{i+1}} 2^i c_p$, and $\text{DistW}(C_{2^i}) \geq \text{Dist}(B_{2^i})$.*

Proof. $\text{Dist}(B_{2^0}) = \text{Dist}(B_1)$ is the (ζ, η) -approximate array of $\text{DistW}(A_{2^0})$, from Lemma 7, C_{2^0} exists.

Now, assume that for all $i < k$, C_{2^i} exists, and we prove C_{2^k} exists. Recall that B'_{2^k} is the summation of $B_{2^{k-1}}$ and $B_{2^{k-1}}$, and we first construct sub-strategy C'_{2^k} which simply executes $C_{2^{k-1}}$ twice. Then the expected cost of C'_{2^k} is $(1 + \eta)^{2^k} 2^k c_p$, and from Corollary 1, we know $\text{DistW}(C'_{2^{k-1}}) \geq \text{Dist}(B'_{2^k})$. Recall B_{2^k} is the (ζ, η) -approximation of B'_{2^k} . From Lemma 7, we know there exists C_{2^k} whose expected cost is $(1 + \eta)^{2^{k+1}} 2^k c_p$, and $\text{DistW}(C_{2^k}) \geq \text{Dist}(B_{2^k})$. Then we complete the proof by induction. □

We can view the sub-strategy as a new type of item with cost following a distribution, and we know the expected cost of the sub-strategy. From the previous argument, we know that the distribution array of B_{2^i} is bounded by 2 other distribution array A_{2^i} and C_{2^i} , whose expected costs are ‘close to each other.’ We show that our computation leads to small error compared with the optimal value.

Proof (Proof of Theorem 3). First, OPT_W is the optimal value for **Min-SUKP**, and OPT_W^* is the optimal value such that for each type p with $c_p < \theta T$, we use a multiple of e_p number of items of type p together. Then from Theorem 2, we have

$$OPT_W \leq OPT_W^* \leq OPT_W + \frac{\epsilon T}{5}.$$

Let OPT_W^{**} denote the optimal value with items $\{(c_p e_p, Y_p)\}_{p \leq n}$, where $c_p e_p$ is the expected cost of the item and $\text{Dist}(Y_p)$ is the weight distribution array. We next show that

$$\frac{1}{(1 + \eta)^{2 \log_2 W + 2}} OPT_W \leq OPT_W^{**} \leq OPT_W^*.$$

We first show that $OPT_W^{**} \leq OPT_W^*$. Note that OPT_W^* is the optimal value with items $\{(c_p e_p, S_{e_p}^p)\}_{p \leq n}$, and from Corollary 2, we have $\text{Dist}(Y_p) \geq \text{Dist}(S_{e_p}^p)$. First, it is obvious that OPT_w^* and OPT_w^{**} are non-decreasing with respect to w . Then let $OPT_w^* = OPT_w^{**} = 1, \forall w \leq 0$, and suppose for all $w \leq k$, $OPT_w^{**} \leq OPT_w^*$. Then for $w = k + 1$,

$$\begin{aligned} OPT_w^* &= \min_i \left\{ c_i e_i + \sum_{j=1}^{W-1} (S_{e_i}^i[j] - S_{e_i}^i[j+1]) OPT_{w-j}^* + S_{e_i}^i[W] OPT_{w-W}^* \right\} \\ &\geq \min_i \left\{ c_i e_i + \sum_{j=1}^{W-1} (S_{e_i}^i[j] - S_{e_i}^i[j+1]) OPT_{w-j}^{**} + S_{e_i}^i[W] OPT_{w-W}^{**} \right\} \\ &= \min_i \left\{ c_i e_i + S_{e_i}^i[1] OPT_{w-1}^{**} + \sum_{j=2}^W S_{e_i}^i[j] (OPT_{w-j}^{**} - OPT_{w-j+1}^{**}) \right\} \\ &\geq \min_i \left\{ c_i e_i + Y_i[1] OPT_{w-1}^{**} + \sum_{j=2}^W Y_i[j] (OPT_{w-j}^{**} - OPT_{w-j+1}^{**}) \right\} \\ &= \min_i \left\{ c_i e_i + \sum_{j=1}^{W-1} (Y_i[j] - Y_i[j+1]) OPT_{w-j}^{**} + Y_i[W] OPT_{w-W}^{**} \right\} \\ &= OPT_w^{**}, \end{aligned}$$

where we use the Abel transformation, the monotonicity of OPT_w^{**} , the fact that $\text{Dist}(Y_p) \geq \text{Dist}(S_{e_p}^p)$, and the fact that $Y_p[1] = S_{e_p}^p[1] = 1$ (all items have positive integer support).

Then we prove that $OPT_W / (1 + \eta)^{2 \log_2 W + 2} \leq OPT_W^{**}$. First from Lemma 9, we know that there exists sub-strategies $\{(c'_p, Z_p)\}_{p \leq n}$ such that $c'_p \leq (1 + \eta)^{2 \log_2 e_p + 2} e_p c_p$ and $\text{Dist}W(Z_p) \geq \text{Dist}(Y_p)$, where Y_p is the output distribution by Algorithm 4. Note that $e_p \leq W$, otherwise we have

$$e_p c_p > W c_p \geq OPT_W \geq 2\theta T \geq e_p c_p,$$

which is impossible. Then we have $c'_p \leq (1 + \eta)^{2 \log_2 W + 2} e_p c_p$. Note that a sub-strategy can be viewed as a type of item with cost following a distribution, and let OPT'_w denote the optimal value with total weight w and types of items $\{(c'_p, Z_p)\}_{p \leq n}$. Without loss of generality, define $OPT'_w = 1, \forall w \leq 0$. Then first, we have $OPT_w^{**} \geq OPT'_w / (1 + \eta)^{2 \log_2 W + 2}, \forall w \leq 0$. Suppose for all $w \leq k$, $OPT_w^{**} \geq OPT'_w / (1 + \eta)^{2 \log_2 W + 2}$. Then for $w = k + 1$,

$$\begin{aligned}
OPT_w^{**} &= \min_i \left\{ c_i e_i + \sum_{j=1}^{W-1} (Y_i[j] - Y_i[j+1]) OPT_{w-j}^{**} + Y_i[W] OPT_{w-W}^{**} \right\} \\
&\geq \min_i \left\{ c_i e_i + \frac{1}{(1 + \eta)^{2 \log_2 W + 2}} \sum_{j=1}^{W-1} (Y_i[j] - Y_i[j+1]) OPT'_{w-j} \right. \\
&\quad \left. + \frac{1}{(1 + \eta)^{2 \log_2 W + 2}} Y_i[W] OPT'_{w-W} \right\} \\
&\geq \frac{1}{(1 + \eta)^{2 \log_2 W + 2}} \min_i \left\{ c'_i + Y_i[1] OPT'_{w-1} \right. \\
&\quad \left. + \sum_{j=2}^W Y_i[j] (OPT'_{w-j} - OPT'_{w-j+1}) \right\} \\
&\geq \frac{1}{(1 + \eta)^{2 \log_2 W + 2}} \min_i \left\{ c'_i + Z_i[1] OPT'_{w-1} \right. \\
&\quad \left. + \sum_{j=2}^W Z_i[j] (OPT'_{w-j} - OPT'_{w-j+1}) \right\} \\
&= \frac{1}{(1 + \eta)^{2 \log_2 W + 2}} \min_i \left\{ c'_i \right. \\
&\quad \left. + \sum_{j=1}^{W-1} (Z_i[j] - Z_i[j+1]) OPT'_{w-j} + Y_i[W] OPT'_{w-W} \right\} \\
&= \frac{1}{(1 + \eta)^{2 \log_2 W + 2}} OPT'_w,
\end{aligned}$$

where we use the Abel transformation, the monotonicity of OPT'_w , the fact that $\text{Dist}(Z_p) \geq \text{Dist}(Y_p)$, the fact that $Y_p[1] = Z_p[1] = 1$ (all items have positive integer support) and the induction assumption. Then we prove $OPT'_W / (1 + \eta)^{2 \log_2 W + 2} \leq OPT_W^{**}$ by induction. Also notice that OPT'_W is the optimal value using the sub-strategies $\{(c'_p, Z_p)\}_{p \leq n}$, and each sub-strategy can be constructed through the original type of items, so it is clear that $OPT_W \leq OPT'_W$. Then we have proved that

$$\frac{1}{(1 + \eta)^{2 \log_2 W + 2}} OPT_W \leq OPT_W^{**} \leq OPT_W^*. \quad (6)$$

Finally, from Theorem 1, we can get

$$(1 - \delta)\left(1 - \frac{\epsilon}{10}\right)\hat{V} \leq OPT_W^{**} \leq \hat{V}.$$

According to Theorem 2 and Equation 6, we can get

$$\frac{1}{(1 + \eta)^{2 \log_2 W + 2}} OPT_W \leq \hat{V} \leq (1 - \delta)^{-1} \left(1 - \frac{\epsilon}{10}\right)^{-1} \left(OPT + \frac{\epsilon T}{5}\right).$$

So we have

$$(1 - \epsilon)OPT_W \leq \hat{V} \leq (1 + \epsilon)OPT_W.$$

□

G Proof of Theorem 4

[Proof of Theorem 4]

Proof. Recall the value of parameters we select:

$$\delta = \frac{\epsilon^2}{100n} \sim O\left(\frac{\epsilon^2}{n}\right),$$

$$\eta = \frac{\epsilon}{10 \log W} \sim O\left(\frac{\epsilon}{\log W}\right),$$

and

$$\zeta = \log_{1+\eta} \frac{W}{\eta} \sim O\left(\frac{\log^2 W}{\epsilon} + \frac{\log W \log \frac{1}{\epsilon}}{\epsilon}\right).$$

Without loss of generality, we assume $\frac{1}{\epsilon}$ is $o(W)$. Otherwise, we can just use the pseudo-polynomial dynamic programming algorithm to compute the optimal value. Then,

$$\zeta \sim O\left(\frac{\log^2 W}{\epsilon}\right).$$

To compute the distribution of the summation of some i.i.d variables, for each type p , we compute approximate distribution array of $S_1, S_2, S_4, S_8, \dots, S_{e_p}$ where e_p is $O(W)$, which means we need to compute $O(\log W)$ distribution arrays. We compute the (ζ, η) -approximate array, and get (ζ, η) -approximate array of S_1 from the oracle by binary search which can be computed in $O(\zeta \log \zeta)$ time. And we compute (ζ, η) -approximate array of $S_2, S_4, S_8, \dots, S_{e_p}$ by convolution in $O(\zeta^2 \log \zeta)$ time. We also need to compute the approximate distribution array of at most n types of items, so the total time is $O(n\zeta^2 \log \zeta \log W)$

In the dynamic programming, we compute f_x where x takes $\frac{1}{\delta}$ different values. When computing f_x , we need to use binary search on a value y , so we compute

the approximate value of OPT_y in $O(\frac{1}{\delta})$ time. We also need to enumerate which item type to select, so the total time is $O(n^{\frac{1}{\delta^2}} \log W)$.

Besides, we need to compute an approximate value T by our constant factor approximation algorithm, which takes $O(n \log W)$ time.

In conclusion, the total time complexity is

$$O\left(\frac{n \log^6 W}{\epsilon^3} + \frac{n^3 \log W}{\epsilon^4}\right).$$

So, Algorithm 5 runs in polynomial time.

□