Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity*

Brian C. Dean
Computer Science and AI Lab (CSAIL)
MIT†
bdean@csail.mit.edu

Michel X. Goemans
Dept. of Mathematics
MIT†
goemans@math.mit.edu

Jan Vondrák
Dept. of Mathematics
MIT†
vondrak@math.mit.edu

Abstract

We consider a stochastic variant of the NP-hard 0/1 knapsack problem in which item values are deterministic and item sizes are independent random variables with known, arbitrary distributions. Items are placed in the knapsack sequentially, and the act of placing an item in the knapsack instantiates its size. Our goal is to compute a solution "policy" that maximizes the expected value of items placed in the knapsack, and we consider both non-adaptive policies (that designate a priori a fixed sequence of items to insert) and adaptive policies (that can make dynamic choices based on the instantiated sizes of items placed in the knapsack thus far). We show that adaptivity provides only a constant-factor improvement by demonstrating a greedy non-adaptive algorithm that approximates the optimal adaptive policy within a factor of 7. *We also design an adaptive polynomial-time algorithm which approximates the optimal adaptive policy within a factor of* $5 + \epsilon$, *for any constant* $\epsilon > 0$.

1. Introduction

The classical NP-hard knapsack problem involves n items with values $v_1 \dots v_n$ and sizes $s_1 \dots s_n$, and asks for a maximum-value set of items that fits into a unit-capacity knapsack. We consider a natural stochastic generalization of this problem in which values are deterministic and item sizes are independent random variables with known, completely arbitrary distributions. The actual size of an item is unknown until we instantiate it by attempting to place it in the knapsack. With a goal of maximizing the expected value of items placed in the knapsack, we seek to design a solution "policy" for sequentially inserting items until the capacity is eventually exceeded (no value is received from the final item whose size overruns the capacity). A *non-adaptive* policy designates a priori a sequence of items to insert, while an *adaptive* policy has the flexibility to choose items dynamically based on the instantiated sizes of items already inserted (see for example Figure 1). It is at least NP-hard to compute optimal non-adaptive and adaptive policies, since both problems generalize the classical knapsack problem.

Our problem is motivated by the application of scheduling a maximum-value subset of n tasks with uncertain durations within a fixed amount of time. To the best of our knowledge, this problem has never been addressed in the literature, although one does find several related problems in the stochastic optimization and scheduling literature, which we summarize in Section 1.2.

1.1. Outline of Results

In this paper we provide both non-adaptive and adaptive approximation algorithms whose expected values are within a constant factor of an optimal adaptive policy. We characterize the *benefit of adaptivity* for this problem, in showing the rather surprising result that the *adaptivity gap* (what we call the ratio of the expected value of an optimal adaptive solution to the expected value of an optimal non-adaptive solution) is only a constant factor. Finally, we provide complexity results indicating that the stochastic knapsack problem has a degree of complexity beyond that of NP-completeness: some natural questions concerning optimal adaptive policies turn out to be PSPACE-hard.

In addition to the analysis of the benefit of adaptivity, a novel aspect of this work is the use of new techniques to characterize and bound the performance of an optimal adaptive algorithm, a task that seems much more difficult than that of bounding an optimal algorithm either in a non-stochastic setting or even in the more classical stochastic scheduling setting. The following is a brief overview of our results and the structure of this extended abstract:

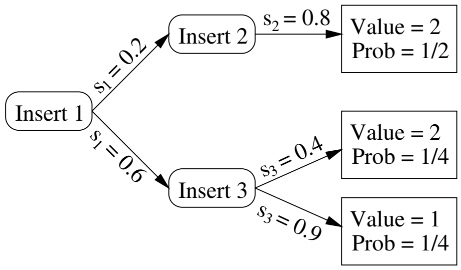
- In Section 3 we discuss the issue of bounding an optimal adaptive policy. We see that if every item has a

* Supported in part by NSF grants CCR-0098018 and ITR-0121495.

† 77 Massachusetts Ave., Cambridge, MA 02139, USA.

	Item	Size Distribution (capacity = 1)	
(a)	1	0.2 (prob 1/2)	0.6 (prob 1/2)
	2	0.8 (prob 1)	
	3	0.4 (prob 1/2)	0.9 (prob 1/2)

(b) Optimal adaptive policy for instance (a) of expected value 1.75 (the optimal non-adaptive expected value is only 1.5.):



	Item	Value	Size Distribution (capacity = 1)	
(c)	1	ε	0 (prob 1/2)	1 (prob 1/2)
	2	ε	1 (prob 1)	
	3	1	0 (prob ε)	1+ε (prob 1-ε)

Figure 1. Sample instances of stochastic knapsack problems.

value equal to its expected size, in a unit-size knapsack it is possible for even a non-adaptive policy to obtain an expected value of 2. We then use martingales to prove that no adaptive policy can achieve an expected value more than 2. This bound allows us to compare our subsequent approximation algorithms with an optimal adaptive policy.

- In Section 4 we demonstrate a simple and efficient greedy algorithm that computes a non-adaptive policy whose expected value is at least $1/7$ times that of an optimal adaptive policy, thereby giving an upper bound of 7 on the adaptivity gap for our problem. The example in Figure 1(c) provides a lower bound of 1.25 on the adaptivity gap, since an optimal adaptive policy delivers expected value 2.5ϵ while the best non-adaptive policy has expected value 2ϵ (ignoring lower-order terms in both cases).
- In Section 5, we consider a slightly weaker model than the non-adaptive policy, where we must choose a priori a set of elements to place in the knapsack, and we only receive its value if the entire set fits. In this model, we show how to obtain an expected value at least $1/9.5$ times that of an optimal adaptive policy.
- In Section 6 we describe another, more intricate, polynomial-time adaptive policy (i.e., a policy that takes polynomial time to decide the next item to insert into the knapsack given the sizes of items inserted so far) whose expected value is at least $1/5 - \epsilon$ times that of an optimal adaptive policy, for any constant $\epsilon > 0$.
- Finally, in Section 7 we prove the PSPACE-hardness of some questions concerning optimal adaptive policies, namely "is it possible to fill the knapsack exactly to its capacity with probability at least p?". Also, it is PSPACE-hard to maximize the expected value achieved by an adaptive policy in a more general setting, where items are allowed to have random values, correlated with the random sizes.

1.2. Literature Review

Our problem is perhaps best characterized as a scheduling problem. Stochastic scheduling problems, where jobs sizes are random variables with known probability distributions, have been studied quite extensively in the literature, dating back as far as 1966 [11]. However, no consideration of our particular objective (packing a subset of jobs within a fixed amount of time) seems to appear in the stochastic scheduling literature. Almost all stochastic scheduling problems studied to date involve scheduling all jobs and minimizing either expected makespan or expected sum of weighted completion times on one or more machines — see [14] for a comprehensive survey of these problems.

The notion of an "adaptive" scheduling policy is quite prevalent in the stochastic scheduling literature, as opposed to the stochastic knapsack literature, where for the most part adaptivity has hardly received any attention. The distinction between adaptive and non-adaptive solutions is an important aspect of stochastic programming. In the stochastic programming literature [2], adaptivity is often studied in the context of optimization with *recourse*, where one commits to a partial solution and then, after witnessing the instantiation of this partial solution, must react appropriately to complete the solution. Often only a single "stage" of recourse is considered (see for example [8]), whereas our problems involve unlimited levels of recourse.

Derman et al. [5] consider the adaptive stochastic knapsack problem where multiple copies of items are permitted, as well as the related *knapsack cover* problem in this same setting, where the goal is to cover the capacity of the knapsack with a minimum-cost set of items (for this variant, multiple copies of items are required since otherwise it might not be possible to cover the knapsack). A prototypical application of the stochastic knapsack cover problem is keeping a machine running for a certain duration, where the machine depends on a critical part (e.g. a light bulb) that periodically fails and must be replaced. The different items correspond to potential replacements, each having a deterministic cost and an uncertain lifetime. Derman et al. provide dynamic programming formulations for these problems, and also prove that if item sizes are exponentially distributed, both problems are solved by greedily scheduling jobs according to value or cost divided by expected size.

Stochastic knapsack problems with deterministic sizes and random values have been studied by several authors [3, 7, 12, 13], all of whom consider the objective of computing a fixed set of items fitting in the knapsack that has maximum probability of achieving some target value (in this setting, maximizing expected value is a much simpler, albeit still NP-hard, problem since we can just replace every item's value with its expectation). Several heuristics have been proposed for this variant (e.g. branch-and-bound, preference-order dynamic programming), and adaptivity is not considered by any of the authors.

Two recent papers due to Kleinberg et al. [9] and Goel and Indyk [6] consider yet another variant of stochastic knapsack problem motivated by the application of assigning potentially bursty data connections to capacitated links in a communication network. Like our model, they consider items with deterministic values and random sizes. However, their objective is quite different: given a specified overflow probability p, they wish to find a maximum-value set of items whose probability of overflowing the knapsack is at most p. Adaptivity is not considered, as it makes less sense in this particular context. Kleinberg et al. consider only the case where item sizes have a Bernoulli-type distribution (with only two possible sizes for each item), and for this case they provide a polynomial-time $O(\log 1/p)$-approximation algorithm as well as several pseudo-approximation results. For job sizes that have Poisson or exponential distributions, Goel and Indyk provide a PTAS, and for Bernoulli-distributed items they give a quasi-polynomial approximation scheme whose running time depends polynomially on n and $\log 1/p$. Kleinberg et al. show that the problem of computing the overflow probability of a set of items, even with Bernoulli distributions, is #P-hard. Consequently, it is also #P-hard to solve the problem variant mentioned above with deterministic sizes and random values, whose goal is maximizing the probability of achieving some specified target value.

In contrast to these cited results, we do not assume that the distributions of item sizes are exponential, Poisson or Bernoulli; our algorithms work for arbitrary distributions.

Game theory provides yet another framework in which our problem can be cast, as a number of researchers have studied "games against nature" (see [10]), in which players try to maximize their utility in a game by making moves in alternation with a player known as "nature" who moves according to some probabilistic model. We will discuss connections with these types of games in more detail when we address complexity in Section 7.

2. Preliminaries

Let $\mathcal{I} = [n]$ denote the set of items in our problem instance. For each item $i \in \mathcal{I}$, let v_i denote its deterministic[1] value and let s_i be a random variable corresponding to its size. We assume without loss of generality that our instance is scaled so the capacity of the knapsack is 1. For each item, we will find it convenient to consider the *mean truncated size* $\mu_i = \mathbf{E}[\min\{s_i, 1\}]$. It is assumed that for every item i we know μ_i, and that we can evaluate any point on the cumulative distribution for s_i in $O(1)$ time. For a set of items S, we define $size(S) = \sum_{i \in S} s_i$, $\mu(S) = \sum_{i \in S} \mu_i$ and $val(S) = \sum_{i \in S} v_i$. Note that *truncated sizes* are used to determine $\mu(S)$. This measure of size will be useful to bound the performance of both adaptive and non-adaptive policies. We estimate the probability that a set fits in the knapsack by the following variant of Markov's inequality.

Lemma 1. $Pr[size(S) < 1] \geq 1 - \mu(S)$.

Proof. $Pr[size(S) \geq 1] \leq \mathbf{E}[\min\{size(S), 1\}] \leq \mathbf{E}\left[\sum_{i \in S} \min\{s_i, 1\}\right] = \mu(S)$. $\square$

A non-adaptive policy is just an ordering of items in $\mathcal{I}$. An adaptive policy is formally defined by a mapping $\mathcal{P} : 2^{\mathcal{I}} \times \mathbb{R}^+ \to \mathcal{I}$ specifying which item to insert next, given the subset of items still available and the total remaining capacity. We can picture an adaptive policy as a decision tree, or more precisely as a decision DAG, whose nodes are a subset of $2^{\mathcal{I}} \times \mathbb{R}^+$. At every node the policy specifies which item to insert next, and the directed edges leaving this node correspond to the possible outcomes of the random instantiation of the size of this item. Note that an explicit representation of this DAG may be of exponential size.

Example. Consider a knapsack of large integral capacity and n unit-value items, half of type A and half of type B. Items of type A take size 1, 2, or 3 each with probability p, and then sizes $5, 7, 9, \ldots$ with probability $2p$. Items of type B take size 1 or 2 with probability p, and sizes $4, 6, 8, \ldots$ with probability $2p$. If p is sufficiently small, then the optimal policy uses a type A item (if available) if the remaining capacity has odd parity and a type B item if the remaining capacity has even parity. The corresponding DAG would have at least $\binom{n}{n/2}$ leaves.

3. Bounding the optimal adaptive policy

The first question we address is the *benefit of adaptivity*. Assuming an arbitrary adaptive policy (which can implicitly use unbounded computational power to decide about

1 If item values are independent random variables with known distributions, we can replace item values by their (deterministic) expectations, since our goal is to maximize expected value.

the next item to put in the knapsack), what nontrivial upper bound can we get on its performance? For now, we disregard the item values, and we consider $\mu(S)$ as our objective function. Can an adaptive policy successfully insert a set of mean size $\mu(S)$ substantially larger than 1?

First, note that if a policy is allowed to see the actual size of an item *before* it decides whether to insert it, there is no reasonable bound on its performance. We call such a policy *omniscient*, and it can be seen that the performance gap between omniscient and adaptive can be $\Omega(n)$.

Example. Consider n items whose sizes are each 0 or 2 with probability $1/2$. The omniscient policy inserts every item whose size turns out to be 0, which is $n/2$ items on the average. An adaptive policy can only insert items blindly, terminating when one of them gets size 2, so the expected number of inserted items is only 1.

Therefore it is crucial to exploit the property of *irrevocable decisions*, which forces the adaptive policy to keep an item even if its size turns out to be unfortunately large. This property reflects the reality of job scheduling, where we cannot go back in time, in case a job has taken too long to complete! The most convenient framework for this kind of random process is that of *martingales*. We define the following sequence of random variables: For an adaptive policy $\mathcal{P}$, let S_t denote the set of the first t items chosen by $\mathcal{P}$. Once the size of S_t overflows, no further items are added to S_t, and S_t remains constant after this. Denote the truncated item sizes by $\tilde{s}_i = \min\{s_i, 1\}$. We define

$$X_t = \sum_{i \in S_t} (\tilde{s}_i - \mu_i).$$

It is easy to verify that X_t is a martingale. First let's condition on a specific decision node, defined by S_t and the sizes of the respective items. Suppose that in this situation, our policy chooses item j. After inserting item j, we get $S_{t+1} = S_t \cup \{j\}$, and $\mathbf{E}[X_{t+1} \mid S_t, \{\tilde{s}_i : i \in S_t\}] = \sum_{i \in S_t} (\tilde{s}_i - \mu_i) + \mathbf{E}[\tilde{s_j} - \mu_j] = \sum_{i \in S_t} (\tilde{s}_i - \mu_i)$. Obviously, this is also true if $size(S_t) > 1$ and no item is added anymore. Conditioning on $X_t = \sum_{i \in S_t} (\tilde{s}_i - \mu_i) = \xi$ for some fixed ξ, we get

$$\mathbf{E}[X_{t+1} | X_t = \xi] = \xi$$

which proves that $\{X_t\}$ is a martingale. Since $X_0 = 0$, we get $\mathbf{E}[X_t] = 0$ for any $t > 0$. This means $\mathbf{E}[\sum_{i \in S_t} \tilde{s}_i] = \mathbf{E}[\mu(S_t)]$. [2] The process stops once $size(S_t) > 1$ or we have no more items left. Since $\sum_{i \in S_t} \tilde{s}_i \leq size(S_t)$ and each $\tilde{s}_i$ is at most 1, we get $\sum_{i \in S_t} \tilde{s}_i \leq 2$ for any $t > 0$. Consequently, $\mathbf{E}[\mu(S_t)] \leq 2$. The mean size of all the items inserted by $\mathcal{P}$ (including the first one which exceeds knapsack capacity) is $\mu(S) = \lim_{t \to \infty} \mu(S_t)$, and therefore $\mathbf{E}[\mu(S)] = \lim_{t \to \infty} \mathbf{E}[\mu(S_t)] \leq 2$. We have proved the following, which will be the key tool in comparing the performance of an optimal adaptive policy to our approximation algorithms, adaptive or non-adaptive.

Lemma 2. *For any adaptive policy $\mathcal{P}$, let S be the (random) set of all items that $\mathcal{P}$ tries to insert. Taking the expectation over executions of the policy,*

$$\mathbf{E}[\mu(S)] \leq 2.$$

Note. It might seem that we get a factor of 2 only because we allow the first overflowing item to be counted as well. But this is not the case. Assume that we have an unlimited supply of items which attain size 1 with probability p and size 0 with probability $1-p$. A policy inserts a sequence of these items, until two of them get size 1 and the knapsack overflows. The expected number of trials before this occurs is $2/p - 1$, and therefore we insert *successfully* a set of items of expected mean size $\mathbf{E}[\mu(S)] = 2 - p$ which can be arbitrarily close to 2.

4. The greedy algorithm

Let ADAPT denote the maximum expected value obtained by an adaptive policy for a given instance of the stochastic knapsack. We show a simple practical algorithm, which achieves an approximation factor of 7, compared to ADAPT. Since our algorithm is not adaptive (it specifies a fixed sequence of items to be inserted), this answers the question whether adaptivity can bring a substantial benefit. The gap between adaptive and non-adaptive policies is only a constant factor. (And we have a lower bound of 1.25 on this factor, shown in Figure 1.)

We assume that all items have nonzero expected size. (If there are items of zero size, we can insert them for free, just like the optimal policy would. This can only improve the approximation factor.) Let's partition the set of items into "light" and "heavy": $\mathcal{I} = \mathcal{I}_L \cup \mathcal{I}_H$. Choose a threshold value $\sigma \in (0,1)$ and call an item *light* if $\mu_i \leq \sigma$ or *heavy* if $\mu_i > \sigma$. Assume that $\mathcal{I}_L = \{1, 2, 3, \ldots\}$ and the light items are ordered so that

$$\frac{v_1}{\mu_1} \geq \frac{v_2}{\mu_2} \geq \frac{v_3}{\mu_3} \geq \ldots$$

We denote by $M_k = \sum_{i=1}^{k} \mu_i$ the cumulative mean size of the first k items. Let n^* be the maximum number such that $M_{n^*} \leq 1$. (We assume for convenience that the total mean size of light items is at least 1, otherwise we can add dummy items of value 0.) Define

- $m_G = \sum_{k=1}^{n^*} v_k (1 - M_k)$
- $m_1 = \max\{v_i \Pr[s_i \leq 1] : i \in \mathcal{I}\}$

2 Note that two different notions of "expectation" are used here. $\mathbf{E}[\ldots]$ refers to expectation over different executions of the adaptive policy, while $\mu(S)$ is the sum of mean truncated sizes μ_i of items $i \in S$.

The algorithm. We calculate m_G and m_1 and choose the larger of the two. If it's m_1, we insert the item which yields expected value $m_1 = v_i \Pr[s_i \leq 1]$ and stop. If m_G is larger, we insert the light items $1, 2, 3, \ldots$ until the knapsack overflows, or all items have been used. By Lemma 1, the probability that the k-th item is inserted successfully is at least $1 - M_k$, and the expected value achieved in this way is at least m_G. Thus our algorithm achieves expected value at least $GREEDY = \max\{m_1, m_G\}$.

In the following, we compare $GREEDY$ to $ADAPT$. The intuitive idea is that the sequential insertion algorithm (m_G) provides good performance, if all items are light. However, there might be heavy items with a very high value. In order to cover this case, we try to insert each item individually as well (m_1).

First let's handle the light items. For that purpose, it will be useful to refer to the *fractional knapsack solution*. Let's define a function $\Phi(c)$ which denotes the best fractional solution using only light items and capacity c:

$$\Phi(c) = \max \left\{ \sum_{i \in \mathcal{I}_L} v_i x_i : 0 \leq x_i \leq 1, \sum_{i \in \mathcal{I}_L} \mu_i x_i \leq c \right\}.$$

This is a relaxation of the knapsack problem with item sizes μ_i. Any set of light items L defines a characteristic vector $\chi_i = [i \in L]$ which is feasible for capacity $c = \mu(L)$. This implies $val(L) = \sum_i v_i \chi_i \leq \Phi(\mu(L))$.

Now it is convenient that the light items are ordered by v_i/μ_i. It is well-known that the optimal solution is to pack items in the order of decreasing v_i/μ_i, using as many as possible and a suitable fraction of the last one. Thus the optimal solution has a simple form:

$$\forall k \in \mathcal{I}_L; \forall \xi \in [0, \mu_k); \quad \Phi(M_{k-1} + \xi) = \sum_{i=1}^{k-1} v_i + v_k \frac{\xi}{\mu_k}.$$

and $\Phi(c) = val(\mathcal{I}_L)$ for $c \geq M_l$ where $l = |\mathcal{I}_L|$ (see Figure 2; as we mentioned, we assume $M_l \geq 1$).

Consequently, $\Phi(c)$ is piecewise linear and concave. This will be useful in our analysis of the greedy algorithm, but first let's show an elegant connection with the performance of any adaptive policy, using only light items.

Lemma 3. *If all items are light, then*

$$ADAPT \leq \Phi(2).$$

Proof. Fix an optimal adaptive policy $\mathcal{P}$ and suppose it inserts a set L, which is a random variable. Let $x_i = \Pr[i \in L]$. By Lemma 2, $\sum_i x_i \mu_i = \mathbf{E}[\mu(L)] \leq 2$ and therefore $\mathbf{E}[val(L)] = \sum_i x_i v_i \leq \Phi(\sum_i x_i \mu_i) \leq \Phi(2)$. $\square$

Lemma 4. $m_G \geq \frac{1-\sigma}{2} \Phi(1)$.

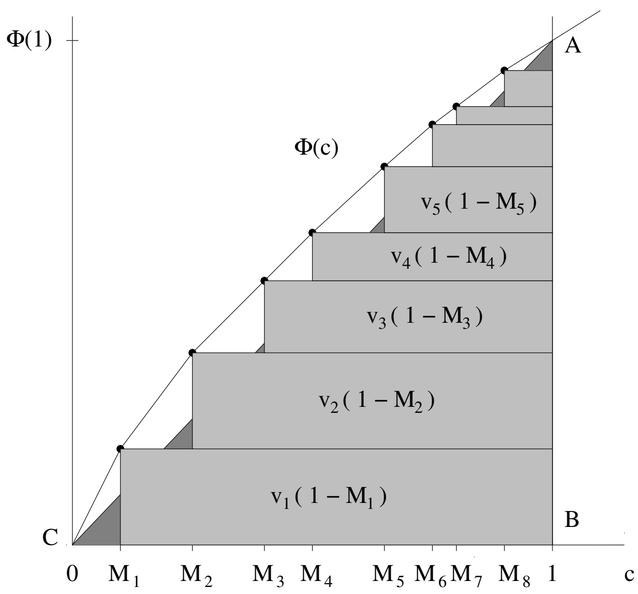


Figure 2. A graph of the fractional knapsack solution $\Phi(c)$. The dark shaded (partially covered) triangle ABC has area $\Phi(1)/2$. The light shaded area marks the performance of the greedy algorithm m_G.

Proof. Compare the area in Figure 2, corresponding to m_G, with the dark shaded triangle ABC of area $\Phi(1)/2$. The part of the triangle which is not covered by area m_G consists of small triangles whose heights sum up to at most $\Phi(1)$, and the base of each small triangle is $\mu_i \leq \sigma$. Therefore $m_G \geq \Phi(1)/2 - \Phi(1)\,\sigma/2$. $\square$

By concavity, $\Phi(2) \leq 2\Phi(1) \leq \frac{4}{1-\sigma} m_G$. Thus we have a $\frac{4}{1-\sigma}$-approximation algorithm for light items. Now, we would like to incorporate heavy items. However, we first derive a bound on $\Phi(c)$ for all $c \geq 0$.

Lemma 5. *For $c \leq 1 - \sigma$, $m_G \geq (1 - c)\,\Phi(c)$.*

Proof. Let k be such that $M_{k-1} \leq c < M_k$. The value obtained by the greedy algorithm is at least

$$m_G \geq \sum_{i=1}^{k} v_i (1 - M_i) \geq (1 - c) \left(\sum_{i=1}^{k-1} v_i + v_k \frac{1 - M_k}{1 - c} \right).$$

Now we use an inequality: $(1 - M_k)(M_k - M_{k-1}) \geq (1 - c)(c - M_{k-1})$. This holds because the sum of each pair of factors is the same $(1 - M_{k-1})$, and $(c - M_{k-1})$ is the smallest of all the factors: $c - M_{k-1} \leq M_k - M_{k-1}$ and

$c - M_{k-1} \le 1 - \sigma - M_{k-1} \le 1 - M_k$. Therefore:

$$m_G \ge (1-c)\left(\sum_{i=1}^{k-1} v_i + v_k \frac{c - M_{k-1}}{M_k - M_{k-1}}\right) = (1-c)\,\Phi(c).$$

$\square$

Lemma 6. *For light items defined by $\sigma = 1/3$ and $c \ge 0$,*

$$\Phi(c) \le (1+3c)\, m_G.$$

Proof. By Lemma 4, we have $\Phi(1) \le \frac{2}{1-\sigma} m_G = 3m_G$. For $c \ge 1$, by concavity: $\Phi(c) \le c\Phi(1) \le 3cm_G$. For $c \in [\frac{2}{3}, 1]$, by monotonicity: $\Phi(c) \le \Phi(1) \le 3m_G \le (1+3c)m_G$. Finally, for $c \in [0, \frac{2}{3}]$, by Lemma 5: $\Phi(c) \le \frac{m_G}{1-c} \le (1+3c)m_G$. $\square$

In general, one can show that $\Phi(c) \le \left(1 + \frac{2c}{1-\sigma}\right) m_G$, but we will not need this as we select $\sigma = 1/3$. Thus for any set L of light items, $val(L) \le \Phi(\mu(L)) \le (1+3\mu(L))m_G$. It remains to bound the value that an adaptive policy can get for heavy items.

Lemma 7. *For an adaptive policy $\mathcal{P}$, denote by $H' \subseteq \mathcal{I}_H$ the (random) set of heavy items that $\mathcal{P}$ attempts to insert in the knapsack, and by $H \subseteq H'$ the subset of those that fit. Then $\mathbf{E}[val(H)] \le \mathbf{E}[|H'|]\, m_1$.*

Proof. Let $x_i = \Pr[i \in H]$ and $x'_i = \Pr[i \in H']$. Since i, whenever inserted, fits with probability at most $\Pr[s_i \le 1]$, we get $x_i \le x'_i \Pr[s_i \le 1]$, and $\mathbf{E}[val(H)] = \sum_i x_i v_i \le \sum_i x'_i v_i \Pr[s_i \le 1] \le \sum_i x'_i m_1 = \mathbf{E}[|H'|]\, m_1$. $\square$

We conclude the analysis of the greedy algorithm.

Theorem 1. *For $\sigma = 1/3$,*

$$ADAPT \le 7 \cdot GREEDY.$$

Proof. Consider an optimal adaptive policy and the set S' which it tries to insert. Partition S' into light and heavy items: $S' = L' \cup H'$. Similarly, write the set of successfully inserted items as $S = L \cup H$. For heavy items, we use Lemma 7:

$$\mathbf{E}[val(H)] \le \mathbf{E}[|H'|]\, m_1 \le 3\mathbf{E}[\mu(H')]\, m_1.$$

For the light items, we have Lemma 6: $val(L) \le \Phi(\mu(L)) \le (1+3\mu(L))\, m_G$, and

$$\begin{aligned} ADAPT &= \mathbf{E}[val(L)] + \mathbf{E}[val(H)] \\ &\le (1 + 3\mathbf{E}[\mu(L')] + 3\mathbf{E}[\mu(H')])\, GREEDY. \end{aligned}$$

Since $\mathbf{E}[\mu(L')] + \mathbf{E}[\mu(H')] = \mathbf{E}[\mu(S')] \le 2$ by Lemma 2, the result follows. $\square$

Example. Consider an instance where all items have unit value, one item has deterministic size ϵ and there is a large number of items whose sizes are either 0 or $1-\epsilon$, with expected size $1/3 + \epsilon$. Our greedy algorithm places only one item in the knapsack, while an optimal policy obtains an expected value of $6 - O(\epsilon)$. Hence, for $\sigma = 1/3$ the greedy algorithm can be as far as a factor of 6 away from the optimal policy. It would be interesting to see if there are examples which are bad for all choices of σ.

5. Ordered policies and fixed sets

We next discuss approximation results for two slightly different models than the non-adaptive model considered above. The first of these is the *fixed set* model, where we must specify apriori a set of items S to insert into the knapsack, and we only receive the value of S if all these items successfully fit. The second is the *ordered* adaptive model, where we must process the items in some given ordering and for each item in sequence, we must (adaptively) decide whether to insert it into the knapsack or discard it forever. The ordered case can be further subdivided based on whether our algorithm is allowed to choose the ordering, or whether the ordering is provided as input. An interesting problem in the first case is computing the "best" ordering — from our greedy result above we know that the expected value obtained by an adaptive policy for the best ordering must be within a factor of 7 of $ADAPT$. Below, we give a polynomial-time algorithm for computing a set of items S for which $val(S)(1-\mu(S)) \ge ADAPT/9.5$, thereby giving a 9.5-approximation in the fixed set model as well as showing that an optimal ordered policy for *any* ordering of the items can be no further than a factor of 9.5 away from $ADAPT$ (since a feasible ordered policy is always just to insert the items in S).

We now consider the computation of a set S whose value times probability of fitting is at least $ADAPT/9.5$. Define:

- $m_1 = \max\{v_i \Pr[s_i < 1] : i \in \mathcal{I}\}$
- $m_2 = \max\{val(S)(1-\mu(S)) : S \subseteq \mathcal{I}\}$

Note that m_1 can be determined easily and m_2 can be approximated to within any relative error by running the standard knapsack approximation scheme with mean sizes. Both values correspond to the expected benefit of inserting an item or a set of items, considering the probability that the set fits in the knapsack. Our set of items is the better of the two: $FIXED = \max\{m_1, m_2\}$.

We now compare $ADAPT$ to $FIXED$.

Lemma 8. *For any set $L \subseteq \mathcal{I}_L$ of light items,*

$$val(L) \le \left(1 + \frac{4\mu(L)}{1-\sigma^2}\right) m_2.$$

Proof. We proceed by induction on $|L|$. If $\mu(L) \geq (1-\sigma)/2$, choose a minimal $K \subseteq L$, such that $\mu(K) \geq (1-\sigma)/2$. Since the items have mean size at most σ, K will not exceed mean size $(1+\sigma)/2$. By induction,

$$val(L \setminus K) \leq \left(1 + \frac{4(\mu(L)-\mu(K))}{1-\sigma^2}\right) m_2$$

and since $m_2 \geq val(K)(1-\mu(K))$, for $\mu(K) \in [\frac{1-\sigma}{2}, \frac{1+\sigma}{2}]$ we get $\mu(K)m_2 \geq val(K)\mu(K)(1-\mu(K)) \geq \frac{1}{4}(1-\sigma^2)val(K)$, and $val(L) = val(L \setminus K) + val(K) \leq \left(1 + \frac{4\mu(L)}{1-\sigma^2}\right) m_2$. The last case is $\mu(L) < (1-\sigma)/2$. Then we can see directly that $val(L) \leq \frac{m_2}{1-\mu(L)} \leq (1 + \frac{4\mu(L)}{1-\sigma^2})\, m_2$. □

Theorem 2. *For an optimal adaptive policy, the expected value of items inserted successfully in the knapsack is*

$$ADAPT \leq 9.5\, FIXED.$$

Proof. Fix an optimal adaptive policy and let $S' = H' \cup L'$ denote the set of heavy and light items that the policy attempts to insert. Let $S = H \cup L$ denote the items successfully inserted. By Lemma 7, we have

$$\mathbf{E}[val(H)] \leq \mathbf{E}[|H'|]m_1 \leq \left(\frac{\mathbf{E}[\mu(H')]}{\sigma}\right) m_1.$$

For light items, we use Lemma 8:

$$\mathbf{E}[val(L)] \leq \left(1 + \frac{4\mathbf{E}[\mu(L)]}{1-\sigma^2}\right) m_2.$$

We choose $\sigma = \sqrt{5} - 2$ which yields

$$\mathbf{E}[val(S)] \leq (1+(2+\sqrt{5})(\mathbf{E}[\mu(H')]+\mathbf{E}[\mu(L')]))\, FIXED.$$

Lemma 2 says that $\mathbf{E}[\mu(S')] = \mathbf{E}[\mu(H')] + \mathbf{E}[\mu(L')] \leq 2$ which proves $ADAPT \leq (5 + 2\sqrt{5})\, FIXED$. [3] □

6. A $(5+\epsilon)$-Approximate Adaptive Policy

In this section we describe an adaptive policy that is within a factor of $5+\epsilon$ of an optimal adaptive policy. For any constant $\epsilon > 0$, the policy requires polynomial time to decide which item to insert next into the knapsack, given the set of items already inserted and their instantiated sizes. Throughout this section, we assume that probability distributions are discrete, and that we are given a value B such that it requires at most B bits to represent any item value, instantiated item size, or probability value obtained by evaluating the cumulative distribution for some item size. The running time of our algorithm is polynomial in n and B.

3 We can only approximate m_2 within $(1+\epsilon)$, but since the constant factor is $5 + 2\sqrt{5} \approx 9.472 < 9.5$, we can find a 9.5-approximate fixed set in polynomial time.

The greedy algorithm gives a $\frac{4}{1-\sigma}$-approximation policy if all items are light (Lemmas 3,4). In this section, we describe a $(1+\epsilon')$-approximate policy if all items are heavy, for any constant $\epsilon' > 0$. Our solution chooses the better of these two policies, achieving an expected value of, say, m. Then $ADAPT \leq \left(\frac{4}{1-\sigma} + (1+\epsilon')\right) m \leq (5+\epsilon)m$, for appropriately chosen constants σ and ϵ'.

We now focus on computing a $(1+\epsilon)$-approximate policy for heavy items, for any constant $\epsilon > 0$. Let us think of an adaptive policy as a decision tree, where at a depth-d node we have a choice from among $|\mathcal{I}_H| - d$ items to insert next (and this decision will be made based on remaining capacity). We first claim that by restricting our policy to have at most a constant depth (depending on ϵ), we only forfeit a small (e.g., ϵ-fraction) amount of expected value. Hence, our decision tree will contain at most a polynomial number of nodes, and we can exhaustively compute an approximately-optimal adaptive policy for each node in a bottom-up fashion — at each depth-d node in the decision tree, our algorithm for determining the appropriate item to insert next will make a polynomial number of calls to algorithms at depth level $d+1$.

Let us define the function $f_{S,k}(c)$ to give the maximum expected value one can achieve if c units of capacity have already been used, we have already inserted items in the set S into the knapsack, and we may only insert k more items into the knapsack. Our approximation algorithm relies on the following crucial result:

Lemma 9. *For any constant $\epsilon > 0$, any value $c \in [0,1]$, any set of heavy items S and any $k = O(1)$, there exists a polynomial-time algorithm (which we call $A_{S,k,\epsilon}$) that computes an item in $\mathcal{I}_H \backslash S$ to insert into the knapsack that constitutes the beginning of an adaptive policy obtaining expected value in the range $[f_{S,k}(c)/(1+\epsilon), f_{S,k}(c)]$. The algorithm also approximates the value of this policy, returning a value $g_{S,k,\epsilon}(c) \in [f_{S,k}(c)/(1+\epsilon), f_{S,k}(c)]$.*

Consider the implications of this lemma (which we prove shortly). According to Lemma 2, $\mathbf{E}[\mu(S_H)] \leq 2$, where S_H denotes the (random) set of heavy items placed in the knapsack by an optimal adaptive policy. Since the expected size of every heavy item is at least σ, we have $\mathbf{E}[|S_H|] \leq 2/\sigma$ and by Markov's inequality, $\Pr[|S_H| \geq \frac{2}{\sigma^2}] \leq \sigma$. Therefore, letting $ADAPT_H$ denote the expected value from heavy items obtained by an optimal adaptive policy, and letting V_k denote the value obtained beyond the kth item, we see that

$$\begin{aligned}
ADAPT_H &= f_{\emptyset,n}(0) \\
&\leq f_{\emptyset,k}(0) + \mathbf{E}[V_k \mid |S_H| \geq k] \Pr[|S_H| \geq k] \\
&\leq f_{\emptyset,k}(0) + ADAPT_H \cdot \sigma \\
&\leq f_{\emptyset,k}(0)/(1-\sigma).
\end{aligned}$$

Since $k = \frac{2}{\sigma^2} = O(1)$, Lemma 9 applies and allows us to compute a $(1+\epsilon')$-approximation to the value of $f_{\emptyset,k}(0)$, as well as a corresponding policy, for any constant $\epsilon' > 0$. By selecting ϵ' and σ appropriately, we can approximate the optimal policy for heavy items to within a $(1+\epsilon)$ factor for any constant $\epsilon > 0$.

Proof of Lemma 9. We use induction on k, taking $k = 0$ as a trivial base case. Assume the lemma now holds for some value of k, so for every heavy item set S we have a polynomial-time algorithm $A_{S,k,\epsilon'}$, where ϵ' is such that $(1+\epsilon')^2 = 1+\epsilon$. We describe how to construct the algorithm $A_{S,k+1,\epsilon}$ for evaluating $g_{S,k+1,\epsilon}(c)$ by making only a polynomial number of "recursive" calls to $A_{\cdot,k,\epsilon'}$. Since we perform at most a constant number of levels of induction (recall the final value of k is constant), each of which makes only a polynomial number of function calls at the preceding level, the overall running time of $A_{S,k+1,\epsilon}$ will be polynomial (albeit with potentially very high degree), and all values of ϵ required by the recursive computations will also be constant.

The algorithm $A_{S,k+1,\epsilon}$ must decide which item in $\mathcal{I}_H \backslash S$ to insert next into the knapsack, given c units of occupied capacity, in order to launch a $(1+\epsilon)$-approximate policy. To do this, we approximate the expected value we get with each item $i \in \mathcal{I}_H \backslash S$ and take the best item. To estimate the expected value if we start with item i, we might try to use random sampling: sample a large number of instances of the size s_i of item i, and for each use the algorithm $A_{S\cup\{i\},k,\epsilon}$ to approximate the expected value $f_{S\cup\{i\},k}(c+s_i)$ obtained by the remainder of an optimal policy starting with i. However, this approach does not work due to the "rare event" problem often encountered with random sampling. If s_i has exponentially small probability of taking very small values for which $f_{S\cup\{i\},k}(c+s_i)$ is very large value, we will likely miss this contribution to the aggregate expected value. To remedy this, we approximate $f_{S\cup\{i\},k}(c)$ by a piecewise constant function $\hat{f}(c)$ with a polynomial number of breakpoints denoted $0 = c_0 \dots c_p = 1$. Initially we compute $\hat{f}(c_0) = g_{S\cup\{i\},k,\epsilon'}(0)$ by a single invocation of $A_{S\cup\{i\},k,\epsilon'}$. We then use binary search to compute each successive breakpoint $c_1 \dots c_{p-1}$: after computing c_{j-1} we set c_j to be the minimum value of c such that $g_{S\cup\{i\},k,\epsilon'}(c) < \hat{f}(c_{j-1})/(1+\epsilon')$.

The maximum number of steps required by the binary search will be polynomial in B and n, since any aggregate sum of n item sizes requires $poly(n, B)$ bits to represent, and each iteration of binary search fixes one of these bits. Each step of the binary search makes one call to $A_{S\cup\{i\},k,\epsilon'}$ to evaluate $g_{S\cup\{i\},k,\epsilon'}(c)$ for some value of c. Note that even though $f_{S\cup\{i\},k}(c)$ is a nonincreasing function of c, its approximation $g_{S\cup\{i\},k,\epsilon'}(c)$ may not be nonincreasing. However, if we ever evaluate $g_{S\cup\{i\},k,\epsilon'}(c)$ and notice its value is larger than $g_{S\cup\{i\},k,\epsilon'}(c')$, where $c' < c$ is a point at which we formerly evaluated the function (or similarly, if we notice that $g_{S\cup\{i\},k,\epsilon'}(c) < g_{S\cup\{i\},k,\epsilon'}(c')$ and $c' > c$), then it suffices to use $g_{S\cup\{i\},k,\epsilon'}(c')$ instead of $g_{S\cup\{i\},k,\epsilon'}(c)$ — this still yields a $(1+\epsilon')$-approximation to $f_{S\cup\{i\},k}(c)$, and allows us to binary search properly.

Each breakpoint of $\hat{f}$ marks a drop in $g_{S\cup\{i\},k,\epsilon'}$ by at least a $(1+\epsilon')$ factor. The value of g can drop by $(1+\epsilon')$ at most a polynomial number of times before it eventually reaches zero. We can argue this inductively, by adding to our inductive hypothesis the claim that g evaluates to a quantity represented by a polynomial number of bits. Since $\hat{f}$ is a $(1+\epsilon')$-approximation to $g_{S\cup\{i\},k,\epsilon'}$, which is in turn a $(1+\epsilon')$-approximation to $f_{S\cup\{i\},k}$, we know that $\hat{f}(c)$ lies in the range $[f_{S\cup\{i\},k}(c)/(1+\epsilon), f_{S\cup\{i\},k}(c)]$. Assuming i is the correct first item to insert into the knapsack, we can compute $g_{S,k+1,\epsilon}(c)$ as:

$$g_{S,k+1,\epsilon}(c) = v_i \Pr[s_i \leq 1-c]$$
$$+ \sum_{j=1}^{p} g_{S\cup\{i\},k,\epsilon'}(c + c_{j-1}) \Pr[s_i \in [c_{j-1}, c_j]]$$

Maximizing over all potential first items $i \in \mathcal{I}_H \backslash S$ gives the final value of $g_{S,k+1,\epsilon}(c)$, which is a $(1+\epsilon)$-approximation to $f_{S,k+1}(c)$. Note that any value of $g_{S,k+1,\epsilon}(c)$ must be representable by a polynomial number of bits as long as the same is true of $g_{S,k,\epsilon'}(c)$ (since we only carry the induction out for a constant number of levels). In total, the algorithm $A_{S,k+1,\epsilon}$ above makes only a polynomial number of calls to $A_{\cdot,k,\epsilon'}$. □

7. Complexity results

In this section, we state several complexity-theoretic results concerning the stochastic knapsack problem. Since classical knapsack is a special case of our problem (with all item sizes deterministic), finding the optimal solution is certainly NP-hard. However, it is interesting to ask whether the stochastic knapsack has a greater degree of complexity than NP. We prove that some natural questions concerning optimal adaptive policies are actually PSPACE-hard. Regarding the question of finding the maximum expected value achieved by an adaptive policy, we prove PSPACE-hardness only in the more general case where item values are also random (and actually equal to the respective item size). Several interesting questions remain, namely whether it is PSPACE-hard to maximize the expected value achieved by an adaptive policy in the setting with deterministic values, and also whether it is possible to prove hardness (even NP-hardness) of approximation for some constant factor > 1.

The proofs of PSPACE-hardness are based on the similarity between stochastic knapsack and Arthur-Merlin games [1], or "games against nature" [10], which are both equivalent to PSPACE. Executing an adaptive policy can be seen as a game where Merlin chooses the item to insert, while Arthur responds by selecting a random size for that item. Merlin is trying to prove that a certain kind of solution exists. Once this solution is found, Arthur accepts. (And we say that the policy *succeeds* in this case.) If the knapsack overflows, Arthur rejects. The probability of acceptance is exactly the probability of success of the optimal adaptive policy. We prove that some versions of the stochastic knapsack problem are in fact powerful enough to simulate any Arthur-Merlin game.

For all our results, we show a reduction from the following PSPACE-complete problem described in [4], Fact 4.1:

Problem: *MAX-PROB SSAT*

Input: Boolean 3-cnf formula Φ with variables $x_1, y_1, \ldots, x_k, y_k$.

Regard $\Phi(x_1, y_1, \ldots x_k, y_k)$ as a function from $\{0,1\}^{2k}$ to $\{0,1\}$ and define:

$$P(\Phi) = \mathcal{M}x_1 \mathcal{A}y_1 \ldots \mathcal{M}x_k \mathcal{A}y_k \Phi(x_1, y_1, \ldots x_k, y_k)$$

where $\mathcal{M}x\ f(x) = \max\{f(0), f(1)\}$
and $\mathcal{A}y\ g(y) = (g(0) + g(1))/2$.

Output:

- YES, if $P(\Phi) = 1$.
- NO, if $P(\Phi) \leq 1/2$.

This is a "promise problem", in the sense that any instance is guaranteed to have either $P(\Phi) = 1$ or $P(\Phi) \leq 1/2$. The formula can be seen as encoding a certain Arthur-Merlin game. We define a stochastic knapsack instance which models this Arthur-Merlin game. The reduction is inspired by the standard reduction from 3-SAT to Knapsack.

Theorem 3. *For a knapsack instance and a fixed ordering of its n items $\mathcal{O}$, let $\hat{p}_O$ denote the maximum over all adaptive policies, allowed to insert items only in the order $\mathcal{O}$, of the probability that the knapsack is filled exactly up to its capacity. Then it is PSPACE-hard to distinguish whether $\hat{p}_O = 1$ or $\hat{p}_O \leq 1/2$.*

Proof. We prove that there is a polynomial-time reduction from *MAX-PROB SSAT* to *Stochastic Knapsack* such that $\hat{p}_O = P(\Phi)$. Suppose that Φ has m clauses, each with at most 3 literals. We define a sequence of items corresponding to the pairs of variables (x_1, y_1), (x_2, y_2), etc. Selecting a particular item will correspond to a choice of value for x_i, and generating a random size will correspond to a choice of value for y_i. Item sizes will be random variables, whose values we write as numbers in base-b representation. We choose b as a constant large enough so that the digits can never overflow ($b = 16$ suffices for all the reductions in this section). The size representation consists of two "blocks": $[VARS \mid CLAUSES]$ where $VARS$ has k digits and $CLAUSES$ has m digits. For each $i \in \{1, 2, \ldots k\}$, we define two items, $I_i(0)$ and $I_i(1)$. Each has two possible sizes, occurring with probability $1/2$, $size(I_i(x_i), 0)$ or $size(I_i(x_i), 1)$:

$$size(I_i(x_i), y_i) = [VARS(i) \mid CLAUSES(i, x_i, y_i)]$$

where $VARS(i)$ has a 1 in the i-th digit and zeros otherwise, and $CLAUSES(i, a, b)$ has 1's marking the clauses satisfied by setting $x_i = a$, $y_i = b$. We also define two "fill-in" items $F_j(0), F_j(1)$ for each clause j, with

$$size(F_j(x)) = [0 \mid CLAUSE(j)],$$

where $CLAUSE(j)$ has a 1 in the j-th digit and zeros otherwise. The fixed ordering $\mathcal{O}$ is

$$(I_1(0), I_1(1), I_2(0), I_2(1), \ldots, F_1(0), F_1(1), \ldots)$$

and we set the knapsack capacity to

$$C = [1111111111 \mid 33333333333333333333].$$

We leave it to the reader to verify that the maximum probability that a policy fills the knapsack exactly up to its capacity is $\hat{p}_O = P(\Phi)$. This follows from the correspondence between adaptive policies and strategies to set the x_i variables, while the y_i variables are set randomly. At the end, an optimal policy uses the fill-in items F_j to increase each digit in the block of $CLAUSES$ to 3, which is possible if and only if each digit is at least 1 after the variable items I_i have been inserted (i.e., each clause has been satisfied).

□

Theorem 4. *For a knapsack instance, let $\hat{p}$ be the maximum probability that an adaptive policy fills the knapsack exactly to its capacity. Then it is PSPACE-hard to distinguish whether $\hat{p} = 1$ or $\hat{p} \leq 3/4$.*

Proof. We already know that the problem is PSPACE-hard if a fixed ordering is imposed on the items. We show how to remove this restriction here. Consider the knapsack instance generated in the previous proof, which has probability of success either $\hat{p}_O = 1$ or $\hat{p}_O \leq 1/2$. We extend the item sizes by a new block called $FLAGS$ with k digits which will enforce that the optimal policy must consider the items $I_1, I_2, I_3, \ldots$ in this order. The fill-in items F_j receive a zero $FLAGS$ block - the adaptive policy cannot gain anything by inserting them earlier anyway. $FLAGS$ will have a digit for each index i and each item $I_i(x)$ will be modified in this way: we define two new items for each item, parametrized by $f_i = 0, 1$, and two new random sizes for each previously possible size, parametrized by $r_i = 0, 1$. The new item sizes will be $size(I_i(x_i, f_i), y_i, r_i)) =$

$$[VARS(i) \mid FLAGS(i, f_i, r_i) \mid CLAUSES(i, x_i, y_i)]$$

where $VARS$ and $CLAUSES$ are defined as before, and

$$FLAGS(i, f_i, r_i) = [0000000 r_i f_i 000000]$$

where f_i is the i-th least significant digit and r_i is the $(i+1)$-th least significant digit. (The items for $i = k$ don't have any r_i digit.) Every $I_i(x_i, f_i)$ item can therefore take 4 different values. Finally, we extend the knapsack capacity to

$$C = [1111111111 \mid 1111111111 \mid 333333333333333].$$

Consider a YES instance, in which case $\hat{p}_O = 1$. Then an optimal policy which inserts items in the standard order $I_1, I_2, I_3, \ldots$, can always choose a suitable value of f_{i+1} to ensure $r_i + f_{i+1} = 1$, which achieves the target value for $FLAGS$. Therefore an optimal policy succeeds with probability $\hat{p} = \hat{p}_O = 1$.

Now consider a NO instance, where $\hat{p}_O \leq 1/2$. A policy may ignore the standard ordering and try to insert an item $I_i(x_i, f_i)$ *after* $I_{i+1}(x_{i+1}, f_{i+1})$. Let's say that "the policy cheats" if this happens (which is a random event in general). In such a case, since the digit r_i is generated randomly, and the only other non-zero entry in this position is f_{i+1}, which has been chosen already, the final entry in this position will be 1 only with probability $1/2$. Therefore in the event that a policy cheats, it can succeed with conditional probability at most $1/2$. Suppose that the optimal ordered policy $\mathcal{P}_O$ succeeds with probability $\hat{p}_O \leq 1/2$. Then the cheating policy can achieve maximum probability of success if it cheats exactly when $\mathcal{P}_O$ would fail (with prob. $1 - \hat{p}_O$), and the total probability of success is $\hat{p} \leq \hat{p}_O + (1 - \hat{p}_O)/2 \leq 3/4$. □

Theorem 5. *For a more general knapsack problem, where item values are also random and possibly correlated with the respective item sizes, let $\hat{p}(V)$ be the maximum probability that an adaptive policy inserts successfully a set of items of total value at least V. Similarly, define $\hat{p}_O(V)$ for ordered policies with a given ordering of items $\mathcal{O}$. Then it is PSPACE-hard to distinguish whether $\hat{p}(V) = 1$ or $\hat{p}(V) \leq 3/4$, and PSPACE-hard to distinguish whether $\hat{p}_O(V) = 1$ or $\hat{p}_O(V) \leq 1/2$, for any fixed ordering of items.*

Proof. In the setting with random values, we can define each to be equal to the size of the respective item. Then the maximum value any policy can achieve is equal to the capacity, and the maximum probability that this value is attained is $\hat{p}(C) = \hat{p}$ (from Theorem 4), or $\hat{p}_O(C) = \hat{p}_O$ (from Theorem 3). Consequently, deciding between $\hat{p} = 1$ and $\hat{p} \leq 3/4$ is PSPACE-hard, as well as deciding between $\hat{p}_O = 1$ and $\hat{p}_O \leq 1/2$ for any fixed ordering of items. □

Note. The threshold values are not important. The success probabilities for NO instances can be made exponentially small by standard methods, here as well as in the preceding theorems. We omit the details.

Theorem 6. *For a knapsack instance with random values, possibly correlated with item sizes, it is PSPACE-hard to maximize the expected value achieved by any adaptive policy, or by any adaptive policy with a fixed ordering of items.*

Proof. We use the reduction from the proof of Theorem 5. In case of a YES instance, the expected value achieved by the optimal adaptive policy is C, because C is achieved with probability 1, and it is the maximum possible value for any policy. In case of a NO instance, the probability of achieving C is less than $3/4$; with probability at least $1/4$, the policy achieves at most value $C - 1$. Thus the expected value in this case cannot be larger than $C - 1/4$. The same holds (with $C - 1/2$ instead), if we restrict ourselves to policies using the standard ordering of items. □

References

[1] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *J. Computer and System Sciences*, 36(2):254–276, 1988.

[2] J. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, 1997.

[3] R. Carraway, R.L.Schmidt, and L. Weatherford. An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns. *Naval Research Logistics*, 40:161–173, 1993.

[4] A. Condon, J. Feigenbaum, C. Lund, and P. Shor. Random debaters and the hardness of approximating stochastic functions. *SIAM J. Computing*, 26:369–400, 1997.

[5] C. Derman, C. Lieberman, and S. Ross. A renewal decision problem. *Management Science*, 24(5):554–561, 1978.

[6] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *FOCS*, pages 579–586, 1999.

[7] M. Henig. Risk criteria in a stochastic knapsack problem. *Operations Research*, 38(5):820–825, 1990.

[8] N. Immorlica, D. Karger, M. Minkoff, and V. Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *SODA*, pages 184–693, 2004.

[9] J. Kleinberg, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. In *STOC*, pages 664–673, 1997.

[10] C. Papadimitriou. Games against nature. *J. of Computer and System Sciences*, 31:288–301, 1985.

[11] M. Rothkopf. Scheduling with random service times. *Management Science*, 12(9):707–713, 1966.

[12] M. Sniedovich. Preference order stochastic knapsack problems: methodological issues. *The Journal of the Operational Research Society*, 31(11):1025–1032, 1980.

[13] E. Steinberg and M. Parks. A preference order dynamic program for a knapsack problem with stochastic rewards. *The Journal of the Operational Research Society*, 30(2):141–147, 1979.

[14] M. Uetz. *Algorithms for deterministic and stochastic scheduling*. PhD thesis, Institut für Mathematik, Technische Universität Berlin, 2001.