MATHEMATICS OF OPERATIONS RESEARCH
Vol. 33, No. 4, November 2008, pp. 945–964
ISSN 0364-765X | EISSN 1526-5471 | 08 | 3304 | 0945

DOI 10.1287/moor.1080.0330
© 2008 INFORMS

Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity

Brian C. Dean
School of Computing, McAdams Hall, Clemson, South Carolina 29634,
bcdean@cs.clemson.edu

Michel X. Goemans
Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139,
goemans@math.mit.edu

Jan Vondrák
Department of Mathematics, Princeton, New Jersey 08544,
jvondrak@gmail.com

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 successfully placed in the knapsack, where the final overflowing item contributes no value. We consider both *nonadaptive* 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). An important facet of our work lies in characterizing the benefit of adaptivity. For this purpose we advocate the use of a measure called the *adaptivity gap*: the ratio of the expected value obtained by an optimal adaptive policy to that obtained by an optimal nonadaptive policy. We bound the adaptivity gap of the stochastic knapsack problem by demonstrating a polynomial-time algorithm that computes a nonadaptive policy whose expected value approximates that of an optimal adaptive policy to within a factor of four. We also devise a polynomial-time adaptive policy that approximates the optimal adaptive policy to within a factor of $3+\varepsilon$ for any constant $\varepsilon > 0$.

Key words: knapsack problem; stochastic models; approximation algorithm; adaptivity
MSC2000 subject classification: Primary: 90C27, 68W25; secondary: 90C59, 90C15
OR/MS subject classification: Primary: analysis of algorithms, programming; secondary: suboptimal algorithms, integer programming
History: Received May 23, 2005; revised July 1, 2007, and March 27, 2008. Published online in *Articles in Advance* November 3, 2008.

1. Introduction.

The classical NP-hard knapsack problem takes as input a set of n items with values $v_1, \ldots, v_n$ and sizes $s_1, \ldots, s_n$, and asks us to compute a maximum-value subset of these items whose total size is at most one. Among the many applications of this problem, we find the following common scheduling problem: given a set of n jobs, each with a known value and duration, compute a maximum-value subset of jobs one can schedule by a fixed deadline on a single machine. In practice, it is often the case that the duration of a job is not known precisely until after the job is completed; beforehand, it is known only in terms of a probability distribution. This motivates us to consider a stochastic variant of the knapsack problem in which item values are deterministic and 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 successfully placed in the knapsack, we seek to design a solution "policy" for sequentially inserting items until the capacity is eventually exceeded. At the moment when the capacity overflows, the policy terminates.

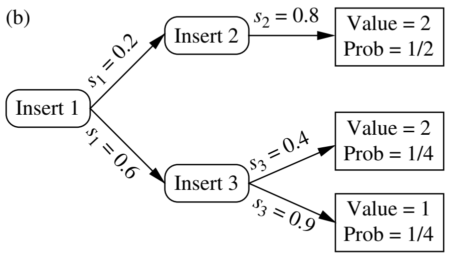
Formally, if $[n] := \{1, 2, \ldots, n\}$ indexes a set of n items, then a solution policy is a mapping $2^{[n]} \times [0,1] \to [n]$ specifying the next item to insert into the knapsack given the set of remaining (uninstantiated) available items as well as the remaining capacity in the knapsack. We typically represent a solution policy in terms of an algorithm that implements this mapping, and we can visualize such an algorithm in terms of a decision tree, as shown in Figure 1. As illustrated by the instances shown in the figure, an optimal policy may need to be *adaptive*, making decisions in a dynamic fashion in reaction to the instantiated sizes of items already placed in the knapsack. By contrast, a *nonadaptive* policy specifies an entire solution in advance, making no further decisions as items are being inserted. In other words, a nonadaptive policy is just a fixed ordering of the items to insert into the knapsack. It is at least NP-hard to compute optimal adaptive and nonadaptive policies for the stochastic knapsack problem, because both of these problems reduce to the classical knapsack problem in the deterministic case.

There are many problems in stochastic combinatorial optimization for which one could consider designing either adaptive or nonadaptive solution policies. In particular, these are problems in which a solution is incrementally constructed via a series of decisions, each of which establishes a small part of the total solution and

(a)

Item	Size distribution (capacity = 1)
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)



(c)

Item	Value	Size distribution (capacity = 1)
1	ε	0 (prob 1/2) 1 (prob 1/2)
2	ε	1 (prob 1)
3	1	0 (prob ε) $1+\varepsilon$ (prob $1-\varepsilon$)

FIGURE 1. Instances of the stochastic knapsack problem.
Notes. For (a), an optimal nonadaptive policy inserts items in the order 1, 2, 3, and achieves expected value 1.5. An optimal adaptive policy, shown as a decision tree in (b), achieves an expected value of 1.75, for an adaptivity gap of 7/6. The instance in (c) has an adaptivity gap arbitrarily close to 5/4: An optimal nonadaptive policy inserts items in the order 1, 3, 2 for an expected value of $2\varepsilon + \frac{1}{2}\varepsilon^2$, and an optimal adaptive policy inserts item 1 followed by items 2 and 3 (if $s_1 = 0$) or item 3 (if $s_1 = 1$), for an expected value of 2.5ε.

also results in the instantiation of a small part of the problem instance. When trying to solve such a problem, it is often a more complicated undertaking to design a good adaptive policy, but this might give us substantially better performance than a nonadaptive policy. To quantify the benefit we gain from adaptivity, we advocate the use of a measure we call the *adaptivity gap*, which measures the maximum (i.e., worst-case) ratio over all instances of a problem of the expected value obtained by an optimal adaptive policy to the expected value obtained by an optimal nonadaptive policy. One of the main results in this paper is a proof that the adaptivity gap of the stochastic knapsack problem is at most four, so we only lose a small constant factor by considering nonadaptive policies. Adaptivity gap plays a similar role to the integrality gap of a fractional relaxation by telling us the best approximation bound we can hope to achieve by considering a particular simple class of solutions. Also, like the integrality gap, one can study the adaptivity gap of a problem independently of any considerations of algorithmic efficiency.

1.1. Outline of results. In this paper we provide both nonadaptive and adaptive approximation algorithms for the stochastic knapsack problem. After giving definitions and preliminary results in §2, we present three main approximation algorithm results in the following sections. Section 3 describes how we can use a simple linear programming relaxation to bound the expected value obtained by an optimal adaptive policy, and we use this bound in §4 to develop a 32/7-approximate nonadaptive policy. We then develop a more sophisticated linear programming bound based on a polymatroid in §5, and use it in §6 to construct a 4-approximate nonadaptive policy. Sections 7 and 8 then describe a $(3+\varepsilon)$-approximate adaptive policy for any constant $\varepsilon > 0$ (the policy takes polynomial time to make each of its decisions).

In §9 we consider what we call the *ordered adaptive* model. Here, the items must be considered in the order they are presented in the input, and for each item we can insert it into the knapsack or irrevocably discard it (and this decision can be adaptive, depending on the instantiated sizes of items already placed in the knapsack). This model is of interest because we can compute optimal ordered policies in pseudopolynomial time using dynamic programming in the event that item-size distributions are discrete, just as the deterministic knapsack problem is commonly approached with dynamic programming if item sizes are discrete. A natural and potentially difficult question with this model is how one should choose the initial ordering of the items. If we start with the ordering produced by our 4-approximate nonadaptive policy, the optimal ordered adaptive policy will have an expected value within a factor of four of the optimal adaptive policy (and it can potentially be much closer). We show in §9 that for *any* initial ordering of items, the optimal ordered adaptive policy will obtain an expected value that differs by at most a factor of 9.5 from that of an optimal adaptive policy.

1.2. Literature review. The stochastic knapsack problem is perhaps best characterized as a scheduling problem, where it could be written as $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \bar{U}_j]$ using the three-field scheduling notation popularized by Lawler et al. [10]. Stochastic scheduling problems, where job durations are random variables with known probability distributions, have been studied quite extensively in the literature, dating back as far as 1966 (Rothkopf [20]). However, for the objective of scheduling a maximum-value collection of jobs prior to a fixed deadline, all previous research seems to be devoted to characterizing which classes of probability distributions allow an exact optimal solution to be computed in polynomial time. For example, if the sizes of items are exponentially distributed, then Derman et al. [6] prove that the greedy nonadaptive policy that chooses items in nonincreasing order of $v_i/\mathbf{E}[s_i]$ is optimal. For extensions and further related results, see also

Pinedo [18], Emmons and Pinedo [7], and Chang et al. [3]. To the best of our knowledge, the only stochastic scheduling results to date that consider arbitrary probability distributions and the viewpoint of approximation algorithms tend to focus on different objectives, particularly minimizing the sum of weighted completion times (see Möhring et al. [16], Skutella and Uetz [23], Uetz [26]).

The notion of adaptivity is quite prevalent in the stochastic scheduling literature and also in the stochastic programming literature (see, e.g., Birge and Louveaux [1]) in general. One of the most popular models for stochastic optimization is a two-stage model in which one commits to a partial solution and then, after witnessing the instantiation of all random quantities present in the instance, computes an appropriate *recourse* as necessary to complete the solution. For example, a stochastic version of bin packing in this model would ask us to pack a collection of randomly sized items into the minimum possible number of unit-sized bins. In the first stage, we can purchase bins at a discounted price, after which we observe the instantiated sizes of all items and then, as a recourse, purchase additional bins that are needed (at a higher price). For a survey of approximation results in the two-stage model, see Immorlica et al. [12], Ravi and Sinha [19], Shmoys and Swamy [22]. By contrast, our model for the stochastic knapsack problem involves unlimited levels of recourse. The notion of adaptivity gap does not seem to have received any explicit attention thus far in the literature. Note that the adaptivity gap only applies to problems for which nonadaptive solutions make sense. Quite a few stochastic optimization problems, such as the two-stage bin-packing example above, are inherently adaptive because one must react to instantiated information in order to ensure the feasibility of the final solution.

Several stochastic variants of the knapsack problem different from ours have been studied in the literature. Stochastic knapsack problems with deterministic sizes and random values have been studied by several authors—Carraway et al. [2], Henig [11], Sniedovich [24], and Steinberg and Parks [25]—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 because 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. Another somewhat related variant, known as the stochastic and dynamic knapsack problem (Kleywegt and Papastavrou [14], Papastavrou et al. [17]), involves items that arrive online according to some stochastic process—we do not know the exact characteristics of an item until it arrives, at which point in time we must irrevocably decide to either accept the item and process it, or discard the item. Two recent papers due to Kleinberg et al. [13] and Goel and Indyk [9] consider a stochastic knapsack problem with "chance" constraints. Like our model, they consider items with deterministic values and random sizes. However, their objective is to find a maximum-value set of items whose probability of overflowing the knapsack is at most some specified value p. 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 pseudoapproximation results. For item sizes that have Poisson or exponential distributions, Goel and Indyk provide a polynomial-time approximation scheme (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 #P-hard to solve the variant mentioned above with deterministic sizes and random values, where the goal is to find a set of items whose probability of exceeding some target value is maximized. To see this, let $Be(s, p)$ denote the Bernoulli distribution taking the value s with probability p and 0 with probability $1-p$, and consider a set of items $i = 1 \dots n$ with size distributions $Be(s_i, p_i)$. To compute the overflow probability p^* of this set of items in a knapsack of capacity 1, we construct an instance of the stochastic knapsack problem above with target value 1 and $n+1$ items: items $i = 1 \dots n$ have size $1/n$ and value $Be(s_i, p_i)$, and the last has size 1 and value $Be(1, p')$. The optimal solution will contain the first n items if $p^* > p'$ or the single last item otherwise, so we can compute p^* using a binary search on p'. Note that it is also #P-hard to solve the restricted problem variant with random sizes and deterministic values, where the goal is to find a set of maximum value given a bound on the overflow probability. Here, we can take a set of items $i = 1 \dots n$ with sizes $Be(s_i, p_i)$ and compute their overflow probability p^* by constructing a stochastic knapsack instance with overflow probability threshold p' and with n items of size $Be(s_i, p_i)$ and unit value. Because the optimal solution to this problem will be the set of all items (total value n) if and only if $p^* \le p'$, we can again compute p^* using a binary search on p'.

In contrast to the results cited above, we do not assume that the distributions of item sizes are exponential, Poisson, or Bernoulli; our algorithms work for arbitrary distributions. The results in this paper substantially improve upon the results given in its extended abstract (Dean et al. [5]), where approximation bounds of 7 and

$5+\varepsilon$ are shown for nonadaptive and adaptive policies. Note that a large part of the analysis in Dean et al. [5] (see also Dean [4]) is no longer needed in this paper, but the different methods used in that analysis may also be of interest to the reader.

2. Preliminaries.

2.1. Definition of the problem. An instance I consists of a collection of n items characterized by *size* and *value*. For each item $i \in [n]$, let $v_i \geq 0$ denote its value and $s_i \geq 0$ its size. We assume that the v_is are deterministic, whereas the s_is are independent random variables with known, arbitrary distributions. Because our objective is to maximize the expected value of items placed in the knapsack, we can allow random v_is as long as they are mutually independent and also independent from the s_is. In this case, we can simply replace each random v_i with its expectation. In the following, we consider only deterministic values v_i. Also, we assume without loss of generality that our instance is scaled so that the capacity of the knapsack is one.

Definition 2.1 (Adaptive Policies). An adaptive policy is a function $\mathcal{P}: 2^{[n]} \times [0,1] \rightarrow [n]$. The interpretation of $\mathcal{P}$ is that given a set of available items J and remaining capacity c, the policy inserts item $\mathcal{P}(J,c)$. This procedure is repeated until the knapsack overflows. We denote by $\mathrm{val}(\mathcal{P})$ the value obtained for all successfully inserted items, which is a random variable resulting from the random process of executing the policy $\mathcal{P}$. We denote by $ADAPT(I) = \max_{\mathcal{P}} \mathbf{E}[\mathrm{val}(\mathcal{P})]$ the optimum[1] expected value obtained by an adaptive policy for instance I.

Definition 2.2 (Nonadaptive Policies). A nonadaptive policy is an ordering of items $\mathcal{O} = (i_1, i_2, i_3, \ldots, i_n)$. We denote by $\mathrm{val}(\mathcal{O})$ the value obtained for successfully inserted items, when inserted in this order. We denote by $NONADAPT(I) = \max_{\mathcal{O}} \mathbf{E}[\mathrm{val}(\mathcal{O})]$ the optimum expected value obtained by a nonadaptive policy for instance I.

In other words, a nonadaptive policy is a special case of an adaptive policy $\mathcal{P}(J,c)$ that does not depend on the residual capacity c. Thus, we always have $ADAPT(I) \geq NONADAPT(I)$. Our main interest in this paper is in the relationship of these two quantities. We investigate how much benefit a policy can gain by being adaptive, i.e., how large $ADAPT(I)$ can be compared to $NONADAPT(I)$.

Definition 2.3 (Adaptivity Gap). We define the adaptivity gap as

$$\sup_I \frac{ADAPT(I)}{NONADAPT(I)},$$

where the supremum is taken over all instances of stochastic knapsack.

This concept extends naturally beyond the stochastic knapsack problem. It seems natural to study the adaptivity gap for any class of stochastic optimization problems where adaptivity is present (Dean [4], Vondrák [27]).

It should be noted that in the definition of the adaptivity gap, there is no reference to computational efficiency. The quantities $ADAPT(I)$ and $NONADAPT(I)$ are defined in terms of all policies that exist, but it is another question whether an optimal policy can be found algorithmically. Observe that an optimal adaptive policy might be quite complicated in structure; for example, it is not even clear that one can always write down such a policy using polynomial space.

Example 1. 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 decision tree corresponding to an optimal policy would have at least $\binom{n}{n/2}$ leaves.

Because finding the optimal solution for the deterministic knapsack problem is NP-hard, and some questions concerning adaptive policies for stochastic knapsack are even PSPACE-hard (Dean et al. [5], Vondrák [27]), constructing or characterizing an optimal adaptive policy seems very difficult. We seek to design *approximation algorithms* for this problem. We measure the quality of an algorithm by comparing its expected value against $ADAPT$. That is, if $A(I)$ denotes the expected value of our algorithm A on instance I, we say that the performance guarantee of A is

$$\sup_I \frac{ADAPT(I)}{A(I)}.$$

[1] One can show that the supremum implicit in the definition of $ADAPT(I)$ is attained.

This measurement of quality differs greatly from the measure we would get from traditional competitive analysis of online algorithms or its relatives (e.g., Koutsoupias and Papadimitriou [15]). Competitive analysis would have us compare the performance of A against an unrealistically powerful optimal algorithm that knows in advance the instantiated sizes of all items, so it only allows us to derive very weak guarantees. For example, let $Be(p)$ denote the Bernoulli probability distribution with parameter p (taking the value zero with probability $1-p$ and one with probability p). In an instance with n items, each of size $(1+\varepsilon)Be(1/2)$ and value one, we have $ADAPT \le 1$, whereas if we know the instantiated item sizes in advance we could achieve an expected value of at least $n/2$ (because we expect $n/2$ item sizes to instantiate to zero).

For similar reasons, we assume that the decision to insert an item into the knapsack is irrevocable—in the scheduling interpretation of our problem, we might want the ability to cancel a job after scheduling it but then realizing after some time that, conditioned on the time it has already been processing, its remaining processing time is likely to be unreasonably large. The same example instance above shows that if cancellation is allowed, our policies *must* somehow take advantage of this fact, or else we can only hope to obtain an $O(1/n)$ fraction of the expected value of an optimal schedule in the worst case. We do not consider the variant of stochastic knapsack with cancellation in this paper.

2.2. Additional definitions. In the following, we use the following quantity defined for each item:

Definition 2.4. The mean truncated size of an item i is

$$\mu_i = \mathbf{E}[\min\{s_i, 1\}].$$

For a set of items S, we define $\text{val}(S)=\sum_{i\in S} v_i$, $size(S)=\sum_{i\in S} s_i$, and $\mu(S)=\sum_{i\in S}\mu_i$. We refer to $\mu(S)$ as the "mass" of set S.

One motivation for this definition is that $\mu(S)$ provides a natural bound on the probability that a set of items overflows the knapsack capacity.

Lemma 2.1. $\Pr[size(S) < 1] \ge 1-\mu(S)$.

Proof. $\Pr[size(S)\ge 1] = \Pr[\min\{size(S),1\}\ge 1] \le \mathbf{E}[\min\{size(S),1\}] \le \mathbf{E}[\sum_{i\in S}\min\{s_i,1\}] = \mu(S)$. □

The mean truncated size is a more useful quantity than $\mathbf{E}[s_i]$ because it is not sensitive to the structure of s_is distribution in the range $s_i > 1$. In the event that $s_i > 1$, the actual value of s_i is not particularly relevant because item i will definitely overflow the knapsack (and therefore contribute no value towards our objective). All of our nonadaptive approximation algorithms look only at the mean truncated size μ_i and the probability of fitting in an empty knapsack $\Pr[s_i \le 1]$ for each item i; no other information about the size distributions is used. However, the $(3+\varepsilon)$-approximate adaptive policy we develop in §8 is assumed to know the complete size distributions, just like all adaptive policies in general.

3. Bounding adaptive policies. In this section we address the question of how much expected value an adaptive policy can possibly achieve. We show that despite the potential complexity inherent in an optimal adaptive policy, a simple linear programming (LP) relaxation can be used to obtain a good bound on its expected value.

Let us fix an adaptive policy $\mathcal{P}$ and consider the random set of items A that $\mathcal{P}$ inserts successfully into the knapsack. In a deterministic setting, it is clear that $\mu(A)\le 1$ for any policy because our capacity is one. It is perhaps surprising that in the stochastic case, $\mathbf{E}[\mu(A)]$ can be larger than one.

Example 2. Suppose we have an infinitely large (random) set of items where each item i has value $v_i = 1$ and size $s_i \sim Be(p)$. In this case, $\mathbf{E}[|A|] = 2/p - 1$ (because we can insert items until the point where two of them instantiate to unit size) and each item i has mean truncated size $\mu_i = p$, so $\mathbf{E}[\mu(A)] = 2-p$. For small $p>0$, this quantity can be arbitrarily close to two. If we count the first overflowing item as well, we insert mass of exactly two. This is a tight example for the following crucial lemma.

Lemma 3.1. *For any stochastic knapsack instance with capacity one and any adaptive policy, let A denote the* (*random*) *set of items that the policy attempts to insert. Then,* $\mathbf{E}[\mu(A)]\le 2$.

Proof. Consider any adaptive policy and denote by A_t the (random) set of the first t items that it attempts to insert. (We set $A_0=\varnothing$.) Eventually, the policy terminates, either by overflowing the knapsack or by exhausting all the available items. If either event happens upon inserting t' items, we set $A_t = A_{t'}$ for all $t > t'$; note that $A_{t'}$ still contains the first overflowing item. Because the process always terminates like this, we have

$$\mathbf{E}[\mu(A)] = \lim_{t\to\infty}\mathbf{E}[\mu(A_t)] = \sup_{t\ge 0}\mathbf{E}[\mu(A_t)].$$

Denote by $\tilde{s}_i = \min\{s_i, 1\}$ the mean truncated size of item i. Observe $\sum_{i\in A_t} \tilde{s}_i \le 2$ for all $t \ge 0$. This is because each $\tilde{s}_i$ is bounded by one and we can count at most one item beyond the capacity of one. We now define a sequence of random variables $\{X_t\}_{t\in\mathbb{Z}_+}$:

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

This sequence $\{X_t\}$ is a *martingale*: Conditioned on a value of X_t and the next inserted item i^*,

$$\mathbf{E}[X_{t+1} \mid X_t, i^*] = X_t + \mathbf{E}[\tilde{s}_{i^*}] - \mu_{i^*} = X_t.$$

If no more items are inserted, $X_{t+1} = X_t$ trivially. We can therefore remove the conditioning on i^*, so $\mathbf{E}[X_{t+1} \mid X_t] = X_t$, and this is the definition of a martingale. We now use the well-known martingale property that $\mathbf{E}[X_t] = \mathbf{E}[X_0]$ for any $t \ge 0$. In our case, $\mathbf{E}[X_t] = \mathbf{E}[X_0] = 0$ for any $t \ge 0$. As we mentioned, $\sum_{i\in A_t} \tilde{s}_i$ is always bounded by two, so $X_t \le 2 - \mu(A_t)$. Taking the expectation, $0 = \mathbf{E}[X_t] \le 2 - \mathbf{E}[\mu(A_t)]$. Thus $\mathbf{E}[\mu(A)] = \sup_{t\ge 0} \mathbf{E}[\mu(A_t)] \le 2$. □

We now show how to bound the value of an optimal adaptive policy using a linear program. We define by $w_i = v_i \cdot \Pr[s_i \le 1]$ the *effective value* of item i, which is an upper bound on the expected value a policy can gain if it attempts to insert item i. Consider now the linear programming relaxation for a knapsack problem with item values w_i and item sizes μ_i, parameterized by the knapsack capacity t:

$$\Phi(t) = \max\left\{\sum_i w_i x_i\colon \sum_i \mu_i x_i \le t,\ x_i \in [0, 1]\right\}.$$

Note that we use w_i instead of v_i in the objective for the same reason that we cannot just use v_i in the deterministic case. If we have a deterministic instance with a single item whose size is larger than one, then we cannot use this item in an integral solution but we can use part of it in a fractional solution, giving us an unbounded integrality gap. To fix this issue, we need to appropriately discount the objective value we can obtain from such large items, which leads us to the use of w_i in the place of v_i. Using the linear program above, we now arrive at the following bound.

Theorem 3.1. *For any instance of the stochastic knapsack problem, ADAPT* $\le \Phi(2)$.

Proof. Consider any adaptive policy $\mathscr{P}$, and as above let A denote the (random) set of items that $\mathscr{P}$ attempts to insert into the knapsack. Consider the vector $\vec{x}$ where $x_i = \Pr[i \in A]$. The expected mass that $\mathscr{P}$ attempts to insert is $\mathbf{E}[\mu(A)] = \sum_i \mu_i x_i$. We know from Lemma 3.1 that this is bounded by $\mathbf{E}[\mu(A)] \le 2$, therefore $\vec{x}$ is a feasible vector and $\sum_i w_i x_i \le \Phi(2)$.

Let $\mathrm{fit}(i, c)$ denote the indicator variable of the event that $s_i \le c$. Let c_i denote the capacity remaining when $\mathscr{P}$ attempts to insert item i. This is a random variable well defined if $i \in A$. The expected profit for item i is

$$\mathbf{E}[v_i \,\mathrm{fit}(i, c_i) \mid i \in A] \cdot \Pr[i \in A] \le \mathbf{E}[v_i \,\mathrm{fit}(i, 1) \mid i \in A] \cdot \Pr[i \in A] = v_i \cdot \Pr[s_i \le 1] \cdot \Pr[i \in A] = w_i x_i$$

because s_i is independent of the event $i \in A$. Therefore, $\mathbf{E}[\mathrm{val}(\mathscr{P})] \le \sum_i w_i x_i \le \Phi(2)$. The expected value obtained by any adaptive policy is bounded in this way, and therefore *ADAPT* $\le \Phi(2)$. □

As we show in the following, this linear program provides a good upper bound on the adaptive optimum, in the sense that it can differ from *ADAPT* at most by a constant factor. The following example shows that this gap can be close to a factor of four, which imposes a limitation on the approximation factor we can possibly obtain using this LP.

Example 3. Using only Theorem 3.1 to bound the performance of an optimal adaptive policy, we cannot hope to achieve any worst-case approximation bound better than four, even with an adaptive policy. Consider items of deterministic size $(1+\varepsilon)/2$ for a small $\varepsilon > 0$. Fractionally, we can pack almost four items within capacity 2, so that $\Phi(2) = 4/(1+\varepsilon)$, whereas only one item can actually fit.

The best approximation bound we can prove using Theorem 3.1 is a bound of $32/7 \approx 4.57$, for a nonadaptive policy presented in the next section. We show that this is tight in a certain sense. Later, in §5, we develop a stronger bound on *ADAPT* that leads to improved approximation bounds.

4. A 32/7-approximation for stochastic knapsack. In this section we develop a randomized algorithm whose output is a nonadaptive policy obtaining expected value of at least $(7/32)\mathit{ADAPT}$. Furthermore, this algorithm can be easily derandomized.

Consider the function $\Phi(t)$, which can be seen as the fractional solution of a knapsack problem with capacity t. This function is easy to describe. Its value is achieved by greedily packing items of maximum possible "value density" and taking a suitable fraction of the overflowing item. Assume that the items are already indexed by decreasing value density:

$$\frac{w_1}{\mu_1} \geq \frac{w_2}{\mu_2} \geq \frac{w_3}{\mu_3} \geq \cdots \geq \frac{w_n}{\mu_n}.$$

We call this the *greedy ordering*. Note that simply inserting items in this order is not sufficient, even in the deterministic case. For instance, consider $s_1 = \varepsilon$, $v_1 = w_1 = 2\varepsilon$ and $s_2 = v_2 = w_2 = 1$. The naive algorithm would insert only the first item of value 2ε, whereas the optimum is 1. Thus, we have to be more careful. Essentially, we use the greedy ordering, but first we insert a random item to prevent the phenomenon we just mentioned.

Let $M_k = \sum_{i=1}^{k} \mu_i$. Then for $t = M_{k-1} + \xi \in [M_{k-1}, M_k]$, we have

$$\Phi(t) = \sum_{i=1}^{k-1} w_i + \frac{w_k}{\mu_k}\xi.$$

Assume without loss of generality that $\Phi(1) = 1$. This can be arranged by scaling all item values by an appropriate factor. We also assume that there are sufficiently many items so that $\sum_{i=1}^{n} \mu_i \geq 1$, which can be arranged by adding dummy items of value zero. Now we are ready to describe our algorithm.

Let r be the minimum index such that $\sum_{i=1}^{r} \mu_i \geq 1$. Denote $\mu'_r = 1 - \sum_{i=1}^{r-1} \mu_i$, i.e., the part of μ_r that fits within capacity one. Set $p' = \mu'_r/\mu_r$ and $w'_r = p' w_r$. For $j = 1, 2, \ldots, r-1$, set $w'_j = w_j$ and $\mu'_r = \mu_r$. We assume $\Phi(1) = \sum_{i=1}^{r} w'_i = 1$.

The randomized greedy algorithm.
- Choose index k with probability w'_k.
- If $k < r$, insert item k. If $k = r$, flip another independent coin and insert item r with probability p' (otherwise discard it).
- Then insert items $1, 2, \ldots, k-1, k+1, \ldots, r$ in the greedy order.

Theorem 4.1. *The randomized greedy algorithm achieves expected value* $\mathit{RNDGREEDY} \geq (7/32)\mathit{ADAPT}$.

Proof. First, assume for simplicity that $\sum_{i=1}^{r} \mu_i = 1$. Also, $\Phi(1) = \sum_{i=1}^{r} w_i = 1$. Then $\mathit{ADAPT} \leq \Phi(2) \leq 2$, but also, more strongly:

$$\mathit{ADAPT} \leq \Phi(2) \leq 1 + \omega$$

where $\omega = w_r/\mu_r$. This follows from the concavity of $\Phi(x)$. Note that

$$\omega = \frac{w_r}{\mu_r} \leq \frac{\sum_{i=1}^{r} w_i}{\sum_{i=1}^{r} \mu_i} = 1.$$

With $\sum_{i=1}^{r} \mu_i = 1$, the algorithm has a simpler form:
- Choose $k \in \{1, 2, \ldots, r\}$ with probability w_k and insert item k first.
- Then, insert items $1, 2, \ldots, k-1, k+1, \ldots, r$ in the greedy order.

We estimate the expected value achieved by this algorithm. Note that we analyze the expectation with respect to the random sizes of items and also our own randomization. Item k is inserted with probability w_k first, with probability $\sum_{i=1}^{k-1} w_i$ after $\{1, 2, \ldots, k-1\}$ and with probability w_j after $\{1, 2, \ldots, k-1, j\}$ (for $k < j \leq r$). If it is the first item, the expected profit for it is simply $w_k = v_k \cdot \Pr[s_k \leq 1]$. If it is inserted after $\{1, 2, \ldots, k-1\}$, we use Lemma 2.1 to obtain

$$\Pr[\text{item } k \text{ fits}] = \Pr\left[\sum_{i=1}^{k} s_i \leq 1\right] \geq 1 - \sum_{i=1}^{k} \mu_i,$$

and the conditional expected profit for item k is in this case $v_k \cdot \Pr[\text{item } k \textit{ fits}] \geq w_k(1 - \sum_{i=1}^{k} \mu_i)$. The case when item k is preceded by $\{1, 2, \ldots, k-1, j\}$ is similar. Let V_k denote our lower bound on the expected profit obtained for item k:

$$\begin{aligned} V_k &= w_k\left(w_k + \sum_{j=1}^{k-1} w_j\left(1 - \sum_{i=1}^{k} \mu_i\right) + \sum_{j=k+1}^{r} w_j\left(1 - \sum_{i=1}^{k} \mu_i - \mu_j\right)\right) \\ &= w_k\left(\sum_{j=1}^{r} w_j\left(1 - \sum_{i=1}^{k} \mu_i\right) + w_k \sum_{i=1}^{k} \mu_i - \sum_{j=k+1}^{r} w_j \mu_j\right). \end{aligned}$$

We have $RNDGREEDY \geq \sum_{k=1}^{r} V_k$ and simplify the estimate using $\sum_{j=1}^{r} w_j = 1$ and $\sum_{j=1}^{r} \mu_j = 1$:

$$\begin{aligned} RNDGREEDY &\geq \sum_{k=1}^{r} w_k \left(1 - \sum_{i=1}^{k} \mu_i + w_k \sum_{i=1}^{k} \mu_i - \sum_{i=k+1}^{r} w_i \mu_i\right) \\ &= 1 + \sum_{1 \leq i \leq k \leq r} (-w_k \mu_i + w_k^2 \mu_i) - \sum_{1 \leq k < i \leq r} w_k w_i \mu_i \\ &= 1 + \sum_{1 \leq i \leq k \leq r} (-w_k \mu_i + w_k^2 \mu_i + w_k w_i \mu_i) - \sum_{i,k=1}^{r} w_k w_i \mu_i \\ &= 1 + \sum_{1 \leq i \leq k \leq r} w_k \mu_i (w_i + w_k - 1) - \sum_{i=1}^{r} w_i \mu_i. \end{aligned}$$

To symmetrize this polynomial, we apply the condition of greedy ordering. For any $i < k$, we have $w_i + w_k - 1 \leq 0$, and the greedy ordering implies $w_k \mu_i \leq w_i \mu_k$, allowing us to replace $w_k \mu_i$ by $\frac{1}{2}(w_k \mu_i + w_i \mu_k)$ for all pairs $i < k$:

$$\begin{aligned} RNDGREEDY &\geq 1 + \tfrac{1}{2} \sum_{1 \leq i < k \leq r} (w_k \mu_i + w_i \mu_k)(w_i + w_k - 1) + \sum_{i=1}^{r} w_i \mu_i (2w_i - 1) - \sum_{i=1}^{r} w_i \mu_i \\ &= 1 + \tfrac{1}{2} \sum_{i,k=1}^{r} w_k \mu_i (w_i + w_k - 1) + \tfrac{1}{2} \sum_{i=1}^{r} w_i \mu_i (2w_i - 1) - \sum_{i=1}^{r} w_i \mu_i \\ &= 1 + \tfrac{1}{2} \sum_{k=1}^{r} w_k \sum_{i=1}^{r} w_i \mu_i + \tfrac{1}{2} \sum_{k=1}^{r} w_k^2 \sum_{i=1}^{r} \mu_i - \tfrac{1}{2} \sum_{k=1}^{r} w_k \sum_{i=1}^{r} \mu_i + \sum_{i=1}^{r} w_i^2 \mu_i - \tfrac{3}{2} \sum_{i=1}^{r} w_i \mu_i \end{aligned}$$

and using again $\sum_{j=1}^{r} w_j = \sum_{j=1}^{r} \mu_j = 1$,

$$\begin{aligned} RNDGREEDY &\geq 1 + \tfrac{1}{2} \sum_{i=1}^{r} w_i \mu_i + \tfrac{1}{2} \sum_{k=1}^{r} w_k^2 - \tfrac{1}{2} + \sum_{i=1}^{r} w_i^2 \mu_i - \tfrac{3}{2} \sum_{i=1}^{r} w_i \mu_i \\ &= \tfrac{1}{2} + \tfrac{1}{2} \sum_{k=1}^{r} w_k^2 + \sum_{i=1}^{r} w_i^2 \mu_i - \sum_{i=1}^{r} w_i \mu_i. \end{aligned}$$

We want to compare this expression to $1 + \omega$ where $\omega = \min\{w_i/\mu_i\colon i \leq r\}$. We use the value of ω to estimate $\sum_{k=1}^{r} w_k^2 \geq \omega \sum_{k=1}^{r} w_k \mu_k$, and we obtain

$$\begin{aligned} RNDGREEDY &\geq \frac{1}{2} + \frac{\omega}{2} \sum_{k=1}^{r} w_k \mu_k + \sum_{i=1}^{r} w_i^2 \mu_i - \sum_{i=1}^{r} w_i \mu_i \\ &= \frac{1}{2} + \sum_{i=1}^{r} \mu_i w_i \left(\frac{\omega}{2} + w_i - 1\right). \end{aligned}$$

Each term in the summation above is a quadratic function of w_i that is minimized at $w_i = 1/2 - \omega/4$, so

$$RNDGREEDY \geq \frac{1}{2} - \sum_{i=1}^{r} \mu_i \left(\frac{1}{2} - \frac{\omega}{4}\right)^2.$$

Finally, $\sum_i \mu_i = 1$ and

$$RNDGREEDY \geq \frac{1}{4} + \frac{\omega}{4} - \frac{\omega^2}{16}.$$

We compare this to the adaptive optimum that is bounded by $1 + \omega$, and minimize over $\omega \in [0, 1]$:

$$\frac{RNDGREEDY}{ADAPT} \geq \frac{1}{4} - \frac{\omega^2}{16(1+\omega)} \geq \frac{7}{32}.$$

It remains to remove the assumption that $\sum_{i=1}^{r} \mu_i = 1$. We claim that if $\sum_{i=1}^{r} \mu_i > 1$, the randomized greedy algorithm performs just like the simplified algorithm we just analyzed, on a modified instance with values w'_j and mean sizes μ'_j (so that $\sum_{i=1}^{r} \mu'_j = 1$; see the description of the algorithm). Indeed, $\Phi(1) = 1$ and $\omega = w_r/\mu_r = w'_r/\mu'_r$ in both cases, so the bound on $ADAPT$ is the same. For an item $k < r$, our estimate of the expected profit in both instances is

$$V_k = w'_k\left(w'_k + \sum_{j=1}^{k-1} w'_j\left(1 - \sum_{i=1}^{k} \mu'_i\right) + \sum_{j=k+1}^{r} w'_j\left(1 - \sum_{i=1}^{k} \mu'_i - \mu'_j\right)\right).$$

For the original instance, this is because the expected contribution of item r to the total size, conditioned on being selected first, is $p'\mu_r = \mu'_r$; if not selected first, its contribution is not counted at all. The expected profit for item r is $V_r = w'_r p' w_r = (w'_r)^2$ in both instances. This reduces the analysis to the case we dealt with already, completing the proof of the theorem. □

Our randomized policy can be easily derandomized. Indeed, we can simply enumerate all *deterministic* nonadaptive policies that can be obtained by our randomized policy: Insert a selected item first, and then follow the greedy ordering. We can estimate the expected value for each such ordering in polynomial time using the lower bound derived in the proof of the theorem, and then choose the best one. This results in a deterministic nonadaptive policy achieving at least $(7/32)ADAPT$.

Example 4. This analysis is tight in the following sense: Consider an instance with eight identical items with $\mu_i = 1/4$ and $w_i = v_i = 1$. Our bound on the adaptive optimum would be $\Phi(2) = 8$, whereas our analysis of any nonadaptive algorithm would imply the following. We always get the first item (because $w_1 = v_1 = 1$), the second one with probability at least $1 - 2/4 = 1/2$, and the third one with probability at least $1 - 3/4 = 1/4$. Thus, our estimate of the expected value obtained is $7/4$. We cannot prove a better bound than $32/7$ with the tools we are using: the LP from Theorem 3.1, and Markov bounds based on mean item sizes. Of course, the actual adaptivity gap for this instance is one, and our algorithm performs optimally.

Example 5. It can be the case that $RNDGREEDY \approx ADAPT/4$. Consider an instance with multiple items of two types: those of size $(1+\varepsilon)/2$ and value $1/2 + \varepsilon$, and those of size $Be(p)$ and value p. Our algorithm will choose a sequence of items of the first type, of which only one can fit. The optimum is a sequence of items of the second type, which yields expected value $2 - p$. For small $p, \varepsilon > 0$, the gap can be arbitrarily close to 4. We have no example where the greedy algorithm performs worse than this. We can only prove that the approximation factor is at most $32/7 \approx 4.57$ but it seems that the gap between 4 and 4.57 is only due to the weakness of Markov bounds.

In the following sections, we actually present a different nonadaptive algorithm which improves the approximation factor to 4. However, the tools we employ to achieve this are more involved. The advantage of the 4.57-approximation algorithm is that it is based on the simple LP from Theorem 3.1 and the analysis uses only Markov bounds. This simpler analysis has already been shown to be useful in the analysis of other stochastic packing and scheduling problems (Vondrák [27], Dean [4]).

5. A stronger bound on the adaptive optimum.

In this section, we develop a stronger upper bound on *ADAPT* and use it to prove an approximation bound of 4 for a simple greedy nonadaptive policy. As before, let A denote the (random) set of items that an adaptive policy attempts to insert. In general, we know that $\mathbf{E}[\mu(A)] \le 2$. Here, we examine more closely how this mass can be distributed among items. By fixing a subset of items J, we show that although the quantity $\mathbf{E}[\mu(A \cap J)]$ can approach 2 for large $\mu(J)$, we obtain a stronger bound for small $\mu(J)$.

Lemma 5.1. *For any adaptive policy, let A be the* (*random*) *set of items that it attempts to insert. Then for any set of items J,*

$$\mathbf{E}[\mu(A \cap J)] \le 2\left(1 - \prod_{j \in J}(1 - \mu_j)\right).$$

Proof. Denote by $A(c)$ the set of items that an adaptive policy attempts to insert, given that the initial available capacity is c. Let $M(J, c) = \sup_{\mathcal{P}} \mathbf{E}[\mu(A(c) \cap J)]$ denote the largest possible expected mass that such a policy can attempt to insert, counting only items from J. We prove by induction on $|J|$ that

$$M(J, c) \le (1 + c)\left(1 - \prod_{j \in J}(1 - \mu_j)\right).$$

Without loss of generality, we can just assume that the set of available items is J; we do not gain anything by inserting items outside of J. Suppose that a policy in a given configuration (J, c) inserts item $i \in J$. The policy collects mass μ_i and then continues, provided that $s_i \le c$. We denote the indicator variable of this event by $\text{fit}(i, c)$, and we set $J' = J \backslash \{i\}$; the remaining capacity will be $c - s_i \ge 0$, and therefore the continuing policy cannot insert more expected mass than $M(J', c-s_i)$. Denoting by $B \subseteq J$ the set of all items in J that the policy attempts to insert, we get

$$\mathbf{E}[\mu(B)] \le \mu_i + \mathbf{E}[\mathrm{fit}(i,c) M(J', c-s_i)].$$

We apply the induction hypothesis to $M(J', c-s_i)$:

$$\mathbf{E}[\mu(B)] \le \mu_i + \mathbf{E}\left[\mathrm{fit}(i,c)(1+c-s_i)\left(1-\prod_{j\in J'}(1-\mu_j)\right)\right].$$

We denote the truncated size of item i by $\tilde{s}_i = \min\{s_i, 1\}$; therefore, we can replace s_i by $\tilde{s}_i$:

$$\mathbf{E}[\mu(B)] \le \mu_i + \mathbf{E}\left[\mathrm{fit}(i,c)(1+c-\tilde{s}_i)\left(1-\prod_{j\in J'}(1-\mu_j)\right)\right],$$

and then we note that $1+c-\tilde{s}_i \ge 0$ holds always, not only when item i fits. Therefore, we can replace $\mathrm{fit}(i,c)$ by 1 and evaluate the expectation:

$$\begin{aligned}\mathbf{E}[\mu(B)] &\le \mu_i + \mathbf{E}\left[(1+c-\tilde{s}_i)\left(1-\prod_{j\in J'}(1-\mu_j)\right)\right]\\ &= \mu_i + (1+c-\mu_i)\left(1-\prod_{j\in J'}(1-\mu_j)\right)\\ &= (1+c)-(1+c-\mu_i)\prod_{j\in J'}(1-\mu_j).\end{aligned}$$

Finally, using $(1+c-\mu_i) \ge (1+c)(1-\mu_i)$, we get:

$$\mathbf{E}[\mu(B)] \le (1+c)-(1+c)(1-\mu_i)\prod_{j\in J'}(1-\mu_j) = (1+c)\left(1-\prod_{j\in J}(1-\mu_j)\right).$$

Because this holds for any adaptive policy, we conclude that $M(J,c) \le (1+c)(1-\prod_{j\in J}(1-\mu_j))$. □

Theorem 5.1. *ADAPT* $\le \Psi(2)$, *where*

$$\Psi(t) = \max\left\{\sum_i w_i x_i : \begin{array}{ll} \forall J \subseteq [n]; & \sum_{i\in J}\mu_i x_i \le t\left(1-\prod_{i\in J}(1-\mu_i)\right) \\ \forall i \in [n]; & x_i \in [0,1] \end{array}\right\}.$$

Proof. Just as in the proof of Theorem 3.1, we consider any adaptive policy $\mathcal{P}$ and derive from it a feasible solution $\vec{x}$ with $x_i = \Pr[i \in A]$ for the LP for $\Psi(2)$ (feasibility now follows from Lemma 5.1 rather than Lemma 3.1). Thus, $\Psi(2)$ is an upper bound on *ADAPT*. □

This is a strengthening of Theorem 3.1 in the sense that $\Psi(2) \le \Phi(2)$. This holds because any solution feasible for $\Psi(2)$ is feasible for $\Phi(2)$. Observe also that $\Psi(t)$ is a concave function and, in particular, $\Psi(2) \le 2\Psi(1)$.

$\Psi(t)$ turns out to be the solution of a polymatroid optimization problem that can be found efficiently. We discuss the properties of this LP in more detail in the appendix. In particular, we show that there is a simple closed-form expression for $\Psi(1)$. The optimal solution is obtained by indexing the items in the order of nonincreasing w_i/μ_i and choosing $x_1, x_2, \ldots$ successively, setting each x_k as large as possible without violating the constraint for $J = \{1, 2, \ldots, k\}$. This yields $x_k = \prod_{i=1}^{k-1}(1-\mu_i)$.

Corollary 5.1. *The adaptive optimum is bounded by ADAPT* $\le 2\Psi(1)$ *where*

$$\Psi(1) = \sum_{k=1}^{n} w_k \prod_{i=1}^{k-1}(1-\mu_i),$$

and the items are indexed in nonincreasing order of w_i/μ_i.

6. A 4-approximation for stochastic knapsack.

Consider the following simple nonadaptive policy, which we call the *simplified greedy algorithm*. First, we compute the value of $\Psi(1)$ according to Corollary 5.1.

- If there is an item i such that $w_i \geq \Psi(1)/2$, then insert only item i.
- Otherwise, insert all items in the greedy order $w_1/\mu_1 \geq w_2/\mu_2 \geq w_3/\mu_3 \geq \cdots \geq w_n/\mu_n$.

We claim that the expected value obtained by this policy, *GREEDY*, satisfies *GREEDY* $\geq \Psi(1)/2$, from which we immediately obtain *ADAPT* $\leq 2\Psi(1) \leq 4$ *GREEDY*. First, we prove a general lemma on sums of random variables. The lemma estimates the expected mass that our algorithm *attempts* to insert.

Lemma 6.1. *Let $X_1, X_2, \ldots, X_k$ be independent, nonnegative random variables and $\mu_i = \mathbf{E}[\min\{X_i, 1\}]$. Let $S_0 = 0$ and $S_{i+1} = S_i + X_{i+1}$. Let $p_i = \Pr[S_i < 1]$. Then*

$$\sum_{j=1}^{k} p_{j-1}\mu_j \geq 1 - \prod_{j=1}^{k}(1-\mu_j).$$

Note. We need not assume anything about the total expectation. This works even for $\sum_{i=1}^{k}\mu_i > 1$.

For the special case of k random variables of equal expectation $\mu_j = 1/k$, Lemma 6.1 implies,

$$\frac{1}{k}\sum_{j=1}^{k}\Pr[S_{j-1} < 1] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}. \tag{1}$$

This seems related to a question raised by Feige [8]: what is the probability that $S_{k-1} < 1$ for a sum of independent random variables $S_{k-1} = X_1 + X_2 + \cdots + X_{k-1}$ with expectations $\mu_j = 1/k$? Feige proves that the probability is at least $1/13$, but conjectures that it is in fact at least $1/e$. A more general conjecture would be that for any $j < k$,

$$p_j = \Pr[S_j < 1] \geq \left(1 - \frac{1}{k}\right)^j. \tag{2}$$

Note that Markov's inequality would give only $p_j \geq 1 - j/k$. Summing up (2) from $j = 0$ up to $k-1$, we would get (1). However, (2) remains a conjecture and we can only prove (1) as a special case of Lemma 6.1.

Proof. Define $\sigma_i = \mathbf{E}[S_i \mid A_i]$, where A_i is the event that $S_i < 1$. By conditional expectations (remember that X_{i+1} is independent of A_i):

$$\begin{aligned}
\sigma_i + \mu_{i+1} &= \mathbf{E}[S_i \mid A_i] + \mathbf{E}[\min\{X_{i+1}, 1\}] = \mathbf{E}[S_i + \min\{X_{i+1}, 1\} \mid A_i] \\
&= \mathbf{E}[S_{i+1} \mid A_{i+1}]\Pr[A_{i+1} \mid A_i] + \mathbf{E}[S_i + \min\{X_{i+1}, 1\} \mid \bar{A}_{i+1} \cap A_i]\Pr[\bar{A}_{i+1} \mid A_i] \\
&\geq \sigma_{i+1}\frac{\Pr[A_{i+1}]}{\Pr[A_i]} + 1 \cdot \left(1 - \frac{\Pr[A_{i+1}]}{\Pr[A_i]}\right) \\
&= \sigma_{i+1}\frac{p_{i+1}}{p_i} + \left(1 - \frac{p_{i+1}}{p_i}\right) \\
&= 1 - (1 - \sigma_{i+1})\frac{p_{i+1}}{p_i}.
\end{aligned}$$

This implies that

$$\frac{p_{i+1}}{p_i} \geq \frac{1 - \sigma_i - \mu_{i+1}}{1 - \sigma_{i+1}}. \tag{3}$$

For $i = 0$ we obtain $p_1 \geq (1-\mu_1)/(1-\sigma_1)$, because $p_0 = 1$ and $\sigma_0 = 0$. Let us now consider two cases. First, suppose that $\sigma_i + \mu_{i+1} < 1$ for all i, $0 \leq i < k$. In this case, the ratio on the right-hand side of (3) is always nonnegative, and we can multiply (3) from $i = 0$ up to $j-1$, for any $j \leq k$:

$$\begin{aligned}
p_j &\geq \frac{1-\mu_1}{1-\sigma_1} \cdot \frac{1-\sigma_1-\mu_2}{1-\sigma_2} \cdots \frac{1-\sigma_{j-1}-\mu_j}{1-\sigma_j} \\
&= (1-\mu_1)\left(1 - \frac{\mu_2}{1-\sigma_1}\right)\cdots\left(1 - \frac{\mu_j}{1-\sigma_{j-1}}\right)\frac{1}{1-\sigma_j}.
\end{aligned}$$

We define

$$\nu_i = \frac{\mu_i}{1-\sigma_{i-1}}.$$

Therefore,

$$p_j \geq \frac{1}{1-\sigma_j} \prod_{i=1}^{j} (1-\nu_i), \tag{4}$$

and

$$\sum_{j=1}^{k} p_{j-1}\mu_j \geq \sum_{j=1}^{k} \nu_j \prod_{i=1}^{j-1} (1-\nu_i) = 1 - \prod_{i=1}^{k} (1-\nu_i). \tag{5}$$

By our earlier assumption, $\mu_i \leq \nu_i \leq 1$ for all $1 \leq i \leq k$. It follows that

$$\sum_{j=1}^{k} p_{j-1}\mu_j \geq 1 - \prod_{i=1}^{k} (1-\mu_i). \tag{6}$$

In the second case, we have $\sigma_j + \mu_{j+1} \geq 1$ for some $j < k$; consider the first such j. Then $\sigma_i + \mu_{i+1} < 1$ for all $i < j$, and we can apply the previous arguments to variables $X_1, \ldots, X_j$. From (5),

$$\sum_{i=1}^{j} p_{i-1}\mu_i \geq 1 - \prod_{i=1}^{j} (1-\nu_i). \tag{7}$$

In addition, we estimate the contribution of the $(j+1)$th item, which has mass $\mu_{j+1} \geq 1-\sigma_j$, and from (4) we get

$$p_j \mu_{j+1} \geq p_j(1-\sigma_j) \geq \prod_{i=1}^{j} (1-\nu_i). \tag{8}$$

Therefore, in this case we obtain from (7) + (8):

$$\sum_{i=1}^{k} p_{i-1}\mu_i \geq \sum_{i=1}^{j} p_{i-1}\mu_i + p_j\mu_{j+1} \geq 1. \quad \square$$

Theorem 6.1. *The simplified greedy algorithm obtains expected value GREEDY* $\geq \Psi(1)/2 \geq$ *ADAPT*$/4$.

Proof. If the algorithm finds an item i to insert with $w_i \geq \Psi(1)/2$, then clearly by inserting just this single item it will obtain an expected value of at least $\Psi(1)/2$. Let us therefore focus on the case where $w_i < \Psi(1)/2$ for all items i.

Let $X_i = s_i$ be the random size of item i. Lemma 6.1 says that the expected mass that our greedy algorithm *attempts to insert*, restricted to the first k items, is at least $1 - \prod_{i=1}^{k}(1-\mu_i)$. As in Lemma 6.1, we denote by p_k the probability that the first k items fit. We estimate the following quantity:

$$\sum_{i=1}^{n} p_{i-1} w_i = \sum_{i=1}^{n} \frac{w_i}{\mu_i} p_{i-1}\mu_i = \sum_{k=1}^{n} \left(\frac{w_k}{\mu_k} - \frac{w_{k+1}}{\mu_{k+1}} \right) \sum_{i=1}^{k} p_{i-1}\mu_i,$$

where we define $w_{n+1}/\mu_{n+1} = 0$ for simplicity. Using Lemma 6.1, we have

$$\begin{aligned}
\sum_{i=1}^{n} p_{i-1} w_i &\geq \sum_{k=1}^{n} \left(\frac{w_k}{\mu_k} - \frac{w_{k+1}}{\mu_{k+1}} \right) \left(1 - \prod_{i=1}^{k} (1-\mu_i) \right) \\
&= \sum_{k=1}^{n} \frac{w_k}{\mu_k} \left(\prod_{i=1}^{k-1} (1-\mu_i) - \prod_{i=1}^{k} (1-\mu_i) \right) \\
&= \sum_{k=1}^{n} \frac{w_k}{\mu_k} \left(\prod_{i=1}^{k-1} (1-\mu_i) \right) (1-(1-\mu_k)) \\
&= \sum_{k=1}^{n} w_k \prod_{i=1}^{k-1} (1-\mu_i) = \Psi(1).
\end{aligned}$$

The simplified greedy algorithm then obtains expected value

$$\sum_{i=1}^{n} p_i w_i = \sum_{i=1}^{n} p_{i-1} w_i - \sum_{i=1}^{n} (p_{i-1}-p_i) w_i \geq \Psi(1) - \frac{\Psi(1)}{2} \sum_{i=1}^{n} (p_{i-1}-p_i) \geq \Psi(1)/2. \quad \square$$

This analysis is tight in that we can have $GREEDY \approx ADAPT/4$. The example showing this is the same as our earlier example that gives $RNDGREEDY \approx ADAPT/4$. Also, the following instance shows that it is impossible to achieve a better factor than four using Theorem 5.1 to bound the adaptive optimum.

Example 6. For an instance with an unlimited supply of items of value $v_i = 1$ and deterministic size $s_i = (1+\varepsilon)/2$, we have $\Psi(1) = 2/(1+\varepsilon)$ and an upper bound $ADAPT \le 2\Psi(1) = 4/(1+\varepsilon)$, whereas only one item can fit. The same holds even for the stronger bound of $\Psi(2)$: Because $x_i = \min\{1, 2^{3-i}\}/(1+\varepsilon)$ is a feasible solution whose value converges to $\sum_{i=1}^{\infty} x_i = 4/(1+\varepsilon)$, we get $\Psi(2) \ge 4/(1+\varepsilon)$, which is almost four times as much as the true optimum.

Example 7. Both greedy algorithms we have developed thus far only examine mean truncated item sizes and probabilities of exceeding the capacity; no other information about the item-size distributions is used. It turns out that under these restrictions, it is impossible to achieve an approximation factor better than three. Suppose we have two types of unit-value items, each in unlimited supply. The first type has size $Be(1/2+\varepsilon)$ and the second has size $s_2 = 1/2+\varepsilon$. The optimal solution is to insert only items of the first kind, which yields an expected number of $2/(1/2+\varepsilon) - 1 = 3 - O(\varepsilon)$ successfully inserted items. However, an algorithm that can only see the mean truncated sizes might be fooled into selecting a sequence of the second kind instead—and it will insert only one item.

7. A $(2+\varepsilon)$-approximation for small items.

Consider a special scenario in which the truncated mean size of each item is very small. We would like to achieve a better approximation ratio in this case. Recall the analysis in §6, which relies on an estimate of the mass that our algorithm *attempts* to insert. Intuitively, the mass of the item that we overcount is very small in this case, so there is negligible difference between the mass we attempt to insert and what we insert successfully. Still, this argument requires a little care, because we need a small relative, rather than additive, error.

Lemma 7.1. *Let $X_1, X_2, \ldots, X_k$ be independent, nonnegative random variables and suppose that for each i, $\mu_i = \mathbf{E}[\min\{X_i, 1\}] \le \varepsilon$. Let $p_k = \Pr[\sum_{i=1}^{k} X_i \le 1]$. Then*

$$\sum_{j=1}^{k} p_j \mu_j \ge (1-\varepsilon)\left(1 - \prod_{j=1}^{k}(1-\mu_j)\right).$$

Proof. We extend the proof of Lemma 6.1. Consider two cases. First, suppose that $\mu_j < 1 - \sigma_{j-1}$ for all $j \in [k]$. Then by applying (5) and using the fact that $\mu_j \le \varepsilon$,

$$\begin{aligned}\sum_{j=1}^{k} p_j\mu_j &= \sum_{j=1}^{k} p_{j-1}\mu_j - \sum_{j=1}^{k}(p_{j-1}-p_j)\mu_j \ge 1 - \prod_{j=1}^{k}(1-\nu_j) - \sum_{j=1}^{k}(p_{j-1}-p_j)\varepsilon \\ &= 1 - \prod_{j=1}^{k}(1-\nu_j) - (p_0 - p_k)\varepsilon = (1-\varepsilon) - \prod_{j=1}^{k}(1-\nu_j) + \varepsilon p_k.\end{aligned}$$

Using (4) and our assumption that $\mu_j \le \nu_j$ for all $j \in [k]$, we now have

$$\sum_{j=1}^{k} p_j\mu_j \ge (1-\varepsilon) - \prod_{j=1}^{k}(1-\nu_j) + \frac{\varepsilon}{1-\sigma_k}\prod_{j=1}^{k}(1-\nu_j) \ge (1-\varepsilon)\left(1 - \prod_{j=1}^{k}(1-\mu_j)\right).$$

On the other hand, if $\mu_{j+1} > 1 - \sigma_j$ for some $j < k$, then consider the smallest such j. By (4),

$$\prod_{i=1}^{j}(1-\nu_j) \le (1-\sigma_j)p_j \le \mu_{j+1}p_j \le \varepsilon p_j.$$

Hence,

$$\sum_{i=1}^{k} p_i\mu_i \ge \sum_{i=1}^{j} p_i\mu_i \ge (1-\varepsilon) - \prod_{i=1}^{j}(1-\nu_i) + \varepsilon p_j \ge (1-\varepsilon) - \varepsilon p_j + \varepsilon p_j = 1-\varepsilon,$$

and this concludes the proof. □

Theorem 7.1. *Suppose that $\mu_i \le \varepsilon$ for all items i. Then the nonadaptive policy inserting items in the greedy order achieves expected value of at least $((1-\varepsilon)/2)\ ADAPT$.*

Proof. We modify the proof of Theorem 6.1 in a straightforward fashion using Lemma 7.1 in the place of Lemma 6.1. The expected value we obtain by inserting items in the greedy ordering is

$$\sum_{i=1}^{n} p_i w_i \geq (1-\varepsilon)\sum_{k=1}^{n}\left(\frac{w_k}{\mu_k}-\frac{w_{k+1}}{\mu_{k+1}}\right)\left(1-\prod_{i=1}^{k}(1-\mu_i)\right)$$
$$= (1-\varepsilon)\sum_{k=1}^{n} w_k \prod_{i=1}^{k-1}(1-\mu_i) = (1-\varepsilon)\Psi(1),$$

and we complete the proof by noting that $ADAPT \leq 2\Psi(1)$ (Corollary 5.1). □

8. An adaptive $(3+\varepsilon)$-approximation. Let S denote the set of small items (with $\mu_i \leq \varepsilon$) and L denote the set of large items ($\mu_i > \varepsilon$) in our instance, and let $ADAPT(S)$ and $ADAPT(L)$, respectively, denote the expected values obtained by an optimal adaptive policy running on just the sets S or L. In the previous section, we constructed a greedy nonadaptive policy whose expected value is $GREEDY \geq ((1-\varepsilon)/2)ADAPT(S)$.

In this section, we develop an *adaptive* policy for large items whose expected value $LARGE$ is at least $(1/(1+\varepsilon))ADAPT(L)$. Suppose we run whichever policy gives us a larger estimated expected value (both policies will allow us to estimate their expected values), so we end up inserting only small items, or only large items. We show that this gives us a $(3+5\varepsilon)$-approximate adaptive policy for arbitrary items.

Theorem 8.1. *Let $0<\varepsilon\leq 1/2$ and define large items by $\mu_i>\varepsilon$ and small items by $\mu_i\leq\varepsilon$. Applying either the greedy algorithm for small items (if $GREEDY \geq LARGE$) or the adaptive policy described in this section for large items (if $LARGE > GREEDY$), we obtain a $(3+5\varepsilon)$-approximate adaptive policy for the stochastic knapsack problem.*

Proof. Let $V = \max(GREEDY, LARGE)$ denote the expected value obtained by the policy described in the theorem. Using the fact that $1/(1-\varepsilon)\leq 1+2\varepsilon$ for $\varepsilon \leq 1/2$, we then have $ADAPT \leq ADAPT(S)+ADAPT(L) \leq (2/(1-\varepsilon))GREEDY + (1+\varepsilon)LARGE \leq (3+5\varepsilon)V$. □

We proceed now to describe our $(1+\varepsilon)$-approximate adaptive policy for large items. Given a set of remaining large items J and a remaining capacity c, it spends polynomial time (assuming ε is constant) and computes the next item to insert in a $(1+\varepsilon)$-approximate policy. Let b be an upper bound on the number of bits required to represent any item value w_i, instantiated item size for s_i, or probability value obtained by evaluating the cumulative distribution for s_i. Note that this implies that our probability distributions are effectively discrete. Assuming ε is a constant, our running time will be polynomial in n and b. Our policy also estimates the value it will eventually obtain, thereby computing a $(1+\varepsilon)$-approximation to $ADAPT(L)$ (i.e., the value of $LARGE$). It is worth noting that in contrast to our previous nonadaptive approaches, the adaptive policy here needs to know for each item i the complete cumulative distribution of s_i, rather than just μ_i and $\Pr[s_i \leq 1]$.

Our adaptive algorithm selects the first item to insert using a recursive computation that is reminiscent of the decision tree model of an adaptive policy in Figure 1. Given an empty knapsack, we first consider which item i we should insert first. For a particular item i, we estimate the expected value of an optimal adaptive policy starting with item i by randomly sampling (or rather, by using a special "assisted" form of random sampling) a polynomial number of instantiations of s_i and recursively computing the optimal expected value we can obtain using the remaining items on a knapsack of capacity $1-s_i$. Whichever item i yields the largest expected value is the item we choose to insert first. We can regard the entire computation as a large tree: The root performs a computation to decide which of the $|L|$ large items to insert first, and in doing so it issues recursive calls to level 1 nodes that decide which of $|L|-1$ remaining items to insert next, and so on. Each node at level l issues a polynomial number of calls to nodes at level $l+1$. We will show that by restricting our computation tree to at most a constant depth (depending on ε), we only forfeit an ε-fraction of the total expected value. Therefore, the entire computation runs in time polynomial in n and b, albeit with very high degree (so this is a result of mainly theoretical interest).

Let us define the function $F_{J,k}(c)$ to give the maximum additional expected value one can achieve if c units of capacity remain in the knapsack, and we may only insert at most k more items drawn from a set of remaining items $J\subseteq L$. For example, $ADAPT(L) = F_{L,|L|}(1)$. The analysis of our adaptive policy relies primarily on the following technical lemma.

Lemma 8.1. *Suppose all items are large, $\mu_i > \varepsilon$ with $\varepsilon\leq 1$. For any constant $\delta\in(0,1)$, any $c\in[0,1]$, any set of remaining large items $J\subseteq L$ and any $k=O(1)$, there exists a polynomial-time algorithm (which we call $A_{J,k,\delta}(c)$) that computes an item in J to insert first, which constitutes the beginning of an adaptive policy* *obtaining expected value in the range* $[F_{J,k}(c)/(1+\delta), F_{J,k}(c)]$. *The algorithm also computes a lower-bound estimate of the expected value it obtains, which we denote by* $G_{J,k,\delta}(c)$. *This function is nondecreasing in* c *and satisfies* $G_{J,k,\delta}(c) \in [F_{J,k}(c)/(1+\delta), F_{J,k}(c)]$.

Note. In this lemma and throughout its proof, polynomial time means a running time bounded by a polynomial in n and b, whose degree depends only on k and δ. We denote this polynomial by $poly_{k,\delta}(n,b)$.

We prove the lemma shortly, but consider for a moment its implications. Suppose we construct an adaptive policy that starts by invoking $A_{L,k,\delta}(1)$, where $\delta = \varepsilon/3$ and $k = 6/\varepsilon^2$. The expected value of this policy is $LARGE \geq G_{L,k,\delta}(1)$. Then

$$F_{L,k}(1) \leq \left(1+\frac{\varepsilon}{3}\right) LARGE.$$

Letting J_L denote the (random) set of large items successfully inserted by an optimal adaptive policy, Lemma 3.1 tells us that $\mathbf{E}[\mu(J_L)] \leq 2$, so Markov's inequality (and the fact that $\mu_i \geq \varepsilon$ for all $i \in L$) implies that $\Pr[|J_L| \geq k] \leq \varepsilon/3$. For any k, we can now decompose the expected value obtained by $ADAPT(L)$ into the value from the first k items, and the value from any items after the first k. The first quantity is upper bounded by $F_{L,k}(1)$ and the second quantity is upper bounded by $ADAPT(L)$ even when we condition on the event $|J_L| > k$. Therefore,

$$ADAPT(L) \leq F_{L,k}(1) + ADAPT(L)\Pr[|J_L| > k] \leq F_{L,k}(1) + \frac{\varepsilon}{3} ADAPT(L),$$

and because $\varepsilon \leq 1$ we have

$$ADAPT(L) \leq \frac{F_{L,k}(1)}{1-\varepsilon/3} \leq \frac{1+\varepsilon/3}{1-\varepsilon/3} LARGE \leq (1+\varepsilon) LARGE.$$

Proof of Lemma 8.1. We use induction on k, taking $k=0$ as a trivial base case. Assume the lemma now holds up to some value of k, so for every set $J \subseteq L$ of large items, for every δ (in particular $\delta/3$), for every $c \leq 1$, we have a polynomial-time algorithm $A_{J,k,\delta/3}(c)$. We use $\delta/3$ as the constant for our inductive step because $(1+\delta/3)^2 \leq 1+\delta$ for $\delta \in [0,1]$ (and we lose the factor of $1+\delta/3$ twice in our argument below). Note that this decreases our constant δ by a factor of three for every level of induction, but because we only carry the induction out to a constant number of levels, we can still treat δ as a constant at every step.

We now describe how to construct the algorithm $A_{\bullet,k+1,\delta}(\cdot)$ using a polynomial number of recursive calls to the polynomial-time algorithm $A_{\bullet,k,\delta/3}(\cdot)$. The algorithm $A_{J,k+1,\delta}(\cdot)$ must decide which item in J to insert first, given that we have c units of capacity remaining, in order to launch a $(1+\delta)$-approximate policy for scheduling at most $k+1$ items. To do this, we approximate the expected value we get with each item $i \in J$ 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 s_i, and for each one we call $A_{J\setminus\{i\},k,\delta/3}(c-s_i)$ to approximate the expected value $F_{J\setminus\{i\},k}(c-s_i)$ 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 an exponentially small probability of taking very small values for which $F_{J\setminus\{i\},k}(c-s_i)$ is large, we will likely miss this contribution to the aggregate expected value.

To remedy the problem above, we employ a sort of "assisted sampling" that first determines the interesting ranges of values of s_i we should consider. For simplicity of notation, let us assume implied subscripts for the moment and let $G(c)$ denote $G_{J\setminus\{i\},k,\delta/3}(c)$. We lower-bound $G(\cdot)$ by a piecewise-constant function $f(\cdot)$ with a polynomial number of breakpoints denoted $0 = c_0, \ldots, c_p = 1$. Our goal is to have $f(\cdot)$ be a $(1+\delta)$-approximation of $F(\cdot)$, so that we can use it to estimate the expected value of our near-optimal algorithm. Initially, we compute $f(c_p) = G(1)/(1+\delta/3)$ by a single invocation of $A_{J\setminus\{i\},k,\delta/3}(1)$. We then use binary search to compute each successive breakpoint $c_{p-1}, \ldots, c_1$ in reverse order. More precisely, once we have computed c_i, we determine c_{i-1} to be the maximum value of c such that $G(c) < f(c_i)$ and we set $f(c_{i-1}) = G(c_{i-1})/(1+\delta/3)$. We illustrate the construction of $f(\cdot)$ in Figure 2. The maximum number of steps required by the binary search will be polynomial in n and b, because we will ensure (by induction to at most a constant number of levels) that $G(c)$ always evaluates to a quantity represented by $poly_{k,\delta}(n,b)$ bits.

Each breakpoint of $f(\cdot)$ marks a change in $G(\cdot)$ by at least a $(1+\delta/3)$ factor. This ensures that f will have $poly_{k,\delta}(n,b)$ breakpoints, because G always evaluates to a quantity represented by a polynomial number of bits. Because f is a $(1+\delta/3)$-approximation to G, which is in turn a $(1+\delta/3)$-approximation to $F_{J\setminus\{i\},k}$, and because $(1+\delta/3)^2 \leq 1+\delta$, we know that $f(c) \in [F_{J\setminus\{i\},k}(c)/(1+\delta), F_{J\setminus\{i\},k}(c)]$.

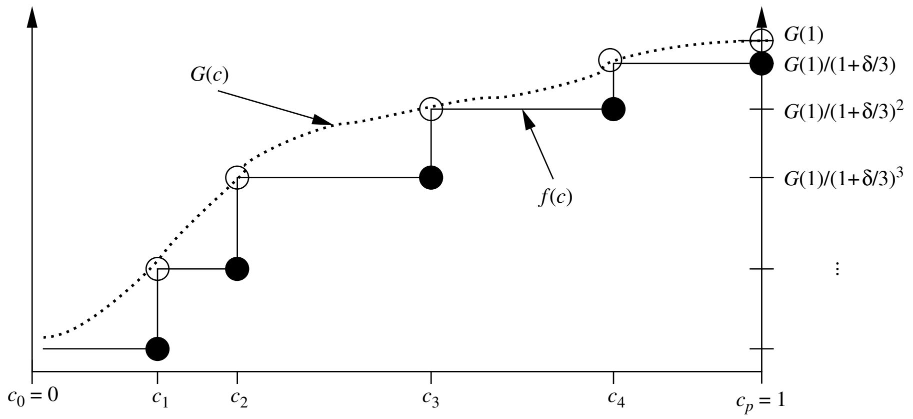


Figure 2. Approximation of the function $G(c) = G_{J\setminus\{i\},k,\delta/3}(c)$ (shown by a dotted line) by a piecewise-constant function $f(c)$ so that $f(c) \in [G(c)/(1+\delta/3), G(c)]$.

Assume for a moment now that i is the best first item to insert (the one that would be inserted first by an adaptive policy optimizing $F_{J,k+1}(c)$); we can write $F_{J,k+1}(c)$ as

$$F_{J,k+1}(c) = v_i \Pr[s_i \le c] + \int_{t=0}^{c} F_{J\setminus\{i\},k}(c-t) h_i(t)\, dt.$$

where $h_i(\cdot)$ is the probability density function for s_i. Because we are willing to settle for a $(1+\delta)$-approximation, we can use f inside the integral, and define the following function:

$$\begin{aligned} G^{(i)}_{J,k+1,\delta}(c) &= v_i \Pr[s_i \le c] + \int_{t=0}^{c} f(c-t) h_i(t)\, dt \\ &= v_i \Pr[s_i \le c] + \sum_{j=1}^{p} f(c_j) \Pr[c_{j-1} < c - s_i \le c_j] + f(0) \Pr[s_i = c]. \end{aligned}$$

This is our estimate of the expected value obtained when the first item inserted is i. Maximizing over all $i \in J$ gives

$$G_{J,k+1,\delta}(c) = \max_{i\in J} G^{(i)}_{J,k+1,\delta}(c),$$

which is a $(1+\delta)$-approximation to $F_{J,k+1}(c)$. Observe that, as f is nondecreasing, each $G^{(i)}_{J,k+1,\delta}(c)$ is a nondecreasing function of c that implies the same for $G_{J,k+1,\delta}(c)$.

Finally, we address the issue of polynomial running time. Any value of $G_{J,k+1,\delta}(c)$ is representable by a polynomial number of bits, because we recurse on k only to a constant depth. On each level, we make a polynomial number of calls to evaluate $G_{\bullet,k,\delta/3}$. This yields a recursion tree with polynomially large degrees and constant depth; therefore, the number of nodes is polynomial. In total, the algorithm $A_{J,k+1,\delta}(\cdot)$ above makes only a polynomial number of calls to $A_{\bullet,k,\delta/3}(\cdot)$. □

9. The ordered adaptive and fixed-set models.

We next discuss approximation results for two slightly different models. The simplest of these is the *fixed-set* model, where we must specify a priori 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 model 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. If we start with the ordering suggested by our previous 4-approximate nonadaptive policy, then an optimal ordered adaptive policy can be no worse in terms of approximation guarantee, because the original nonadaptive policy is a feasible solution in the ordered adaptive model. One way to view the ordered adaptive model is, therefore, as a heuristic means of squeezing extra performance out of our existing nonadaptive policies. If we are not allowed to choose the ordering of items, an optimal ordered adaptive policy must be at least as good as an optimal solution in the fixed-set model, because it is a feasible ordered adaptive policy to simply insert items in a fixed set, discarding all others (it is likely that the ordered adaptive policy will obtain more value than we would get in the fixed-set model, because it gets "partial credit" even if only some of the items manage to fit). The main result we prove below is an approximation algorithm for the fixed-set model that delivers a solution of expected value $FIXED \geq (1/9.5) ADAPT$. Therefore, the expected value obtained by an optimal ordered adaptive policy using *any* initial ordering of items must fall within a factor of 9.5 of $ADAPT$.

Ordered adaptive models are worthwhile to consider, because for a given ordering of items, we can compute an optimal ordered adaptive policy in pseudopolynomial time using dynamic programming (DP), as long as all item-size distributions are discrete. By "discrete," we mean that for some $\delta > 0$, the support of s_i's distribution lies in $\{0, \delta, 2\delta, 3\delta, \ldots\}$ for all items i. If the s_is are deterministic, then it is well known that an optimal solution to the knapsack problem can be computed in $O(n/\delta)$ time via a simple dynamic program. The natural generalization of this dynamic program to the stochastic case gives us an $O((n/\delta)\log n)$ algorithm for computing an optimal ordered policy. Let $V(j,k)$ denote the optimal expected value one can obtain via an ordered adaptive policy using only items $j, \ldots, n$ with $k\delta$ units of capacity left. Then

$$V(j,k) = \max\left\{V(j+1,k),\ v_j \Pr[s_j \leq k\delta] + \sum_{t=0}^{k} V(j+1,k-t)\Pr[s_j = t\delta]\right\}.$$

A straightforward DP implementation based on this recurrence runs in $O(n^2/\delta)$ time, but we can speed this up to $O((n/\delta)\log n)$ by using the Fast Fourier Transform to handle the convolution work for each row $V(j,\cdot)$ in our table of subproblem solutions. An optimal adaptive solution is implicitly represented in the "traceback" paths through the table of subproblem solutions.

Although DP only applies to problems with discrete size distributions and only gives us pseudopolynomial running times, we can discretize *any* set of size distributions in a manner that gives us a polynomial running time, at the expense of only a slight loss in terms of feasibility—our policy may overrun the capacity of the knapsack by a $(1+\varepsilon)$ factor, for a small constant $\varepsilon > 0$ of our choosing. Suppose we discretize the distribution of s_i into a new distribution s_i' with $\delta = \varepsilon/n$ (so s_i' is represented by a vector of length n/ε), such that $\Pr[s_i' = k\delta] := \Pr[k\delta \leq s_i < (k+1)\delta]$. That is, we "round down" the probability mass in s_i to the next-lowest multiple of δ. Because the "actual" size of each item (according to s_i) may be up to ε/n larger than its "perceived" size (according to s_i'), our policy may insert up to $(1+\varepsilon)$ units of mass before it thinks it has reached a capacity of one.

9.1. An approximation algorithm for the fixed-set model. We now consider the computation of a set of items whose value times probability of fitting is at least $ADAPT/9.5$. Letting S denote the small items ($\mu_i \leq \varepsilon$) in our instance, we define

- $m_1 = \max_i w_i = \max_i\{v_i \Pr[s_i \leq 1]\}$, and
- $m_2 = \max\{\mathrm{val}(J)(1-\mu(J)) \colon J \subseteq S\}$.

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 either a single item i or a set J of small items, counting only the event that the entire set fits in the knapsack. Our fixed set of items is the better of the two: $FIXED = \max\{m_1, m_2\}$. We now compare $ADAPT$ to $FIXED$.

Lemma 9.1. *For any set $J \subseteq S$ of small items,*

$$\mathrm{val}(J) \leq \left(1 + \frac{4\mu(J)}{1-\varepsilon^2}\right) m_2.$$

Proof. We proceed by induction on $|J|$. For $J = \varnothing$, the statement is trivial. If $\mu(J) \geq (1-\varepsilon)/2$, choose a minimal $K \subseteq J$, such that $\mu(K) \geq (1-\varepsilon)/2$. Because the items have mean size of at most ε, $\mu(K)$ cannot exceed $(1+\varepsilon)/2$. By induction,

$$\mathrm{val}(J \backslash K) \leq \left(1 + \frac{4(\mu(J) - \mu(K))}{1-\varepsilon^2}\right) m_2,$$

and because $m_2 \geq \mathrm{val}(K)(1-\mu(K))$, for $\mu(K) \in [(1-\varepsilon)/2, (1+\varepsilon)/2]$ we have $\mu(K) m_2 \geq \mathrm{val}(K)\mu(K) \cdot (1-\mu(K)) \geq \frac{1}{4}(1-\varepsilon^2)\mathrm{val}(K)$, and $\mathrm{val}(J) = \mathrm{val}(J\backslash K) + \mathrm{val}(K) \leq (1 + 4\mu(J)/(1-\varepsilon^2)) m_2$. Finally, if $\mu(J) < (1-\varepsilon)/2$, then it easily follows that $\mathrm{val}(J) \leq m_2/(1-\mu(J)) \leq (1 + 4\mu(J)/(1-\varepsilon^2))\ m_2$. □

Theorem 9.1. *We have $ADAPT \leq 9.5\ FIXED$.*

Proof. Fix an optimal adaptive policy $\mathcal{P}$ and let $J = J_S \cup J_L$ denote the (random) set of items that P attempts to insert into the knapsack, partitioned into small and large items. For any large item i, let $x_i = \Pr[i \in J_L]$. Then the expected value P obtains from large items is bounded by $\sum_{i\in L} x_i w_i$. It follows that

$$\begin{aligned}
ADAPT &\le \mathbf{E}[\mathrm{val}(J_S)] + \sum_{i\in L} x_i w_i \\
&\le \left(1+\frac{4\mathbf{E}[\mu(J_S)]}{1-\varepsilon^2}\right)m_2 + \left(\sum_{i\in L} x_i\right)m_1 \\
&= \left(1+\frac{4\mathbf{E}[\mu(J_S)]}{1-\varepsilon^2}\right)m_2 + \mathbf{E}[|J_L|]m_1 \\
&\le \left(1+\frac{4\mathbf{E}[\mu(J_S)]}{1-\varepsilon^2}\right)m_2 + \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon}m_1 \\
&\le \left(1+\frac{4\mathbf{E}[\mu(J_S)]}{1-\varepsilon^2} + \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon}\right)FIXED \\
&\le \left(1+\max\left(\frac{4}{1-\varepsilon^2},\frac{1}{\varepsilon}\right)\mathbf{E}[\mu(J_S\cup J_L)]\right)FIXED \\
&\le \left(1+2\max\left(\frac{4}{1-\varepsilon^2},\frac{1}{\varepsilon}\right)\right)FIXED
\end{aligned}$$

and for $\varepsilon=\sqrt{5}-2\approx 0.236$ this gives us $ADAPT \le 9.48$ $FIXED$. Recall that we do not know how to compute m_2 exactly in polynomial time, although we can approximate this quantity to within an arbitrary constant factor. Taking this factor to be small enough, we obtain a 9.5-approximation algorithm that runs in polynomial time. ☐

10. Conclusion.

In this paper, we have developed tools for analyzing adaptive and nonadaptive strategies and their relative merit for a basic stochastic knapsack problem. Extensions to more complex problems, such as packing, covering, and scheduling problems, as well as slightly different stochastic models can be found in the Ph.D. theses of two of the authors: Dean [4], Vondrák [27].

Appendix A. Notes on the polymatroid LP.

Theorem 5.1 gives an upper bound of $\Psi(2)$ on the adaptive optimum, where $\Psi(t)$ is defined by a linear program in the following form:

$$\Psi(t) = \max\left\{\sum_i w_i x_i : \begin{array}{ll} \forall J\subseteq[n]; & \sum_{i\in J}\mu_i x_i \le t\left(1-\prod_{i\in J}(1-\mu_i)\right) \\ \forall i\in[n]; & x_i\in[0,1] \end{array}\right\}.$$

Here we show that although this LP has an exponential number of constraints, it can be solved efficiently. In fact, the optimal solution can be written in a closed form. The important observation here is that $f(J) = 1-\prod_{j\in J}(1-\mu_j)$ is a *submodular function*. This can be seen, for example, by interpreting $f(J)$ as $\Pr[\bigcup_{j\in J} E_j]$ where E_j are independent events occurring with probabilities μ_j. Such a function is submodular for any collection of events, because for any $K\subset J$, we have

$$f(J\cup\{x\}) - f(J) = \Pr\left[E_x \Big\backslash \bigcup_{j\in J} E_j\right] \le \Pr\left[E_x \Big\backslash \bigcup_{j\in K} E_j\right] = f(K\cup\{x\}) - f(K).$$

From now on, we assume that $\mu_i>0$ for each item, because items with $\mu_i=0$ can be inserted for free—in an optimal solution, they will be always present with $x_i=1$, and this only increases the value of $\Psi(t)$ by a constant. Therefore, assume $\mu_i>0$ and substitute $z_i=\mu_i x_i$. Then $\Psi(t)$ can be written as

$$\Psi(t) = \max\left\{\sum_i \frac{w_i}{\mu_i} z_i : \begin{array}{ll} \forall J\subseteq[n]; & z(J)\le tf(J) \\ \forall i\in[n]; & z_i\in[0,\mu_i] \end{array}\right\}$$

where $f(J)$ is a submodular function. Naturally, $tf(J)$ is submodular as well, for any $t\ge 0$. For now, ignore the constraints $z_i\le\mu_i$ and define

$$\tilde{\Psi}(t) = \max\left\{\sum_i \frac{w_i}{\mu_i} z_i : \begin{array}{ll} \forall J\subseteq[n]; & z(J)\le tf(J) \\ \forall i\in[n]; & z_i\ge 0 \end{array}\right\}.$$

Observe that for $t\le 1$, we have $\tilde{\Psi}(t)=\Psi(t)$, because $z_i\le\mu_i$ is implied by the condition for $J=\{i\}$. However, now we can describe the optimal solution defining $\tilde{\Psi}(t)$ explicitly.

As before, we assume that $w_1/\mu_1 \geq w_2/\mu_2 \geq \cdots$. The linear program defining $\widetilde{\Psi}(t)$ is a *polymatroid* with rank function $tf(J)$. The optimal solution can be found by a greedy algorithm that essentially sets the values of $z_1, z_2, z_3, \ldots$ successively as large as possible, without violating the constraints $\sum_{i=1}^{k} z_i \leq tf(\{1,2,\ldots,k\})$. The solution is $z_1 = t\mu_1$, $z_2 = t\mu_2(1-\mu_1)$, etc.:

$$z_k = tf(\{1,2,\ldots,k\}) - tf(\{1,2,\ldots,k-1\}) = t\mu_k \prod_{i=1}^{k-1}(1-\mu_i),$$

and submodularity guarantees that this in fact satisfies the constraints for all subsets J (Schrijver [21]). Thus, we have a closed form for $\widetilde{\Psi}(t)$:

$$\widetilde{\Psi}(t) = t\sum_{k=1}^{n} w_k \prod_{i=1}^{k-1}(1-\mu_i),$$

which yields, in particular, the formula for $\Psi(1) = \widetilde{\Psi}(1)$ that we mentioned at the end of §5.

Our original LP is an intersection of a polymatroid LP with a box; this is a polymatroid as well, see Schrijver [21]. It can be described using a different submodular function $g(J,t)$:

$$\Psi(t) = \max\left\{ \sum_i \frac{w_i}{\mu_i} z_i : \begin{array}{ll} \forall J \subseteq [n]; & z(J) \leq g(J,t) \\ \forall i \in [n]; & z_i \geq 0 \end{array} \right\}.$$

Note that the constraints $z_i \leq \mu_i$ are removed now. In general, the function $g(J,t)$ can be obtained as

$$g(J,t) = \min_{A\subseteq J}(tf(A) + \mu(J\backslash A))$$

(see Schrijver [21]). Here we get an even simpler form; we claim that it is enough to take the minimum over $A \in \{\varnothing, J\}$. Indeed, suppose the minimum is attained for a proper subset $\varnothing \neq M \subset J$. Recall that we assume $\mu_i > 0$ for all items. Choose $x \in M$, $y \in J\backslash M$ and let $M_1 = M\backslash\{x\}$, $M_2 = M \cup \{y\}$. We have

$$f(M) = 1 - \prod_{i\in M}(1-\mu_i) = 1 - (1-\mu_x)\prod_{i\in M_1}(1-\mu_i) = f(M_1) + \mu_x \prod_{i\in M_1}(1-\mu_i)$$

and, similarly,

$$f(M_2) = 1 - \prod_{i\in M_2}(1-\mu_i) = f(M) + \mu_y \prod_{i\in M}(1-\mu_i).$$

Now we distinguish two cases: If $t\prod_{i\in M}(1-\mu_i) < 1$, then

$$tf(M_2) + \mu(J\backslash M_2) = tf(M) + \mu_y t\prod_{i\in M}(1-\mu_i) + \mu(J\backslash M) - \mu_y < tf(M) + \mu(J\backslash M).$$

In case $t\prod_{i\in M}(1-\mu_i) \geq 1$, we have $t\prod_{i\in M_1}(1-\mu_i) > 1$:

$$tf(M) + \mu(J\backslash M) = tf(M_1) + \mu_x t\prod_{i\in M_1}(1-\mu_i) + \mu(J\backslash M_1) - \mu_x > tf(M_1) + \mu(J\backslash M_1).$$

Both cases contradict our assumption of minimality on M. Thus, we have

$$g(J,t) = \min\{tf(J), \mu(J)\} = \min\left\{ t\left(1 - \prod_{i\in J}(1-\mu_i)\right), \sum_{i\in J}\mu_i \right\}.$$

Again, we can find the optimal solution for this polymatroid using the greedy algorithm:

$$\begin{aligned} z_k &= g(\{1,2,\ldots,k\},t) - g(\{1,2,\ldots,k-1\},t) \\ &= \min\left\{ t\left(1-\prod_{i=1}^{k}(1-\mu_i)\right), \sum_{i=1}^{k}\mu_i \right\} - \min\left\{ t\left(1-\prod_{i=1}^{k-1}(1-\mu_i)\right), \sum_{i=1}^{k-1}\mu_i \right\}. \end{aligned}$$

To further simplify, we can use the fact that $f(\{1,2,\ldots,k\}) = 1 - \prod_{i=1}^{k}(1-\mu_i)$ is "concave" as a function of $\sum_{i=1}^{k}\mu_i$ (when extended to a piecewise-linear function $\tilde{f}$ satisfying $\tilde{f}(\sum_{i=1}^{k}\mu_i) = f(\{1,2,\ldots,k\})$). Therefore, there is at most one breaking point.

Define b to be the maximum $k \leq n$ such that

$$t\left(1-\prod_{i=1}^{k}(1-\mu_i)\right) \geq \sum_{i=1}^{k}\mu_i.$$

This certainly holds for $k = 0$ and due to concavity, it holds exactly up to $k = b$. Therefore,

- For $0 \leq k \leq b$: $g(\{1,\ldots,k\},t) = \sum_{i=1}^{k}\mu_i$.
- For $b < k \leq n$: $g(\{1,\ldots,k\},t) = t(1-\prod_{i=1}^{b}(1-\mu_i))$.

For the optimal LP solution, we get

- For $1 \le k \le b$: $z_k = \mu_k$.
- For $k = b+1$: $z_{b+1} = t(1 - \prod_{i=1}^{b+1}(1-\mu_i)) - \sum_{i=1}^{b}\mu_i$.
- For $b+2 \le k \le n$: $z_k = t\mu_k \prod_{i=1}^{k-1}(1-\mu_i)$.

The value of the optimal solution is

$$\Psi(t) = \sum_{i=1}^{n} \frac{w_i}{\mu_i} z_i = \sum_{k=1}^{b} w_k + \frac{w_{b+1}}{\mu_{b+1}} \left(t\left(1 - \prod_{i=1}^{b+1}(1-\mu_i)\right) - \sum_{i=1}^{b}\mu_i \right) + t \sum_{k=b+2}^{n} w_k \prod_{i=1}^{k-1}(1-\mu_i).$$

(where some of the terms might be void if $b = 0$, $n-1$, or n). Thus, we have the solution of $\Psi(t)$ in a closed form. In some cases, it can be stronger than the formula presented in Corollary 5.1. Nonetheless, we know that both of them can differ from the actual optimum by a factor of four.

Acknowledgments. The authors thank the anonymous referees for many useful suggestions. This research was supported in part by NSF Grants CCR-0098018, ITR-0121495, and CCF-0515221 and by ONR Grant N00014-05-1-0148. A preliminary version of this paper with somewhat weaker results appeared in Dean et al. [5].

References

[1] Birge, J. R., F. V. Louveaux. 1997. *Introduction to Stochastic Programming*. Springer Verlag, New York.
[2] Carraway, R. L., R. L. Schmidt, L. R. Weatherford. 1993. An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns. *Naval Res. Logist.* **40** 161–173.
[3] Chang, C.-S., X. Chao, M. Pinedo, R. R. Weber. 1992. On the optimality of LEPT and $c\mu$ rules for machines in parallel. *J. Appl. Probab.* **29** 667–681.
[4] Dean, B. C. 2005. Approximation algorithms for stochastic scheduling problems. Ph.D. thesis, Massachusetts Institute of Technology, Boston.
[5] Dean, B. C., M. X. Goemans, J. Vondrák. 2004. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Proc. 45th IEEE Sympos. Found. Comput. Sci.* (*FOCS*), Rome, 208–217.
[6] Derman, C., C. J. Lieberman, S. M. Ross. 1978. A renewal decision problem. *Management Sci.* **24**(5) 554–561.
[7] Emmons, H., M. Pinedo. 1990. Scheduling stochastic jobs with due dates on parallel machines. *Eur. J. Oper. Res.* **47** 49–55.
[8] Feige, U. 2004. On the sum of independent random variables with unbounded variance, and estimating the average degree in a graph. *Proc. 36th ACM-SIAM Sympos. Discrete Algorithms* (*SODA*), 594–603.
[9] Goel, A., P. Indyk. 1999. Stochastic load balancing and related problems. *Proc. 40th IEEE Found. Comput. Sci.*, 579–586.
[10] Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.* **5** 287–326.
[11] Henig, M. 1990. Risk criteria in a stochastic knapsack problem. *Oper. Res.* **38**(5) 820–825.
[12] Immorlica, N., D. Karger, M. Minkoff, V. Mirrokni. 2004. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, 184–693.
[13] Kleinberg, J., Y. Rabani, E. Tardos. 1997. Allocating bandwidth for bursty connections. *Proc. 29th ACM Sympos. Theory Comput.*, 664–673.
[14] Kleywegt, A., J. D. Papastavrou. 2001. The dynamic and stochastic knapsack problem with random sized items. *Oper. Res.* **49**(1) 26–41.
[15] Koutsoupias, E., C. H. Papadimitriou. 2000. Beyond competitive analysis. *SIAM J. Comput.* **30**(1) 300–317.
[16] Möhring, R. H., A. S. Schulz, M. Uetz. 1999. Approximation in stochastic scheduling: The power of LP-based priority policies. *J. ACM* **46**(6) 924–942.
[17] Papastavrou, J. D., S. Rajagopalan, A. Kleywegt. 1996. The dynamic and stochastic knapsack problem with deadlines. *Management Sci.* **42**(12) 1706–1718.
[18] Pinedo, M. 1983. Stochastic scheduling with release dates and due dates. *Oper. Res.* **31** 559–572.
[19] Ravi, R., A. Sinha. 2004. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. *Proc. 10th Integer Programming Combin. Optim.* (*IPCO*), 101–115.
[20] Rothkopf, M. 1966. Scheduling with random service times. *Management Sci.* **12**(9) 707–713.
[21] Schrijver, A. 2003. *Combinatorial Optimization—Polyhedra and Efficiency*. Springer Verlag, Berlin.
[22] Shmoys, D. B., C. Swamy. 2004. Stochastic optimization is (almost) as easy as deterministic optimization. *Proc. 45th IEEE Found. Comput. Sci.* (*FOCS*), 228–237.
[23] Skutella, M., M. Uetz. 2005. Stochastic machine scheduling with precedence constraints. *SIAM J. Comput.* **34**(4) 788–802.
[24] Sniedovich, M. 1980. Preference order stochastic knapsack problems: Methodological issues. *J. Oper. Res. Soc.* **31**(11) 1025–1032.
[25] Steinberg, E., M. S. Parks. 1979. A preference order dynamic program for a knapsack problem with stochastic rewards. *J. Oper. Res. Soc.* **30**(2) 141–147.
[26] Uetz, M. 2001. Algorithms for deterministic and stochastic scheduling. Ph.D. thesis, Institut für Mathematik, Technische Universität Berlin.
[27] Vondrák, J. 2005. Probabilistic methods in combinatorial and stochastic optimization. Ph.D. thesis, Massachusetts Institute of Technology, Boston.