Optimistic Planning for the Stochastic Knapsack Problem

Anonymous Author 1
Unknown Institution 1

Anonymous Author 2
Unknown Institution 2

Anonymous Author 3
Unknown Institution 3

Abstract

The stochastic knapsack problem is a stochastic resource allocation problem that arises frequently and yet is exceptionally hard to solve. We derive and study an optimistic planning algorithm specifically designed for the stochastic knapsack problem. Unlike other optimistic planning algorithms for MDPs, our algorithm, `OpStoK`, avoids the use of discounting and is adaptive to the amount of resources available. We achieve this behavior by means of a concentration inequality that simultaneously applies to a capacity and reward estimate. Crucially, we are able to guarantee that the aforementioned confidence regions hold collectively over all time steps by an application of Doob's inequality. We demonstrate that the method returns an ϵ-optimal solution to the stochastic knapsack problem with high probability. To the best of our knowledge, our algorithm is the first which provides such guarantees for the stochastic knapsack problem. Furthermore, our algorithm is an anytime algorithm and will return a good solution even if stopped prematurely. This is particularly important given the difficulty of the problem. We also provide theoretical conditions to guarantee `OpStoK` does not expand all policies and demonstrate favorable performance in a simple experimental setting.

1 Introduction

The stochastic knapsack problem (Dantzig, 1957), is a classic resource allocation problem that consists of selecting a subset of items to place into a knapsack of a given capacity. Placing each item in the knapsack consumes a random amount of the capacity and provides a stochastic reward. Many real world scheduling, investment, portfolio selection, and planning problems can be formulated as the stochastic knapsack problem. Consider, for instance, a fitness app that suggests a one hour workout to a user. Each exercise (item) will take a random amount of time (size) and burn a random amount of calories (reward). To make optimal use of the available time the app needs to track the progress of the user and adjust accordingly. Once an item is placed in the knapsack, we assume we observe its realized size and can use this to make future decisions. This enables us to consider adaptive or closed loop strategies, which will generally perform better (Dean et al., 2008) than open loop strategies in which the schedule is invariant of the remaining budget.

Finding exact solutions to the simpler deterministic knapsack problem, in which item weights and rewards are deterministic, is known to be NP-hard and it has been stated that the stochastic knapsack problem is PSPACE-hard (Dean et al., 2008). Due to the difficulty of the problem, there are currently no algorithms that are guaranteed to find satisfactory approximations in acceptable computation time. While ultimately one aims to have algorithms that can approach large scale problems, the current state-of-the-art makes it apparent that the small scale stochastic knapsack problem must be tackled first. The emphasis in this paper is therefore on this small scale stochastic knapsack setting. The current state-of-the-art approaches to the stochastic knapsack problem where the reward and resource consumption distributions are known, were introduced in Dean et al. (2008). Their algorithm groups the available items into small and large items and fills the knapsack exclusively with items of one of the two groups, ignoring potential high reward items in the other group, but still returning a policy that comes within a factor of $1/(3+\kappa)$ of the optimal, where $\kappa > 0$ is used to set the size of the small items. The strategy for small items is non-adaptive and orders the items according to their reward - consumption ratio, placing items into the knapsack according to this ordering. For the large items, a de-

Preliminary work. Under review by AISTATS 2017. Do not distribute.

cision tree is built to some predefined depth d and an exhaustive search for the best policy in that decision tree is performed. For most non-trivial problems, this tree can be exceptionally large. The notion of small items is also underlying recent work in machine learning where the reward and consumption distributions are assumed to be unknown (Badanidiyuru et al., 2015). The approach in Badanidiyuru et al. (2015) works with a knapsack size that converges (in a suitable way) to infinity, rendering all items small. The stochastic knapsack problem is also a generalization of the the pure exploration combinatorial bandit problem, eg Chen et al. (2014); Gabillon et al. (2016)

It is desirable to have methods for the stochastic knapsack problem that can make use of all available resources and adapt with the remaining capacity. For this, the tree structure from Dean et al. (2008) can be useful. We propose using ideas from optimistic planning (Busoniu and Munos, 2012; Szörényi et al., 2014) to significantly accelerate the tree search approach and find adaptive strategies. Most optimistic planning algorithms were developed for discounted MDPs and as such rely on discount factors to limit future reward, effectively reducing the search tree to a tree with small depth. However, these discount factors are not present in the stochastic knapsack problem. Furthermore, in our problem, the random variable representing state transitions also provides us with information on the future rewards. To avoid the use of discount factors and use the transition information, we work with confidence bounds that incorporate estimates of the remaining capacity and use these estimates to determine how many samples we need. For this, we need techniques that can deal with weak dependencies and that give us confidence regions that hold simultaneously for multiple sample sizes. We therefore combine Doob's martingale inequality with Azuma-Hoeffding bounds to create our high probability bounds. Following the optimistic planning approach, we use these bounds to develop an algorithm that adapts to the complexity of the problem instance: in contrast to the current state-of-the-art, it is guaranteed to find an ϵ-good approximation independent of how difficult the problem is and, if the problem instance is easy to solve, it expands only a moderate sized tree. Our algorithm is also an 'anytime' algorithm in the sense that, it improves rapidly to begin with and if stopped prematurely, it will still return a good solution. For our algorithm, we only require access to a generative model of item sizes and rewards, and no further knowledge of the distributions.

We measure the performance of our algorithm in terms of the number of policies it expands. In a theoretical manner we define the set of ϵ-critical policies to be the set of policies an algorithm may expand to obtain a solution within ϵ of the optimal. We also show that, in practice, the number of policies explored by our algorithm `OpStoK` is small and compares favorably to that of the algorithm from Dean et al. (2008).

1.1 Related work

Due to the difficulty of the stochastic knapsack problem, the main approximation algorithms focus on the variant of the problem with deterministic sizes and stochastic rewards (eg. Steinberg and Parks (1979) and Morton and Wood (1998)), or stochastic sizes and deterministic rewards (eg. Dean et al. (2008) and Bhalgat et al. (2011)). Of these, the most relevant to us are Dean et al. (2008) and Bhalgat et al. (2011) where decision trees are used to obtain approximate adaptive solutions to the problem. To limit the size of the decision tree Dean et al. (2008) use a greedy strategy for 'small' items while Bhalgat et al. (2011) group items together in blocks. Morton and Wood (1998) use a Monte-Carlo sampling strategy to generate a non-adaptive (open loop) solution in the case with stochastic rewards and deterministic sizes.

The bandits with knapsacks problem of Badanidiyuru et al. (2015) is different to ours since it does not require access to a generative model of item sizes and rewards but learns the distributions by playing items multiple times. This requires a large budget and the resulting strategies are not adaptive. In Burnetas et al. (2015) adaptive strategies are considered for deterministic item sizes and renewable budgets.

The UCT style of bandit based tree search algorithms (Kocsis and Szepesvári, 2006) use upper confidence bounds at each node of the tree to select the best action. UCT has been shown to work well in practice, however, it may be too optimistic (Coquelin and Munos, 2007) and theoretical results on the performance have proved difficult to obtain. Optimistic planning was developed for tree search in large deterministic (Hren and Munos, 2008) and stochastic systems, both open (Bubeck and Munos, 2010) and, closed loop (Busoniu and Munos, 2012). The general idea is to use the upper confidence principle of the UCB algorithm for multi-armed bandits (Auer et al., 2002) to expand a tree. This is achieved by expanding nodes (states) that have the potential to lead to good solutions by using bounds that take into account both the reward received in getting to a node and the reward that could be obtained after moving on from that node. The closest work to ours is Szörényi et al. (2014) who use optimistic planning in discounted MDPs, requiring only a generative model of the rewards and transitions. Instead of the UCB algorithm, like ours, their work relies on the best arm identification algo-

rithm of Gabillon et al. (2012).

There are several key differences between our problem and the MDPs optimistic planning algorithms are typically designed for. Generally, in optimistic planning it is assumed that the state transitions do not provide any information about future reward. However, in our problem this information is relevant and should be considered when defining the high confidence bounds. Furthermore, optimistic planning algorithms are used to approximate complex systems at just one point and return a near optimal first action. In our case, the decision tree is a good approximation to the entire problem so we can output a near-optimal policy. Furthermore, to the best of our knowledge, our algorithm is the first optimistic planning algorithm to iteratively build confidence bounds which are used to determine whether it is necessary to sample more. One would imagine that the `StOP` algorithm from Szörényi et al. (2014) could be easily adapted to the stochastic knapsack problem. However, as discussed in Section 4.1, the assumptions required for this algorithm to terminate are too strong for it to be considered feasible for this problem.

1.2 Our contribution

Our main contributions are the anytime algorithm `OpStoK` (Algorithm 1)) and subroutine `BoundValueShare` (Algorithm 2). These are supported by the confidence bounds in Lemma 1 and Proposition 2 that allow us to simultaneously estimate remaining capacity and reward with guarantees that hold uniformly over multiple sample sizes, and Proposition 3, which shows that we can avoid discount based arguments and still return an adaptive policy with value within ϵ of the optimal policy, with high probability and while using adaptive capacity estimates. This makes `OpStoK` the first algorithm to provably return an ϵ-optimal solution. Theorem 5 and Corollary 6 provide bounds on the number of samples our algorithm uses in terms of how many policies are ϵ-close to the best policy. The empirical performance of `OpStoK` is then discussed in Section 7.

2 Problem formulation

We consider the problem of selecting a subset of items from a set of K items, I, to place into a knapsack of capacity B where each item can be played at most once. For each item $i \in I$, let C_i and R_i be bounded random variables defined on a joint probability space $(\Omega, \mathcal{A}, P)$ which represent the size and reward of item i. It is assumed that we can simulate from the generative model of (R_i, C_i) for all $i \in I$ and we will use lower case c_i and r_i, to denote realizations of the random variables. We assume that the random variables (R_i, C_i) are independent of (R_j, C_j) for all $i, j \in I$, $i \neq j$. Further, it is believed that item sizes and rewards do not change dependent on the other items in the knapsack. We assume the problem is non-trivial, in the sense that it is not possible to fit all items in the knapsack at once. If we place an item i in the knapsack and the consumption C_i is strictly greater than the remaining capacity then we gain no reward for that item. Our final important assumption is that there exists some non-decreasing function $\Psi(\cdot)$, satisfying $\lim_{b\to 0} \Psi(b) = 0$ and $\Psi(B) < \infty$, such that the reward that can be achieved with budget b is upper bounded by $\Psi(b)$.

Representing the stochastic knapsack problem as a tree requires that all item sizes take discrete values. While in this work, it will generally be assumed that this is the case, in some problem instances, continuous item sizes need to be discretized. In this case, let ξ^* be the discretization error of the optimal policy. Then $\Psi(\xi^*)$ is an upper bound on the extra reward that could be gained from the space lost due to discretization. For discrete sizes, we assume there are s possible values the random variable can take and that there exists $\theta > 0$ such that $C_i \geq \theta$ for all $i \in I$.

2.1 Planning trees and policies

The stochastic knapsack problem can be thought of as a planning tree with the initial empty state as the root at level 0. Each node on an even level is an *action* node and its children represent placing an item in the knapsack. The nodes on odd levels are *transition* nodes with children representing item sizes. We define a *policy* Π as a finite subtree where each action node has at most one child and each transition node has s children. The *depth* of a policy Π, $d(\Pi)$, is defined as the number of transition nodes in any realization of the policy (where each transition node has one child), or equivalently, the number of items. Let $d^* = \lfloor B/\theta \rfloor$ be the maximal depth of any policy. For any $1 \leq d \leq d^*$, the number of policies of depth d is,

$$N_d = \prod_{i=0}^{d-1} (K-i)^{s^i} \tag{1}$$

where $K = |I|$ is the number of items, and s the number of discrete sizes.

We define a *child* policy, Π', of a policy Π as a policy that follows Π up to depth $d(\Pi)$ then plays additional items and has depth $d(\Pi') = d(\Pi) + 1$. In this setting, Π is the *parent* policy of Π'. A policy Π' is a *descendant* policy of Π, if Π' follows Π up to depth $d(\Pi)$ but is then continued to depth $d(\Pi') \geq d(\Pi) + 1$. Conversely, in this setting, Π is said to be an *ancestor* of Π'. A policy is said to be *incomplete* if the remaining budget allows

for another item to be inserted into the knapsack (see Section 4.2 for a formal definition).

The *value* of a policy Π can be defined as the cumulative expected reward obtained by playing items according to Π, $V_\Pi = \sum_{t=1}^{T} E[R_{i_t}]$ where i_t is the t-th item chosen by Π. Let $\mathcal{P}$ be the set of all policies, then define the *optimal policy* as $\Pi^* = \arg\max_{\Pi\in\mathcal{P}} V_\Pi$, and corresponding *optimal value* as $v^* = \max_{\Pi\in\mathcal{P}} V_\Pi$. Our algorithm returns an ϵ-*optimal* policy with value $v^* - \epsilon$. For any policy Π, we define a *sample* of Π as follows. The first item of any policy is fixed so we take a sample of the reward and size from the generative model of that item. We then use Π to tell us which item to sample next (based on the size of the previous item) and sample the reward and size of that item. This continues until the policy finishes or the cumulative sampled sizes of the selected items exceeds B.

3 High confidence bounds

In this section, we develop confidence bounds for the value of a policy. Observe that a policy Π need not consume all available budget, in fact our algorithm will construct iteratively longer policies starting from the shortest policies of playing a single item. Consequently, we are also interested in R_Π^+, the expected maximal reward that can be obtained after playing according to policy Π until all budget is consumed. Let B_Π be a random variable representing the remaining budget after playing according to a policy Π. Our assumptions guarantee that there exists a function Ψ such that $R_\Pi^+ \leq E\Psi(B_\Pi)$. We define V_Π^+ to be the maximal expected value of any continuation of policy Π so $V_\Pi^+ = V_\Pi + R_\Pi^+ \leq V_\Pi + E\Psi(B_\Pi)$.

From m samples of the reward of policy Π, we estimate the value of Π as $\overline{V_\Pi}_m = \frac{1}{m}\sum_{j=1}^{m}\sum_{d=1}^{d(\Pi)} r_{i(d)}^{(j)}$, where $r_{i(d)}^{(j)}$ is the reward of item $i(d)$ chosen at depth d of sample j. However, our real interest is in the value of V_Π^+ since we wish to identify the policy with greatest reward when continued until the budget is exhausted. From Hoeffding's inequality, $P\left(|\overline{V_\Pi}_{m_1} - V_\Pi^+| > E\Psi(B_\Pi) + \sqrt{\frac{\Psi(B)^2\log(2/\delta)}{2m}}\right) \leq \delta$. This bound depends on the quantity $E\Psi(B_\Pi)$ which is typically not known. The following lemma shows how our bound can be improved by independently sampling $\Psi(B_\Pi)$ m times to get samples $\psi_1, \ldots, \psi_m$ and estimating $\overline{\Psi(B_\Pi)}_m = \frac{1}{m}\sum_{j=1}^{m}\psi_j$.

Lemma 1 *Let $(\Omega, \mathcal{A}, P)$ be the probability space from Section 2, then for $m_1 + m_2$ independent samples of policy Π, and $\delta_1, \delta_2 > 0$, with probability $1 - \delta_1 - \delta_2$,*

$$\overline{V_\Pi}_{m_1} - k_1 \leq V_\Pi^+ \leq \overline{V_\Pi}_{m_1} + \overline{\Psi(B_\Pi)}_{m_2} + k_1 + k_2.$$

Where, $k_1 := \sqrt{\frac{\Psi(B)^2\log(2/\delta_1)}{2m_1}}$, $k_2 := \sqrt{\frac{\Psi(B)^2\log(1/\delta_2)}{2m_2}}$.

We will not use the bound in this form since our algorithm will work by sampling $\Psi(B_\Pi)$ until we are confident enough that it is small or large. This introduces weak dependencies into the sampling process so we need guarantees to hold simultaneously for multiple sample sizes, m_2. For this, we work with martingale techniques and use Azuma-Hoeffding like bounds (Azuma, 1967), similar to the technique used in Perchet et al. (2016). Specifically, in Lemma 8 (supplementary material), we use Doob's maximal inequality and a peeling argument to get Azuma like bounds for the maximal deviation of the sample mean from the expectation under boundedness. Assuming we sample the reward of a policy m_1 times and the remaining capacity m_2 times, the following key result holds.

Proposition 2 *The Algorithm* `BoundValueShare` *(Algorithm 2) returns confidence bounds,*

$$L(V_\Pi^+) = \overline{V_\Pi}_{m_1} - c_1$$
$$U(V_\Pi^+) = \overline{V_\Pi}_{m_1} + \overline{\Psi(B_\Pi)}_{m_2} + c_1 + c_2$$

which hold with probability $1 - \delta_1 - \delta_2$, where $c_1 = \sqrt{\frac{\Psi(B)^2\log(2/\delta_1)}{2m_1}}, c_2 = 2\Psi(B)\sqrt{\frac{1}{m_2}\log\left(\frac{8n}{\delta_2 m_2}\right)}$.

This upper bound depends on n, the maximum number of samples of $\Psi(B_\Pi)$. For any policy Π, the minimum width a confidence interval of $\Psi(B_\Pi)$ created by `BoundValueShare` will ever need to be is $\epsilon/4$. Taking,

$$n = \left\lceil \frac{16^2\Psi(B)^2\log(8/\delta)}{\epsilon^2} \right\rceil, \tag{2}$$

ensures that for all policies, $2c_2 \leq \epsilon/4$ when $m_2 = n$. This is a necessary condition for the termination of our algorithm, `OpStoK`, as will be discussed in Section 4.2

4 Algorithms

Before presenting our algorithm for optimistic planning of the stochastic knapsack problem, we first discuss a simple adaptation of the algorithm `StOP` from Szörényi et al. (2014).

4.1 Stochastic optimistic planning for knapsacks

One naive approach to optimistic planning in the stochastic knapsack problem is to adapt the algorithm `StOP` from Szörényi et al. (2014). We call this adaptation `StOP-K` and replace the $\frac{\gamma^d}{1-\gamma}$ discounting term used to control future rewards with $\Psi(B - d\theta)$. This is the best upper bound on the future reward that can

be achieved without using sample of item sizes. The upper bound on V_Π^+ is then $\overline{V_\Pi}_m + \Psi(B - d\theta) + c$, for m samples and confidence bound c. With this, most of the results from Szörényi et al. (2014) follow fairly naturally. Although `StOP-K` appears to be an intuitive extension of `StOP` to the stochastic knapsack setting, it can be shown that for a finite number of samples, unless $\Psi(B - \theta d^*) \leq \frac{\epsilon}{2}$, the algorithm will not terminate. As such, unless this restrictive assumption is satisfied `StOP-K` will not converge.

4.2 Optimistic stochastic knapsacks

In the stochastic knapsack problem, the process of sampling the reward of a policy involves sampling item sizes to decide which item to play next. We propose to make better use of this data by using the samples of item sizes to calculate $U(\Psi(B_\Pi))$ which is then incorporated into $U(V_\Pi^+)$. Instead of the worst case bound $\Psi(B - d\theta)$, our algorithm, `OpStoK`, uses the tighter upper bound $U(\Psi(B_\Pi))$. We also pool samples of the reward and size of items across policies, thus reducing the number of calls to the generative model. `OpStoK` benefits from an adaptive sampling scheme that reduces sample complexity and ensures that an entire ϵ-optimal policy is returned when the algorithm stops (line 5, Algorithm 1). This is achieved by using the bound in Proposition 2 and n as defined in (2).

In the main algorithm, `OpStoK` (Algorithm 1) is very similar to `StOP-K` Szörényi et al. (2014) with the key differences appearing in the sampling and construction of confidence bounds which are defined in `BoundValueShare`. `OpStoK` proceeds by maintaining a set of 'active' policies. As in Szörényi et al. (2014) and Gabillon et al. (2012), at each time step t, a policy, Π_t to expand is chosen by comparing the upper confidence bounds of the two best active policies. We select the policy with most uncertainty in the bounds since we want to be confident enough in our estimates of the near-optimal policies to say that the policy we ultimately select is better (see Figure 4, supplementary material). Once we have selected a policy, Π_t, if the stopping criteria is not met, we replace Π_t in the set of active policies with all its children. For each child policy, we use `BoundValueShare` to bound its reward. In order for all our bounds to hold simultaneously with probability greater than $1 - \delta_{0,1} - \delta_{0,2}$ (as shown in Lemma 12, supplementary material), `BoundValueShare` must be called with parameters

$$\delta_{d,1} = \frac{\delta_{0,1}}{d^*} N_{d(\Pi)}^{-1} \quad \text{and} \quad \delta_{d,2} = \frac{\delta_{0,2}}{d^*} N_{d(\Pi)}^{-1} \tag{3}$$

where N_d is the number of policies of depth d as given in (1). Our algorithm, `OpStoK` is given in Algorithm 1. The algorithm relies on `BoundValueShare` (Algorithm 2) and subroutines, `EstimateValue` (Algorithm 3, supplementary material) and `SampleBudget` (Algorithms 4, supplementary material), which sample the reward and budget of policies.

In `BoundValueShare`, we use samples of both item size and reward to bound the value of a policy. We define upper and lower bounds on the value of any extension of a policy Π as,

$$U(V_\Pi^+) = \overline{V_\Pi}_{m_1} + \overline{\Psi(B_\Pi)}_{m_2} + c_1 + c_2,$$
$$L(V_\Pi^+) = \overline{V_\Pi}_{m_1} - c_1,$$

with c_1 and c_2 as in Proposition 2. It is also possible to define upper and lower bounds on $\Psi(B_\Pi)$ with m_2 samples and confidence δ_2. From this, we can formally define a *complete* policy as a policy Π with $U(B_\Pi) = \overline{\Psi(B_\Pi)}_{m_2} + c_2 \leq \frac{\epsilon}{2}$. For complete policies, since there is very little capacity left, it is more important to get tight confidence bounds on the value of the policy. Hence, in `BoundValueShare`, we sample the remaining budget of a policy as much as is necessary to conclude whether the policy is complete or not. As soon as we realize we have a complete policy ($U(B_\Pi) \leq \epsilon/2$), we sample the value of that policy sufficiently to get a confidence interval of width less than ϵ. Then, when it comes to choosing an optimal policy to return, the confidence intervals of all complete policies will be narrow enough for this to happen. This is appropriate since, pre-specifying the number of samples may not lead to confidence bounds tight enough to select an ϵ-optimal policy. Furthermore, this method will focus sampling efforts only on promising policies that are near completion. If a complete policy is chosen as $\Pi_t^{(1)}$ in `OpStoK`, for some t, the algorithm will stop and this policy will be returned. For this to happen, we also need the stopping criterion to be checked before selecting a policy to expand. Note that in `BoundValueShare`, the reward and remaining budget must be sampled separately as we are considering closed-loop planning so the item chosen may depend on the size of the previous item, and hence the reward will depend on the instantiated item sizes. In line 6 of `BoundValueShare`, for an incomplete policy, the number of samples of the reward, m_1, is defined to ensure that the uncertainty in the estimate of V_Π is less than $u(\Psi(B)) = \min\{U(\Psi(B_\Pi)), \Psi(B)\}$, since a natural upper bound for the reward is $\Psi(B)$.

Since at each times step `OpStoK` expands a policy with best or second best upper confidence bound, the policy it expands will always have the potential to be optimal. Therefore, if the algorithm is stopped before the termination criteria is met (line 5 Algorithm 1) and the active policy with best mean reward is selected, this policy will be the best policy of those with the potential to be optimal that have already been explored

Algorithm 1: `OpStoK` $(I, \delta_{0,1}, \delta_{0,2}, \epsilon)$

Initialization: $\text{ACTIVE} = \emptyset$
forall the $i \in I$ **do**
  Π_i = policy consisting of just playing item i.
  $d(\Pi_i) = 1$
  $\delta_{1,1} = \frac{\delta_{0,1}}{d^*} N_1^{-1} \quad \delta_{1,2} = \frac{\delta_{0,2}}{d^*} N_1^{-1}$
  $(L(V_{\Pi_i}^+), U(V_{\Pi_i}^+)) =$ `BoundValueShare` $(\Pi_i, \delta_{0,1}, \delta_{0,2}, \mathcal{S}^*, \epsilon)$
  $\text{ACTIVE} = \text{ACTIVE} \cup \{\Pi_i\}$.
end
for $t = 1, 2, \ldots$ **do**
  $\Pi_t^{(1)} = \arg\max_{\Pi \in \text{Active}} U(V_\Pi^+)$
  $\Pi_t^{(2)} = \arg\max_{\Pi \in \text{Active} \setminus \{\Pi_t^{(1)}\}} U(V_\Pi^+)$
  if $L(V_{\Pi_t^{(1)}}^+) + \epsilon \geq \max_{\Pi \in \text{ACTIVE}} U(V_\Pi^+)$ **then**
    Stop: $\Pi^* = \Pi_t^{(1)}$;
  $\Pi_t = \Pi_t^{(a^*)}$, where $a^* = \arg\max_{a \in \{1,2\}} U(\Psi(B_{\Pi_t^{(a)}}))$
  $\text{ACTIVE} = \text{ACTIVE} \setminus \{\Pi_t\}$
  forall the *children* Π' *of* Π_t **do**
    $d(\Pi') = d(\Pi_t) + 1$
    $\delta_1 = \frac{\delta_{0,1}}{d^*} N_{d(\Pi')}^{-1}$ and $\delta_2 = \frac{\delta_{0,2}}{d^*} N_{d(\Pi')}^{-1}$
    $(L(V_{\Pi'}^+), U(V_{\Pi'}^+)) =$ `BoundValueShare` $(\Pi', \delta_1, \delta_2, \mathcal{S}^*, \epsilon)$
    $\text{ACTIVE} = \text{ACTIVE} \cup \{\Pi'\}$
  end
end

Algorithm 2: `BoundValueShare`$(\Pi, \delta_1, \delta_2, S^*, \epsilon)$

Input: Π: policy; δ_1: probability capacity confidence bound fails; δ_2: probability reward confidence bound fails; S^*: observed samples for all items; ϵ: tolerated approximation error.

Initialization: For all $i \in I$, let $\mathcal{S}_i = \mathcal{S}_i^*$

Set $m_2 = 1$ and $(\psi_1, \mathcal{S}) =$ `SampleBudget`$(\Pi, \mathcal{S})$
`/* draw a sample of the remaining budget */`
$\overline{\Psi(B_\Pi)}_{m_2} = \frac{1}{m_2} \sum_{j=1}^{m_2} \psi_j$
$U(\Psi(B_\Pi)) = \overline{\Psi(B_\Pi)}_{m_2} + 2\Psi(B)\sqrt{\frac{1}{m_2} \log\left(\frac{8n}{\delta m_2}\right)}$,
$L(\Psi(B_\Pi)) = \overline{\Psi(B_\Pi)}_{m_2} - 2\Psi(B)\sqrt{\frac{1}{m_2} \log\left(\frac{8n}{\delta m_2}\right)}$
`/* calculate upper and lower bounds on the remaining budget */`
if $U(\Psi(B_\Pi)) \leq \frac{\epsilon}{2}$ **then** $m_1 = \left\lceil \frac{8\Psi(B)^2 \log(2/\delta_1)}{\epsilon^2} \right\rceil$;
else if $L(\Psi(B_\Pi)) \geq \frac{\epsilon}{4}$ **then**
  $m_1 = \left\lceil \frac{1}{2} \frac{\Psi(B)^2 \log(2/\delta_1)}{u(\Psi(B))^2} \right\rceil$
else
  Set $m_2 = m_2 + 1$, $(\psi_{m_2}, \mathcal{S}) =$ `SampleBudget`$(\Pi, \mathcal{S})$ and go back to 2
$\overline{V_\Pi}_{m_1} =$ `EstimateValue`(Π, m_1)
$L(V_\Pi^+) = \overline{V_\Pi}_{m_1} - \sqrt{\frac{\Psi(B)^2 \log(2/\delta_1)}{2m_1}}$
$U(V_\Pi^+) = \overline{V_\Pi}_{m_1} + \overline{\Psi(B_\Pi)}_{m_2} + \sqrt{\frac{\Psi(B)^2 \log(2/\delta_1)}{2m_1}} + 2\Psi(B)\sqrt{\frac{1}{m_2} \log\left(\frac{8n}{\delta m_2}\right)}$
return $(L(V_\Pi^+), U(V_\Pi^+))$

and so will be a good policy (or beginning of policy). `OpStoK`, also considerably reduces the number of calls to the generative model by creating sets $\mathcal{S}_i^*$ of samples of the reward and size of each item $i \in I$. When it is necessary to sample the reward and size of an item for the evaluation of a policy, we sample without replacement from $\mathcal{S}_i^*$, until $|\mathcal{S}_i^*|$ samples have been taken. At this point new calls to the generative model are made and the new samples added to the sets for use by future policies. This is illustrated in `EstimateValue` (Algorithm 3, supplementary material) and `SampleBudget` (Algorithm 4, supplementary material). We denote by $\mathcal{S}^*$ the collection of all sets $\mathcal{S}_i^*$.

5 ϵ-critical policies

The set of ϵ-critical policies is the set of all policies an algorithm may potentially expand in order to obtain an ϵ-optimal solution. The number of ϵ-critical policies in this set represents a bound on the number of policies an algorithm may explore in order to obtain this ϵ-optimal solution.

To define the set of ϵ-critical policies associated with `OpStoK`, let

$$\mathcal{Q}_{IC}^\epsilon = \{\Pi; V_\Pi + 6E\Psi(B_\Pi) - {}^{3\epsilon}/_4 \geq v^* \\ -6E\Psi(B_\Pi) + {}^{3\epsilon}/_4 + \epsilon\}$$

and $\mathcal{Q}_C^\epsilon = \{\Pi; V_\Pi + \epsilon \geq v^*\}$,

represent the set of potentially optimal incomplete and complete policies. The set of all ϵ-critical policies is then $\mathcal{Q}^\epsilon = \mathcal{Q}_{IC}^\epsilon \bigcup \mathcal{Q}_C^\epsilon$. The following lemma then shows that all policies expanded by `OpStoK` are in $\mathcal{Q}^\epsilon$.

Lemma 3 *For any policy* $\Pi \in$ ACTIVE *assume that* $L(V_\Pi^+) \leq V_\Pi \leq U(V_\Pi^+)$ *holds simultaneously for all policies in the active set with* $U(V_\Pi^+)$ *and* $L(V_\Pi^+)$ *as defined in Proposition 2. Then,* $\Pi_t \in \mathcal{Q}^\epsilon$ *at every time point* t *considered by the algorithm* `OpStoK`*, except for possibly the last one.*

We now turn to demonstrating that under certain con-

ditions, OpStoK will not expand all policies (although in practice this claim should hold even when some of the assumptions are violated). From considering the definition of $\mathcal{Q}^\epsilon_{IC}$ from Section 6, it can be shown that if there exists a subset I' of items and $\lambda > 0$ satisfying,

$$\sum_{i\in I'} E[R_i] < v^* - \epsilon, \quad \text{and,}$$
$$E\left[\Psi\left(B - \sum_{i\in I'} C_i\right)\right] < \frac{5\epsilon}{24} + \frac{\lambda}{12} \tag{4}$$

then $\mathcal{Q}^\epsilon_{IC}$ is a proper subset of all incomplete policies and as such, not all incomplete policies will need to be evaluated by OpStoK. Furthermore, since any policy of depth $d > 1$ will only be evaluated by OpStoK if a descendant of it has previously been evaluated, it follows that a complete policy in $\mathcal{Q}^\epsilon_C$ must have an incomplete descendant in $\mathcal{Q}^\epsilon_{IC}$. Therefore, since $\mathcal{Q}^\epsilon_{IC}$ is not equal to the set of all incomplete policies, $\mathcal{Q}^\epsilon_C$ will also be a proper subset of all complete policies and so $\mathcal{Q}^\epsilon \subsetneq \mathcal{P}$. Note that the bounds used to obtain these conditions are worst case as they involve assuming the true value of $\Psi(B_\pi)$ lies at one extreme of the confidence interval. Hence, even if the conditions in (4) are not satisfied, it is unlikely that OpStoK will evaluate all policies.

The conditions in (4) are easily satisfied. Consider, for example, the problem instance where $\epsilon = 0.05, \Psi(b) = b \quad \forall 0 \le b \le B, v^* = 1$ and $B = 1$. Assume there are 3 items $i_1, i_2, i_3 \in I$ with $E[R_i] < 1/8$ and $E[C_i] = 8/25$. Then if $I' = \{i_1, i_2, i_3\}$ and $\lambda = 5/8$, the conditions of (4) are satisfied and OpStoK will not evaluate all policies.

6 Analysis

In this section we state some theoretical guarantees on the performance of OpStoK with the proofs of all results given in Appendix C.2. We begin with the consistency result:

Proposition 4 *With probability at least* $(1 - \delta_{0,1} - \delta_{0,2})$*, the algorithm* OpStoK *returns an action with value at least* $v^* - \epsilon$ *for* $\epsilon > 0$.

To obtain a bound on the sample complexity of OpStoK, we return to the definition of ϵ-critical policies from Section 5. The set of ϵ-critical policies, $\mathcal{Q}^\epsilon$, can be represented as the union of three disjoint sets, $\mathcal{Q}^\epsilon = \mathcal{A}^\epsilon \cup \mathcal{B}^\epsilon \cup \mathcal{C}^\epsilon$, as illustrated in Figure 1 where $\mathcal{A}^\epsilon = \{\Pi \in \mathcal{Q}^\epsilon | E\Psi(B_\Pi) \le \epsilon/4\}, \mathcal{B}^\epsilon = \{\Pi \in \mathcal{Q}^\epsilon | E\Psi(B_\Pi) \ge \epsilon/2\}$ and $\mathcal{C}^\epsilon = \{\Pi \in \mathcal{Q}^\epsilon | \epsilon/4 < E\Psi(B_\Pi) < \epsilon/2\}$. Using this, in Theorem 5 the total number of samples of item size or reward required by OpStoK can be bounded as follows.

Theorem 5 *With probability greater than* $1-\delta_{0,2}$*, the total number of samples required by* OpStoK *is bounded from above by,*

$$\sum_{\Pi\in\mathcal{Q}^\epsilon} (m_1(\Pi) + m_2(\Pi))\, d(\Pi).$$

Where, for $\Pi \in \mathcal{A}^\epsilon, m_1(\Pi) = \left\lceil 8\Psi(B)^2 \log(\frac{2}{\delta_{d(\Pi),1}})/\epsilon^2 \right\rceil$,
for $\Pi \in \mathcal{B}^\epsilon, m_1(\Pi) \le \left\lceil \Psi(B)^2 \log(\frac{2}{\delta_{d(\Pi),1}})/2E\Psi(B_\Pi)^2 \right\rceil$,
and for $\Pi \in \mathcal{C}^\epsilon, m_1(\Pi) \le \max\left\{ \left\lceil 8\Psi(B)^2 \log(\frac{2}{\delta_{d(\Pi),1}})/\epsilon^2 \right\rceil, \left\lceil 2\Psi(B)^2 \log(\frac{2}{\delta_{d,1}})/E\Psi(B_\Pi)^2 \right\rceil \right\}$.

And $m_2(\Pi) = m^*$*, where* m^* *is the smallest integer satisfying,*

$$32\Psi(B)^2/(E\Psi(B_\Pi) - \epsilon/2)^2 \le m/\log(4n/m\delta_2) \text{ for } \Pi \in \mathcal{A}^\epsilon,$$
$$32\Psi(B)^2/(E\Psi(B_\Pi) - \epsilon/4)^2 \le m/\log(4n/m\delta_2) \text{ for } \Pi \in \mathcal{B}^\epsilon,$$
$$32\Psi(B)^2/(\epsilon/4)^2 \le m/\log(4n/m\delta_2) \text{ for } \Pi \in \mathcal{C}^\epsilon.$$

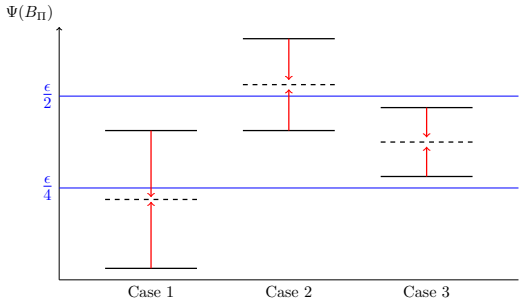

Figure 1: The three possible cases of $E\Psi(B_\Pi)$. In the first case, $E\Psi(B_\Pi) \le \frac{\epsilon}{4}$ so $\Pi \in \mathcal{A}^\epsilon$, in the second case $E\Psi(B_\Pi) \ge \frac{\epsilon}{2}$ so $\Pi \in \mathcal{B}^\epsilon$, and in the final case $\frac{\epsilon}{4} < E\Psi(B_\Pi) < \frac{\epsilon}{2}$ so $\Pi \in \mathcal{C}^\epsilon$.

In order to bound the number of calls to the generative model, we consider the expected number of times item i needs to be sampled by a policy Π. Let $i_1, \ldots, i_q$ denote the q nodes in policy Π where item i is played. Then for each node $i_k (1 \le k \le q)$, denote by ζ_{i_k} the unique route to node i_k. Define $d(\zeta_{i_k})$ to be the depth of node i_k, or the number of items played along route ζ_{i_k}. Then the probability of reaching node i_k (or taking route ζ_{i_k}) is $P(\zeta_{i_k}) = \prod_{\ell=1}^{d(\zeta_{i_k})} p_{\ell,\Pi}(i_{k,\ell})$, where $i_{k,\ell}$ denotes the ℓth item on the route to item i_k and, $p_{l,\Pi}(i_j)$ is the probability of choosing item i_j at depth l of policy Π. Denote the probability of playing item i in policy Π by $P_\Pi(i)$, then, $P_\Pi(i) = \sum_{k=1}^{q} P(\zeta_{i_k})$. Using this, the expected number of samples of the reward and size of item i required by policy Π are less than $m_1(\Pi)P_\Pi(i)$, and $m_2(\Pi)P_\Pi(i)$ respectively. Since samples are shared between policies, the expected number of calls to the generative model of item i is as given below and used in Corollary 6,

$$M(i) \le \max_{\Pi\in\mathcal{Q}^\epsilon} \left\{ \max\{m_1(\Pi)P_\Pi(i), m_2(\Pi)P_\Pi(i)\} \right\}.$$

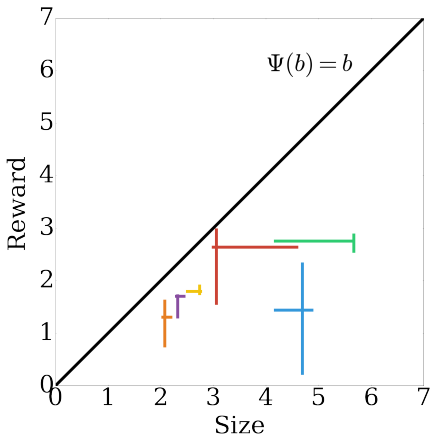

Figure 2: Item sizes and rewards. Each color represents an item with horizontal lines between the two possible sizes and vertical lines between minimum and maximum reward. The lines cross at the point (mean size, mean reward).

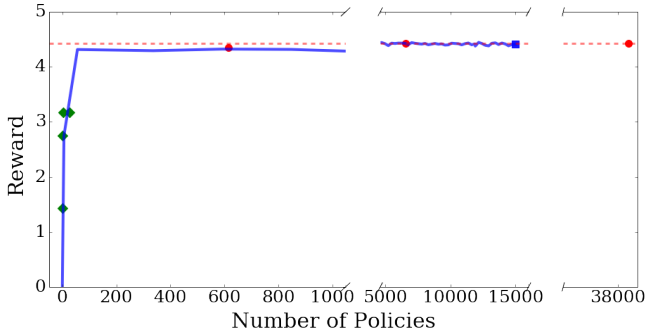

Figure 3: Num policies vs reward. The blue line is the best reward of the best policy so far found by `OpStoK` with a square where it terminates. The green diamonds are the best reward for the algorithm from Dean et al. (2008) when small items are chosen, and red circles when it chooses large items. The mean reward of the best solution from Dean et al. (2008) is given by the red dashed line.

Corollary 6 *The expected total number of calls to the generative model by* `OpStoK` *for a stochastic knapsack problem of K items is bounded from above by $\sum_{i=1}^{K} M(i)$.*

7 Experimental results

We demonstrate the performance of `OpStoK` on a simple experimental setup with 6 items. Each item i can take two sizes with probability x_i, and the rewards come from scaled and shifted Beta distributions. The budget is 7 meaning that a maximum of 3 items can be placed in the knapsack. We take $\Psi(b) = b$ and set the parameters of the algorithm to $\delta_{0,1} = \delta_{0,2} = 0.1$ and $\epsilon = 0.5$. Figure 2 illustrates the problem.

We compare the performance of `OpStoK` in this setting to the algorithm in Dean et al. (2008) with various values of κ, the parameter used to define the small items limit. We chose κ to ensure that we consider all cases from 0 small items to 6 small items. Note that the algorithm in Dean et al. (2008) is designed for deterministic rewards so in order to apply it to our problem, we sampled the rewards for each item at the start and then used the estimates as true rewards. When it came to evaluating the value of a policy, we re-sampled the final policies as discussed in Section 2.1. The results of this experiment are shown in Figure 3. From this, the anytime property of our algorithm can be seen; it is able to find a good policy early on (after less than 100 policies) so if it was stopped early, it would still return a policy with a high expected reward. Furthermore, at termination, the algorithm is very close to the best solution from Dean et al. (2008) which required more than twice as many policies to be evaluated. Thus this experiment has shown that our algorithm not only returns a policy with near optimal value, it does this after evaluating significantly fewer policies and can even be stopped prematurely to return a good policy.

These experimental results were obtained using the `OpStoK` algorithm as stated in Algorithm 1. This algorithm incorporates the sharing of samples between policies and preferential sampling of complete policies to improve performance. For large problems, the computational performance of `OpStoK` can be further improved by parallelization. In particular, the expansion of a policy can be done in parallel with each leaf of the policy being expanded on a different core and then recombined. It is also possible to sample the reward and remaining budget of a policy in parallel.

8 Conclusion

In this paper we have presented a new algorithm `OpStoK`, an anytime optimistic planning algorithm specifically tailored to the stochastic knapsack problem. For this algorithm, we provide confidence intervals, consistency results, bounds on the sample size and show that it needn't evaluate all policies to find an ϵ-optimal solution; making it the first such algorithm for the stochastic knapsack problem. By using estimates of the remaining budget and reward, `OpStok` is adaptive and also benefits from a unique sampling scheme. While `OpStoK` was developed for the stochastic knapsack problem, it is hoped that it is just the first step towards using optimistic planning to tackle many frequently occurring resource allocation problems.

References

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

K. Azuma. Weighted sums of certain dependent random variables. *Tohoku Math. J. (2)*, 1967.

A. Badanidiyuru, R. Kleinberg, and A. Slivkins. Bandits with knapsacks. In *arXiv preprint arXiv:1305.2545*, 2015.

A. Bhalgat, A. Goel, and S. Khanna. Improved approximation results for stochastic knapsack problems. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1647–1665. SIAM, 2011.

S. Bubeck and R. Munos. Open loop optimistic planning. In *Conference on Learning Theory*, pages 477–489, 2010.

A. N. Burnetas, O. Kanavetas, and M. N. Katehakis. Asymptotically optimal multi-armed bandit policies under a cost constraint. *arXiv preprint arXiv:1509.02857*, 2015.

L. Busoniu and R. Munos. Optimistic planning for markov decision processes. In *15th International Conference on Artificial Intelligence and Statistics*, volume 22, pages 182–189, 2012.

S. Chen, T. Lin, I. King, M. R. Lyu, and W. Chen. Combinatorial pure exploration of multi-armed bandits. In *Advances in Neural Information Processing Systems*, pages 379–387, 2014.

P.-A. Coquelin and R. Munos. Bandit algorithms for tree search. *arXiv preprint arXiv:cs/0703062*, 2007.

G. B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957.

B. C. Dean, M. X. Goemans, and J. Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008.

E. Even-Dar, S. Mannor, and Y. Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *The Journal of Machine Learning Research*, 7:1079–1105, 2006.

V. Gabillon, M. Ghavamzadeh, and A. Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Advances in Neural Information Processing Systems*, pages 3212–3220, 2012.

V. Gabillon, A. Lazaric, M. Ghavamzadeh, R. Ortner, and P. Barlett. Improved learning complexity in combinatorial pure exploration bandits. In *19th International Conference on Artificial Intelligence and Statistics*, pages 1004–1012, 2016.

A. Garivier and E. Kaufmann. Optimal best arm identification with fixed confidence. *arXiv preprint arXiv:1602.04589*, 2016.

J.-F. Hren and R. Munos. Optimistic planning of deterministic systems. In *Recent Advances in Reinforcement Learning*, pages 151–164. Springer, 2008.

L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293. 2006.

D. P. Morton and R. K. Wood. *On a stochastic knapsack problem and generalizations*. Springer, 1998.

V. Perchet, P. Rigollet, S. Chassang, and E. Snowberg. Batched bandit problems. *The Annals of Statistics*, 44(2):660–681, 2016.

A. Sabharwal, H. Samulowitz, and C. Reddy. Guiding combinatorial optimization with uct. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 356–361. Springer, 2012.

E. Steinberg and M. Parks. A preference order dynamic program for a knapsack problem with stochastic rewards. *Journal of the Operational Research Society*, pages 141–147, 1979.

B. Szörényi, G. Kedenburg, and R. Munos. Optimistic planning in markov decision processes using a generative model. In *Advances in Neural Information Processing Systems*, pages 1035–1043, 2014.

D. Williams. *Probability with martingales*. Cambridge university press, 1991.