Optimistic Planning for the Stochastic Knapsack Problem

Ciara Pike-Burke
Lancaster University

Steffen Grünewälder
Lancaster University

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 capacity and reward estimates. 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 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 items chosen are invariant of the remaining budget. We assume that we do not know the reward and size distributions of the items but are able to sample these from a generative model.

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 size distributions are known, were introduced in Dean et al. (2008). Their algorithm splits the items into small and large items and fills the knapsack exclusively with items of one of the two groups, ignoring potentially good items in the other group. This returns a solution that comes within a factor of $1/(3+\kappa)$ of the optimal, where $\kappa > 0$ is used to set a threshold for the small items. The strategy for small items is non-adaptive and places items in the knapsack accord-

Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54. Copyright 2017 by the author(s).

ing to their reward - consumption ratio. For the large items, a decision tree is built to some predefined depth and an exhaustive search for the best solution 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., 2013). The approach in Badanidiyuru et al. (2013) works with a knapsack size that converges (in a suitable way) to infinity, rendering all items small. In Burnetas et al. (2015) adaptive strategies are considered for deterministic item sizes and renewable capacities. The stochastic knapsack problem is also a generalization of the pure exploration combinatorial bandit problem 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 to 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 rewards, 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 variables representing state transitions (item sizes) also provide us with information on the remaining capacity which relates to possible 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. We also use these estimates to determine how many samples we need from the generative model of the reward/size of an item. For this, we need techniques that can deal with weak dependencies and give confidence regions that hold simultaneously for multiple sample sizes. We therefore combine Doob's martingale inequality (Doob, 1990) with Azuma-Hoeffding bounds (Azuma, 1967) 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 for all problem instances and, if the problem instance is easy to solve, it expands only a moderate sized tree. Our algorithm, `OpStoK`, is also an 'anytime' algorithm in the sense that it improves rapidly to begin with and, even if stopped prematurely, it will still return a good solution. For `OpStoK`, we only require access to a generative model of item sizes and rewards, and no further knowledge of the distributions.

A solution to the stochastic knapsack problem will take the form of a policy. A policy can be thought of as a sub-tree or a set of rules telling us which item to play next depending on previous item sizes (see supplementary material for examples). We define the value of policy to be its expected cumulative reward and seek to find policies whose value is within ϵ of the optimal value. The performance of our algorithm is measured in terms of the number of policies it expands in order to find such an ϵ-optimal policy, since this quantity relates to the run-time and complexity. In practice, the number of policies explored by our algorithm `OpStoK` is small and compares favorably to 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)), where the relevant distributions are known. 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 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. Morton and Wood (1998) use a Monte-Carlo sampling strategy to generate a non-adaptive solution in the case with stochastic rewards and deterministic sizes.

The UCT style of bandit based tree search algorithms (Kocsis and Szepesvári, 2006) uses upper confidence bounds at each node of the tree to select the best action. UCT has been shown to work 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 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 algorithm 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 the stochastic knapsack problem this information is relevant and should be taken into account when defining the high confidence bounds. Furthermore, optimistic planning algorithms are typically used to approximate complex systems at just one point and so only return a near optimal first action. In our case, the decision tree is a good approximation to the entire problem, so we 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 Proposition 2 that allow us to simultaneously estimate remaining capacity and value with guarantees that hold uniformly over multiple sample sizes. Proposition 4 shows how we can avoid discount based arguments and use adaptive capacity estimates in our algorithm, and still return an adaptive policy whose value comes within ϵ of the optimal policy with high probability. 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 considered in Section 7.

2 PROBLEM FORMULATION

We consider the problem of selecting a subset of items from a set, I, of K items, to place into a knapsack of capacity (or budget) B where each item can be played at most once. For each item $i \in I$, let C_i and R_i be non-negative, bounded random variables defined on a joint probability space $(\Omega, \mathcal{A}, P)$ which represent its size and reward. 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 these 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 a known, non-decreasing function $\Psi(\cdot)$, satisfying $\lim_{b \to 0} \Psi(b) = 0$ and $\Psi(B) < \infty$, such that the total reward that can be achieved with budget b is upper bounded by $\Psi(b)$. It will always be possible to define such a Ψ, however, the choice of Ψ will impact the performance of the algorithm, so we will choose it to be as tight as possible.

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 C_i 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. The branches from the root represent playing an item. Similarly, each node on an even level is an *action* node and its branches represent placing an item in the knapsack. The nodes on odd levels are *transition* nodes with branches representing item sizes. We define a *policy* Π as a finite subtree where each action node has at most one branch from it and each transition node has s branches (see supplementary material for examples). The *depth* of a policy Π, $d(\Pi)$, is the number of transition nodes in any realization of the policy (where each transition node links to one branch), 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,

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

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 $d(\Pi') = d(\Pi) + 1$. We say 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 $d(\Pi') \geq d(\Pi) + 1$. Correspondingly, we say Π is an *ancestor* of Π'. A policy is said to be *incomplete* if the remaining capacity allows for another item to be inserted into the knapsack (see Section 4.2 for a formal definition). Note that the policy outputted by an algorithm may be incomplete, as it could be that any continuation of it is optimal.

The *(expected) value* of a policy Π is defined as the cumulative expected reward obtained by playing items according to Π, $V_\Pi = \sum_{d=1}^{d(\Pi)} E[R_{i(d)}]$ where $i(d)$ is the d-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 and the observed size of the previous item to tell us which item to sample next 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 order to select policies to expand, we require confidence bounds for the value of a continuation of a policy. A policy Π may not consume all available budget, and our algorithm will work by constructing iteratively longer policies, starting from the shortest policies of playing a single item. Consequently, we are interested in R_Π^+, the expected maximal extra reward that can be obtained after playing according to policy Π until all the budget is consumed. Let B_Π be a random variable representing the remaining budget after playing policy Π. Our assumptions guarantee that there exists a function Ψ such that $R_\Pi^+ \leq E\Psi(B_\Pi)$. We then 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_1 samples of the value of policy Π, we estimate the true value of Π as $\overline{V_\Pi}_{m_1} = \frac{1}{m_1}\sum_{j=1}^{m_1}\sum_{d=1}^{d(\Pi)} r_{i(d)}^{(j)}$, where $r_{i(d)}^{(j)}$ is the reward of item $i(d)$ chosen at depth d of sample j. However, we wish to identify the policy with greatest value when continued until the budget is exhausted, so our real interest is in the value of V_Π^+. From Hoeffding's inequality, $P\left(|\overline{V_\Pi}_{m_1} - V_\Pi^+| > E\Psi(B_\Pi) + \sqrt{\frac{\Psi(B)^2\log(2/\delta)}{2m_1}}\right) \leq \delta$. This bound depends on the quantity $E\Psi(B_\Pi)$ which is typically not known. Lemma 1 shows how this bound can be significantly improved by independently sampling B_Π m_2 times to get samples $\psi_1, \cdots, \psi_{m_2}$ of $\Psi(B_\Pi)$ and estimating $\overline{\Psi(B_\Pi)}_{m_2} = \frac{1}{m}\sum_{j=1}^{m_2}\psi_j$.

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 sample $\Psi(B_\Pi)$ until we are sufficiently confident 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 martingales 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 (Doob, 1990) and a peeling argument to get bounds on the maximal deviation of $\overline{\Psi(B_\Pi)}_{m_2}$ from its expectation. Assuming we sample the value of a policy m_1 times and the remaining budget 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 $\Psi(B_\Pi)$ will ever need to be is $\epsilon/4$. Hence, taking

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

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 samples 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 `OpStoK` we aim to be more efficient by only exploring promising policies and making better use of all information. In the stochastic knapsack problem, in order to sample the value of a policy, we must sample item sizes to decide which item to play next. We propose to make better use of these samples by calculating $U(\Psi(B_\Pi))$ from the item size samples, and then incorporating this into $U(V_\Pi^+)$. 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. The performance of this sampling strategy is guaranteed by Proposition 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` (Algorithm 2). The general intuition is that only promising policies are explored. `OpStoK` maintains 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 our estimates of the near-optimal policies to be such that we can confidently conclude that the policy we output is better (see Figure 5, supplementary material). Once we have selected a policy, Π_t, if the stopping criteria in Line 12 is not met, we replace Π_t in the set of active policies with all its children. We refer to this as *expanding* a policy. For each child policy, Π', we bound its value using `BoundValueShare` with parameters

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

where N_d is the number of policies of depth d as given in (1). This ensures that all our bounds to hold simultaneously with probability greater than $1 - \delta_{0,1} - \delta_{0,2}$ (as shown in Lemma 12, supplementary material). The algorithm stops in Line 12 and returns a policy Π^* if $L(V_{\Pi^*}^+) + \epsilon \geq \max_{\Pi \in \text{ACTIVE} \setminus \{\Pi^*\}} U(V_\Pi^+)$ and we can be confident Π^* is within ϵ of optimal. `OpStoK` relies on `BoundValueShare` (Algorithm 2) and subroutines, `EstimateValue` and `SampleBudget` (Algorithms 3 and 4, supplementary material), which sample the value 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 on V_Π^+ 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, we 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 check the stopping criterion before selecting a policy to expand. Note that in `BoundValueShare`, the value and remaining budget of a policy 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 value will depend on the instantiated item sizes. For an incomplete policy, the number of samples of the value, m_1, is defined to ensure that the uncertainty in the estimate of V_Π is less than $u(\Psi(B_\Pi)) = \min\{U(\Psi(B_\Pi)), \Psi(B)\}$, since a maximal upper bound for the value of Π is $\Psi(B)$.

Since at each time step `OpStoK` expands the policy with best or second best upper confidence bound, the policy

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

```
   Initialization: $\textsc{Active} = \emptyset$.
1  for all $i \in I$ do
2      $\Pi_i$ = policy consisting of just playing item $i$;
3      $d(\Pi_i) = 1$;
4      $\delta_{1,1} = \frac{\delta_{0,1}}{d^*} N_1^{-1}, \quad \delta_{1,2} = \frac{\delta_{0,2}}{d^*} N_1^{-1}$;
5      $(L(V^+_{\Pi_i}), U(V^+_{\Pi_i}))$ = BoundValueShare
           $(\Pi_i, \delta_{1,1}, \delta_{1,2}, \mathcal{S}^*, \epsilon)$;
6      $\textsc{Active} = \textsc{Active} \cup \{\Pi_i\}$;
7  end
8  for $t = 1, 2, \ldots$ do
9      $\Pi_t^{(1)} = \arg\max_{\Pi \in \text{Active}} U(V^+_\Pi)$;
10     $\Pi_t^{(2)} = \arg\max_{\Pi \in \text{Active} \setminus \{\Pi_t^{(1)}\}} U(V^+_\Pi)$;
11     if $L(V^+_{\Pi_t^{(1)}}) + \epsilon \geq U(V^+_{\Pi_t^{(2)}})$ then
12         Stop: $\Pi^* = \Pi_t^{(1)}$;
13     $a^* = \arg\max_{a \in \{1,2\}} U(\Psi(B_{\Pi_t^{(a)}}))$;
14     $\Pi_t = \Pi_t^{(a^*)}$;
15     $\textsc{Active} = \textsc{Active} \setminus \{\Pi_t\}$
16     for all children $\Pi'$ of $\Pi_t$ do
17         $d(\Pi') = d(\Pi_t) + 1$;
18         $\delta_{d(\Pi'),1} = \frac{\delta_{0,1}}{d^*} N^{-1}_{d(\Pi')}, \delta_{d(\Pi'),2} = \frac{\delta_{0,2}}{d^*} N^{-1}_{d(\Pi')}$
19         $(L(V^+_{\Pi'}), U(V^+_{\Pi'}))$ = BoundValueShare
               $(\Pi', \delta_{d(\Pi'),1}, \delta_{d(\Pi'),2}, \mathcal{S}^*, \epsilon)$;
20         $\textsc{Active} = \textsc{Active} \cup \{\Pi'\}$;
21     end
22 end
```

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

```
   Initialization: For all $i \in I$, $\mathcal{S}_i = \mathcal{S}_i^*$.
1  Set $m_2 = 1$ and $(\psi_1, \mathcal{S})$ = SampleBudget$(\Pi, \mathcal{S})$;
   /* sample the remaining budget                */
2  $\overline{\Psi(B_\Pi)}_{m_2} = \frac{1}{m_2} \sum_{j=1}^{m_2} \psi_j$;
3  $U(\Psi(B_\Pi)) = \overline{\Psi(B_\Pi)}_{m_2} + 2\Psi(B)\sqrt{\frac{1}{m_2} \log\left(\frac{8n}{\delta m_2}\right)}$,
   $L(\Psi(B_\Pi)) = \overline{\Psi(B_\Pi)}_{m_2} - 2\Psi(B)\sqrt{\frac{1}{m_2} \log\left(\frac{8n}{\delta m_2}\right)}$;
   /* calculate bounds on $\Psi(B_\Pi)$           */
4  if $U(\Psi(B_\Pi)) \leq \frac{\epsilon}{2}$ then
   $m_1 = \left\lceil \frac{8\Psi(B)^2 \log(2/\delta_1)}{\epsilon^2} \right\rceil$;;
5  else if $L(\Psi(B_\Pi)) \geq \frac{\epsilon}{4}$ then
6      $m_1 = \left\lceil \frac{1}{2} \frac{\Psi(B)^2 \log(2/\delta_1)}{u(\Psi(B_\Pi))^2} \right\rceil$;
7  else
8      Set $m_2 = m_2 + 1$;
9      $(\psi_{m_2}, \mathcal{S})$ = SampleBudget$(\Pi, \mathcal{S})$ and go to 2
10 $\overline{V_\Pi}_{m_1}$ = EstimateValue$(\Pi, m_1)$;
11 $L(V^+_\Pi) = \overline{V_\Pi}_{m_1} - \sqrt{\frac{\Psi(B)^2 \log(2/\delta_1)}{2m_1}}$;
12 $U(V^+_\Pi) = \overline{V_\Pi}_{m_1} + \overline{\Psi(B_\Pi)}_{m_2} + \sqrt{\frac{\Psi(B)^2 \log(2/\delta_1)}{2m_1}}$
       $+2\Psi(B)\sqrt{\frac{1}{m_2} \log\left(\frac{8n}{\delta m_2}\right)}$;
13 return $(L(V^+_\Pi), U(V^+_\Pi))$
```

it expands will always have the potential to be optimal. Therefore, if the algorithm is stopped before the termination criteria is met and the active policy with best estimated value is selected, this policy will be the best of those with the potential to be optimal that have already been explored. Hence, it will be a good policy (or beginning of policy). `OpStoK` 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, i, 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` and `SampleBudget`(Algorithms 3 and 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 associated with an algorithm is the set of all policies the algorithm may potentially expand in order to obtain an ϵ-optimal solution. Hence, the number of ϵ-critical policies 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

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

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

Lemma 3 *Assume that $L(V^+_\Pi) \leq V_\Pi \leq U(V^+_\Pi)$ holds simultaneously for all policies $\Pi \in$ Active with $U(V^+_\Pi)$ and $L(V^+_\Pi)$ as defined in Proposition 2. Then, $\Pi_t \in \mathcal{Q}^\epsilon$ for every policy, Π_t, selected by `OpStoK` at every time point t, except for possibly the last one.*

We now turn to demonstrating that under certain conditions, 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. However, 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 give theoretical guarantees on the performance of OpStoK, with the proofs of all results in the supplementary material. We begin with the consistency result:

Proposition 4 *With probability at least $(1 - \delta_{0,1} - \delta_{0,2})$, the algorithm OpStoK returns a policy 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*

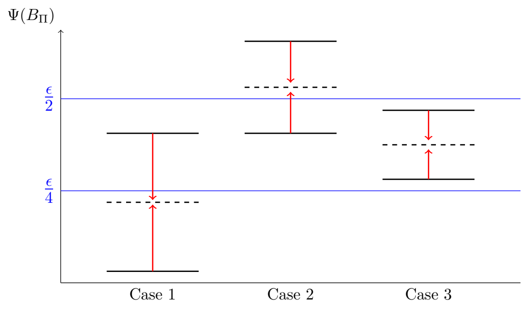

Figure 1: The three possible cases of $E\Psi(B_\Pi)$. In the first case, $E\Psi(B_\Pi) \le \frac{\epsilon}{4}$ so $\Pi \in \mathcal{A}^\epsilon$, in the second case $E\Psi(B_\Pi) \ge \frac{\epsilon}{2}$ so $\Pi \in \mathcal{B}^\epsilon$, and in the final case $\frac{\epsilon}{4} < E\Psi(B_\Pi) < \frac{\epsilon}{2}$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

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,*

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

We now bound the number of calls to the generative model required by OpStoK. We consider the expected number of times item i needs to be sampled by a policy Π. Let $i_1, \dots, i_q$ 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 node i_k and $p_{l,\Pi}(i)$ is the probability of playing item i at depth l of policy Π for given size distributions. 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,

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

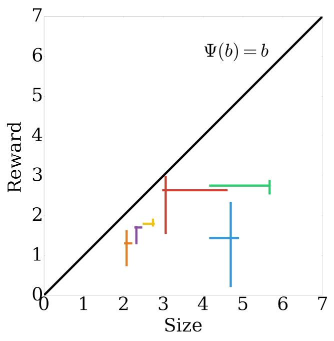

Figure 2: Item sizes and rewards. Each color is an item with horizontal lines between the two sizes and vertical lines between minimum and maximum reward. The lines cross at the point (mean size, mean reward).

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

7 EXPERIMENTAL RESULTS

We demonstrate the performance of `OpStoK` on a simple experimental setup with 6 items. Each item i can take two sizes and is larger with probability x_i. 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) run with various values of κ, the parameter used to define the small items threshold. 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 we sampled the rewards for each item at the start to get estimates of the true rewards. When sampling item sizes for Dean et al. (2008), we used the `OpStoK` sampling strategy. For both algorithms, when evaluating the value of a policy, we re-sampled the value of the chosen 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 value. Furthermore, at termination, the algorithm has almost reached 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, but it does this after evaluating significantly fewer policies and, even

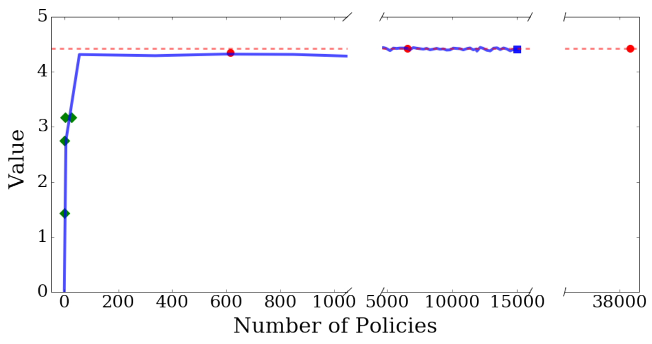

Figure 3: Num policies vs value. The blue line is the estimated value of the best policy so far found by `OpStoK` which terminates at the square. The green diamonds are the best value for Dean et al. (2008) when small items are chosen, and red circles when it chooses large items. The estimated value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

if stopped prematurely, it will 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 value and remaining budget of a policy in parallel.

8 CONCLUSION

In this paper we have presented `OpStoK`, a new anytime optimistic planning algorithm specifically tailored to the stochastic knapsack problem. For this algorithm, we have provided confidence intervals, consistency results, bounds on the sample size and shown 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 value, `OpStoK` is adaptive and also benefits from a unique streamlined 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.

Acknowledgments

We are grateful for the support of the EPSRC funded EP/L015692/1 STOR-i centre for doctoral training and Sparx.

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 Mathematical Journal, Second Series*, 19(3):357–367, 1967.

A. Badanidiyuru, R. Kleinberg, and A. Slivkins. Bandits with knapsacks. In *IEEE 54th Annual Symposium on Foundations of Computer Science*, 2013.

A. Bhalgat, A. Goel, and S. Khanna. Improved approximation results for stochastic knapsack problems. In *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*, 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. In *Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 67–74, 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.

J. L. Doob. Stochastic processes. 1990.

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.

J.-F. Hren and R. Munos. Optimistic planning of deterministic systems. In *European Workshop on Reinforcement Learning*, pages 151–164, 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. In *Advances in computational and stochastic optimization, logic programming, and heuristic search*, pages 149–168. Springer, 1998.

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

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.