# A Rollout Algorithm Framework for Heuristic Solutions to Finite-Horizon Stochastic Dynamic Programs

Justin C. Goodson* Barrett W. Thomas† Jeffrey W. Ohlmann†

September 16, 2016

**Abstract**

Rollout algorithms have enjoyed success across a variety of domains as heuristic solution procedures for stochastic dynamic programs (SDPs). However, because most rollout implementations are closely tied to specific problems, the visibility of advances in rollout methods is limited, thereby making it difficult for researchers in other fields to extract general procedures and apply them to different areas. We present a rollout algorithm framework to make recent advances in rollout methods more accessible to researchers seeking heuristic policies for large-scale, finite-horizon SDPs. We formalize rollout variants exploiting the pre- and post-decision state variables as a means of overcoming computational limitations imposed by large state and action spaces. We present a unified analytical discussion, generalizing results from the literature and introducing new results that relate the performance of the rollout variants to one another. Relative to the literature, our policy-based approach to presenting and proving results makes a closer connection to the underpinnings of dynamic programming. Finally, we illustrate our framework and analytical results via application to a dynamic and stochastic multi-compartment knapsack problem.

**Keywords:** dynamic programming; rollout algorithm; stochastic dynamic programming; approximate dynamic programming

*Corresponding Author; Department of Operations & Information Technology Management; John Cook School of Business; Saint Louis University; goodson@slu.edu; 3674 Lindell Blvd. St. Louis, MO 63108; 314.977.2027

†Department of Management Sciences; Tippie College of Business; University of Iowa; barrett-thomas@uiowa.edu; jeffrey-ohlmann@uiowa.edu

# 1 Introduction

Dynamic programs serve as a model for a variety of real-world challenges, including problems in the management of supply and distribution networks, health care delivery, energy, financial portfolios, and a range of other sequential decision problems. Yet, for many dynamic programs of practical interest, the well-known curse of dimensionality creates computational issues that impede efforts to solve the value functions and to obtain optimal policies. Hence, much research focuses on heuristic methodologies to approximate solutions to the value functions and obtain high-quality policies.

A fundamental challenge in approximate dynamic programming is identifying an optimal action to be taken from a given state. In this work, we focus on action selection via rollout algorithms, forward dynamic programming-based lookahead procedures that estimate rewards-to-go through suboptimal policies. These estimates then guide action selection in the current state. Although rollout algorithms may serve as sequential solution procedures for deterministic discrete optimization problems (Bertsekas, 2013), we focus on the application of rollout algorithms to finite-horizon stochastic dynamic programs (SDPs).

Following the initial work of Bertsekas et al. (1997), rollout algorithms applied to dynamic programs have enjoyed success across a variety of domains spanning stochastic scheduling (Bertsekas and Castanon, 1998), revenue management (Bertsimas and Popescu, 2003), multi-dimensional knapsack problems (Bertsimas and Demir, 2002), Steiner problems (Duin and Voß, 1999), sequential ordering (Guerriero and Mancini, 2003), multi-class task scheduling (Kim and Chang, 2003), job-shop scheduling (Meloni et al., 2004; Guerriero, 2008), parallel machine scheduling (Ciavotta et al., 2009; Pacciarelli et al., 2011; Ciavotta et al., 2016), and stochastic vehicle routing (Secomandi, 2000, 2001; Novoa and Storer, 2008; Goodson et al., 2013, 2016). However, because most rollout implementations are closely tied to specific problems, the visibility of advances in rollout methods is limited, thereby making it difficult for researchers in other fields to extract general procedures and apply them to different areas.

We present a rollout algorithm framework with the aim of making recent advances in rollout methods more accessible to the research community, particularly to researchers seeking heuristic solution methods for large-scale, finite-horizon SDPs. We make three contributions. First, we formalize rollout algorithm variants that exploit the pre- and post-decision state variables as a means of overcoming computational limitations imposed by large state and action spaces. In contrast, most of the rollout literature focuses on one- and multi-step procedures that often yield good policies for problems in which the state and action spaces are small enough to allow them to be applied effectively. Second, we present a unified analytical discussion of rollout algorithm

variants, generalizing results from the literature (Propositions 1, 2, and 6) and introducing new results that relate the performance of the rollout variants to one another (Propositions 3, 4, and 5). Additionally, we offer a new policy-based approach to presenting and proving performance improvement properties of rollout algorithms. Relative to the literature, which relies primarily on sample path arguments, our policy-based proofs make a stronger connection to the underpinnings of dynamic programming. Third, we illustrate our framework and analytical results via application to a dynamic and stochastic multi-compartment knapsack problem, demonstrating how one might use rollout algorithms to obtain heuristic policies for a computationally challenging problem.

In §2, we establish notation and terminology for our treatment of SDPs. We present our rollout framework in §3. Performance improvement properties are given in §4 with accompanying proofs in the Appendix. In §5, we present an illustrative application of the rollout framework to a knapsack problem and suggest guidelines for choosing a rollout algorithm in practice. We conclude the paper in §6.

# 2 A Finite-Horizon Stochastic Dynamic Program

We consider a SDP with a finite horizon in which decisions must be made at decision epochs $0, 1, \ldots, K$, where $K$ may be a random variable. The $k^{\text{th}}$ decision epoch marks the beginning of the $k^{\text{th}}$ period at which time the system occupies state $s_k$ in state space $\mathcal{S}$ and at which time the decision maker chooses an action $a$ from the set of feasible actions $\mathcal{A}(s_k)$. A state transition from state $s_k$ in decision epoch $k$ to state $s_{k+1}$ in decision epoch $k+1$ is a function of the selected action $a$ and the set of random variables $W_{k+1}$ representing the random information arriving between decision epochs $k$ and $k+1$. We denote the state transition as $s_{k+1} = S(s_k, a, W_{k+1})$. As discussed in Powell (2011), we split the state transition into two parts – a transition from pre-decision state $s_k$ to post-decision state $s_k^a$ and a transition from $s_k^a$ to pre-decision state $s_{k+1}$. We denote the deterministic transition to the post-decision state by the function $S^A(s_k, a)$ and the random transition to the next pre-decision state by the function $S^W(s_k^a, W_{k+1})$. Thus, $s_{k+1} = S(s_k, a, W_{k+1}) = S^W(S^A(s_k, a), W_{k+1})$. The decision tree in Figure 1 provides a visual representation of the model elements. Square nodes represent pre-decision states, solid arcs depict actions, round nodes are post-decision states, and dashed arcs denote random information.

Let $\hat{R}_{k+1}(s_k, a, W_{k+1})$ be the random reward earned at decision epoch $k$ when selecting action $a$ in state $s_k$ and observing random information $W_{k+1}$. Because $W_{k+1}$ may not be realized when selecting action $a$, we define the reward in decision epoch $k$ as the expected reward $R_k(s_k, a) = \mathbb{E}[\hat{R}_{k+1}(s_k, a, W_{k+1})|s_k, a]$, where $\mathbb{E}[\cdot]$ denotes the expectation operator (in this case with respect

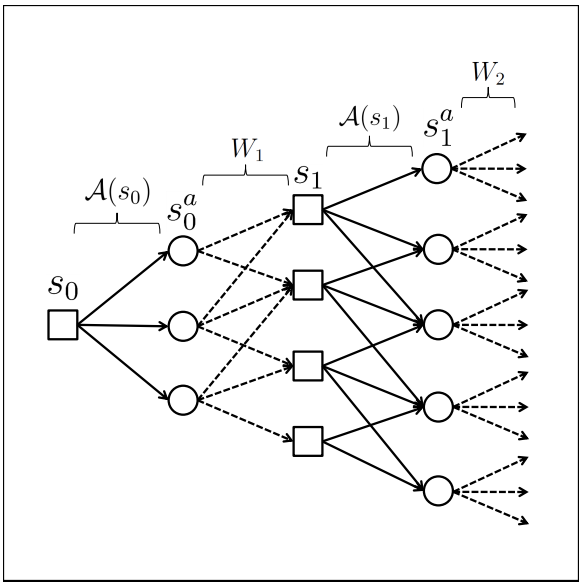


Figure 1: Stochastic Dynamic Program Depicted as a Decision Tree

to $W_{k+1}$). We define $\mathbb{P}\{\cdot|s,a\}$ as the state transition probabilities conditioned on a given state $s$ and action $a$ in $\mathcal{A}(s)$.

Let $\Pi$ be the set of all Markovian deterministic policies. A policy $\pi$ in $\Pi$ is a sequence of decision rules: $\pi = (\delta_0^\pi, \delta_1^\pi, \ldots, \delta_K^\pi)$, where each decision rule $\delta_k^\pi(s_k) : s_k \mapsto \mathcal{A}(s_k)$ is a function that specifies the action choice when the process occupies state $s_k$ and follows policy $\pi$. In the context of Figure 1, a policy indicates which action is to be selected at each decision node. Denoting by $\Delta_k$ the set of all decision rules when the process is at decision epoch $k$, a policy $\pi$ in $\Pi$ belongs to the space of decision rules $\Delta_0 \times \Delta_1 \times \cdots \times \Delta_K$. We seek a policy $\pi$ in $\Pi$ that maximizes the total expected reward, conditional on initial state $s_0$: $\mathbb{E}[\sum_{k=0}^{K} R_k(s_k, \delta^\pi(s_k))|s_0]$. Discounting can be incorporated into the reward function, but we omit discount factors to simplify notation.

# 3 Rollout Algorithm Framework

In this section, we formalize a rollout algorithm framework that provides a structure for the use of heuristic optimization techniques to solve stochastic dynamic programs. Presentation of the framework begins in §3.1, where we define *restricted policy classes*, *heuristics*, and *heuristic policies*. In §3.2, we introduce different types of decision rules that can be used in a rollout algorithm. In §3.2.1, we present and discuss the computational challenge associated with the evaluation of the *one-step decision rule* proposed in the literature. In §3.2.2 and §3.2.3, we formalize the notions of *post-* and *pre-decision state* decision rules, respectively, both designed to mitigate the compu-

tational issues faced in the evaluation of the one-step decision rule. In §3.2.4, we discuss how limitations of pre- and post-decision state decision rules may be overcome by combining the decision rules into a *hybrid* decision rule. Given these decision rules, we formally present the rollout algorithm and define a rollout policy in §3.3.

## 3.1 Restricted Policies, Heuristics, and Heuristic Policies

In this section, we formally define the methods (heuristics) for specifying the decision rules within a domain of a policy class. We begin in Definition 1 by defining *restricted policy classes*. Then, in Definition 2, we define *heuristics* and *heuristic policies*. Definition 2 is analogous to the previous definition of a heuristic (given by Bertsekas et al. (1997) for deterministic dynamic programs and Secomandi (2003) for stochastic dynamic programs). The key differences are a generalization that includes post-decision states and that we characterize a heuristic as a method that returns a policy rather than as a method that returns a set of (sample) paths from a given state to a terminal state. Because policies are feasible solutions to dynamic programs, we believe our policy-based definition makes a stronger connection to the underpinnings of dynamic programming, yielding more intuitive proofs for our analytical results. Further, as sample paths are the result of the execution of a policy and realizations of random variables, policies may be viewed as more general.

Throughout the paper, we sometimes refer to generic states $s$ without an index and to a decision epoch $k$. In these cases, $s$ may be either a pre- or post-decision state. When $s$ is a pre-decision state, $k$ is the decision epoch associated with $s$. When $s$ is a post-decision state, the decision epoch associated with $s$ is $k-1$. At times we also refer to a generic state $s'$ on a given sample path. State $s'$ may also be a pre- or post-decision state.

**Definition 1** (Restricted Policy Classes)**.** Let $\Pi$ be the set of deterministic Markovian policies. Let restricted policy class $\bar{\Pi}$ be a not necessarily strict subset of $\Pi$ in which the decision rules at each epoch $k$ lie within the restricted set of decision rules $\bar{\Delta}_k \subseteq \Delta_k$. Thus, a policy $\pi$ in $\bar{\Pi}$ is a sequence of decision rules in $\bar{\Delta}_0 \times \bar{\Delta}_1 \times \cdots \times \bar{\Delta}_K$. Let $\bar{\mathcal{A}}(s_k) = \{\delta_k(s_k) : \delta_k \in \bar{\Delta}_k\} \subseteq \mathcal{A}(s_k)$ be the restricted set of feasible actions at decision epoch $k$ when the process occupies state $s_k$ and the decision rules at epoch $k$ are restricted to be in $\bar{\Delta}_k$.

In addition to restricted policy classes, Definition 1 also introduces the concept of a restricted action set $\bar{\mathcal{A}}(s_k)$. We note that, as a subset of $\mathcal{A}(s_k)$, $\bar{\mathcal{A}}(s_k)$ may equal $\mathcal{A}(s_k)$.

**Definition 2** (Heuristics and Heuristic Policies)**.** Let $s$ be a (pre- or post-decision) state in state space $\mathcal{S}$. A heuristic $\mathcal{H}(s)$ is any method to select decision rules in epochs $k, k+1, \ldots, K$ within the space of restricted decision rules $\bar{\Delta}_k \times \bar{\Delta}_{k+1} \times \cdots \times \bar{\Delta}_K$. The resulting heuristic policy,

denoted $\pi_{\mathcal{H}(s)}$, is the sequence of these decision rules: $\pi_{\mathcal{H}(s)} = (\delta_k^{\pi_{\mathcal{H}(s)}}, \delta_{k+1}^{\pi_{\mathcal{H}(s)}}, \ldots, \delta_K^{\pi_{\mathcal{H}(s)}})$. Due to the restrictions imposed on the decision rules, policy $\pi_{\mathcal{H}(\cdot)}$ prescribes actions in restricted action space $\bar{\mathcal{A}}(\cdot)$.

Definitions 1 and 2 make a distinction between a heuristic and a heuristic policy. The heuristic can be a simple rule, a search method, or a mathematical program that is used to establish a heuristic policy. Indeed, there is not a one-to-one mapping from a heuristic to a heuristic policy. A method to select policies need not be tied to a particular policy or even to a particular class of policies. This distinction allows us to differentiate the computation time required to find the heuristic policy from the heuristic policy itself. It is the computation time of the heuristic that motivates the discussion of the post-decision state and hybrid decision rules in §3.2. Further, as demonstrated in Goodson et al. (2016), the choice of heuristic and the subsequent quality of the policy it returns can affect the quality of the rollout policy.

Definitions 1 and 2 generalize the concept of heuristic policies as they are best known in the rollout literature. Notably, some of the rollout literature refers to a "base policy" (see for example Bertsekas (2005a)). Base policies are suboptimal policies defined prior to the problem horizon. To "roll out" a base policy in future epochs, the policy must define a sequence of feasible decision rules from any given initial state (Bertsekas, 2005a). This feasibility is often achieved by modifying the base policy with recourse to account for states already visited or actions already executed (for example, see Secomandi (2001)). Our concepts of heuristics and heuristic policies are compatible with the notion of base policies applied in rollout algorithms in that the heuristic can be viewed as the recourse rule paired with a base policy to generate the resulting heuristic policy. In addition to base policies, Definitions 1 and 2 also cover the case in which, through the use of a heuristic, we update the heuristic policy at each decision epoch. Thus, instead of using recourse to modify a given base policy at each decision epoch, we allow for a heuristic that searches a given restricted policy class for an updated heuristic policy.

We illustrate Definitions 1 and 2 with a set of examples. Consider a basic job shop scheduling problem, where a set of jobs must be assigned to resources at particular times such that a given objective is optimized, e.g., minimize the makespan. The set of all non-preemptive scheduling policies is a restricted policy class. The set of restricted decision rules for non-preemptive policies does not include a decision rule that interrupts the processing of a job and then schedules it to continue processing at a later time. Thus, the restricted action space includes all feasible actions except those that preempt in-process jobs. A procedure to search the space of non-preemptive scheduling policies constitutes a heuristic. Examples of heuristics for scheduling are given in Pranzo et al. (2003) and Meloni et al. (2004).

As a second example, consider the work of Goodson et al. (2013) that develops rollout policies to obtain dynamic policies for a vehicle routing problem with stochastic demand. In their work, the restricted policy class is a class of *fixed-route* policies. A fixed-route policy requires vehicles to visit customers in the order they appear in a pre-specified sequence of customers. The restricted action space is composed of the actions resulting from the application of all possible fixed-route policies at a given decision epoch. The fixed-route local search procedure of Goodson et al. (2013), combined with their fixed-route policy class, constitutes a heuristic.

Secomandi (2008) introduces the concept of a *control algorithm*, formally defined as the pairing of a mathematical program and control-policy class to approximate the solution to a SDP. For a revenue management problem, one of the approaches Secomandi (2008) considers is restricting attention to the class of *booking-limit* policies, which are characterized by a single parameter called the booking limit. Requests for a product are accepted up to and including the booking limit, provided there is sufficient product to fill the request. Product requests beyond the booking limit are rejected. Secomandi (2008) establishes the booking limit for each product at all future decision epochs by solving a mathematical program. The instantiated booking limits serve as a policy.

Control algorithms are compatible with our coupled definitions of a restricted policy class and heuristic. Control algorithms operate on restricted policy classes and specify policy selection from these classes via mathematical programming, usually via the selection of a parameter in a threshold-like decision rule. Our definition of a heuristic does not specify a representation or solution method, only that it selects a policy from a restricted policy class. Thus, control algorithms and their use of mathematical programs fit our definition of a heuristic. Further, control algorithms may serve as heuristics to obtain the decision rules presented in §3.2.1-§3.2.4.

Finally, we note that, in Definitions 1 and 2 as well as in the rest of this paper, we focus on heuristics that specify decision rules from a current epoch $k$ to a terminal epoch $K$. In general, however, a heuristic can be applied to specify decision rules in epochs $k, k+1, \ldots, k+H$, where $H$ is some horizon $H \leq K - k$. The use of the truncated horizon is natural in rollout methods applied to infinite-horizon problems, which are often solved via rolling-horizon methods. As another example, in model predictive control, an action is identified at a given decision epoch by solving a control problem, often a deterministic approximation of the stochastic system. The solution to the control problem identifies a feasible action in the current period, but not necessarily all future periods. From the perspective of our rollout algorithm framework, model predictive control may be viewed as a limited-horizon heuristic. Bertsekas (2005b) presents a more in-depth study of connections between model predictive control and rollout algorithms. Further, as we illustrate in §5.2, methods to identify feasible actions may be used in sequence to form a full policy through the final decision epoch. Further, we note that, while in practice decision rules may be drawn from different

restricted policy classes at each epoch, the performance improvement properties in §4 assume the policy class remains the same throughout the execution of the algorithm.

## 3.2 Heuristic Decision Rules for Rollout Algorithms

Rollout algorithms step forward in time. Thus, unlike with traditional backward programming, it is necessary to estimate the reward-to-go when evaluating decision rules. In rollout algorithms, these estimates are generated by looking ahead from certain states and then applying a heuristic to generate a heuristic policy. The algorithm evaluates these heuristic policies to generate an estimate for the reward-to-go. In this section, we present decision rules, differentiated by their degree of lookahead, that will be evaluated and used to generate rollout policies in the subsequently presented rollout algorithm.

In the following sections, we discuss one-step, post-decision state, pre-decision state, and hybrid decision rules. We note a rollout algorithm typically does not directly use the actions prescribed by the heuristic policy, but rather uses the heuristic policies as means for approximating the value function and evaluating candidate actions at the current state. As noted in Chang et al. (2013, p. 197), "...what we are really interested in is the ranking of actions, not the degree of approximation. Therefore, as long as the [approximation] preserves the true ranking of actions well, the resulting policy will perform fairly well." As we will discuss, two of these decision rules incorporate heuristic policies as a means of approximating value functions and two of the decision rules use the heuristic policy directly to specify an action in the current state. Although we focus on looking no further than one step into the future, it is possible to look ahead arbitrarily far into the horizon, although such multi-step lookahead is often too computationally prohibitive.

### 3.2.1 One-Step Decision Rules

In the literature, most rollout algorithms use a decision rule that looks ahead one step and generates heuristic policies at each possible future state to approximate the reward-to-go. This one-step rollout is closely related to the pilot method of Voß et al. (2005). In this section, we formalize one-step decision rules for use in the rollout algorithm.

Let $\mathcal{S}(s_k, a) = \{s_{k+1} : \mathbb{P}\{s_{k+1}|s_k, a\} > 0\}$ be the set of reachable states when the process occupies state $s_k$ and action $a$ is selected. From each state $s_{k+1}$ in $\mathcal{S}(s_k, a)$, the one-step decision rule executes a heuristic to obtain a policy $\pi_{\mathcal{H}(s_{k+1})}$. That is, we have a heuristic policy for each $s_{k+1}$ in $\mathcal{S}(s_k, a)$ with each $s_{k+1}$ as an initial state for its respective heuristic policy. The generation of these policies is shown in Figure 2a, which depicts heuristic $\mathcal{H}(\cdot)$ being applied to each of the

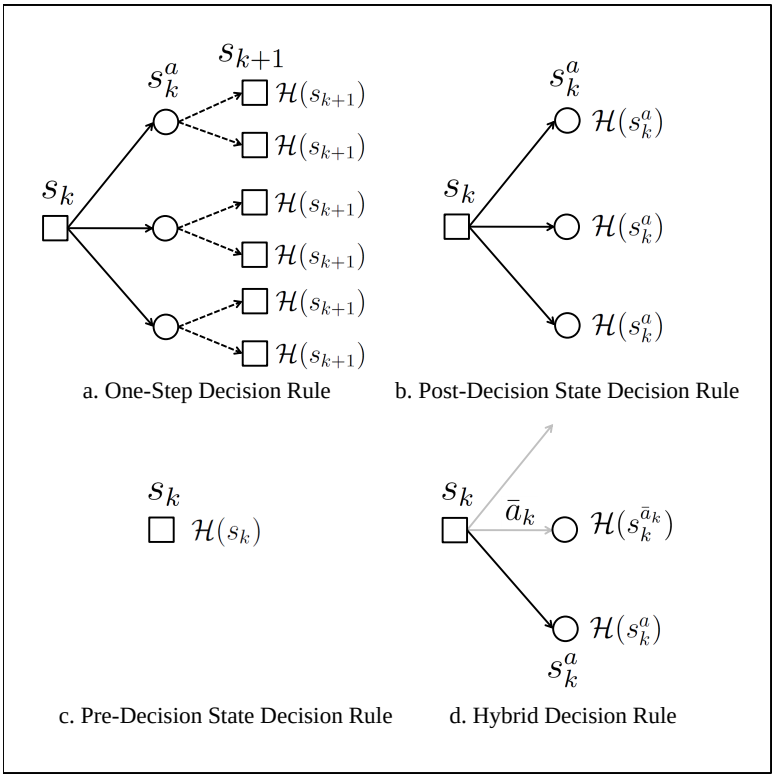


Figure 2: Rollout Algorithm Framework

six possible states at the next decision epoch $k+1$, thus resulting in six heuristic policies, one for each $s_{k+1}$.

In the one-step decision rule, the estimate of the reward-to-go when selecting action $a$ in state $s_k$ is given by the expected reward-to-go of the heuristic policies obtained in all possible states $s_{k+1}$: $\mathbb{E}[\sum_{i=k+1}^{K} R_i(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}})|s_k] = \sum_{s_{k+1}\in\mathcal{S}(s_k,a)} \mathbb{E}[\sum_{i=k+1}^{K} R_i(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}})|s_{k+1}] \times \mathbb{P}\{s_{k+1}|s_k, a\}$. The one-step decision rule selects an action $a$ in feasible action set $\mathcal{A}(s_k)$ that maximizes the value of $R_k(s_k, a) + \mathbb{E}[\sum_{i=k+1}^{K} R_i(s_i, \delta_i^{\mathcal{H}(s_{k+1})})|s_k]$. Definition 3 formalizes the decision rule employed by one-step rollout policies.

**Definition 3** (One-Step Decision Rule)**.** At decision epoch $k$ when the process occupies state $s_k$, the one-step decision rule $\delta_k^{\pi_{\text{one}}}(s_k)$ maps $s_k$ to an element in the set

$$\underset{a\in\mathcal{A}(s_k)}{\arg\max} \left\{ R_k(s_k, a) + \mathbb{E}\left[ \sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_k \right] \right\}. \tag{1}$$

Algorithm 1 illustrates the steps required to evaluate (1) and return an action. In Algorithm 1, the variables $\lambda$ and $\eta$ are local variables that facilitate calculation. The loop beginning on line 2 iterates through the feasible action set. The current-period reward is calculated on line 3. The

loop beginning on line 4 iterates through the set of reachable states. Line 5 executes the heuristic and line 6 updates the expected value of following the heuristic policy from decision epoch $k+1$ onward. The logic beginning on line 7 tracks the best action. The best action is returned on line 10 to be used in the rollout policies discussed in §3.3.

When expected values are difficult to compute via exact methods, simulation may be used to estimate the expected current-period reward on line 3 and the heuristic policy's expected reward-to-go on line 6. The simulation procedure in Bertsekas and Tsitsiklis (1996, chap. 5.2) may be applied to estimate policy values for the one-step decision rule and for the decision rules we discuss below.

**Algorithm 1** Evaluating the One-Step Decision Rule

1: $\lambda \leftarrow -\infty$
2: **for** $a \in \mathcal{A}(s_k)$ **do**
3: $\quad \eta \leftarrow R_k(s_k, a)$
4: $\quad$ **for** $s_{k+1} \in \mathcal{S}(s_k, a)$ **do**
5: $\quad\quad$ execute $\mathcal{H}(s_{k+1})$ to obtain $\pi_{\mathcal{H}(s_{k+1})}$
6: $\quad\quad \eta \leftarrow \eta + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i))|s_{k+1}\right] \times \mathbb{P}\left\{s_{k+1}|s_k, a\right\}$
7: $\quad$ **if** $\eta > \lambda$ **then**
8: $\quad\quad \lambda \leftarrow \eta$
9: $\quad\quad a^\star \leftarrow a$
10: Return $a^\star$

Algorithm 1 highlights the computational burden of evaluating a one-step decision rule. Specifically, evaluating the one-step decision rule requires $\sum_{a \in \mathcal{A}(s_k)} |\mathcal{S}(s_k, a)|$ applications of heuristic $\mathcal{H}(\cdot)$. As $|\mathcal{A}(s_k)|$ and $|\mathcal{S}(s_k, a)|$ increase, evaluating (1) becomes a computational challenge even when heuristic $\mathcal{H}(\cdot)$ is a simple procedure.

### 3.2.2 Post-Decision State Decision Rules

Several methods have been suggested for mitigating the computational burden of evaluating the one-step decision rule including restricting the action space (Guerriero et al., 2002) and parallelization (Guerriero and Mancini, 2003, 2005; Kim and Chang, 2003). In this subsection, we introduce an alternate approach that leverages the notion of the post-decision state. This paper is the first to explicitly state and analyze this decision rule and the rollout policy that results from its use.

The key computational difference between the one-step and post-decision state decision rules

is that we identify a single heuristic policy at each post-decision state rather than a heuristic policy at all possible pre-decision states in the subsequent epoch. Figure 2b depicts a post-decision state decision rule, where heuristic $\mathcal{H}(\cdot)$ is applied in each of three post-decision states. A post-decision state decision rule can be viewed as taking a "half-step" from the current state $s_k$ to a post-decision state $s_k^a$ in comparison to a one-step decision rule that looks ahead a full step from the current state $s_k$ to each possible future state $s_{k+1}$. Definition 4 formalizes the post-decision state decision rule.

**Definition 4** (Post-Decision State Decision Rule)**.** At decision epoch $k$ when the process occupies state $s_k$, the post-decision state decision rule $\delta_k^{\pi_{\text{post}}}(s_k)$ maps $s_k$ to an element in the set

$$\underset{a\in\mathcal{A}(s_k)}{\arg\max}\left\{R_k(s_k,a)+\mathbb{E}\left[\sum_{i=k+1}^{K}R_i\left(s_i,\delta_i^{\pi_{\mathcal{H}\left(s_k^a\right)}}(s_i)\right)\middle|s_k\right]\right\}. \tag{2}$$

Algorithm 2 illustrates the steps required to evaluate (2) and return an action. Line 2 iterates through the set of feasible actions. Line 3 executes the heuristic from post-decision state $s_k^a$ and line 4 calculates the value of the current-period reward plus the expected future reward of following policy $\pi_{\mathcal{H}(s_k^a)}$ from decision epoch $k+1$ onward. The logic beginning on line 5 of Algorithm 2 tracks the best action, which is returned on line 8.

A comparison of Algorithms 1 and 2 reveals that the evaluation of the post-decision state decision rule executes a heuristic $|\mathcal{A}(s_k)|$ times, which is potentially many fewer times than the $\sum_{a\in\mathcal{A}(s_k)}|\mathcal{S}(s_k,a)|$ executions required by the one-step decision rule. The complexity of evaluating the one-step decision rule depends on the size of both the state and action spaces whereas the complexity of evaluating the post-decision state decision rule depends only on the size of the action space.

**Algorithm 2** Evaluating the Post-Decision State Decision Rule

1: $\lambda \leftarrow -\infty$
2: **for** $a \in \mathcal{A}(s_k)$ **do**
3: execute $\mathcal{H}(s_k^a)$ to obtain $\pi_{\mathcal{H}(s_k^a)}$
4: $\eta \leftarrow R_k(s_k,a)+\mathbb{E}\left[\sum_{i=k+1}^{K}R_i(s_i,\delta_i^{\pi_{\mathcal{H}(s_k^a)}}(s_i))|s_k\right]$
5: **if** $\eta > \lambda$ **then**
6: $\lambda \leftarrow \eta$
7: $a^\star \leftarrow a$
8: Return $a^\star$

While flavors of post-decision state decision rules appear in the literature, only Goodson et al. (2013) explicitly acknowledges the post-decision state decision rule and post-decision rollout. To

offer the reader examples of post-decision state decision rules in practice as well as to help unify the literature, we highlight three papers from the literature. In the development of a rollout policy for a vehicle routing problem with stochastic demand, Secomandi (2001) makes use of a post-decision state decision rule without explicit mention. The use of a post-decision decision rule is evident in the stated complexity of the rollout approach (see Theorem 1, the proof which is found in Proposition 26 of Secomandi (1998)). Definition 4 generalizes this problem- and heuristic-specific technique to general SDPs.

The *branch-and-regret* procedure applied in Hvattum et al. (2007) also implicitly employs a post-decision state decision rule. Hvattum et al. (2007) address a vehicle routing problem with stochastic demand where some customer requests arrive randomly over a given time horizon. At pre-defined time intervals, a procedure is executed to update the routing policy. Part of the procedure determines whether or not certain customer requests will be serviced in the current time period, or in a later time period. These choices are evaluated by making transitions to the corresponding post-decision states, a process the authors refer to as *branching*.

Goodson et al. (2013) present a post-decision state decision rule in the context of a vehicle routing problem with stochastic demand and duration limits. This paper generalizes that presentation. Goodson et al. (2013) demonstrates that using post-decision state decision rules is effective and offers some computational benefits. However, such decision rules can still be challenging to evaluate for dynamic programs with large action spaces. For example, Goodson et al. (2013) find even for moderately-sized vehicle routing problem instances (e.g., 50 customers and several vehicles), the number of feasible actions at a decision epoch may range from several hundred to several million. As the number of feasible actions grows, executing even simple heuristics once for each action becomes computationally prohibitive.

### 3.2.3 Pre-Decision State Decision Rules

Motivated by the potential computational issues associated with applying a post-decision state decision rule, we formalize a *pre-decision state decision rule* that executes heuristic $\mathcal{H}(\cdot)$ only once from the current, pre-decision state. The pre-decision state decision rule can be viewed as looking ahead "zero-steps" in contrast to the one-step lookahead of one-step rollout and half-step lookahead of post-decision state rollout. Although a rollout algorithm employing the pre-decision state decision rule falls outside the traditional view of rollout algorithms as lookahead procedures, the definition of a pre-decision state decision rule provides completeness of the performance improvement properties presented in Section 4. Further, our final decision rule, the hybrid decision rule, requires the formalization of a pre-decision state decision rule.

In contrast to the previously presented decision rules, the pre-decision state decision rule does not make use of the re-application of a heuristic to approximate the reward-to-go from reachable future states. Rather, the action selected at decision epoch $k$ is the action prescribed by policy $\pi_{\mathcal{H}(s_k)}$. Figure 2c depicts a pre-decision rollout decision rule where heuristic $\mathcal{H}(\cdot)$ is executed only once in pre-decision state $s_k$. Definition 5 formalizes the decision rule and Algorithm 3 illustrates the steps required to evaluate (3). Line 1 of Algorithm 3 executes the heuristic from the current, pre-decision state, and line 2 returns the action prescribed by the heuristic policy.

**Definition 5** (Pre-Decision State Decision Rule)**.** At decision epoch $k$ when the process occupies state $s_k$, the pre-decision state decision rule is:

$$\delta_k^{\pi_{\text{pre}}}(s_k) = \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k). \tag{3}$$

**Algorithm 3** Evaluating the Pre-Decision State Decision Rule

1: execute $\mathcal{H}(s_k)$ to obtain $\pi_{\mathcal{H}(s_k)}$
2: $a^\star \leftarrow \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)$

The notion of a pre-decision state decision rule implicitly arises in *rolling-horizon* procedures that involve a single re-solving execution of a mathematical program or metaheuristic to select an action from the current state. As an example, for a vehicle routing problem with stochastic demand, Novoa and Storer (2008) implicitly implement a pre-decision state decision rule in the method they call "n2reopt." At each decision epoch, the method returns an updated policy via a single execution of a heuristic from the pre-decision state. Additional examples from the routing literature can be found in Goodson et al. (2013) and Goodson et al. (2016). The control-algorithm method of Secomandi (2008) is compatible with our definition of a pre-decision state decision rule. At each state, this control-algorithm implementation uses a single execution of the control-algorithm heuristic to update the parameters associated with a threshold-like decision rule.

### 3.2.4 Hybrid Decision Rules

In this section, we introduce a *hybrid decision rule* that combines pre- and post-decision state decision rules. Although pre-decision state decision rules provide a computational advantage over post-decision state decision rules, because the heuristic policy is usually a member of a restricted policy class, by definition certain actions may not available to the heuristic policy. While one-step and post-decision state decision rules overcome this limitation by their design, the pre-decision rule may never consider some actions or may mis-evaluate the value of some actions. The aim of

a hybrid decision rule is to reduce the computational burden of (2) while yielding a better decision rule than (3).

The hybrid decision rule limits the number of post-decision states from which the heuristic is executed. The basic premise of the hybrid decision rule is that a single heuristic execution from the current pre-decision state $s_k$ can be used to implicitly evaluate a subset of the action space, the restricted action set $\bar{\mathcal{A}}(s_k) \subseteq \mathcal{A}(s_k)$. Then, specific actions are identified and evaluated by applying the heuristic from the respective post-decision states to estimate the rewards-to-go. This is in contrast to one-step and post-decision state decision rules that explicitly evaluate each feasible action by executing the heuristic from the resulting pre- or post-decision states to estimate the respective rewards-to-go.

The first step in evaluating the hybrid decision rule is to evaluate the pre-decision state decision rule by executing the heuristic from the current state. Denote the action returned by the pre-decision state decision rule at decision epoch $k$ when the process occupies state $s_k$ by $\bar{a}_k = \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)$ . By Definition 2, $\bar{a}_k$ is in restricted action set $\bar{\mathcal{A}}(s_k)$ because heuristic $\mathcal{H}(\cdot)$ operates on the restricted policy class $\bar{\Pi}$.

The hybrid decision rule then evaluates, from the post-decision state, actions in $\check{\mathcal{A}}(s_k) \subseteq \left\{\mathcal{A}(s_k) \setminus \bar{\mathcal{A}}(s_k)\right\}$, a subset of feasible actions not included in the restricted action set. The hybrid decision rule also evaluates from the post-decision state the actions belonging to $\hat{\mathcal{A}}(s_k)$, a select subset of actions belonging to the restricted action space $\bar{\mathcal{A}}(s_k)$. The set $\hat{\mathcal{A}}(\cdot)$ contains actions deemed by the decision maker to warrant more careful evaluation from the post-decision state and this choice is problem-dependent. For the performance improvement properties in §4 to hold, $\bar{a}_k$ must be an element of $\hat{\mathcal{A}}(s_k)$. Definition 6 formalizes the hybrid decision rule.

**Definition 6** (Hybrid Decision Rule)**.** Let $\bar{a}_k = \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)$ be the action prescribed by policy $\pi_{\mathcal{H}(s_k)}$ at decision epoch $k$ when the process occupies state $s_k$. Let $\bar{\mathcal{A}}(s_k)$ be the restricted action set as defined by Definition 1. Let $\hat{\mathcal{A}}(s_k) \subseteq \bar{\mathcal{A}}(s_k)$ be a chosen subset of the restricted action set containing at least $\bar{a}_k$ and $\check{\mathcal{A}}(s_k) \subseteq \left\{\mathcal{A}(s_k) \setminus \bar{\mathcal{A}}(s_k)\right\}$ be a chosen subset of actions not included in the restricted action set. Then, the hybrid decision rule $\delta_k^{\pi_{\text{hy}}}(s_k)$ maps $s_k$ to an element in the set:

$$\underset{a \in \{\check{\mathcal{A}}(s_k) \cup \hat{\mathcal{A}}(s_k)\}}{\arg\max} \left\{ R_k(s_k, a) + \mathbb{E}\left[ \sum_{i=k+1}^{K} R_i\left( s_i, \delta_i^{\pi_{\mathcal{H}}\left(s_k^a\right)}(s_i) \right) \middle| s_k \right] \right\}. \tag{4}$$

Algorithm 4 illustrates the steps required to evaluate (4) and return an action. Line 2 executes the heuristic from pre-decision state $s_k$, then line 3 uses the resulting policy to obtain action $\bar{a}_k$. Line 4 iterates through the actions in $\check{\mathcal{A}}(s_k) \cup \hat{\mathcal{A}}(s_k)$. Line 5 executes the heuristic from the post-decision state and line 6 calculates the value of the current-period reward plus the expected future

reward of following policy $\pi_{\mathcal{H}(s_k^a)}$ from decision epoch $k+1$ onward. The logic beginning on line 7 of Algorithm 4 tracks the best action, which is returned on line 10.

**Algorithm 4** Evaluating the Hybrid Decision Rule

1: $\lambda \leftarrow -\infty$
2: execute $\mathcal{H}(s_k)$ to obtain $\pi_{\mathcal{H}(s_k)}$
3: $\bar{a}_k \leftarrow \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)$
4: **for** $a \in \{\check{\mathcal{A}}(s_k) \cup \hat{\mathcal{A}}(s_k)\}$ **do**
5: execute $\mathcal{H}(s_k^a)$ to obtain $\pi_{\mathcal{H}(s_k^a)}$
6: $\eta \leftarrow R_k(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i(s_i, \delta_i^{\pi_{\mathcal{H}(s_k^a)}}(s_i))|s_k\right]$
7: **if** $\eta > \lambda$ **then**
8: $\lambda \leftarrow \eta$
9: $a^\star \leftarrow a$
10: Return $a^\star$

Figure 2d depicts a hybrid decision rule. The two gray arcs represent actions belonging to restricted action space $\bar{\mathcal{A}}(s_k)$ and the black arc represents a single action comprising $\check{\mathcal{A}}(s_k)$. Heuristic $\mathcal{H}(\cdot)$ is executed once from pre-decision state $s_k$, yielding action $\bar{a}_k$. In this small example, $\hat{\mathcal{A}}(s_k) = \{\bar{a}_k\}$. Then, we execute heuristic $\mathcal{H}(\cdot)$ from the post-decision state corresponding to the single action composing $\check{\mathcal{A}}(s_k)$ and from the post-decision state corresponding to the single action $\bar{a}_k$ composing $\hat{\mathcal{A}}(s_k)$.

Goodson et al. (2013) develop a rollout algorithm using the hybrid decision rule for a vehicle routing problem with stochastic demand. In their work, the set $\check{\mathcal{A}}(s_k)$ consists of all actions that simultaneously replenish the capacity of all vehicles before capacity is depleted. Recognizing that some preemptive capacity replenishment actions belonging to $\bar{\mathcal{A}}(\cdot)$ are often evaluated inaccurately by pre-decision decision rules, Goodson et al. (2013) include such actions in $\hat{\mathcal{A}}(\cdot)$. Goodson et al. (2013) find the use of a hybrid decision rule facilitates real-time execution on problems 50 percent larger than those amenable to real-time execution when employing a post-decision decision rule. In general, the hybrid decision rules executes a heuristic 1 + $|\check{\mathcal{A}}(s_k) \cup \hat{\mathcal{A}}(s_k)|$ times, which depending on the specification of $\check{\mathcal{A}}(s_k)$ and $\hat{\mathcal{A}}(s_k)$ can dramatically reduce the computational expense relative to one-step and post-decision state decision rules.

## 3.3 Rollout Algorithm and Rollout Policy

In this section, we present the rollout algorithm and define rollout policies. The rollout algorithm using a one-step decision rule is given in Algorithm 5. As Algorithm 5 illustrates, a rollout algo-

rithm is an online forward dynamic programming procedure that selects actions for realized states in real time by employing a lookahead decision rule like those described in §3.2.

Line 1 of Algorithm 5 initializes the state variable and line 2 iterates the subsequent steps until a terminal state is reached. Line 3 evaluates the one-step decision returning an action $a^\star$ for the current state. Line 4 records the rollout decision rule for epoch $k$ by mapping state $s_k$ to the chosen action. Lines 5 and 6 transition to the next state and decision epoch. We note that rollout algorithms are executed in real time and hence $w_{k+1}$ represents the exogenous information that is learned between epochs $k$ and $k+1$. To employ post-decision, pre-decision, and hybrid decision rules in the algorithm, we replace $\delta_k^{\pi_{\text{one}}}(s_k)$ in line 3 with $\delta_k^{\pi_{\text{post}}}(s_k)$, $\delta_k^{\pi_{\text{pre}}}(s_k)$, and $\delta_k^{\pi_{\text{hy}}}(s_k)$, respectively.

**Algorithm 5** Rollout Algorithm

1: Initialize current state, $k \leftarrow 0$ and $s_k \leftarrow s_0$
2: **while** $s_k$ is not a terminal state **do**
3: Evaluate $\delta_k^{\pi_{\text{one}}}(s_k)$ returning $a^\star$
4: $\delta_k^{\pi_{\text{rollout}}}(s_k) \leftarrow a^\star$
5: $s_{k+1} \leftarrow S\left(s_k, \delta_k^{\pi_{\text{rollout}}}(s_k), w_{k+1}\right)$
6: $k \leftarrow k+1$

The iterative evaluation of the chosen decision rule results in a rollout policy $\pi_{\text{rollout}}$, which we formalize in Definition 7.

**Definition 7** (Rollout Policies)**.** A rollout policy $\pi_{\text{rollout}} = (\delta_0^{\pi_{\text{rollout}}}, \delta_1^{\pi_{\text{rollout}}} \ldots, \delta_K^{\pi_{\text{rollout}}})$ is a sequence of decision rules where each decision rule $\delta_k^{\pi_{\text{rollout}}}$ is given by step 4 of Algorithm 5.

As Algorithm 5 implies, a rollout policy evaluates decision rules only for realized states. We call the rollout algorithm that uses the one-step decision rule the one-step rollout algorithm. Analogously, we call the rollout algorithm using post-decision state, pre-decision state, and hybrid decision rules post-decision, pre-decision, and hybrid rollout algorithms, respectively.

# 4 Performance Improvement Properties

Generally speaking, the primary contribution of post-decision, pre-decision, and hybrid rollout policies is a reduction in computation over one-step rollout policies. Unless special structure is imposed on heuristic $\mathcal{H}(\cdot)$, we must rely on experimentation to compare the performance of different types of rollout policies for a given problem. In this section, we outline performance improvement properties for rollout policies by discussing methods to achieve the *rollout improvement property*,

which we state in Definition 8. A rollout policy that is rollout improving does not degrade the performance of the heuristic policy. Although the results throughout this section require exact values for expected rewards-to-go, the computational work in §5 suggests the results hold when estimating values via simulation. Our definition of the rollout improvement property specifies a sum from the current epoch $k$ to the end of the horizon $K$. Here and throughout the remainder of the paper, we assume all heuristic policies specify decision rules through the end of the horizon as well.

**Definition 8** (Rollout Improvement Property)**.** For heuristic $\mathcal{H}(\cdot)$ and a rollout policy $\pi$, we say $\pi$ is rollout improving if for $k = 0, 1, \ldots, K$,

$$\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k)}}(s_i)\right) \middle| s_k\right] \leq \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi}(s_i)\right) \middle| s_k\right]. \tag{5}$$

In §4.1, we show *sequentially improving* heuristics lead to rollout policies that are rollout improving. In §4.2, we discuss *sequentially consistent* heuristics, a class of sequentially improving heuristics. In §4.3, we show how to achieve the rollout improvement property with rollout policies based on any heuristic. Proofs for our results can be found in the Appendix.

## 4.1 Sequentially Improving Heuristics

Definition 9 defines a *sequentially improving* heuristic, a concept first introduced by Bertsekas et al. (1997) for deterministic dynamic programming problems and restated for control algorithms by Secomandi (2008). Definition 9 generalizes the concept for general SDPs. In the definition, we refer to a state $s'$ on the sample path induced by a heuristic policy $\pi_{\mathcal{H}(s)}$, implying $s'$ belongs to one of the potential state trajectories obtained by selecting actions via $\pi_{\mathcal{H}(s)}$.

**Definition 9** (Sequentially Improving Heuristics)**.** Let $s$ be a (pre- or post-decision) state in state space $\mathcal{S}$ and let $s'$ be a state such that it is on a sample path induced by policy $\pi_{\mathcal{H}(s)}$. Then, a heuristic $\mathcal{H}(\cdot)$ is sequentially improving if, for all $s$ and subsequent $s'$,

$$\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s)}}(s_i)\right) \middle| s'\right] \leq \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s')}}(s_i)\right) \middle| s'\right]. \tag{6}$$

Proposition 1 shows rollout policies based on sequentially improving heuristics are rollout improving. Bertsekas et al. (1997) demonstrate an analogous result for one-step rollout applied to deterministic dynamic programs.

**Proposition 1** (Sequential Improvement Implies Rollout Improvement)**.** *If a heuristic is sequentially improving, then rollout policies* $\pi_{\text{one}}$*,* $\pi_{\text{post}}$*,* $\pi_{\text{pre}}$*, and* $\pi_{\text{hy}}$ *based on that heuristic are rollout improving.*

## 4.2 Sequentially Consistent Heuristics

Bertsekas et al. (1997) introduce *sequentially consistent* heuristics in discussing rollout algorithms for deterministic dynamic programs. Secomandi (2003) extends the concept of sequential consistency of one-step rollout to SDPs. In our rollout framework, a sequentially consistent heuristic is characterized by the sequence of decision rules generated when the heuristic is executed in a state $s$ and again in a state $s'$ lying on a sample path generated by policy $\pi_{\mathcal{H}(s)}$. If the decision rules are equivalent, then the heuristic is sequentially consistent. We formalize this concept in Definition 10. Examples of sequentially consistent heuristics for deterministic and stochastic problems can be found in Bertsekas et al. (1997) and Secomandi (2003), respectively.

**Definition 10** (Sequentially Consistent Heuristics)**.** Let $s$ be a (pre- or post-decision) state in state space $\mathcal{S}$ and let $s'$ be a state such that it is on a sample path induced by policy $\pi_{\mathcal{H}(s)}$. Heuristic $\mathcal{H}(\cdot)$ is sequentially consistent if for all $s$ and subsequent $s'$,

$$\left(\delta_k^{\pi_{\mathcal{H}(s)}}, \delta_{k+1}^{\pi_{\mathcal{H}(s)}}, \ldots, \delta_K^{\pi_{\mathcal{H}(s)}}\right) = \left(\delta_k^{\pi_{\mathcal{H}(s')}}, \delta_{k+1}^{\pi_{\mathcal{H}(s')}}, \ldots, \delta_K^{\pi_{\mathcal{H}(s')}}\right). \tag{7}$$

Definition 10 frames sequential consistency in terms of decision rules, whereas Bertsekas et al. (1997) and Secomandi (2003) define sequential consistency on sample paths. The two definitions may be viewed as equivalent, but we choose to use decision rules because we define a heuristic as a mechanism to select decision rules and because policies (sequences of decision rules) are feasible solutions to dynamic programs.

In Proposition 2, we show a sequentially consistent heuristic is also sequentially improving. Thus, by Proposition 1, one-step, post-decision, pre-decision, and hybrid rollout policies based on a sequentially consistent heuristic are rollout improving. Bertsekas et al. (1997) show an analogous result for sequentially consistent heuristics and one-step rollout applied to deterministic dynamic programs. Because Proposition 2 applies to SDPs and a broader class of rollout policies, it generalizes Bertsekas et al. (1997). Secomandi (2003) shows one-step rollout policies based on sequentially consistent heuristics are rollout improving for general SDPs, but does not show sequential consistency implies sequential improvement. For a definition of sequential consistency modified for control algorithms, Secomandi (2008) demonstrates that, when coupled with additional properties, sequentially consistent control algorithms are sequentially improving.

**Proposition 2** (Sequential Consistency Implies Sequential Improvement)**.** *If a heuristic is sequentially consistent, then it is also sequentially improving.*

By the definition of a sequentially consistent heuristic, it is straightforward to show Proposition 1 holds at equality for a pre-decision rollout policy based on a sequentially consistent heuristic. We

formalize this result in Proposition 3, which implies there is no value in re-executing the heuristic at future decision epochs in a pre-decision rollout algorithm because the heuristic policy will not deviate from the policy induced by executing the heuristic in initial state $s_0$.

**Proposition 3** (Sequentially Consistent Pre-Decision Rollout)**.** *If heuristic $\mathcal{H}(\cdot)$ is sequentially consistent, then* (5) *holds at equality, thereby implying*

$$\mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_0)}}(s_i)\right) \middle| s_0\right] = \mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{pre}}}(s_i)\right) \middle| s_0\right]. \tag{8}$$

In Proposition 4, we demonstrate one-step, post-decision, and hybrid rollout policies built on a sequentially consistent heuristic perform no worse than a pre-decision rollout policy built on the same heuristic. For a vehicle routing problem with stochastic demand, the computational experiments of Secomandi (2001) and Novoa and Storer (2008) demonstrate one-step and multi-step lookahead rollout policies, built on the sequentially consistent cyclic heuristic of Bertsimas et al. (1995), perform much better than the "base heuristic," i.e., following the policy given by executing the heuristic in the initial state. By Proposition 3, the "base heuristics" of Secomandi (2001) and Novoa and Storer (2008) are equivalent to a pre-decision rollout policy. Thus, although Proposition 4 states only weak improvement, these computational experiments suggest the performance of pre-decision rollout, built on a sequentially consistent heuristic, can be significantly improved by considering other types of rollout policies.

**Proposition 4** (Weak Improvement Over Pre-Decision Rollout)**.** *If heuristic $\mathcal{H}(\cdot)$ is sequentially consistent, then*

$$\mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{pre}}}(s_i)\right) \middle| s_0\right] \leq \mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{one}}}(s_i)\right) \middle| s_0\right], \tag{9}$$

$$\mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{pre}}}(s_i)\right) \middle| s_0\right] \leq \mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{post}}}(s_i)\right) \middle| s_0\right], \tag{10}$$

*and*

$$\mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{pre}}}(s_i)\right) \middle| s_0\right] \leq \mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{hy}}}(s_i)\right) \middle| s_0\right]. \tag{11}$$

Further, one-step and post-decision rollout policies built on the same sequentially consistent heuristic yield the same expected rewards. Additionally, if each rollout policy employs the same rule for breaking ties among actions with equivalent evaluations, then one-step and post-decision rollout decision rules are equivalent. We state this new result in Proposition 5. Computationally,

the result is notable because post-decision state decision rules significantly reduce the number of times a heuristic must be executed to select an action. As noted in §3.2.2, the approach in Secomandi (2001) implicitly leverages this fact for a vehicle routing problem with stochastic demand. Proposition 5 explicitly recognizes this behavior and extends the concept to general SDPs. Secomandi (2000) and Novoa and Storer (2008) implicitly exploit the same property in their work on vehicle routing problems with stochastic demand.

**Proposition 5** (Sequentially Consistent One-Step and Post-Decision Rollout)**.** *If heuristic $\mathcal{H}(\cdot)$ is sequentially consistent, and if policies $\pi_{\text{one}}$ and $\pi_{\text{post}}$ employ the same rule for breaking ties among actions with equivalent evaluations, then for $k = 0, 1, \ldots, K$,*

$$\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{one}}}(s_i)\right) \middle| s_k\right] = \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{post}}}(s_i)\right) \middle| s_k\right]. \tag{12}$$

*Further,* $\delta_k^{\pi_{\text{one}}} = \delta_k^{\pi_{\text{post}}}$.

## 4.3 Fortified Rollout

It can be challenging to identify heuristics that achieve the rollout improvement property. In this section, we discuss how to obtain the rollout improvement property with any heuristic. The idea is straightforward: given an in-hand policy $\pi_{\text{best}}$, select an action via the in-hand policy or the rollout policy, whichever attains the larger expected reward, and then update $\pi_{\text{best}}$ accordingly. The idea extends to general SDPs the concept of a *fortified rollout algorithm*, first introduced by Bertsekas et al. (1997) for deterministic dynamic programs.

Definition 11 states the decision rule employed by fortified one-step rollout policy $\pi_{\text{f-one}}$. Proposition 6 introduces a new result that states policy $\pi_{\text{f-one}}$ is rollout improving with respect to the in-hand policy, which may be initialized via $\pi_{\mathcal{H}(\cdot)}$.

**Definition 11** (Fortified One-Step Rollout)**.** Let $\pi_{\text{best}}$ be an in-hand policy. At decision epoch $k$, when the process occupies state $s_k$ fortified one-step rollout policy $\pi_{\text{f-one}}$ employs the following decision rule:

$$\delta_k^{\pi_{\text{f-one}}}(s_k) = \begin{cases} \delta_k^{\pi_{\text{best}}}(s_k), & \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{best}}}(s_i)\right) \middle| s_k\right] \\ & \geq \max_{a \in \mathcal{A}(s_k)} \left\{ R_k(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_k\right] \right\}, \\ \delta_k^{\pi_{\text{one}}}(s_k), & \text{otherwise.} \end{cases} \tag{13}$$

Policy $\pi_{\text{best}}$ is set to

$$\pi_{\text{best}} = \begin{cases} \pi_{\text{best}}, & \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{best}}}(s_i)\right) \middle| s_k\right] \\ & \geq \max_{a \in \mathcal{A}(s_k)} \left\{ R_k(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_k\right] \right\}, \\ \pi_{\mathcal{H}(s_{k+1})}, & \text{otherwise.} \end{cases} \tag{14}$$

In the second case where $\pi_{\text{best}}$ is set to $\pi_{\mathcal{H}(s_{k+1})}$, state $s_{k+1}$ equals $S(s_k, a^\star, w_{k+1})$ with $a^\star$ being the action returned by one-step rollout and $w_{k+1}$ the realization of $W_{k+1}$. This update of $\pi_{\text{best}}$ occurs when state $s_{k+1}$ is realized.

**Proposition 6** (Fortified Rollout is Sequentially Improving)**.** *Given an in-hand policy* $\pi_{\text{best}}$*, for* $k = 0, 1, \ldots, K$*,*

$$\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{best}}}(s_i)\right) \middle| s_k\right] \leq \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{f-one}}}(s_i)\right) \middle| s_k\right]. \tag{15}$$

The concept of fortified rollout can be extended to post-decision, hybrid, and pre-decision rollout. The definitions and proofs are analogous to those of Definition 11 and Proposition 6. As an example of fortification, Goodson et al. (2013) apply the various rollout variants to a vehicle routing problem with stochastic demand. Goodson et al. (2013) fortify a non-sequentially consistent heuristic to generate rollout policies with the rollout improvement property. Although Goodson et al. (2013) found fortification necessary to achieve rollout improvement, whether or not fortification is required depends on the problem. Regardless, if the additional computational effort to evaluate $\pi_{\text{best}}$ once at each decision epoch is not prohibitive, then fortification guarantees rollout improvement.

# 5 Application of Rollout Framework

To illustrate the rollout algorithm variants, we formulate a *dynamic and stochastic multi-compartment knapsack problem* (DSMKP) as a stochastic dynamic program and apply our framework. The DSMKP is a multi-dimensional variant of the problem presented in Papastavrou et al. (1996). We formulate the problem in §5.1, discuss the heuristic in §5.2, and present computational results in §5.3.

## 5.1 DSMKP Formulation

The objective of the DSMKP is to pack a knapsack with items presented over a finite time horizon such that total expected reward is maximized subject to constraints on compartmental capacities and overall knapsack capacity. At each decision epoch, the decision maker is presented with a random set of items, any subset of which may be considered for potential inclusion in the knapsack. Although multiple items may be presented simultaneously, we assume at most one item is available to each compartment at each epoch. Compartment capacities, item sizes, and the reward for including a given subset of items in the knapsack are fixed and known. However, prior to the presentation of items, the set of items available at an epoch is known only in distribution.

The state of the system captures all information required to accept or reject items at decision epoch $k$ and includes remaining compartment capacities, remaining overall capacity, and the subset of items available at the current epoch. We denote the set of knapsack compartments by $\mathcal{C} = \{1, 2, \ldots, C\}$, the remaining capacity of compartment $c$ in $\mathcal{C}$ at epoch $k$ as $q_k^c$, and the vector of remaining compartment capacities at epoch $k$ by $q_k = (q_k^c)_{c\in\mathcal{C}}$. We denote remaining overall knapsack capacity at epock $k$ by $Q_k$. Let $W_{k+1}^c$ be a binary random variable equal to one if an item is available to compartment $c$ following action selection in period $k$ and zero otherwise. Let $W_{k+1} = (W_{k+1}^c)_{c\in\mathcal{C}}$ be the vector of random variables with realization $w_{k+1} = (w_{k+1}^c)_{c\in\mathcal{C}}$ marking the beginning of period $k+1$.

The pre-decision state at decision epoch $k$ is $s_k = (q_k, Q_k, w_k)$. Denoting by $\bar{q}^c$ the initial capacity of compartment $c$ and by $\bar{Q}$ the initial overall capacity, state $s_k$ belongs to state space $\mathcal{S} = [0, \bar{q}^1] \times [0, \bar{q}^2] \times \cdots \times [0, \bar{q}^C] \times [0, \bar{Q}] \times \{0, 1\}^C$. At the final decision epoch $K$, state $s_K$ belongs to the set of terminal states $\mathcal{S}_K = [0, \bar{q}^1] \times [0, \bar{q}^2] \times \cdots \times [0, \bar{q}^C] \times [0, \bar{Q}] \times \{0\}^C$.

An action $a = (a^1, a^2, \ldots, a^C)$ is a $C$-dimensional binary vector representing a decision to accept or reject each available item at decision epoch $k$, where $a^c = 1$ if the item presented to compartment $c$ is accepted and $a^c = 0$ otherwise. When an item is presented to compartment $c$ in $\mathcal{C}$, the fixed and known size of the item is $d^c$. When the system occupies state $s_k$, the feasible

action set is

$$\mathcal{A}(s_k) = \Bigg\{ a \in \{0,1\}^C : \tag{16}$$

$$a^c = 0, \forall\, c \in \{c' : w_k^{c'} = 0\}, \tag{17}$$

$$a^c d^c \leq q_k^c, \forall\, c \in \mathcal{C}, \tag{18}$$

$$\sum_{c \in \mathcal{C}} a^c d^c \leq Q_k \Bigg\}. \tag{19}$$

Condition (16) requires the action to be in the space of binary $C$-vectors. Condition (17) disallows acceptance of unavailable items. Condition (18) enforces the compartmental capacity constraint and condition (19) implements the overall capacity constraint.

When the system occupies state $s_k$ and action $a$ is selected, a reward $R(s_k, a)$ is accrued. Although the reward function may take a variety of forms, we adopt the following:

$$R(\cdot, a) = \sum_{c=1}^{C} r^c a^c + \eta \max \left\{ \sum_{c=1}^{C} r^c a^c - \gamma, 0 \right\}, \tag{20}$$

where $r^c$ is a fixed and known base reward for accepting an item into compartment $c$, $\eta$ is a parameter in the range $[0, 1]$, and $\gamma$ is a non-negative parameter. The first term is the sum of the base rewards for accepted items and the second term is an $\eta \times 100$ percent bonus of the portion of the base reward exceeding threshold $\gamma$. From a sales perspective, equation (20) may be interpreted as a commission with an incentive to exceed sales of $\gamma$.

Selecting action $a$ from pre-decision state $s_k$ triggers a deterministic transition to post-decision state $s_k^a = (q_{k,a}, Q_{k,a})$ in which the remaining capacity in compartment $c$ is $q_{k,a}^c = q_k^c - a^c d^c$ and remaining overall capacity is $Q_{k,a} = Q_k - \sum_{c \in \mathcal{C}} a^c d^c$. A stochastic transition to the next pre-decision state $s_{k+1}$ is triggered by the arrival of random information $w_{k+1}$, the presentation of a new set of items.

It is instructive to note connections to the single-compartment problem of Papastavrou et al. (1996). If the following two conditions are satisfied, then the DSMKP can be decomposed into $C$ single-compartment dynamic and stochastic knapsack problems, each of which can be solved via the methods of Papastavrou et al. (1996). First, the overall knapsack capacity must not be binding, i.e., $Q \geq \sum_{c \in \mathcal{C}} \bar{q}^c$. Second, the reward accrued for a subset of items must be equal to the sum of the rewards of the individual items. Our problem instances violate these conditions, thereby making the problems unsolvable via the exact methods of Papastavrou et al. (1996) and therefore more difficult.

Generally speaking, because the state and action spaces grow exponentially in size with the number of knapsack compartments $C$, identifying optimal DSMKP policies via standard backward induction is computationally intractable even for moderately-sized problem instances. Thus, heuristic optimization techniques are warranted.

## 5.2 Greedy Policy

To facilitate our rollout algorithms, we develop a greedy heuristic policy $\pi_{\mathcal{H}(s)} = (\delta_k^{\text{greedy}}, \delta_{k+1}^{\text{greedy}}, \ldots, \delta_K^{\text{greedy}})$, where each algorithmic decision rule $\delta^{\text{greedy}}$ is defined by the greedy action construction procedure of Algorithm 6. Given current state $s_k$ at decision epoch $k$, Algorithm 6 identifies a feasible action as follows. Line 1 of Algorithm 6 takes as input the current state and a user-defined parameter $\alpha$ in the range $(0, 1]$. Line 2 initializes the action to the zero-vector. Let $e(c)$ be a $C$-dimensional vector with the $c^{\text{th}}$ element equal to one and all other elements equal to zero. Vector $e(c)$ represents the decision to accept only the item presented to compartment $c$ and line 3 calculates the reward for all such actions. Line 4 identifies an internal ordered set of compartment indices $\tilde{\mathcal{C}}$ by sorting rewards in descending order and breaking ties by giving preference to lower-numbered compartments. Line 5 sets internal variable $\tilde{Q}$ to remaining knapsack capacity $Q_k$. Each iteration of the loop comprising lines 6-11 selects a compartment index in $\tilde{\mathcal{C}}$ and accepts the presented item if doing so does not violate compartment and overall capacity constraints. Line 7 chooses compartment $c$ randomly from the first $\lceil \alpha |\tilde{\mathcal{C}}| \rceil$ element(s) of $\tilde{\mathcal{C}}$. A purely greedy selection is achieved by setting $\alpha$ small enough so that $\lceil \alpha |\tilde{\mathcal{C}}| \rceil$ yields one, while larger values of $\alpha$ permit randomized greedy selection from the first $\lceil \alpha |\tilde{\mathcal{C}}| \rceil$ compartments in $\tilde{\mathcal{C}}$. Lines 8-9 make the feasibility check and line 10 removes the selected compartment from future consideration. Line 11 returns the action.

Similar to the "base policies" of Bertsekas (2005a), calling heuristic $\mathcal{H}(\cdot)$ triggers the identification of an action in the current state and any subsequent state, which we achieve by executing Algorithm 6. The restricted policy class $\bar{\Pi}$ associated with $\mathcal{H}(\cdot)$ is the set of maximal cardinality greedy policies – policies that at each epoch select as many items as will obey the capacity constraints given the randomized greedy order that the items are selected in Line 7. Thus, the restricted action set $\bar{\mathcal{A}}(s_k)$ contains all maximal cardinality greedy actions available from state $s_k$. As we describe in §5.3, the greedy aspect of the heuristic naturally leads to the consideration of non-greedy actions in a hybrid rollout algorithm.

We also note that when $\alpha$ is set sufficiently small to result in purely (versus randomized) greedy action construction, then the greedy heuristic is sequentially consistent. Generally speaking, constructing a policy via a sequence of deterministic decision rules – deterministic in the sense that they always return the same action for a given state – satisfies Definition 10, thereby guaranteeing

**Algorithm 6** Greedy Decision Rule $\delta^{\text{greedy}}$

**input**: $s_k = (q_k, Q_k, w_k)$ and $\alpha \in (0, 1]$
$a^c \leftarrow 0$ for all $c \in \mathcal{C}$
Calculate reward $R(s_k, e(c))$ for each $c \in \{c' \in \mathcal{C} : w_k^{c'} = 1\}$
Sort rewards in descending order, giving preference to larger compartment indices in cases of ties and call the resulting ordered set of indices $\tilde{\mathcal{C}}$
$\tilde{Q} \leftarrow Q_k$
**repeat**
    Randomly select compartment $c$ from the first $\lceil \alpha |\tilde{\mathcal{C}}| \rceil$ elements of $\tilde{\mathcal{C}}$
    **if** $d^c \leq \tilde{Q}$ and $d^c \leq q_k^c$ **then**
        $a^c \leftarrow 1$ and $\tilde{Q} \leftarrow \tilde{Q} - d^c$
    $\tilde{\mathcal{C}} \leftarrow \tilde{\mathcal{C}} \setminus \{c\}$
**until** $\tilde{\mathcal{C}} = \emptyset$
**return** $a$

the rollout improvement property via Propositions 1 and 2. When $\alpha$ is larger the rollout improvement property may still be achieved via the fortified rollout procedure of §4.3.

Although the expected reward-to-go of the greedy policy is not readily computable in closed form, the value can easily be estimated via simulation. Similar to the procedure of Bertsekas and Tsitsiklis (1996, chap. 5.2), from a given state $s_k$, we randomly generate 1000 sequences of item presentations $(W_k, W_{k+1}, \ldots, W_K)$ and average the reward earned by the heuristic policy $\pi_{\mathcal{H}(\cdot)}$ across all sequences. We estimate heuristic policy rewards-to-go in this fashion for each rollout algorithm in our computational experiments.

## 5.3 Computational Comparison of Rollout Variants

To obtain dynamic, state-dependent solutions to the DSMKP, we employ our greedy heuristic to instantiate the rollout algorithms of §3. We implement our procedures in C++ and execute all computational experiments on 2.8GHz Intel Xeon processors with between 12 and 48GB of RAM. Our procedure never tested the limits of the RAM. We do not utilize parallel processing.

To highlight the computational implications of the various rollout algorithms, we review the computational requirements of Algorithms 1-4 applied to the DSMKP. The one-step rollout algorithm selects an action at the current epoch by looking ahead one step and applying Algorithm 6 in the $\sum_{a \in \mathcal{A}(s_k)} |\mathcal{S}(s_k, a)|$ possible $s_{k+1}$ states. For the DSMKP, this results in $O(2^{2C})$ applications of the greedy heuristic at each epoch. The post-decision rollout algorithm selects an action at the

current epoch by looking ahead a half-step and applying Algorithm 6 in the $|\mathcal{A}(s_k)|$ possible $s_k^a$ states. For the DSMKP, this results in $O(2^C)$ applications of the greedy heuristic at each epoch. The pre-decision rollout algorithm selects an action at the current epoch via a single application of Algorithm 6. An example of the various rollout procedures applied to the DSMKP can be found in the Appendix.

As we show below in our computational experiments, although the pre-decision rollout algorithm requires less computational effort than the one-step and post-decision rollout algorithms, pre-decision rollout always selects a maximal cardinality greedy action, which may lead to poor performance. Our hybrid rollout algorithm aims to improve the performance of pre-decision rollout while decreasing the computational requirement of post-decision rollout. In our hybrid rollout algorithm, we set action set $\hat{\mathcal{A}}(s_k) = \{\bar{a}_k\}$ to be the maximal cardinality greedy action returned by the heuristic executed in the current state. Recognizing that non-greedy actions may lead to better performance, we set $\check{\mathcal{A}}(s_k) = \{\mathbf{0}\}$ to the $C$-dimensional zero-vector, the action that rejects all presentations made to the knapsack at epoch $k$. We note that additional non-greedy actions could also be included in $\check{\mathcal{A}}(s_k)$ (such as a random subsets of the greedy set specified by $\{\bar{a}_k\}$), but our computational experiments suggest consideration of the "reject-all" action bridges a significant portion of the gap between pre- and post-decision rollout. Thus, at each decision epoch, the hybrid rollout algorithm executes Algorithm 6 at most three times, once from the current state to identify $\bar{a}_k$ and once from the post-decision state corresponding to each action in $\{\check{\mathcal{A}}(s_k) \cup \hat{\mathcal{A}}(s_k)\} = \{\bar{a}_k \cup \mathbf{0}\}$.

We conduct computational experiments on the DSMKP by considering instances generated by the Cartesian product of the parameter settings in Table 1. For each of the resulting 128 instances, we randomly generate 100 sequences of item presentations and estimate the expected reward achieved by a rollout policy on a given problem instance as the average performance across the realizations. Several entries in Table 1 require clarification. The size of items presented to compartment $c$, $d^c$, is selected randomly as an integer in the range $[1, 3]$ and is fixed at this value for all problem instances. Similarly, the reward for accepting an item to compartment $c$, $r^c$, is selected randomly as an integer in the range $[1, 10]$ and is fixed at this value for all problem instances. Values of overall knapsack capacity $\bar{Q}$ represent 50 and 75 percent of the sum of compartment capacities, respectively. Values of threshold $\gamma$ represent 10 and 30 percent of the expected reward presented to the knapsack at a decision epoch, respectively.

Table 2 displays the performance of the static application of the greedy policy – i.e., the expected reward accrued by following policy $\pi_{\mathcal{H}}(s_0)$ – and the dynamic application of the four rollout variants for three values of $\alpha$. When $\alpha$ equals 0.01, the greedy heuristic is sequentially consistent. When $\alpha$ equals 0.3 or 0.5, we employ the fortified rollout procedure of §4.3. We present results aggregated over the number of compartments, $C$. For each method, we display the estimate of the

Table 1: Problem Instance Parameters

| Parameter | Values |
|---|---|
| $C$ | 5, 15 |
| $K$ | 10, 30 |
| $d^c$ | Uniform(1,3) |
| $\mathbb{P}\{W_k^c = w_k^c\}$ | 0.3, 0.7 |
| $\bar{q}^c$ | 5, 15 |
| $\bar{Q}$ | $0.50C\bar{q}^c$, $0.75C\bar{q}^c$ |
| $r^c$ | Uniform(1,10) |
| $\eta$ | 0.25, 0.75 |
| $\gamma$ | $0.1\sum_{c\in\mathcal{C}}\mathbb{P}\{W_k^c = w_k^c\}r^c$, $0.3\sum_{c\in\mathcal{C}}\mathbb{P}\{W_k^c = w_k^c\}r^c$ |

expected reward accrued. For a given method, across all values of $\alpha$, we state the average number of CPU seconds required to execute the method over a single realization.

Table 2: Rollout Algorithm Performance

| | $C = 5$ | | | | $C = 15$ | | | |
|---|---|---|---|---|---|---|---|---|
| | Expected Reward | | | | Expected Reward | | | |
| **Method** | $\alpha = 0.01$ | $\alpha = 0.3$ | $\alpha = 0.5$ | **CPU** | $\alpha = 0.01$ | $\alpha = 0.3$ | $\alpha = 0.5$ | **CPU** |
| **Greedy Policy** | 137.9 | 138.0 | 138.3 | 0.0 | 342.3 | 343.1 | 344.3 | 0.0 |
| **Pre-Decision** | 137.9 | 138.0 | 138.3 | 0.9 | 342.3 | 343.2 | 344.3 | 2.4 |
| **Hybrid** | 154.0 | 154.2 | 154.2 | 1.1 | 367.6 | 367.9 | 368.9 | 2.9 |
| **Post-Decision** | 181.5 | 181.8 | 182.4 | 1.7 | 423.9 | 425.3 | 427.1 | 311.8 |
| **One-Step** | 181.5 | 181.9 | 182.5 | 54.9 | – | – | – | – |

Table 2 demonstrates the expected tradeoff between computational effort and policy value: as the heuristic decision rule looks farther ahead from pre-decision to hybrid to post-decision to one-step, rollout performance improves and the CPU requirement increases. When $C = 15$, the one-step rollout algorithm poses excessive computational requirements and is unable to execute across all problem instances and realizations even after several days on a computing cluster with over 100 cores. The hybrid rollout algorithm does not perform as well as the post-decision rollout procedure, but posts markedly better performance than pre-decision rollout, requiring only slightly more CPU time. Thus, for larger DSMKP instances, the hybrid rollout algorithm offers an attractive compromise between myopic pre-decision rollout and computationally intensive one-step rollout.

In addition to illustrating the computational benefits of our rollout framework, Table 2 illustrates our analytical results. While the performance improvement properties of §4 technically depend on the exact calculation of mathematical expectation, our computational experiments suggest they hold when estimating expectations with a sufficiently large number of simulations.

Because the greedy heuristic is sequentially consistent when $\alpha = 0.01$, per Propositions 1 and 2, each rollout variant performs at least as well as the static implementation of the greedy policy. Per Proposition 3, the dynamic application of the greedy heuristic in pre-decision rollout yields the same expected reward as the static greedy policy, demonstrating there is no value in re-optimization when the heuristic is sequentially consistent. Further, as Proposition 4 suggests, the hybrid, post-decision, and one-step rollout algorithms perform at least as well as the pre-decision rollout algorithm. Moreover, as implied by Proposition 5, we observe post-decision rollout achieves the same expected reward as one-step rollout, but requires only a fraction of the computational effort.

When $\alpha$ equals 0.3 and 0.5, the adopted fortification guarantees the rollout improvement property is achieved via Proposition 6. Additionally, we observe the expected reward increases as $\alpha$ increases, suggesting that for the DSMKP, there exists a non-sequentially consistent greedy heuristic that yields better performance than a sequentially consistent greedy heuristic. In general, this result shows that sequential consistency, the method most common in the literature for establishing rollout improvement, is not necessarily the best means of achieving rollout improvement. A fortified heuristic may achieve better performance.

# 6 Conclusion

We present a rollout algorithm framework with the aim of making recent advances in rollout methods more accessible to the research community, particularly to researchers seeking heuristic solution methods for large-scale SDPs. Our framework formalizes rollout algorithm variants exploiting the pre- and post-decision state variables as a means of overcoming computational limitations imposed by large state and action spaces. Our analytical results generalize results from the literature and introduce new results that relate the performance of the rollout variants to one another. Relative to the literature, our policy-based approach to presenting and proving results makes a closer connection to the underpinnings of dynamic programming. We illustrate our framework and analytical results via application to the DSMKP, a challenging sequential decision problem.

Based on our framework, our experience with the DSMKP, and computational results reported in the literature, we can make general comments regarding how one might choose decision rules

when implementing a rollout algorithm in practice. For a given heuristic, our recommendation is to select decision rules that can be evaluated within the time required to select actions and that look ahead as many steps as possible. For those interested in guaranteeing performance based on the results in §4, one also requires heuristic policies that are valid to the end of the horizon. Such policies are commonly found in myopic or greedy policies, as we illustrate with the DSMKP. Further, generalizing from the stochastic vehicle routing literature, rollout algorithms offer a method to transform semi-static policies (e.g., policies based on simple recourse rules or thresholds) into more dynamic policies that perform at least as well.

Future work might consider extending alternative methods for achieving the rollout improvement property first proposed in Bertsekas et al. (1997). Some of these methods may be extensible to the stochastic case. Another avenue of future research is to consider more broadly the work of Bertazzi (2012) and Mastin and Jaillet (2015), which characterizes the performance of rollout algorithms for knapsack problems. Similar results may be possible for other classes of problems. At the same time, while research such as Novoa and Storer (2008) and Goodson et al. (2016) demonstrate heuristics capable of exploring less restrictive policy classes have benefit, finding the appropriate heuristic for each problem is currently more of an art than a science. Future work providing more insight into this question would be valuable. Relatedly, it is possible for a rollout algorithm to make use of heuristic policies that are feasible for only some number of decision epochs in the future rather than to the end of the horizon. While such cases are not the focus of this paper, an interesting area of future work would be computational studies comparing the quality of the rollout policies obtained from such heuristic policies with the quality of rollout policies using methods discussed in this paper.

# Acknowledgments

The authors gratefully acknowledge the comments and suggestions of Nicola Secomandi and Dimitri Bertsekas as well as the comments of the anonymous referees.

# References

Bertazzi, L. (2012). Minimum and worst-case performance ratios of rollout algorithms. *Journal of Optimization Theory and Applications 152*(2), 378–393.

Bertsekas, D. (2005a). *Dynamic programming and optimal control* (3rd ed.), Volume I. Belmont, MA: Athena Scientific.

Bertsekas, D. (2005b). Dynamic programming and suboptimal control: a survey from ADP to MPC. *European Journal of Control 11*, 310–334.

Bertsekas, D. (2013). Rollout algorithms for discrete optimization: a survey. In P. Pardolos, D. Du, and R. Graham (Eds.), *Handbook of Combinatorial Optimization*, pp. 2989–2013. New York: Springer Science+Business Media.

Bertsekas, D. and D. Castanon (1998). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics 5*(1), 89–108.

Bertsekas, D. and J. Tsitsiklis (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Bertsekas, D., J. Tsitsiklis, and C. Wu (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics 3*(3), 245–262.

Bertsimas, D., P. Chervi, and M. Peterson (1995). Computational approaches to stochastic vehicle routing problems. *Transportation Science 29*(4), 342–352.

Bertsimas, D. and R. Demir (2002). An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science 48*(4), 550–565.

Bertsimas, D. and I. Popescu (2003). Revenue management in a dynamic network environment. *Transportation Science 37*(3), 257–277.

Chang, H., M. Fu, J. Hu, and S. Marcus (2013). *Simulation-Based Algorithms for Markov Decision Processes* (Second ed.)., Chapter 5, pp. 179–226. Communications and Control Engineering. London: Springer.

Ciavotta, M., C. Meloni, and M. Pranzo (2009). Scheduling dispensing and counting in secondary pharmaceutical manufacturing. *AIChE Journal 55*(5), 1161–1170.

Ciavotta, M., C. Meloni, and M. Pranzo (2016). Speeding up a rollout algorithm for complex parallel machine scheduling. *International Journal of Production Research 54*(16), 1–17.

Duin, C. and S. Voß(1999). The Pilot Method : A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs. *Networks 34*(3), 181–191.

Goodson, J., J. W. Ohlmann, and B. W. Thomas (2013). Rollout policies for dynamic solutions to the multi-vehicle routing problem with stochastic demand and duration limits. *Operations Research 61*(1), 138–154.

Goodson, J. C., B. W. Thomas, and J. W. Ohlmann (2016). Restocking-based rollout policies for the vehicle routing problem with stochastic demand and duration limits. *Transportation Science 50*(2), 591–607.

Guerriero, F. (2008). Hybrid rollout approaches for the job shop scheduling problem. *Journal of optimization theory and applications 139*(2), 419–438.

Guerriero, F. and M. Mancini (2003). A cooperative parallel rollout algorithm for the sequential ordering problem. *Parallel Computing 29*(5), 663–677.

Guerriero, F. and M. Mancini (2005). Parallelization strategies for rollout algorithms. *Computational Optimization and Applications 31*(2), 221–244.

Guerriero, F., M. Mancini, and R. Musmanno (2002). New rollout algorithms for combinatorial optimization problems. *Optimization Methods and Software 17*(4), 627–654.

Hvattum, L., A. Løkketangen, and G. Laporte (2007). A branch-and-regret heuristic for stochastic and dynamic vehicle routing problems. *Networks 49*(4), 330–340.

Kim, S. and H. Chang (2003). Parallelizing parallel rollout algorithm for solving markov decision processes. *Lecture Notes in Computer Science 2716*, 122–136.

Mastin, A. and P. Jaillet (2015). Average-case performance of rollout algorithms for knapsack problems. *Journal of Optimization Theory and Applications 165*(3), 964–984.

Meloni, C., D. Pacciarelli, and M. Pranzo (2004). A Rollout Metaheuristic for Job Shop Scheduling. *Annals of Operations Research 131*, 215–235.

Novoa, C. and R. Storer (2008). An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research 196*(2), 509–515.

Pacciarelli, D., C. Meloni, and M. Pranzo (2011). Models and Methods for Production Scheduling in the Pharmaceutical Industry. In K. G. Kempf, P. Keshinocak, and R. Uzsoy (Eds.), *Planning Production and Inventories in the Extended Enterprise*, Volume 151 of *International Series in Operations Research & Management Science*, Chapter 17, pp. 429–459. New York: Springer.

Papastavrou, J., S. Rajagopalan, and A. Kleywegt (1996). The dynamic and stochastic knapsack problem with deadlines. *Management Science 42*(12), 1706–1718.

Powell, W. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality* (Second ed.). Hoboken, NJ, USA: John Wiley and Sons.

Pranzo, M., C. Meloni, and D. Pacciarelli (2003). A New Class of Greedy Heuristics for Job Shop Scheduling Problems. In K. Jansen, M. Margraf, M. Mastrolilli, and J. D. P. Rolim (Eds.), *Experimental and Efficient Algorithms*, Volume 2647 of *Lecture Notes in Computer Science*, pp. 223–236. Berlin: Springer.

Secomandi, N. (1998). *Exact and heuristic dynamic programming algorithms for the vehicle routing problem with stochastic demands*. Ph. D. thesis, University of Houston, Houston, TX.

Secomandi, N. (2000). Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers and Operations Research 27*(11-12), 1201–1225.

Secomandi, N. (2001). A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research 49*(5), 796–802.

Secomandi, N. (2003). Analysis of a rollout approach to sequencing problems with stochastic routing applications. *Journal of Heuristics 9*(4), 321–352.

Secomandi, N. (2008). An analysis of the control-algorithm re-solving issue in inventory and revenue management. *Manufacturing and Service Operations Management 10*(3), 468–483.

Voß, S., A. Fink, and C. Duin (2005). Looking Ahead with the Pilot Method. *Annals of Operations Research 136*, 285–302.

# Appendix

Recall, we sometimes refer to generic states $s$ without an index and to a decision epoch $k$. In these cases, $s$ may be either a pre- or post-decision state. When $s$ is a pre-decision state, $k$ is the decision epoch associated with $s$. When $s$ is a post-decision state, the decision epoch associated with $s$ is $k-1$. A times we also refer to a generic state $s'$ on a given sample path. State $s'$ may also be a pre- or post-decision state.

# Proof of Proposition 1

Proofs are by induction. We begin with one-step rollout policy $\pi_{\text{one}}$. The result holds trivially for the $i = K$ case. We assume the result holds for $i = k+1, \ldots, K-1$. Then, for the $i = k$ case:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k)}}(s_i)\right) \middle| s_k\right] \\
&\leq \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \qquad (21) \\
&\leq \max_{a \in \mathcal{A}(s_k)} \mathbb{E}\left[R_k(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \qquad (22)
\end{aligned}$$

$$= \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\text{one}}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \tag{23}$$

$$\leq \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{one}}}(s_i)\right) \middle| s_k\right]. \tag{24}$$

Equation (21) follows from the assumption heuristic $\mathcal{H}(\cdot)$ is sequentially improving and iterated expectations. Equation(22) follows from the maximization. The equality in Equation(23) results from the definition of one-step rollout. Equation (24) follows from the induction hypothesis.

Letting $\bar{a}_k = \delta_k^{\mathcal{H}(s_k)}(s_k)$ and $a^\star = \delta_k^{\pi_{\text{post}}}(s_k)$, the proof for post-decision rollout policy $\pi_{\text{post}}$ is similar to the proof for $\pi_{\text{one}}$, the only addition being equation (28), which follows from the assumption heuristic $\mathcal{H}(\cdot)$ is sequentially improving:

$$\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k)}}(s_i)\right) \middle| s_k\right]$$

$$\leq \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k^{\bar{a}_k})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \tag{25}$$

$$\leq \max_{a \in \mathcal{A}(s_k)} \mathbb{E}\left[R_k\left(s_k, a\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k^{a})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \tag{26}$$

$$= \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\text{post}}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k^{a^\star})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \tag{27}$$

$$\leq \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\text{post}}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \tag{28}$$

$$\leq \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{post}}}(s_i)\right) \middle| s_k\right]. \tag{29}$$

The proof for hybrid rollout policy $\pi_{\text{hy}}$ is identical to the proof for $\pi_{\text{post}}$, except the maximization in equation (26) is over the actions $\{\check{\mathcal{A}}(s_k) \cup \hat{\mathcal{A}}(s_k)$ and $a^\star = \delta_k^{\pi_{\text{hy}}}(s_k)$. The proof for pre-decision rollout policy $\pi_{\text{pre}}$ follows suit:

$$\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k)}}(s_i)\right) \middle| s_k\right]$$

$$\leq \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \tag{30}$$

$$= \mathbb{E}\left[ R_k\left(s_k, \delta_k^{\pi_{\text{pre}}}(s_k)\right) + \mathbb{E}\left[ \sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1} \right] \middle| s_k \right] \tag{31}$$

$$\leq \mathbb{E}\left[ \sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{pre}}}(s_i)\right) \middle| s_k \right]. \tag{32}$$

# Proof of Proposition 2

Because heuristic $\mathcal{H}(\cdot)$ is sequentially consistent, for any state $s$ in $\mathcal{S}$ and state $s'$ such that $s'$ is on a sample path induced by policy $\pi_{\mathcal{H}(s)}$,

$$\mathbb{E}\left[ \sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s)}}(s_i)\right) \middle| s' \right] = \mathbb{E}\left[ \sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s')}}(s_i)\right) \middle| s' \right], \tag{33}$$

which satisfies the definition of a sequentially improving heuristic.

# Proof of Proposition 3

The proof is by induction. The result holds trivially for the $i = K$ case. We assume the result holds for $i = k+1, \ldots, K-1$. Then, for the $i = k$ case:

$$\mathbb{E}\left[ \sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_k)}}(s_i)\right) \middle| s_k \right]$$

$$= \mathbb{E}\left[ R_k\left(s_k, \delta_k^{\pi_{\mathcal{H}(s_k)}}(s_k)\right) + \mathbb{E}\left[ \sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1} \right] \middle| s_k \right] \tag{34}$$

$$= \mathbb{E}\left[ R_k\left(s_k, \delta_k^{\pi_{\text{pre}}}(s_k)\right) + \mathbb{E}\left[ \sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1} \right] \middle| s_k \right] \tag{35}$$

$$= \mathbb{E}\left[ \sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{pre}}}(s_i)\right) \middle| s_k \right]. \tag{36}$$

Equation (34) follows from the assumption heuristic $\mathcal{H}(\cdot)$ is sequentially consistent and iterated expectations. Equation (35) follows from the definition of pre-decision rollout. Equation (36) holds by the induction hypothesis.

# Proof of Proposition 4

Because heuristic $\mathcal{H}(\cdot)$ is sequentially consistent, it follows from Proposition 2 $\mathcal{H}(\cdot)$ is also sequentially improving. Thus, by Proposition 1, for each rollout policy $\pi$ in $\{\pi_{\text{one}}, \pi_{\text{post}}, \pi_{\text{hy}}\}$,

$$\mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}}(s_0)}(s_i)\right) \middle| s_0\right] \leq \mathbb{E}\left[\sum_{i=0}^{K} R_i\left(s_i, \delta_i^{\pi}(s_i)\right) \middle| s_0\right]. \tag{37}$$

Equations (9), (10), and (11) follow from Proposition 3.

# Proof of Proposition 5

Because heuristic $\mathcal{H}(\cdot)$ is sequentially consistent, for a given action $a$ and state $s_k$, $\mathbb{E}[\sum_{i=k+1}^{K} R_i(s_i, \delta_i^{\pi_{\mathcal{H}}(s_k^a)}(s_i))|s_k] = \mathbb{E}[\sum_{i=k+1}^{K} R_i(s_i, \delta_i^{\pi_{\mathcal{H}}(s_{k+1})}(s_i))|s_k]$. Thus,

$$\begin{aligned}\max_{a\in\mathcal{A}(s_k)} &\left\{R_k(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}}\left(s_k^a\right)}(s_i)\right) \middle| s_k\right]\right\} \\ &= \max_{a\in\mathcal{A}(s_k)} \left\{R_k(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}}(s_{k+1})}(s_i)\right) \middle| s_k\right]\right\}, \end{aligned} \tag{38}$$

which implies

$$R_k\left(s_k, \delta_k^{\pi_{\text{post}}}(s_k)\right) = R_k\left(s_k, \delta_k^{\pi_{\text{one}}}(s_k)\right). \tag{39}$$

The proof proceeds by induction. The result holds trivially for the case $i = K$. We assume the result holds for $i = k+1, \ldots, K-1$. Then, for the case $i = k$:

$$\begin{aligned}&\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{one}}}(s_i)\right) \middle| s_k\right] \\ &= \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\text{one}}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{post}}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \qquad (40) \\ &= \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{post}}}(s_i)\right) \middle| s_k\right] \qquad (41)\end{aligned}$$

Equation (40) holds by the induction hypothesis and iterated expectations. Equation (41) follows from equation (39) and iterated expectations. If $\pi_{\text{one}}$ and $\pi_{\text{post}}$ employ the same rule for breaking ties among actions with equivalent evaluations, then equation (38) implies $\delta_k^{\pi_{\text{one}}} = \delta_k^{\pi_{\text{post}}}$ for $k = 0, 1, \ldots, K$.

# Proof of Proposition 6

The proof is by induction. The result holds trivially for the case $i = K$. We assume the result holds for $i = k+1, \ldots, K-1$. Then, for the case $i = k$:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{best}}}(s_i)\right) \middle| s_k\right] \\
&\leq \max\left\{\mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{best}}}(s_i)\right) \middle| s_k\right],\right. \\
&\qquad \left.\max_{a \in \mathcal{A}(s_k)} \mathbb{E}\left[R_k(s_k, a) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\mathcal{H}(s_{k+1})}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right]\right\} \qquad (42) \\
&= \mathbb{E}\left[R_k\left(s_k, \delta_k^{\pi_{\text{f-one}}}(s_k)\right) + \mathbb{E}\left[\sum_{i=k+1}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{best}}}(s_i)\right) \middle| s_{k+1}\right] \middle| s_k\right] \qquad (43) \\
&\leq \mathbb{E}\left[\sum_{i=k}^{K} R_i\left(s_i, \delta_i^{\pi_{\text{f-one}}}(s_i)\right) \middle| s_k\right]. \qquad (44)
\end{aligned}$$

Equation (42) follows from the outer maximization and iterated expectations, equation (43) from the definition of fortified one-step rollout, and equation (44) from the induction hypothesis and iterated expectations.

# Small DSMKP Example

To complement the application of our rollout algorithm framework to the DSMKP in §5, we present a small example to further illustrate DSMKP system dynamics and the computational tradeoffs among various rollout algorithms. Consider a DSMKP instance with $C = 2$ compartments that, at decision epoch $k$, is in state $s_k = (q_k, Q_k, w_k) = ((5\ 5), 5, (1\ 1))$. Base rewards are $r^1 = 4$ and $r^2 = 2$ with parameters $\eta = 0.25$ and $\gamma = 0.42$. Item sizes are $d^1 = d^2 = 3$. Table 3 shows for each action $a$ in feasible action set $\mathcal{A}(s_k)$ the associated post-decision state $s_k^a$ and the potential pre-decision states $s_{k+1}$ at the subsequent epoch. The one-step rollout procedure of Algorithm 1 requires the greedy procedure of Algorithm 6 be executed 12 times, once from each combination of action and potential subsequent pre-decisions state $s_{k+1}$ at the subsequent decision epoch. The post-decision rollout procedure of Algorithm 2 reduces the number of Algorithm 6 applications to three, once from each post-decision state $s_k^a$. The pre-decision rollout procedure of Algorithm 3 executes Algorithm 6 once from pre-decision state $s_k$.

The number of executions of the heuristic in the hybrid rollout procedure of Algorithm 4 depends on the construction of $\hat{\mathcal{A}}(s_k)$ and $\check{\mathcal{A}}(s_k)$. Assuming we define $\hat{\mathcal{A}}(s_k) = \{\bar{a}_k\}$ and $\check{\mathcal{A}}(s_k) = \{(0\ 0)\}$, the hybrid rollout procedure of Algorithm 4 applies Algorithm 6 three times. First, Algorithm 4 executes the heuristic from the pre-decision state $s_k$ and returns action $\bar{a}_k$. The possibilities for $\bar{a}_k$ are either $(0\ 1)$ or $(1\ 0)$ as the greedy heuristic does not consider $(0\ 0)$, and $(1\ 1)$ is infeasible. Suppose $\bar{a}_k = (0\ 1)$. Then, Algorithm 4 executes the heuristic from the post-decision state $s_k^a$ for all $a \in \{\check{\mathcal{A}}(s_k) \cup \hat{\mathcal{A}}(s_k)\} = \{(0\ 1), (0\ 0)\}$. Note that we include the reject-all action in $\check{\mathcal{A}}(s_k)$ because the greedy heuristic does not consider action $(0\ 0)$.

Table 3: DSMKP System Dynamics

| Action $a$ | Post-Decision State $s_k^a = (q_{k,a}, Q_{k,a})$ | Pre-Decision State $s_{k+1} = (q_{k+1}, Q_k, w_k)$ |
|---|---|---|
| $(0\ 0)$ | $((5\ 5), 5)$ | $((5\ 5), 5, (0\ 0))$<br>$((5\ 5), 5, (0\ 1))$<br>$((5\ 5), 5, (1\ 0))$<br>$((5\ 5), 5, (1\ 1))$ |
| $(0\ 1)$ | $((5\ 2), 2)$ | $((5\ 2), 2, (0\ 0))$<br>$((5\ 2), 2, (0\ 1))$<br>$((5\ 2), 2, (1\ 0))$<br>$((5\ 2), 2, (1\ 1))$ |
| $(1\ 0)$ | $((2\ 5), 2)$ | $((2\ 5), 2, (0\ 0))$<br>$((2\ 5), 2, (0\ 1))$<br>$((2\ 5), 2, (1\ 0))$<br>$((2\ 5), 2, (1\ 1))$ |