Interpretable optimal stopping

Dragos Florin Ciocan
INSEAD; Boulevard de Constance 77305, Fontainebleau, France, florin.ciocan@insead.edu

Velibor V. Mišić
Anderson School of Management, University of California, Los Angeles; 110 Westwood Plaza, Los Angeles, CA 90095, USA, velibor.misic@anderson.ucla.edu

Optimal stopping is the problem of deciding when to stop a stochastic system to obtain the greatest reward, arising in numerous application areas such as finance, healthcare and marketing. State-of-the-art methods for high-dimensional optimal stopping involve approximating the value function or to the continuation value, and then using that approximation within a greedy policy. Although such policies can perform very well, they are generally not guaranteed to be interpretable; that is, a decision maker may not be able to easily see the link between the current system state and the policy's action. In this paper, we propose a new approach to optimal stopping, wherein the policy is represented as a binary tree, in the spirit of naturally interpretable tree models commonly used in machine learning. We formulate the problem of learning such policies from observed trajectories of the stochastic system as a sample average approximation (SAA) problem. We prove that the SAA problem converges under mild conditions as the sample size increases, but that computationally even immediate simplifications of the SAA problem are theoretically intractable. We thus propose a tractable heuristic for approximately solving the SAA problem, by greedily constructing the tree from the top down. We demonstrate the value of our approach by applying it to the canonical problem of option pricing, using both synthetic instances and instances calibrated with real S&P 500 data. Our method obtains policies that (1) outperform state-of-the-art non-interpretable methods, based on simulation-regression and martingale duality, and (2) possess a remarkably simple and intuitive structure.

Key words: optimal stopping; approximate dynamic programming; interpretability; decision trees; option pricing.
History:

1. Introduction

We consider the problem of optimal stopping, which can be described as follows: a system evolves stochastically from one state to another in discrete time steps. At each decision epoch, a decision maker (DM) chooses whether to stop the system or to allow it to continue for one more time step. If the DM chooses to stop the system, she garners a reward that is dependent on the current state of the system; if she chooses to allow it to continue, she does not receive any reward in the current period, but can potentially stop it at a future time to obtain a higher reward. The DM must specify a policy, which prescribes the action to be taken (stop/continue) for each state the system may enter. The optimal stopping problem is to find the policy that achieves the highest possible reward in expectation.

The optimal stopping problem is a key problem in stochastic control and arises in many important applications; we name a few below:

1. **Option pricing**. One of the most important applications of optimal stopping is to the pricing of financial options that allow for early exercise, such as American and Bermudan options. The system is the collection of underlying securities (typically stocks) that the option is written on. The prices of the securities comprise the system state. The decision to stop the system corresponds to exercising the option and receiving the corresponding payoff; thus, the problem of obtaining the highest expected payoff from a given option corresponds to finding an optimal stopping policy. The highest expected payoff attained by such an optimal stopping (optimal exercise) policy is then the price that the option writer should charge for the option.

2. **Healthcare**. Consider an organ transplant patient waiting on a transplant list, who may be periodically offered an organ for transplantation. In this context, the system corresponds to the patient and the currently available organ, and the system state describes the patient's health and the attributes of the organ. The decision to stop corresponds to the patient accepting the organ, where the reward is the estimated quality-adjusted life years (QALYs) that the patient will garner upon transplantation. The problem is then to find a policy that prescribes for a given patient and a given available organ whether the patient should accept the organ, or wait for the next available organ, so as to maximize the QALYs gained from the transplant.
3. **Marketing**. Consider a retailer selling a finite inventory of products over some finite time horizon. The system state describes the remaining inventory of the products, which evolves over time as customers buy the products from period to period. The action of stopping the system corresponds to starting a price promotion that will last until the end of the finite time horizon. The problem is to decide at each period, based on the remaining inventory, whether to commence the promotion, or to wait one more period, in order to maximize the total revenue garnered by the end of the horizon.

Large-scale optimal stopping problems that occur in practice are typically solved by approximate dynamic programming (ADP) methods. The goal in such ADP methods is to approximate the optimal value function that, for a given system state, specifies the best possible expected reward that can be attained when one starts in that state. With an approximate value function in hand, one can obtain a good policy by considering the greedy policy with respect to the approximate value function.

In ADP methods, depending on the approximation architecture, the approximate value function may provide some insight into which states of the system state space are more desirable. However, the policy that one obtains by being greedy with respect to the approximate value function need not have any readily identifiable structure. This is disadvantageous, because in many optimal stopping problems, we are not only interested in policies that attain high expected reward, but also policies that are *interpretable*. A policy that is interpretable is one where we can directly see how the state of the system maps to the recommended action, and the relation between state and action is sufficiently transparent.

Interpretability is desirable for three reasons. First, in modeling a real world system, it is useful to obtain some insight about what aspects of the system state are important for controlling it optimally or near optimally. Second, a complex policy, such as a policy that is obtained via an ADP method, may not be operationally feasible in many real life contexts. Lastly – and most importantly – in many application domains where the decision maker is legally responsible or accountable for the action taken by a policy, interpretability is not merely a desirable feature, but a requirement: in such settings a decision maker will simply not adopt the policy without the ability to explain the policy's mechanism of action. There is moreover a regulatory push to increase the transparency and interpretability of customer facing data-driven algorithms; as an example, General Data Protection Regulation rules set by the EU (Doshi-Velez and Kim 2017) dictate that algorithms which can differentiate between users must provide explanations for their decisions if such queries arise.

In this paper, we consider the problem of constructing interpretable optimal stopping policies from data. Our approach to interpretability is to consider policies that are representable as a binary tree. In such a tree, each leaf node is an action (stop or go), and each non-leaf node (also called a *split* node) is a logical condition in terms of the state variables which determines whether we proceed to the left or the right child of the current node. To determine the action we should take, we take the current state of the system, run it down the tree until we reach a leaf, and take the action prescribed in that leaf. An example of such a tree-based policy is presented in Figure 1a. Policies of this kind are simple, and allow the decision maker to directly see the link between the current system state and the action.

Before delving into our results, we comment on how one might compare such an aforementioned interpretable policy with an optimal or, possibly, near-optimal heuristic policy. Our informal goal

is to look for the best policy within the constraints of an interpretable policy architecture, which in our case is tree-based policies. However, without any a priori knowledge that a given optimal stopping problem is "simple enough", one would expect that enforcing that the policy architecture be interpretable carries a performance price; in other words, interpretable policies should generally not perform as well as a state-of-the-art heuristic. On the other hand, one can hope that there exist stopping problems where the price of interpretability is low, in that interpretable policies, while sub-optimal, do not carry a large optimality gap. The present paper is an attempt to (a) exhibit stopping problems of practical interest for which tree-based policies attain near-optimal performance, along with being interpretable and (b) provide an algorithm to find such interpretable policies directly from data.

We make the following specific contributions:

1. **Sample average approximation.** We formulate the problem of learning an interpretable tree-based policy from a sample of trajectories of the system (sequences of states and payoffs) as a sample average approximation (SAA) problem. To the best of our knowledge, the problem of directly learning a policy in the form of a tree for optimal stopping problems (and Markov decision processes more generally) has not been studied before. We show that under mild conditions, the tree policy SAA problem defined using a finite sample of trajectories converges almost surely in objective value to the tree policy problem defined with the underlying stochastic process, as the number of trajectories available as data grows large.
2. **Computational tractability.** From a computational complexity standpoint, we establish that three natural simplifications of the SAA problem are NP-Hard and thus finding good solutions to the SAA problem is challenging.

   In response, we present a computationally tractable methodology for solving the learning problem. Our method is a construction algorithm: starting from a degenerate tree consisting of a single leaf, the algorithm grows the tree in each iteration by splitting a single leaf into two new child leaves. The split is chosen greedily, and the algorithm stops when there is no longer a sufficient improvement in the sample-based reward. Key to the procedure is determining the optimal split point at a candidate split; we show that this problem can be solved in a computationally efficient manner. In this way, the overall algorithm is fully data-driven, in that the split points are directly chosen from the data and are not artificially restricted to a collection of split points chosen a priori. While the algorithm resembles top-down tree induction methods from classification/regression, several important differences arising from the temporal nature of the problem make this construction procedure algorithmically nontrivial.
3. **Practical performance versus state-of-the-art ADP methods.** We numerically demonstrate the value of our methodology by applying it to the problem of pricing a Bermudan option, which is a canonical optimal stopping problem in finance. We show that our tree policies outperform two state-of-the-art approaches, namely the simulation-regression approach of Longstaff and Schwartz (2001) and the martingale duality-based approach of Desai et al. (2012b). At the same time, we also show that the tree policies produced by our approach are remarkably simple and intuitive. We further investigate the performance of our approach by testing it on a stylized one-dimensional optimal stopping problem (not drawn from option pricing), where the exact optimal policy can be computed; in general, our policy is either optimal or very close to optimal.

The rest of this paper is organized as follows. In Section 2, we discuss the relevant literature in ADP, machine learning and interpretable decision making. In Section 3, we formally define our optimal stopping problem and its sample-based approximation, we define the problem of finding a tree policy from sample data, and analyze its complexity. In Section 4, we present a heuristic procedure for greedily constructing a tree directly from data. In Section 5, we present an extensive computational study in option pricing comparing our algorithm to alternate approaches. In Section 6, we evaluate our algorithm on the aforementioned one-dimensional problem. Finally, we conclude in Section 7.

2. Literature review

Our paper relates to three different broad streams of research: the optimal stopping and ADP literature; the machine learning literature; and the multi-stage stochastic/robust optimization literature. We survey each of these below.

Approximate dynamic programming (ADP). ADP has been extensively studied in the operations research community since the mid-1990s as a solution technique for Markov decision processes (Powell 2007, Van Roy 2002). In the last fifteen years, there has been significant interest in solving MDPs by approximating the linear optimization (LO) model; at a high level, one formulates the MDP as a LO problem, reduces the number of variables and constraints in a tractable manner, and solves the more accessible problem to obtain a value function approximation. Some examples of this approach include De Farias and Van Roy (2003), Adelman and Mersereau (2008), Desai et al. (2012a) and Bertsimas and Mišić (2016).

A large subset of the ADP research literature, originating in both the operations research and finance communities, has specifically studied optimal stopping problems. The seminal papers of Carriere (1996), Longstaff and Schwartz (2001) and Tsitsiklis and Van Roy (2001) propose simulation-regression approaches, where one simulates trajectories of the system state and uses regression to compute an approximation to the optimal continuation value at each step. Later research has considered the use of martingale duality techniques. The idea in such approaches is to relax the non-anticipativity requirement of the policy by allowing the policy to use future information, but to then penalize policies that use this future information, in the spirit of Lagrangean duality. This approach yields upper bounds on the optimal value and can also be used to derive high quality stopping policies. Examples include Rogers (2002), Andersen and Broadie (2004), Haugh and Kogan (2004), Chen and Glasserman (2007), Brown et al. (2010) and Desai et al. (2012b).

Our approach differs from these generic ADP approaches and optimal stopping-specific approaches in two key ways. First, general purpose ADP approaches, as well as those specifically designed for optimal stopping, are focused on obtaining an approximate value function or an upper bound on the value function, which is then used in a greedy manner. In contrast, our approach involves optimizing over a policy directly, without computing/optimizing over a value function. Second, our approach is designed with interpretability in mind, and produces a policy that can be easily visualized as a binary tree. In contrast, previously proposed ADP methods are not guaranteed to result in policies that are interpretable.

Lastly, we note that some methods in finance for pricing options with early exercise involve tree representations; examples include the binomial lattice approach (Cox et al. 1979) and the random tree method (Broadie and Glasserman 1997). However, the trees found in these methods represent discretizations of the sample paths of the underlying asset prices, which provide a tractable way to perform scenario analysis. In contrast, the trees in our paper represent policies, not sample paths. As such, our approach is unrelated to this prior body of work.

Machine learning. Interpretability has been a goal of major interest in the machine learning community, starting with the development of decision trees in the 1980s (Breiman et al. 1984, Quinlan 1986, 1993). A stream of research has considered interpretable scoring rules for classification and risk prediction; recent examples include Ustun and Rudin (2015, 2016) and Zeng et al. (2017). Another stream of research considers the design of disjunctive rules and rule lists; recent examples include Wang et al. (2015), Wang and Rudin (2015), Wang et al. (2017), Letham et al. (2015), Angelino et al. (2017) and Lakkaraju et al. (2016).

The algorithm we will present is closest in spirit to classical tree algorithms like CART and ID3. Our algorithm differs from these prior methods in that it is concerned with optimal stopping, which is a stochastic control problem that involves making a decision over time, and is fundamentally different from classification and regression. In particular, a key part of estimating a classification or regression tree is determining the leaf labels, which in general is a computationally simple task. As we will see in Section 3.5, the analogous problem in the optimal stopping realm is NP-Hard. As a result, this leads to some important differences in how the tree construction must be done to account for the temporal nature of the problem; we comment on these in more detail in Section 4.3.

Interpretable decision making. In the operations research community, there is growing interest in interpretability as it pertains to dynamic decision making; we provide some recent examples here. With regard to dynamic problems, Bertsimas et al. (2013) considers the problem of designing a dynamic allocation policy for allocating deceased donor kidneys to patients requiring a kidney transplant that maximizes efficiency while respecting certain fairness constraints. To obtain a policy that is sufficiently interpretable to policy makers, Bertsimas et al. (2013) further propose using ordinary least squares regression to find a scoring rule that predicts the fairness-adjusted match quality as a function of patient and donor characteristics. In more recent work, Azizi et al. (2018) consider a related approach for dynamically allocating housing resources to homeless youth. The paper proposes a general mixed-integer optimization framework for selecting an interpretable scoring rule (specifically, linear scoring rules, decision tree rules with axis-aligned or oblique splits, or combinations of both linear and decision tree rules) for prioritizing youth on the waiting list. Lastly, the paper of Bravo and Shaposhnik (2017) considers the use of machine learning for analyzing optimal policies to MDPs. The key idea of the paper is to solve instances of a given MDP to optimality, and then to use the optimal policies as inputs to machine learning methods. The paper applies this methodology to classical problems such as inventory replenishment, admission control and multi-armed bandits. This approach differs from ours, in that we do not have access to the optimal policy; in fact, the goal of our method is to directly obtain a near-optimal policy.

3. Problem definition

We begin by defining the optimal stopping problem in Section 3.1, and its sample-based counterpart in Section 3.2. We then define the tree policy sample-average approximation (SAA) problem, where the class of policies is restricted to those that can be represented by a binary tree, in Section 3.3. We show that the tree policy SAA problem converges in Section 3.4. Finally, in Section 3.5, we show that the tree policy SAA problem is NP-Hard when one considers three specific simplifications.

3.1. Optimal stopping model

Consider a system with state given by $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \triangleq \mathcal{X} \subseteq \mathbb{R}^n$. We let $\mathbf{x}(t)$ denote the state of the system at time t, which evolves according to some stochastic process. We let $\mathcal{A} = \{\textbf{stop}, \textbf{go}\}$ denote the action space of the problem; we may either stop the system (**stop**) or allow it to continue for one more time step (**go**). We assume a finite horizon problem with T periods, starting at period $t = 1$. We let $g(t, \mathbf{x})$ denote the reward or payoff from stopping the system when the current period is t and the current system state is $\mathbf{x}$. We assume that all rewards are discounted by a factor of β for each period.

We define a policy π as a mapping from the state space $\mathcal{X}$ to the action space $\mathcal{A}$. We let $\Pi = \{\pi \,|\, \pi : [T] \times \mathcal{X} \to \mathcal{A}\}$ be the set of all possible policies, where we use the notation $[N] = \{1, \ldots, N\}$ for any integer N. For a given realization of the process $\{\mathbf{x}(t)\}_{t=1}^T$, we define the stopping time τ_π as the first time at which the policy π prescribes the action **stop**:

$$\tau_\pi = \min\{t \in [T] \,|\, \pi(t, \mathbf{x}(t)) = \textbf{stop}\}, \tag{1}$$

where we take the minimum to be $+\infty$ if the set is empty. Our goal is to find the policy that maximizes the expected discounted reward over the finite horizon, which can be represented as the following optimization problem:

$$\underset{\pi \in \Pi}{\text{maximize}}\ \mathbb{E}\left[\beta^{\tau_\pi - 1} \cdot g(\tau_\pi, \mathbf{x}(\tau_\pi)) \mid \mathbf{x}(1) = \mathbf{x}\right]. \tag{2}$$

For any policy π, we let $J^\pi(\mathbf{x}) \triangleq \mathbb{E}\left[\beta^{\tau_\pi - 1} \cdot g(\tau_\pi, \mathbf{x}(\tau_\pi)) \mid \mathbf{x}(1) = \mathbf{x}\right]$. For simplicity, we assume that there exists a starting state $\bar{\mathbf{x}}$ at which the system is started, such that $\mathbf{x}(1) = \bar{\mathbf{x}}$ always.

3.2. Sample average approximation

In order to solve problem (2), we need to have a full specification of the stochastic process $\{\mathbf{x}(t)\}_{t=1}^T$. In practice, we may not have this specification or it may be too difficult to work with directly. Instead of this specification, we may instead have data, that is, we may have access to specific realizations or trajectories of the process $\{\mathbf{x}(t)\}_{t=1}^T$. In this section, we describe a *sample-average approximation* (SAA) formulation of the optimal stopping problem (2) that will allow us to design a policy directly from these trajectories, as opposed to a probabilistic definition of the stochastic process.

We will assume that we have a set of $\Omega \in \mathbb{N}^+$ trajectories, indexed by $\omega \in [\Omega]$. We denote the state of the system in trajectory ω at time t by $\mathbf{x}(\omega, t)$. Thus, each trajectory ω corresponds to a sequence of system states $\mathbf{x}(\omega,1), \mathbf{x}(\omega,2), \ldots, \mathbf{x}(\omega,T)$, with $\mathbf{x}(\omega,1) = \bar{\mathbf{x}}$.

We can now define a sample-average approximation (SAA) version of the optimal stopping problem (2). Let $\tau_{\pi,\omega}$ denote the time at which the given policy π recommends that the system be stopped in the trajectory ω; mathematically, it is defined as

$$\tau_{\pi,\omega} = \min\{t \in [T] \,|\, \pi(t, \mathbf{x}(\omega,t)) = \textbf{stop}\}, \tag{3}$$

where we again take the minimum to be $+\infty$ if the set is empty. Our SAA optimal stopping problem can now be written as

$$\underset{\pi \in \Pi}{\text{maximize}} \; \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \beta^{\tau_{\pi,\omega}-1} \cdot g(\tau_{\pi,\omega}, \mathbf{x}(\omega, \tau_{\pi,\omega})). \tag{4}$$

Note that, in order to handle the case when π does not stop on a given trajectory, we define $\beta^{\tau_{\pi,\omega}-1}$ to be 0 if $\tau_{\pi,\omega} = +\infty$. Lastly, we introduce the following short-hand notation for the sample average value of a policy π:

$$\hat{J}^{\pi}(\bar{\mathbf{x}}) \triangleq \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \beta^{\tau_{\pi,\omega}-1} \cdot g(\tau_{\pi,\omega}, \mathbf{x}(\omega, \tau_{\pi,\omega})).$$

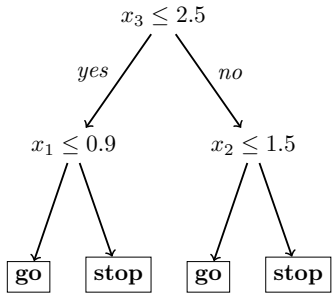
3.3. Tree policies

Problem (4) defines an approximation of the original problem (2) that uses data – specifically, a finite sample of trajectories of the stochastic process $\{\mathbf{x}(t)\}_{t=1}^T$. However, despite this simplification that brings the problem closer to being solvable in practice, problem (4) is still difficult because it is an optimization problem over the set of all possible stopping policies Π. Moreover, as discussed in Section 1, we wish to restrict ourselves to policies that are sufficiently simple and interpretable.

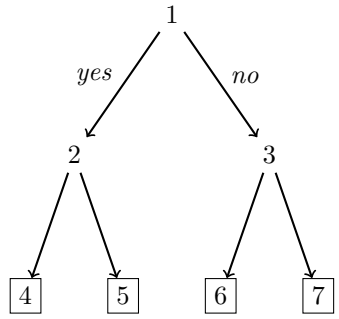
In this section, we will define the class of *tree policies*. A tree policy is specified by a binary tree that corresponds to a recursive partitioning of the state space $\mathcal{X}$. Each tree consists of two types of nodes: split nodes and leaf nodes. Each split node is associated with a query of the form $x_i \leq \theta$; we call the state variable x_i that participates in the query the *split variable*, the index i the *split variable index* and the constant value θ in the inequality the *split point*. If the query is true, we proceed to the left child of the current split node; otherwise, if it is false, we proceed to the right child.

We let $\mathcal{N}$ denote the set of all nodes (splits and leaves) in the tree. We use **splits** to denote the set of split nodes and **leaves** to denote the set of leaf nodes. We define the functions $\textbf{leftchild} : \textbf{splits} \to \mathcal{N}$ and $\textbf{rightchild} : \textbf{splits} \to \mathcal{N}$ to indicate the left and right child nodes of each split node, i.e., for a given split node s, $\textbf{leftchild}(s)$ is its left child and $\textbf{rightchild}(s)$ is its right child. We use $\mathcal{T}$ to denote the *topology* of the tree, which we define as the tuple $\mathcal{T} = (\mathcal{N}, \textbf{leaves}, \textbf{splits}, \textbf{leftchild}, \textbf{rightchild})$.

Given the topology $\mathcal{T}$, we use $\mathbf{v} = \{v(s)\}_{s \in \textbf{splits}}$ and $\boldsymbol{\theta} = \{\theta(s)\}_{s \in \textbf{splits}}$ to denote the collection of all split variable indices and split points, respectively, where $v(s)$ is the split variable index and $\theta(s)$ is the split point of split s. We let $\mathbf{a} = \{a(\ell)\}_{\ell \in \textbf{leaves}}$ denote the collection of leaf actions, where $a(\ell) \in \mathcal{A}$ is the action we take if the current state is mapped to leaf ℓ. A complete tree is therefore



(a) Tree policy.



(b) Node indices for tree topology. (Numbers indicate index of node in $\mathcal{N}$.)

Figure 1 Visualization of tree policy in Example 1, for $\mathcal{X} = \mathbb{R}^3$.

specified by the tuple $(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$, which specifies the tree topology, the split variable indices, the split points and the leaf actions.

Given a complete tree $(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$, we let $\ell(\mathbf{x}; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta})$ denote the leaf in **leaves** that the system state $\mathbf{x} \in \mathcal{X}$ is mapped to, and define the stopping policy $\pi(\cdot; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$ by taking the action of the leaf to which $\mathbf{x}$ is mapped:

$$\pi(\mathbf{x}; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a}) = a(\, \ell(\mathbf{x}; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta})\,).$$

We provide an example of a tree policy below.

EXAMPLE 1. Consider a system where $\mathcal{X} = \mathbb{R}^3$, for which the policy is the tree given in Figure 1a. In this example, suppose that we number the nodes with the numbers 1 through 7, from top to bottom, left to right. Then, $\mathcal{N} = \{1, \ldots, 6\}$, $\textbf{leaves} = \{4, 5, 6, 7\}$ and $\textbf{splits} = \{1, 2, 3\}$. The **leftchild** and **rightchild** mappings are:

$$\begin{aligned} &\textbf{leftchild}(1) = 2, && \textbf{rightchild}(1) = 3, \\ &\textbf{leftchild}(2) = 4, && \textbf{rightchild}(2) = 5, \\ &\textbf{leftchild}(3) = 6, && \textbf{rightchild}(3) = 7. \end{aligned}$$

The topology with the split and leaf labels is visualized in Figure 1b.

The split variable indices and split points for the split nodes $\{1, 2, 3\}$ are

$$\begin{aligned} &v(1) = 3, && \theta(1) = 2.5, \\ &v(2) = 1, && \theta(2) = 0.9, \\ &v(3) = 2, && \theta(3) = 1.5; \end{aligned}$$

and the leaf actions for the leaf nodes $\{4, 5, 6, 7\}$ are

$$\begin{aligned} &a(4) = \textbf{go}, && a(6) = \textbf{go}, \\ &a(5) = \textbf{stop}, && a(7) = \textbf{stop}. \end{aligned}$$

As an example, suppose that the current state of the system is $\mathbf{x} = (1.2, 0.8, 2.2)$. To map this observation to an action, we start at the root and check the first query, $x_3 \leq 2.5$. Since $x_3 = 2.2$, the query is true, and we proceed to the left child of the root node. This new node is again a split, so we check its query, $x_1 \leq 0.9$. Since $x_1 = 1.2$, this query is false, so we proceed to its right child node, which is a leaf. The action of this leaf is **stop**, and thus our policy stops the system. $\square$

Letting Π_{tree} denote the set of all tree policies specified as above, we wish to find the tree policy that optimizes the sample-average reward:

$$\underset{\pi \in \Pi_{\text{tree}}}{\text{maximize}} \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \beta^{\tau_{\pi,\omega}-1} \cdot g(\tau_{\pi,\omega}, \mathbf{x}(\omega, \tau_{\pi,\omega})). \tag{5}$$

We refer to this problem as the *tree policy SAA problem*. In addition, we also define the counterpart of problem (2) restricted to policies in Π_{tree}, which we refer to simply as the *tree policy problem*:

$$\underset{\pi \in \Pi_{\text{tree}}}{\text{maximize}} \mathbb{E}\left[\beta^{\tau_\pi - 1} \cdot g(\tau_\pi, \mathbf{x}(\tau_\pi)) \mid \mathbf{x}(1) = \mathbf{x}\right], \tag{6}$$

where τ_π is defined as in Section 3.1.

We comment on two aspects of this modeling approach. First, the class of tree policies, as defined above, are stationary: the policy's behavior does not change with the period t. This turns out to not be a limitation, because it is always possible to augment the state space with an additional state variable to represent the current period t. By then allowing the tree to split on t, it is possible to obtain a time-dependent policy. We will follow this approach in our numerical experiments with option pricing in Section 5 and the stylized one-dimensional problem in Section 6.

Second, our approach requires access to trajectories that are *complete*, that is, the trajectories are not terminated early/censored by the application of some prior policy, and the system state is known at every $t \in [T]$. In the case of censored trajectories, our methodology can potentially be applied by first fitting a stochastic model to the available trajectories and then simulating the model to fill in the missing data. The extension of our methodology to censored trajectories is beyond the scope of this paper, and left to future research.

3.4. Convergence of tree policy SAA problem

A natural expectation for the SAA problem (5) is that its objective approaches the optimal value of (6), as the number of trajectories Ω available as samples increases. In this section, we show that this is indeed a property of our SAA problem. More precisely, we show that in the limit of $\Omega \to \infty$ and if we restrict the policy class Π_{tree} to only trees of arbitrary *bounded* depth, the optimal value of the SAA problem converges to the value of the optimal tree-based policy for the true problem almost surely.

We note that establishing such a convergence result in the case of tree-based policies is challenging due to two difficulties. First, a tree policy is defined by a tuple $(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$; the space of such tuples could in general be uncountable, and thus we cannot directly invoke the strong law of large numbers to guarantee that almost sure convergence holds simultaneously over all $\pi \in \Pi_{\text{tree}}$. Secondly, invoking the strong law of large numbers over all members of a finite cover of this space is also non-trivial. This is due to the fact that $\hat{J}^\pi(\bar{\mathbf{x}})$ is not necessarily continuous in π, even as we change the split points $\boldsymbol{\theta}$ and keep all other tree parameters constant, and as such we cannot rely on Lipschitz continuity arguments.

In order to handle these challenges, we restrict our convergence analysis to the class of tree policies corresponding to trees of depth at most d for some finite parameter d. We denote this class of tree policies by $\Pi_{\text{tree}}(d) \subseteq \Pi_{\text{tree}}$. This restriction limits the set of $(\mathcal{T}, \mathbf{v}, \mathbf{a})$ parameters specifying a tree-based policy to be finite, although the $\boldsymbol{\theta}$ parameters still lie in a potentially uncountable set. We remark that, because our focus is on interpretable policies, limiting the potential depth of allowable trees that our SAA approach can optimize over is reasonable.

We also make some relatively mild assumptions regarding the underlying structure of the optimal stopping problem, which facilitate the subsequent analysis. The first two are relatively standard, enforcing the compactness of the state space $\mathcal{X}$, as well as placing a universal upper bound on the magnitude of the cost function $g(\cdot, \cdot)$:

Assumption 1. *The state space $\mathcal{X}$ is compact.*

ASSUMPTION 2. *There exists a constant G such that for any $t \in [T]$, $\mathbf{x} \in \mathcal{X}$, $0 \leq g(t, \mathbf{x}) \leq G$.*

The third assumption essentially imposes that the distributions of the marginals of the state variable are sufficiently smooth. In particular, we assume that the probability that the v-th component of the state variable, x_v, lies in some interval $[a, b]$ can be bounded by a function which only depends on $|b-a|$, but not on a and b specifically, and which can be controlled to go to zero as the width of the interval $[a, b]$ vanishes.

ASSUMPTION 3. *There exists a function $f : [0, \infty) \to [0, 1]$ such that,*
1. $\Pr[x_v(t) \in [a, b]] \leq f(|b-a|)$, *for every $t \in [T], v \in [n]$, and any $[a, b] \subseteq \mathcal{X}_v$ with $b > a$.*
2. $f(0) = 0$.
3. *f is strictly increasing and continuous.*

We add that Assumption 3 is satisfied by distributions with probability density functions that can be uniformly upper bounded across their entire domain.

Having set up these assumptions, the following theorem proves almost sure convergence of $\hat{J}^\pi(\bar{\mathbf{x}})$ to $J^\pi(\bar{\mathbf{x}})$ over all policies π in $\Pi_{\text{tree}}(d)$.

THEOREM 1. *For any finite tree depth d, initial state $\bar{\mathbf{x}}$ and arbitrary $\epsilon > 0$, almost surely, there exists a finite sample size Ω_0 such that for all $\Omega \geq \Omega_0$ and all tree policies $\pi \in \Pi_{\text{tree}}(d)$,*

$$\left| J^\pi(\bar{\mathbf{x}}) - \hat{J}^\pi(\bar{\mathbf{x}}) \right| \leq \epsilon.$$

Theorem 1 is the main technical result of this section and enables us to prove our desired convergence of the SAA optimal objective. This is stated in the following corollary:

COROLLARY 1. *For any finite tree depth d, initial state $\bar{\mathbf{x}}$ and arbitrary $\epsilon > 0$, almost surely, there exists a finite sample size Ω_0 such that for all $\Omega \geq \Omega_0$,*

$$\left| \sup_{\pi \in \Pi_{\text{tree}}(d)} J^\pi(\bar{\mathbf{x}}) - \sup_{\pi \in \Pi_{\text{tree}}(d)} \hat{J}^\pi(\bar{\mathbf{x}}) \right| \leq \epsilon.$$

The corollary above shows that the decision maker indeed obtains via solving the sample average problem (5), a policy whose value approximates with arbitrarily high precision the value of the best tree-based policy for the true problem, as long as the decision maker has access to a sufficiently large sample of state trajectories.

As the proofs of Theorem 1 and Corollary 1 are quite involved, we present them in Section EC.1.1.

3.5. Complexity of tree policy SAA problem

Having established that the tree policy SAA problem converges to the exact tree policy optimization problem, we now turn our attention to the computational complexity of solving the tree policy SAA problem. In this section, we will consider three simplified versions of problem (5) and show that each of these is theoretically intractable.

To motivate the first of our intractability results, observe that problem (5) allows for the tree policy to be optimized along all four dimensions: the topology $\mathcal{T}$, the split variable indices $\mathbf{v}$, the split points $\boldsymbol{\theta}$ and the leaf actions $\mathbf{a}$. Rather than optimizing over all four variables, let us consider instead a simplified version of problem (5), where the topology $\mathcal{T}$, split variable indices $\mathbf{v}$ and split points $\boldsymbol{\theta}$ are fixed, and the leaf actions $\mathbf{a}$ are the only decision variables. We use $\Pi(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta})$ to denote the set of all policies with the given $\mathcal{T}$, $\mathbf{v}$ and $\boldsymbol{\theta}$, i.e.,

$$\Pi(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta}) = \{\pi(\cdot; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a}) \,|\, \mathbf{a} \in \{\mathbf{stop}, \mathbf{go}\}^{\mathbf{leaves}}\}. \tag{7}$$

The *leaf action SAA problem* is to find the policy in $\Pi(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta})$ that optimizes the sample average reward:

$$\underset{\pi \in \Pi(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta})}{\text{maximize}} \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \beta^{\tau_{\pi,\omega}-1} \cdot g(\tau_{\pi,\omega}, \mathbf{x}(\omega, \tau_{\pi,\omega})). \tag{8}$$

While problem (5) is difficult to analyze in generality, the simpler problem (8) is more amenable to analysis. It turns out, perhaps surprisingly, that this simplified problem is already hard to solve:

PROPOSITION 1. *The leaf action problem* (8) *is NP-Hard.*

The proof of this result (see Section EC.1.2.1) follows by a reduction from the minimum vertex cover problem.

This result is significant for two reasons. First, it establishes that even in this extremely simplified case, where we have already selected a tree topology, split variable indices and split points, the resulting problem is theoretically intractable.

Second, this result points to an important distinction between optimal stopping tree policies and classification trees that are used in machine learning. For binary classification, a classification tree consists of a tree topology, split variable indices, split points and leaf labels. Given $\mathcal{T}$, $\mathbf{v}$ and $\boldsymbol{\theta}$, determining the leaf labels $\mathbf{a}$ that minimize the 0-1 classification error on a training sample is trivial: for each leaf, we simply predict the class that is most frequent among those observations that are mapped to that leaf. More importantly, each leaf's label can be computed independently. The same holds true for the regression setting, where $a(\ell)$ is the continuous prediction of leaf ℓ: to find the values of $\mathbf{a}$ that minimize the squared prediction error, we can set each $a(\ell)$ as the average of the dependent variable for all training observations that are mapped to that leaf.

For optimal stopping, the situation is strikingly different. Determining the leaf actions is much more difficult because a single trajectory has a time dimension: as time progresses, a tree policy may map the current state of that trajectory to many different leaves in the tree. As a result, the decision of whether to stop or not in one leaf (i.e., to set $a(\ell) = \textbf{stop}$ for a leaf ℓ) cannot be made independently of the decisions to stop or not in the other leaves: the leaf actions are coupled together. For example, if we set $a(\ell) = \textbf{stop}$ for a given leaf ℓ and $a(\ell') = \textbf{stop}$ for a different leaf ℓ', then a trajectory that reaches ℓ before it reaches ℓ' could never stop at ℓ': thus, whether we choose to stop at ℓ' depends on whether we choose to stop at ℓ.

Proposition 1 establishes that when the topology, split variable indices and split points are fixed, optimizing over the leaf actions is an intractable problem. It turns out that optimizing individually over the split variable indices and the split points, with the rest of the tree policy parameters fixed, is also intractable. Let $\Pi(\mathcal{T}, \boldsymbol{\theta}, \mathbf{a})$ denote the set of all tree policies with the given $\mathcal{T}$, $\boldsymbol{\theta}$ and $\mathbf{a}$ (i.e., the split variable indices $\mathbf{v}$ may be optimized over), and let $\Pi(\mathcal{T}, \mathbf{v}, \mathbf{a})$ denote the set of all tree policies with the given $\mathcal{T}$, $\mathbf{v}$ and $\mathbf{a}$ (i.e., the split points $\boldsymbol{\theta}$ may be optimized over):

$$\Pi(\mathcal{T}, \boldsymbol{\theta}, \mathbf{a}) = \{\pi(\cdot; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a}) \,|\, \mathbf{v} \in [n]^{\textbf{splits}}\}, \tag{9}$$

$$\Pi(\mathcal{T}, \mathbf{v}, \mathbf{a}) = \{\pi(\cdot; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a}) \,|\, \boldsymbol{\theta} \in \mathbb{R}^{\textbf{splits}}\}. \tag{10}$$

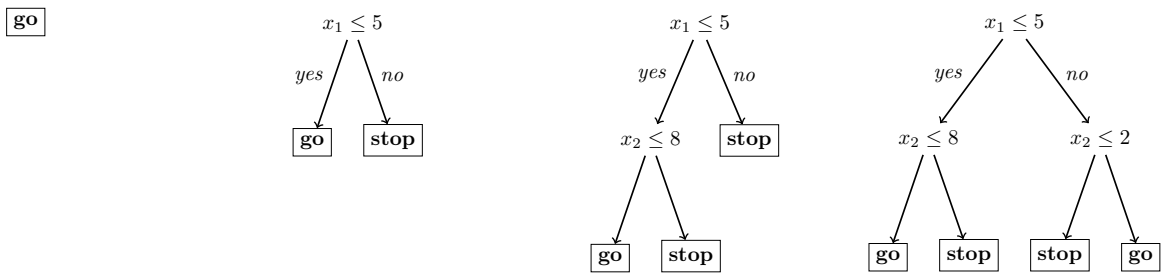
Let us define the *split variable index SAA problem* as the problem of finding a policy in $\Pi(\mathcal{T}, \boldsymbol{\theta}, \mathbf{a})$ to maximize the sample-based reward:

$$\underset{\pi \in \Pi(\mathcal{T}, \boldsymbol{\theta}, \mathbf{a})}{\text{maximize}} \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \beta^{\tau_{\pi,\omega}-1} \cdot g(\tau_{\pi,\omega}, \mathbf{x}(\omega, \tau_{\pi,\omega})). \tag{11}$$

Similarly, let us define the *split point SAA problem* as the analogous problem of optimizing over policies in $\Pi(\mathcal{T}, \mathbf{v}, \mathbf{a})$:

$$\underset{\pi \in \Pi(\mathcal{T}, \mathbf{v}, \mathbf{a})}{\text{maximize}} \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \beta^{\tau_{\pi,\omega}-1} \cdot g(\tau_{\pi,\omega}, \mathbf{x}(\omega, \tau_{\pi,\omega})). \tag{12}$$

We then have the following two intractability results.



(a) Iteration 0. (b) Iteration 1. (c) Iteration 2. (d) Iteration 3.

Figure 2 Example of evolution of tree with each iteration of construction procedure (Algorithm 1).

PROPOSITION 2. *The split variable index SAA problem* (11) *is NP-Hard.*

PROPOSITION 3. *The split point SAA problem* (12) *is NP-Hard.*

Like Proposition 1, the proofs of Propositions 2 and 3, found in Sections EC.1.2.2 and EC.1.2.3 respectively, also follow by a reduction from the minimum vertex cover problem.

4. Construction algorithm

As we saw in Section 3.3, three major simplifications of our SAA problem are theoretically intractable. Expanding the scope of the problem to allow for joint optimization over the split points, split variable indices and leaf actions, as well as the topology, renders the problem even more difficult. Given the difficulty of this problem, we now present a practical heuristic algorithm for approximately solving the SAA problem. Our algorithm is a greedy procedure that grows/induces the tree from the top down. In Section 4.1, we provide a description of the overall algorithm. In Section 4.2, we provide a procedure for performing the key step of our construction algorithm, which is finding the optimal split point. Finally, in Section 4.3, we compare our construction algorithm and extant classification tree approaches such as CART (Breiman et al. 1984).

4.1. Algorithm description

Our algorithm to heuristically solve (5) is a construction procedure where we greedily grow a tree up to the point where we no longer observe an improvement by adding another split point. At a high level, our algorithm works as follows. We start from a degenerate tree, consisting of a single leaf with the action of **go** as the root node. At this leaf, we consider placing a split. Such a split will result in a left child and a right child, with both child nodes being leaves. For each potential split variable, we find the best possible split point assuming that the left child leaf will be **stop** (and the right child will be **go**), and assuming that the right child leaf will be **stop** (and the left child will be **go**). We find the best possible combination of the split variable, split point and direction (left child is **stop** and right child is **go**, or right child is **stop** and left child is **go**), and add the split to the tree. We continue the procedure if the split resulted in a sufficient improvement on the current objective; otherwise, we terminate the procedure.

In the next iteration, we repeat the process, except that now we also optimize over the leaf that we select to split: we compute the best possible split at each leaf in the current tree, and take the best split at the best leaf. We continue in this way until there is no longer sufficient improvement in an iteration. At each iteration, we expand the tree at a single leaf and add two new leaves to the tree. Figure 2 provides a visualization of several iterations of the algorithm.

The notion of sufficient improvement that we use in our algorithm is that of relative improvement. Specifically, if Z' is the objective value with the best split and Z is the current objective, then we continue running the algorithm if $Z' \geq (1+\gamma)Z$, where $\gamma \geq 0$ is a user-specified tolerance on the relative improvement; otherwise, if the relative improvement is lower than γ, the algorithm is terminated. Lower values of γ correspond to trees that are deeper with a larger number of nodes.

Our construction algorithm is defined formally as Algorithm 1. The tree policy is initialized to the degenerate policy described above, which prescribes the action **go** at all states. Each iteration of the loop first computes the objective attained from choosing the best split point at each leaf ℓ with each variable v assuming that either the left child leaf will be a **stop** action or the right child leaf will be a **stop** action. In the former case, we call the subtree rooted at node ℓ a left-stop subtree, and in the latter case, we call it a right-stop subtree; Figure 3 shows both of these subtrees. Then, if the best such split achieves an objective greater than the current objective value, we grow the tree in accordance to that split: we add two child nodes to the leaf ℓ^* using the GrowTree function, which is defined formally as Algorithm 2; we set the actions of those leaves according to D^*; we set the split variable index of the new split as v^*; and finally, we set the split point as $\theta^*_{\ell^*,v^*,D^*}$. The **existsImprovement** flag is used to terminate the algorithm when it is no longer possible to improve on the objective of the current tree by at least a factor of γ.

Algorithm 1 Tree construction algorithm.

Require: User-specified parameter γ (relative improvement tolerance)
Initialization:
$\mathcal{T} \leftarrow (\{1\}, \{1\}, \emptyset, \textbf{leftchild}, \textbf{rightchild})$, $\mathbf{v} \leftarrow \emptyset$, $\boldsymbol{\theta} \leftarrow \emptyset$, $a(1) \leftarrow \textbf{go}$
$Z \leftarrow 0$
existsImprovement $\leftarrow$ **true**

while **existsImprovement** **do**
for $\ell \in$ **leaves**, $v \in [n]$, $D \in \{\text{left}, \text{right}\}$ **do**
$Z^*_{\ell,v,D}, \theta^*_{\ell,v,D} \leftarrow \textsc{OptimizeSplitPoint}(\ell, v, D; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$
end for

existsImprovement $\leftarrow \mathbb{I}\left[\max_{\ell,v,D} Z^*_{\ell,v,D} \geq (1+\gamma)Z\right]$

if $\max_{\ell,v,D} Z^*_{\ell,v,D} > Z$ **then**
$(\ell^*, v^*, D^*) \leftarrow \arg\max_{\ell,v,D} Z^*_{\ell,v,D}$
$\textsc{GrowTree}(\mathcal{T}, \ell^*)$
$v(\ell^*) \leftarrow v^*$
$\theta(\ell^*) \leftarrow \theta^*_{\ell^*,v^*,D^*}$
if $D^* = \text{left}$ **then**
$a(\textbf{leftchild}(\ell^*)) \leftarrow \textbf{stop}$, $a(\textbf{rightchild}(\ell^*)) \leftarrow \textbf{go}$
else
$a(\textbf{leftchild}(\ell^*)) \leftarrow \textbf{go}$, $a(\textbf{rightchild}(\ell^*)) \leftarrow \textbf{stop}$
end if
$Z \leftarrow Z^*_{\ell^*,v^*,D^*}$
end if
end while
return Tree policy $(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$.

We comment on three important aspects of Algorithm 1. The first aspect is the determination of the optimal split point for a given leaf, a given split variable index and the direction of the stop action (left/right). At present, this optimization is encapsulated in the function OptimizeSplitPoint to aid in the exposition of the overall algorithm; we defer the description of this procedure to the next section (Section 4.2).

The second aspect is the assumption of how the child actions are set when we split a leaf. At present, we consider two possibilities: the left child is **stop** and the right child is **go** (this corresponds to $D^* = \text{left}$) or the left child is **go** and the right child is **stop** (this corresponds to $D^* = \text{right}$).

(a) Left-stop subtree. (b) Right-stop subtree.

Figure 3 **Visualization of the two different types of subtrees that can be used to split a leaf in the construction algorithm.**

Algorithm 2 GrowTree function.

Require: Tree topology $\mathcal{T} = (\mathcal{N}, \textbf{leaves}, \textbf{splits}, \textbf{leftchild}, \textbf{rightchild})$; target leaf ℓ.
$\mathcal{N} \leftarrow \mathcal{N} \cup \{|\mathcal{N}|+1, |\mathcal{N}|+2\}$
$\textbf{leaves} \leftarrow (\textbf{leaves} \setminus \{\ell\}) \cup \{|\mathcal{N}|+1, |\mathcal{N}|+2\}$
$\textbf{splits} \leftarrow \textbf{splits} \cup \{\ell\}$
$\textbf{leftchild}(\ell) = |\mathcal{N}|+1$
$\textbf{rightchild}(\ell) = |\mathcal{N}|+2$

These are not the only two possibilities, as we can also consider setting both child nodes to **go** or to **stop**. We do not explicitly consider these. First, one of these will result in exactly the same behavior as the current tree (for example, if $a(\ell^*) = \textbf{go}$ in the current tree and we set $a(\textbf{leftchild}(\ell^*)) = a(\textbf{rightchild}(\ell^*)) = \textbf{go}$ in the new tree, the new tree policy will prescribe the same actions as the current tree), and thus cannot result in an improvement. Second, OptimizeSplitPoint can potentially return a value of $-\infty$ or $+\infty$, effectively resulting in a degenerate split where we always go to the left or to the right. It is straightforward to see that such a split is equivalent to setting both child actions to **go** or **stop**.

The third aspect is complexity control. The algorithm in its current form stops building the tree when it is no longer possible to improve the objective value by a factor of at least γ. The parameter γ plays an important role in controlling the complexity of the tree: if γ is set to a very low value, the algorithm may produce deep trees with a large number of nodes, which is undesirable because (1) the resulting tree may lack interpretability and (2) the resulting tree may not generalize well to new data. In contrast, higher values of γ will cause the algorithm to terminate earlier with smaller trees whose training set performance will be closer to their out-of-sample performance; however, these trees may not be sufficiently complex to lead to good performance. The appropriate value of γ can be calibrated through standard techniques like k-fold cross-validation.

4.2. Finding the optimal split point

The key part of Algorithm 1 is the OptimizeSplitPoint function, which aims to answer the following question: what split point θ should we choose for the split at node ℓ on variable v so as to maximize the sample-based reward? At first glance, this question appears challenging because we could choose any real number to be θ, and it is not clear how we can easily optimize over θ. Fortunately, it will turn out that the sample average reward of the new tree will be a piecewise-constant function of θ, which we will be able to optimize over easily. The piecewise-constant function is obtained by taking a weighted combination of a collection of trajectory-specific piecewise-constant functions. These trajectory-specific piecewise-constant functions can be computed by carefully analyzing how the variable v changes along each trajectory.

Assume that the split variable index v, the leaf ℓ and the subtree direction D are fixed. Our first step is to determine the behavior of the trajectories with respect to the leaf ℓ. For each trajectory,

we first find the time at which each trajectory is stopped at a leaf different from ℓ. We call this the *no-stop time*. We denote it with $\tau_{-\ell,\omega}$ and define it as

$$\tau_{-\ell,\omega} = \min\{t \in [T] \mid \ell(\mathbf{x}(\omega,t)) \neq \ell \text{ and } a(\ell(\mathbf{x}(\omega,t)) = \textbf{stop}\}, \tag{13}$$

where we use the notation $[N]$ to denote the set $\{1,\ldots,N\}$ for an integer N, and we define the minimum to be $+\infty$ if the set is empty. We can effectively think of this as the time at which trajectory ω will stop if the new split that we place at ℓ does not result in the trajectory being stopped or equivalently, if we were to set the action of leaf ℓ to **go**. We define the *no-stop value*, $f_{\omega,\text{ns}}$, as the reward we garner if we allow the trajectory to stop at the no-stop time:

$$f_{\omega,\text{ns}} = \begin{cases} g(\omega,\tau_{-\ell,\omega}) & \text{if } \tau_{-\ell,\omega} < +\infty, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Note that the value is zero if the trajectory is never stopped outside of leaf ℓ. Lastly, we also determine the set of periods S_ω when the policy maps the system state to the leaf ℓ. We call these the *in-leaf periods*, and define the set as

$$S_\omega = \{t \in [T] \mid \ell(\mathbf{x}(\omega,t)) = \ell \text{ and } t < \tau_{-\ell,\omega}\}. \tag{15}$$

The second step is to determine the *permissible stop periods*. Depending on whether we are optimizing for the left-stop subtree or for the right-stop subtree, these are the periods at which the subtree could stop. The right-stop permissible stop periods are defined as

$$P_\omega = \{t \in S_\omega \mid x_v(\omega,t) > \max\{x_v(\omega,t') \,|\, t' \in S_\omega \text{ and } t' < t\}\}, \tag{16}$$

where the maximum is defined as $-\infty$ if the corresponding set is empty. The left-stop permissible stop periods are similarly defined as

$$P_\omega = \{t \in S_\omega \mid x_v(\omega,t) < \min\{x_v(\omega,t') \,|\, t' \in S_\omega \text{ and } t' < t\}\}, \tag{17}$$

where the minimum is defined as $+\infty$ if the corresponding set is empty.

The third step is to construct the trajectory-specific piecewise-constant functions. Each such piecewise constant function will tell us how the reward of a trajectory ω varies as a function of θ. Let us order the times in P_ω as $P_\omega = \{t_{\omega,1}, t_{\omega,2}, \ldots, t_{\omega,|P_\omega|}\}$, where $t_{\omega,1} < t_{\omega,2} < \cdots < t_{\omega,|P_\omega|}$. We use those times to define the corresponding breakpoints of our piecewise constant function; the ith breakpoint, $b_{\omega,i}$, is defined as

$$b_{\omega,i} = x_v(\omega, t_{\omega,i}).$$

The function value of the ith piece is given by $f_{\omega,i}$, which is defined as the value if we stopped at the ith permissible stop period:

$$f_{\omega,i} = g(\omega, t_{\omega,i}).$$

For a right-stop subtree, the corresponding piecewise constant function is denoted by F_ω, and it is defined as follows:

$$F_\omega(\theta) = \begin{cases} f_{\omega,1} & \text{if } \theta < b_{\omega,1}, \\ f_{\omega,2} & \text{if } b_{\omega,1} \leq \theta < b_{\omega,2}, \\ f_{\omega,3} & \text{if } b_{\omega,2} \leq \theta < b_{\omega,3}, \\ \vdots & \vdots \\ f_{\omega,|P_\omega|} & \text{if } b_{\omega,|P_\omega|-1} \leq \theta < b_{\omega,|P_\omega|}, \\ f_{\omega,\text{ns}} & \text{if } b_{\omega,|P_\omega|-1} \geq \theta. \end{cases} \tag{18}$$

For a left-stop subtree, the piecewise constant function is defined similarly, as

$$F_\omega(\theta) = \begin{cases} f_{\omega,1} & \text{if } \theta \geq b_{\omega,1}, \\ f_{\omega,2} & \text{if } b_{\omega,2} \leq \theta < b_{\omega,1}, \\ f_{\omega,3} & \text{if } b_{\omega,3} \leq \theta < b_{\omega,2}, \\ \vdots & \vdots \\ f_{\omega,|P_\omega|} & \text{if } b_{\omega,|P_\omega|} \leq \theta < b_{\omega,|P_\omega|-1}, \\ f_{\omega,\text{ns}} & \text{if } \theta < b_{\omega,|P_\omega|}. \end{cases} \tag{19}$$

The function value $F_\omega(\theta)$ is exactly the reward that will be garnered from trajectory ω if we set the split point to θ. By averaging these functions over ω, we obtain the function F, which returns the average reward, over the whole training set of trajectories, that ensues from setting the split point to θ:

$$F(\theta) = (1/\Omega) \sum_{\omega=1}^{\Omega} F_\omega(\theta). \tag{20}$$

We wish to find the value of θ that maximizes the function F, i.e.,

$$\theta^* \in \arg\max_{\theta \in \mathbb{R}} F(\theta).$$

Note that because each $F_\omega(\cdot)$ is a piecewise constant function of θ, the overall average $F(\cdot)$ will also be a piecewise constant function of θ, and as a result there is no unique maximizer of $F(\cdot)$. This is analogous to the situation faced in classification, where there is usually an interval of split points that lie between the values of the given coordinate of two points. Let $I = \arg\max_{\theta \in \mathbb{R}} F(\theta)$ be the interval on which $F(\cdot)$ is maximized. For convenience, consider the interior $\text{int}(I)$ of this interval, which will always be an interval of the form $(-\infty, b)$, (b, b') or (b, ∞) for some values $b, b' \in \mathbb{R}$, with $b < b'$. We can then set the split point as follows:

$$\theta^* = \begin{cases} -\infty & \text{if } \text{int}(I) = (-\infty, b) \text{ for some } b \in \mathbb{R}, \\ (b+b')/2 & \text{if } \text{int}(I) = (b, b') \text{ for some } b, b' \in \mathbb{R},\ b < b', \\ +\infty & \text{if } \text{int}(I) = (b, +\infty) \text{ for some } b \in \mathbb{R}. \end{cases} \tag{21}$$

In the case that I is a bounded interval, we set the split point to the midpoint of the interval. If I is an unbounded interval, then there is no well-defined midpoint of the interval, and we set the split point to be either $+\infty$ or $-\infty$.

We summarize the procedure formally as Algorithm 3, and illustrate it with an example.

Example 2. In this example, let us suppose that we are at the first iteration of the construction algorithm, where the tree policy is simply a single leaf with the action **go**. We want to find the optimal split point with respect to variable 1 for a split on this leaf, assuming that the subtree we place will be a right-stop subtree.

To illustrate, we fix a single trajectory ω. Figure 4 shows the relevant data for this trajectory, which are the values $x_1(\omega, t)$ and the rewards $g(\omega, t)$. We assume that $T = 18$ for all trajectories.

Since we are at the first iteration, we obtain that for the root node ℓ, the no-stop time is $\tau_{-\ell,\omega} = +\infty$, because there are no other leaves in the tree, and thus we have that $S_\omega = \{1, \ldots, 18\}$, i.e., every period is a valid in-leaf period for the root node. Observe also that the no-stop value $f_{\omega,\text{ns}} = 0$, because there is no other leaf in which the trajectory is stopped.

Having determined the in-leaf periods, we determine the set of permissible stop periods P_ω. To do this, since we are placing a right-stop subtree, we follow the computation in equation (16). The left-hand side of Figure 5 shows the same data as in Figure 4, but with the permissible stop periods indicated with red squares.

To intuitively understand the computation in equation (16), one can imagine a horizontal line, superimposed on the top plot of Figure 4, that starts at $-\infty$ and that slowly moves up towards $+\infty$.

Algorithm 3 OptimizeSplitPoint function.

Require: Coordinate v, leaf ℓ, direction of subtree D, current tree policy $(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$.
  for $\omega \in [\Omega]$ **do**
    Compute no-stop time $\tau_{-\ell,\omega}$ using equation (13).
    Compute no-stop value $f_{\omega,\text{ns}}$ using equation (14).
    Compute in-leaf periods S_ω using equation (15).
    if $D = \text{left}$ **then**
      Compute the permissible stop periods P_ω for left-stop subtree using equation (17).
      Order P_ω as $P_\omega = \{t_{\omega,1}, t_{\omega,2}, \ldots, t_{\omega,|P_\omega|}\}$, where $t_{\omega,1} < t_{\omega,2} < \cdots < t_{\omega,|P_\omega|}$.
      Compute the trajectory function $F_\omega(\cdot)$ using equation (19).
    else
      Compute the permissible stop periods P_ω for right-stop subtree using equation (16).
      Order P_ω as $P_\omega = \{t_{\omega,1}, t_{\omega,2}, \ldots, t_{\omega,|P_\omega|}\}$, where $t_{\omega,1} < t_{\omega,2} < \cdots < t_{\omega,|P_\omega|}$.
      Compute the trajectory function $F_\omega(\cdot)$ using equation (19).
    end if
  end for
  Compute the sample average function $F(\cdot)$ using equation (20).
  Compute the maximizer set $I \leftarrow \arg\max_{\theta \in \mathbb{R}} F(\theta)$.
  Compute θ^* using equation (21).
  return θ^*.

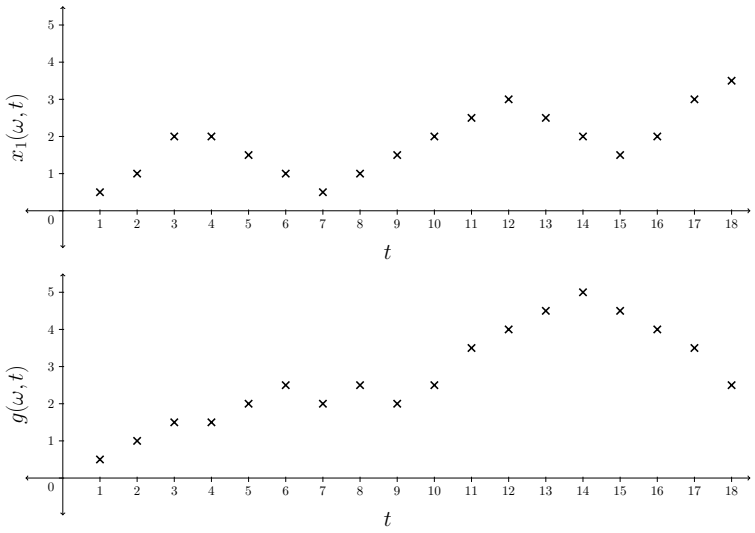


Figure 4 **Data for Example 2. The top plot shows coordinate 1 (x_1) of a single trajectory ω, and the bottom plot shows the corresponding reward (g).**

The vertical height of this horizontal line is the split point θ. As we move this line, we track the first period in time at which the trajectory exceeds this horizontal line: this is where the trajectory would stop, if we fixed the split point to the height of that line. Notice that when we start, the first period is $t = 1$. As soon as we exceed $x_1(\omega, 1) = 0.5$, we will no longer stop at $t = 1$, but at $t = 2$. As soon as our line goes above $x_1(\omega, 2)$, the new period at which we stop will be $t = 3$. Once our line goes above $x_1(\omega, 3)$, we will stop at $t = 11$, as this is the earliest that the trajectory passes above our line. We keep going in this way, until the line exceeds the value $\max_t x_1(\omega, t)$ and we have determined all of the permissible stop periods.

To further elaborate on this process, notice that $t = 1, 2, 3$ are permissible stop periods, but period $t = 5$ is not. In order to stop at $t = 5$, we would have to set the threshold θ to a value lower

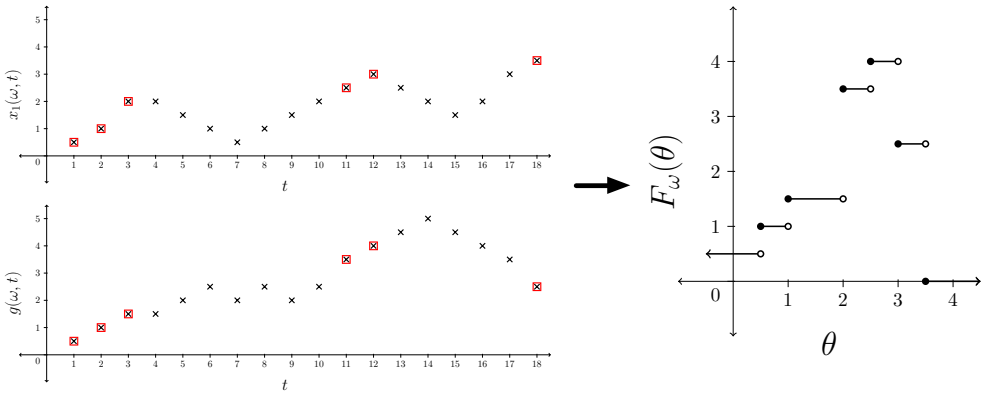


Figure 5 **Process for creating the function F_ω. The left-hand side shows $x_1(\omega,t)$ and $g(\omega,t)$ with the permissible stop periods indicated by red squares, while the right-hand side shows the function $F_\omega(\theta)$ that corresponds to this trajectory.**

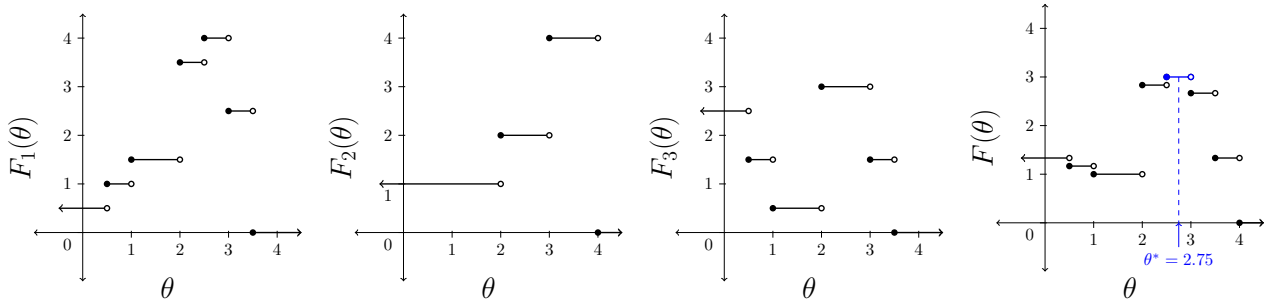


Figure 6 **Process for creating the function F from F_1, F_2, F_3. The left three graphs show the trajectory-specific functions F_1, F_2, F_3 while the right-most graph shows the overall sample average function $F(\theta)$, obtained by averaging F_1, F_2, F_3. On the plot of F, the blue piece of the function is the one on which F is maximized; the midpoint of that interval is $\theta^* = 2.75$, which is used as the final split point.**

than $x_1(\omega,5) = 1.5$, and $t = 5$ would have to be the first period at which the trajectory exceeds θ, i.e., $x_1(\omega,5) > \theta$. This is impossible with the trajectory of Figure 4. For this reason, the values of x_1 and g at $t = 5$ are not relevant in determining the reward garnered from the trajectory as θ is varied.

The permissible stop periods $P_\omega = \{t_1, t_2, \ldots, t_{|P_\omega|}\}$ allow us to define the breakpoints $b_{\omega,1}, \ldots, b_{\omega,|P_\omega|}$ and the values $f_{\omega,1}, \ldots, f_{\omega,|P_\omega|}$ and $f_{\omega,\text{ns}}$ of our piecewise constant function. The corresponding piecewise constant function is shown on the right-hand side of Figure 5. We then repeat this construction for all of the trajectories in our training set, and average them to obtain the overall function $F(\cdot)$. Figure 6 visualizes this averaging process, assuming that there are three trajectories in total (i.e., $\Omega = 3$), including the one trajectory displayed in Figures 4 and 5 (which we denote by $\omega = 1$).

After f_1, f_2, f_3 are averaged, we determine $\arg\max_{\theta\in\mathbb{R}} F(\theta)$ to be $[2.5, 3)$. Since this is a bounded interval, we take the midpoint of this interval, which is 2.75, to be our final split point θ^*.

4.3. Comparison to top-down tree induction for classification

We note that our algorithm is inspired by classical top-down classification tree induction algorithms such as CART (Breiman et al. 1984), ID3 (Quinlan 1986) and C4.5 (Quinlan 1993). However, there are a number of important differences, in both the problem and the algorithm. We have already discussed in Section 3.3 that the problem of finding a stopping policy in the form of a tree is structurally different than finding a classifier in the form of a tree – in particular, deciding leaf

labels in a tree classifier is easy to do, whereas deciding leaf actions in a tree policy for a stopping problem is an NP-Hard problem.

With regard to the algorithms themselves, there are several important differences. First, in our algorithm, we directly optimize the in-sample reward as we construct the tree. This is in contrast to how classification trees are built for classification, where typically it is not the classification error that is directly minimized, but rather an impurity metric such as the Gini impurity (as in CART) or the information gain/entropy (as in ID3 and C4.5). Second, in the classification tree setting, each leaf can be treated independently; once we determine that we should no longer split a leaf (e.g., because there is no more improvement in the impurity, or the number of points in the leaf is too low), we never need to consider that leaf again. In our algorithm, we must consider every leaf in each iteration, even if that leaf may not have resulted in improvement in the previous iteration; this is because the actions we take in the leaves interact with each other and are not independent of each other. Lastly, as mentioned earlier, determining the optimal split point is much more involved than in the classification tree setting: in classification, the main step is to sort the observations in a leaf by the split variable. In our optimal stopping setting, determining the split point is a more involved calculation that takes into account when each trajectory is in a given leaf and how the cumulative maximum/minimum of the values of a given split variable change with the time t.

5. Application to option pricing

In this section, we report on the performance of our method in an application drawn from option pricing. We define the option pricing problem in Section 5.1. We compare our policies against two benchmarks in terms of out-of-sample performance and computation time in Sections 5.2 and 5.3, respectively. In Section 5.4, we present our tree policies for this application and discuss their structure. Finally, in Section 5.5, we evaluate our policies on instances calibrated with real S&P500 stock price data. Additional numerical results are provided in Section EC.2.

5.1. Problem definition

High-dimensional option pricing is one of the classical applications of optimal stopping, and there is a wide body of literature devoted to developing good heuristic policies (see Glasserman 2013 for a comprehensive survey). In this section, we illustrate our tree-based method on a standard family of option pricing problems from Desai et al. (2012b). We consider two benchmarks. Our first benchmark is the least-squares Monte Carlo method from Longstaff and Schwartz (2001), which is a commonly used method for option pricing. Our second benchmark is the pathwise optimization (PO) method from Desai et al. (2012b), which was shown in that paper to yield stronger exercise policies than the method of Longstaff and Schwartz. Both of these methods involve building regression models that estimate the continuation value at each time t using a collection of basis functions of the underlying state, and using them within a greedy policy.

In each problem, the option is a Bermudan max-call option, written on n underlying assets. The stopping problem is defined over a period of 3 calendar years with $T = 54$ equally spaced exercise opportunities. The price paths of the n assets are generated as a geometric Brownian motion with drift equal to the annualized risk-free rate $r = 5\%$ and annualized volatility $\sigma = 20\%$, starting at an initial price $\bar{p}$. The pairwise correlation ρ_{ij} between different assets $i \neq j$ is set to $\bar{\rho} = 0$. (The strike price for each option is set at $K = 100$. Each option has a knock-out barrier $B = 170$, meaning that if any of the underlying stock prices exceeds B at some time t_0, the option is "knocked out" and the option value becomes 0 at all times $t \geq t_0$. Time is discounted continuously at the risk-free rate, which implies a discrete discount factor of $\beta = \exp(-0.05 \times 3/54) = 0.99723$.

Our state variable is defined as $\mathbf{x}(t) = (t, p_1(t), \ldots, p_n(t), y(t), g(t))$, where t is the index of the period; $p_j(t)$ is the price of asset j at exercise time $t \in \{1, \ldots, T\}$; $y(t)$ is a binary variable that is 0 or 1 to indicate whether the option has not been knocked out by time t, defined as

$$y(t) = \mathbb{I}\left\{\max_{1 \leq j \leq n, 1 \leq t' \leq t} p_j(t) < B\right\}; \tag{22}$$

and $g(t)$ is the payoff at time t, defined as

$$g(t) = \max\left\{0, \max_{1\leq j\leq n} p_j(t) - K\right\} \cdot y(t). \tag{23}$$

In our implementation of the tree optimization algorithm, we vary the subset of the state variables that the tree model is allowed to use. We use TIME to denote t, PRICES to denote $p_1(t), \ldots, p_n(t)$, PAYOFF to denote $g(t)$, and KOIND to denote $y(t)$. We set the relative improvement parameter γ to 0.005, requiring that we terminate the construction algorithm when the improvement in in-sample objective becomes lower than 0.5%. We report on the sensitivity of our algorithm to the parameter γ in Section EC.2.1.

In our implementation of the Longstaff-Schwartz (LS) algorithm, we vary the basis functions that are used in the regression. We follow the same notation as for the tree optimization algorithm in denoting different subsets of state variables; we additionally define the following sets of basis functions:

- ONE: the constant function, 1.
- PRICESKO: the knock-out (KO) adjusted prices defined as $p_i(t) \cdot y(t)$ for $1 \leq i \leq n$.
- MAXPRICEKO and MAX2PRICEKO: the largest and second largest KO adjusted prices.
- PRICES2KO: the knock-out adjusted second-order price terms, defined as $p_i(t) \cdot p_j(t) \cdot y(t)$ for $1 \leq i \leq j \leq n$.

In our implementation of the PO algorithm, we also vary the basis functions, and follow the same notation as for LS. We use 500 inner samples, as in Desai et al. (2012b).

We vary the number of stocks as $n = 4, 8, 16$ and the initial price of all assets as $\bar{p} = 90, 100, 110$. For each combination of n and $\bar{p}$, we consider ten replications. In each replication, we generate 20,000 trajectories for training the methods and 100,000 trajectories for out-of-sample testing. All replications were executed on the Amazon Elastic Compute Cloud (EC2) using a single instance of type `r4.4xlarge` (Intel Xeon E5-2686 v4 processor with 16 virtual CPUs and 122 GB memory). All methods were implemented in the Julia technical computing language, version 0.6.2 (Bezanson et al. 2017). All linear optimization problems for the pathwise optimization method were formulated using the JuMP package for Julia (Lubin and Dunning 2015, Dunning et al. 2017) and solved using Gurobi 8.0 (Gurobi Optimization, Inc. 2018).

5.2. Out-of-sample performance

Table 1 shows the out-of-sample reward garnered by the LS and PO policies for different basis function architectures and the tree policies for different subsets of the state variables, for different values of the initial price $\bar{p}$. The rewards are averaged over the ten replications, with standard errors reported in parentheses. For ease of exposition, we focus only on the $n = 8$ assets, as the results for $n = 4$ and $n = 16$ are qualitatively similar; for completeness, these results are provided in Section EC.2.2. In addition, Section EC.2.3 provides additional results for when the common correlation $\bar{\rho}$ is not equal to zero.

From this table, it is important to recognize three key insights. First, for all three values of $\bar{p}$, the best tree policies – specifically, those tree policies that use the TIME and PAYOFF state variables – are able to outperform all of the LS and PO policies. Relative to the best LS policy for each $\bar{p}$, these improvements range from 2.04% ($\bar{p} = 110$) to 3.02% ($\bar{p} = 90$), which is substantial given the context of this problem. Relative to the best PO policy for each $\bar{p}$, the improvements range from 1.12% ($\bar{p} = 100$) to 1.66% ($\bar{p} = 90$), which is still a remarkable improvement.

Second, observe that this improvement is attained despite an experimental setup biased in favor of LS and PO. In this experiment, the tree optimization algorithm was only allowed to construct policies using subsets of the primitive state variables. In contrast, both the LS and the PO method were tested with a richer set of basis function architectures that included knock-out adjusted prices, highest and second-highest prices and second-order price terms. From this perspective, it is significant that our tree policies could outperform the best LS policy and the best PO policy.

Table 1 Comparison of out-of-sample performance between LSM, PO and tree policies for $n = 8$ assets, for different initial prices $\bar{p}$. In each column, the best performance is indicated in bold.

n	Method	State variables / Basis functions	Initial Price $\bar{p} = 90$	$\bar{p} = 100$	$\bar{p} = 110$
8	LS	ONE	33.82 (0.021)	38.70 (0.023)	43.13 (0.015)
8	LS	PRICES	33.88 (0.019)	38.59 (0.023)	43.03 (0.014)
8	LS	PRICESKO	41.45 (0.027)	49.33 (0.017)	53.08 (0.009)
8	LS	PRICESKO, KOIND	41.86 (0.021)	49.36 (0.020)	53.43 (0.012)
8	LS	PRICESKO, KOIND, PAYOFF	43.79 (0.022)	49.86 (0.013)	53.07 (0.009)
8	LS	PRICESKO, KOIND, PAYOFF, MAXPRICEKO	43.83 (0.021)	49.86 (0.013)	53.07 (0.009)
8	LS	PRICESKO, KOIND, PAYOFF, MAXPRICEKO, MAX2PRICEKO	43.85 (0.022)	49.87 (0.012)	53.06 (0.008)
8	LS	PRICESKO, PAYOFF	44.06 (0.013)	49.61 (0.010)	52.65 (0.008)
8	LS	PRICESKO, PRICES2KO, KOIND, PAYOFF	44.07 (0.013)	49.93 (0.010)	53.11 (0.010)
8	PO	PRICES	40.94 (0.012)	44.84 (0.016)	47.48 (0.014)
8	PO	PRICESKO, KOIND, PAYOFF	44.01 (0.019)	50.71 (0.011)	53.82 (0.009)
8	PO	PRICESKO, KOIND, PAYOFF, MAXPRICEKO, MAX2PRICEKO	44.07 (0.017)	50.67 (0.011)	53.81 (0.011)
8	PO	PRICESKO, PRICES2KO, KOIND, PAYOFF	44.66 (0.018)	50.67 (0.010)	53.77 (0.008)
8	Tree	PAYOFF, TIME	**45.40 (0.018)**	**51.28 (0.016)**	**54.52 (0.006)**
8	Tree	PRICES	35.86 (0.170)	43.42 (0.118)	46.95 (0.112)
8	Tree	PRICES, PAYOFF	39.13 (0.018)	48.37 (0.014)	53.61 (0.010)
8	Tree	PRICES, TIME	38.21 (0.262)	40.21 (0.469)	42.69 (0.133)
8	Tree	PRICES, TIME, PAYOFF	**45.40 (0.017)**	**51.28 (0.016)**	54.51 (0.006)
8	Tree	PRICES, TIME, PAYOFF, KOIND	**45.40 (0.017)**	**51.28 (0.016)**	54.51 (0.006)

This also highlights an advantage of our tree optimization algorithm, which is its nonparametric nature: if the boundary between **stop** and **go** in the optimal policy is highly nonlinear with respect to the state variables, then by estimating a tree policy one should (in theory) be able to closely approximate this structure with enough splits. In contrast, the performance of LS and PO is highly dependent on the basis functions used, and requires the DM to specify a basis function architecture.

Third, with regard to our tree policies specifically, we observe that policies that use TIME and PAYOFF perform the best. The TIME state variable is critical to the success of the tree policies because the time horizon is finite: as such, a good policy should behave differently near the end of the horizon from how it behaves at the start of the time horizon. The LS algorithm handles this automatically because it regresses the continuation value from $t = T - 1$ to $t = 1$, so the resulting policy is naturally time-dependent. The PO policy is obtained in a similar way, with the difference that one regresses an upper bound from $t = T - 1$ to $t = 1$, so the PO policy is also time-dependent. Without time as an explicit state variable, our tree policies will estimate stationary policies, which are unlikely to do well given the nature of the problem. Still, it is interesting to observe some instances in our results where our time-*in*dependent policies outperform time-dependent ones from LS (for example, compare tree policies with PRICES only to LS with PRICES or ONE). With regard to PAYOFF, we note that because the payoff $g(t)$ is a function of the prices $p_1(t), \ldots, p_n(t)$, one should in theory be able to replicate splits on PAYOFF using a collection of splits on PRICE; including PAYOFF explicitly helps the construction algorithm recognize such collections of splits through a single split on the payoff variable.

Table 2 Comparison of estimation time between Longstaff-Schwartz, pathwise optimization and tree policies for $n = 8$ assets, for different initial prices $\bar{p}$ and common correlation $\bar{\rho} = 0$.

n	Method	State variables / Basis functions	Initial Price $\bar{p} = 90$	$\bar{p} = 100$	$\bar{p} = 110$
8	LS	ONE	1.2 (0.0)	1.2 (0.0)	1.2 (0.0)
8	LS	PRICES	1.4 (0.0)	1.4 (0.0)	1.4 (0.0)
8	LS	PRICESKO	1.4 (0.1)	1.5 (0.1)	1.5 (0.1)
8	LS	PRICESKO, KOIND	1.6 (0.1)	1.4 (0.1)	1.5 (0.1)
8	LS	PRICESKO, KOIND, PAYOFF	1.9 (0.2)	1.9 (0.1)	1.8 (0.1)
8	LS	PRICESKO, KOIND, PAYOFF, MAXPRICEKO	2.5 (0.2)	2.4 (0.2)	2.4 (0.2)
8	LS	PRICESKO, KOIND, PAYOFF, MAXPRICEKO, MAX2PRICEKO	2.7 (0.2)	2.6 (0.3)	2.4 (0.2)
8	LS	PRICESKO, PAYOFF	1.7 (0.2)	1.6 (0.1)	1.4 (0.1)
8	LS	PRICESKO, PRICES2KO, KOIND, PAYOFF	5.5 (0.4)	4.4 (0.2)	4.5 (0.2)
8	PO	PRICES	33.3 (0.7)	35.5 (0.7)	32.8 (0.7)
8	PO	PRICESKO, KOIND, PAYOFF	76.8 (2.8)	73.8 (5.0)	56.9 (2.1)
8	PO	PRICESKO, KOIND, PAYOFF, MAXPRICEKO, MAX2PRICEKO	104.7 (5.8)	79.6 (3.3)	66.8 (3.8)
8	PO	PRICESKO, PRICES2KO, KOIND, PAYOFF	221.3 (9.4)	180.7 (4.1)	142.0 (4.2)
8	Tree	PAYOFF, TIME	7.6 (0.3)	3.9 (0.3)	3.2 (0.1)
8	Tree	PRICES	124.7 (8.6)	125.5 (4.8)	125.0 (5.8)
8	Tree	PRICES, PAYOFF	5.5 (0.1)	5.3 (0.1)	5.1 (0.1)
8	Tree	PRICES, TIME	158.8 (12.5)	101.0 (12.8)	51.1 (2.7)
8	Tree	PRICES, TIME, PAYOFF	20.4 (1.0)	10.9 (0.6)	9.3 (0.1)
8	Tree	PRICES, TIME, PAYOFF, KOIND	21.5 (1.5)	11.2 (0.6)	9.3 (0.2)

5.3. Computation time

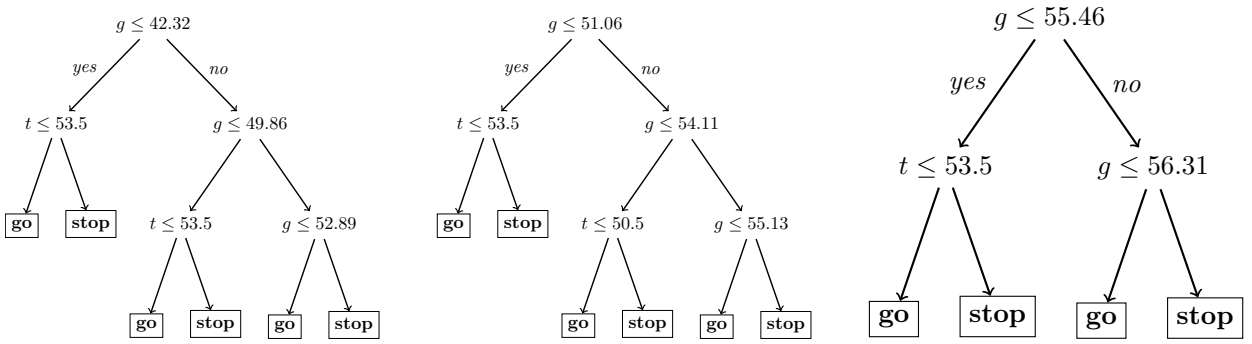
Table 2 reports the computation time for the LS, PO and tree policies for $n = 8$ and $\bar{p} \in \{90, 100, 110\}$, for the uncorrelated ($\bar{\rho} = 0$) case. The computation times are averaged over the ten replications for each combination of n and $\bar{p}$. As with the performance results, we focus on $n = 8$ to simplify the exposition; additional timing results for $n = 4$ and $n = 16$ are provided in Section EC.2.4. For LS, the computation time consists of only the time required to perform the regressions from $t = T - 1$ to $t = 1$. For PO, the computation time consists of the time required to formulate the linear optimization problem in JuMP, the solution time of this problem in Gurobi, and the time required to perform the regressions from $t = T - 1$ to $t = 1$ (as in Longstaff-Schwartz). For the tree method, the computation consists of the time required to run Algorithm 1.

From this table, we can see that although our method requires more computation time than LS, the times are in general quite modest: our method requires no more than 2.5 minutes on average in the largest case. (In experiments with $n = 16$, reported in Section EC.2.4, we find that the method requires no more than 5 minutes on average in the largest case.) The computation times of our method also compare quite favorably to the computation times for the PO method. We also remark here that our computation times for the PO method do *not* include the time required to generate the inner paths and to pre-process them in order to formulate the PO linear optimization problem. For $n = 8$, including this additional time increases the computation times by a large amount, ranging from 540 seconds (using PRICES only; approximately 9 minutes) to 2654 seconds (using PRICESKO, PRICES2KO, KOIND, PAYOFF; approximately 44 minutes).

5.4. Policy structure

It is also interesting to examine the structure of the policies that emerge from our tree optimization algorithm. Figure 7 shows trees obtained using PRICES, TIME, PAYOFF and KOIND for one replication with $\bar{p} = 90$, for $n = 4, 8, 16$. This figure presents a number of important qualitative insights

Figure 7 **Examples of tree policies for initial price $\bar{p} = 90$ with state variables prices, time, payoff and KOind for a single replication.**



(a) $n = 4$. (b) $n = 8$. (c) $n = 16$.

about our algorithm. First, observe that the trees are extremely simple: there are no more than seven splits in any of the trees. The policies themselves are easy to understand and sensible. Taking $n = 8$ as an example, we see that if the payoff is lower than 51.06, it does not stop unless we are in the last period ($t = 54$), because there is still a chance that the payoff will be higher by $t = 54$. If the payoff is greater than 51.06 but less than or equal to 54.11, then the policy does not stop unless we are in the last four periods ($t = 51$, 52, 53 or 54). If the payoff is greater than 54.11 but less than or equal to 55.13, then we continue. Otherwise, if the payoff is greater than 55.13, then we stop no matter what period we are in; this is likely because when such a payoff is observed, it is large enough and far enough in the horizon that it is unlikely a larger reward will be realized later. In general, as the payoff becomes larger, the policy will recommend stopping earlier in the horizon. Interestingly, the policies do not include any splits on the prices and the KO indicator, despite the construction algorithm being allowed to use these variables: this further underscores the ability of the construction algorithm to produce simple policies. It is also interesting to note that the tree structure is quite consistent across all three values of n. To the best of our knowledge, we do not know of any prior work suggesting that simple policies as in Figure 7 can perform well against mainstream ADP methods for high-dimensional option pricing.

We observe in the $n = 4$ and $n = 8$ trees that there is some redundancy in the splits. For example, for $n = 4$, observe that the left subtree of the split $g \leq 42.32$ is identical to the left subtree of the split $g \leq 49.86$; thus, the policy will take the same action whether $g \in [0, 42.32]$ or $g \in (42.32, 49.86]$, and the entire tree could be simplified by replacing it with the subtree rooted at the split $g \leq 49.86$. The reason for this redundancy is due to the greedy nature of the construction procedure. At the start of the construction procedure, splitting on $g \leq 42.32$ leads to the best improvement in the reward, but as this split and other splits are added, the split on $g \leq 49.86$ becomes more attractive.

In addition to this redundancy, we also note that in all three trees, there is an interval $(g_1, g_2]$ such that if $g \in (g_1, g_2]$, the policy will continue (for example, in $n = 8$, this interval is $(54.11, 55.13]$). This property of the policies is suboptimal because if g is inside that interval, the policy may choose to continue even if $t = 54$, leading to a reward of zero for that trajectory; thus, we could in theory improve the performance of the policy by requiring the policy to stop if $t = 54$ (i.e., adding a right-stop subtree with the split $t \leq 53.5$). This may occur for two reasons: first, due to the sample-based nature of the optimization algorithm, the number of training set trajectories that are inside the interval $(g_1, g_2]$ at $t = 54$ may be small enough that adding the split $t \leq 53.5$ will not improve the overall sample-based objective by a relative factor of more than $1 + \gamma = 1.005$. The second reason is that, even if the sample is sufficiently large, the split may still fail to offer a sufficient improvement to the objective; this could be the case if the policy is such that there is a very small probability that a trajectory makes it to $t = 54$ and that $g(t)$ is in the interval $(g_1, g_2]$ at $t = 54$, so that taking the optimal action has a negligible impact on the objective.

Finally, it is worth qualitatively comparing the policies that arise from our algorithm to policies derived from LS or the PO method. The trees in Figure 7 fully specify the policy: the decision to stop or go can be made by checking at most three logical conditions. Moreover, the trees in Figure 7 directly tell us when the policy will stop: either when the reward is very high or when we are in the final period of the horizon. In contrast, a policy derived from the LS or the PO method will consist of regression coefficients for all basis functions for each t in the time horizon; we would not consider this policy interpretable due to the high dimensionality of its specification, as well as the fact that it does not drop, or clearly delineate, which basis functions are non-informative to the policy. From such a specification, it is difficult to immediately understand what the behavior of the policy will be (i.e., at what times and in what part of the state space the policy will stop), and even more so when the basis functions are more complex (such as the second-order price terms). We provide an example of a LS policy for $n = 8$ in Section EC.2.5 to further illustrate this.

5.5. Out-of-sample performance with S&P-500 data calibration

In the previous sections, we considered options with artificial parameters specifying the stock price dynamics. In this section, we consider another set of experiments where the stock price dynamics are calibrated using real data. We again consider a max-call option, where the initial price $\bar{p}$ is set to 100, the strike price K is set to 100 and the barrier price is set to 300. We assume that payoffs are discounted at a continuous (annualized) interest rate r, for which we test values of 0.02, 0.05 and 0.10. We assume that the stock expires after 2 years, with 24 equally spaced exercise opportunities per year, for a total of $T = 48$ exercise opportunities.

To calibrate the stock dynamics, we use daily prices of real stocks in the S&P500 in two periods: the year 2017, and the period 2007–2017. We randomly sample 10 sets of $n = 8$ stocks from each period, so as to ensure that price data is not missing for more than 10% of the days in the chosen period. Using this data, we compute the daily log returns, and use the `mvnmle` package in R (Gross 2018) to estimate the drift and covariance of the corresponding geometric Brownian motion model. For each of the ten sets of stocks in each of the two periods (2017 and 2007 – 2017) at each interest rate $r \in \{0.02, 0.05, 0.10\}$, we use the estimated parameters to simulate ten replications of 20,000 trajectories for training and 100,000 trajectories for out-of-sample testing. This gives rise to $10 \times 2 \times 3 = 60$ different option pricing instances, which are replicated ten times.

We again compare our tree optimization approach against LS and PO. We test the same basis function/state variable architectures as in Section 5.2. For simplicity, we report only the knock-out adjusted first-order (PRICESKO, PAYOFF, KOIND) and second-order (PRICESKO, PRICES2KO, PAYOFF, KOIND) architectures for LS and PO, as these gave the best performance for these two methods. For the tree policies, we report the performance of the largest state variable architecture (PRICES, PAYOFF, TIME, KOIND) as this architecture gave the best performance. (We remark that as in the synthetic dynamics experiments, all tree policies using PAYOFF and TIME as state variables gave very similar results.)

Table 3 compares the three methods on the ten instances for each of the two periods for a single interest rate ($r = 0.05$); the values shown are computed out-of-sample using the 100,000 test set trajectories, and averaged over ten replications. (Additional results for $r = 0.02$ and $r = 0.10$, as well as information on the stocks comprising the ten instances in each period, can be found in Section EC.2.6.) From this table, we observe the same qualitative behavior as in our experiments in Section 5.2, namely that our tree policies deliver performance that is competitive with and often significantly better performance than the LS and PO policies. The purpose of this experiment and the calibration procedure is to investigate how our method performs with more realistic and complex dynamics than those in Section 5.2. Based on these results, our method has the potential to deliver good performance for realistic dynamics as well as the simpler dynamics of Section 5.2.

Period	Instance	LS - 1st. O.	LS - 2nd O.	PO - 1st. O.	PO - 2nd O.	Tree
2007 – 2017	1	93.09 (0.057)	95.67 (0.064)	88.27 (0.040)	97.19 (0.051)	**98.70 (0.065)**
	2	95.26 (0.063)	97.82 (0.060)	90.31 (0.072)	99.33 (0.065)	**100.84 (0.066)**
	3	98.77 (0.089)	102.23 (0.049)	94.02 (0.116)	103.66 (0.031)	**105.43 (0.064)**
	4	81.92 (0.062)	84.37 (0.045)	77.56 (0.078)	85.66 (0.050)	**86.71 (0.073)**
	5	93.88 (0.084)	96.51 (0.066)	88.93 (0.070)	97.86 (0.049)	**98.39 (0.095)**
	6	95.96 (0.051)	98.88 (0.048)	89.50 (0.129)	100.53 (0.052)	**102.43 (0.054)**
	7	78.70 (0.088)	80.98 (0.075)	73.72 (0.095)	82.22 (0.075)	**83.27 (0.118)**
	8	96.21 (0.063)	98.33 (0.060)	92.33 (0.097)	99.59 (0.051)	**100.82 (0.067)**
	9	83.04 (0.052)	85.48 (0.054)	76.95 (0.068)	86.69 (0.041)	**87.11 (0.191)**
	10	86.10 (0.058)	88.42 (0.045)	80.77 (0.072)	89.69 (0.036)	**90.33 (0.123)**
2017	1	146.51 (0.065)	149.57 (0.039)	151.31 (0.035)	152.29 (0.034)	**155.83 (0.072)**
	2	138.95 (0.060)	142.10 (0.035)	142.62 (0.082)	144.54 (0.024)	**147.71 (0.232)**
	3	144.77 (0.051)	147.88 (0.029)	149.34 (0.043)	150.85 (0.025)	**154.32 (0.181)**
	4	147.52 (0.045)	152.81 (0.034)	151.53 (0.028)	153.92 (0.028)	**157.39 (0.036)**
	5	141.59 (0.047)	144.84 (0.044)	145.72 (0.038)	146.38 (0.042)	**149.53 (0.034)**
	6	130.12 (0.080)	134.57 (0.044)	130.29 (0.072)	136.25 (0.039)	**137.08 (0.245)**
	7	153.87 (0.049)	161.40 (0.030)	159.25 (0.038)	163.12 (0.030)	**169.74 (0.028)**
	8	151.12 (0.025)	154.76 (0.031)	155.09 (0.023)	156.64 (0.021)	**161.14 (0.050)**
	9	145.93 (0.057)	148.39 (0.038)	149.16 (0.044)	149.88 (0.034)	**151.64 (0.063)**
	10	140.46 (0.034)	143.56 (0.030)	144.45 (0.028)	145.77 (0.027)	**149.22 (0.194)**

Table 3 Comparison of LS, PO and tree policies using realistically calibrated max-call option instances, with interest rate $r = 0.05$. Values shown are averages over ten replications, with standard errors in parentheses; bold is used to indicate the best method for each instance in each period. The terms "1st. O." and "2nd. O." indicate the first-order basis function architecture (pricesKO, payoff, KOind) and the second-order basis function architecture (pricesKO, prices2KO, payoff, KOind).

6. Application to one-dimensional uniform problem

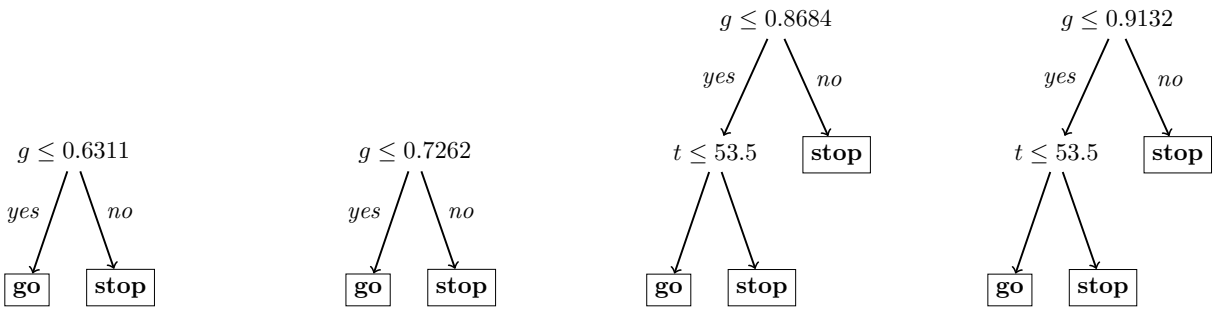
In this section, we consider the following simple optimal stopping problem: we have a one-dimensional stochastic process, $x(1), x(2), \ldots, x(T)$, where at each time $x(t)$ is independently drawn from a continuous Uniform$(0, 1)$ distribution. The payoff at each t is given by $g(t, x) = x$ and rewards are discounted by a discount factor of β. The rationale for considering this problem is its simplicity: it is small and simple enough that we can solve for the optimal policy directly, and obtain insight from comparing the performance of our tree policies to this optimum.

We compare both our tree policies and Longstaff-Schwartz against the optimal policy. We run the tree construction algorithm with PAYOFF and TIME as state variables, and we run LS with the constant basis function (1). For the construction algorithm we set $\gamma = 0.005$. Note that for LS, it does not make sense to use $x(t)$ as a predictor in the regression; this is because the state variable $x(t)$ is drawn independently at each time, and so the optimal continuation value does not vary with the current $x(t)$. For this reason, we only consider the basis function architecture consisting of 1. We use 20,000 trajectories to build each model, and 100,000 to perform out-of-sample evaluation. We test values of β in $\{0.9, 0.95, 0.97, 0.98, 0.99, 0.995, 0.999, 0.9999, 1.0\}$.

Figure 8 displays the tree policies for $\beta = 0.9, 0.95, 0.99, 1.0$. From this figure we can see that for $\beta = 0.9$ and $\beta = 0.95$, the tree policy does not depend on time: at any t, we simply check whether the payoff g is greater than or equal to some threshold value. For $\beta = 0.99$ and $\beta = 1.0$, we stop if either the payoff is greater than or equal to some threshold, or if we are in the last period. To compare these against the LS and optimal policies, we also plot in Figure 9 the effective thresholds used by the three policies from $t = 1$ to $t = 54$.

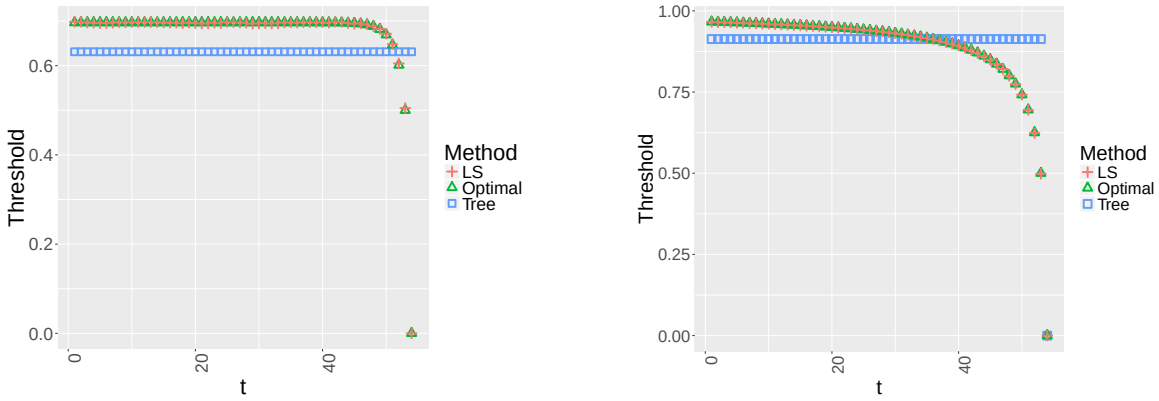
From these two figures, we can see that the optimal thresholds are time-dependent, as we would expect, and the thresholds used by LS are essentially the same as those used by the optimal policy. In contrast, the thresholds used by the tree policies are significantly less time-dependent (for $\beta = 0.9$, the threshold is constant, while for $\beta = 1.0$, the threshold only changes at $t = 54$, and is otherwise constant).

Given the large differences in the policy structure, one would expect that the tree policies would be highly suboptimal. Surprisingly, however, this turns out not to be the case. To compare the



(a) $\beta = 0.9$. (b) $\beta = 0.95$. (c) $\beta = 0.99$. (d) $\beta = 1.0$.

Figure 8 **Examples of tree policies for $\beta = 0.9, 0.95, 0.99, 1.0$.**



(a) $\beta = 0.9$. (b) $\beta = 1.0$.

Figure 9 **Comparison of thresholds used by the three different policies at $\beta = 0.9$ and $\beta = 1.0$.**

policies, we show the out-of-sample performance of the tree, LS and optimal policies in Table 4. At low discount factors ($\beta \leq 0.99$), both the tree and LS policies are essentially optimal. For $\beta > 0.99$, LS displays a slight edge over the tree policies, and is essentially optimal; in the largest case, there is a difference of about 0.013 between the tree policies and the LS/optimal policies.

β	Tree	LS	Optimal (sim.)	Optimal (true)
0.9	0.6962	0.6961	0.6961	0.6964
0.95	0.7622	0.7622	0.7622	0.7620
0.97	0.8043	0.8043	0.8043	0.8044
0.98	0.8342	0.8342	0.8342	0.8340
0.99	0.8762	0.8763	0.8763	0.8763
0.995	0.9078	0.9086	0.9086	0.9087
0.999	0.9427	0.9507	0.9507	0.9507
0.9999	0.9528	0.9647	0.9647	0.9648
1	0.9532	0.9665	0.9665	0.9666

Table 4 **Comparison of tree, LS and optimal policies for the 1D uniform problem. With the exception of "Optimal (true)" (the theoretical optimal reward for the problem), all values shown are out-of-sample rewards averaged over 5 independent replications. All standard errors are smaller than 0.0005.**

This experiment provides several insights. First, even for a simple problem such as this one, there may be near-optimal policies that are simpler than the optimal policy (for example, compare the constant threshold policy for $\beta = 0.9$ to the time-dependent threshold policy that is optimal) and that our construction algorithm can potentially discover such policies. Second, our approach is not a free lunch; indeed, for the higher discount rates, LS is able to recover the optimal policy, while our tree policies are suboptimal (albeit only by a small amount).

Lastly, we remark that in theory, the optimal policy for this problem could be represented as a tree policy, where one would have a sequence of splits on t, and each such split would be followed by a split on g. In this set of examples, our algorithm does not recover such a policy due to its greedy nature. One question in this direction is how to recognize splits that do not immediately yield an improvement in reward, but enable later splits that yield significant improvements. The answer to this question is not obvious, especially in light of the structure of the optimal stopping problem that is leveraged to efficiently optimize split points in our construction algorithm (Section 4), and is an interesting direction for future research.

7. Conclusion

In this paper, we consider the problem of designing interpretable policies for optimal stopping, based on binary trees. We formulate the problem as an SAA problem, which we show to be theoretically intractable. Thus motivated, we develop a heuristic algorithm that greedily constructs a policy from the top down, by exploiting the stopping problem's structure in finding the optimal split point at each leaf. In numerical experiments on a standard option pricing problem, our algorithm attains better performance than state-of-the-art ADP methods for option pricing, while simultaneously producing policies that are significantly simpler and more transparent. We believe that this represents an exciting starting point for future research at the intersection of stochastic control and interpretable machine learning.

References

Adelman, D., A. J. Mersereau. 2008. Relaxations of weakly coupled stochastic dynamic programs. *Operations Research* **56**(3) 712–727.

Andersen, L., M. Broadie. 2004. Primal-dual simulation algorithm for pricing multidimensional American options. *Management Science* **50**(9) 1222–1234.

Angelino, E., N. Larus-Stone, D. Alabi, M. Seltzer, C. Rudin. 2017. Learning certifiably optimal rule lists for categorical data. *arXiv preprint arXiv:1704.01701* .

Azizi, M. J., P. Vayanos, B. Wilder, E. Rice, M. Tambe. 2018. Designing fair, efficient, and interpretable policies for prioritizing homeless youth for housing resources. *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 35–51.

Bertsimas, D., V. F. Farias, N. Trichakis. 2013. Fairness, efficiency, and flexibility in organ allocation for kidney transplantation. *Operations Research* **61**(1) 73–87.

Bertsimas, D., V. V. Mišić. 2016. Decomposable Markov Decision Processes: A Fluid Optimization Approach. *Operations Research* **64**(6) 1537–1555.

Bezanson, J., A. Edelman, S. Karpinski, V. B. Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Review* **59**(1) 65–98.

Bravo, F., Y. Shaposhnik. 2017. Discovering optimal policies: A machine learning approach to model analysis. *Working paper* Available at SSRN: `https://ssrn.com/abstract=3069690`.

Breiman, L., J. Friedman, C. J. Stone, R. A. Olshen. 1984. *Classification and regression trees*. CRC press.

Broadie, M., P. Glasserman. 1997. Pricing american-style securities using simulation. *Journal of Economic Dynamics and Control* **21**(8-9) 1323–1352.

Brown, D. B., J. E. Smith, P. Sun. 2010. Information relaxations and duality in stochastic dynamic programs. *Operations research* **58**(4-part-1) 785–801.

Carriere, J. F. 1996. Valuation of the early-exercise price for derivative securities using simulations and splines. *Insurance: Mathematics and Economics* **19**(1) 19–30.

Chen, N., P. Glasserman. 2007. Additive and multiplicative duals for American option pricing. *Finance and Stochastics* **11**(2) 153–179.

Cox, J. C., S. A. Ross, M. Rubinstein. 1979. Option pricing: A simplified approach. *Journal of Financial Economics* **7**(3) 229–263.

De Farias, D. P., B. Van Roy. 2003. The linear programming approach to approximate dynamic programming. *Operations research* **51**(6) 850–865.

Desai, V. V., V. F. Farias, C. C. Moallemi. 2012a. Approximate dynamic programming via a smoothed linear program. *Operations Research* **60**(3) 655–674.

Desai, V. V., V. F. Farias, C. C. Moallemi. 2012b. Pathwise optimization for optimal stopping problems. *Management Science* **58**(12) 2292–2308.

Doshi-Velez, F., B. Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* .

Dunning, I., J. Huchette, M. Lubin. 2017. Jump: A modeling language for mathematical optimization. *SIAM Review* **59**(2) 295–320.

Garey, M. R., D. S. Johnson. 1979. *Computers and intractability*. W. H. Freeman New York.

Glasserman, P. 2013. *Monte Carlo methods in financial engineering*, vol. 53. Springer Science & Business Media.

Gross, K. 2018. `mvnmle`: Ml estimation for multivariate normal data with missing values. *R package version* **0.1-11.1**.

Gurobi Optimization, Inc. 2018. Gurobi Optimizer Reference Manual. URL `http://www.gurobi.com`.

Haugh, M. B., L. Kogan. 2004. Pricing American options: a duality approach. *Operations Research* **52**(2) 258–270.

Lakkaraju, H., S. H. Bach, J. Leskovec. 2016. Interpretable decision sets: A joint framework for description and prediction. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 1675–1684.

Letham, B., C. Rudin, T. H. McCormick, D. Madigan. 2015. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics* **9**(3) 1350–1371.

Longstaff, F. A., E. S. Schwartz. 2001. Valuing American options by simulation: a simple least-squares approach. *The Review of Financial Studies* **14**(1) 113–147.

Lubin, M., I. Dunning. 2015. Computing in operations research using julia. *INFORMS Journal on Computing* **27**(2) 238–248.

Powell, W. B. 2007. *Approximate Dynamic Programming: Solving the curses of dimensionality*, vol. 703. John Wiley & Sons.

Quinlan, J. R. 1986. Induction of decision trees. *Machine learning* **1**(1) 81–106.

Quinlan, J. R. 1993. C4. 5: Programming for machine learning. *Morgan Kauffmann* .

Rogers, L. C. G. 2002. Monte Carlo valuation of American options. *Mathematical Finance* **12**(3) 271–286.

Tsitsiklis, J. N., B. Van Roy. 2001. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks* **12**(4) 694–703.

Ustun, B., C. Rudin. 2015. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning* **102**(3) 349–391.

Ustun, B., C. Rudin. 2016. Learning optimized risk scores on large-scale datasets. *arXiv preprint arXiv:1610.00168* .

Van Roy, B. 2002. Neuro-dynamic programming: Overview and recent trends. *Handbook of Markov decision processes*. Springer, 431–459.

Wang, T., C. Rudin. 2015. Learning optimized or's of and's. *arXiv preprint arXiv:1511.02210* .

Wang, T., C. Rudin, F. Doshi, Y. Liu, E. Klampfl, P. MacNeille. 2017. A bayesian framework for learning rule set for interpretable classification. *Journal of Machine Learning Research* .

Wang, T., C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl, P. MacNeille. 2015. Or's of and's for interpretable classification, with application to context-aware recommender systems. *arXiv preprint arXiv:1504.07614* .

Zeng, J., B. Ustun, C. Rudin. 2017. Interpretable classification models for recidivism prediction. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* **180**(3) 689–722.

Proofs and Additional Numerical Results

EC.1. Proofs

EC.1.1. Proofs and additional lemmas for Section 3.4

In this section, we focus now on proving Theorem 1 and Corollary 1. Our proof relies on the fact that, via the strong law of large numbers, $J^\pi(\bar{\mathbf{x}}) \to \hat{J}^\pi(\bar{\mathbf{x}})$ almost surely for a *fixed* policy π. On the other hand, Theorem 1 requires almost sure convergence to hold simultaneously for *all* policies in $\Pi_{\text{tree}}(d)$, which is a possibly uncountable set. This implies that we cannot directly apply the previous fact which applies only for a fixed policy; instead, we construct a more intricate argument.

First, we deal with the issue of the uncountability of $\Pi_{\text{tree}}(d)$, where each policy π in this space is induced by a tuple $(\mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$ specifying a tree. Observe that since we have bounded the depth of the tree by d, the space of tree topologies, split variable indices and leaf actions $(\mathcal{T}, \mathbf{v}, \mathbf{a})$ is finite, although the space of split points $\boldsymbol{\theta}$ remains uncountable. Thus, we need to prove that, fixing the tuple $(\mathcal{T}, \mathbf{v}, \mathbf{a})$, we have almost sure convergence over all possible choice of split points $\boldsymbol{\theta} \in \mathcal{X}$.

In order to make the parametrization of a tree policy π in terms of the tree parameters explicit, we use the notation $\pi(\cdot; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$. When unambiguous, for readability we use $\pi_{\boldsymbol{\theta}}$ to denote the tree policy $\pi(\cdot; \mathcal{T}, \mathbf{v}, \boldsymbol{\theta}, \mathbf{a})$ and $\tau_{\boldsymbol{\theta}}$ to denote $\tau_{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta},\mathbf{a})}$, the stopping time induced by this policy.

LEMMA EC.1. *Under Assumptions 1, 2 and 3, for any fixed tree parameters $(\mathcal{T}, \mathbf{v}, \mathbf{a})$, starting state $\bar{\mathbf{x}}$ and arbitrary $\epsilon > 0$, almost surely, there exists finite $\Omega_1 \in \mathbb{N}^+$ such that for all $\Omega \geq \Omega_1$ and all split points $\boldsymbol{\theta} \in \mathcal{X}$,*

$$\left| J^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta},\mathbf{a})}(\bar{\mathbf{x}}) - \hat{J}^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta},\mathbf{a})}(\bar{\mathbf{x}}) \right| \leq \epsilon.$$

Proof. For some parameter $\delta > 0$ which we will set later, let us pick a set $Q_\delta = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$ to be a δ-cover of $\mathcal{X}$ in the $||\cdot||_\infty$ norm. That is, for any $\boldsymbol{\theta} \in \mathcal{X}$, there exists $\tilde{\boldsymbol{\theta}} \in Q_\delta$ such that $||\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}||_\infty \leq \delta$. The existence of such a cover is guaranteed by Assumption 1. For any $\boldsymbol{\theta} \in \mathcal{X}$, we then have

$$\begin{aligned}
\left| J^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta},\mathbf{a})}(\bar{\mathbf{x}}) - \hat{J}^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta},\mathbf{a})}(\bar{\mathbf{x}}) \right| &= \left| J^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}}) - \hat{J}^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}}) \right| \\
&\leq \underbrace{\left| J^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}}) - J^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) \right|}_{(a)} + \underbrace{\left| J^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) - \hat{J}^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) \right|}_{(b)} + \underbrace{\left| \hat{J}^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) - \hat{J}^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}}) \right|}_{(c)},
\end{aligned}$$

for some $\tilde{\boldsymbol{\theta}} \in Q_\delta$. We now bound each of the three terms above. By Lemma EC.2 and using the fact that $|\theta_s - \tilde{\theta}_s| \leq \delta$ due to Q_δ being a δ-cover, (a) is upper bounded by

$$GT \sum_{s \in \mathbf{splits}} f(|\theta_s - \tilde{\theta}_s|) \leq GT|\mathbf{splits}| f(\delta).$$

For term (b), by the strong law of large numbers and using the fact that Q_δ is finite, there exists some $\Omega_b \in \mathbb{N}^+$ such that for all $\Omega \geq \Omega_b$ and any $\tilde{\boldsymbol{\theta}} \in Q_\delta$, $\left| J^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) - \hat{J}^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) \right| \leq \epsilon/3$ almost surely.

Finally, for (c), by Lemma EC.3,

$$\begin{aligned}
\left| \hat{J}^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) - \hat{J}^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}}) \right| &\leq \frac{G}{\Omega} \sum_{\omega=1}^{\Omega} \sum_{t \in [T]} \sum_{s \in \mathbf{splits}} \mathbb{I}\left\{ x_{v(s)}(\omega, t) \in [\min\left\{\theta_s, \tilde{\theta}_s\right\}, \max\left\{\theta_s, \tilde{\theta}_s\right\}] \right\} \\
&\leq \frac{G}{\Omega} \sum_{\omega=1}^{\Omega} \sum_{t \in [T]} \sum_{s \in \mathbf{splits}} \mathbb{I}\left\{ x_{v(s)}(\omega, t) \in [\tilde{\theta}_s - \delta, \tilde{\theta}_s + \delta] \right\}.
\end{aligned}$$

Furthermore, since Q_δ is a finite set, we can again invoke the strong law of large numbers to show that for some $\gamma > 0$ to be chosen later, there exists some Ω_c such that for all $\Omega \geq \Omega_c$ and all $\tilde{\boldsymbol{\theta}} \in Q_\delta$,

$$\begin{aligned}\frac{G}{\Omega}\sum_{\omega=1}^{\Omega}\sum_{t\in[T]}\sum_{s\in\mathbf{splits}}\mathbb{I}\left\{x_{v(s)}(\omega,t)\in[\tilde{\theta}_s-\delta,\tilde{\theta}_s+\delta]\right\} &\leq G\sum_{t\in[T]}\sum_{s\in\mathbf{splits}}\Pr\left[x_{v(s)}\in[\tilde{\theta}_s-\delta,\tilde{\theta}_s+\delta]\right]+\gamma\\ &\leq GT|\mathbf{splits}|f(2\delta)+\gamma\end{aligned}$$

where in the last inequality we have used Assumption 3. Thus, for all $\tilde{\boldsymbol{\theta}} \in Q_\delta$,

$$\left|\hat{J}^{\pi_{\tilde{\boldsymbol{\theta}}}}(\bar{\mathbf{x}}) - \hat{J}^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}})\right| \leq GT|\mathbf{splits}|f(2\delta)+\gamma.$$

Setting $\gamma = \epsilon/6$, and δ such that $GT|\mathbf{splits}|f(2\delta) \leq \epsilon/6$, which we can do by parts 2 and 3 of Assumption 3, we obtain that $GT|\mathbf{splits}|f(2\delta)+\gamma \leq \epsilon/3$.

Putting everything together, we obtain that, almost surely, for any $\Omega \geq \max\{\Omega_b, \Omega_c\}$,

$$\left|J^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}}) - \hat{J}^{\pi_{\boldsymbol{\theta}}}(\bar{\mathbf{x}})\right| \leq \epsilon, \quad \text{for all } \boldsymbol{\theta} \in \mathcal{X}. \quad \square$$

We now state and prove two lemmas which we have used in proving Lemma EC.1 above:

LEMMA EC.2. *Under Assumptions 2 and 3, for initial state $\bar{\mathbf{x}}$ and any fixed tree parameters $(\mathcal{T}, \mathbf{v}, \mathbf{a})$ and $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathcal{X}$,*

$$\left|J^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta}_1,\mathbf{a})}(\bar{\mathbf{x}}) - J^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta}_2,\mathbf{a})}(\bar{\mathbf{x}})\right| \leq GT\sum_{s\in\mathbf{splits}} f(|\theta_{1,s}-\theta_{2,s}|).$$

Proof. We construct an upper bound on the difference in values of the two policies defined by $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. First, by Jensen's inequality, we have,

$$\begin{aligned}\left|J^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta}_1,\mathbf{a})}(\bar{\mathbf{x}}) - J^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta}_2,\mathbf{a})}(\bar{\mathbf{x}})\right| &= \left|J^{\pi_{\boldsymbol{\theta}_1}}(\bar{\mathbf{x}}) - J^{\pi_{\boldsymbol{\theta}_2}}(\bar{\mathbf{x}})\right|\\ &= \left|\mathbb{E}\left[\beta^{\tau_{\boldsymbol{\theta}_1}-1}g(\tau_{\boldsymbol{\theta}_1},\mathbf{x}(\tau_{\boldsymbol{\theta}_1})) - \beta^{\tau_{\boldsymbol{\theta}_2}-1}g(\tau_{\boldsymbol{\theta}_2},\mathbf{x}(\tau_{\boldsymbol{\theta}_2}))\right]\right|\\ &\leq \mathbb{E}\left[\left|\beta^{\tau_{\boldsymbol{\theta}_1}-1}g(\tau_{\boldsymbol{\theta}_1},\mathbf{x}(\tau_{\boldsymbol{\theta}_1})) - \beta^{\tau_{\boldsymbol{\theta}_2}-1}g(\tau_{\boldsymbol{\theta}_2},\mathbf{x}(\tau_{\boldsymbol{\theta}_2}))\right|\right].\end{aligned}$$

We further bound the RHS of the last equation by conditioning on whether the stopping times of the two policies induced by $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ agree:

$$\begin{aligned}&\mathbb{E}\left[\left|\beta^{\tau_{\boldsymbol{\theta}_1}-1}g(\tau_{\boldsymbol{\theta}_1},\mathbf{x}(\tau_{\boldsymbol{\theta}_1})) - \beta^{\tau_{\boldsymbol{\theta}_2}-1}g(\tau_{\boldsymbol{\theta}_2},\mathbf{x}(\tau_{\boldsymbol{\theta}_2}))\right|\right]\\
&\quad= \mathbb{E}\left[\left|\beta^{\tau_{\boldsymbol{\theta}_1}-1}g(\tau_{\boldsymbol{\theta}_1},\mathbf{x}(\tau_{\boldsymbol{\theta}_1})) - \beta^{\tau_{\boldsymbol{\theta}_2}-1}g(\tau_{\boldsymbol{\theta}_2},\mathbf{x}(\tau_{\boldsymbol{\theta}_2}))\right| \middle| \tau_{\boldsymbol{\theta}_1} = \tau_{\boldsymbol{\theta}_2}\right]\Pr\left[\tau_{\boldsymbol{\theta}_1} = \tau_{\boldsymbol{\theta}_2}\right]\\
&\qquad+ \mathbb{E}\left[\left|\beta^{\tau_{\boldsymbol{\theta}_1}-1}g(\tau_{\boldsymbol{\theta}_1},\mathbf{x}(\tau_{\boldsymbol{\theta}_1})) - \beta^{\tau_{\boldsymbol{\theta}_2}-1}g(\tau_{\boldsymbol{\theta}_2},\mathbf{x}(\tau_{\boldsymbol{\theta}_2}))\right| \middle| \tau_{\boldsymbol{\theta}_1} \neq \tau_{\boldsymbol{\theta}_2}\right]\Pr\left[\tau_{\boldsymbol{\theta}_1} \neq \tau_{\boldsymbol{\theta}_2}\right]\\
&\quad\leq 0\cdot\Pr\left[\tau_{\boldsymbol{\theta}_1} = \tau_{\boldsymbol{\theta}_2}\right] + G\cdot\Pr\left[\tau_{\boldsymbol{\theta}_1} \neq \tau_{\boldsymbol{\theta}_2}\right]\\
&\quad\leq G\cdot\Pr\left[\exists t\in[T] \text{ s.t. } \pi_{\boldsymbol{\theta}_1}(t,\mathbf{x}(t)) \neq \pi_{\boldsymbol{\theta}_2}(t,\mathbf{x}(t))\right]\\
&\quad\leq G\sum_{t\in[T]}\Pr\left[\pi_{\boldsymbol{\theta}_1}(t,\mathbf{x}(t)) \neq \pi_{\boldsymbol{\theta}_2}(t,\mathbf{x}(t))\right]\\
&\quad\leq G\sum_{t\in[T]}\Pr\left[\exists s\in\mathbf{splits} \text{ s.t. } x_{v(s)}(t)\in[\min\{\theta_{1s},\theta_{2s}\},\max\{\theta_{1s},\theta_{2s}\}]\right]\\
&\quad\leq G\sum_{t\in[T]}\sum_{s\in\mathbf{splits}}\Pr\left[x_{v(s)}(t)\in[\min\{\theta_{1s},\theta_{2s}\},\max\{\theta_{1s},\theta_{2s}\}]\right]\\
&\quad\leq GT\sum_{s\in\mathbf{splits}} f(|\theta_{1s}-\theta_{2s}|),\end{aligned}$$

where in the first inequality, we have used Assumption 2, which guarantees that the difference in values of π_{θ_1} and π_{θ_2} cannot be larger than G; in the second, the fact that the stopping times being different implies that $\pi_{\boldsymbol{\theta}_1}$ and $\pi_{\boldsymbol{\theta}_2}$ differ in their action at some $t \in [T]$; in the third and fifth, a union bound over times and splits, respectively; in the fourth, the fact that $\pi_{\boldsymbol{\theta}_1}$ and $\pi_{\boldsymbol{\theta}_2}$ disagreeing at t implies that there is some split at which the policies disagree (i.e., the split variable is mapped to different child nodes in the two policies); and in the last, Assumption 3. $\square$

LEMMA EC.3. *Under Assumption 2, for starting state $\bar{\mathbf{x}}$ and any fixed tree parameters $(\mathcal{T}, \mathbf{v}, \mathbf{a})$ and $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathcal{X}$,*

$$\left|\hat{J}^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta}_1,\mathbf{a})}(\bar{\mathbf{x}}) - \hat{J}^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta}_2,\mathbf{a})}(\bar{\mathbf{x}})\right| \leq \frac{G}{\Omega}\sum_{\omega=1}^{\Omega}\sum_{t\in[T]}\sum_{s\in\textbf{splits}}\left[\mathbb{I}\left\{x_{v(s)} \in [\min\{\theta_{1s},\theta_{2s}\}, \max\{\theta_{1s},\theta_{2s}\}]\right\}\right].$$

Proof. Similarly to the argument for Lemma EC.2,

$$\begin{aligned}
&\left|\hat{J}^{\pi(\cdot;\mathcal{T}_0,\mathbf{v}_0,\boldsymbol{\theta}_1,\mathbf{a}_0)}(\bar{\mathbf{x}}) - \hat{J}^{\pi(\cdot;\mathcal{T}_0,\mathbf{v}_0,\boldsymbol{\theta}_2,\mathbf{a}_0)}(\bar{\mathbf{x}})\right| \\
&= \left|\frac{1}{\Omega}\sum_{\omega=1}^{\Omega}\left[\beta^{\tau_{\boldsymbol{\theta}_1,\omega}-1} g(\tau_{\boldsymbol{\theta}_1,\omega}, \mathbf{x}(\tau_{\boldsymbol{\theta}_1,\omega})) - \frac{1}{\Omega}\sum_{\omega=1}^{\Omega}\beta^{\tau_{\boldsymbol{\theta}_2,\omega}-1} g(\tau_{\boldsymbol{\theta}_2,\omega}, \mathbf{x}(\tau_{\boldsymbol{\theta}_2,\omega}))\right]\right| \\
&\leq \frac{1}{\Omega}\sum_{\omega=1}^{\Omega}\left|\beta^{\tau_{\boldsymbol{\theta}_1,\omega}-1} g(\tau_{\boldsymbol{\theta}_1,\omega}, \mathbf{x}(\tau_{\boldsymbol{\theta}_1,\omega})) - \beta^{\tau_{\boldsymbol{\theta}_2,\omega}-1} g(\tau_{\boldsymbol{\theta}_2,\omega}, \mathbf{x}(\tau_{\boldsymbol{\theta}_2,\omega}))\right| \\
&\leq \frac{1}{\Omega}\sum_{\omega=1}^{\Omega} G\mathbb{I}\left\{\tau_{\boldsymbol{\theta}_1,\omega} \neq \tau_{\boldsymbol{\theta}_2,\omega}\right\} \\
&\leq \frac{1}{\Omega}\sum_{\omega=1}^{\Omega} G\sum_{t\in[T]}\mathbb{I}\left\{\pi_{\boldsymbol{\theta}_1}(t, \mathbf{x}(\omega,t)) \neq \pi_{\boldsymbol{\theta}_2}(t, \mathbf{x}(\omega,t))\right\} \\
&\leq \frac{G}{\Omega}\sum_{\omega=1}^{\Omega}\sum_{t\in[T]}\sum_{s\in\textbf{splits}}\mathbb{I}\left\{x_{v(s)}(\omega,t) \in [\min\{\theta_{1s},\theta_{2s}\}, \max\{\theta_{1s},\theta_{2s}\}]\right\},
\end{aligned}$$

where in the second inequality we used Assumption 2. $\square$

We end this section with the proofs of Theorem 1 and Corollary 1 from Section 3.4.

Proof of Theorem 1. Since we are restricting our analysis to policies in $\Pi_{\text{tree}}(d)$, it follows that there is a finite number of tuples $(\mathcal{T}, \mathbf{v}, \mathbf{a})$. For each choice of such tuple, almost surely, $\left|J^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta},\mathbf{a})}(\bar{\mathbf{x}}) - \hat{J}^{\pi(\cdot;\mathcal{T},\mathbf{v},\boldsymbol{\theta},\mathbf{a})}(\bar{\mathbf{x}})\right| \leq \epsilon$ for all $\boldsymbol{\theta} \in \mathcal{X}$ and $\Omega \geq \Omega(\mathcal{T}, \mathbf{v}, \mathbf{a})$. The result follows by taking $\Omega_0 = \max_{(\mathcal{T},\mathbf{v},\mathbf{a})}\{\Omega(\mathcal{T}, \mathbf{v}, \mathbf{a})\}$. $\square$

Proof of Corollary 1. Consider the case that $\sup_{\pi\in\Pi_{\text{tree}}(d)} J^{\pi}(\bar{\mathbf{x}}) \geq \sup_{\pi\in\Pi_{\text{tree}}(d)} \hat{J}^{\pi}(\bar{\mathbf{x}})$. This is without loss of generality since the other case is symmetric. We know that there must exist some policy $\underline{\pi} \in \Pi_{\text{tree}}(d)$ such that

$$J^{\underline{\pi}}(\bar{\mathbf{x}}) \geq \sup_{\pi\in\Pi_{\text{tree}}(d)} J^{\pi}(\bar{\mathbf{x}}) - \frac{\epsilon}{2}. \tag{EC.1}$$

Additionally, by Theorem 1, almost surely there exists Ω_0 such that for all $\Omega \geq \Omega_0$, $\left|J^{\pi}(\bar{\mathbf{x}}) - \hat{J}^{\pi}(\bar{\mathbf{x}})\right| \leq \epsilon/2$ for all policies $\pi \in \Pi_{\text{tree}}(d)$ and thus,

$$\hat{J}^{\underline{\pi}}(\bar{\mathbf{x}}) \geq J^{\underline{\pi}}(\bar{\mathbf{x}}) - \frac{\epsilon}{2} \text{ a.s.} \tag{EC.2}$$

Together, equations (EC.1) and (EC.2) imply that

$$\begin{aligned}\left|\sup_{\pi\in\Pi_{\text{tree}}(d)} J^\pi(\bar{\mathbf{x}}) - \sup_{\pi\in\Pi_{\text{tree}}(d)} \hat{J}^\pi(\bar{\mathbf{x}})\right| &= \sup_{\pi\in\Pi_{\text{tree}}(d)} J^\pi(\bar{\mathbf{x}}) - \sup_{\pi\in\Pi_{\text{tree}}(d)} \hat{J}^\pi(\bar{\mathbf{x}}) \\ &\leq \sup_{\pi\in\Pi_{\text{tree}}(d)} J^\pi(\bar{\mathbf{x}}) - \hat{J}^{\bar{\pi}}(\bar{\mathbf{x}}) \\ &\leq J^{\bar{\pi}}(\bar{\mathbf{x}}) - \hat{J}^{\bar{\pi}}(\bar{\mathbf{x}}) + \frac{\epsilon}{2} \\ &\leq \epsilon \text{ a.s.} \quad \square\end{aligned}$$

EC.1.2. Proofs for Section 3.5

EC.1.2.1. Proof of Proposition 1 We will prove this by showing that the minimum vertex cover problem reduces to the leaf action SAA problem (8). The minimum vertex cover problem is stated as follows:

MINIMUM VERTEX COVER: Given a graph (V, E), where V is the set of nodes and E is the set of edges, find the smallest subset of nodes $S \subseteq V$ such that every edge is covered, i.e., for all $e \in E$, there exists a node i in S such that e is incident to i.

Given a graph (V, E), we will now define an instance of the leaf action SAA problem. To do this, we need to specify a sample of trajectories, a tree topology, split variable indices and split points. We start by defining the trajectories. We assume that the nodes in V are indexed from 1 to $|V|$, and the edges in E are indexed from 1 to $|E|$.

Trajectories. We will assume that the ambient space of our stochastic process is $\mathcal{X} = \mathbb{R}^{|V|+1}$. We assume that we have $\Omega = |V| + |E|$ trajectories, each with $T = 3$ periods. We will consider two types of trajectories:

1. *Edge trajectories*: These trajectories are indexed by $\omega = 1, \ldots, |E|$, and each such trajectory corresponds to an edge in the graph. For each edge e, let e_1 and e_2 be the two vertices to which edge e is incident. For each trajectory ω that corresponds to an edge $e \in E$, we assume that it takes the following values:

$$\begin{aligned}x_i(1) &= 0, \quad \forall\, i \in \{1, \ldots, |V|+1\}, \\ x_i(2) &= \mathbb{I}\{i = e_1\}, \quad \forall\, i \in \{1, \ldots, |V|+1\}, \\ x_i(3) &= \mathbb{I}\{i = e_2\}, \quad \forall\, i \in \{1, \ldots, |V|+1\}.\end{aligned}$$

   In words, a trajectory for an edge e is zero at all coordinates, except at the vertices to which e is incident, for which the corresponding coordinates tick up to 1 at periods $t = 2$ and $t = 3$.
2. *Vertex trajectories*: These trajectories are indexed by $\omega = |E|+1, \ldots, |E|+|V|$, and each such trajectory corresponds to a vertex in the graph. For each such trajectory ω that corresponds to a vertex $v \in V$, we assume that it takes the following values:

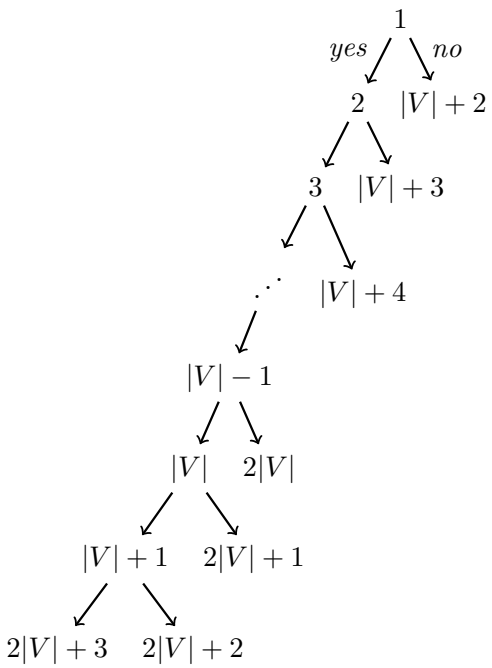
$$\begin{aligned}x_i(1) &= \mathbb{I}\{i = v\}, \quad \forall\, i \in \{1, \ldots, |V|+1\}, \\ x_i(2) &= 0, \quad \forall\, i \in \{1, \ldots, |V|+1\}, \\ x_i(3) &= \mathbb{I}\{i = |V|+1\}, \quad \forall\, i \in \{1, \ldots, |V|+1\}.\end{aligned}$$

   In words, this trajectory is zero in all coordinates, except for coordinates v and $|V|+1$: for coordinate v, it starts at 1 at $t = 1$, and comes down to 0 at $t = 2$, and for coordinate $|V|+1$, it start at 0 at $t = 1$ and ticks up to 1 at $t = 3$.

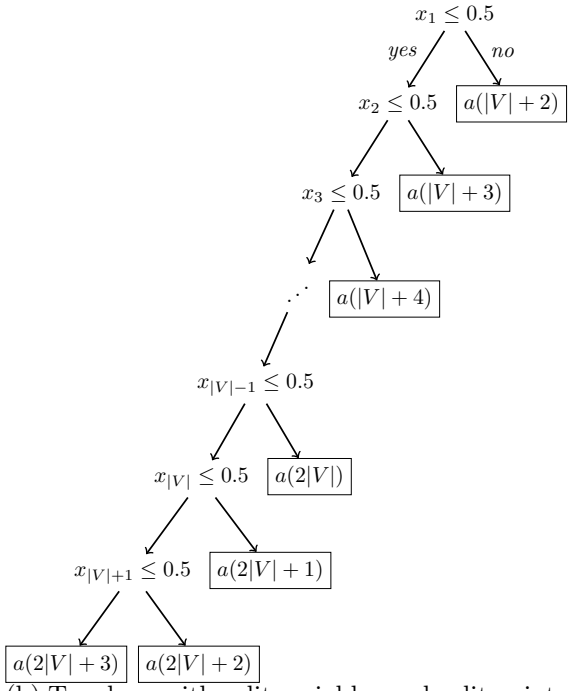
We define the payoff function g as follows:

$$g(t, \mathbf{x}) = \begin{cases} 1 & \text{if } t \in \{2, 3\} \text{ and } x_i = 1 \text{ for any } i \in \{1, \ldots, |V|\}, \\ 1/(|V|+1) & \text{if } t = 3 \text{ and } x_{|V|+1} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Figure EC.1 Visualization of topology of tree policy for proof of Proposition 1.



(a) Topology with split node and leaf node indices for Proposition 1 proof.



(b) Topology with split variables and split points for Proposition 1 proof.

To understand the payoff function, we receive a reward of 1 if we stop at $t=2$ or $t=3$ in a state $\mathbf{x}$ where any coordinate $i \in \{1,\dots,|V|\}$ is one; we receive a reward of $1/(|V|+1)$ if we stop at $t=3$ in a state $\mathbf{x}$ where coordinate $|V|+1$ is one; and for any other state-time pair, we receive zero. For each edge trajectory $\omega = 1,\dots,|E|$, this means that if we stop at $t=2$ or $t=3$ of any edge trajectory $\omega = 1,\dots,|E|$, we will get a reward of 1; otherwise, if we stop at $t=1$, we will get a reward of 0. For each vertex trajectory $\omega = |E|+1,\dots,|E|+|V|$, stopping at $t=1$ or $t=2$ gives us a reward of zero, and stopping at $t=3$ gives us a reward of $1/(|V|+1)$.

Tree topology. Our tree consists of $|V|+1$ splits and $|V|+2$ leaves. We number the split nodes from 1 to $|V|+1$, and we number the leaf nodes from $|V|+2$ to $2|V|+3$. We define the left and right children of the split nodes as follows:

$$\begin{aligned}
&\textbf{leftchild}(s) = s+1, \quad s = 1,\dots,|V|,\\
&\textbf{rightchild}(s) = |V|+1+s, \quad s = 1,\dots,|V|,\\
&\textbf{leftchild}(|V|+1) = 2|V|+3,\\
&\textbf{rightchild}(|V|+1) = 2|V|+2.
\end{aligned}$$

Figure EC.1a visually displays this topology.
Split variable indices and split points. For each split $s = 1,\dots,|V|+1$, we define the variable $v(s) = s$, and we define the split point $\theta(s) = 0.5$. Figure EC.1b shows the tree topology together with the split variable indices and split points.

Note that with this tree topology, split points and split variable indices, the policy will behave as follows:

1. For an edge trajectory corresponding to an edge e:

- At $t=1$, the policy will take action $a(2|V|+3)$;
- At $t=2$, the policy will take action $a(|V|+e_1)$; and
- At $t=3$, the policy will take action $a(|V|+e_2)$.

2. For a vertex trajectory corresponding to a vertex v:
   - At $t=1$, the policy will take action $a(|V|+1+v)$;
   - At $t=2$, the policy will take action $a(2|V|+3)$; and
   - At $t=3$, the policy will take action $a(2|V|+2)$.

Reduction of vertex cover to leaf action problem. Consider now the problem of deciding the leaf actions. We will show that the optimal solution of this problem provides an optimal solution of the vertex cover problem.

Let $\mathbf{a}^*$ be an optimal collection of leaf actions (i.e., an optimal solution of the leaf action SAA problem). Observe that the leaf actions for $\ell = |V|+2, \ldots, 2|V|+1$ can be interpreted as a collection of vertices: if the leaf action is **stop**, this indicates that the vertex is in the collection, whereas if the action is **go**, this indicates that the vertex is not in the collection. We therefore specify the set of vertices as $S = \{v \in V \mid a^*(|V|+1+v) = \textbf{stop}\}$.

We consider two cases:

Case 1: $E = \emptyset$, i.e., there are no edges. In this degenerate case, an upper bound on the reward we can achieve is given by $|V|/(|V|+1)$. (This is because the highest possible reward for each of the vertex trajectories is $1/(|V|+1)$, and there are $|V|$ such trajectories; recall that there are no edge trajectories.) This upper bound can only be attained by setting $a^*(|V|+1+v) = \textbf{go}$ for $v = 1, \ldots, |V|$, setting $a^*(2|V|+2) = \textbf{stop}$ and setting $a^*(2|V|+3) = \textbf{go}$. Note that in this case, the corresponding set of vertices S is exactly the empty set, which is exactly the optimal solution of the vertex cover problem in this case.

Case 2: $E \neq \emptyset$, i.e., there is at least one edge.

We first argue that the set S must be a bona fide cover. To see this, we argue by contradiction. Suppose that S is not a cover of E; then there exists an edge e such that the corresponding vertices i and j are not in the cover. This would mean that both $a^*(|V|+1+i)$ and $a^*(|V|+1+j)$ are set to **go**. This, in turn means that the reward of 1 of the edge trajectory e cannot be earned by the tree policy. (To obtain the reward, we must stop at $t=2$ or $t=3$ in this trajectory; given the trajectory and the topology, split variable indices and split points, this can only happen if at least one of the two actions is **stop**.) This means that the reward that the policy can obtain is upper bounded by $|E|-1+|V|/(|V|+1)$. However, if we set $\mathbf{a}$ so that the corresponding vertex set S of $\mathbf{a}$ covers E, and we additionally set $a(2|V|+2) = \textbf{stop}$ and $a(2|V|+3) = \textbf{go}$, our reward will be at least $|E|$. Therefore, it must be that S covers all edges.

We now argue that S is a minimal cover. To see this, we again argue by contradiction. Suppose that there is a cover S' such that $|S'| < |S|$. Consider the set of leaf actions $\mathbf{a}'$, which is defined as follows:

$$
\begin{aligned}
a'(|V|+1+v) &= \begin{cases} \textbf{stop} & \text{if } v \in S', \\ \textbf{go} & \text{if } v \notin S', \end{cases} \\
a'(2|V|+2) &= \textbf{stop}, \\
a'(2|V|+3) &= \textbf{go}.
\end{aligned}
$$

The corresponding tree policy with $\mathbf{a}'$ has the following behavior. For all edge trajectories, it stops at $t=2$ or $t=3$, because S' is a cover; as a result, it accrues a reward of 1 from all edge trajectories. For those vertex trajectories corresponding to vertices in S', it stops at $t=2$, accruing a reward of zero. For those vertex trajectories corresponding to vertices *not* in S', it stops at $t=3$, accruing a reward of $1/(|V|+1)$. Therefore, the total reward is $|E| + (|V| - |S'|)/(|V|+1)$.

Now, we show that for $\mathbf{a}^*$, the reward is upper bounded by $|E| + (|V| - |S|)/(|V|+1)$. Observe that if this is the case, we immediately have our contradiction because the reward $|E| + (|V| -$

$|S|)/(|V|+1)$ will be smaller than $|E|+(|V|-|S'|)/(|V|+1)$ (this follows from $|S|>|S'|$), which will contradict the fact that $\mathbf{a}^*$ is an optimal solution of the leaf action SAA problem. To see this, observe that for each edge trajectory, since S is a cover, the policy can garner at most a reward of 1, resulting in a total reward of $|E|$ from the edge trajectories. For the vertex trajectories, observe that for all vertices $v \in S$, the reward for the corresponding vertex trajectory must be zero, because the tree policy must stop at $t=1$ (the reward for any vertex trajectory at $t=1$ is zero). For each vertex trajectory corresponding to a vertex $v \notin S$, the most reward that can be garnered is $1/(|V|+1)$ (obtained by stopping at $t=3$). Therefore, an upper bound for the reward is

$$\begin{aligned}&|E|+|V\setminus S|/(|V|+1)\\ =&|E|+(|V|-|S|)/(|V|+1).\end{aligned}$$

This gives us our contradiction, and establishes that S is a minimal cover.

We have thus shown that any instance (V,E) of the minimum vertex cover problem can be transformed to an instance of the leaf action problem (8); moreover, the instance is polynomially-sized in terms of V and E. Since the minimum vertex cover problem is known to be NP-Complete (Garey and Johnson 1979), it follows that the leaf action problem is NP-Hard. □

EC.1.2.2. Proof of Proposition 2 To prove that the split variable index SAA problem is NP-Hard, we will again show that the minimum vertex cover problem reduces to it. We will prove the result by considering an instance (V,E) of the minimum vertex cover problem, and considering the same set of trajectories, payoff function and tree topology from the proof of Proposition 1.

In this version of the overall tree optimization problem, the leaf actions $\mathbf{a}$ are now fixed, and the split variable indices must be chosen. For the leaf actions, we set them as follows:

$$\begin{aligned}&a(|V|+1+i)=\textbf{stop}, \quad \forall\, i\in V,\\ &a(2|V|+2)=\textbf{stop},\\ &a(2|V|+3)=\textbf{go}.\end{aligned}$$

We also use $v(1),\ldots,v(|V|+1)$ to denote the split variable indices of splits $s=1,\ldots,|V|+1$. Figure EC.2 visualizes the tree policy structure.

Note that with this tree topology, split points and leaf actions, the policy will behave as follows:

1. For an edge trajectory corresponding to an edge e:
   - At $t=1$, the policy will take the action **go**;
   - At $t=2$, the policy will take the action **stop** if and only if $v(s)=e_1$ for some $s\in\{1,\ldots,|V|+1\}$; and
   - At $t=3$, the policy will take the action **stop** if and only if $v(s)=e_2$ for some $s\in\{1,\ldots,|V|+1\}$.
2. For a vertex trajectory corresponding to a vertex v:
   - At $t=1$, the policy will take the action **stop** if and only if $v(s)=v$ for some $s\in\{1,\ldots,|V|+1\}$;
   - At $t=2$, the policy will take the action **go**; and
   - At $t=3$, the policy will take the action **stop** if and only if $v(s)=|V|+1$, for some $s\in\{1,\ldots,|V|+1\}$.

Let $\mathbf{v}^*$ be an optimal solution of the corresponding split variable index SAA problem. Let $S=\{i\in V, |\, v(s)=i \text{ for some } s\in\textbf{splits}\}$ be the set of split variable indices. Effectively, $\mathbf{v}^*$ encodes a subset of the vertices of V, and we denote this subset by S. We show that S is an optimal solution of the vertex cover problem. We split our analysis in two cases:

Case 1: $E=\emptyset$. In the case that there are no edges, an upper bound on the objective value of the split variable index SAA problem is $|V|/(|V|+1)$, which follows because the most reward that can be obtained from any of the vertex trajectories is $1/(|V|+1)$ and there are $|V|$ such

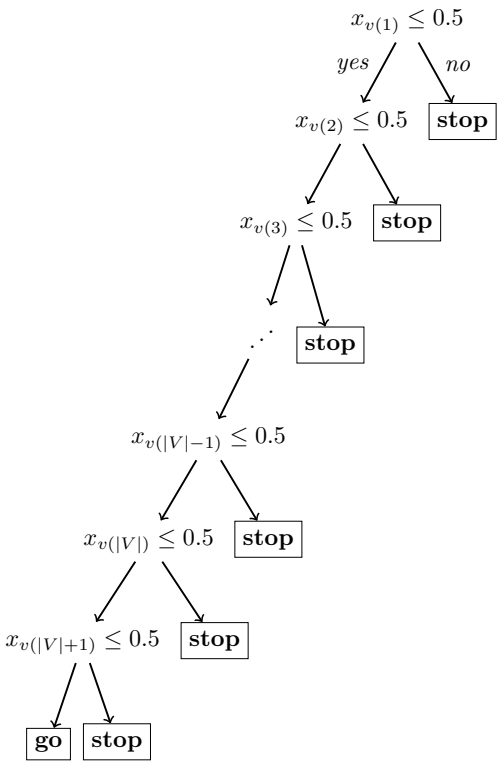


Figure EC.2 Tree policy for Proposition 2 proof.

trajectories. This upper bound can only be attained by setting $v(1) = \cdots = v(|V|+1) = |V|+1$, which corresponds to a set of vertices $S = \emptyset$, which is precisely the optimal solution in this case.

Case 2: $E \neq \emptyset$. In this case, we show that S is both a feasible cover and a minimal cover.

To see that S is feasible, suppose that S were not a feasible cover. This would mean that there exists an edge e such that the two vertices that are incident to this edge, e_1 and e_2, are not chosen as split variable indices – in other words, for all $s \in \mathbf{splits}$, $v(s) \notin \{e_1, e_2\}$. This means that the policy cannot garner the reward of 1 from the edge trajectory corresponding to e. As a result, the reward of the tree policy that corresponds to $\mathbf{v}^*$ is upper bounded by $|E| - 1 + |V|/(|V|+1)$. However, note that one can attain a reward of at least $|E|$, which is higher, by simply setting $v(1) = 1, v(2) = 2, \ldots, v(|V|) = |V|$ in the tree policy (this policy is guaranteed to obtain a reward of 1 from every edge trajectory). This leads to a contradiction, because $\mathbf{v}^*$ is assumed to be optimal. Therefore, it must be the case that S is a feasible cover.

To see that S is a minimal cover, suppose that we could find $S' \subseteq V$ that covers E such that $|S'| < |S|$. Enumerate the nodes in S' as $S' = \{i_1, \ldots, i_{|S'|}\}$, and consider the solution $\mathbf{v}'$ to the split variable index SAA problem obtained by setting

$$
\begin{aligned}
&v(1) = i_1,\\
&v(2) = i_2,\\
&\vdots\\
&v(|S'|) = i_{|S'|},\\
&v(|S'|+1) = |V|+1,\\
&\vdots\\
&v(|V|) = |V|+1,
\end{aligned}
$$

$$v(|V|+1) = |V|+1.$$

The resulting tree policy will stop at either $t=2$ or $t=3$ in every edge trajectory (because S' is assumed to be a cover), and accrue a reward of 1 from each such trajectory. For each vertex $i \in S'$, it will stop at $t=1$ in the corresponding vertex trajectory, and obtain a reward of 0. For each vertex $i \notin S'$, it will stop at $t=3$ in the ith vertex trajectory and obtain a reward of $1/(|V|+1)$. The total reward is therefore $|E| + (|V| - |S'|)/(|V|+1)$.

Observe that for $\mathbf{v}^*$, the total reward is at most $|E| + (|V| - |S|)/(|V|+1)$. Since $|S'| < |S|$, it follows that

$$|E| + (|V| - |S|)/(|V|+1) < |E| + (|V| - |S'|)/(|V|+1),$$

which implies that $\mathbf{v}'$ attains a higher objective than $\mathbf{v}^*$ in the split variable index SAA problem. This immediately results in a contradiction, because $\mathbf{v}^*$ is assumed to be an optimal solution of that problem. It therefore follows that the cover S we defined above in terms of $\mathbf{v}^*$ is a minimal cover.

Thus, by solving the split variable index problem, we are able to solve the minimum vertex cover problem. Since the minimum vertex cover problem is NP-Complete and the instance we have described is polynomially sized in terms of E and V, it follows that the split variable index problem must be NP-Hard. □

EC.1.2.3. Proof of Proposition 3 The proof that the split point SAA problem is NP-Hard follows along similar lines as the proofs of Propositions 1 and 2 – in particular, by fixing a specific tree topology, split variable indices and leaf actions, and a specific set of trajectories, one can show that the split point SAA problem can be used to solve the minimum vertex cover problem.

Given an instance (V, E) of the minimum vertex cover problem, consider the same set of trajectories and payoffs as in the proofs of Propositions 2, and the same tree topology and split variable indices. Define the leaf actions as

$$\begin{aligned}
&a(|V|+2) = \textbf{stop},\\
&a(|V|+3) = \textbf{stop},\\
&\vdots\\
&a(2|V|+1) = \textbf{stop},\\
&a(2|V|+2) = \textbf{stop},\\
&a(2|V|+3) = \textbf{go}.
\end{aligned}$$

Figure EC.3 visualizes the tree policy structure.

Note that with this tree topology, split variable indices and leaf actions, the policy will behave as follows:

1. For an edge trajectory corresponding to an edge e:
   - At $t=1$, the policy will take the action **go** if and only if $\theta(s) \geq 0$ for all s;
   - At $t=2$, the policy will take the action **go** if and only if $\theta(s) \geq 0$ for all $s \neq e_1$ and $\theta(e_1) \geq 1$; and
   - At $t=3$, the policy will take the action **go** if and only if $\theta(s) \geq 0$ for all $s \neq e_2$ and $\theta(e_2) \geq 1$.
2. For a vertex trajectory corresponding to a vertex v:
   - At $t=1$, the policy will take the action **go** if and only if $\theta(s) \geq 0$ for all $s \neq v$ and $\theta(v) \geq 1$;
   - At $t=2$, the policy will take the action **go** if and only if $\theta(s) \geq 0$ for all s; and
   - At $t=3$, the policy will take the action **go** if and only if $\theta(s) \geq 0$ for all $s \neq |V|+1$ and $\theta(|V|+1) \geq 1$.

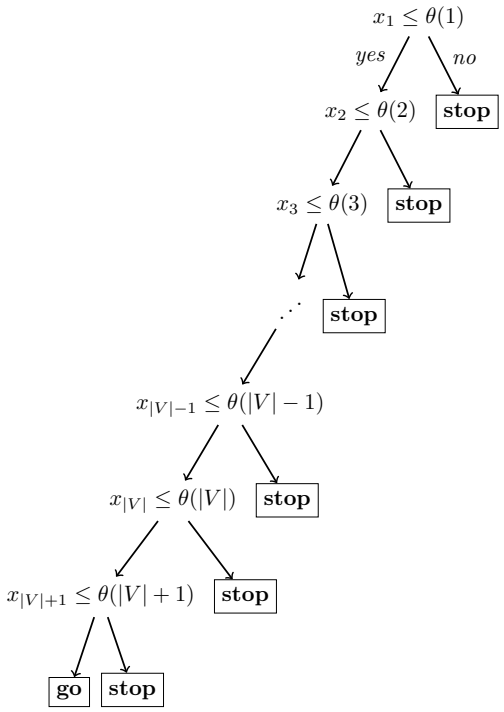


Figure EC.3 Tree policy for Proposition 3 proof.

Let $\boldsymbol{\theta}^* = (\theta^*(1), \theta^*(2), \ldots, \theta^*(|V|+1))$ be an optimal solution to the split point SAA problem. Define the set of vertices S as

$$S = \{i \in V \,|\, \theta(i) \in [0,1)\}.$$

We will now show that S is an optimal solution of the vertex cover problem.

Before we show this, we first argue that in any optimal solution $\boldsymbol{\theta}^*$ of the split point SAA problem, it must be that $\theta^*(i) \geq 0$ for all $i = 1, \ldots, |V|+1$. To see this, observe that if there exists an i such that $\theta^*(i) < 0$, then at $t = 1$, we will stop in every trajectory, because every coordinate of every trajectory is greater than or equal to 0 at $t = 1$. As a result, we will accrue a total reward of zero from such a policy. However, observe that if we set every $\theta(i)$ to 0.5, then we are guaranteed to stop at $t = 2$ or $t = 3$ in every edge trajectory and accrue a reward of 1 from each such trajectory; as a result, the total reward will be at least $|E|$, which would contradict the optimality of $\boldsymbol{\theta}^*$.

We now prove that S solves the minimum vertex cover problem. We consider two cases:

Case 1: $E = \emptyset$. In this case, observe that an upper bound on the reward that can be obtained is $|V|/(|V|+1)$. This reward can only be attained by setting $\theta(v) \geq 1$ for every $v = 1, \ldots, |V|$, and $\theta(|V|+1) \in [0,1)$. Note that the set of vertices S that corresponds to $\boldsymbol{\theta}$ in this case is the empty set, which is exactly the optimal solution of the minimum vertex cover problem in this case.

Case 2: $E \neq \emptyset$. In this case, we show that S both a feasible cover and a minimal cover.

To see that S is a feasible cover, suppose that S were not a feasible cover. This would mean that there is an edge e that is not covered, i.e., that $e_1, e_2 \notin S$. By the definition of S, this would mean that $\theta(e_1) \notin [0,1)$ and $\theta(e_2) \notin [0,1)$. Note that by the definition of the tree policy and the trajectories, this would mean that the policy defined by $\boldsymbol{\theta}^*$ either stops at $t = 1$ for the edge trajectory corresponding to e (resulting in a reward of zero from that trajectory), or it does not stop at any t (again, resulting in a reward of zero from the trajectory). As a result, the reward obtained by $\boldsymbol{\theta}^*$ would be upper bounded by $|E| - 1 + |V|/(|V|+1)$. However, observe that just by setting all $\boldsymbol{\theta}(s) = 0.5$ for $s = 1, \ldots, |V|+1$, we would obtain a policy that stops each edge trajectory

at either $t=2$ or $t=3$, which would guarantee a reward of at least $|E|$, which is greater. Since this would contradict the optimality of $\boldsymbol{\theta}^*$, it follows that S must be a feasible cover.

To see that S is a minimal cover, suppose this were not the case. This would mean that there exists a set S' that covers E and is smaller, i..e, $|S'|<|S|$. For this cover S', define a new split point vector $\boldsymbol{\theta}'$ as

$$\begin{aligned}
&\theta'(s)=0.5, \quad \forall s \in S', \\
&\theta'(s)=1, \quad \forall s \in V \setminus S', \\
&\theta'(|V|+1)=0.5.
\end{aligned}$$

Since S' covers E, this policy is guaranteed to stop at $t=2$ or $t=3$ for every edge trajectory, and thus will garner a reward of 1 from each such trajectory. For each vertex $i \in S$, it will stop at $t=1$ in the corresponding vertex trajectory, and garner a reward of zero. For each vertex $i \notin S$, it will stop at $t=3$ in the corresponding vertex trajectory, and earn a reward of $1/|V|+1$. Thus, the reward of the policy defined by $\boldsymbol{\theta}'$ will be $|E|+(|V|-|S'|)/(|V|+1)$.

Now, observe that for the optimal policy $\boldsymbol{\theta}^*$, since S corresponds to a feasible cover, the total reward from all edge trajectories for $\boldsymbol{\theta}^*$ is upper bounded by $|E|$. For $s \in S$, by definition of S, the reward from the vertex trajectory corresponding to s must be zero. For $s \in V \setminus S$, the reward from the vertex trajectory corresponding to s is upper bounded by $1/|V|+1$. Therefore, the reward of $\boldsymbol{\theta}^*$ is upper bounded by $|E|+(|V|-|S|)/(|V|+1)$. Since $|S'|<|S|$, this would imply that the reward of $\boldsymbol{\theta}'$ is greater than the reward of $\boldsymbol{\theta}$; however, this would contradict the fact that $\boldsymbol{\theta}^*$ is an optimal solution to the split point SAA problem. Therefore, it must be that S as defined above is a minimal cover.

This establishes that the split point SAA problem is NP-Hard, as required. $\square$

EC.2. Additional numerical results

EC.2.1. Additional results on sensitivity to γ

In our previous experiments, we estimated the tree policies with a fixed γ value of 0.005, which corresponds to a requirement of a 0.5% relative improvement in the objective with each iteration of the construction algorithm. In this section, we consider how the tree policies and their performance changes as γ varies.

To understand how the performance changes as a function of γ, we randomly generate training and testing sets consisting of 2000 and 100000 trajectories, respectively, for values of $n \in \{4, 8, 16\}$ and $\bar{p} \in \{90, 100, 110\}$, for 10 replications. We use a common correlation of $\bar{\rho} = 0$. For each replication, we run our construction algorithm on the training set and at each iteration, we compute the performance of the current tree on the training set and the testing set; we stop the construction algorithm at a γ value of 0.0001. (We do not test smaller values of γ, due to the prohibitively large number of iterations that such smaller tolerances result in.) We test three different sets of basis functions: PRICES; PRICES, TIME; and PRICES, TIME, PAYOFF, KOIND.

Table EC.1 shows the in-sample and out-of-sample performance of the tree policies at different values of γ for the three sets of basis functions. The values are averaged over the ten replications. To simplify the presentation, we focus on $n = 8$ and $\bar{p} = 90$, as the results for other values of n and $\bar{p}$ are qualitatively similar. While the in-sample performance improves as γ decreases, the out-of-sample performance improves up to a point, after which it begins to decrease. For example, with PRICES only, the out-of-sample performance begins to decrease for γ lower than 0.001. However, in all cases the deterioration appears to be mild (relative to the best average out-of-sample objective, the lowest value of γ is only a few percent lower). This suggests that some care needs to be taken in selecting the right value of γ, as the tree policy may overfit the available training data. One possibility, as alluded to earlier, is to use part of the training data for building the tree policy and to use the remainder as a validation set, and to select the value of γ that gives the highest objective on the validation set; then, the whole training set would be used to re-estimate the tree policy.

Table EC.1 **In-sample and out-of-sample rewards for tree policies as a function of γ for $n = 8$ assets and $\bar{p} = 90$ and common correlation $\bar{\rho} = 0$.**

State variables	γ	Training Obj.	Test. Obj.	Num. Iter.
PRICES	0.1000	20.38	19.52	3.2
	0.0500	33.37	32.66	10.8
	0.0100	35.48	34.62	14.2
	0.0050	36.74	35.60	19.7
	0.0010	39.93	37.56	53.0
	0.0005	40.89	37.48	83.8
	0.0001	42.60	36.60	239.0
PRICES, TIME	0.1000	27.68	27.10	2.0
	0.0500	28.17	27.52	2.4
	0.0100	36.10	35.47	12.5
	0.0050	39.76	38.89	23.8
	0.0010	44.81	43.22	59.1
	0.0005	45.65	43.53	82.5
	0.0001	46.96	42.36	209.4
PRICES, TIME,	0.1000	44.65	44.68	2.9
PAYOFF, KOIND	0.0500	44.81	44.84	3.0
	0.0100	45.40	45.35	4.6
	0.0050	45.47	45.38	5.0
	0.0010	45.64	45.41	7.5
	0.0005	45.78	45.39	11.6
	0.0001	46.05	45.31	36.2

EC.2.2. Performance results for $\bar{\rho} = 0$

Table EC.2 provides additional performance results in the $\bar{\rho} = 0$ case, analogous to those in Table 1, for $n = 4$ and $n = 16$.

Table EC.2 Comparison of out-of-sample performance between Longstaff-Schwartz and tree policies for $n = 4$ and $n = 16$ assets, for different initial prices $\bar{p}$ and common correlation $\bar{\rho} = 0$. In each column, the best performance is indicated in bold.

n	Method	State variables / Basis functions	Initial Price $\bar{p} = 90$	$\bar{p} = 100$	$\bar{p} = 110$
4	LS	one	24.68 (0.021)	31.77 (0.020)	37.47 (0.017)
4	LS	prices	25.76 (0.021)	32.06 (0.024)	37.40 (0.023)
4	LS	pricesKO	28.52 (0.018)	38.31 (0.027)	46.58 (0.027)
4	LS	pricesKO KOind	30.23 (0.016)	39.05 (0.021)	46.58 (0.023)
4	LS	pricesKO KOind payoff	32.73 (0.029)	41.22 (0.022)	47.75 (0.015)
4	LS	pricesKO KOind payoff maxpriceKO	33.04 (0.023)	41.31 (0.020)	47.76 (0.014)
4	LS	pricesKO KOind payoff maxpriceKO max2priceKO	32.98 (0.022)	41.35 (0.022)	47.81 (0.012)
4	LS	pricesKO payoff	33.47 (0.021)	41.70 (0.015)	47.72 (0.007)
4	LS	pricesKO prices2KO KOind payoff	33.43 (0.022)	41.81 (0.022)	48.03 (0.013)
4	PO	prices	31.43 (0.017)	38.91 (0.018)	43.40 (0.016)
4	PO	pricesKO KOind payoff	31.39 (0.037)	40.61 (0.044)	48.46 (0.015)
4	PO	pricesKO KOind payoff maxpriceKO max2priceKO	32.20 (0.032)	41.11 (0.037)	48.49 (0.015)
4	PO	pricesKO prices2KO KOind payoff	33.66 (0.026)	42.49 (0.013)	48.77 (0.013)
4	Tree	payoff time	**34.30 (0.028)**	**43.08 (0.022)**	**49.38 (0.019)**
4	Tree	prices	27.14 (0.048)	36.91 (0.021)	45.15 (0.043)
4	Tree	prices payoff	27.34 (0.022)	37.13 (0.015)	45.77 (0.014)
4	Tree	prices time	33.76 (0.126)	39.51 (0.470)	40.94 (0.370)
4	Tree	prices time payoff	**34.30 (0.028)**	**43.08 (0.022)**	**49.38 (0.019)**
4	Tree	prices time payoff KOind	**34.30 (0.028)**	**43.08 (0.022)**	**49.38 (0.019)**
16	LS	one	39.09 (0.021)	43.19 (0.016)	47.12 (0.025)
16	LS	prices	38.99 (0.022)	43.11 (0.015)	47.04 (0.027)
16	LS	pricesKO	50.34 (0.024)	53.40 (0.009)	54.68 (0.007)
16	LS	pricesKO KOind	50.41 (0.026)	53.72 (0.008)	54.97 (0.009)
16	LS	pricesKO KOind payoff	50.51 (0.020)	53.30 (0.009)	54.78 (0.012)
16	LS	pricesKO KOind payoff maxpriceKO	50.51 (0.020)	53.30 (0.009)	54.78 (0.012)
16	LS	pricesKO KOind payoff maxpriceKO max2priceKO	50.50 (0.020)	53.29 (0.009)	54.78 (0.013)
16	LS	pricesKO payoff	50.30 (0.018)	52.93 (0.012)	54.45 (0.012)
16	LS	pricesKO prices2KO KOind payoff	50.27 (0.016)	53.06 (0.011)	54.59 (0.012)
16	PO	prices	45.58 (0.025)	48.05 (0.019)	50.34 (0.017)
16	PO	pricesKO KOind payoff	51.26 (0.012)	53.92 (0.007)	55.28 (0.009)
16	PO	pricesKO KOind payoff maxpriceKO max2priceKO	51.22 (0.015)	53.92 (0.006)	55.27 (0.008)
16	Tree	payoff time	**51.85 (0.015)**	**54.62 (0.008)**	**56.00 (0.010)**
16	Tree	prices	39.70 (0.155)	42.60 (0.125)	43.91 (0.144)
16	Tree	prices payoff	49.35 (0.012)	54.17 (0.008)	55.96 (0.008)
16	Tree	prices time	39.51 (0.089)	43.10 (0.020)	46.17 (0.502)
16	Tree	prices time payoff	**51.85 (0.015)**	**54.62 (0.007)**	**56.00 (0.010)**
16	Tree	prices time payoff KOind	**51.85 (0.015)**	**54.62 (0.007)**	**56.00 (0.010)**

EC.2.3. Performance results for $\bar{\rho} \in \{-0.05, +0.05, +0.10, +0.20\}$

Tables EC.3, EC.4, EC.5 and EC.6 report the performance of the tree and Longstaff-Schwartz policies for $\bar{\rho}$ values of -0.05, $+0.05$, $+0.10$ and $+0.20$, respectively. The instantaneous correlation matrix of the asset price process is set so that $\rho_{ii} = 1$ and $\rho_{ij} = \bar{\rho}$ for $i \neq j$. The same experimental set-up as for $\bar{\rho} = 0$ is followed (the number of assets n varies in $\{4, 8, 16\}$, the initial price $\bar{p}$ varies in $\{90, 100, 110\}$, and each value reported for each method is averaged over ten replications).

Table EC.3 Comparison of out-of-sample performance between Longstaff-Schwartz and tree policies for $n \in \{4, 8, 16\}$ assets, for different initial prices $\bar{p}$ and common correlation $\bar{\rho} = -0.05$. In each column, the best performance is indicated in bold.

n	Method	State variables / Basis functions	Initial Price $\bar{p}=90$	$\bar{p}=100$	$\bar{p}=110$
4	LS	one	25.61 (0.021)	32.57 (0.010)	38.11 (0.014)
4	LS	prices	26.54 (0.019)	32.79 (0.016)	38.00 (0.016)
4	LS	pricesKO	29.48 (0.027)	39.39 (0.023)	47.48 (0.023)
4	LS	pricesKO KOind	31.07 (0.012)	40.04 (0.022)	47.48 (0.017)
4	LS	pricesKO KOind payoff	33.64 (0.022)	42.12 (0.026)	48.47 (0.014)
4	LS	pricesKO KOind payoff maxpriceKO	33.88 (0.024)	42.20 (0.026)	48.47 (0.016)
4	LS	pricesKO KOind payoff maxpriceKO max2priceKO	33.85 (0.024)	42.23 (0.021)	48.51 (0.013)
4	LS	pricesKO payoff	34.36 (0.026)	42.56 (0.020)	48.41 (0.013)
4	LS	pricesKO prices2KO KOind payoff	34.26 (0.022)	42.65 (0.018)	48.70 (0.019)
4	Tree	payoff time	**35.20 (0.064)**	**43.86 (0.022)**	**50.02 (0.020)**
4	Tree	prices	27.83 (0.024)	37.76 (0.026)	46.08 (0.037)
4	Tree	prices payoff	28.06 (0.028)	37.97 (0.026)	46.61 (0.019)
4	Tree	prices time	34.74 (0.062)	40.33 (0.341)	41.61 (0.347)
4	Tree	prices time payoff	**35.20 (0.064)**	**43.86 (0.021)**	**50.03 (0.020)**
4	Tree	prices time payoff KOind	**35.20 (0.064)**	**43.86 (0.021)**	**50.03 (0.020)**
8	LS	one	34.99 (0.018)	39.57 (0.014)	44.02 (0.017)
8	LS	prices	34.95 (0.021)	39.44 (0.016)	43.92 (0.021)
8	LS	pricesKO	43.00 (0.021)	50.51 (0.017)	53.76 (0.012)
8	LS	pricesKO KOind	43.40 (0.013)	50.55 (0.016)	54.03 (0.009)
8	LS	pricesKO KOind payoff	45.15 (0.017)	50.76 (0.018)	53.58 (0.006)
8	LS	pricesKO KOind payoff maxpriceKO	45.15 (0.017)	50.75 (0.018)	53.58 (0.006)
8	LS	pricesKO KOind payoff maxpriceKO max2priceKO	45.17 (0.016)	50.74 (0.019)	53.57 (0.005)
8	LS	pricesKO payoff	45.35 (0.012)	50.56 (0.017)	53.25 (0.008)
8	LS	pricesKO prices2KO KOind payoff	45.27 (0.014)	50.77 (0.018)	53.58 (0.009)
8	Tree	payoff time	**46.58 (0.020)**	**52.11 (0.024)**	**54.92 (0.011)**
8	Tree	prices	36.73 (0.101)	44.54 (0.091)	47.83 (0.089)
8	Tree	prices payoff	40.56 (0.018)	49.66 (0.020)	54.35 (0.010)
8	Tree	prices time	39.13 (0.196)	40.81 (0.503)	43.41 (0.123)
8	Tree	prices time payoff	**46.58 (0.020)**	**52.11 (0.024)**	**54.92 (0.010)**
8	Tree	prices time payoff KOind	**46.58 (0.020)**	**52.11 (0.024)**	**54.92 (0.010)**
16	LS	one	40.34 (0.021)	44.27 (0.035)	48.10 (0.022)
16	LS	payoff time	43.78 (0.035)	45.83 (0.045)	48.19 (0.018)
16	LS	prices	40.25 (0.022)	44.21 (0.036)	48.05 (0.020)
16	LS	pricesKO	51.78 (0.015)	54.13 (0.007)	55.06 (0.008)
16	LS	pricesKO KOind	51.76 (0.012)	54.16 (0.009)	55.16 (0.011)
16	LS	pricesKO KOind payoff	51.56 (0.013)	53.78 (0.010)	55.12 (0.008)
16	LS	pricesKO KOind payoff maxpriceKO	51.56 (0.013)	53.78 (0.010)	55.12 (0.008)
16	LS	pricesKO KOind payoff maxpriceKO max2priceKO	51.54 (0.013)	53.76 (0.009)	55.11 (0.010)
16	LS	pricesKO payoff	51.51 (0.014)	53.65 (0.010)	55.01 (0.010)
16	LS	pricesKO prices2KO KOind payoff	51.30 (0.015)	53.53 (0.015)	54.93 (0.008)
16	Tree	payoff time	**52.75 (0.011)**	**54.94 (0.006)**	**56.14 (0.009)**
16	Tree	prices	40.81 (0.089)	43.11 (0.167)	44.06 (0.118)
16	Tree	prices payoff	51.09 (0.011)	54.81 (0.007)	56.13 (0.009)
16	Tree	prices time	41.16 (0.076)	44.23 (0.095)	47.28 (0.509)
16	Tree	prices time payoff	52.74 (0.011)	**54.94 (0.006)**	56.13 (0.009)
16	Tree	prices time payoff KOind	52.74 (0.011)	**54.94 (0.006)**	56.13 (0.009)

Table EC.4 Comparison of out-of-sample performance between Longstaff-Schwartz and tree policies for $n \in \{4, 8, 16\}$ assets, for different initial prices $\bar{p}$ and common correlation $\bar{\rho} = +0.05$. In each column, the best performance is indicated in bold.

n	Method	State variables / Basis functions	Initial Price $\bar{p}=90$	$\bar{p}=100$	$\bar{p}=110$
4	LS	one	23.84 (0.022)	31.00 (0.020)	36.77 (0.013)
4	LS	payoff time	32.07 (0.031)	39.32 (0.015)	42.96 (0.073)
4	LS	prices	25.00 (0.023)	31.37 (0.019)	36.72 (0.012)
4	LS	pricesKO	27.68 (0.029)	37.42 (0.033)	45.68 (0.029)
4	LS	pricesKO KOind	29.50 (0.023)	38.21 (0.021)	45.73 (0.026)
4	LS	pricesKO KOind payoff	31.93 (0.028)	40.41 (0.028)	47.03 (0.019)
4	LS	pricesKO KOind payoff maxpriceKO	32.30 (0.027)	40.55 (0.029)	47.04 (0.018)
4	LS	pricesKO KOind payoff maxpriceKO max2priceKO	32.28 (0.027)	40.59 (0.029)	47.08 (0.017)
4	LS	pricesKO payoff	32.69 (0.024)	40.88 (0.020)	47.03 (0.014)
4	LS	pricesKO prices2KO KOind payoff	32.71 (0.024)	41.01 (0.023)	47.34 (0.008)
4	Tree	payoff time	**33.68 (0.048)**	**42.26 (0.033)**	**48.70 (0.024)**
4	Tree	prices	26.47 (0.021)	36.00 (0.044)	44.29 (0.054)
4	Tree	prices payoff	26.74 (0.023)	36.29 (0.027)	44.95 (0.019)
4	Tree	prices time	32.97 (0.088)	38.90 (0.423)	40.48 (0.287)
4	Tree	prices time payoff	**33.68 (0.048)**	**42.26 (0.033)**	**48.70 (0.024)**
4	Tree	prices time payoff KOind	**33.68 (0.048)**	**42.26 (0.033)**	**48.70 (0.024)**
8	LS	one	32.65 (0.022)	37.73 (0.020)	42.31 (0.018)
8	LS	payoff time	40.40 (0.044)	42.95 (0.030)	44.46 (0.025)
8	LS	prices	32.81 (0.023)	37.61 (0.019)	42.19 (0.019)
8	LS	pricesKO	39.95 (0.034)	48.18 (0.017)	52.40 (0.009)
8	LS	pricesKO KOind	40.40 (0.027)	48.15 (0.022)	52.74 (0.017)
8	LS	pricesKO KOind payoff	42.48 (0.033)	48.93 (0.015)	52.53 (0.014)
8	LS	pricesKO KOind payoff maxpriceKO	42.57 (0.032)	48.93 (0.016)	52.52 (0.014)
8	LS	pricesKO KOind payoff maxpriceKO max2priceKO	42.60 (0.033)	48.96 (0.013)	52.52 (0.015)
8	LS	pricesKO payoff	42.79 (0.027)	48.69 (0.013)	52.06 (0.012)
8	LS	pricesKO prices2KO KOind payoff	42.88 (0.030)	49.07 (0.011)	52.61 (0.014)
8	Tree	payoff time	**44.24 (0.021)**	**50.43 (0.008)**	**54.08 (0.009)**
8	Tree	prices	34.17 (0.217)	42.65 (0.105)	46.41 (0.104)
8	Tree	prices payoff	37.75 (0.025)	47.08 (0.023)	52.84 (0.012)
8	Tree	prices time	37.74 (0.259)	40.02 (0.337)	41.81 (0.109)
8	Tree	prices time payoff	**44.24 (0.022)**	**50.43 (0.008)**	**54.08 (0.010)**
8	Tree	prices time payoff KOind	**44.24 (0.022)**	**50.43 (0.008)**	**54.08 (0.010)**
16	LS	one	37.96 (0.023)	42.12 (0.023)	46.19 (0.019)
16	LS	payoff time	42.70 (0.041)	44.48 (0.020)	46.73 (0.031)
16	LS	prices	37.84 (0.026)	42.01 (0.029)	46.09 (0.019)
16	LS	pricesKO	48.95 (0.011)	52.60 (0.010)	54.33 (0.011)
16	LS	pricesKO KOind	49.00 (0.015)	53.08 (0.011)	54.72 (0.010)
16	LS	pricesKO KOind payoff	49.44 (0.014)	52.73 (0.009)	54.45 (0.012)
16	LS	pricesKO KOind payoff maxpriceKO	49.44 (0.014)	52.73 (0.009)	54.45 (0.012)
16	LS	pricesKO KOind payoff maxpriceKO max2priceKO	49.44 (0.015)	52.72 (0.008)	54.44 (0.011)
16	LS	pricesKO payoff	49.14 (0.010)	52.21 (0.013)	53.96 (0.013)
16	LS	pricesKO prices2KO KOind payoff	49.26 (0.012)	52.51 (0.011)	54.29 (0.012)
16	Tree	payoff time	**50.86 (0.019)**	**54.17 (0.007)**	**55.86 (0.010)**
16	Tree	prices	38.76 (0.147)	41.83 (0.114)	43.49 (0.140)
16	Tree	prices payoff	47.62 (0.021)	53.29 (0.012)	55.73 (0.012)
16	Tree	prices time	38.74 (0.151)	42.07 (0.028)	44.82 (0.642)
16	Tree	prices time payoff	**50.86 (0.019)**	**54.17 (0.007)**	**55.86 (0.010)**
16	Tree	prices time payoff KOind	**50.86 (0.019)**	**54.17 (0.007)**	**55.86 (0.010)**

Table EC.5 Comparison of out-of-sample performance between Longstaff-Schwartz and tree policies for $n \in \{4, 8, 16\}$ assets, for different initial prices $\bar{p}$ and common correlation $\bar{\rho} = +0.10$. In each column, the best performance is indicated in bold.

n	Method	State variables / Basis functions	Initial Price $\bar{p} = 90$	$\bar{p} = 100$	$\bar{p} = 110$
4	LS	one	23.03 (0.023)	30.22 (0.018)	36.13 (0.024)
4	LS	payoff time	31.24 (0.019)	38.71 (0.028)	42.70 (0.058)
4	LS	prices	24.30 (0.022)	30.70 (0.018)	36.14 (0.019)
4	LS	pricesKO	26.88 (0.026)	36.43 (0.030)	44.74 (0.016)
4	LS	pricesKO KOind	28.77 (0.026)	37.36 (0.019)	44.85 (0.019)
4	LS	pricesKO KOind payoff	31.07 (0.020)	39.58 (0.019)	46.29 (0.015)
4	LS	pricesKO KOind payoff maxpriceKO	31.51 (0.019)	39.77 (0.019)	46.32 (0.015)
4	LS	pricesKO KOind payoff maxpriceKO max2priceKO	31.49 (0.026)	39.79 (0.019)	46.36 (0.021)
4	LS	pricesKO payoff	31.89 (0.014)	40.10 (0.020)	46.36 (0.013)
4	LS	pricesKO prices2KO KOind payoff	31.92 (0.018)	40.26 (0.017)	46.64 (0.016)
4	Tree	payoff time	**32.81 (0.030)**	**41.53 (0.023)**	**47.99 (0.020)**
4	Tree	prices	25.83 (0.021)	35.22 (0.029)	43.39 (0.042)
4	Tree	prices payoff	26.07 (0.014)	35.53 (0.022)	44.11 (0.029)
4	Tree	prices time	32.37 (0.112)	38.31 (0.322)	39.90 (0.253)
4	Tree	prices time payoff	**32.81 (0.030)**	41.52 (0.021)	**47.99 (0.020)**
4	Tree	prices time payoff KOind	**32.81 (0.030)**	41.52 (0.021)	**47.99 (0.020)**
8	LS	one	31.46 (0.020)	36.76 (0.022)	41.48 (0.018)
8	LS	payoff time	39.52 (0.042)	42.58 (0.035)	44.00 (0.034)
8	LS	prices	31.75 (0.015)	36.68 (0.021)	41.36 (0.023)
8	LS	pricesKO	38.45 (0.035)	46.95 (0.017)	51.68 (0.015)
8	LS	pricesKO KOind	38.97 (0.026)	46.85 (0.018)	52.00 (0.022)
8	LS	pricesKO KOind payoff	41.16 (0.027)	47.93 (0.020)	51.95 (0.017)
8	LS	pricesKO KOind payoff maxpriceKO	41.29 (0.025)	47.94 (0.021)	51.94 (0.016)
8	LS	pricesKO KOind payoff maxpriceKO max2priceKO	41.33 (0.024)	47.98 (0.019)	51.95 (0.014)
8	LS	pricesKO payoff	41.52 (0.017)	47.74 (0.017)	51.46 (0.013)
8	LS	pricesKO prices2KO KOind payoff	41.67 (0.021)	48.15 (0.010)	52.06 (0.016)
8	Tree	payoff time	**43.09 (0.022)**	**49.58 (0.024)**	**53.53 (0.017)**
8	Tree	prices	33.80 (0.196)	41.40 (0.094)	45.77 (0.117)
8	Tree	prices payoff	36.46 (0.015)	45.85 (0.022)	51.98 (0.011)
8	Tree	prices time	36.55 (0.218)	39.51 (0.238)	41.23 (0.139)
8	Tree	prices time payoff	**43.09 (0.022)**	**49.58 (0.024)**	**53.53 (0.017)**
8	Tree	prices time payoff KOind	**43.09 (0.022)**	**49.58 (0.024)**	**53.53 (0.017)**
16	LS	one	36.74 (0.017)	41.06 (0.016)	45.25 (0.020)
16	LS	payoff time	42.25 (0.032)	43.85 (0.032)	46.07 (0.024)
16	LS	prices	36.62 (0.019)	40.93 (0.019)	45.12 (0.024)
16	LS	pricesKO	47.47 (0.015)	51.77 (0.013)	53.94 (0.015)
16	LS	pricesKO KOind	47.42 (0.017)	52.28 (0.017)	54.40 (0.016)
16	LS	pricesKO KOind payoff	48.22 (0.016)	52.06 (0.016)	54.09 (0.011)
16	LS	pricesKO KOind payoff maxpriceKO	48.23 (0.016)	52.06 (0.016)	54.09 (0.012)
16	LS	pricesKO KOind payoff maxpriceKO max2priceKO	48.24 (0.015)	52.05 (0.016)	54.09 (0.012)
16	LS	pricesKO payoff	47.93 (0.013)	51.47 (0.011)	53.53 (0.013)
16	LS	pricesKO prices2KO KOind payoff	48.11 (0.012)	51.89 (0.017)	53.96 (0.013)
16	Tree	payoff time	**49.78 (0.021)**	**53.61 (0.009)**	**55.63 (0.013)**
16	Tree	prices	37.11 (0.103)	41.12 (0.083)	43.10 (0.092)
16	Tree	prices payoff	45.92 (0.023)	52.31 (0.015)	55.35 (0.012)
16	Tree	prices time	37.54 (0.111)	41.14 (0.076)	44.45 (0.270)
16	Tree	prices time payoff	**49.78 (0.021)**	**53.61 (0.009)**	**55.63 (0.013)**
16	Tree	prices time payoff KOind	**49.78 (0.021)**	**53.61 (0.009)**	**55.63 (0.013)**

Table EC.6 Comparison of out-of-sample performance between Longstaff-Schwartz and tree policies for $n \in \{4, 8, 16\}$ assets, for different initial prices $\bar{p}$ and common correlation $\bar{\rho} = +0.20$. In each column, the best performance is indicated in bold.

n	Method	State variables / Basis functions	Initial Price $\bar{p}=90$	$\bar{p}=100$	$\bar{p}=110$
4	LS	one	21.41 (0.025)	28.60 (0.023)	34.77 (0.018)
4	LS	payoff time	29.70 (0.026)	37.27 (0.034)	42.07 (0.038)
4	LS	prices	22.85 (0.024)	29.28 (0.027)	34.84 (0.017)
4	LS	pricesKO	25.22 (0.031)	34.46 (0.026)	42.83 (0.019)
4	LS	pricesKO KOind	27.36 (0.018)	35.66 (0.016)	43.17 (0.023)
4	LS	pricesKO KOind payoff	29.48 (0.027)	37.80 (0.017)	44.79 (0.023)
4	LS	pricesKO KOind payoff maxpriceKO	30.04 (0.023)	38.09 (0.016)	44.85 (0.022)
4	LS	pricesKO KOind payoff maxpriceKO max2priceKO	30.02 (0.026)	38.12 (0.018)	44.90 (0.020)
4	LS	pricesKO payoff	30.34 (0.017)	38.42 (0.021)	44.95 (0.014)
4	LS	pricesKO prices2KO KOind payoff	30.42 (0.024)	38.59 (0.021)	45.16 (0.025)
4	Tree	payoff time	**31.40 (0.047)**	**39.90 (0.025)**	**46.58 (0.014)**
4	Tree	prices	24.59 (0.039)	33.47 (0.030)	41.66 (0.039)
4	Tree	prices payoff	24.79 (0.014)	33.86 (0.024)	42.44 (0.014)
4	Tree	prices time	30.82 (0.124)	37.06 (0.227)	38.58 (0.188)
4	Tree	prices time payoff	**31.40 (0.047)**	**39.90 (0.025)**	**46.58 (0.014)**
4	Tree	prices time payoff KOind	**31.40 (0.047)**	**39.90 (0.025)**	**46.58 (0.014)**
8	LS	one	29.13 (0.021)	34.94 (0.028)	39.94 (0.013)
8	LS	payoff time	37.69 (0.024)	41.99 (0.039)	43.32 (0.031)
8	LS	prices	29.73 (0.026)	34.94 (0.033)	39.81 (0.016)
8	LS	pricesKO	35.56 (0.025)	44.48 (0.028)	50.16 (0.014)
8	LS	pricesKO KOind	36.38 (0.027)	44.42 (0.032)	50.30 (0.020)
8	LS	pricesKO KOind payoff	38.65 (0.030)	45.92 (0.017)	50.64 (0.019)
8	LS	pricesKO KOind payoff maxpriceKO	38.93 (0.031)	45.96 (0.014)	50.63 (0.018)
8	LS	pricesKO KOind payoff maxpriceKO max2priceKO	38.96 (0.026)	46.01 (0.015)	50.65 (0.017)
8	LS	pricesKO payoff	39.13 (0.026)	45.83 (0.014)	50.19 (0.013)
8	LS	pricesKO prices2KO KOind payoff	39.37 (0.025)	46.27 (0.011)	50.78 (0.016)
8	Tree	payoff time	**40.73 (0.014)**	**47.74 (0.025)**	**52.30 (0.013)**
8	Tree	prices	32.11 (0.170)	39.28 (0.318)	44.34 (0.177)
8	Tree	prices payoff	33.98 (0.033)	43.36 (0.021)	50.16 (0.015)
8	Tree	prices time	36.14 (0.172)	37.78 (0.334)	40.17 (0.231)
8	Tree	prices time payoff	**40.73 (0.014)**	**47.74 (0.025)**	**52.30 (0.014)**
8	Tree	prices time payoff KOind	**40.73 (0.014)**	**47.74 (0.025)**	**52.30 (0.014)**
16	LS	one	34.45 (0.020)	39.00 (0.024)	43.47 (0.017)
16	LS	payoff time	41.39 (0.042)	42.94 (0.037)	44.84 (0.025)
16	LS	prices	34.41 (0.017)	38.82 (0.025)	43.30 (0.019)
16	LS	pricesKO	44.41 (0.027)	50.02 (0.010)	52.94 (0.016)
16	LS	pricesKO KOind	44.24 (0.024)	50.35 (0.019)	53.47 (0.015)
16	LS	pricesKO KOind payoff	45.64 (0.021)	50.54 (0.012)	53.21 (0.013)
16	LS	pricesKO KOind payoff maxpriceKO	45.72 (0.024)	50.54 (0.012)	53.20 (0.014)
16	LS	pricesKO KOind payoff maxpriceKO max2priceKO	45.76 (0.020)	50.55 (0.012)	53.20 (0.015)
16	LS	pricesKO payoff	45.48 (0.017)	49.95 (0.011)	52.59 (0.011)
16	LS	pricesKO prices2KO KOind payoff	45.73 (0.019)	50.44 (0.011)	53.17 (0.013)
16	Tree	payoff time	**47.50 (0.026)**	**52.25 (0.012)**	**54.91 (0.010)**
16	Tree	prices	34.95 (0.128)	39.89 (0.127)	42.47 (0.070)
16	Tree	prices payoff	42.62 (0.018)	50.05 (0.022)	54.19 (0.014)
16	Tree	prices time	35.21 (0.122)	38.94 (0.120)	41.96 (0.199)
16	Tree	prices time payoff	**47.50 (0.026)**	**52.25 (0.012)**	54.90 (0.010)
16	Tree	prices time payoff KOind	**47.50 (0.026)**	**52.25 (0.012)**	54.90 (0.010)

EC.2.4. Timing results for $n \in \{4, 16\}$ and $\bar{\rho} = 0$

Table EC.7 reports the computation time for the tree and Longstaff-Schwartz policies for $n \in \{4, 16\}$ and $\bar{p} \in \{90, 100, 110\}$, for the uncorrelated ($\bar{\rho} = 0$) case. The computation times are averaged over the ten replications for each combination of n and $\bar{p}$.

Table EC.7 Comparison of estimation time between Longstaff-Schwartz, pathwise optimization and tree policies for $n \in \{4, 16\}$ assets, for different initial prices $\bar{p}$ and common correlation $\bar{\rho} = 0$.

n	Method	State variables / Basis functions	Initial Price $\bar{p}=90$	$\bar{p}=100$	$\bar{p}=110$
4	LS	one	1.3 (0.1)	1.3 (0.1)	1.3 (0.0)
4	LS	prices	1.3 (0.1)	1.2 (0.1)	1.2 (0.0)
4	LS	pricesKO	1.6 (0.1)	1.6 (0.1)	1.4 (0.1)
4	LS	pricesKO KOind	1.7 (0.2)	1.9 (0.2)	1.5 (0.2)
4	LS	pricesKO KOind payoff	1.7 (0.1)	1.7 (0.1)	1.5 (0.2)
4	LS	pricesKO KOind payoff maxpriceKO	2.0 (0.1)	2.2 (0.2)	2.2 (0.2)
4	LS	pricesKO KOind payoff maxpriceKO max2priceKO	2.0 (0.2)	1.9 (0.1)	1.9 (0.2)
4	LS	pricesKO payoff	2.0 (0.1)	1.9 (0.1)	1.5 (0.1)
4	LS	pricesKO prices2KO KOind payoff	3.2 (0.3)	4.0 (0.3)	3.2 (0.2)
4	PO	prices	25.9 (0.8)	27.6 (0.9)	27.4 (1.1)
4	PO	pricesKO KOind payoff	65.0 (5.0)	79.7 (9.0)	60.9 (3.7)
4	PO	pricesKO KOind payoff maxpriceKO max2priceKO	86.0 (4.3)	89.9 (4.9)	82.3 (4.2)
4	PO	pricesKO prices2KO KOind payoff	165.9 (10.1)	157.5 (7.2)	125.8 (4.4)
4	Tree	payoff time	10.8 (1.5)	6.4 (0.6)	4.9 (0.4)
4	Tree	prices	22.3 (2.1)	39.2 (4.6)	30.8 (2.4)
4	Tree	prices payoff	4.3 (0.5)	3.1 (0.3)	4.0 (0.4)
4	Tree	prices time	96.1 (7.3)	141.8 (21.9)	75.9 (10.6)
4	Tree	prices time payoff	18.6 (2.8)	11.9 (1.5)	8.4 (0.6)
4	Tree	prices time payoff KOind	19.6 (2.0)	11.5 (0.8)	9.4 (1.1)
16	LS	one	1.1 (0.0)	1.1 (0.0)	1.1 (0.0)
16	LS	prices	2.1 (0.0)	2.1 (0.0)	2.0 (0.0)
16	LS	pricesKO	2.4 (0.2)	2.1 (0.2)	2.3 (0.2)
16	LS	pricesKO KOind	2.2 (0.1)	2.4 (0.2)	2.1 (0.1)
16	LS	pricesKO KOind payoff	2.2 (0.1)	1.9 (0.1)	2.3 (0.2)
16	LS	pricesKO KOind payoff maxpriceKO	2.3 (0.2)	2.2 (0.1)	1.9 (0.1)
16	LS	pricesKO KOind payoff maxpriceKO max2priceKO	2.2 (0.2)	2.1 (0.3)	2.1 (0.1)
16	LS	pricesKO payoff	1.9 (0.2)	2.0 (0.2)	2.1 (0.2)
16	LS	pricesKO prices2KO KOind payoff	36.0 (1.8)	35.7 (1.8)	33.5 (1.0)
16	PO	prices	59.4 (0.9)	59.4 (1.7)	53.5 (1.3)
16	PO	pricesKO KOind payoff	114.7 (6.0)	85.6 (4.3)	60.6 (2.0)
16	PO	pricesKO KOind payoff maxpriceKO max2priceKO	96.6 (2.9)	85.2 (3.2)	67.3 (2.9)
16	Tree	payoff time	4.0 (0.1)	3.5 (0.1)	2.2 (0.1)
16	Tree	prices	281.1 (11.3)	260.8 (8.4)	202.9 (4.1)
16	Tree	prices payoff	10.0 (0.1)	9.4 (0.2)	9.1 (0.2)
16	Tree	prices time	195.0 (2.2)	175.3 (3.2)	150.8 (13.6)
16	Tree	prices time payoff	17.5 (0.2)	16.0 (0.3)	10.0 (0.2)
16	Tree	prices time payoff KOind	17.8 (0.2)	16.0 (0.2)	10.3 (0.1)

EC.2.5. Example of LS policy

Table EC.8 below provides an example of a policy produced by LS (namely, the regression coefficients for predicting the continuation value at each t) for the basis function architecture consisting of pricesKO, KOind and payoff.

t	$r_{p_1(t)y(t)}$	$r_{p_2(t)y(t)}$	$r_{p_3(t)y(t)}$	$r_{p_4(t)y(t)}$	$r_{p_5(t)y(t)}$	$r_{p_6(t)y(t)}$	$r_{p_7(t)y(t)}$	$r_{p_8(t)y(t)}$	$r_{y(t)}$	$r_{g(t)}$
1	0.187	0.157	0.093	0.141	0.153	0.096	0.153	0.116	-53.234	1.244
2	0.160	0.137	0.117	0.105	0.137	0.103	0.138	0.108	-46.019	0.938
3	0.166	0.136	0.122	0.117	0.139	0.091	0.151	0.122	-49.177	0.253
4	0.163	0.126	0.130	0.113	0.137	0.091	0.151	0.132	-49.359	0.136
5	0.148	0.127	0.122	0.114	0.121	0.092	0.140	0.134	-45.788	0.223
6	0.145	0.122	0.129	0.124	0.118	0.088	0.146	0.132	-46.506	0.183
7	0.137	0.118	0.126	0.121	0.119	0.095	0.139	0.130	-45.206	0.188
8	0.131	0.114	0.123	0.117	0.115	0.096	0.132	0.130	-43.315	0.189
9	0.128	0.108	0.125	0.114	0.111	0.100	0.128	0.125	-42.122	0.178
10	0.130	0.106	0.125	0.110	0.111	0.098	0.127	0.124	-41.591	0.169
11	0.126	0.101	0.121	0.114	0.109	0.094	0.126	0.123	-40.556	0.178
12	0.118	0.096	0.121	0.108	0.108	0.095	0.121	0.114	-38.273	0.201
13	0.120	0.093	0.119	0.107	0.107	0.094	0.120	0.112	-37.695	0.208
14	0.111	0.094	0.117	0.103	0.102	0.095	0.114	0.112	-36.255	0.222
15	0.110	0.095	0.109	0.098	0.098	0.090	0.108	0.109	-34.205	0.241
16	0.107	0.092	0.106	0.097	0.096	0.089	0.107	0.107	-33.227	0.248
17	0.109	0.086	0.106	0.095	0.089	0.087	0.097	0.103	-31.397	0.267
18	0.109	0.086	0.107	0.094	0.084	0.085	0.098	0.100	-30.839	0.264
19	0.108	0.082	0.105	0.095	0.083	0.085	0.094	0.097	-29.996	0.263
20	0.102	0.075	0.101	0.098	0.082	0.084	0.092	0.096	-28.817	0.272
21	0.101	0.074	0.092	0.093	0.077	0.077	0.091	0.095	-27.092	0.291
22	0.099	0.068	0.084	0.098	0.077	0.077	0.087	0.093	-26.011	0.296
23	0.091	0.069	0.077	0.094	0.071	0.076	0.083	0.089	-23.579	0.308
24	0.088	0.062	0.083	0.090	0.066	0.073	0.072	0.087	-21.895	0.333
25	0.086	0.067	0.078	0.089	0.070	0.071	0.075	0.083	-22.130	0.332
26	0.079	0.068	0.073	0.091	0.060	0.072	0.076	0.087	-21.551	0.339
27	0.082	0.065	0.071	0.086	0.060	0.061	0.076	0.081	-19.951	0.340
28	0.079	0.064	0.069	0.081	0.056	0.060	0.076	0.076	-19.046	0.373
29	0.078	0.062	0.074	0.084	0.056	0.064	0.073	0.078	-19.757	0.354
30	0.074	0.056	0.069	0.079	0.060	0.060	0.067	0.074	-18.156	0.376
31	0.074	0.056	0.066	0.081	0.055	0.058	0.069	0.072	-18.033	0.384
32	0.068	0.051	0.059	0.075	0.056	0.059	0.069	0.065	-16.544	0.414
33	0.070	0.051	0.067	0.072	0.055	0.059	0.070	0.067	-17.626	0.402
34	0.066	0.052	0.064	0.075	0.047	0.059	0.064	0.064	-16.432	0.408
35	0.068	0.045	0.059	0.071	0.049	0.051	0.059	0.056	-14.880	0.447
36	0.062	0.047	0.059	0.066	0.058	0.057	0.056	0.058	-15.804	0.451
37	0.066	0.047	0.057	0.063	0.054	0.056	0.055	0.055	-15.697	0.457
38	0.062	0.044	0.059	0.064	0.057	0.052	0.056	0.060	-16.549	0.467
39	0.061	0.047	0.056	0.059	0.052	0.045	0.056	0.059	-16.029	0.499
40	0.063	0.042	0.056	0.063	0.047	0.044	0.052	0.062	-16.101	0.502
41	0.059	0.045	0.048	0.058	0.049	0.048	0.053	0.054	-15.502	0.511
42	0.056	0.041	0.048	0.051	0.050	0.053	0.046	0.048	-15.099	0.545
43	0.052	0.042	0.045	0.058	0.050	0.052	0.045	0.050	-15.850	0.550
44	0.053	0.039	0.046	0.050	0.049	0.041	0.047	0.037	-14.233	0.579
45	0.046	0.038	0.039	0.042	0.041	0.039	0.033	0.040	-11.716	0.603
46	0.050	0.032	0.043	0.042	0.045	0.037	0.039	0.041	-13.478	0.613
47	0.039	0.034	0.039	0.039	0.041	0.038	0.036	0.036	-12.441	0.639
48	0.040	0.034	0.036	0.036	0.027	0.034	0.026	0.023	-9.532	0.652
49	0.031	0.025	0.029	0.028	0.033	0.040	0.033	0.029	-9.949	0.672
50	0.023	0.018	0.027	0.026	0.027	0.027	0.024	0.031	-7.687	0.716
51	0.027	0.019	0.019	0.027	0.032	0.029	0.023	0.025	-8.271	0.712
52	0.008	0.017	0.015	0.016	0.023	0.019	0.021	0.016	-4.681	0.774
53	0.011	0.014	0.019	0.011	0.008	0.014	0.009	0.016	-3.458	0.808
54	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table EC.8 Example of LS regression coefficients for $n = 8$, with pricesKO, KOind and payoff basis functions.

EC.2.6. Additional S&P500 results

In this section, we provide additional results for our experiments calibrated with real S&P500 stock data. Table EC.9 and EC.10 report the results for interest rates of $r = 0.10$ and $r = 0.02$, respectively. Table EC.11 reports the stocks comprising each of the ten instances in the two time periods.

Period	Instance	LS - 1st O.	LS - 2nd O.	PO - 1st O.	PO - 2nd O.	Tree
2007 – 2017	1	87.87 (0.058)	89.99 (0.048)	84.33 (0.087)	91.39 (0.038)	**92.68 (0.060)**
	2	89.52 (0.063)	91.50 (0.034)	85.85 (0.068)	92.87 (0.045)	**94.21 (0.077)**
	3	92.65 (0.062)	95.59 (0.056)	89.31 (0.105)	97.01 (0.051)	**98.61 (0.051)**
	4	77.20 (0.035)	79.12 (0.054)	73.92 (0.093)	80.37 (0.030)	**81.33 (0.073)**
	5	88.17 (0.049)	90.20 (0.056)	84.34 (0.049)	91.48 (0.039)	**91.85 (0.095)**
	6	90.01 (0.071)	92.43 (0.049)	85.08 (0.099)	93.97 (0.069)	**95.55 (0.062)**
	7	73.27 (0.057)	74.95 (0.053)	69.56 (0.070)	76.14 (0.054)	**76.91 (0.076)**
	8	90.42 (0.038)	92.14 (0.046)	87.72 (0.077)	93.40 (0.047)	**94.42 (0.058)**
	9	77.12 (0.079)	79.01 (0.080)	72.63 (0.055)	**80.19 (0.083)**	80.11 (0.101)
	10	80.10 (0.060)	81.86 (0.048)	76.03 (0.081)	83.02 (0.046)	**83.34 (0.146)**
2017	1	135.95 (0.022)	138.82 (0.029)	140.43 (0.018)	141.39 (0.018)	**144.86 (0.044)**
	2	128.69 (0.052)	131.57 (0.040)	132.26 (0.080)	133.83 (0.040)	**136.59 (0.165)**
	3	133.78 (0.035)	136.62 (0.023)	138.00 (0.035)	139.35 (0.016)	**142.36 (0.133)**
	4	140.97 (0.037)	146.22 (0.037)	144.96 (0.024)	147.30 (0.032)	**150.69 (0.028)**
	5	132.29 (0.062)	135.41 (0.031)	136.24 (0.031)	136.92 (0.026)	**140.12 (0.030)**
	6	119.90 (0.044)	123.82 (0.030)	120.63 (0.051)	125.35 (0.035)	**125.36 (0.184)**
	7	145.09 (0.033)	152.09 (0.023)	150.09 (0.021)	153.73 (0.020)	**159.95 (0.013)**
	8	143.96 (0.024)	147.53 (0.042)	147.79 (0.018)	149.31 (0.035)	**153.60 (0.025)**
	9	137.66 (0.049)	140.07 (0.038)	140.75 (0.042)	141.49 (0.039)	**143.30 (0.064)**
	10	129.91 (0.057)	132.76 (0.037)	133.75 (0.033)	134.91 (0.020)	**137.70 (0.138)**

Table EC.9 Comparison of LS, PO and tree policies using realistically calibrated max-call option instances, with interest rate $r = 0.10$. Values shown are averages over ten replications, with standard errors in parentheses; bold is used to indicate the best method for each instance in each period. The terms "1st. O." and "2nd. O." indicate the first-order basis function architecture (pricesKO, payoff, KOind) and the second-order basis function architecture (pricesKO, prices2KO, payoff, KOind).

Period	Instance	LS - 1st O.	LS - 2nd O.	PO - 1st O.	PO - 2nd O.	Tree
2007 – 2017	1	96.50 (0.067)	99.39 (0.040)	90.81 (0.102)	100.90 (0.042)	**102.50 (0.062)**
	2	99.14 (0.088)	102.00 (0.106)	93.15 (0.097)	103.51 (0.112)	**105.14 (0.142)**
	3	102.54 (0.077)	106.33 (0.076)	96.74 (0.084)	107.68 (0.061)	**109.62 (0.080)**
	4	84.81 (0.071)	87.66 (0.064)	79.61 (0.076)	89.03 (0.054)	**90.21 (0.108)**
	5	97.69 (0.058)	100.65 (0.041)	91.79 (0.070)	101.96 (0.044)	**102.42 (0.109)**
	6	99.87 (0.080)	103.21 (0.050)	92.30 (0.121)	104.82 (0.035)	**106.87 (0.060)**
	7	82.32 (0.052)	84.89 (0.065)	76.24 (0.097)	86.08 (0.053)	**87.20 (0.100)**
	8	100.01 (0.057)	102.42 (0.039)	95.45 (0.104)	103.66 (0.063)	**104.99 (0.096)**
	9	86.68 (0.059)	89.53 (0.035)	79.76 (0.074)	90.68 (0.036)	**91.45 (0.130)**
	10	90.14 (0.100)	92.71 (0.066)	83.74 (0.076)	93.82 (0.047)	**94.85 (0.091)**
2017	1	153.23 (0.065)	156.41 (0.048)	158.24 (0.037)	159.25 (0.034)	**162.94 (0.065)**
	2	145.61 (0.046)	149.02 (0.039)	149.40 (0.062)	151.48 (0.033)	**154.48 (0.210)**
	3	151.82 (0.055)	155.12 (0.024)	156.62 (0.041)	158.22 (0.030)	**161.70 (0.218)**
	4	151.60 (0.047)	156.91 (0.033)	155.65 (0.035)	158.02 (0.047)	**161.55 (0.037)**
	5	147.47 (0.034)	150.76 (0.029)	151.75 (0.030)	152.28 (0.039)	**155.68 (0.052)**
	6	136.63 (0.064)	141.47 (0.065)	136.42 (0.094)	143.17 (0.063)	**143.85 (0.123)**
	7	159.41 (0.040)	167.23 (0.025)	165.02 (0.027)	169.07 (0.026)	**175.90 (0.026)**
	8	155.62 (0.041)	159.29 (0.040)	159.67 (0.024)	161.21 (0.027)	**165.81 (0.033)**
	9	151.08 (0.035)	153.61 (0.024)	154.42 (0.040)	155.12 (0.031)	**156.87 (0.061)**
	10	147.32 (0.039)	150.54 (0.040)	151.47 (0.053)	152.79 (0.033)	**156.32 (0.189)**

Table EC.10 Comparison of LS, PO and tree policies using realistically calibrated max-call option instances, with interest rate $r = 0.02$. Values shown are averages over ten replications, with standard errors in parentheses; bold is used to indicate the best method for each instance in each period. The terms "1st. O." and "2nd. O." indicate the first-order basis function architecture (pricesKO, payoff, KOind) and the second-order basis function architecture (pricesKO, prices2KO, payoff, KOind).

Period	Instance	Stocks
2017	1	DE, AVGO, FFIV, UDR, PIR, INFO, CL, IDXX
	2	GILD, AOS, GPN, HSIC, CMG, EQR, AVGO, HCA
	3	IBM, RMD, WHR, PNC, ROST, A, HCP, MTD
	4	UDR, XOM, INTC, NCLH, CTSH, TTWO, APD, AGN
	5	VIAB, ILMN, CCL, DTE, RF, CMA, EW, NDAQ
	6	DGX, JCI, AET, LB, FIS, MDLZ, HES, BBT
	7	BA, TAP, ZBH, IBM, MCD, APD, TEL, VRSN
	8	BIIB, WYNN, NFX, ALXN, RMD, PYPL, CNP, BWA
	9	XRAY, CMCSA, RF, WMT, MU, TMK, INTC, BEN
	10	STI, ES, SJM, DISCK, LKQ, CTAS, PVH, JPM
2007 – 2017	1	TTWO, COH, AMG, TRV, CSC, ALB, GPN, FLS
	2	ITI, ALB, XRAY, MRK, AGN, GPN, RHT, BXP
	3	ACN, EXPD, IFF, UTX, REGN, TRV, SBUX, AMG
	4	JPM, FCX, HCP, FRT, CSX, ECL, ROK, AMG
	5	RSG, ALK, STX, ORLY, ETR, MRK, AIZ, CAG
	6	HRS, EQIX, ADBE, ATVI, BHI, BBT, CINF, ROST
	7	RE, GLW, BLK, RHI, FE, CDNS, MSFT, ADP
	8	CMT, OMC, HPQ, YUM, CL, ECL, CSC, WMB
	9	BBY, EXC, NSC, AES, ADP, CB, ROP, AAP
	10	MMC, EQR, OKE, JBHT, UPS, AAP, AMGN, DGX

Table EC.11 Stocks used to construct S&P500 instances.