The Choice Function Framework for Online Policy Improvement

Murugeswari Issakkimuthu, Alan Fern, Prasad Tadepalli
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331, USA

Abstract

There are notable examples of online search improving over hand-coded or learned policies (e.g. AlphaZero) for sequential decision making. It is not clear, however, whether or not policy improvement is guaranteed for many of these approaches, even when given a perfect leaf evaluation function and transition model. Indeed, simple counterexamples show that seemingly reasonable online search procedures can hurt performance compared to the original policy. To address this issue, we introduce the choice function framework for analyzing online search procedures for policy improvement. A choice function specifies the actions to be considered at every node of a search tree, with all other actions being pruned. Our main contribution is to give sufficient conditions for stationary and non-stationary choice functions to guarantee that the value achieved by online search is no worse than the original policy. In addition, we describe a general parametric class of choice functions that satisfy those conditions and present an illustrative use case of the empirical utility of the framework.

1 Introduction

For many applications of sequential decision making, it is possible to learn or hand-code a reactive policy for online operation, e.g. (Hussein et al. 2017; Kober, Bagnell, and Peters 2013; Schatzmann et al. 2006). While such policies are computationally cheap to apply, they will generally be suboptimal in some or many states. This motivates using additional computation during online operation to improve upon such base policies. The focus of this paper is on approaches that use online lookahead search for this purpose, which we refer to as Online Search for Policy Improvement (OSPI).

At each state encountered during online operation, OSPI approaches use an environment simulator or model to construct a search tree that includes the base-policy action choices along with a subset of other action choices. The action values at the root can then be used to select an action. Well-known examples of OSPI include the policy-rollout algorithm (Tesauro and Galperin 1997), which was first shown to improve Backgammon policies, and AlphaZero (Silver et al. 2017), which improves over an underlying greedy policy via Monte-Carlo Tree Search.

Ideally, due to the additional online computation, we would like an OSPI procedure to yield improved performance compared to the base policy. Perhaps more importantly, we would at least like to guarantee that an OSPI procedure is "safe" in the sense that it does not perform worse than just using the base policy. For example, the policy rollout algorithm is guaranteed to be safe in this sense. However, as we show in Section 3, many OSPI procedures are not safe, even when 1) using a perfect transition model, 2) using the exact policy value function for leaf evaluation, and 3) the base policy action is expanded at each tree node.

Our primary goal is to derive safety conditions for OSPI. For this purpose, we introduce the choice-function framework for analyzing OSPI procedures. The key idea is to notice that OSPI procedures primarily differ in their choice of which actions other than the base policy action to expand at each tree node. Thus, each procedure can be characterized by a choice function, which specifies the actions to consider at each node of the search tree. Thus, we can characterize properties of an OSPI procedure, such as safety, via properties of the corresponding choice function.

Our main contribution is to give sufficient conditions on choice functions that guarantee safety. This is done for both stationary and non-stationary choice functions. In addition, we describe a parametric class of safe choice functions, that captures a number of existing approaches. This allows for hyperparameter search over a safe space of OSPI procedures in order to optimize online performance. Using this class we provide illustrative empirical results that demonstrate the practical potential of the framework.

2 Related Work

An early approach for OSPI is the policy-rollout algorithm (Bertsekas and Tsitsiklis 1996; Tesauro and Galperin 1997), which has been shown to significantly improve policies in a variety of applications, e.g. Backgammon (Tesauro and Galperin 1997), combinatorial optimization (Bertsekas, Tsitsiklis, and Wu 1997) and stochastic scheduling (Bertsekas and Castanon 1999). Nested rollout (Yan et al. 2005; Cazenave 2009) allows for leveraging additional computation time to further improve a policy by approximating multiple steps of policy iteration. Policy Switching (Chang, Gi-

van, and Chong 2004) allows rolling out multiple policies instead of just one and improves over all the base policies.

Monte-Carlo Tree Search (MCTS) has commonly used policies as a form of knowledge to guide and prune the search, often as part of the rollout policy applied at the leaves (Browne et al. 2012). Recent, high-profile examples include AlphaGo and AlphaZero (Silver et al. 2016; 2017), which combine a learned base policy and value function to guide MCTS. One view of the search approach of AlphaZero is as OSPI, where the search aims to improve over the learned greedy base policy. Indeed, the basis for learning is to use search to generate training data from a (hopefully) improved policy. A related approach (Pinto and Fern 2017) uses a learned policy to prune actions from consideration at each tree node that are not highly ranked by the policy. Another example of combining MCTS with policies (Nguyen et al. 2014) allows the base policy to be treated as a temporally extended action at each node in the search tree.

The idea of searching around a base policy has also been considered in the area of deterministic heuristic search. Limited Discrepancy Search (LDS) (Harvey and Ginsberg 1995) uses a heuristic to define a greedy policy for guiding search. LDS generates all paths in the search tree that disagree with at most K choices of the base policy and returns the best solution uncovered by the search. LDS has been used effectively in a variety of search problems ranging from standard benchmarks to structured prediction (Doppa, Fern, and Tadepalli 2014) and non-deterministic AND/OR search graphs (Larrosa Bondia, Rollón Rico, and Dechter 2016).

3 Problem Setup

We formulate sequential decision making in the formalism of Markov Decision Processes (MDPs). An MDP is a 4-tuple $\langle S, A, P, R \rangle$, where S is a finite set of states, A is a finite set of actions, $P : S \times A \times S \to [0, 1]$ is the state-transition function and $R : S \times A \to \mathbb{R}$ is the reward function. $P_{ss'}(a)$ denotes the probability of reaching state s' from state s taking action a and $R(s, a)$ denotes the immediate reward for taking action a in state s. We focus on the discounted infinite-horizon setting with discount factor γ. A policy, $\pi : S \to A$, is a mapping from states to actions with value function V^π given by

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) \cdot V^\pi(s')$$

for all $s \in S$. Offline computation of an optimal policy can be computationally expensive and impractical for applications with large state spaces. In such cases, online search is a practical alternative to offline solutions.

Online Search for MDPs. In online search, actions are selected only for the states actually encountered during online operation. At each decision point, online search constructs a finite-horizon search tree rooted at the current state. A search tree alternates between layers of state and action nodes. The leaf nodes of a search tree are state nodes and are often evaluated via a state evaluation function. The tree is used to estimate action values at the root and the action with the highest estimate is executed in the environment.

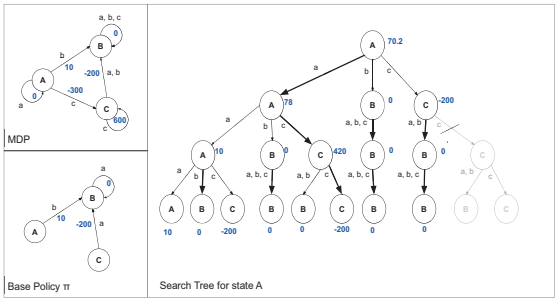

Figure 1: Figure shows a search tree constructed by an unspecified search procedure for the shown deterministic MDP. The base policy is shown and the leaf nodes are assigned the value of the base policy and every internal state node includes the base-policy action. The grayed out part of the tree is the part not expanded by the search procedure. The values of the internal state nodes have been computed via Bellman backup using a discount factor of 0.9. The best action at the root is a since it leads to the depth 1 state with the highest value. The text describes how this choice is not π-safe.

When a model of the MDP is available, an expectimax tree can be built that assigns exact probabilities to child states of actions. For large enough search depths or accurate leaf evaluations, near-optimal actions can be selected. In some applications, only a simulator of the MDP is available, which allows for sampling state transitions and rewards. Monte-Carlo sampling can then be used to construct an approximation to the exact tree by sampling a number of child states for each action node. The Sparse Sampling algorithm (Kearns, Mansour, and Ng 2002) follows this approach and guarantees near optimal action selection in time independent of the size of the state space. Monte-Carlo Tree Search algorithms also use simulators for online search, typically producing search trees of non-uniform depth.

Online Search for Policy Improvement. In practice, online computational constraints can limit the search tree size, which can lead to poor performance of online search. To address this issue, it is common to learn or provide different types of prior knowledge into the search process. For example, the search depth can be reduced by utilizing higher-quality leaf evaluation functions, or the search breadth can be reduced via action pruning functions.

While such knowledge sources can reduce computational cost, there are typically no guarantees on the value achieved by the online search procedure. Most theoretical results aim to guarantee near optimal performance (e.g. (Kearns, Mansour, and Ng 2002)), but require impractical computational costs. Rather, it is desirable to develop approaches that support performance guarantees within practical computational limits. This motivates the framework of OSPI.

An OSPI procedure takes a policy, an environment simulator, an optional leaf evaluation function and a state as input and produces an action as output. OSPI aims for performance guarantees relative to a base policy π. The policy may be learned or hand-coded, but is assumed to be computation-

ally cheap to apply. While π could be directly used for online action selection, this may not fully use the computational resources. OSPI aims to leverage those resources to improve over π via online search to explore the decision space around π. An OSPI procedure is π-*safe* when its online performance is guaranteed to be as good or better than that of π. That is, if π' is the online policy computed by an OSPI procedure, then the procedure is π-safe if $V^{\pi'}(s) \geq V^{\pi}(s)$ for all states s. While safety is a less powerful guarantee compared to near optimality, in practice, it is more attainable and still useful.

It can be difficult to determine, in general, whether a given OSPI procedure is π-safe. For example, one might expect that an OSPI procedure that considers the actions of π at every tree node and uses V^{π} for leaf evaluation might be π-safe when combined with a perfect environment model. However, this is not the case as the following counterexample shows. The next section develops a framework for assessing the safety of OSPI procedures.

Counterexample. Figure 1 shows a deterministic MDP, a base policy π, and the search tree constructed for state A by an unspecified deterministic search procedure, which produces the same tree every time state A is encountered. The tree respects the exact MDP model and the leaf evaluation function is exactly V^{π}. Further, each node of the search tree includes the action corresponding to the choice of π.

The action choice of the search procedure (i.e. the highest valued root actions) will be denoted by the policy π'. Given the tree properties relative to π, we might expect that the value of π' would be at least as good as π. For state A, the base policy selects $\pi(A) = b$ and the corresponding value of state A under π is $V^{\pi}(A) = 10$. However, the online search suggests the alternative action $\pi'(A) = a$, which results in a lower value of $V^{\pi'}(A) = 0$. Thus, the online search procedure is not π-safe, at least for state A.

To understand the failure to be π-safe at state A, consider using the search tree to make a decision at state A at time step t. The reason action a looks best is that the state-action sequence $A \to a \to A \to c \to C \to c \to C$ achieves a high value due to the 600 reward of the final transition. However, after actually taking action a and ending up in state A again at time step $t + 1$ the tree does not include the promising path of $A \to c \to C \to c \to C$, due to pruning of the lower levels of the search tree. Thus, at time step $t + 1$ the search procedure does not recognize the value of taking action c in A and takes a again. This is just one of several failure-mode types of OSPI procedures, even when their trees satisfy the assumptions of this example relative to π.

4 The Choice Function Framework

Search trees encode the future trajectories to be considered when evaluating actions at a state. Search algorithms vary in how they expand the paths, which results in different search trees and hence different action values. Thus, one way to characterize online search approaches is by describing the trees they construct. In our framework, this is done using choice functions, which allows for properties, such as safety of search, to be analyzed via choice-function properties.

4.1 Choice Functions for General Online Search

Search trees have two sources of branching: 1) action branching and 2) state branching. Choice functions describe the action branching by specifying the subset of possible action choices to be considered at each state node. Leaf nodes are assigned the empty set of choices. For state branching, we assume an exact MDP model so that all non-zero probability child states of an action node are included in the tree. When a model is not available, but a simulator is, sparse sampling can be used to approximate the dynamics.

State and action nodes in a tree are identified by paths that list the alternating sequence of states and actions starting at the root state. The path of a state node labeled with state s will be denoted by $p; s$ where p is the path starting at the root leading to the parent action node of the state node. Action nodes will often similarly be denoted by $p; s; a$, where $p; s$ designates the parent state node. The length of any path denoted by $|p|$ is the number of actions that it contains. Thus, a path corresponding to a single state s has length zero. The set of all paths that end with a state is denoted by $\mathcal{SP}$ and a *choice function* is a mapping $\psi : \mathcal{SP} \to 2^A$ from paths that end with a state to action subsets.

In order to define the trees associated with a choice function, several definitions are needed. A path is ψ-*satisfying* if all of its actions are "allowed" by ψ. That is, for each prefix $p'; s'; a'$ of the path, we have $a' \in \psi(p'; s')$. A state path $p; s$ is a *leaf path* of ψ if it is ψ-satisfying and $\psi(p; s) = \emptyset$. A leaf path of ψ cannot be extended to a ψ-satisfying path. A choice function ψ is *finite horizon* if there is a finite upper bound on the length of any ψ-satisfying state path. For finite-horizon ψ, the *horizon* $H(\psi)$ is the maximum length of any ψ-satisfying path, or equivalently, of any leaf path.

Given a current, or root state, s_0, the tree corresponding to ψ, denoted $T^{\psi}(s_0)$, is the tree containing exactly the ψ-satisfying state paths that begin with s_0. Thus, the leaf nodes of $T^{\psi}(s_0)$ correspond to leaf paths of ψ. The tree will be finite when ψ is finite horizon, with $H(\psi)$ bounding the depth of any leaf node. In this paper, we will restrict attention to finite-horizon choice functions and hence finite trees.

To use the tree $T^{\psi}(s_0)$ for action selection at state s_0, it is necessary to specify a *leaf evaluation function* u, which is a function of states $u : S \to \mathbb{R}$. Often u will be a learned or hand-coded function that provides an estimate of a state's optimal value or value under a policy. Alternatively, u may be uninformative and return a constant value. Together, a choice function ψ and leaf evaluation function u allow us to define the value of each state node $p; s$ in $T^{\psi}(s_0)$, denoted $V_u^{\psi}(p; s)$, and each action node $p; s; a$, denoted $Q_u^{\psi}(p; s; a)$.

$$V_u^{\psi}(p; s) = \begin{cases} u(s), & \psi(p; s) = \emptyset \\ \max\limits_{a \in \psi(p;s)} Q_u^{\psi}(p; s; a), & \textit{otherwise} \end{cases}$$

$$Q_u^{\psi}(p; s; a) = R(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^{\psi}(p; s; a; s')$$

The online-search action policy, denoted Π_u^{ψ}, returns a maximum valued action at state s allowed by ψ, with ties broken arbitrarily: $\Pi_u^{\psi}(s) = \arg\max\limits_{a \in \psi(s)} Q_u^{\psi}(s; a)$.

4.2 Choice Functions for OSPI

Given a base policy π we would like to define choice functions and corresponding leaf-evaluation functions that result in (approximately) π-safe OSPI procedures. That is, we would like to guarantee that Π_u^ψ is π-safe. Section 5 develops sufficient conditions on choice functions to give such a guarantee. First, however, we provide examples of choice functions for several existing OSPI procedures whose safety will later be assessed according to the conditions.

Policy Rollout. This simple OSPI procedure (Tesauro and Galperin 1997) returns the action at state s that maximizes a Monte-Carlo estimate of $Q^\pi(s,a)$. This estimate can be viewed as evaluating a tree that considers all actions at the root s and then only contains the actions of π thereafter until some horizon H. Policy rollout can thus be characterized by the following choice function.

$$\psi_{\text{ro}}(p;s) = \begin{cases} A, & |p| = 0 \\ \pi(s), & 0 < |p| < H \\ \emptyset, & \text{otherwise} \end{cases}$$

Policy rollout can be proven to be π-safe as it corresponds to the policy improvement step of the policy iteration algorithm. Our π-safe conditions will imply this for ψ_{ro}.

Limited Discrepancy Search (LDS). This procedure was originally introduced for deterministic offline search problems (Harvey and Ginsberg 1995). LDS searches around π by limiting the number of discrepancies (off-policy actions) along every root-to-leaf path to K up to some maximum horizon H. The idea was later extended to offline nondeterministic AND/OR tree/graph search (Larrosa Bondia, Rollón Rico, and Dechter 2016) using a similar limit on discrepancies. LDS for MDPs is easily captured via the following choice function, where $\#[p \neq \pi]$ is the number of off-policy actions in path p.

$$\psi_{lds}(p;s) = \begin{cases} A, & |p| < H \text{ and } \#[p \neq \pi] < K \\ \pi(s), & |p| < H \text{ and } \#[p \neq \pi] = K \\ \emptyset, & |p| = H \end{cases}$$

Our conditions will imply that ψ_{lds} is π-safe.

Pruned Online Search with Learned Policies. Reinforcement Learning (RL) algorithms typically learn policies that select actions by maximizing an action ranking function, such as a Q-function or probability distribution over actions. Such action rankings can be used for action pruning in online tree search. Let $q(p;s,a)$ be the learned action ranking function, which may depend on the full path $p;s$ (e.g. when q is a recurrent neural network) or depend only on s. A simple pruning approach such as the one studied in (Pinto and Fern 2017), allows only the set of top k actions at each search node, denoted $\text{TOP}_{q,k}(p;s)$, as captured by the following choice function.

$$\psi_{q,k}(p;s) = \begin{cases} \text{TOP}_{q,k}(p;s), & |p| < H \\ \emptyset, & |p| = H \end{cases}$$

As discussed in Section 6, our results will help clarify conditions on q that ensure safety.

5 Performance Guarantee

Our goal is to identify properties of a choice function ψ that guarantee that the online-search action policy Π_u^ψ is approximately π-safe. That is, we seek to bound $V^\pi(s) - V^{\Pi_u^\psi}(s)$. A natural property to suggest is that ψ be consistent with π. A choice function ψ is π-consistent if $\pi(s) \in \psi(p;s)$ for each path $p;s$ that ends with a state. Our counterexample in section 3, however, is based on a π-consistent choice function, since all tree nodes include π. Thus, π-consistency of ψ is not sufficient for π-safety, requiring the introduction of additional concepts and notation.

We will often treat value functions as vectors, indexed by states, with arithmetic and comparison operators being applied element-wise. The max-norm of a vector $\|V\|_\infty$ returns the maximum absolute value of the elements. The *min-horizon* of ψ, denoted $h(\psi)$, is the minimum depth of any leaf node in $T^\psi(s_0)$ for any state s_0. Given a path $p;s$ we let $\_p;s$ denote the path obtained by removing the leftmost state-action pair of $p;s$. We say that ψ is *monotonic* if the set of actions returned by ψ for state s when reached via path p is a subset of the set of actions returned for s when reached via the path with the leftmost state-action pair of p removed, i.e., $\psi(p;s) \subseteq \psi(\_p;s)$ for all $p;s \in \mathcal{SP}$. We can now give our main result.

Theorem 1. *For any MDP, discount factor γ, and policy π, if ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty \leq \epsilon$, then for $\pi' = \Pi_u^\psi$,*

$$V^\pi - V^{\pi'} \leq \frac{2\epsilon\gamma^{h(\psi)}}{1-\gamma}.$$

This bound implies that in the ideal case when $u = V^\pi$, monotonicity and π-consistency together are sufficient for safety. It also shows that the impact of inaccuracy in u with respect to V^π decreases exponentially with the min-horizon due to discounting of future returns.

From the theorem we get an immediate corollary that applies to the set of all policies ψ is consistent with, denoted $\mathcal{C}_\psi$, where $\epsilon(u,\pi) = \|u - V^\pi\|_\infty$.

Corollary 1. *For any MDP, discount factor γ, leaf evaluation function u, π-consistent and monotonic choice function ψ, the policy $\pi' = \Pi_u^\psi$ satisfies*

$$V^{\pi'} \geq \max_{\pi \in \mathcal{C}_\psi} \left[V^\pi - \frac{2\epsilon(u,\pi)\gamma^{h(\psi)}}{1-\gamma} \right].$$

This implies that for a large min-horizon the online policy is guaranteed to be safe with respect to the best policy that ψ is consistent with. In general, this shows the performance trade-off between larger min-horizons (i.e., minimum search depth) and the closeness of u to a good policy.

5.1 Proof of Theorem 1

All proofs not in the main text are in the supplementary material. The high-level idea of our proof is inspired by the analysis of offline multi-step policy improvement (Bertsekas and Tsitsiklis 1996). This procedure starts with the value function $V_0 = V^\pi$ and then performs m applications of the *Bellman Backup* operator B to get a sequence of value functions $V_i = B[V_{i-1}]$, where B is defined as follows.

$$B[V](s) = \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a)\, V(s')$$

The greedy policy π_m with respect to the final value function V_m is then returned as the improved policy over π. The monotonicity of B can be used to guarantee that $V^{\pi_m} \geq V^{\pi}$.

An OSPI procedure for computing $\pi_m(s)$ is to evaluate a depth $m+1$ tree with root s using V^{π} for leaf values. This computes the greedy action with respect to V_m, but without synchronously updating all states from the bottom up. Thus, the offline guarantee carries over to OSPI. OSPI with choice functions can be viewed similarly but with backups restricted to actions allowed by the choice function. Our analysis below generalizes these ideas to path-sensitive choice functions and approximate leaf values.

To prove the main result we start by introducing a number of lemmas. It will be useful to introduce the *policy-restricted Bellman Backup* operator $B_\pi[V]$, which restricts backups to only consider the actions of π.

$$B_\pi[V](s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s))\, V(s').$$

Lemma 1 gives a lower bound on V^π in terms of a value vector V and how much B_π decreases the value of V.

Lemma 1. *For any policy π and value vector V, if $V - B_\pi[V] \leq \delta$, then $V - V^\pi \leq \frac{\delta}{1-\gamma}$.*

Next, Lemma 2 generalizes the conditions that guarantee policy improvement in the offline multi-step lookahead policy improvement procedure to OSPI with choice functions. Proposition 1 follows from lemma 2.

Lemma 2. *If a stationary choice function ψ is π-consistent and monotonic and $u = V^\pi$ then for any path $p; s$ such that $1 \leq |p; s| \leq H(\psi)$, $V_u^\psi(\lrcorner p; s) \geq V_u^\psi(p; s)$.*

Proposition 1. *If a stationary choice function ψ is π-consistent and monotonic and $u = V^\pi$, then $V_u^\psi(s) \geq V^\pi(s)$.*

Lemma 3 below bounds the values of paths of length less than or equal to $h(\psi)$ for two different leaf evaluation functions u and u' satisfying $\|u - u'\|_\infty \leq \epsilon$. This will be useful for quantifying the impact of the leaf evaluation function being an approximation to the base policy value function.

Lemma 3. *If ψ is a stationary choice function and $\|u - u'\|_\infty \leq \epsilon$ then for any path $p; s$ with $|p; s| \leq h(\psi)$, $\left|V_u^\psi(p; s) - V_{u'}^\psi(p; s)\right| \leq \epsilon \gamma^{h(\psi) - |p;s|}$.*

For the following we use the notation $V_{u,k}^\psi$ to denote the vector consisting of the elements of V_u^ψ for $|p; s| = k$. In particular, $V_{u,0}^\psi$ is the vector of the values of all root nodes, i.e., $|p; s| = 0$. Lemma 4 and proposition 2 below are key results that combine to bound the difference between $V_{u,0}^\psi$ and the value of Π_u^ψ.

Lemma 4. *If a stationary choice function ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty \leq \epsilon_\pi$, then for $\pi' = \Pi_u^\psi$, $V_{u,0}^\psi - B_{\pi'}[V_{u,0}^\psi] \leq \epsilon_\pi \gamma^{h(\psi)}(1+\gamma)$.*

Proposition 2. *If a stationary choice function ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty \leq \epsilon_\pi$, then for $\pi' = \Pi_u^\psi$, $V_{u,0}^\psi - V^{\pi'} \leq \frac{\epsilon_\pi \gamma^{h(\psi)}(1+\gamma)}{1-\gamma}$.*

Proof. Directly combine lemmas 4 and 1. □

Using the above lemmas we can now prove the main result.

Theorem 1. *If a stationary choice function ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty = \epsilon_\pi$, then for $\pi' = \Pi_u^\psi$,*

$$V^\pi - V^{\pi'} \leq \frac{2\epsilon_\pi \gamma^{h(\psi)}}{1-\gamma}.$$

Proof. In the proof, V^π is denoted as $\bar{u}$ to simplify notation.

$$\begin{aligned}
V^\pi(s) - V^{\pi'}(s) &= V^\pi(s) - V_{u,0}^\psi(s) + V_{u,0}^\psi(s) - V^{\pi'}(s), \\
&\leq V^\pi(s) - (V_{\bar{u},0}^\psi(s) - \epsilon_\pi \gamma^{h(\psi)}) + V_{u,0}^\psi(s) - V^{\pi'}(s), \\
&\qquad \textit{by lemma 3} \\
&\leq \epsilon_\pi \gamma^{h(\psi)} + V_{u,0}^\psi(s) - V^{\pi'}(s), \quad \textit{since } V_{\bar{u},0}^\psi(s) \geq V^\pi(s) \\
&\qquad \textit{by proposition 1} \\
&\leq \epsilon_\pi \gamma^{h(\psi)} + \frac{\epsilon_\pi \gamma^{h(\psi)}(1+\gamma)}{1-\gamma}, \qquad \textit{by proposition 2} \\
&= \frac{2\epsilon_\pi \gamma^{h(\psi)}}{1-\gamma}.
\end{aligned}$$

□

5.2 Non-Stationary Choice Functions

We have assumed that choice functions are stationary, i.e., the same choice function is used across online decision steps. Some OSPI approaches, however, correspond to a non-stationary choice function which vary across steps. For example, some search algorithms use a sub-tree produced at time step t as a starting point for search at time step $t+1$ or randomized OSPI approaches are non-stationary due to different random seeds across steps. Here, we extend our analysis to the non-stationary case.

A non-stationary choice function $\Psi = (\psi_1, \psi_2, \psi_3, \ldots)$ is a sequence of time-step indexed stationary choice functions ψ_t. To relate two different stationary choice functions ψ and ψ' we say that ψ subsumes ψ', denoted $\psi \supseteq \psi'$, if for every path $p; s$, $\psi(p; s) \supseteq \psi'(p; s)$. We can extend the bound in Theorem 1 to a non-stationary choice function Ψ when each ψ_t satisfies the conditions of that theorem, each ψ_t has the same set of leaf paths, and $\psi_{t+1} \supseteq \psi_t$ for each time-step t.

Theorem 2. *Let $\Psi = (\psi_1, \psi_2, \ldots)$ be a non-stationary choice function such that each component choice function ψ_t is monotonic and π-consistent and $\|u - V^\pi\|_\infty = \epsilon_\pi$. If all ψ_t have the same set of leaf paths and for each time-step t, $\psi_{t+1} \supseteq \psi_t$, then for $\pi' = \Pi_u^\psi$ and all $s \in S$,*

$$V^\pi(s) - V^{\pi'}(s) \leq \frac{2\epsilon_\pi \gamma^{h(\psi_1)}}{1-\gamma}.$$

The proof is in the supplementary material. This result has implications on the design of OSPI procedures that correspond to non-stationary choice functions. For example, many MCTS-based approaches, such as that used by AlphaZero, do not appear to correspond to non-stationary choice functions that satisfy these conditions. This does not mean that they will not perform well in a particular application, but suggests that for some applications they can have fundamental issues that degrade the performance of a base policy,

even ignoring inaccuracies due to Monte-Carlo sampling. One way to adjust some of these and other algorithms to achieve the subsumption property is to build upon the relevant subtrees constructed at time t at time step $t+1$. The practical impact of such a change is an empirical question worth investigating.

6 Limited Discrepancy Choice Functions

We do not expect a single type of choice function to perform best across all problems. Rather, in practice, the selection of a choice function can be similar to the selection of other application-dependent hyperparameters. This motivates defining parametric families of choice functions that span different trade-offs between decision time and quality. In particular, given an application's decision-time constraints, offline optimization can be used to select a high-performing choice function that satisfies the constraints.

To support this vision, we introduce the parametric family of *Limited Discrepancy Choice Functions (LDCFs)*. We show that all LDCFs are monotonic and π-consistent for any π and hence satisfy our safety conditions. We then analyze how the parameters of the LDCF family relate to the computational complexity of online action selection. Finally we relate LDCFs to previously introduced examples.

LDCF Definition and Safety. An LDCF ψ defines a uniform-horizon tree, which limits the discrepancies w.r.t. a base policy along root-to-leaf paths by their number and depth. In cases where a base policy only makes occasional errors the discrepancy limit makes intuitive sense. Indeed, search can improve the policy by "correcting" the rare errors along paths by introducing discrepancies. This suggests that there can be a computational advantage to bounding search by discrepancies in applications of OSPI to learned policies that already perform well but can still be improved.

A discrepancy w.r.t. π in path p is a consecutive state-action pair (s,a) in p such that $a \neq \pi(s)$. LDCFs are parameterized by the tuple (π, H, K, D, Δ), where π is the base policy, H is the uniform horizon bound, $K \leq H$ is a bound on the number of discrepancies in a path, and $D < H$ is a bound on the maximum depth that a discrepancy may appear in a path. Finally, the *discrepancy proposal function* $\Delta : S \times \{0, \ldots, D\} \rightarrow 2^A$ maps pairs of states and depths to action subsets. Intuitively Δ returns the discrepancies that can be considered at a state node $p; s$ at depth $|p|$ which has not yet reached the discrepancy limit imposed by K and D. We allow Δ to depend on depth, since it is often useful to allow for more discrepancies at shallower depths. Given parameters $\theta = (\pi, H, K, D, \Delta)$ the corresponding LDCF is defined as follows.

$$\psi_\theta(p; s) = \begin{cases} \Delta(s, |p|) \cup \{\pi(s)\}, & |p| \leq D \text{ and } \#[p \neq \pi] < K \\ \pi(s), & D < |p| < H \text{ or } \#[p \neq \pi] = K \\ \emptyset, & |p| = H \end{cases}$$

All members of the LDCF family are π-consistent by construction. However, monotonicity of an LDCF requires constraining Δ. We say that Δ is *depth monotonic* if $\Delta(s,d) \supseteq \Delta(s,d+1)$ for all $s \in S$ and d.

Theorem 3. *For LDCF parameters $\theta = (\pi, H, K, D, \Delta)$, if Δ is depth monotonic, then ψ_θ is monotonic.*

The proof is in the supplementary material. A straightforward way to obtain a depth monotonic Δ is to use a learned action-ranking function over states and return the top ranked actions at a state, where the number of returned actions is non-increasing with tree depth.

Application to Special Cases. The choice function ψ_{lds} is a restricted LDCF with $\Delta(s,d) = A$, which trivially satisfies our safety conditions. The LDCF space provides more flexibility on how to better control the introduction of discrepancies compared to traditional LDS.

The policy rollout choice function ψ_{ro} is a special case of an LDCF with $D = 0$ and $\Delta(s,0) = A$, which allows all choices at the root and only the base policy's choices thereafter. Since Δ is trivially depth monotonic, our safety result applies. This can be generalized to multi-step look-ahead rollout (Bertsekas and Tsitsiklis 1996), where the top M levels of the tree are fully expanded followed by restricting actions to those of the base policy until the horizon. Specifically, $D = M$ and $\Delta(s,d) = A$, which again satisfies our safety conditions. Note that when $D = H - 1$ this degenerates to value iteration with horizon H.

Finally, the pruned-search choice function $\psi_{q,k}$ is an LDCF with a specific choice of discrepancy function $\text{TOP}_{q,k}$. Our safety conditions specify sufficient constraints that the action ranking function q should satisfy. When q is history independent, $\text{TOP}_{q,k}$ is depth monotonic. Otherwise, if q is history-dependent, e.g. a recurrent neural network, no such guarantee can be made. However, it is relatively straightforward to put a wrapper around such a q to ensure depth monotonicity.

Computational Complexity. Increasing H, K, and D, and the size of sets returned by Δ can be expected to improve $\Pi_u^{\psi_\theta}$ for reasonable u. This comes at the cost of higher computational complexity typically dominated by the number of leaves in T^{ψ_θ}. In addition to the LDCF parameters, the number of leaf nodes depends on the *state branching factor* C, which is the maximum number of state nodes under an action node. When an exact model is used, this is the maximum number of non-zero probability successor states. For Monte-Carlo algorithms, this is the number of successor states sampled for each action node. Given the state branching factor and an upper bound on the number of actions returned by Δ we get the following bound on tree size.

Proposition 3. *Let ψ_θ be an LDCF with $\theta = (\pi, H, K, D, \Delta)$, such that $\Delta(s) \leq W$ for any $s \in S$. The number of leaf nodes in T^{ψ_θ} with state branching factor C is upper bounded by $2C^H$ for $(D+1)W = 1$ and by $\frac{((D+1)W)^{K+1}-1}{(D+1)W-1} C^H = O\left((DW)^K C^H\right)$ for $(D+1)W \neq 1$.*

Ignoring the impact of C, which is controlled by the search algorithm, the complexity is dominated by K. We also see that for deterministic domains where $C = 1$, there is no exponential dependence on H.

7 Illustrative Empirical Results

Our primary contribution is theoretical, however, here we illustrate the potential practical utility of our framework using

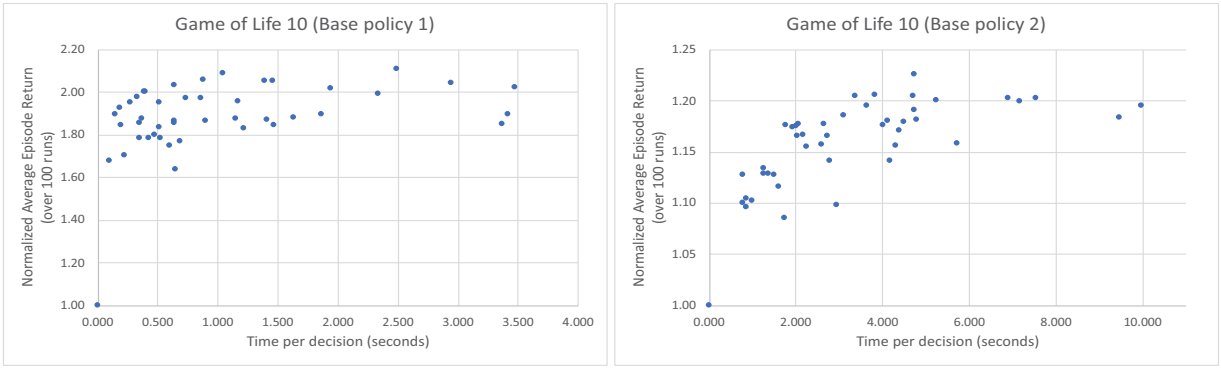

Figure 2: Performance vs time-per-step for LCDF choice functions applied to linear (left) and non-linear (right) base policies.

the LDCF family. The experiments are intended to illustrate the viewpoint of a choice function being a hyperparameter to be tuned offline.

We implemented a variant of Forward Search Sparse Sampling (FSSS) (Walsh, Goschin, and Littman 2010) for approximately computing the online policy Π_u^ψ for any LDCF ψ and leaf-evaluation function u using an MDP simulator. The key parameter, other than the choice-function, is the sampling width C, which controls how many state transitions are sampled for each action node. The supplementary material contains a summary of the algorithm. Our implementation is generally applicable and will be released upon publication with information to reproduce our experiments.

Experiments Setup. A full image of our experimental environment will be available upon publication. We run experiments in the domain Game-of-Life, a benchmark domain from the International Probabilistic Planning Competition. This is a grid-based domain with each grid-cell either being alive with some probability depending on its neighbors. Actions allow for selecting one cell at each time step (or none) to set to be alive in the next time step. The reward is based on the number of alive cells at each step. There are 10 problems of grid sizes 3×3, 4×4, 5×5 and 10×3. Problems have different levels of stochasticity in the dynamics.

We used supervised learning via imitation of a planner to train two base policies represented as neural networks, using the same approach as in (Issakkimuthu, Fern, and Tadepalli 2018). Each network outputs a probability distribution over actions. Policies 1 and 2 are base policies. Policy 1 is a linear network, while Policy 2 is a non-linear network with 3 hidden layers. For each base policy, we consider four leaf evaluation functions. The first is the constant zero function. The remaining three are neural networks with different configurations trained on a dataset of 5000 state-value pairs obtained via Monte-Carlo simulation of each policy. All the networks have been trained using Tensorflow to minimize the mean squared error.

We experiment with 11 choice functions in the LDCF family with parameters H in $\{3, 4, 5\}$ and (D, K) in $\{(0, 1), (1, 1), (1, 2), (2, 1)\}$. The combination $(2, 1)$ is not applicable for $H = 3$. The number of discrepancies considered at the root node is 9 and other internal nodes is 1. The discrepancies are determined from the action probabilities given by the base policy. We use $C = 3$ for FSSS.

	LDCF Policy 1		**LDCF Policy 2**	
Prob. #	Normalized Avg. Reward	Decision Time (s)	Normalized Reward	Decision Time (s)
1	2.57 ± 0.07	0.081	1.08 ± 0.04	0.491
2	1.27 ± 0.10	0.345	0.95 ± 0.06	0.631
3	1.11 ± 0.05	0.129	0.92 ± 0.03	0.374
4	1.51 ± 0.03	0.523	1.03 ± 0.02	0.830
5	1.14 ± 0.03	1.084	1.00 ± 0.03	6.298
6	1.05 ± 0.02	1.223	0.96 ± 0.02	9.163
7	1.54 ± 0.02	0.523	1.05 ± 0.01	1.957
8	1.21 ± 0.02	4.488	1.02 ± 0.02	10.463
9	1.13 ± 0.02	3.262	0.96 ± 0.01	3.749
10	2.11 ± 0.04	2.493	1.23 ± 0.02	4.746

Table 1: Game-of-Life - Best Normalized reward.

Given one of these policies and a problem, we would like to identify the best combination of LDCF parameters and leaf evaluation function given a constraint on the time-per-step. In practice, this could be done in an offline tuning phase where different configurations are evaluated. Figure 2 shows a scatter plot of the normalized reward versus time-per-step for each of the 44 configurations (11 LDCF settings and 4 leaf evaluation functions). The normalized reward is the average reward per episode divided by the average reward per episode of the base policy. Values greater than one perform better than the base policy. For both base policies all LDCF configurations perform better. There is also a larger improvement for base policy 1, which makes sense due to the fact that policy 2 is a much higher-quality policy and hence more difficult to improve over. We also see that the LDCF space shows considerable coverage of the performance vs. time space, which shows the utility of offline tuning. There is a general trend toward better performance for larger times, but this is not uniformly true. There are complex interactions between the LDCF parameters and a particular problem, which makes it unlikely that a single feature such as time-per-decision is always the best indicator.

Table 1 gives results for each of the 10 problems, which includes the normalized rewards with confidence intervals for the best performing LDCF configuration for each of the

policies. We see that for the linear policy, the best LDCF configuration is never significantly worse (lower interval is greater than 1) and often significantly better. For the second non-linear policy, we see that for most problems the LDCF performance is not significantly worse than the policy (confidence interval contains 1) and sometimes significantly better. For three problems, the upper confidence bound is less than one, indicating a significant decrease in performance. These problems happen to be among the most stochastic problems in the benchmark set. This suggests that a likely reason for the decrease in performance is due to the relatively small sampling width used for FSSS ($C = 3$), which provides a poor approximation for such problems.

8 Summary

We have introduced a framework for analyzing online search procedures for policy improvement guarantees. The key idea is to separate the action specification part of search from the search process and create an abstract concept called choice functions. A choice function instance will then be a parameter of search. We identify properties of choice functions to provide sufficient conditions for guaranteed online policy improvement when the leaf evaluation function is perfect. Our main result is a bound on the performance of the online policy relative to the base policy for any leaf evaluation function. We have also introduced a parameterized class of choice functions called LDCF. Our next directions are to explore the practical application of the framework across a wide range of problems and to integrate notions of state abstraction into the framework.

9 Acknowledgements

This work was supported by NSF grants IIS-1619433, IIS-1724360, and DARPA contract N66001-19-2-4035. We thank Intel for compute support.

References

Bertsekas, D. P., and Castanon, D. A. 1999. Rollout Algorithms for Stochastic Scheduling Problems. *Journal of Heuristics* 5(1):89–108.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bertsekas, D. P.; Tsitsiklis, J. N.; and Wu, C. 1997. Rollout Algorithms for Combinatorial Optimization. *Journal of Heuristics* 3(3):245–262.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1):1–43.

Cazenave, T. 2009. Nested Monte-Carlo Search. In *Twenty-First IJCAI*.

Chang, H. S.; Givan, R.; and Chong, E. K. P. 2004. Parallel Rollout for Online Solution of Partially Observable Markov Decision Processes. *Discrete Event Dynamic Systems* 14(3):309–341.

Doppa, J. R.; Fern, A.; and Tadepalli, P. 2014. Structured Prediction via Output Space Search. *JMLR* 15(1):1317–1350.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited Discrepancy Search. In *IJCAI (1)*, 607–615.

Hussein, A.; Gaber, M. M.; Elyan, E.; and Jayne, C. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50(2):21.

Issakkimuthu, M.; Fern, A.; and Tadepalli, P. 2018. Training deep reactive policies for probabilistic planning problems. In *Twenty-Eighth ICAPS*.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A Sparse Sampling Algorithm for near-optimal Planning in Large Markov Decision Processes. *Machine Learning* 49(2-3):193–208.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11):1238–1274.

Larrosa Bondia, F. J.; Rollón Rico, E.; and Dechter, R. 2016. Limited Discrepancy AND/OR Search and its Application to Optimization Tasks in Graphical Models. In *Proceedings of the Twenty-Fifth IJCAI*, 617–623.

Nguyen, T.-H. D.; Silander, T.; Lee, W.-S.; and Leong, T.-Y. 2014. Bootstrapping Simulation-based Algorithms with a Suboptimal Policy. In *Twenty-Fourth ICAPS*.

Pinto, J., and Fern, A. 2017. Learning Partial Policies to Speedup MDP Tree Search via Reduction to IID Learning. *JMLR* 18(1):2179–2213.

Schatzmann, J.; Weilhammer, K.; Stuttle, M.; and Young, S. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The knowledge engineering review* 21(2):97–126.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529(7587):484.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the Game of Go without Human Knowledge. *Nature* 550(7676):354.

Tesauro, G., and Galperin, G. R. 1997. On-line Policy Improvement using Monte-Carlo Search. In *Advances in Neural Information Processing Systems*.

Walsh, T. J.; Goschin, S.; and Littman, M. L. 2010. Integrating sample-based planning and model-based reinforcement learning. In *AAAI'10*, 612–617. AAAI Press.

Yan, X.; Diaconis, P.; Rusmevichientong, P.; and Roy, B. V. 2005. Solitaire: Man versus Machine. In *Advances in Neural Information Processing Systems*, 1553–1560.