Journal of Artificial Intelligence Research 60 (2017) 717-777 Submitted 02/17; published 12/17

Sample-Based Tree Search with Fixed and Adaptive State Abstractions

Jesse Hostetler HOSTETJE@LIFETIME.OREGONSTATE.EDU
Alan Fern AFERN@EECS.OREGONSTATE.EDU
Thomas Dietterich TGD@EECS.OREGONSTATE.EDU
Oregon State University,
Department of Electrical Engineering and Computer Science,
Corvallis, OR 97331 USA

Abstract

Sample-based tree search (SBTS) is an approach to solving Markov decision problems based on constructing a lookahead search tree using random samples from a generative model of the MDP. It encompasses Monte Carlo tree search (MCTS) algorithms like UCT as well as algorithms such as sparse sampling. SBTS is well-suited to solving MDPs with large state spaces due to the relative insensitivity of SBTS algorithms to the size of the state space. The limiting factor in the performance of SBTS tends to be the exponential dependence of sample complexity on the depth of the search tree. The number of samples required to build a search tree is $O((|\mathcal{A}|B)^d)$, where $|\mathcal{A}|$ is the number of available actions, B is the number of possible random outcomes of taking an action, and d is the depth of the tree. State abstraction can be used to reduce B by aggregating random outcomes together into abstract states. Recent work has shown that abstract tree search often performs substantially better than tree search conducted in the ground state space.

This paper presents a theoretical and empirical evaluation of tree search with both fixed and adaptive state abstractions. We derive a bound on regret due to state abstraction in tree search that decomposes abstraction error into three components arising from properties of the abstraction and the search algorithm. We describe versions of popular SBTS algorithms that use fixed state abstractions, and we introduce the Progressive Abstraction Refinement in Sparse Sampling (PARSS) algorithm, which adapts its abstraction during search. We evaluate PARSS as well as sparse sampling with fixed abstractions on 12 experimental problems, and find that PARSS outperforms search with a fixed abstraction and that search with even highly inaccurate fixed abstractions outperforms search without abstraction. These results establish progressive abstraction refinement as a promising basis for new tree search algorithms, and we propose directions for future work within the progressive refinement framework.

1. Introduction

Sequential decision-making problems of practical interest often have large state spaces. Scaling up Markov decision process (MDP) solvers to handle these large state spaces is an ongoing challenge. A solution to an MDP is a policy π mapping the set of states $\mathcal{S}$ to the set of actions $\mathcal{A}$. Solution methods for MDPs can be categorized roughly by whether they implement π via *offline* or *online* computation. Offline approaches compute and explicity represent an executable policy, such as by storing the action to take in each different state in a lookup table. Reinforcement learning (RL) and dynamic programming algorithms

typically compute their policies offline; although an RL agent may learn during execution, the bulk of policy optimization has already occurred. A policy computed offline must be defined in every reachable state, and thus the time and space complexity of computing the policy depend on the size of the state representation. A naïve tabular representation requires polynomial time and space in $|\mathcal{S}|$, making such a representation impractical for problems where $|\mathcal{S}|$ is large. Much research in reinforcement learning focuses on learning and representing the policy in a compact form.

Online planning (OP) methods, by contrast, estimate the optimal action in one state at a time, only in those states that are actually encountered during execution. Instead of representing a complete policy, OP methods compute the policy on demand, creating a partial policy defined in only a subset of the state space. Because a single execution trajectory visits only a relatively small number of states, OP algorithms can achieve good performance while enumerating only a small subset of the state space.

The "planning" portion of online planning can be realized in many forms. This paper focuses on *sample-based tree search* (SBTS) algorithms, which plan by constructing a lookahead tree using random samples from a generative model of the MDP. Within the SBTS family of algorithms, Monte Carlo tree search (MCTS) (Browne et al., 2012) and in particular the UCT algorithm (Kocsis & Szepesvári, 2006) has risen to prominence, largely due to the success of UCT variants in the game of *go* (Gelly & Silver, 2007; Silver et al., 2016) and in other complex domains (e.g. Balla & Fern, 2009; Guo, Singh, Lee, Lewis, & Wang, 2014). The key property of tree search algorithms like UCT and sparse sampling (SS) (Kearns, Mansour, & Ng, 2002) is that they achieve bounded regret in the root state with a number of samples that does not depend on the state space size. This property makes these algorithms attractive choices for problems like *go* that have large state spaces.

The primary disadvantage of OP algorithms is that they require significant computation at execution time in order to make a decision. An online planning algorithm controlling a real system will always face constraints on computational resources that limit how many samples can be drawn before a decision must be made. The number of samples required theoretically to guarantee meaningful error bounds is usually impractically large. In this *anytime* online planning setting, the planner might have to halt and produce an answer at any time, so it is important that the planner produces a reasonable initial solution quickly even if that solution is not optimal. Any remaining planning time can then be spent improving the initial solution.

State abstraction is one way of trading optimality for speed. A state abstraction reduces the size of the state space by treating some states as equivalent. State abstractions are ubiquitous in applications of reinforcement learning, where they provide a direct benefit because the complexity of the algorithms depends on the state space size. Recently, there has been increasing interest in applying state abstraction to SBTS algorithms as well (Van den Broeck & Driessens, 2011; Jiang, Singh, & Lewis, 2014; Anand, Grover, Mausam, & Singla, 2015; Hostetler, Fern, & Dietterich, 2014, 2015). The size of a search tree, and thus the number of transition samples necessary to build it, is $O((|\mathcal{A}|B)^d)$, where $|\mathcal{A}|$ is the size of the action set, B is the maximum number of possible stochastic outcomes of any action (the *stochastic branching factor*), and d is the search depth. Although the asymptotic complexity of tree search does not depend on the state space size, the stochastic branching factor B of a search tree is bounded by the size of the state space, and thus an appropriate abstraction

can reduce the sample complexity of the search by reducing B. State abstraction introduces a new source of error, but the regret due to abstraction can be bounded (Hostetler et al., 2014; Jiang et al., 2014). Experimentally, abstract tree search algorithms have been found to perform better than search without abstraction in a variety of test domains (Jiang et al., 2014; Hostetler et al., 2015; Anand et al., 2015).

This article consolidates and expands upon work on state abstraction in tree search that first appeared in two earlier articles (Hostetler et al., 2014, 2015). Its contents are based on the Ph.D. dissertation of one of the authors (Hostetler, 2017). Hostetler et al. (2014) derived a regret bound for a class of state aggregation abstractions applied to SBTS, and described implementations of abstract UCT and abstract SS. These algorithms build search trees with respect to a fixed abstraction, which must be provided as input. To address the problem of abstraction specification, Hostetler et al. (2015) introduced the Progressive Abstraction Refinement for Sparse Sampling (PARSS) algorithm, which progressively refines its state abstraction during search beginning from an initial coarse abstraction. The current paper contributes an expanded discussion and analysis of the algorithms presented in these two earlier papers, as well as a more comprehensive experimental evaluation of the PARSS algorithm, including new variations on the basic PARSS algorithm as well as additional experimental domains. Our experiments compare PARSS to sparse sampling in the ground state space and sparse sampling with fixed abstractions. Consistent with earlier work, the results show that PARSS outperforms sparse sampling with a fixed state representation and that search with fixed, uninformed abstractions often outperforms search in the ground state space. We also find that principled heuristics for controlling the abstraction refinement process yield superior performance compared to uninformed strategies. This suggests a direction for future work in learning to control the refinement process.

We begin the remainder of the paper by introducing background and notation for MDPs and SBTS algorithms (Section 2). We view SBTS algorithms as methods for sampling an approximate model of an MDP over state-action histories, and all of our algorithms and theory are presented in this context. Section 3 formalizes state abstraction for tree search and presents our main theoretical result, which decomposes the regret due to abstraction into three components linked to different properties of the abstraction and the search algorithm. In Section 4, we show how two major categories of SBTS algorithms can be modified to exploit fixed state abstractions. We then present and analyze the PARSS algorithm (Section 5), and describe several variations of PARSS based on different abstraction refinement strategies (Section 6). Section 7 reviews related work. We then present our experiments with abstract tree search algorithms and their results in Sections 8 and 9. We conclude with a summary and directions for future work in Section 10.

2. Background

Our work focuses on sample-based tree search algorithms for anytime online planning in Markov decision processes. We incorporate MDP state abstraction and progressive abstraction refinement into SBTS algorithms to improve their anytime performance. This section introduces our notation for MDPs and describes the two categories of SBTS algorithms that we will adapt to create abstract tree search algorithms.

2.1 Markov Decision Processes

We consider MDPs of the form $M = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are finite sets of states and actions, $P(s'|s,a)$ is the transition probability function, $R(s)$ gives the instantaneous reward in s,[1] and $\gamma \in [0,1]$ is the discount factor. We assume that rewards are bounded, and without loss of generality that they lie in the unit interval, $R(s) \in [0,1]$.

A solution of an MDP is a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maps each state to an action. The set of policies for an MDP M is denoted $\Pi(M)$. An *episode* following policy π starting from state s_0 generates a sequence of states $s_0 s_1 \ldots$ where each $s_i \sim P(\cdot|s_{i-1}, \pi(s_{i-1}))$ and a corresponding sequence of rewards $r_t = R(s_t)$. The *value* of a policy is the expected discounted sum of future rewards when following the policy,

$$V^\pi(s) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r_t \Big| s, \pi\Big],$$

where the expectation is over episodes of π sampled from P. The value function is more commonly expressed in the equivalent recursive form

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s').$$

We require that the value is bounded, meaning that there exist finite constants $V_{\min}$ and $V_{\max}$ such that $V_{\min} \leq V^\pi(s) \leq V_{\max}$ for all $s \in \mathcal{S}$ and $\pi \in \Pi$. We exploit the assumption of bounded rewards to derive these value bounds. A trivial lower bound is $V_{\min} = 0$. When $\gamma < 1$, we have the upper bound $V_{\max} = \sum_{t=0}^{\infty} \gamma^t$. If $\gamma = 1$, this series diverges, so we require M to be a finite horizon MDP, meaning that there exists a finite constant D such that $t \geq D \Rightarrow r_t = 0$ with probability 1 for any π. In the finite horizon case, $V_{\max} = D$ is an upper bound.

A policy π is *optimal* if $V^\pi = V^*$, where V^* is the optimal value function

$$V^*(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s').$$

The optimal policy is *greedy* with respect to the optimal action-value function Q^*, meaning that $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$ where

$$Q^*(s,a) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s').$$

One can also define the Q-function of an arbitrary policy

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\pi(s').$$

Many MDP algorithms, including the ones we consider, estimate the optimal policy by estimating Q^* and behaving greedily with respect to that estimate.

1. The reward is often defined as a function of both the state and action, i.e. $R(s,a)$. All of our discussion applies to this case as well with trivial modifications, and we focus on the shorter form $R(s)$ for brevity. We found it convenient to implement both forms in our algorithms: the state-dependent part $R(s)$ models the "desirability" of a state (such as a goal state), while the action-dependent part $R(s,a)$ models action rewards (or costs).

2.2 Anytime Online Planning

Online planning is a family of solution techniques united by the common theme of planning for only one state at a time, as those states are encountered during execution. A prototypical OP algorithm is policy rollout (Bertsekas & Castañon, 1999). Policy rollout improves a fixed policy π by implementing a new policy

$$\pi_{\text{pr}}(s) = \arg\max_{a \in \mathcal{A}} \hat{Q}^{\pi}(s, a) \tag{1}$$

that acts greedily with respect to an estimate of the Q-function of π. More sophisticated online planning algorithms can be thought of as replacing $\hat{Q}^{\pi}(s, a)$ with a different action ranking function. The parallel rollout algorithm (Chang, Givan, & Chong, 2004) replaces the single rollout policy with a set of rollout policies. Policy switching (Chang et al., 2004; King, Fern, & Hostetler, 2013) is a similar algorithm that chooses the action prescribed by the best policy in the policy set rather than the greedy action with respect to the estimated Q-function.

Because online planning methods compute a policy only for those states that are encountered during execution, their complexity is generally independent of the size of the state space. This makes online planning a good fit for problems in which the state space is large, especially if good decisions can be made based on local exploration. It is also straightforward to incorporate diverse kinds of prior knowledge into the online planning framework, including expert policies, action preferences, and state evaluation heuristics.

In *anytime* online planning (AOP), we require that the planning algorithm can be halted at any time to produce an answer. Typically, an AOP algorithm computes an approximate solution quickly and then improves the solution incrementally until the algorithm is stopped. Anytime algorithms are useful when the amount of time available for deliberation is unknown or uncertain. For example, they can be combined with metareasoning algorithms for allocating deliberation time across multiple decisions, such as when playing chess or *go* with a full-game time limit.

2.3 Sample Based Tree Search

Sample-based tree search algorithms work by sampling state-action *histories* starting from the current state s_0 and gathering statistics of those history samples into a search tree. The search tree is used to guide further sampling and ultimately to estimate the optimal Q-function $Q^*(s_0, \cdot)$ in the root state. The specifics of how sampling is organized and how the estimate of Q^* is obtained differentiate different algorithms.

We begin this section by introducing the notion of an MDP over state-action *histories*. The history MDP will be the basis of our formal descriptions of SBTS algorithms. We then introduce two major paradigms for SBTS: sparse sampling and trajectory sampling.

2.3.1 MDPs over State-Action Histories

Given an MDP $M = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ and a designated state $s_0 \in \mathcal{S}$, the set of state-action *histories* beginning in s_0 is the set $\mathcal{H}^{\infty}(M, s_0) = \{s_0\} \times \mathcal{A} \times \mathcal{S} \times \cdots$. Note that $\mathcal{H}^{\infty}(M, s_0)$ may be infinite even though $\mathcal{S}$ is finite. The set of histories of length at most d is denoted $\mathcal{H}^{d}(M, s_0)$. Given a history $h = s_0 a_0 s_1 \ldots a_{t-1} s_t$, we write $s(h) \stackrel{\text{def}}{=} s_t$ and $a(h) \stackrel{\text{def}}{=} a_{t-1}$ for

the final state and action in the history, $p(h) \stackrel{\text{def}}{=} s_0 a_0 s_1 \dots a_{t-2} s_{t-1}$ for the prefix of h, and $\ell(h) \stackrel{\text{def}}{=} t$ for the length of h. The prefix of h of length k is denoted $h_k \stackrel{\text{def}}{=} s_0 a_0 s_1 \dots a_{k-1} s_k$.

A *history MDP* is an MDP $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ whose state space $\mathcal{H}$ is a subset of $\mathcal{H}^\infty(M, s_0)$ for the ground MDP M with the restriction that $h \in \mathcal{H} \Rightarrow p(h) \in \mathcal{H}$. The dynamics of T are given by overloading the P and R functions to apply to histories,

$$P(h'|h,a) \stackrel{\text{def}}{=} \mathbb{1}_{p(h')=h} \mathbb{1}_{a(h')=a} P(s(h')|s(h),a),$$
$$R(h) \stackrel{\text{def}}{=} R(s(h)),$$

where $\mathbb{1}_\phi$ is the indicator function defined by $\mathbb{1}_\phi = 1$ if ϕ holds and 0 otherwise.

A policy π for a history MDP maps histories to actions, $\pi : \mathcal{H} \mapsto \mathcal{A}$. The set of all policies for T is denoted $\Pi(T)$. We overload the value functions V and Q for history MDPs in the obvious way,

$$V^\pi(h) = R(h) + \gamma \sum_{h' \in \mathcal{H}} P(h'|h, \pi(h)) V^\pi(h'),$$
$$Q^\pi(h,a) = R(h) + \gamma \sum_{h' \in \mathcal{H}} P(h'|h,a) V^\pi(h').$$

The state transition graph of a history MDP is a tree, and thus the search trees generated by lookahead search algorithms can be viewed as finite history MDPs. The classical expectimax search algorithm (Russell & Norvig, 2010), for example, solves the history MDP $\langle \mathcal{H}^d, \mathcal{A}, R, P, \gamma, s_0 \rangle$ exactly for a fixed depth d. Because of our focus on tree search algorithms, we will deal almost exclusively with history MDPs in this paper.

2.3.2 Sparse Sampling

Sparse sampling (Kearns et al., 2002) is a systematic approach to tree search. In the SS algorithm, each action is sampled C times in the root state s_0, yielding $|\mathcal{A}| \cdot C$ successors, some of which may be duplicates. Sampling is then carried out recursively in each successor state, and this is continued until the tree has uniform depth d (Figure 1a). The constants C and d can be chosen independently of the size of the state space to achieve bounded error in the root state Q estimates, which ensures that the greedy action choice at the root is near-optimal with high probability. Sparse sampling is essentially an approximate expectimax search in which the transition distribution at each node is approximated by an empirical distribution of C samples.

As suggested by Kearns et al. (2002), the practical sample complexity of SS can be improved by incorporating a pruning mechanism. Forward search sparse sampling (FSSS) (Walsh, Goschin, & Littman, 2010) realizes this idea. FSSS constructs the SS tree incrementally by expanding nodes along one path from the root node to a leaf node at a time. The choice of path is guided by upper and lower bounds on the value estimate of the full SS tree, in a manner similar to Bounded Real-Time Dynamic Programming (BRTDP) (McMahan, Likhachev, & Gordon, 2005). The value bounds allow FSSS to avoid sampling portions of the tree that cannot affect the choice of action in the root state, while providing the same worst-case guarantees as SS.

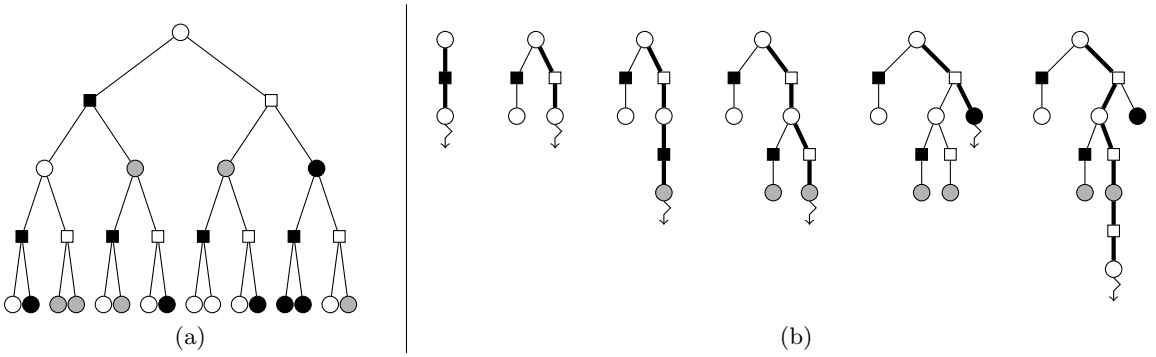


Figure 1: (a) A sparse sampling (SS) tree of width $C = 2$ and depth $d = 2$. Circles and squares represent states and actions respectively, and shading indicates their identity. We show a tree of full width for exposition; duplicate sibling state nodes typically would be merged if present. (b) The first six trajectories of a UCT-like trajectory sampling (TS) algorithm. Actions are chosen according to tree statistics until the trajectory reaches a novel state. That state is then added as a new leaf, and the trajectory continues as a random rollout (represented by the wavy arrow). The sampling policy is biased toward paths with higher value estimates, resulting in a tree of non-uniform depth.

2.3.3 Trajectory Sampling

Trajectory sampling (TS) algorithms build a sample tree from complete trajectories of a sampling policy. The sampling policy typically operates in two phases. During the *tree policy* phase, which begins in s_0 and continues until the trajectory reaches a leaf node, the sampling policy is based on statistics of the search tree combined with a mechanism to balance exploration and exploitation. Once the trajectory reaches a leaf node, a new successor node is added and the sampling policy switches to the *evaluation* phase. In the evaluation phase, an estimate of the new leaf's value is computed, typically either by sampling the return of a *rollout policy* or by evaluating a heuristic function. Search trees built in this way are not of uniform width and depth like SS trees (Figure 1b). Statistics of the nodes near the root will be based on many more samples than statistics of nodes near the leaves, and the search tree will be deeper in areas of the state space that are more likely to be reached under the sampling policy.

Compared to sparse sampling, trajectory sampling imposes weaker requirements on the generative model used for planning. Whereas SS algorithms require a *strong simulator*, capable of generating a sample from $P(\cdot|h, a)$ for any h and a, TS algorithms require only a *weak simulator*, which need only be capable of generating a complete episode following a fixed policy from the root state. This distinction has important implications when using state abstraction in search (Section 4).

The most well-known TS algorithm is UCT (Kocsis & Szepesvári, 2006). The term "Monte Carlo tree search" is often used to denote algorithms with the prototypical TS structure we described above (e.g. Browne et al., 2012). We use the term "trajectory sampling" to avoid ambiguity and to emphasize that building the tree out of *complete trajectories* is the distinguishing property of this class of algorithms.

3. State Aggregation Abstractions for Tree Search

Our focus in this paper is on improving the anytime performance of SBTS algorithms through the use of state abstraction. State abstraction, broadly speaking, includes any way of reducing the amount of information needed to describe the states of an MDP. We focus on the simplest form of MDP state abstraction, which is state aggregation (Li, Walsh, & Littman, 2006; Van Roy, 2006; Hostetler et al., 2014). State aggregation abstractions define abstract states as equivalence classes of ground states. In our history MDP setting, the "states" are histories, and so the abstract states are equivalence classes of ground histories in $\mathcal{H}$. An equivalence relation on the set of histories $\mathcal{H}$ is a binary relation $\chi \subseteq \mathcal{H} \times \mathcal{H}$ that is reflexive, symmetric, and transitive. We say that two histories h and g are equivalent with respect to χ, denoted $h \simeq_\chi g$, if and only if $\langle h, g\rangle \in \chi$. The equivalence class of a history h with respect to χ, denoted $[h]_\chi$, is the set $[h]_\chi = \{g \in \mathcal{H} : h \simeq_\chi g\}$. The quotient set of $\mathcal{H}$ by χ, denoted $\mathcal{H}/\chi$, is the set of equivalence classes of $\mathcal{H}$ with respect to χ. We use uppercase letters, e.g. $H \in \mathcal{H}/\chi$, to denote abstract histories, to emphasize that abstract histories are sets of ground histories.

In order to plan in the abstract state space, we need to define the dynamics of the abstract MDP in terms of the dynamics of the ground MDP. We do this by introducing a *weight function* $\mu : \mathcal{H}/\chi \times \mathcal{H} \mapsto [0, 1]$, where for each $H \in \mathcal{H}/\chi$, $\mu(H, \cdot)$ is a probability mass function over the ground states in H. The abstract dynamics $\mathcal{P}_\mu$ and $\mathcal{R}_\mu$ are defined as μ-weighted convex combinations of the ground dynamics,

$$\mathcal{P}_\mu(H'|H, a) = \sum_{h \in H} \mu(H, h) \sum_{h' \in H'} P(h'|h, a)$$
$$\mathcal{R}_\mu(H) = \sum_{h \in H} \mu(H, h) R(h).$$

A state abstraction, then, consists of two parts: an equivalence relation χ and a weight function μ.

Definition 1. A *history aggregation abstraction* (hereafter called a *state abstraction*) is a tuple $\langle \chi, \mu\rangle$ consisting of an *abstraction relation* χ and a *weighting function* μ, where χ is an equivalence relation on $\mathcal{H}$ satisfying[2] $h \simeq_\chi g \Rightarrow [p(h) \simeq_\chi p(g) \wedge a(h) = a(g)]$ and $\mu : \mathcal{H}/\chi \times \mathcal{H} \mapsto [0, 1]$ defines, for each equivalence class $H \in \mathcal{H}/\chi$, a probability mass function $\mu(H, \cdot)$ supported on H.

A state abstraction applied to a history MDP $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0\rangle$ induces an abstract MDP $T/\langle \chi, \mu\rangle = \langle \mathcal{H}/\chi, \mathcal{A}, \mathcal{P}_\mu, \mathcal{R}_\mu, \gamma, s_0\rangle$. Given an abstraction $\alpha = \langle \chi, \mu\rangle$, a policy π for the abstract problem $\mathcal{T} = T/\alpha$ maps abstract states to actions, $\pi : \mathcal{H}/\chi \mapsto \mathcal{A}$. The value of π in $\mathcal{T}$ is given by the abstract value function

$$\mathcal{V}^\pi_\alpha(H) = \mathcal{R}_\mu(H) + \gamma \sum_{H' \in \mathcal{H}/\chi} \mathcal{P}_\mu(H'|H, \pi(H)) \mathcal{V}^\pi_\alpha(H').$$

and the Q-function of π in $\mathcal{T}$ is given by

$$\mathcal{Q}^\pi_\alpha(H, a) = \mathcal{R}_\mu(H) + \gamma \sum_{H' \in \mathcal{H}/\chi} \mathcal{P}_\mu(H'|H, a) \mathcal{V}^\pi_\alpha(H').$$

2. The condition that $h \simeq_\chi g \Rightarrow [p(h) \simeq_\chi p(g) \wedge a(h) = a(g)]$ ensures that the state transition graph of $T/\langle \chi, \mu\rangle$ remains a tree.

The optimal abstract value functions are denoted $\mathcal{V}^*_\alpha$ and $\mathcal{Q}^*_\alpha$.

Each abstract policy π induces a ground policy $\downarrow\pi$ defined by

$$\downarrow\pi(h) = \pi([h]_\chi) \qquad \text{for all } h \in \mathcal{H}. \tag{2}$$

Using the induced policy, we can define the *ground* value function of an abstract policy π as $V^{\downarrow\pi}$. An abstraction α is *sound* if every optimal policy π^* for the abstract problem T/α induces a ground policy $\downarrow\pi^*$ that achieves the optimal value in the ground MDP, $V^{\downarrow\pi^*} = V^*$. Note that this need not imply that the *abstract* value of π is equal to the optimal ground value. All that is required is that the greedy action with respect to the abstract value function is the same as the greedy action with respect to V^*; that is $\arg\max_{a\in\mathcal{A}} \mathcal{Q}^*_\alpha(\{s_0\}, a) = \arg\max_{a\in\mathcal{A}} Q^*(s_0, a)$.

3.1 A Regret Bound for State Abstraction in Tree Search

Naturally, state abstraction introduces a new source of value estimation error. The magnitude of this abstraction error depends on the properties of the two components of the abstraction: the abstraction relation χ and the weighting function μ.

We consider the abstraction relation χ first. Following Hostetler et al. (2014), we define a class of state space partitions parameterized by $p, q \in \mathbb{R}^{\geq 0}$.

Definition 2. A partition $\mathcal{H}/\chi$ is (p, q)-*consistent* if for all $H \in \mathcal{H}/\chi$,

$$\exists a^* \in \mathcal{A} \;.\; \forall h \in H: \quad V^*(h) - Q^*(h, a^*) \leq p \tag{3}$$

$$\forall h, g \in H: \quad \left|V^*(h) - V^*(g)\right| \leq q. \tag{4}$$

An abstraction relation χ is (p, q)-consistent if and only if $\mathcal{H}/\chi$ is (p, q)-consistent.

The p condition (3) requires that in each abstract history $H \in \mathcal{H}/\chi$, there is an action a^* that is near-optimal in every ground history in $h \in H$. This bounds the loss from following an abstract policy that is constrained to play the same action in all equivalent ground histories. The q condition (4) requires that the optimal values of all ground histories $h \in H$ are close to one another. The value of q is related to the error in approximating the dynamics of the abstract process as a weighted average of the ground process dynamics of the different ground states in H.

The (p, q)-consistency property is a generalization of the a^*-irrelevance and π^*-irrelevance properties identified by Li et al. (2006) in their study of sound state aggregation abstractions. The π^*-irrelevance property, which requires that all aggregated states share an optimal action, is equivalent to $(0, \infty)$-consistency. The a^*-irrelevance property, which requires additionally that the Q^* values of the optimal action are the same in all aggregated states, is equivalent to $(0, 0)$-consistency. The coarsest abstraction satisfying π^*-irrelevance is also the coarsest sound abstraction, and the hierarchy of sound abstractions proposed by Li et al. (2006) consists of refinements of π^*-irrelevance.

Although $(0, \infty)$-consistent abstractions are sound, learning with these abstractions can be problematic in general MDPs. A simple example due to Li et al. (2006) illustrates the problem (Figure 2a). When s^1 and s^2 are aggregated into a single abstract state S, the value of S becomes non-Markovian. If S was reached via action a, then $\mathcal{V}^*(S) = 1$, while if S

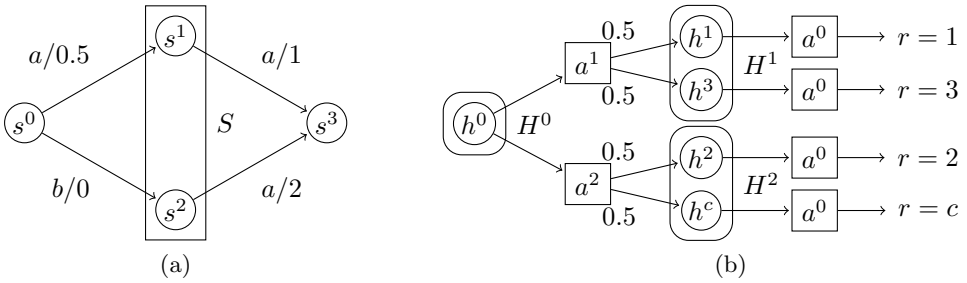


Figure 2: (a) An example of an MDP for which a $(0, \infty)$-consistent abstraction is unsound (Li et al., 2006). The edge labels like "$a/0.5$" mean action a yields immediate reward 0.5. (b) A history MDP for which a $(0, \infty)$-consistent abstraction $\langle \chi, \mu \rangle$ is unsound for some weight functions $\mu \neq \mu^*$. Edge labels denote transition probabilities.

was reached via action b, then $\mathcal{V}^*(S) = 2$. Nevertheless, the estimated abstract value $\mathcal{V}(S)$ must be a single number. The greedy policy with respect to this estimated value function will not be optimal because it will choose a in s^0 due to its larger immediate reward.

History aggregation abstractions (Definition 1) cannot create the structure in Figure 2a because they will not aggregate histories that result from different action sequences. Nevertheless, history aggregation abstractions are subject to a related problem if the weight functions are not correct, illustrated in Figure 2b. If the weight function μ is

$$\mu(H^1, h^1) = 0 \qquad \mu(H^2, h^2) = 1$$
$$\mu(H^1, h^3) = 1 \qquad \mu(H^2, h^c) = 0,$$

then $\mathcal{Q}^*_\alpha(H^0, a^1) = 3$ and $\mathcal{Q}^*_\alpha(H^0, a^2) = 2$, while the ground values are $Q^*(h^0, a^1) = 2$ and $Q^*(h^0, a^2) = c/2 + 1$. If on the other hand $\mu(H^1, \cdot) = \mu(H^2, \cdot) = [0.5, 0.5]$, then $\mathcal{Q}^*(H^0, \cdot) = Q^*(h^0, \cdot)$ and the abstraction is lossless.

The previous example illustrates the role of the weight function μ in determining the accuracy of an abstraction. Intuitively, the second choice of weight function is superior because it faithfully preserves the relative probability of the different ground histories that are aggregated in the abstract history. We denote the probability of a ground history by

$$P(h|s_0) = \mathbb{1}_{h_0 = s_0} \prod_{i=1}^{\ell(h)} P(h_i | h_{i-1}, a(h_i)).$$

Using this notation, we define the *optimal* weight function.

Definition 3. Let $T = \langle \mathcal{H}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, s_0 \rangle$ be a history MDP, h a ground history in $\mathcal{H}$, and $H \in \mathcal{H}/\chi$ an abstract history with respect to some abstraction relation χ. The *optimal* weight function for T, denoted μ^*_T (or μ^* if T is clear from context), weights ground states by the conditional probability of the transition $p(H) \to h$ given that the transition $p(H) \to H$

occurred:

$$\mu_T^*(H,h) = \mathbb{1}_{h\in H}\frac{\sum_{g\in p(H)} P(g|s_0)P(h|g,a(h))}{\sum_{f\in p(H)} P(f|s_0)\sum_{f'\in H} P(f'|f,a(H))}$$
$$= \mathbb{1}_{h\in H}\frac{\sum_{g\in p(H)} \mu_T^*(p(H),g)P(h|g,a(h))}{\mathcal{P}_{\mu_T^*}(H|p(H),a(H))}.$$

The recursive definition of μ^* in Definition 3 is especially natural in the tree search setting. We can view this recursive form as an operator W acting on a weight function μ to give an *exact update* of μ,

$$W\mu(H,h) = \frac{\sum_{g\in p(H)} \mu(p(H),g)P(h|g,a(h))}{\mathcal{P}_\mu(H|p(H),a(H))}. \tag{5}$$

Using this notation, μ^* is simply the weight function satisfying $\mu^* = W\mu^*$. We can now define the *single-step divergence* $\delta_\mathcal{T}(H)$ of state H in the abstract problem $\mathcal{T} = T/\langle\chi,\mu\rangle$,

$$\delta_\mathcal{T}(H) = \frac{1}{2}\Big\|\mu(H,\cdot) - W\mu(H,\cdot)\Big\|_1, \tag{6}$$

to quantify the error introduced by μ in the single step $p(H)\to H$. The *divergence* of $\mathcal{T}$ is the maximum of $\delta_\mathcal{T}$,

$$\delta_\mathcal{T} = \max_{H\in\mathcal{H}/\chi} \delta_\mathcal{T}(H),$$

which bounds the error due to μ across all abstract states.

The simple regret due to acting in ground state h according to the optimal abstract policy π^* with respect to abstraction $\alpha = \langle\chi,\mu\rangle$ is given by

$$J_\alpha(h) = \max_{a\in\mathcal{A}} Q^*(h,a) - Q^*(h,{\downarrow}\pi^*(h)).$$

We now present our main theoretical result, which shows that for (p,q)-consistent abstractions, this regret can be bounded. The statement of the theorem involves the "discounted" depth of the MDP, given by the sum of the first d powers of the discount factor,

$$\beta_\gamma(d) = \sum_{i=1}^{d}\gamma^d.$$

We actually prove the following stronger result, which shows that when we estimate the value of *any* action using the abstract action-value function $\mathcal{Q}_\alpha^*$, the error in that estimate due to abstraction is bounded.

Theorem 1. *Let $T = \langle\mathcal{H},\mathcal{A},P,R,\gamma,s_0\rangle$ be a history MDP such that the maximum length of a history in $\mathcal{H}$ is $d = \max_{h\in\mathcal{H}}\ell(h)$. Let $\alpha = \langle\chi,\mu\rangle$ be an abstraction of T where χ is (p,q)-consistent and let $\delta \stackrel{\text{def}}{=} \delta_{T/\alpha}$. For any action $a\in\mathcal{A}$,*

$$\Big|Q^*(s_0,a) - \mathcal{Q}_\alpha^*(\{s_0\},a)\Big| \le \beta_\gamma(d)(p+\delta q).$$

Proof. Appendix A.1. □

Our desired bound is an immediate consequence of Theorem 1.

Corollary 1. *The simple regret due to acting in the ground problem greedily with respect to $\mathcal{Q}^*_\alpha$ is bounded by*

$$J_\alpha(h) \leq 2\beta_\gamma(d)(p + \delta q).$$

Proof. Let $a^* = \arg\max_{a \in \mathcal{A}} Q^*(h, a)$ and let $\hat{a} = \arg\max_{a \in \mathcal{A}} \mathcal{Q}^*_\alpha(h, a)$. Acting greedily with respect to $\mathcal{Q}^*_\alpha$ results in error when $a^* \neq \hat{a}$, which occurs when $\mathcal{Q}^*_\alpha(s_0, a^*) < \mathcal{Q}^*_\alpha(s_0, \hat{a})$. By Theorem 1, in the worst case we have $\mathcal{Q}^*_\alpha(s_0, \hat{a}) = Q^*(s_0, \hat{a}) + \beta_\gamma(d)(p + \delta q)$ and $\mathcal{Q}^*_\alpha(s_0, a^*) = Q^*(s_0, a^*) - \beta_\gamma(d)(p + \delta q)$, for a combined error of $2\beta_\gamma(d)(p + \delta q)$. □

It is apparent that if χ is a $(0, 0)$-consistent abstraction relation, then for any weight function μ, $J_{\langle \chi, \mu \rangle} = 0$. This mirrors the result for general MDPs of Li et al. (2006) that the optimal abstract policy with respect to an a^*-irrelevance abstraction induces an optimal ground policy. If χ is $(0, q)$-consistent for $q > 0$, then we can still achieve zero error if we have the optimal weight function μ^*, since in that case $\delta q = 0$. Thus we see that π^*-irrelevance abstractions in the tree search setting have more-favorable properties compared to the general MDP setting. Namely, for any π^*-irrelevance abstraction χ there is a weight function μ^* such that the abstract value with respect to abstraction $\alpha = \langle \chi, \mu^* \rangle$ is equal to the ground value, that is $\mathcal{Q}^*_\alpha(\{s_0\}, \cdot) = Q^*(s_0, \cdot)$.

Note that the bound of Theorem 1 is a formal bound, since actually computing p, q, and $\delta_\mathcal{T}$ would require solving the MDP. The purpose of Theorem 1 is to separate the different sources of abstraction error and provide guidance for designing or computing good abstractions. For example, we use it to design an abstraction refinement heuristic in Section 6.2.2.

3.2 Abstract MDPs as Partially Observable MDPs

Bai, Srivastava, and Russell (2015), in their work on abstraction in MCTS, take the view that an abstract MDP is a partially observable Markov decision process (POMDP), where the abstract states are *observations* that give us information about the hidden ground state, and the weight function μ plays the role of a belief distribution. Our goal in this section is to show how to translate between the two formalisms.

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, P, R, \Omega, \gamma \rangle$. The components $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ define an ordinary discounted MDP. The set $\mathcal{Z}$ is the set of observations, and Ω gives the conditional probability of an observation given a state, $\Omega(z|s) : \mathcal{Z} \times \mathcal{S} \mapsto [0, 1]$. A policy for a POMDP cannot observe the state. Instead a policy is a mapping from an observation-action sequence $z_0 a_0 z_1 \ldots z_k$ to an action.

Consider a history MDP $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ and an abstraction $\alpha = \langle \chi, \mu \rangle$. The abstract MDP T/α can be defined as a POMDP as follows. The state space is the set of ground histories $\mathcal{H}$, and the actions and dynamics are as in T. The observation set is the set of abstract histories $\mathcal{Z} = \mathcal{H}/\chi$. Finally the observation function is $\Omega(H|h) = \mathbb{1}_{h \in H}$.

The weight function μ appears via the definition of the belief state in the POMDP. A POMDP has an equivalent formulation as a (fully observed) MDP over a continuous state space called the belief space that represents the probability of being in each state given an observation history. Let B denote the belief set. Its elements $b \in B$ are probability measures on the state set, $b : \mathcal{S} \mapsto [0, 1]$. The belief update operation F maps a belief b and

an action-observation pair $\langle a, z\rangle$ to a new belief $F(b, a, z)$ defined by

$$F(b,a,z)(s') = \frac{\sum_{s\in\mathcal{S}} b(s)\Omega(z|s')P(s'|s,a)}{\sum_{s\in\mathcal{S}} b(s)\sum_{s''\in\mathcal{S}}\Omega(z|s'')P(s''|s,a)}.$$

Compare this to Equation 5, which when the notation is expanded reads

$$W\mu(H,h) = \frac{\sum_{g\in p(H)}\mu(p(H),g)P(h|g,a(h))}{\sum_{g\in p(H)}\mu(p(H),g)\sum_{h\in H}P(h|g,a(h))}.$$

Remembering that $\Omega(z|s)$ is just the indicator of whether s is "in" z, it is apparent that the two updates are equivalent, with μ playing the role of the belief state b. Thus in the POMDP view, $W\mu$ is an exact belief update of the belief μ, and the δ term in Theorem 1 is a measure of inaccuracy in belief updating.

Viewing abstraction in this way exposes a strong connection to POMDP solution methods. Since the belief space of a POMDP is continuous and high-dimensional, a common solution approach is to search in a structured space of policies whose complexity can be controlled (e.g. Hansen, 1998; Meuleau, Kim, Kaelbling, & Cassandra, 1999; Poupart & Boutilier, 2003). One can view abstract tree search algorithms as searching for an evaluation policy within the set of tree-shaped finite-state controllers that have one state for each history equivalence class under the abstraction relation χ. Unlike in these works on POMDPs, which seek a structured policy that is effective over the entire reachable portion of the belief space, in abstract tree search the policy is used only to evaluate the current state and thus need only be effective locally.

4. Tree Search Algorithms using Fixed Abstractions

Adapting tree search algorithms to use state abstraction is straightforward. The main complication is that we need to sample state transitions in the abstract problem $\mathcal{T} = T/\alpha$, but we have access only to a simulator of the ground problem T. In this section, we describe versions of SS and TS algorithms that sample abstract search trees given an abstraction relation χ and a simulator of the ground problem. Ideally, these algorithms would search in the abstract problem $T/\langle\chi, \mu^*\rangle$ with respect to the optimal weight function μ^*, since then the δq term of the abstraction error would be 0 (Theorem 1). Our analysis will show that this can be achieved in trajectory sampling (TS) algorithms, but generally not in sparse sampling (SS) algorithms.

4.1 Framework and Notation

We will view abstract SBTS algorithms as producing two structures. The first is the *sample tree*, which is a search tree in the ground state space constructed in a similar fashion to non-abstract SBTS algorithms. The second is the *abstract tree*, which is the tree that results from applying some abstraction $\langle\chi, \mu\rangle$ to the sample tree and that is used to guide sampling decisions.

4.1.1 The Sample Tree

The sample tree is a multiset of ground histories, defined by the multiplicity function $n : \mathcal{H} \mapsto \mathbb{Z}^{\geq 0}$ giving the number of times that each history $h \in \mathcal{H}$ has been sampled. We

Algorithm 1 Abstract SBTS Framework

1: **procedure** AbstractTreeSearch(s_0)
2: **while** not converged **do**
3: Choose sampling actions according to statistics of the abstract tree N
4: Draw samples from the ground simulator and add them to the sample tree n
5: Update the structure and statistics of N

will sometimes treat n as an ordinary set, in which case we will write $h \in n$ if and only if $n(h) > 0$. Conceptually, the sample tree is a bipartite tree consisting of *state nodes* and *action nodes*. State nodes correspond to histories $h \in \mathcal{H}$, while action nodes correspond to a history-action pair. We denote action nodes by juxtaposing a history and an action like ha.

The tree structure of the sample tree is described by the successor relation k_n, which maps each state node $h \in n$ and action $a \in \mathcal{A}$ to a set of successors,

$$k_n(h, a) = \{h' \in n : p(h') = h, a(h') = a\}.$$

State nodes that have no successors are called *leaf nodes*. The sample count for action nodes, denoted $m_n(h, a)$, is given in terms of k_n as

$$m_n(h, a) = \sum_{h' \in k_n(h,a)} n(h').$$

We will normally omit the n subscripts when n is clear from context. Particular algorithms will also record other statistics of the tree, which we will denote similarly as functions taking histories as arguments.

4.1.2 The Abstract Tree

The second product of abstract SBTS — the *abstract tree* — is the quotient multiset $N = n/\chi$ obtained by partitioning n according to an abstraction relation χ. The multiplicity function of the quotient multiset is denoted $N : \mathcal{H}/\chi \mapsto \mathbb{Z}^{\geq 0}$ and is defined by

$$N(H) = \sum_{h \in H} n(h) \quad \forall H \in \mathcal{H}/\chi.$$

As before, we write $H \in N$ if and only if $N(H) > 0$. The successor relation K_N and the action sample count $M_N(H, a)$ for the abstract tree are defined analogously to those for the sample tree,

$$K_N(H, a) = \{H' \in N : p(H') = H, a(H') = a\},$$
$$M_N(H, a) = \sum_{H' \in K_N(H,a)} N(H'),$$

and as before we will usually omit the N subscripts.

We can now outline a generic abstract tree search algorithm (Algorithm 1). Sampling decisions are made according to the abstract tree (Line 3), but the ground samples are retained in the sample tree (Line 4). In the following sections we instantiate this algorithm skeleton to obtain abstract versions of SS and TS algorithms.

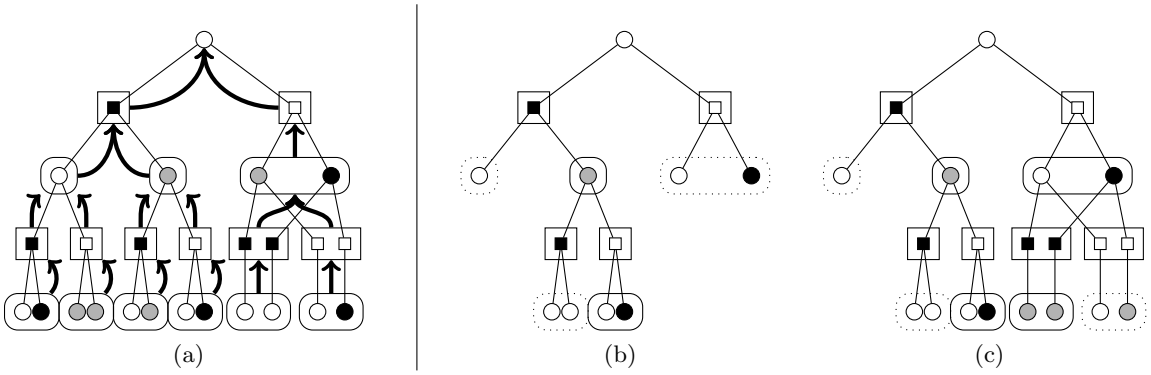


Figure 3: (a) An AbstractSS tree of width $C = 2$ and depth $d = 2$. Ground nodes in the *sample tree* are aggregated into abstract nodes in the *abstract tree*, but the structure of the sample tree is retained. The arrows show how value estimates propagate in the abstract tree. (b) (c) Two iterations of AFSSS. The abstract state nodes with dotted borders have not been expanded yet, and their upper and lower value bounds are initialized to $V_{\max}$ and $V_{\min}$ respectively.

4.1.3 The Abstraction Relation

In the tree search setting, it is natural to decompose the monolithic abstraction relation χ, which is defined on histories, as a collection of abstraction relations $\chi(H, a)$ on the ground state space $\mathcal{S}$. For each abstract action node Ha in the abstract tree, its abstract successors are the abstract histories HaS, where each $S \in \mathcal{S}/\chi(H, a)$. The equivalence class of a ground history $h = s_0 a_0 s_1 \dots a_{d-1} s_d$ is the set

$$[h]_\chi = S_0 a_0 S_1 \dots a_{d-1} S_d, \qquad \text{where } S_i = [s_i]_{\chi(S_{i-1}, a_{i-1})}.$$

Any history aggregation abstraction satisfying the constraints of Definition 1 can be represented in this fashion. Naturally, some or all of these component abstraction relations could be the same. Decomposing the abstraction in this manner facilitates making "local" refinements to the abstraction (Section 5).

4.2 Abstract Sparse Sampling

Sparse sampling (Kearns et al., 2002) builds a sample tree of uniform width and depth. It is a systematic approach to tree search, in the sense that the amount of sampling that takes place in different regions of the state space is not related to the probability of reaching those regions from the start state under any particular policy. To accomplish this type of sampling, it is necessary to sample *transitions* $\langle h, a, h', r \rangle$ from the single-step dynamics P and R. Sparse sampling draws a constant number C of transition samples for each action $a \in \mathcal{A}$ recursively in every state node h such that $\ell(h) < d$. The algorithm achieves small error with high probability with a sample complexity that does not depend on the size of the state space $|\mathcal{S}|$ (Kearns et al., 2002, Thm. 1). The AbstractSS algorithm (Algorithm 2) employs the same systematic sampling strategy, but it operates in the abstract state space.

Algorithm 2 Abstract Sparse Sampling

procedure AbstractSS(s_0, C, d, χ)
  global C, χ
  Expand($\{s_0\}$, d)
  return $\arg\max_{a \in \mathcal{A}} \mathcal{Q}(\{s_0\}, a)$
procedure Expand(H, d)
  if H is terminal **or** $d = 0$ **then**
    $\mathcal{Q}(H, a) \leftarrow \mathcal{R}_{\bar{\mu}}(H)$ for all $a \in \mathcal{A}$
  else
    for all $a \in \mathcal{A}$ **do**
      Sample(H, a)
      for all $H' \in K(H, a)$ **do**
        Expand(H', $d-1$)
      $\mathcal{Q}(H, a) \leftarrow \mathcal{R}_{\bar{\mu}}(H) + \gamma \sum_{H' \in K(H,a)} \frac{N(H')}{C} \max_{a' \in \mathcal{A}} \mathcal{Q}(H', a')$
procedure Sample(H, a)
  for C times **do**
    Let $h \sim \bar{\mu}(H, \cdot)$, where $\bar{\mu}(H, h) = \mathbb{1}_{h \in H} \frac{n(h)}{N(H)}$
    Let $h' \sim P(\cdot | h, a)$
    $n(h') \leftarrow n(h') + 1$

To implement AbstractSS, we need to sample transitions from $\mathcal{P}_\mu$ for some μ. We would like to sample from $\mathcal{P}_{\mu^*}$, but in general we will have to settle for a sampled approximation of μ^*. The obvious choice is the empirical probability of the ground histories $h \in H$,

$$\bar{\mu}(H, h) = \mathbb{1}_{h \in H} \frac{n(h)}{N(H)}. \tag{7}$$

We will specify AbstractSS in terms of $\bar{\mu}$, but note that better estimators may be available for particular problem domains.

Because AbstractSS must estimate μ^*, the algorithm might introduce abstraction error via the δ term in Theorem 1. We analyze AbstractSS by separating the error due to finite sampling from the error due to abstraction. In effect, AbstractSS is performing ordinary sparse sampling in an abstract MDP for which we can characterize the abstraction error. We can thus apply the same finite sample analysis as Kearns et al. (2002) employed for SS in order to characterize the sampling error in AbstractSS.

There is a small technical difficulty in this analysis, which is that the abstract value function $\mathcal{V}^*_{\langle \chi, \bar{\mu} \rangle}$ is not well-defined because $\bar{\mu}$ is only defined over a subset of the abstract history set $\mathcal{H}/\chi$. We work around this by introducing a "completed" weight function $\bar{\mu}^+$ that is defined over the entire state space. Let $\mathbf{dom}(\bar{\mu}) \subseteq \mathcal{H}/\chi$ be the subset of the abstract history set on which $\bar{\mu}$ is defined. Then $\bar{\mu}^+$ is given by

$$\bar{\mu}^+(H, h) = \begin{cases} \bar{\mu}(H, h) & \text{if } H \in \mathbf{dom}(\bar{\mu}), \\ \mu^*(H, h) & \text{otherwise.} \end{cases} \tag{8}$$

Clearly for any state node $H \in \mathbf{dom}(\bar{\mu})$, we have $\mathcal{P}_{\bar{\mu}}(\cdot|H,a) = \mathcal{P}_{\bar{\mu}^+}(\cdot|H,a)$ for any $a \in \mathcal{A}$. We will denote the completed abstraction as $\alpha^+ = \langle \chi, \bar{\mu}^+ \rangle$. Note that while $\bar{\mu}^+$ is defined in terms of the exact weight function μ^*, we use this fact only for our analysis; AbstractSS does not actually compute μ^*. Intuitively, since the sampled tree does not contain histories that are not in $\mathbf{dom}(\bar{\mu})$, abstraction error in these states does not affect the value estimate and we can assume that the error is as small as possible.

The analysis of Kearns et al. (2002) also requires an upper bound on the value achievable in the problem. Define the quantity $V^d_{\max}$ to be an upper bound on the value function $V^*(h)$ for all histories $h \in \mathcal{H}$ of length $\ell(h) = d$. Since our rewards are bounded in $[0,1]$, one possible definition of $V^d_{\max}$ is

$$V^d_{\max} = \begin{cases} \sum_{t=0}^{\infty} \gamma^t & \text{if } \gamma < 1, \\ D - d & \text{if } \gamma = 1, \end{cases}$$

where D is the maximum length of a trajectory in a finite-horizon problem.

We now have the tools we need to derive the following formal guarantee on the performance of AbstractSS.

Proposition 2. *Let $T = \langle \mathcal{H}^\infty, \mathcal{A}, P, R, \gamma, s_0 \rangle$ be a (possibly infinite) history MDP and let χ be a (p,q)-consistent history equivalence relation on $\mathcal{H}^d$ for some depth parameter d. Then the procedure* AbstractSS(s_0, C, d, χ)*, with probability at least $1 - (|\mathcal{A}|C)^d \cdot 2e^{-2\lambda^2 C/(V^d_{\max})^2}$, returns an action choice a^* such that*

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\Big[\beta_\gamma(d)(\lambda + p + \delta q) + \gamma^d V^d_{\max}\Big],$$

where δ is the divergence (Eq. 6) of the completed empirical weight function $\bar{\mu}^+$ derived from the empirical weight function $\bar{\mu}$ computed by AbstractSS.

Proof. Appendix A.2. □

This result combines Theorem 1 with the sample complexity result for sparse sampling proven by Kearns et al. (2002). It shows that AbstractSS achieves the same error bounds as running ordinary SS in the abstract problem $T/\langle \chi, \bar{\mu} \rangle$, despite $\bar{\mu}$ being computed "on the fly" by AbstractSS rather than being fixed beforehand. The λ term accounts for error due to finite sampling, while the $\gamma^d V^d_{\max}$ term accounts for error due to truncating the search tree to a depth of d.

4.3 Abstract Forward Search Sparse Sampling

Forward Search Sparse Sampling (FSSS) (Walsh et al., 2010) is an enhancement of SS that incorporates pruning based on upper and lower bounds on the values of tree nodes. It provides the same performance guarantees as SS and often performs less computation. Abstract FSSS (AFSSS; Algorithm 3) is a straightforward extension of FSSS. In addition to the data structures required by AbstractSS, each abstract state node H in the AFSSS tree has associated upper and lower value bounds $U(H)$ and $L(H)$, and each action node Ha has similar bounds $U(H,a)$ and $L(H,a)$. These bounds bracket the value estimate that would be computed for the corresponding node in a full SS tree.

To understand the principle of AFSSS, we first consider an AbstractSS(s_0, C, d, χ) tree G rooted at s_0. AbstractSS acts greedily with respect to the value estimate $\mathcal{Q}_G(\{s_0\}, \cdot)$ obtained from the search tree. The abstract tree G is a random variable, and with some abuse of notation we will say that $G \sim$ AbstractSS. Let Σ denote the sample space of such abstract trees.

The AFSSS algorithm also samples an abstract tree from AbstractSS, but it does so incrementally through a sequence of top-down *trials*. These trials build up an abstract tree N. Figures 3b and 3c illustrate a possible result of two such trials. For any such tree N, only a subset of the trees in Σ contain N as a subtree. We say that N is a *subtree* of another abstract tree G if for all $H \in N$, $N(H) = G(H)$ (recall that N and G are multisets of abstract histories). Let Σ_N denote the set of all trees in Σ of which N is a subtree. AFSSS keeps adding nodes to N until all of the trees $G \in \Sigma_N$ are such that their value estimates $\mathcal{Q}_G(\{s_0\}, \cdot)$ prescribe the same optimal action in the root state s_0.

The upper and lower value bounds allow us to detect when this condition has been met. We say that the search has *converged* if

$$L(H_0, a^*) \geq \max_{a \neq a^*} U(H_0, a), \tag{9}$$

where $a^* = \arg\max_{a \in \mathcal{A}} L(H_0, a)$. Since the lower bound on the value of a^* exceeds the upper bounds on the values of all other actions, we know that a^* will remain the chosen action if the search tree is built to completion. For this early stopping criterion to be sound, the value bounds L and U must bracket the value estimate that AbstractSS *would* compute if the search tree were completely filled out. We now define a condition on L and U that ensures that this is the case. We call this condition *admissibility*, but note that this definition is different from the typical definition of admissibility employed in algorithms like A^* search.

Definition 4. Let Σ denote the set of AbstractSS(s_0, C, d, χ) trees of width and depth C and d, under abstraction relation χ, and rooted at s_0. Let N be an abstract tree and let Σ_N be the subset of trees in Σ of which N is a subtree. A pair of upper and lower state value bounds $U, L : N \mapsto \mathbb{R}$ is *abmissible with respect to* N if for all $H \in N$,

$$L(H) \leq \mathcal{V}_G(H) \leq U(H) \text{ for all } G \in \Sigma_N,$$

where $\mathcal{V}_G$ is the value function estimate obtained from the search tree G. A pair of upper and lower action value bounds $U, L : N \times \mathcal{A} \mapsto \mathbb{R}$ is *admissible with respect to* N if for all $H \in N$ and all $a \in \mathcal{A}$,

$$L(H, a) \leq \mathcal{Q}_G(H, a) \leq U(H, a) \text{ for all } G \in \Sigma_N.$$

If the bounds U and L are admissible, then further sampling after convergence cannot change the estimate of the optimal action in the root state H_0. Any un-expanded portions of the search tree at the time of convergence are effectively pruned away without being sampled. Due to this pruning, AFSSS can give the same worst-case performance guarantees as AbstractSS while often using fewer samples in practice.

The next definition formalizes the structural features of an abstract FSSS tree. Note that we continue to assume the use of the empirical weight function $\bar{\mu}$ (Eq. 7).

Definition 5. An *abstract FSSS tree with respect to* χ of width C and depth d, or a χ-*FSSS(C, d) tree*, is a tuple $\mathcal{F} = \langle N, L, U, H_0, \chi \rangle$, where N is an abstract tree, L and U are lower and upper value bound functions, $H_0 \in N$ is the root state, and χ is an abstraction relation, such that all of the following conditions are satisfied:

1. For each abstract history $H \in N$, $\forall h, g \in H$, $h \simeq_\chi g$;
2. For each abstract history $H \in N$, if $expanded(H)$ then $M(H, a) \geq C$ for all $a \in \mathcal{A}$;
3. L and U are admissible with respect to N (Definition 4);
4. $\mathcal{F}$ satisfies the convergence criterion (9).

If $\mathcal{F}$ satisfies at least conditions 1, 2, and 3, then $\mathcal{F}$ is a *partial χ-FSSS tree*.

The AFSSS algorithm (Algorithm 3) constructs a χ-FSSS(C, d) tree for a fixed abstraction relation χ. Like FSSS, AFSSS proceeds in a series of top-down trials that each traverse a path from the root node to a leaf state node. When extending a path, the algorithm chooses action nodes optimistically (Line 13), and chooses state nodes with the largest gap between U and L (Line 14). If the path reaches an unvisited state node (Line 12), that node is *expanded* by initializing and sampling its action node successors. The backup operation (Line 30) combines the average immediate reward over ground states with the discounted future return bounds over abstract states. The additional parameters to AFSSS are the sparse sampling width and depth C and d, admissible values bounds $V_{\min}$ and $V_{\max}$, and a *default abstraction* $\chi_0 \subseteq \mathcal{S} \times \mathcal{S}$ that is used to initialize $\chi(H, a)$ when expanding a new state node H. Note that χ_0 is a relation on the state set $\mathcal{S}$, not $\mathcal{H}$.

The AFSSS implementation in Algorithm 3 is generalized to accept a partial χ-FSSS(C, d) tree as input and transform it into a converged χ-FSSS(C, d) tree. Starting from a partial tree allows us to use AFSSS without major changes as a building block of the PARSS algorithm that we will introduce next (Section 5.2). To build an abstract FSSS tree from scratch, one calls AFSSS with an empty tree as input. Given an initial state s_0 and admissible value bounds $V_{\min}$ and $V_{\max}$, the empty abstract FSSS tree is defined by

$$\begin{aligned} \mathcal{F}_0(s_0, V_{\min}, V_{\max}) &= \langle N_0, L, U, H_0, \chi \rangle, \\ \text{where } \; H_0 &= \{s_0\}, N_0(H) = \mathbb{1}_{H=H_0}, \\ L(H_0) &= V_{\min}, U(H_0) = V_{\max}, \chi = \{\langle s_0, s_0 \rangle\}. \end{aligned} \tag{10}$$

Thus to build a χ-FSSS tree rooted at s_0 according to abstraction relation χ, we call AFSSS($\mathcal{F}_0(s_0, V_{\min}, V_{\max})$, C, d, $V_{\min}$, $V_{\max}$, χ).

Proposition 3. *Let $T = \langle \mathcal{H}^\infty, \mathcal{A}, P, R, \gamma, s_0 \rangle$ be a (possibly infinite) history MDP and let χ be a (p, q)-consistent history equivalence relation on $\mathcal{H}^d$ for some depth parameter d. Let $\mathcal{E} = \mathcal{F}_0(s_0, V_{\min}, V_{\max})$ be the empty abstract* FSSS *tree with admissible value bounds $V_{\min}$ and $V_{\max}$. The procedure* AFSSS($\mathcal{E}$, C, d, $V_{\min}$, $V_{\max}$, χ), *with probability at least $1 - (|\mathcal{A}|C)^d 2e^{-2\lambda^2 C/(V^d_{\max})^2}$, returns an action choice a^* such that*

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\Big[\beta_\gamma(d)(\lambda + p + \delta q) + \gamma^d V^d_{\max}\Big],$$

where δ is the divergence (Eq. 6) of the completed empirical weight function $\bar{\mu}^+$ derived from the empirical weight function $\bar{\mu}$ computed by AFSSS.

Algorithm 3 Abstract Forward Search Sparse Sampling

Require: $\mathcal{F} = \langle N, L, U, H_0, \chi \rangle$: a partial χ-FSSS tree
Require: χ_0: a *default abstraction* for new state nodes
Ensure: $\mathcal{F}$ is a converged χ-FSSS tree

procedure AFSSS($\mathcal{F} = \langle N, L, U, H_0, \chi \rangle$, C, d, $V_{\min}$, $V_{\max}$, χ_0)
  global $N, L, U, H_0, \chi, C, V_{\min}, V_{\max}, \chi_0$
  while not Converged(H_0) **do**
    Visit(H_0, d)
function Converged(H)
  Let $a^* = \arg\max_{a \in \mathcal{A}} L(H, a)$
  return $L(H, a^*) \geq \max_{a \neq a^*} U(H, a)$
procedure Visit(H, d)
  if H is terminal **or** $d = 0$ **then**
    $L(H) \leftarrow \mathcal{R}_{\bar{\mu}}(H)$, $U(H) \leftarrow \mathcal{R}_{\bar{\mu}}(H)$
  else
    if not *expanded*(H) **then** Expand(H, χ_0)
    Let $a^* = \arg\max_a U(H, a)$
    Let $H^* = \arg\max_{H' \in K(H,a^*)} [U(H') - L(H')]$
    Visit(H^*, $d - 1$)
    Backup(H, a^*)
    Backup(H)
procedure Expand(H)
  for all $a \in \mathcal{A}$ **do**
    $\chi(H, a) \leftarrow \chi_0$
    $\langle L(H, a), U(H, a) \rangle \leftarrow \langle V_{\min}, V_{\max} \rangle$
    Sample(H, a)
    $\langle L(H'), U(H') \rangle \leftarrow \langle V_{\min}, V_{\max} \rangle \quad \forall H' \in K(H, a)$
  expanded(H) $\leftarrow$ **true**
procedure Sample(H, a)
  for C times **do**
    Let $h \sim \bar{\mu}(H, \cdot)$
    Let $h' \sim P(\cdot | h, a)$
    $n(h') \leftarrow n(h') + 1$
procedure Backup(H, a)
  $L(H, a) \leftarrow \mathcal{R}_{\bar{\mu}}(H) + \gamma \sum_{H' \in K(H,a)} \frac{N(H')}{M(H,a)} L(H')$
  $U(H, a) \leftarrow \mathcal{R}_{\bar{\mu}}(H) + \gamma \sum_{H' \in K(H,a)} \frac{N(H')}{M(H,a)} U(H')$
procedure Backup(H)
  $L(H) \leftarrow \max_a L(H, a)$
  $U(H) \leftarrow \max_a U(H, a)$

Proof. Let $\mathcal{F} = \langle N, L, U, H_0, \chi \rangle$ be the search tree produced by AFSSS. The value bounds $L(H)$ and $U(H)$ for all leaf states H are admissible because they are initialized to $V_{\min}$ and $V_{\max}$. The Backup operations (Algorithm 3, Lines 30 and 33) preserve admissibility. Thus the root state action-value bounds $L(H_0, a)$ and $U(H_0, a)$ are admissible for all $a \in \mathcal{A}$. Since $\mathcal{F}$ satisfies the convergence criterion (Eq. 9), the action $a^* = \arg\max_{a \in \mathcal{A}} L(H_0, a)$ is such that $L(H_0, a^*) \geq U(H_0, a)$ for all $a \neq a^*$. By admissibility, we conclude $\mathcal{Q}_G(H_0, a^*) \geq \mathcal{Q}_G(H_0, a)$ for all $a \neq a^*$, for all AbstractSS trees $G \in \Sigma_N$. Thus when AFSSS samples N, it makes the same decision as when AbstractSS samples any $G \in \Sigma_N$.

It remains to show that $\Pr(N) = \sum_{G \in \Sigma_N} \Pr(G)$, where $\Pr(X)$ is the probability of sampling X from its appropriate algorithm. The nodes in N are sampled using the same procedure as AbstractSS. The distribution of samples $\mathcal{P}_{\bar{\mu}}(\cdot|H, a)$ in a state node H is determined by $\bar{\mu}(H, \cdot)$, which itself is a random variable whose distribution is determined by the statistics of H and its ancestor nodes. In both AbstractSS and AFSSS, whenever Expand is called on a state node H, it has already been called on all ancestors of H. Thus the different order of node expansion does not matter and $\Pr(N) = \sum_{G \in \Sigma_N} \Pr(G)$. We conclude that AFSSS provides the same guarantees as AbstractSS (Proposition 2). □

4.4 Abstract Trajectory Sampling

Abstract TS algorithms have notably different properties from abstract SS algorithms. The defining characteristic of TS algorithms is that they can be implemented in terms of a *weak simulator*, which is a generative model from which complete state-action histories can be sampled under a fixed sampling policy. The process of generating a single history sample is called a *sampling episode*. Because the sampled histories must be of finite length, a TS algorithm also requires a stopping condition. We model this by augmenting the action space with the special action ω, which causes the sampling episode to be terminated but which is not appended to the sampled history. The sampling policy is often stochastic; in UCT, for example, it is common for the sampling policy to be deterministic until reaching a leaf node in the search tree, at which point it switches to a randomized "rollout" policy. Thus in this section we define the sampling policy to be a mapping from state-action pairs to probabilities, $\xi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, where $\xi(h, a)$ gives the probability of taking action a in state h. Given a history $h = s_0 a_0 s_1 \ldots a_{d-1} s_d$, the probability of sampling h under sampling policy ξ is denoted $P^\xi(h|s_0)$ and given by

$$P^\xi(h|s_0) = \mathbb{1}_{h_0 = s_0} \xi(h, \omega) \prod_{t=0}^{d-1} \xi(h_t, a_t) P(h_{t+1}|h_t, a_t).$$

To implement an abstract TS algorithm, we need to sample abstract histories $H = S_0 a_0 S_1 \ldots a_{d-1} S_d$ from the probability distribution over trajectories in T/α when executing a fixed stochastic abstract policy π, which again uses the modified action space $\mathcal{A} \cup \{\omega\}$. We denote this probability by $\mathcal{P}^\pi_\mu(H|S_0)$, where

$$\mathcal{P}^\pi_\mu(H|S_0) = \mathbb{1}_{H_0 = S_0} \pi(H, \omega) \prod_{t=0}^{d-1} \pi(H_t, a_t) \mathcal{P}_\mu(H_{t+1}|H_t, a_t).$$

We need to sample from $\mathcal{P}^\pi_\mu$, but we have access only to P^ξ. The obvious approach is to sample a ground history from P^ξ and then apply the abstraction relation χ to it. At the

Algorithm 4 Abstract Trajectory Sampling (with UCT Variation)

```
1: procedure AbstractTS(s_0, c, χ)
2:     global c, χ
3:     while time remains do
4:         Visit(s_0)
5:     return arg max_{a∈A} Q({s_0}, a)
6: procedure Visit(h)
7:     if h is terminal then
8:         return 0
9:     Let H = [h]_χ
10:    if N(H) = 0 then
11:        Let v = Evaluate(h)
12:    else
13:        Let a = Select(H)
14:        Let h' ~ P(·|h, a)
15:        Let q = Visit(h')
16:        Let v = R(h) + γq
17:    Update(H, a, v)
18:    n(h) ← n(h) + 1
19:    return v
20: procedure Select(H)
21:    if ∃a ∈ A : M(H, a) = 0 then
22:        return a
23:    Let U(H, a) = Q(H, a) + c sqrt(log N(H) / M(H,a))
24:    return arg max_{a∈A} U(H, a)
25: procedure Evaluate(h)
26:    if ℓ(h) = depth limit then
27:        return 0
28:    else
29:        Let a ~ Uniform(A)
30:        Let h' ~ P(·|h, a)
31:        return R(h) + γ Evaluate(h')
32: procedure Update(H, a, v)
33:    Q(H, a) ← Q(H, a) + (v − Q(H,a)) / M(H,a)
```

same time, we want the sampling process to be guided by the statistics of the *abstract* tree N. Thus ξ should be the *grounded* version of an abstract policy π, that is $\xi = {\downarrow}\pi$ (Eq. 2), where $\pi = \pi(N)$ is parameterized by N. The following result shows that this approach in fact yields a sample from $\mathcal{P}^{\pi}_{\mu^*}$, which is the abstract trajectory distribution with respect to the *optimal* weight function μ^*.

Proposition 4. *Consider a history MDP $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ and an abstraction $\alpha = \langle \chi, \mu^* \rangle$ of T composed of equivalence relation χ and the optimal weight function μ^*. Let π be a stochastic abstract policy $\pi : \mathcal{H}/\chi \times (\mathcal{A} \cup \{\omega\}) \mapsto [0, 1]$ whose action space is augmented with the "stop" action ω. Let H be a random variable such that $H \sim \mathcal{P}^{\pi}_{\mu^*}(\cdot|\{s_0\})$ and let h be a random variable such that $h \sim P^{\downarrow\pi}(\cdot|s_0)$. Then the random variable $[h]_\chi$ is equal in distribution to H.*

Proof. Appendix A.3. □

Proposition 4 shows that we can sample an abstract history starting in $\{s_0\}$ from $\mathcal{P}^{\pi}_{\mu^*}(\cdot|\{s_0\})$ by sampling a history from the ground dynamics $P^{\downarrow\pi}(\cdot|s_0)$ and then abstracting the ground history with χ, without explicitly computing μ^*. Algorithm 4 implements a generic abstract TS algorithm based on this approach. We also show the concrete implementations of Select, Evaluate, and Update that together create the abstract UCT algorithm with a uniform random rollout policy. Note that all calls to Select happen before all calls to Update. Thus the sampling policy is fixed while the next trajectory is

being generated and Proposition 4 applies. AbstractTS therefore operates in the abstract state space $T/\langle\chi, \mu^*\rangle$ for any choice of χ. Contrast this with AbstractSS, for which μ is estimated and therefore subject to error.

4.5 Handling Action Constraints

In some problems, not every action is allowed in every state. Let $\mathcal{A}(h) \subseteq \mathcal{A}$ denote the subset of *legal* actions in ground state h. The *illegal* actions in h are the complement of this set. If a problem has different legal action sets in different states, then the abstraction relation χ might aggregate ground states with different legal action sets into the same abstract state, so that there exist states h and g such that $h \simeq_\chi g$ but $\mathcal{A}(h) \neq \mathcal{A}(g)$. It is then not obvious what the set of legal actions should be in the abstract state $H = [h]_\chi = [g]_\chi$. We have identified three ways of addressing this issue.

1. Require χ to be such that $h \simeq_\chi g \Rightarrow \mathcal{A}(h) = \mathcal{A}(g)$.

2. Set $\mathcal{A}(H) = \bigcap_{h \in H} \mathcal{A}(h)$ for each $H \in \mathcal{H}/\chi$.

3. Set $\mathcal{A}(H) = \bigcup_{h \in H} \mathcal{A}(h)$ for all H, but model illegal state-action combinations (i.e. $\langle h, a\rangle$ such that $a \notin \mathcal{A}(h)$) as having no effect and/or giving a penalty.

Option 1 is reasonable if it is rare that two states with different action sets are reached by the same sequence of actions. If different action sets are common, then it becomes difficult to find nontrivial abstractions that satisfy condition 1, and the benefits of abstraction are lost. The viability of option 2 depends on whether the intersection of the ground action sets usually contains the actions necessary for good performance. Option 2 also makes implementing abstraction refinement (Section 5) more difficult, since refining the abstraction can expand the legal action sets for the aggregate states. Note that option 2 has a similar effect to giving a penalty of $-\infty$ for attempting an illegal action.

For our experiments (Section 8), we selected domains in which option 3 could be used, to maximize opportunities for state aggregation. When the problem involves controlling an "embodied" agent, such as in typical navigation tasks, it is natural for all actions to be possible in all states. In some other problems, one can model an illegal action as having the same effect as the "most similar" legal action. For example, in Tetris (Section 8.3.9) some types of blocks have more legal positions than others, so we map illegal positions to the nearest legal position. This makes the planning problem slightly harder, since there are now redundant actions that waste samples.

The problem of action constraints is a fundamental obstacle to state abstraction. The framework of approximate MDP homomorphisms (Ravindran & Barto, 2004) addresses the problem by extending the notion of state abstraction to abstractions of $\langle s, a\rangle$ pairs. This allows action symmetries to be modeled, which is one way of enlarging the intersection of action sets (option 2 above). Anand et al. (2015) developed abstract UCT algorithms based on homomorphisms of history MDPs. The problem of identifying useful subsets of the action set in MCTS was studied by Pinto and Fern (2014), and similar methods could be used to identify a common action set or determine that no useful one exists.

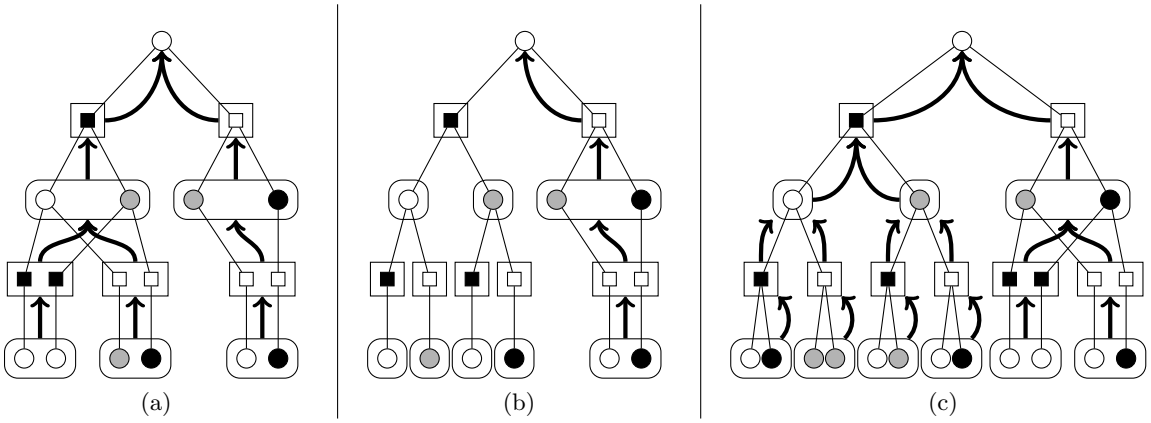


Figure 4: One iteration of abstraction refinement in the PARSS algorithm. (a) An abstract FSSS tree with respect to $\top$ of width $C = 2$ and depth $d = 2$. Note that part of the tree was not expanded. (b) After refining one state abstraction, the ground samples are re-partitioned to respect the new abstraction. The abstract FSSS invariant (Definition 5) no longer holds. (c) After up-sampling and value backups, the tree again satisfies the abstract FSSS invariant. The pruned subtree had to be expanded because abstraction refinement changed the value estimates.

5. Abstraction Refinement Algorithms

It is difficult to assess the quality of a state abstraction for tree search *a priori* because of the complex interaction of abstract state space size, abstraction accuracy, sample budget, and search depth. Planning problems often have "critical horizons", meaning that important consequences of actions only manifest sufficiently far in the future. For example, a car must begin braking well before entering a turn. If an online planning algorithm cannot search to the critical horizon, it will not recognize the possibility of a crash until it is too late to prevent it. Although abstractions may introduce error, the corresponding increase in search depth may give an overall performance gain by allowing the search to reach a critical horizon. Further, state abstraction reduces the space of policies, so that even if the optimal policy is not representable in the abstract state space, many poor policies may be excluded along with it, resulting in a net benefit.

We address the problem of abstraction specification by designing a sparse sampling algorithm that refines its abstraction during search, so that the abstraction becomes finer as the number of samples increases. This allows the representation to adapt automatically to the search budget.

5.1 Abstraction Refinement

To refine an abstraction relation χ means, intuitively, to define a new abstraction relation ψ that preserves more detail about the ground state space than χ. Abstraction refinement gives rise to an ordering of abstraction relations.

Definition 6. Abstraction ψ is *finer* than χ, denoted $\psi \preceq \chi$, if $h \simeq_\psi g \Rightarrow h \simeq_\chi g$. If in addition $\psi \neq \chi$, then ψ is *strictly finer* than χ, denoted $\psi \prec \chi$.

Algorithm 5 A generic abstraction refinement procedure

```
1: procedure PAR(F = ⟨N, L, U, H_0, χ⟩)
2:    Let H = SELECT(N)
3:    if H ≠ ∅ then
4:       χ(p(H), a(H)) ← REFINE(χ(p(H), a(H)))
5:       SPLIT(p(H), a(H), χ)
6:       UPDATETREE(p(H), a(H))
```

State equivalence abstractions form a complete lattice under this ordering. The finest abstraction is the *bottom* or *ground* abstraction $\perp$, which maps all states to singleton sets, $[h]_\perp = \{h\}\ \forall h$. The coarsest abstraction is the *top* abstraction $\top$, which maps all ground histories of the same length and containing the same action sequence to the same abstract history, $[h]_\top = \{g \in \mathcal{H}^{\ell(h)} : a(h_i) = a(g_i), i = 1, \ldots, \ell(h)\}$. Searching in the abstract problem $T/\top$ amounts to searching for an *open-loop* policy that executes a particular sequence of actions regardless of the states encountered during execution. Searching in $T/\perp$ is equivalent to searching in the ground state space.

Given a refinement operator F such that $F(\chi) \prec \chi$, the lattice structure implies that repeated application of F eventually yields the bottom abstraction, that is $F^*(\chi) = \perp$. The search algorithm we describe next relies on this property to enable it to exploit state abstractions during search while still providing the performance guarantees of search in the ground state space.

5.2 Progressive Abstraction Refinement for Sparse Sampling

The Progressive Abstraction Refinement for Sparse Sampling (PARSS) algorithm (Algorithm 7), originally proposed by Hostetler et al. (2015), is an adaptation of AFSSS that refines its abstraction during search. PARSS begins by building a complete $\top$-FSSS tree. PARSS then iteratively refines the abstraction and revises the search tree to respect the new abstraction until there are no more useful refinements to perform. We present an improved version of PARSS that incorporates lessons learned from our experiences with the original algorithm.

PARSS combines a slightly modified AFSSS algorithm (Algorithm 3) with the generic refinement procedure PAR described in Algorithm 5. The PAR procedure consists of four steps. The SELECT function either returns a state node H whose associated abstraction relation $\chi(p(H), a(H))$ should be refined, or indicates that no refinement is to be done. The REFINE procedure performs the refinement of the selected abstraction relation. After refinement, the subtree below the refined state node is SPLIT recursively according to the new abstraction. Finally, UPDATETREE revises the part of the tree affected by the refinement.

Algorithm 7 includes implementations of SPLIT and UPDATETREE. The SPLIT procedure traverses the subtree affected by an abstraction refinement and alters its structure to respect the new abstraction. UPDATETREE proceeds in two steps. First, the UPSAMPLE procedure adds additional samples to the affected subtree so that each action node has been sampled at least C times and recomputes the value bounds in the subtree with BACKUP. Then, the value bounds of the affected subtree are propagated along the path to the root

of the search tree using Backup. The remaining two operations, Select and Refine, can be realized in many ways, and we describe several possibilities in Section 6.

After each PAR operation, PARSS calls AFSSS on the refined tree. This is necessary because refinement may have changed the value bounds of the root node in such a way that the tree no longer satisfies the convergence criterion. After AFSSS returns, the resulting tree is an abstract FSSS tree with respect to the newly refined abstraction. During both the AFSSS and PAR procedures, PARSS conducts its sampling using the SampleModified procedure defined in Algorithm 6. In the usual sparse sampling Sample procedure, for an abstract action node Ha, we sample C ground states $h^1, \ldots, h^C$ where $h^i \sim \bar{\mu}(H, \cdot)$ and then sample one successor for each from $P(\cdot|h^i, a)$. In the modified version, we instead repeatedly sample one successor from $P(\cdot|h, a)$ for *every* $h \in H$ until we have *at least* C ground successor samples. We do this to ensure that all ground states $h \in H$ in an abstract state H contribute to the statistics of H. This reduces variance in the abstract value estimates, and ensures that the Select and Refine procedures have data about all of the ground states on which to base their decisions. If PARSS is run to completion, then the final search tree will be a $\perp$-FSSS tree, and each abstract state node H will be a singleton $\{h\}$. The modified sampling procedure ensures that at this point, $m(h, a) = C$ for all action nodes ha, and thus that PARSS will have used no more samples to construct this tree than would have been required by $\perp$-SS (Proposition 6).

5.3 Analysis of PARSS

The PARSS algorithm can be viewed as a different way of orchestrating the sampling of a sparse tree. In this section, we establish that PARSS provides the same bounded suboptimality guarantees with the same sample complexity as ordinary sparse sampling, provided that the Select and Refine operations of PARSS satisfy some simple conditions that ensure that the abstraction refinement procedure PAR continues to make progress. Namely, Select must be *complete*, while Refine must be *strict*. To define these terms, we first need some vocabulary for the different possible dispositions of state nodes.

Definition 7 (Expanded state node)**.** A state node H is *expanded* if *expanded*(H) is **true**.

Definition 8 (Pure state node)**.** A state node H is *pure* if H is expanded and there exists a single ground history $h \in H$ such that $n(h) = N(H)$.

If a state node H is pure, then nothing is accomplished by further refining H, because all of the ground samples in the equivalence class H correspond to the same ground history. Note that this need not imply that $\chi(p(H), a(H)) = \perp$, since it may be that not all ground histories in the equivalence class H have been encountered during sampling.

We can now state the necessary conditions for the Select and Refine operations.

Definition 9. A Select implementation is *complete* if it returns a state node H, whenever such an H exists, such that H is expanded and H is not pure.

Definition 10. A Refine implementation is *strict* if Refine$(\chi) \prec \chi$ whenever $\chi \succ \perp$.

Definition 9 ensures that Select eventually selects every state node H such that refining H could possibly change the optimal root action. We can exclude un-expanded state

Algorithm 6 Modified SAMPLE procedure for PARSS

procedure SAMPLEMODIFIED(H, a)
  for all $h \in H$ **do**
    while $m(h,a) < \lceil \frac{C}{N(H)} \rceil$ **do**
      Let $h' \sim P(\cdot|h,a)$
      $n(h') \leftarrow n(h') + 1$

Algorithm 7 Progressive Abstraction Refinement for SS

procedure PARSS(s_0, C, d)
  Let $\mathcal{F} = \mathcal{F}_0(s_0, V_{\min}, V_{\max})$ (Eq. 10)
  AFSSS($\mathcal{F}$, C, d, $\top$) ▷ Using SAMPLEMODIFIED
  while time remains **and** some $\chi(H,a) \succ \perp$ **do**
    PAR($\mathcal{F}$)
    AFSSS($\mathcal{F}$, C, d, $\top$) ▷ Using SAMPLEMODIFIED
procedure SPLIT(H, a, χ, $H^{\text{old}} = H$) ▷ H^{old} contains H in the old tree
  if H is a leaf **then return**
  for all $H' \in K(H^{\text{old}}, a)$ **do**
    Let $\mathcal{G}' = H'/\chi(H,a)$ ▷ Refined partition
    for all $\langle G', a' \rangle \in \mathcal{G}' \times \mathcal{A}$ **do**
      $N(G') \leftarrow \sum_{g \in G'} n(g)$ ▷ Update abstract tree
      $\chi(G', a') \leftarrow \chi(H', a')$ ▷ Copy old abstraction relation
      SPLIT(G', a', χ, $H^{\text{old}} : aH'$)
    free data structures for H'
procedure UPDATETREE(H, a)
  for all $H' \in K(H,a)$ **do**
    UPSAMPLE(H')
  for t from 0 to $\ell(H)$ **do** ▷ Backup path to root
    for all $a \in \mathcal{A}$ **do** BACKUP(H, a)
    BACKUP(H)
    Let $H = p(H)$
procedure UPSAMPLE(H)
  if H is a leaf **then**
    $L(H) \leftarrow \mathcal{R}_{\bar{\mu}}(H)$, $U(H) \leftarrow \mathcal{R}_{\bar{\mu}}(H)$
  else if *expanded*(H) **then**
    for all $a \in \mathcal{A}$ **do**
      SAMPLEMODIFIED(H, a)
      for all $H' \in K(H,a)$ **do** UPSAMPLE(H')
      BACKUP(H, a)
    BACKUP(H)

nodes in Definition 9 because if H is un-expanded then $L(H) = V_{\min}$ and $U(H) = V_{\max}$ (Algorithm 3, Line 23), so refining H cannot increase $U(H)$ or decrease $L(H)$ and thus cannot change the optimal root action. Definition 10 simply requires that Refine actually refines the abstraction, which is always possible when H is not pure.

Our analysis of PARSS will proceed as follows. We begin by observing that PARSS produces a sequence of abstraction relations $(\chi_0, \chi_1, \ldots)$ with $\chi_{t+1} \preceq \chi_t$ and a sequence of abstract FSSS trees $(\mathcal{F}_0, \mathcal{F}_1, \ldots)$ with respect to each χ_t. Proposition 5 establishes that $\mathcal{F}_t$ is an abstract FSSS tree with respect to χ_t for each t. Next, Proposition 6 shows that there exists a finite τ such that $\mathcal{F}_\tau$ is a $\perp$-FSSS tree. Lemma 7 shows that $\perp$-SS achieves the same performance guarantees as ordinary SS. Finally, we combine these results in Proposition 8 to conclude that PARSS achieves the same performance guarantees as ordinary SS.

Proposition 5. *Consider a* PARSS *implementation where the* Select *and* Refine *operations satisfy the conditions of Definitions 9 and 10. If the current search tree $\mathcal{F}$ is a* χ-FSSS *tree, then after one iteration of the loop in Algorithm 7, Line 4, the resulting tree $\mathcal{F}'$ is a* ψ-FSSS *tree for some ψ such that $\psi \prec \chi$.*

Proof. By assumption, Refine(H) produces a new abstraction ψ such that $\psi(p(H), a(H)) \prec \chi(p(H), a(H))$, and therefore $\psi \prec \chi$. The Split operation partitions the subtree rooted at H according to ψ, establishing Condition 1 of Definition 5. The UpSample loop in UpdateTree (Line 17) adds samples and performs backups in the subtree of H to establish Conditions 2 and 3 for the subtree. Then values are backed up from H to the root node (Line 19), which establishes Condition 3 for the rest of the tree. Finally, the call to AFSSS (Line 6) establishes convergence (Condition 4). □

Now that we have established that each iteration of refinement produces an abstract FSSS tree with respect to a strictly refined abstraction, we can exploit the lattice structure of abstraction relations to argue that this iterative refinement will eventually produce a $\perp$-FSSS tree.

Proposition 6. *If* PARSS *does not exhaust its time budget, it terminates after drawing at most $(|\mathcal{A}|C)^d$ samples from the transition function P, and the resulting search tree $\mathcal{F}$ is an abstract* FSSS *tree with respect to $\perp$.*

Proof. By Proposition 5, each iteration of the loop in Algorithm 7, Line 4 produces a strictly refined AFSSS tree. Due to the lattice structure of aggregation abstractions (Section 5.1), the abstraction relations $\chi(H, a)$ will induce the same equivalence classes as $\perp$ for all H, a after a finite number of iterations. The tree at this point is an abstract FSSS tree with respect to $\perp$.

The worst-case sample complexity occurs if all abstract nodes H in the fully-refined tree are singletons and no pruning takes place. In this case, each abstract state node is a singleton set $H = \{h\}$, and its successors $K(H, a)$ are singleton sets of the ground successors in $k(h, a)$. Note that the SampleModified procedure (Algorithm 6) samples sucessors for every ground state h until $|k(h, a)| = \lceil C/|H| \rceil$. Since $\lceil C/|H| \rceil$ achieves its maximum of C when $|H| = 1$, the tree in which every abstract state node is a singleton represents the worst-case sample complexity, and its size is $(|\mathcal{A}|C)^d$. □

The next lemma formalizes the intuitive result that aggregating $\perp$-equivalent states in the SS algorithm does not affect its performance guarantees, that is that a $\perp$-SS(C, d) tree provides the same guarantees as an ordinary SS(C, d) tree. This result was stated by Kearns et al. (2002), and we prove it here for completeness.

Lemma 7. *Abstract sparse sampling with the bottom abstraction $\perp$ achieves the same sample complexity and bounded suboptimality guarantees as ordinary sparse sampling.*

Proof. The $\perp$-SS tree estimates the same value function as ordinary SS. The analysis of the probability of error for SS proceeds by bounding the probability of error in a single tree node and then applying the union bound to derive the probability that *no* tree node exceeds this error bound. The $\perp$-SS tree never contains more nodes than the ordinary SS tree, thus the overall probability of error is no larger for $\perp$-SS. □

We can now combine Proposition 6 and Lemma 7 to establish our desired result.

Proposition 8. PARSS *achieves the same bounded suboptimality guarantees with the same sample complexity as ordinary sparse sampling.*

Proof. Proposition 6 establishes that PARSS yields a $\perp$-FSSS tree $\mathcal{F}$ with the same worst-case sample complexity as SS, i.e. $O((|\mathcal{A}|C)^d)$. $\mathcal{F}$ is different from a ground FSSS tree in that states that are equal in the ground representation are aggregated in $\mathcal{F}$. Because the FSSS pruning mechanism is sound when L and U are admissible, $\mathcal{F}$ achieves the same error bounds as a $\perp$-SS tree. By Lemma 7, such a $\perp$-SS tree achieves the same guarantees as ordinary sparse sampling. The conclusion follows. □

From this analysis, we conclude that PARSS can be expected to perform similarly to SS and FSSS if the searches are run to completion. From a practical standpoint, the rate of performance improvement during search is also important. This is a difficult issue to address theoretically because of the complicated dynamics of tree search. Instead, we show empirically (Sections 8 and 9) that PARSS has an advantage compared to FSSS and AFSSS in this regard.

5.4 Optimizing Memory Usage

A drawback of PARSS is that for any abstract state node H that might later be refined, the ground state samples $s(h)$ for every $h \in H$ must be retained. This is because after refinement, more successor samples might need to be drawn from $\mathcal{P}_{\bar{\mu}}(\cdot|H, a)$, which involves sampling a ground history $h \in H$ from $\bar{\mu}(H, \cdot)$ and then simulating action a in $s(h)$. The memory cost of retaining these ground state samples may be significant if there are many state variables, so we would like to free the memory associated with samples that are no longer needed. We will show that the ground state samples associated with an abstract state node H can be discarded if H satisfies the following criterion.

Definition 11 (Closed state node)**.** A state node H is *closed* if H is pure and all state node ancestors of H are pure.

If we know that a node H is closed, we can free the memory used to store the ground states $h \in H$, due to the following fact.

Proposition 9. Sample *is never called on a closed state node.*

Proof. The Sample procedure (Algorithm 3 Line 25) is called only when either expanding an un-expanded state node (Algorithm 3 Line 18) or when up-sampling a newly refined subtree (Algorithm 7 Line 23). In the first case, a closed node will not be sampled because it is pure and thus by definition already expanded. In the second case, UpSample will not be called on a closed node or any of its ancestors because Select never selects a pure node. □

Since Sample is never called on a closed state node H, no further successor samples will be drawn from $\mathcal{P}_{\bar{\mu}}(\cdot|H, a)$ and memory used to store the ground states $s(h)$ for each $h \in H$ can be freed. The algorithm need only retain the value estimates and upper and lower bounds associated with H. We found this optimization to be important in practice. Note that when doing sparse sampling with a *fixed* abstraction (including $\perp$), we can discard the ground state samples as soon as the abstract state that contains them is expanded. This is a disadvantage of PARSS compared to AFSSS with a fixed abstraction, since AFSSS with a fixed abstraction does not need to store the ground state samples for non-leaf state nodes, while PARSS might need to retain every ground state sample drawn so far. Thus PARSS has a larger memory footprint than AFSSS.

5.5 Abstraction Refinement in Trajectory Sampling

It is easy to imagine a "Progressive Abstraction Refinement for Trajectory Sampling" algorithm designed along similar lines as PARSS. Besides the advantages of TS algorithms compared to SS algorithms when abstractions are used (Section 4), TS algorithms are more popular in applications (Browne et al., 2012). We have not thoroughly investigated such a "PARTS" algorithm, but our preliminary work raised some concerns that prompted us to pursue the sparse sampling-based alternative.

One concern is that whereas an SS tree for fixed C and d contains a finite number of nodes, in principle a TS algorithm could go on adding samples indefinitely. Thus one must make a somewhat arbitrary choice of when to pause sampling and consider abstraction refinements. A second concern is that because TS algorithms are not systematic, they might be slow to explore a newly-refined subtree, especially if it is not part of the optimal subtree with respect to the old abstraction. Thus one might want to tweak the exploration parameters or value estimates to encourage exploration. These considerations add degrees of freedom to the design of the algorithm, making it harder to isolate the effect of abstraction from the effects of other design choices. Nevertheless, most other work on abstraction in tree search is based on TS algorithms, not SS algorithms (Section 7), and we feel that abstraction refinement in TS is an important area for further work.

6. Refinement Strategies

To instantiate the PAR procedure, we need to implement the Select and Refine operations. This section describes the strategies that we implemented for our experiments.

6.1 State Node Selection

Besides satisfying the conditions of Definition 9, the Select procedure should return a state node in which a useful refinement is likely to be available. We investigated three selection strategies in our experiments.

6.1.1 Breadth-First Selection

The first work with PARSS (Hostetler et al., 2015) used a breadth-first selection order. The breadth-first order is a natural choice in discounted problems ($\gamma < 1$) because the values of nodes near the root are less affected by discounting when calculating the root value. Improving the value estimate in shallow nodes has an exponentially larger impact on the root value than improving the estimate in deeper nodes. Since shallow nodes also have exponentially more descendants than deep nodes, refining shallow nodes first causes the refinement process to take large "steps" through the space of policy sets. Each refinement adds many policies to the set of policies whose values the search tree model can estimate. These large steps mean that more sampling will be done after each refinement, since a large portion of the tree is affected. Breadth-first selection also has the practical benefit that a state node is *closed* as soon as it becomes *pure*, since its ancestors are already closed. This makes breadth-first selection the easiest to implement.

6.1.2 Uniform Selection

Breadth-first selection is a poor choice if relevant randomness only occurs deep in the tree. For example, an action might cause a value-relevant random event after a delay of several time steps. Breadth-first selection would waste samples refining nodes at depths less than the time delay, where the abstraction is already sound.

Uniform selection avoids this problem by selecting an open state node to refine uniformly at random. An obvious shortcoming of uniform selection is that nodes at greater depths are exponentially more likely to be selected, and refinements to deep nodes are less likely to affect the value estimate in the root node. We do not expect uniform selection to be the best choice, but it provides a useful comparison due to its naïvete.

6.1.3 Heuristic Guided Selection

Most generally, we can define a priority ordering over the set of open abstract state nodes and refine the highest-priority state nodes first. One obvious general-purpose heuristic is to refine state nodes H in which there is high variance across the action value estimates for the constituent ground states $h \in H$. Let $q(h, a)$ denote the value estimate for action a based on the subtree of the sample tree rooted at h. This quantity is defined recursively in the usual way,

$$q(h,a) = R(h) + \frac{1}{m(h,a)} \sum_{h' \in k(h,a)} n(h') \max_{a' \in \mathcal{A}(h)} q(h', a'). \tag{11}$$

These values can be computed along with the statistics for the abstract states during the Backup step (Algorithm 3, Line 30).

Let $\sigma^2(H,a) = \frac{1}{M(H,a)} \sum_{h \in H} n(h)\big(q(h,a) - \bar{q}(H,a)\big)^2$ denote the sample variance of the set $\{q(h,a) : h \in H\}$, where $\bar{q}(H,a) = \frac{1}{M(H,a)} \sum_{h \in H} n(h) q(h,a)$ is the average value of action a over the samples in H. We can define a priority heuristic for an abstract state node H by taking the average of these variances over all actions,

$$f_{\sigma^2}(H) = \frac{1}{\sum_{a \in \mathcal{A}} M(H,a)} \sum_{a \in \mathcal{A}} M(H,a)\sigma^2(H,a).$$

Refining state nodes for which f_{σ^2} is large makes sense in light of Theorem 1, since if $f_{\sigma^2}(H) = 0$, then H is part of a $(0,0)$-consistent partition of the sampled collection of ground states, and thus the abstraction error due to H is zero.

Note that the breadth-first and uniform selection strategies can also be defined in terms of heuristic functions,

$$f_{\text{bf}}(H) = \frac{1}{\ell(H)},$$
$$f_{\text{unif}}(H) = 1,$$

with ties being broken randomly.

6.2 State Abstraction Refinement

The Refine procedure splits the abstract state chosen by Select into two new abstract states, ideally lowering the abstraction error in the process. In our experiments, we evaluated the following two approaches to abstraction representation and refinement.

6.2.1 Random Refinement

Given an abstract state node H chosen by Select, the Random refinement strategy randomly permutes the equivalence classes in $H/\perp$ and greedily divides them into two sets containing approximately the same number of ground samples. This option is fast to compute and places no requirements on the ground state representation, but it does not exploit structure in the ground state space. During search, previously unseen histories h are added to the abstract state H that currently has the smallest value of $N(H)$.

6.2.2 Decision Tree-Based Refinement

It is common for states in an MDP to have a factored form, so that each state s is identified with a feature vector, $s = \langle \phi_1(s), \phi_2(s), \ldots \rangle$. We can exploit this structure with a more sophisticated approach. The DT refinement strategy is based on incrementally-constructed decision trees. Each abstraction relation $\chi(H,a)$ is defined by a decision tree D. The leaves of D define the members of a partition of the successors of Ha. Interior nodes are labeled with a feature i and a threshold θ. The refinement operation adds a new split to D dividing the leaf node corresponding to H into two new sets X and Y, with i and θ chosen greedily to maximize an evaluation function $f(X,Y)$.

The evaluation function f can be designed to encourage desired properties in the partitions. For example, if χ is such that $\mathcal{H}/\chi$ is $(0,0)$-consistent (Definition 2) then χ is

sound in sparse sampling (Theorem 1). We define an evaluation function that encourages $(0, 0)$-consistency using upper bounds $u(h)$ and $u(h, a)$ for ground state values, where

$$u(h) = R(h) + \gamma \begin{cases} \max_{a \in \mathcal{A}([h]_\chi)} u(h, a) & h \text{ is not a leaf} \\ 0 & \text{otherwise} \end{cases},$$

$$u(h, a) = \frac{1}{m(h, a)} \sum_{h' \in k(h,a)} n(h')u(h').$$

Like the ground state q-function (11), $u(h, a)$ and $u(h)$ can be computed during the Backup step. Using these bounds on the ground states, we define the evaluation function

$$f(X, Y) = |\bar{u}(X) - \bar{u}(Y, a^*)| + |\bar{u}(Y) - \bar{u}(X, b^*)|,$$

where $\bar{u}(H) = \frac{1}{n(H)} \sum_{h \in H} n(h)u(h)$ and $\bar{u}(H, a) = \frac{1}{n(H)} \sum_{h \in H} n(h)u(h, a)$ are averages of the ground state upper bounds, $a^* = \arg\max_{a \in \mathcal{A}} \bar{u}(X, a)$, and $b^* = \arg\max_{b \in \mathcal{A}} \bar{u}(Y, b)$. Splits that maximize f will tend to put ground states that have different optimal actions or different optimal values into different abstract states.

DT is similar to the mechanism used by Van den Broeck and Driessens (2011) in their Tree Learning Search algorithm, as well as to the UTree mechanism (McCallum, 1996). Variations on the DT theme could be created by replacing the "feature-value" splits with a different decision rule. The decision tree could also be replaced with a different representation of the abstraction relation.

7. Related Work

Much of the theory of state abstraction in MDPs is based on the framework of stochastic bisimilarity (Givan, Dean, & Greig, 2003). Bisimilarity is a strong equivalence criterion; two states are bisimilar if and only if they cannot be distinguished by observing reward sequences received under any policy. Bisimilarity metrics (Ferns, Panangaden, & Precup, 2004) generalize bisimilarity to include approximate equivalence. Li et al. (2006) provided a taxonomy of state equivalence criteria that are weaker than bisimilarity but still sound. Their criteria of π^*- and a^*-irrelevance are the basis of our (p, q)-consistency criterion (Definition 2). Van Roy (2006) derived regret bounds with a similar form to Theorem 1 for value iteration with state aggregation.

The idea of adaptive refinement or revision of an abstraction has been the basis for several MDP abstraction algorithms, including the G algorithm (Chapman & Kaelbling, 1991), the Parti-Game algorithm (Moore & Atkeson, 1995), and the UTree algorithm (McCallum, 1996). Baum, Nicholson, and Dix (2012) proposed an adaptive state abstraction that is varied according to heuristics including proximity to the agent and differences in action outcomes. The abstraction refinement heuristics in all of these works are based on similar intuitions, and many are similar to the heuristics we use in PARSS (Section 6). Unlike PARSS, these algorithms maintain a complete policy for the current abstract problem and execute it, whereas PARSS is an OP algorithm and thus replans every time step.

Most other work on abstraction in tree search has focused on trajectory sampling algorithms. PARSS is most similar to the TLS algorithm proposed by Van den Broeck and Driessens (2011), which is based on UCT. TLS is targeted at continuous action spaces,

and it works by progressively refining the action continuum at individual state nodes in the tree, exactly analogous to PARSS but applied to actions rather than states. The AS-UCT algorithm proposed by Jiang et al. (2014), also based on UCT, differs by taking a "batch" approach to abstraction construction, as opposed to the incremental approach of TLS and PARSS. In this batch approach, a tree is first sampled under the current abstraction (which begins as $\perp$). After the sampling period, an approximate abstraction is calculated from the sampled tree. The process is then iterated using the new abstraction for sampling. Jiang et al. (2014) derived suboptimality bounds for their abstractions using the theory of MDP homomorphisms (Ravindran & Barto, 2004). The ASAP-UCT algorithm of Anand et al. (2015) extends AS-UCT to abstract $\langle h, a \rangle$ pairs, which enables the abstraction to take advantage of action symmetries. OGA-UCT (Anand, Noothigattu, Mausam, & Singla, 2016) is an incremental version of ASAP-UCT that interleaves sampling and abstraction revision.

Like PARSS, algorithms in the AS-UCT family maintain an abstract "view" of the samples drawn so far and use it to guide sampling. The abstraction used to construct these views is then periodically revised. Besides being based on a different search algorithm, the major difference between PARSS and these abstract UCT algorithms is that in PARSS the abstraction revision is always a refinement, while the AS-UCT algorithms revise their abstractions to more closely approximate a *target* abstraction $\chi \succ \perp$ with particular properties. A more minor difference is that in AS-UCT and its descendants, the abstract "view" is a directed acyclic graph (DAG), while in PARSS it is a tree. The use of a DAG is a common optimization applied in practice to reduce memory usage. PARSS could be adapted to use a DAG structure as well, but our theoretical bound on regret due to abstraction (Theorem 1) does not apply in this case.

State abstraction has also been applied in classical planning to create a class of domain-independent admissible heuristics called *abstraction heuristics*. This work began with pattern databases (Culberson & Schaeffer, 1998; Edelkamp, 2001) and has been developed into methods such as merge-and-shrink heuristics (Helmert, Haslum, & Hoffmann, 2007). Abstraction heuristics compute a lower bound on the cost-to-go in the planning problem by solving a "relaxed" version of the problem created through state abstraction. The heuristic is used to guide search, but the search still takes place in the ground problem.

State abstraction is one of three major types of abstraction available in MDPs, the other two being action abstraction and temporal abstraction. Action abstraction consists of reducing the space of possible actions at each step, while temporal abstraction reduces the effective planning horizon through the use of temporally extended actions. Both types of abstraction have been applied in tree search. The TLS algorithm already mentioned (Van den Broeck & Driessens, 2011) builds a search tree over action equivalence classes. Pinto and Fern (2014) used action pruning to speed up UCT and were able to learn pruning functions for which the regret of the tree search procedure is bounded. Bai et al. (2015) extended UCT to include temporal abstraction in the form of options (Sutton, Precup, & Singh, 1999). Their algorithm is hierarchical, so that options can invoke sub-options, and so on recursively until reaching a primitive action. Hierarchical action decomposition is commonly used in classical planning in the form of hierarchical task networks (HTNs) (Erol, Hendler, & Nau, 1994; Nau, Au, Ilghami, Kuter, Murdock, Wu, & Yaman, 2003).

The POMDP view of abstraction (Section 3.2) illuminates a connection between abstract tree search and policy search algorithms for POMDPs. Sparse sampling itself derives from earlier work using sample trees to evaluate policies during policy search (Kearns, Mansour, & Ng, 1999). PARSS essentially enumerates and evaluates policies in a certain order determined by the order of abstraction refinements. By starting from the top abstraction $\top$, PARSS evaluates open-loop policies first, and then each abstraction refinement expands the policy search set by including new policies that make more distinctions between states. Several works have explored the use of open loop policies for value estimation in POMDPs. Weinstein and Littman (2012) applied this idea in continuous action MDPs, drawing on theory developed by Bubeck and Munos (2010). Weinstein and Littman (2013) later developed a related algorithm with a different optimization mechanism and applied it to legged locomotion tasks. Hauser (2011) used forward search with open loop policies to plan in partially observable continuous spaces.

The idea of aggregating histories rather than states also has roots in the study of POMDPs. The UTree algorithm (McCallum, 1996) takes a progressive refinement approach to discovering an effective history abstraction. UTree constructs abstractions that map histories to abstract *states* and builds an empirical model of the abstract MDP. A policy for the abstract problem is then computed using standard methods. The theory of such history-to-state abstractions has been further developed by Hutter (2014).

8. Experiments

Our experiments compare multple variations of PARSS to one another and to AFSSS with fixed abstractions on a variety of problem domains. The complete source code used in our experiments is available at `https://github.com/jhostetler/jmcplan/releases/tag/v0.1`.

8.1 Objective

The objective of our experiments is to thoroughly evaluate the effectiveness of state abstraction and abstraction refinement in comparison to search in the ground state space, in a setting with as few confounding variables as possible. We therefore compare only sparse sampling-based algorithms, and we do not compare these algorithms to trajectory sampling algorithms like UCT or to competition planners such as PROST (Keller & Eyerich, 2012). By restricting our comparison to algorithms with a similar structure, we can be confident that any performance differences we observe are due to the use of abstraction rather than details of particular algorithms that make them better or worse at particular problems.

8.2 Algorithms

We tested six different variations of PARSS representing the cross product of the three node selection strategies Breadth-First (BF), Uniform, and Variance (Section 6.1) and the two refinement strategies DT and Random (Section 6.2). We compared these PARSS variants to $\bot$-FSSS and $\top$-FSSS, and also to AFSSS with two random abstractions of different granularities (rand-FSSS).

Algorithm 8 Modified Sample procedure for rand-FSSS

```
1: procedure SampleRand(H, a)
2:   for all h ∈ H do
3:     while m(h, a) < ⌈C / n(H)⌉ do
4:       Let h' ~ P(·|h, a)
5:       n(h') ← n(h') + 1
6:   for all h' ∈ ⋃_{h∈H} k(h, a) do
7:     if ∃G ∈ K(H, a), g ∈ G where h' = g then
8:       continue
9:     else if |K(H, a)| < B then
10:      Add new equivalence class {h'} to χ(H, a)
11:    else
12:      Let G = arg min_{H'∈K(H,a)} N(H')
13:      Modify χ(H, a) so that [h']_{χ(H,a)} = G.
```

The random abstraction search algorithm is obtained by changing the definition of the Sample function of AFSSS (Algorithm 3) to the one in Algorithm 8. The modified Sample(H, a) procedure places novel ground successor states into their own equivalence class until $|K(H, a)| = B$, where B is a parameter of the algorithm. Once $|K(H, a)| = B$, subsequent novel ground successor states are added to the member $H' \in K(H, a)$ with the smallest value of $N(H')$. The resulting tree has a maximum stochastic branching factor of B, but is likely to incur a high abstraction error since the abstractions are random. The purpose of rand-FSSS is to provide a simple baseline abstraction that is between $\perp$ and $\top$ in granularity.

8.3 Problems

Our problem pool includes the domains used by Hostetler et al. (2015) as well as several additional problems. All of the problems are episodic with a maximum episode length, and all are undiscounted ($\gamma = 1$). In the goal-oriented problems, episodes terminate as soon as a goal is reached, except in the IPC problems, for which we use the IPC specifications without alteration.[3] In a few domains, we provide an admissible heuristic evaluation function whose value is added to the immediate reward in all leaf nodes. Our domains have relatively small state spaces, but for SBTS algorithms it is the branching factor and the necessary search depth, rather than the raw state space size, that influence the difficulty of the problem. We now describe each domain in detail.

8.3.1 Saving

The Saving problem (Hostetler et al., 2015) is designed specifically to illustrate the effect of certain structural features of the problem on the different tree search algorithms. The agent must accumulate wealth by choosing to either *save*, *invest*, or *borrow* at each time step. The problem is parameterized by integers $\langle p_{\min}, p_{\max}, T_b, T_i, T_m \rangle$ where $p_{\min} \leq p_{\max}$ and

3. We implemented the IPC problems by hand rather than using an RDDL simulator, for efficiency.

$T_i, T_b, T_m > 0$. Its state space consists of integers $\langle p, t_b, t_i, t_m \rangle$, where $p \in \{p_{\min}, \ldots, p_{\max}\}$, $t_b \in \{0, \ldots, T_b\}$, $t_i \in \{0, \ldots, T_i\}$, and $t_m \in \{0, \ldots, T_m\}$.

The *save* action always yields an immediate reward of 1. The *borrow* action takes out a "loan", which gives an immediate reward of 2 and starts a countdown timer t_b from T_b to 0. The agent cannot *borrow* again while $t_b > 0$. When t_b reaches 0, the agent receives a reward of -3, representing repaying the loan with interest. Thus the value of *borrow* is -1, unless the episode will end before the loan is repaid. The *invest* action gives 0 immediate reward, but gives the agent the right to take the *sell* action during a period of time in the future. If *invest* is played at time t, then t_m first counts down from T_m to 0, representing a "maturity" period. When t_m reaches 0, t_i begins counting down from T_i to 0, and the *sell* action is available as long as $t_i > 0$. The *sell* action gives a reward of p, where p is a state variable that evolves randomly over time according to $p \sim \text{DiscreteUniform}\{p_{\min}, p_{\max}\}$. The agent cannot *invest* again while $t_m > 0$ or $t_i > 0$.

We instantiate the Saving problem with parameters $p_{\min} = -4$, $p_{\max} = 4$, $T_i = 4$, and $T_b = 4$, and with an episode length of 30. With these parameters, *invest* is nearly always optimal, but only if the agent takes advantage of the investment period T_i in order to sell the investment for more than $\mathbb{E}[p] = 0$. *Borrow* is almost always the worst action, but the agent must search to a depth of at least T_b to discover its negative consequences.

These parameter choices achieve two goals. First, there is a critical planning horizon of T_b, before which the non-optimal *borrow* action appears to be optimal. This is expected to cause $\top$-FSSS to outperform $\bot$-FSSS for small budgets, since $\top$-FSSS can search deeper with the same budget. Second, *invest* is optimal when it is available and thus $Q^*(s, \textit{invest}) > Q^*(s, \textit{save})$, but there are some abstract policies π — in particular, the optimal policy $\pi^*_\top$ under abstraction $\top$ — for which $Q^{\downarrow\pi}(s, \textit{invest}) < Q^{\downarrow\pi}(s, \textit{save})$. Recall that $\pi^*_\top$ is an *open-loop* policy, meaning that it prescribes a sequence of actions to be taken regardless of random outcomes. The value of *invest* depends on the value that the policy can achieve by later taking the *sell* action. Because $\pi^*_\top$ does not discriminate between states with different values of the price p, the expected immediate reward from taking the *sell* action is $\mathbb{E}[p] = 0$. Thus the optimal policy under $\top$ is to always *save*. When we estimate Q-values using this policy, we find that $Q^{\downarrow\pi^*_\top}(s, \textit{invest}) < Q^{\downarrow\pi^*_\top}(s, \textit{save})$ because *save* gives a larger immediate reward. This is the failure mode of open loop replanning noted by Weinstein and Littman (2012).

The addition of the maturity period T_m extends the original Saving problem described by Hostetler et al. (2015), which is recovered when $T_m = 1$. Our experiments use two versions of Saving, with $T_m = 1$ and $T_m = 3$ respectively. We expect that setting $T_m > 1$ will negatively affect the performance of the breadth-first node selection order (Section 6.1). Because the randomness in the problem is relevant only when the *sell* action is available, refining the abstraction in state nodes where the investment has not yet matured will decrease performance by increasing the size of the tree to no benefit. We would expect the performance of the Uniform and Variance orderings not to be so affected.

8.3.2 Sailing

Sailing is based on a test domain used by Kocsis and Szepesvári (2006) and Jiang et al. (2014). The agent controls a sailboat on a 10×10 grid and must navigate from the starting position at $(0, 0)$ to the goal at $(9, 9)$. The boat can move in 8 directions, and the cost of

a move is the Euclidean distance to the neighboring location divided by the boat's speed. The wind also blows in one of 8 directions, and either stays the same or switches to a neighboring direction uniformly at random every step. Our model of the boat's speed at different relative wind angles is based on data for a Laser Standard dinghy in 10kts wind (Binns, Bethwaite, & Saunders, 2002, Figure 13). We used two variations of Sailing, one in which the grid is empty and one in which random obstacles are placed independently in each square with probability 0.2. We used the same sequence of random problem instances for all of the algorithms to reduce variance. The episode length is 30. The leaf evaluation function returns the Euclidean distance to the goal multiplied by the smallest possible cost-per-move.

8.3.3 Racetrack

Racetrack is a classic domain introduced by Barto, Bradtke, and Singh (1995). The agent controls a racecar in a grid world. Actions alter the velocity of the car by applying accelerations in $\{-1, 0, 1\} \times \{-1, 0, 1\}$. Both components of the acceleration are subject independently to a "slip" probability of 0.2, which causes no acceleration to be applied in that direction. Each time step has a fixed cost of -1, so the agent must get from the start to the goal in as few steps as possible. We used both the Small and Large grid topologies of Barto et al. (1995). The episode length is 30 in the Small topology and 50 in the Large topology. The leaf evaluation function returns the negative of the shortest path distance to the goal scaled to lie in the interval $[-1, 0]$.

8.3.4 Spanish Blackjack

Spanish Blackjack is a more complicated version of the casino game Blackjack. The different rules of Spanish Blackjack cause episodes to be longer on average than in ordinary Blackjack, but the gameplay is otherwise similar. We use an infinite deck so that card counting is not helpful.

8.3.5 Academic Advising

Academic Advising ("Advising") is a modification of the IPC problem of the same name (Guerin, Hanna, Ferland, Mattei, & Goldsmith, 2012). The agent must take and pass all of the required courses in an academic program. The courses are linked by prerequisite relationships, and the chance of passing a course depends on how many of its prerequisites have been passed. We used MDP instance 1 from the IPC 2014. We implemented a generalized problem that has integer grades in the range $\{0, \ldots, g\}$ to increase stochastic branching. The probability of passing a course given prerequisite grades $\{p_1, \ldots, p_n\}$ is

$$\Pr(pass|\{p_i\}) = \eta + (1 - \eta)\frac{\sum_i p_i}{(n + 1)g}.$$

If a course has no prerequisites, the agent passes with probability η_0. If the agent passes, it receives a random grade from DiscreteUniform$\{1, g\}$. The agent receives a penalty of -5 in each step if it has not achieved a grade of g^* in all required courses, and there is an action cost of -1 for taking a course for the first time and -2 for repeating a course. The IPC problem includes a "no-op" action, but we omit this action in our version and instead terminate the episode once all required courses have been passed. In our experiments, we

set $g = 4$, $g^* = 2$, $\eta = 0.2$, and $\eta_0 = 0.8$. The episode length is 40, which is the same as the IPC version.

8.3.6 IPC Crossing Traffic

Crossing Traffic is a grid navigation problem in which the agent must cross several lanes of traffic (obstacles that move right-to-left) without being hit. New obstacles spawn randomly at the rightmost square of each lane, and obstacles exiting the leftmost square are removed. The agent incurs a fixed step cost of -1. We used MDP instance 4 from the IPC 2014.

The IPC Crossing Traffic problem is encoded in a way that is particularly difficult for planning. Hitting an obstacle prevents the agent from moving for the rest of the episode but gives no immediate penalty. Thus a planner cannot recognize that hitting an obstacle is bad unless it has already found a policy that reaches the goal with non-zero probability.

8.3.7 IPC Elevators

In Elevators, the agent controls one or more elevator cars and must use them to pick up and drop off passengers. Passengers arrive stochastically at each floor where they wait until an elevator stops that is going in their desired direction (up or down). Passengers going up get off at the top floor and passengers going down get off at the bottom floor. The agent incurs a penalty for each passenger that is not at its destination. We used MDP instance 7 from the IPC 2014.

Due to the limitations of the domain description language used for the IPC (RDDL; Sanner, 2010), the Elevators domain does not track the *number* of passengers waiting or in an elevator. Thus its stochastic branching factor is lower than might be expected. The problem even becomes deterministic when all floors have passengers waiting and no one boards an elevator, since further random arrival events at occupied floors have no effect.

8.3.8 IPC Tamarisk

In Tamarisk, the agent is trying to prevent the invasive *tamarisk* plant from colonizing a river system. The world is a directed graph of *reaches*, each of which have a fixed number of *slots* that each can be either unoccupied, occupied with a *native* plant, or occupied with a *tamarisk* plant. Plants spread stochastically to unoccupied slots with a much higher probability of spreading downriver. At each time step, the agent can *eradicate* a reach, *restore* a reach, or do nothing. The *eradicate* action changes each *tamarisk* slot to *empty* independently with a fixed probability. The *restore* action stochastically changes *empty* slots to *native*, which prevents tamarisk plants from growing there. There is a per-slot and per-reach penalty for the presence of tamarisk plants as well as action costs for non-default actions. We used MDP instance 2 from the IPC 2014.

8.3.9 Tetris

Tetris is the classic videogame of stacking differently shaped blocks. It has quite a long history in AI research (e.g. Bertsekas & Ioffe, 1996; Gabillon, Ghavamzadeh, & Scherrer, 2013). Whereas in the Tetris video game the player's actions translate or rotate the falling block by one step, in our version the agent positions the block in the top row at any

horizontal position and with any rotation, and the block then immediately drops to the bottom. This change makes the problem easier for tree search because it greatly reduces plan lengths. The agent receives a reward of 1 each time it "clears" a row of blocks. An episode terminates if an action causes two blocks to overlap, which becomes unavoidable as the screen fills with uncleared blocks. The shape and initial orientation of the next block to appear is chosen uniformly at random, thus the agent must average over possible future sequences in order to find the best location for the current block. We use the popular "Bertsekas features" (Bertsekas & Ioffe, 1996) as the ground representation. The episode length is 100.

8.4 Methods

Since we are interested in *anytime* online planning, we compare the algorithms on each domain for a range of sample budgets. Let $\rho_M(A, b; \theta)$ denote the empirical average return of algorithm A running on problem M with budget b and parameters θ. These averages are based on between 1000 and 10000 random trajectories, depending on the problem. Given a problem M and range of budgets $\mathcal{B} = \{b_1, \ldots, b_n\}$, we compute $\rho^*_M(A, b) = \max_{\theta \in \Theta} \rho_M(A, b; \theta)$ for each algorithm A and each $b \in \mathcal{B}$.[4] The parameter search space Θ covers a range of values of the width parameter C and depth parameter d. For the RANDOM abstraction, Θ also covers different settings of the stochastic branching factor B.

For most problems, $C \in \{1, 2, 5, 10, 20, 50\}$, and $d \in \{i, i + 1, \ldots, i + m\}$ where i and m are small integers. We expanded the range of C to $\{1, 2, 5, \ldots, 100, 200\}$ for Spanish Blackjack due to its large stochastic branching factor. The specific ranges of d were chosen based on pilot experiments. We attempted to expand the range of d until the best value of d was not at either extreme of the range, but this was not feasible in all problems due to memory limits. The random branching factor B was varied over either $\{2, 3, 5\}$ or $\{2, 4\}$ depending on the problem. Early experiments used $\{2, 3, 5\}$, while in later experiments we reduced this to $\{2, 4\}$ to limit the parameter search space.

In real-world applications, planning budgets typically will be given in units of time rather than in numbers of samples. Thus, ultimately we are interested in the algorithms' performance with a time budget. We use sample budget as a proxy for time budget for reasons of practicality. Time budget experiments take much longer to run in our computing environment due to the expensive system calls needed to measure execution time accurately. The computational cost of these experiments is substantial due to the need to search for the best parameter settings and to average over many random execution trajectories. Focusing on sample budgets allowed us to examine more problems and to more thoroughly optimize the algorithm parameters. We discuss this issue further and provide empirical evidence supporting sample budget as a proxy in Section 9.9.

4. Hostetler et al. (2015) compared algorithms using a different criterion, similar in form to $\rho^*_M(A) = \max_{\theta \in \Theta} \sum_{b \in \mathcal{B}} \rho_M(A, b; \theta)$. The $\rho^*_M(A)$ criterion selects a single parameter set that performs best over all budgets simultaneously, whereas $\rho^*_M(A, b)$ optimizes parameters separately for each budget. We have come to view $\rho^*_M(A, b)$ as the superior criterion, primarily because in $\rho^*_M(A)$ the parameter selection is sensitive to the range of values spanned by the budgets in $\mathcal{B}$. It would be unusual in practice to require a planning algorithm to perform well over multiple orders of magnitude of the search budget with the same parameters. Thus we find the "sum of maxes" criterion $\rho^*_M(A, b)$ to be more realistic.

9. Results

Our results support three main conclusions. First, that PARSS performed better overall than any of the algorithms that used static representations. Second, that $\top$-FSSS often but not always outperformed $\bot$-FSSS, and thus that some (but not all) of the advantage of PARSS likely comes from its utilization of $\top$-FSSS as a starting point. These results are consistent with earlier experimental results with PARSS (Hostetler et al., 2015). Third, the choice of Select and Refine implementations affects the performance of PARSS. In particular, the combination Variance+DT appears to be best when considering the entire problem set, while Uniform+Random is worst.

The comparisons between algorithms are summarized in Figures 5, 6, and 7. The error bars in these figures are 95% confidence intervals. Note when interpreting these charts that some of the algorithms are equivalent for certain parameterizations. For example, if $C = 1$ then all of the algorithms are equivalent. Rather than conducting identical experiments for multiple equivalent algorithms, we instead run a single experiment and proceed as though all of the equivalent parameterizations produced exactly that result. When the lines in the charts overlap exactly, it is because the overlapping algorithms are equivalent under their best parameterization for that problem and budget.

9.1 Performance of PARSS

PARSS was the best algorithm overall in five problems: the two Saving problems, the two Racetrack problems, and Advising (Figure 5). In all other problems (Figures 6 and 7), PARSS performed as well as the best alternative algorithm. In Saving, we see that $\top$-FSSS plateaus at a suboptimal value, while $\bot$-FSSS converges more slowly than PARSS. $\top$-FSSS also plateaus in Racetrack Large but $\bot$-FSSS surpasses it only for the largest budgets. Presumably a similar pattern would be apparent in Racetrack Small if that experiment were to be continued to larger budgets.[5] In Advising, all of the algorithms improve steadily with increasing budgets, but PARSS is consistently best. It is noteworthy that $\top$-FSSS is *worse* than $\bot$-FSSS in Advising, and thus that the superior performance of PARSS is due to some other factor besides starting from $\top$-FSSS.

9.2 Performance of $\top$-FSSS

There were five problems in which $\top$-FSSS outperformed $\bot$-FSSS over most of the range of budgets (Figure 6). $\top$-FSSS also performed well in the two Racetrack problems (Figure 5), although its performance began to plateau at larger budgets. We would expect this plateau to occur in most problems if we continued the experiments to sufficiently large budgets, since the optimal ground policy usually will not be representable in the $\top$-abstract state space.

$\top$-FSSS was generally inferior to $\bot$-FSSS on the Saving and Sailing problems, and to some extent also in Advising. This was the expected result in Saving because $\top$-FSSS cannot estimate the value of the *invest* action correctly. Note however that $\top$-FSSS is superior to $\bot$-FSSS for the smallest budgets because $\bot$-FSSS estimates the value of

5. The Racetrack Small problem actually has a longer critical horizon than Racetrack Large because the longest straight section of the track is longer and thus the agent's top speed is higher.

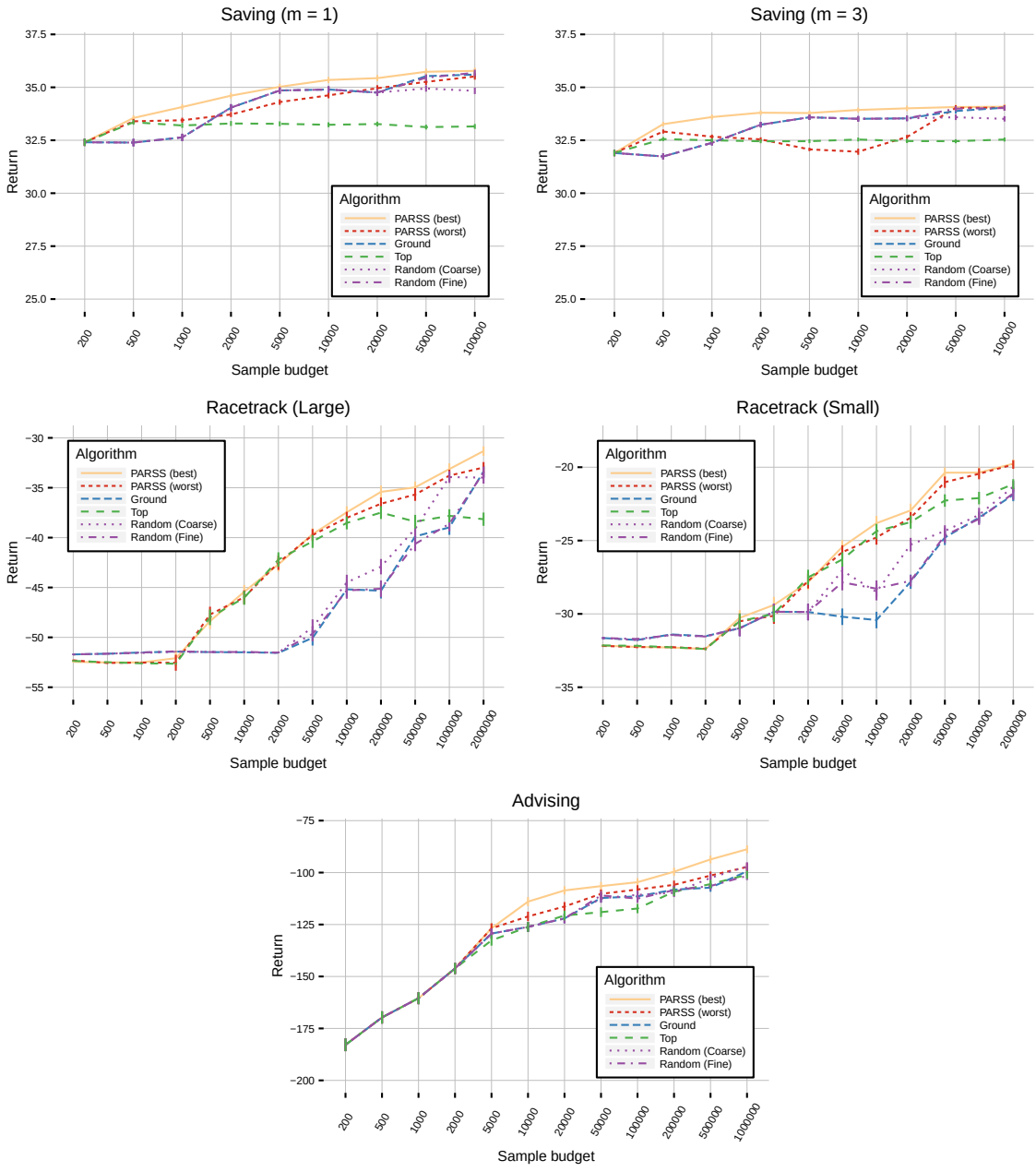


Figure 5: Problems where PARSS outperformed all other algorithms.

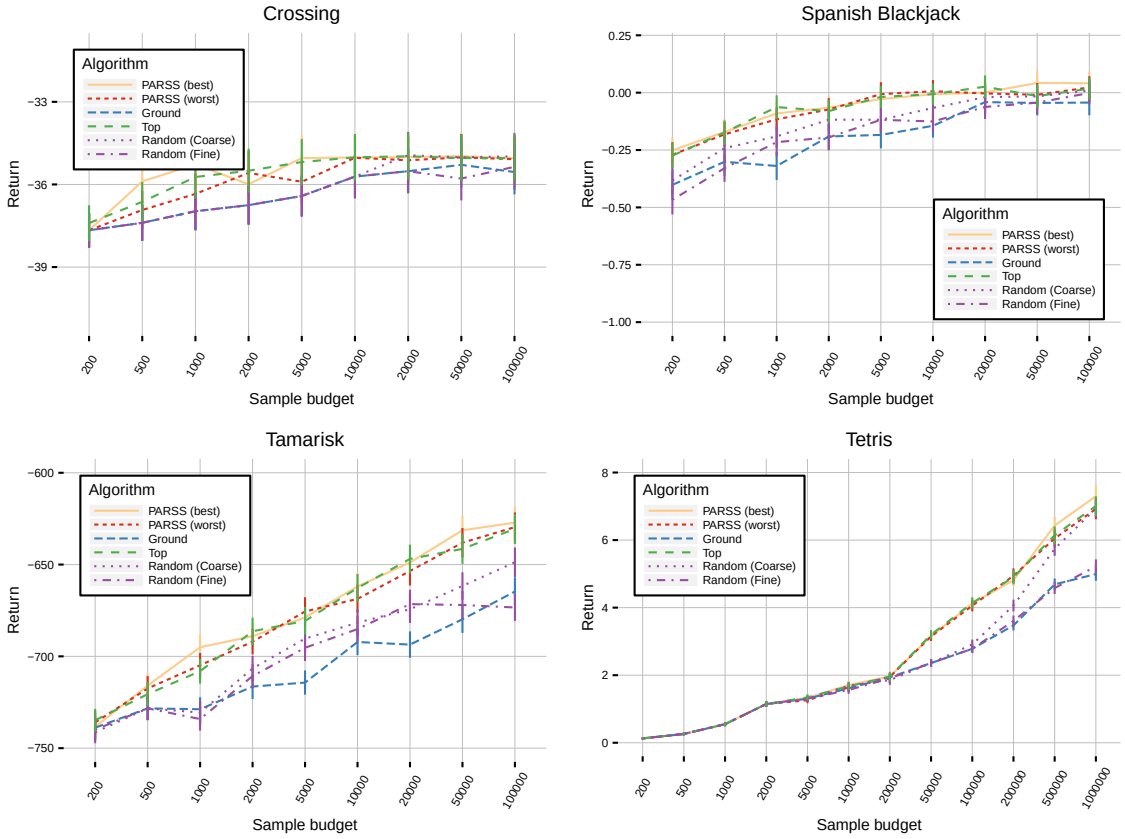


Figure 6: Problems where ⊤-FSSS outperformed Ground and Random. Note that all PARSS variants performed equally as well as ⊤-FSSS.

borrow incorrectly due to horizon effects. In Sailing, ⊤-FSSS cannot account for the randomly shifting wind and so its policy will always sail more or less directly toward the goal. This accounts for its flat performance curve.

9.3 Performance of ⊥-FSSS

⊥-FSSS performed well on the two Sailing problems and on Elevators (Figure 7). We attribute this performance to the fact that these domains have the smallest stochastic branching factors. In Sailing, the branching factor is 3, while in Elevators it could be as high as 2^6 if no passengers are waiting or as low as 1 if a passenger is waiting at every floor. The fact that ⊥-FSSS with $C = 1$ was the best parameterization for Elevators would seem to indicate that the latter, near-deterministic situation is common. Note that PARSS did equally as well as ⊥-FSSS in these domains.

9.4 Comparing PARSS Variations

We can see from Figures 5, 6, and 7 that the gap between the best and worst variations of PARSS tends to be small. Only in the two variations of Saving and in Advising is the

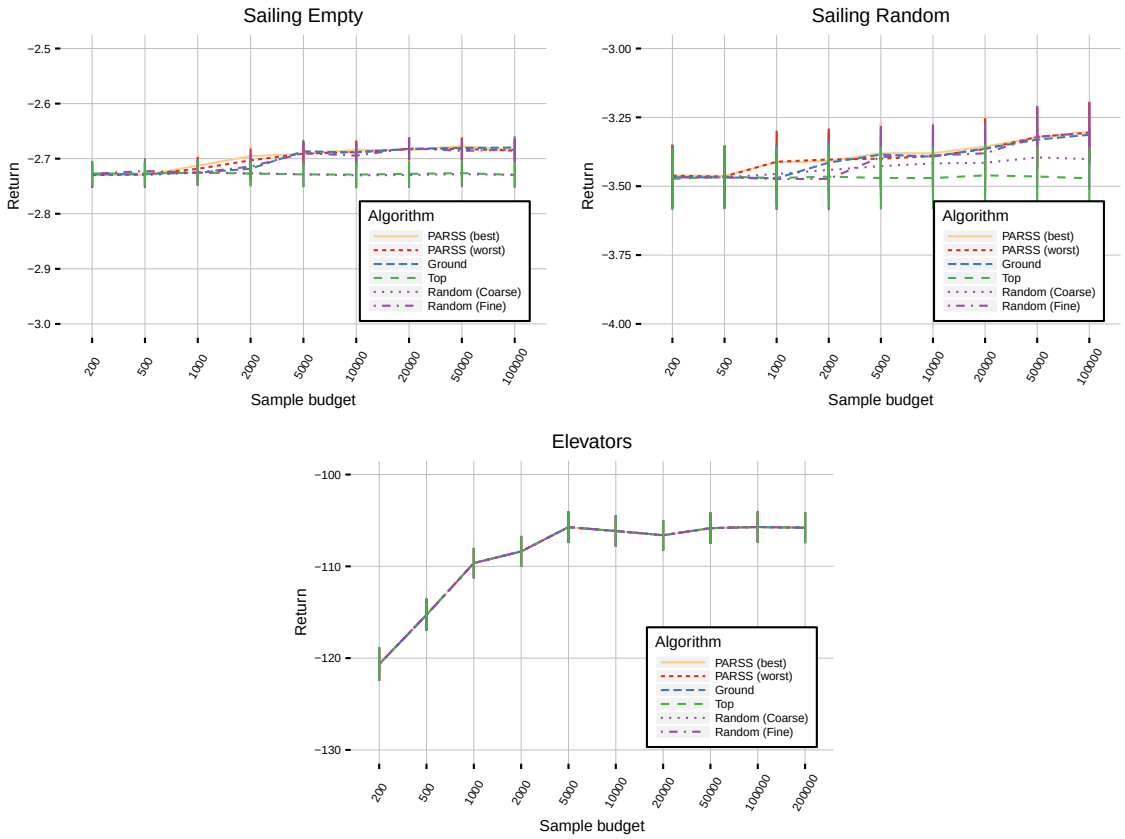


Figure 7: Problems where $\perp$-FSSS was best. Note that all PARSS variants performed equally as well as $\perp$-FSSS. In the Elevators problem, the best performance occurred when the width parameter was $C = 1$. Since all the algorithms are equivalent if $C = 1$, the results shown are identical.

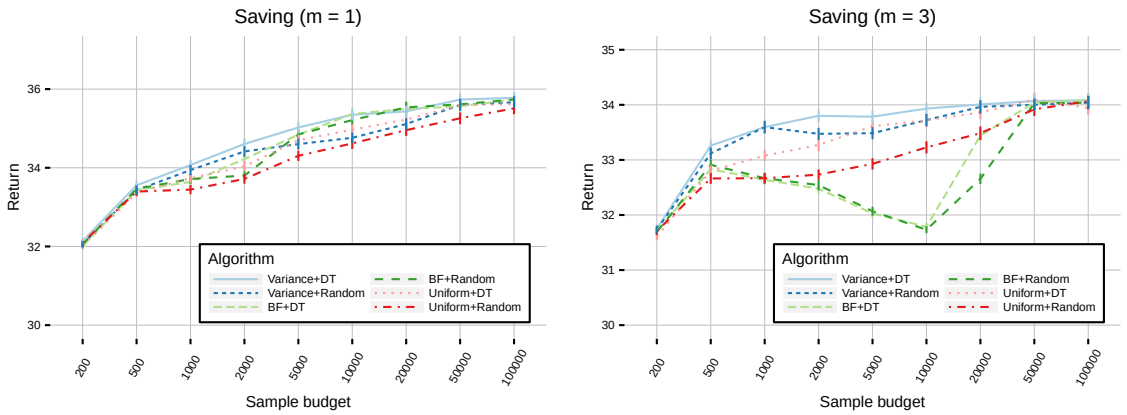


Figure 8: Comparing all PARSS variations on the Saving domain. The BF order performs poorly when $T_m = 3$ because it refines many abstraction relations that are already sound.

difference between PARSS variations comparable to the difference between PARSS and $\perp$-FSSS. In Saving, this is because the problem is designed to favor PARSS in general, and to favor the Variance priority ordering specifically when $m > 1$. Figure 8 compares the performance of the six PARSS variants on the Saving problem. Increasing the maturity period to $m = 3$ had the expected result of causing the BF selection order to perform poorly, because it refines abstractions near the root that are already sound.

We made a statistical comparison of the overall relative performance of the six PARSS variations using Friedman's test (Demšar, 2006), which is a non-parametric test based on rank comparison that detects an overall effect of the choice of algorithm on performance across multiple experiments. We consider each combination of problem plus sample budget as a separate "experiment" for the purpose of the test, giving a total of 123 experiments. We thus compare the PARSS variants on performance across all sample budgets and problems. The test showed strong support for an overall effect of PARSS variation on performance ($F(5, 610) = 10.06$, $p < 10^{-8}$).

After determining that an overall effect of algorithm choice exists in the results, we examined the pairwise differences among the algorithms using Nemenyi's test (Demšar, 2006), which is a *post hoc* test based on the studentized range distribution with a correction for multiple comparisons. These pairwise comparisons are summarized in Figure 9 using a *critical difference plot* (Demšar, 2006). While there was no single best or worst algorithm (at the $p = 0.05$ level), we can see that in general the Uniform selection order performed poorly. Variance+DT outperformed the largest number of other algorithms, but both BF+DT and BF+Random had identical performance to Variance+DT whereas Variance+Random was worse than Variance+DT. It may be that the breadth-first selection order is less sensitive to the choice of refinement mechanism because the BF order fully refines all nodes at depth d before refining any at depth $d + 1$. This results in a larger number of state nodes becoming fully-refined compared to other selection orders, and in fully-refined state nodes the refinement mechanism is no longer relevant.

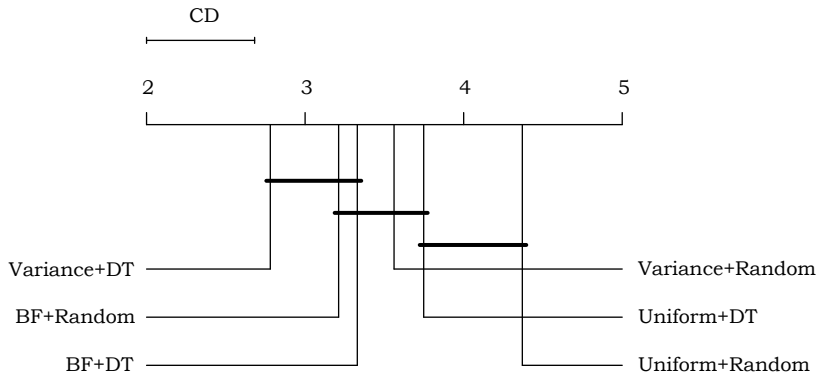


Figure 9: A critical difference plot (Demšar, 2006) showing the pairwise differences in performance among the PARSS variants. The horizontal scale shows the average rank of each algorithm, with smaller ranks indicating better performance. Algorithms connected by a dark line had statistically identical performance at the $p = 0.05$ level. This plot was produced by the R package `scmamp` (Calvo & Santafe, 2015).

9.5 Performance of rand-FSSS

The random abstractions typically had intermediate performance between $\bot$-FSSS and $\top$-FSSS. It is useful to compare these results to the performance of the Uniform+Random version of PARSS. The Uniform+Random variant may well produce intermediate trees with similarly inaccurate abstractions as rand-FSSS, but although Uniform+Random was the worst PARSS variant overall, rand-FSSS seldom outperformed it on any given problem. This suggests that the tree exploration dynamics of PARSS may be qualitatively different from those of AFSSS with a fixed abstraction.

Specifically, since PARSS begins by building a $\top$-FSSS tree until convergence, the first state nodes to be refined by PARSS will be nodes that were *not* pruned by $\top$-FSSS. This results in an implicit bias in the order in which the progressivly more-complex trees are examined. Each refinement occurs in a state that was not pruned (or that became un-pruned) in a previous step. It is possible that the dynamics of PARSS result in more effective pruning and thus better-focused exploration than the dynamics of search with a fixed abstraction, even if the final search trees are similar in terms of abstraction accuracy and average branching factor.

9.6 Stochastic Branching Factor vs. Best Algorithm

The four problems where $\top$-FSSS was (tied for) best were also the four problems with by far the largest minimum stochastic branching factors (Table 1). This suggests that it is the raw reduction in tree size that plays a key role in the strong performance of $\top$-FSSS. The $\top$-FSSS search is able to average over more random outcomes while still searching to a reasonably large depth. There is no clear trend between branching and algorithm performance in the other problems, suggesting that the performance gap between PARSS and $\bot$-FSSS on some of these problems is due to other structural features of the problem in addition to the branching factor.

Problem	Branching Min	Branching Max	
Saving	9	9	PARSS best
Racetrack	1	4	
Advising	5	5	
Crossing	2^5	2^5	PARSS = $\top$-FSSS
Blackjack	52	$\approx 52^4$	
Tamarisk	2^{12}	3^{12}	
Tetris	40	40	
Sailing	3	3	PARSS = $\bot$-FSSS
Elevators	1	2^6	

Table 1: Minimum and maximum stochastic branching factors of the experimental domains. Note that the maximum branching factor of Spanish Blackjack might be higher than 52^4, but this occurs only when completing the dealer's hand and only extremely rarely.

9.7 Explaining the Performance of $\top$-FSSS

Some of the performance advantage of PARSS can be attributed to the fact that PARSS begins as a $\top$-FSSS search, and $\top$-FSSS often performs well by itself. We expect $\top$-FSSS to do well when rollout with the right open-loop policy will correctly rank the values of root actions. We can interpret $\top$-FSSS as a policy rollout algorithm (Eq. 1) that uses an approximately optimal open-loop policy as its evaluation policy. It is possible for the policy rollout agent to behave optimally even if the evaluation policy π is not optimal, provided that $\arg\max_{a \in \mathcal{A}} Q^*(s,a) = \arg\max_{a \in \mathcal{A}} \hat{Q}^\pi(s,a)$. In Spanish Blackjack, for example, the simple evaluation policy $\pi(s) = \textit{pass}$ will correctly evaluate the majority of *hit vs. pass* decisions. Although optimal play may dictate hitting more than once, in such cases hitting once and then passing is often still better than passing immediately.

PARSS improved upon $\top$-FSSS in 7 of the 12 problems. To explain this improvement, we can begin by noting that these problems exhibit aspects of the Weinstein-Littman structure (Weinstein & Littman, 2012), which is problematic for open-loop replanning. The essence of the Weinstein-Littman structure is that the optimal action can give worse return than a different action if it is not followed up by additional correct actions. The Saving problems were designed to ensure that $\top$-FSSS could not be optimal by explicitly including this structure. In Racetrack, the optimal agent accelerates to as high a speed as possible before braking for a turn. Since braking actions fail stochastically, the best open-loop "braking policy" must be conservative and plan to execute enough consecutive braking actions to stop the car even if several actions fail. If these braking actions end up not failing, the car is left moving slowly or even moving backwards. The result is that the agent underestimates the value of driving faster. A similar effect occurs in Sailing, where it may be optimal to sail away from the goal temporarily in order to align the remaining path to the goal with the likely wind direction.

The results in Advising are somewhat different, in that both $\top$-FSSS and $\bot$-FSSS have similar performance while PARSS is superior. Examining the best parameters for each algorithm reveals that $\top$-FSSS is able to search with larger width and depth parameters (C

and d) than $\perp$-FSSS for the same budgets. While $\top$-FSSS incurs error due to abstraction (Theorem 1), $\perp$-FSSS estimates state node values from a smaller number of samples and thus may incur error from the higher variance in its value estimates. PARSS may get the best of both worlds in this domain, benefiting from the increased depth of $\top$-FSSS as well as the decreased abstraction error due to refinement.

9.8 Memory Consumption and Large Action Spaces

We encountered practical difficulties in Advising and especially in Tetris due to the relatively large size of the action set ($|\mathcal{A}| = 10$ in Advising and $|\mathcal{A}| = 40$ in Tetris). None of our algorithms make any attempt to reduce action branching. In the Tetris experiments, the parameter search space Θ had to be curtailed because the search algorithms were exceeding the 16GB memory limit of our hardware. Integrating both state and action abstraction in the same algorithm is critical for scaling up to these types of problems, and should be a focus of further work in abstract tree search.

More generally, the main disadvantage of PARSS compared to AFSSS with a fixed abstraction is that PARSS must retain more ground states in memory in case the abstract state node that contains them is later chosen for refinement. When searching with a fixed abstraction, the ground states associated with internal tree nodes can be discarded, since no more successors will be drawn for those interior nodes (Section 5.4). Algorithms like recursive best-first search (Korf, 1993) reduce memory usage by discarding tree nodes that are not needed currently and regenerating them later if they are needed. This idea could be incorporated into PARSS. It would be best applied to nodes on the search frontier in PARSS, since a large proportion of state nodes are on the frontier and such nodes have no descendants that would also need to be resampled.

9.9 Sample Budgets vs. Time Budgets

We earlier made the claim that a sample budget is a reasonable proxy for a "wall clock" time budget; we now present experimental evidence in support of this claim. It is important not to over-interpret these results. Our experiments were conducted on a shared computer cluster comprised of heterogeneous hardware, and timing results obtained on different compute nodes under different load profiles are not necessarily comparable. Thus there may be systematic biases in the data. The methods we now describe are intended to mitigate some of these biases.

During our experiments with sample budgets, we measured the mean time spent building individual search trees using the Java API function `System.currentTimeMillis()`, as well as the mean number of transition samples used to build the trees. From these two values, we computed the *sample rate* (samples per millisecond). We restricted our attention to the best parameterizations of each algorithm as described in Section 8.4. For each domain, we first found all values of the sample budget for which all search algorithms had a mean search time of at least 10ms. This is intended to exclude timing results that might be excessively affected by the limited accuracy of the system time.[6] We then averaged the mean sample rate over all such budgets for each algorithm. Averaging over search budgets should mitigate, to some

6. The accuracy of `System.currentTimeMillis()` is typically about 1ms in Linux systems like the ones used for our experiments.

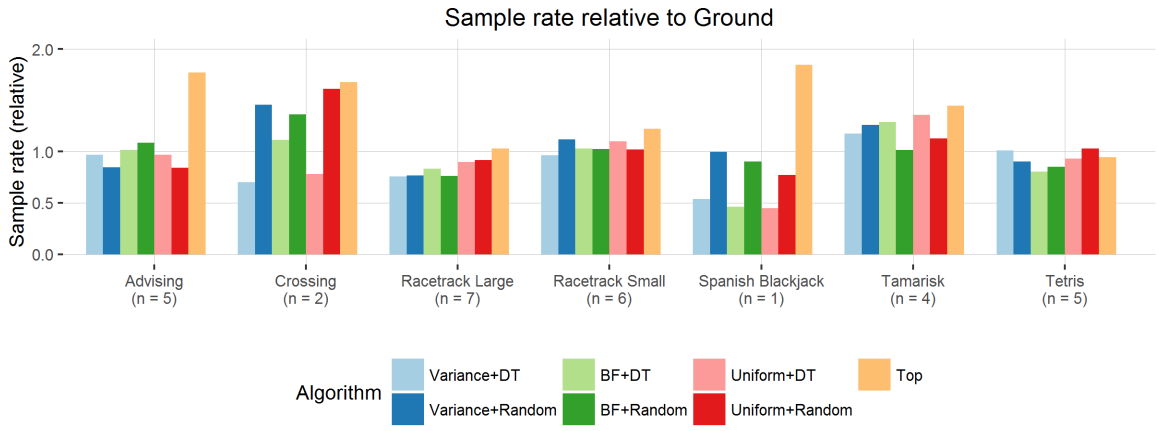


Figure 10: Sample rate relative to ⊥-FSSS on a subset of the experimental problems. The results for each problem are based on the number of data points listed in parentheses.

extent, bias due to different computer hardware, since the experiments for each budget are separate processes that might be scheduled on different machines.

Figure 10 shows the sample rate for each domain that had sufficient data satisfying the above criteria as a proportion of the sample rate of ⊥-FSSS (i.e. ⊥-FSSS has a sample rate of 1). These results support the conclusion that the sample rate of the PARSS algorithms is comparable to that of ⊥-FSSS, and thus that a sample budget is a reasonable proxy for a time budget for the purpose of comparing PARSS to ⊥-FSSS. This is consistent with earlier results indicating that the relative performance of the algorithms was similar for both time and sample budgets (Hostetler et al., 2015), although those experiments used different evaluation criteria.

9.10 Summary of Results

The experimental results indicate that PARSS is superior or equal to ⊥-FSSS on a range of problems in terms of performance with a sample budget. Although we did not compare the algorithms' performance with a time budget, previous experiments (Hostetler et al., 2015) have indicated that this pattern of relative performance remains the same in the time budget setting, and empirically the PARSS algorithms expend a similar amount of time per sample as ⊥-FSSS. Since PARSS provides the same bounded error guarantees as ⊥-FSSS (Proposition 8), there seems to be little reason not to use PARSS in preference to FSSS (Walsh et al., 2010) and ordinary SS (Kearns et al., 2002). Among the PARSS variants, Variance+DT was consistently the best combination of node selection and refinement criteria, and thus seems to be a good default choice among general-purpose heuristics.

10. Summary and Future Work

This paper consolidates and extends earlier work on state abstraction and progressive abstraction refinement in sample-based tree search. We first presented an expanded discussion of a theoretical bound on simple regret due to abstraction in tree search as well as algo-

rithms for abstract sparse sampling (Kearns et al., 2002) and UCT (Kocsis & Szepesvári, 2006) that first appeared in the work of Hostetler et al. (2014). This analysis provides guidance for designing or computing abstractions and reveals differences in the interaction of abstraction with sparse sampling *vs.* trajectory sampling search algorithms. We then described the Progressive Abstraction Refinement for Sparse Sampling (PARSS) algorithm (Hostetler et al., 2015), which addresses the problem of choosing the correct abstraction by progressively refining an initially coarse abstraction during search. Our analysis of PARSS showed that it provides the same finite sample performance guarantees as SS and FSSS. We compared the original PARSS algorithm of Hostetler et al. (2015) as well as 5 new variants of PARSS to FSSS with a variety of fixed abstractions (including the ground abstraction) on a set of 12 decision-making problems, and found that PARSS outperformed FSSS. Drawbacks of PARSS include additional implementation complexity and sometimes a higher memory footprint.

Progressive abstraction refinement is a promising basis for new kinds of tree search algorithms. Our immediate future work will investigate ways of incorporating new forms of abstraction, including action pruning (Pinto & Fern, 2014) and temporal abstractions such as options (Sutton et al., 1999; Bai et al., 2015), into the progressive refinement search framework. Progressive refinement algorithms based on trajectory sampling should also be explored and compared to other abstract TS algorithms such as OGA-UCT (Anand et al., 2016). The abstraction refinement framework provides a new mechanism for algorithms that "learn to plan." Such algorithms could learn to control the refinement process to make it more effective. Data about abstraction refinements could also be collected and used to improve the ground representation of the problem over time, facilitating the learning of reactive policies — which are highly sensitive to representation — for fragments of the larger problem.

Acknowledgments

This research was supported by NSF grants IIS 1320943, 0958482, and 1619433. We thank the anonymous reviewers for their insightful criticism, which improved both the clarity and completeness of the paper.

Appendix A. Proofs

A.1 Proof of Theorem 1

Theorem 1. *Let $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ be a history MDP such that the maximum length of a history in $\mathcal{H}$ is $d = \max_{h \in \mathcal{H}} \ell(h)$. Let $\alpha = \langle \chi, \mu \rangle$ be an abstraction of T where χ is (p, q)-consistent and let $\delta \stackrel{\text{def}}{=} \delta_{T/\alpha}$. For any action $a \in \mathcal{A}$,*

$$\left| Q^*(s_0, a) - \mathcal{Q}^*_\alpha(\{s_0\}, a) \right| \leq \beta_\gamma(d)(p + \delta q).$$

Proof. The proof is by structural induction on the tree of abstract histories, from the leaf states upwards. Let Ω denote the set of leaf states, $\Omega = \{H \in \mathcal{H}/\chi : \forall H' \in \mathcal{H}/\chi \; . \; H \neq$

$p(H')\}$. We generalize the desired error bound to apply to abstract states,

$$E(H,a) = \left|\mathcal{Q}^*_\alpha(H,a) - \sum_{h\in H}\mu(H,h)Q^*(h,a)\right|.$$

Note that $E(\{s_0\},a) = |\mathcal{Q}^*_\alpha(\{s_0\},a) - Q^*(s_0,a)|$.

Base case Consider a terminal state $H \in \Omega$. Since terminal states have no successors, we have

$$E(H,a) = \left|\mathcal{Q}^*_\alpha(H,a) - \sum_{h\in H}\mu(H,h)Q^*(h,a)\right| = \left|\mathcal{R}_\mu(H) - \sum_{h\in H}\mu(H,h)R(H)\right| = 0. \quad (12)$$

Inductive step We now consider interior states $H \in \overline{\Omega}$ and assume the inductive hypothesis $E(H',a') \le \beta_\gamma(k)(p+\delta q)$ for all $H' \in K(H,a)$ and all $a' \in \mathcal{A}$, where $K(H,a) = \{H' \in \mathcal{H}/\chi : p(H') = H, a(H') = a\}$ is the set of successors of Ha. Since the immediate reward terms do not affect $E(H,a)$ (Eq. 12), the difference between the optimal value in the abstract tree and the true optimal value is the error in the discounted future return estimates,

$$E(H,a) = \gamma\Big|\sum_{H'\in\mathcal{H}/\chi}\mathcal{P}_\mu(H'|H,a)\mathcal{V}^*_\alpha(H') - \sum_{h\in H}\mu(H,h)\sum_{h'\in\mathcal{H}}P(h'|h,a)V^*(h')\Big|.$$

We decompose the error as $E(H,a) \le \gamma(E_Q + E_\chi)$, where,

$$E_Q = \Big|\sum_{H'\in\mathcal{H}/\chi}\mathcal{P}_\mu(H'|H,a)\mathcal{V}^*_\alpha(H') - \sum_{H'\in\mathcal{H}/\chi}\mathcal{P}_\mu(H'|H,a)\sum_{h'\in H'}\mu(H',h')V^*(h')\Big|$$
$$E_\chi = \Big|\sum_{H'\in\mathcal{H}/\chi}\mathcal{P}_\mu(H'|H,a)\sum_{h'\in H'}\mu(H',h')V^*(h') - \sum_{h\in H}\mu(H,h)\sum_{h'\in\mathcal{H}}P(h'|h,a)V^*(h')\Big|.$$

E_Q is the error due to using the abstract value function below the current node. E_χ is the error introduced by aggregating states at the current level.

We analyze E_Q first. By $(p,\cdot)$-consistency of χ, we have the bound

$$\sum_{h'\in H'}\mu(H',h')\max_{a'\in\mathcal{A}}Q^*(h',a') - \max_{a'\in\mathcal{A}}\sum_{h'\in H'}\mu(H',h')Q^*(h',a') \le p. \quad (13)$$

Note that this difference is non-negative. We relate this to $E(H',a')$ by observing that for any $H' \in \mathcal{H}/\chi$,

$$\Big|\max_{a'\in\mathcal{A}}\mathcal{Q}^*_\alpha(H',a') - \max_{a'\in\mathcal{A}}\sum_{h'\in H'}\mu(H',h')Q^*(h',a')\Big| \le \max_{a'\in\mathcal{A}}E(H',a'), \quad (14)$$

because of the general fact that $|\max_x f(x) - \max_x g(x)| \le \max_x |f(x) - g(x)|$ for real-valued functions f and g on the same domain. Combining (13) and (14) with the triangle inequality, we have

$$\Big|\max_{a'\in\mathcal{A}}\mathcal{Q}^*_\alpha(H',a') - \sum_{h'\in H'}\mu(H',h')\max_{a'\in\mathcal{A}}Q^*(h',a')\Big| \le p + \max_{a'\in\mathcal{A}}E(H',a').$$

Applying the inductive hypothesis, we conclude that

$$\Big|\max_{a'\in\mathcal{A}} \mathcal{Q}^*_\alpha(H',a') - \sum_{h'\in H'} \mu(H',h') \max_{a'\in\mathcal{A}} Q^*(h',a')\Big| \leq p + \beta_\gamma(k)(p+\delta q)$$

for any $h' \in \mathcal{H}$. We then plug this bound into E_Q to obtain

$$E_Q = \Big|\sum_{H'\in\mathcal{H}/\chi} \mathcal{P}_\mu(H'|H,a)\Big[\mathcal{V}^*_\alpha(H') - \sum_{h'\in H'} \mu(H',h')V^*(h')\Big]\Big| \leq p + \beta_\gamma(k)(p+\delta q).$$

We now analyze the single-step abstraction error E_χ. This error comes from assigning incorrect weights to ground states within the current abstract state. We can write the second part of E_χ in terms of the exact update of the weight function (Eq. 5),

$$\begin{aligned}
&\sum_{h\in H} \mu(H,h) \sum_{h'\in\mathcal{H}} P(h'|h,a)V^*(h') \\
&= \sum_{H'\in\mathcal{H}/\chi} \sum_{h'\in H'} \Big[\sum_{h\in H} \mu(H,h)P(h'|h,a)\Big]V^*(h') \\
&= \sum_{H'\in\mathcal{H}/\chi} \sum_{h'\in H'} \mathcal{P}_\mu(H'|H,a) \frac{\Big[\sum_{h\in H} \mu(H,h)P(h'|h,a)\Big]}{\mathcal{P}_\mu(H'|H,a)} V^*(h') \\
&= \sum_{H'\in\mathcal{H}/\chi} \mathcal{P}_\mu(H'|H,a) \sum_{h'\in H'} W\mu(H',h')V^*(h').
\end{aligned}$$

We can then express E_χ as

$$E_\chi = \sum_{H'\in\mathcal{H}/\chi} \mathcal{P}_\mu(H'|H,a)\Big|\sum_{h'\in H'} \mu(H',h')V^*(h') - \sum_{h'\in H'} W\mu(H',h')V^*(h')\Big|.$$

Let $D(H')$ denote the difference in values that appears in E_χ,

$$D(H') = \Big|\sum_{h'\in H'} \mu(H',h')V^*(h') - \sum_{h'\in H'} W\mu(H',h')V^*(h')\Big|.$$

Let $v(H) = \min_{h\in H} V^*(h)$ be the minimum value among states in H. By $(\cdot,q)$-consistency of χ, we have $0 \leq V^*(h) - v(H) \leq q$ for all $h \in H$. Let $\Delta(H,h) = V^*(h) - v(H) - \frac{q}{2}$ denote this difference in value shifted to lie in the interval $[-q/2, q/2]$. We now express $D(H')$ in terms of Δ,

$$\begin{aligned}
D(H') &= \Big|\sum_{h'\in H'} \mu(H',h')[v(H') + \frac{q}{2} + \Delta(H',h')] - \sum_{h'\in H'} W\mu(H',h')[v(H') + \frac{q}{2} + \Delta(H',h')]\Big| \\
&= \Big|\sum_{h'\in H'} \mu(H',h')\Delta(H',h') - \sum_{h'\in H'} W\mu(H',h')\Delta(H',h')\Big| \\
&\leq \sum_{h'\in H'} \Big|\Delta(H',h')\big[\mu(H',h') - W\mu(H',h')\big]\Big| \\
&\leq \frac{q}{2} \sum_{h'\in H'} \Big|\mu(H',h') - W\mu(H',h')\Big| \\
&= q\cdot\frac{1}{2}\big\|\mu(H',\cdot) - W\mu(H',\cdot)\big\|_1 = q\delta(\mu,H') \leq \delta q.
\end{aligned}$$

Since E_χ is a convex combination of $D(H')$ for different H', we conclude that $E_\chi \leq \delta q$.

Combining the two sources of error, we obtain,

$$E_Q + E_\chi \leq \beta_\gamma(k)(p + \delta q) + p + \delta q = (1 + \beta_\gamma(k))(p + \delta q). \tag{15}$$

Since $E(H, a) \leq \gamma(E_Q + E_\chi)$, we multiply (15) by the discount factor γ to obtain

$$E(H, a) \leq \gamma(1 + \beta_\gamma(k))(p + \delta q) = \beta_\gamma(k + 1)(p + \delta q)$$

for all $H \in \mathcal{H}/\chi$ such that $\ell(H) = d - k$, for all $a \in \mathcal{A}$. This completes the inductive argument, and we conclude that in the root state $\{s_0\}$, $E(\{s_0\}, a) \leq \beta_\gamma(d)(p + \delta q)$ for all $a \in \mathcal{A}$.

□

A.2 Proof of Proposition 2

Proposition 2. *Let $T = \langle \mathcal{H}^\infty, \mathcal{A}, P, R, \gamma, s_0 \rangle$ be a (possibly infinite) history MDP and let χ be a (p, q)-consistent history equivalence relation on $\mathcal{H}^d$ for some depth parameter d. Then the procedure* AbstractSS(s_0, C, d, χ)*, with probability at least $1 - (|\mathcal{A}|C)^d \cdot 2e^{-2\lambda^2 C/(V^d_{\max})^2}$, returns an action choice a^* such that*

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\Big[\beta_\gamma(d)(\lambda + p + \delta q) + \gamma^d V^d_{\max}\Big],$$

where δ is the divergence (Eq. 6) of the completed empirical weight function $\bar{\mu}^+$ derived from the empirical weight function $\bar{\mu}$ computed by AbstractSS.

Proof. The proof is a small modification of the analysis of SS by Kearns et al. (2002). Let $\hat{\mathcal{Q}}$ and $\hat{\mathcal{V}}$ denote the value functions estimated by AbstractSS(s_0, C, d, χ). Recall the definition of the completed empirical weight function (Eq. 8),

$$\bar{\mu}^+(H, h) = \begin{cases} \bar{\mu}(H, h) & \text{if } H \in \mathbf{dom}(\bar{\mu}), \\ \mu^*(H, h) & \text{otherwise,} \end{cases}$$

where $\mathbf{dom}(\bar{\mu}) \subseteq \mathcal{H}/\chi$ is the subset of the abstract history set on which $\bar{\mu}$ is defined. We will denote the completed abstraction as $\alpha = \langle \chi, \bar{\mu}^+ \rangle$. With this abstraction, the abstract value function $\mathcal{V}^*_\alpha$ is well-defined. Notice, however, that every time the abstract reward and transition functions appear in the proof, we can use $\bar{\mu}$ rather than $\bar{\mu}^+$ because the two weight functions are equivalent for abstract histories that are in the search tree (and thus in $\mathbf{dom}(\bar{\mu})$). The completed weight function $\bar{\mu}^+$ never appears in a context where it actually needs to be computed.

The error due to estimating $\mathcal{Q}^*_\alpha$ with $\hat{\mathcal{Q}}$ in AbstractSS is given by

$$\begin{aligned} E(H, a) &= \Big|\hat{\mathcal{Q}}(H, a) - \mathcal{Q}^*_\alpha(H, a)\Big| \\ &= \Big|\Big[\frac{1}{C}\sum_{h \in H} n(h)R(h) + \gamma\frac{1}{C}\sum_{H' \in K(H,a)} N(H')\hat{\mathcal{V}}(H')\Big] \\ &\qquad - \Big[\mathcal{R}_{\bar{\mu}}(H) + \gamma\sum_{H' \in \mathcal{H}/\chi} \mathcal{P}_{\bar{\mu}}(H'|H, a)\mathcal{V}^*_\alpha(H')\Big]\Big| \\ &= \gamma\Big|\frac{1}{C}\sum_{H' \in K(H,a)} N(H')\hat{\mathcal{V}}(H') - \sum_{H' \in \mathcal{H}/\chi} \mathcal{P}_{\bar{\mu}}(H'|H, a)\mathcal{V}^*_\alpha(H')\Big| \end{aligned}$$

Following the proof of Kearns et al. (2002), we introduce the quantity

$$\mathcal{U}^*(H,a) = \mathcal{R}_{\bar{\mu}}(H) + \gamma \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \mathcal{V}^*_\alpha(H').$$

The difference $|\mathcal{Q}^*_\alpha(H,a) - \mathcal{U}^*(H,a)|$ captures the error introduced in one step due to finite sampling. Expanding this difference and canceling the immediate reward terms gives

$$\Big|\mathcal{Q}^*_\alpha(H,a) - \mathcal{U}^*(H,a)\Big| = \gamma \Big| \mathbb{E}_{H' \sim \mathcal{P}_{\bar{\mu}}(\cdot|H,a)} \mathcal{V}^*_\alpha(H') - \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \mathcal{V}^*_\alpha(H') \Big|,$$

which is the absolute difference between the expected value of successors sampled from $\mathcal{P}_{\bar{\mu}}(\cdot|H,a)$ and the mean over C iid successor samples from $\mathcal{P}_{\bar{\mu}}(\cdot|H,a)$. We can thus apply Hoeffding's inequality (Hoeffding, 1963, Thm. 2) to conclude that

$$\Pr\Big(\Big|\mathcal{Q}^*_\alpha(H,a) - \mathcal{U}^*(H,a)\Big| \leq \lambda \leq \frac{\lambda}{\gamma} \Big) \geq 1 - 2e^{-2\lambda^2 C / (V^d_{\max})^2}. \tag{16}$$

Using this result, we decompose the sampling error in terms of $\mathcal{U}^*$ as

$$\begin{aligned}
E(H,a) &\leq \gamma \Big| \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \hat{\mathcal{V}}(H') - \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \mathcal{V}^*_\alpha(H') \Big| \\
&\quad + \gamma \Big| \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \mathcal{V}^*_\alpha(H') - \mathbb{E}_{H' \sim \mathcal{P}_{\bar{\mu}}(\cdot|H,a)} \mathcal{V}^*_\alpha(H') \Big| \\
&\leq \lambda + \gamma \Big| \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \hat{\mathcal{V}}(H') - \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \mathcal{V}^*_\alpha(H') \Big|.
\end{aligned}$$

We will now bound the overall error $E(H,a)$ by the recursive quantity

$$\eta_{t+1} = \gamma(\lambda + \eta_t)$$

with $\eta_0 = V^d_{\max}$. We have

$$\eta_d = \beta_\gamma(d)\lambda + \gamma^d V^d_{\max}.$$

The quantity $\gamma^d V^d_{\max}$ is the error due to truncating the search tree to a depth of d (remembering that the d in $V^d_{\max}$ is a depth index and not an exponent).

We now argue by induction that $E(H_0,a) \leq \eta_d$.

Base case Consider an arbitrary leaf node $H \in \Omega$. We have $E(H,a) \leq V^d_{\max} = \eta_0$.

Inductive step Now consider an arbitrary interior node $H \in \overline{\Omega}$ and action $a \in \mathcal{A}$ and assume the inductive hypothesis $E(H',a') \leq \eta_t$ for all $H' \in K(H,a)$ and $a' \in \mathcal{A}$. We have

$$\begin{aligned}
E(H,a) &= \gamma \Big| \mathbb{E}_{H' \sim \mathcal{P}_{\bar{\mu}}(\cdot|H,a)} [\mathcal{V}^*_\alpha(H')] - \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \hat{\mathcal{V}}(H') \Big| \\
&\leq \gamma \Big(\Big| \mathbb{E}_{H' \sim \mathcal{P}_{\bar{\mu}}(\cdot|H,a)} [\mathcal{V}^*_\alpha(H')] - \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \mathcal{V}^*_\alpha(H') \Big| \\
&\quad + \Big| \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \mathcal{V}^*_\alpha(H') - \frac{1}{C} \sum_{H' \in K(H,a)} N(H') \hat{\mathcal{V}}(H') \Big| \Big) \\
&\leq \gamma(\lambda + \eta_t) = \eta_{t+1}.
\end{aligned}$$

This completes the inductive argument, and we conclude that $E(H_0, a) \leq \eta_d = \beta_\gamma(d)\lambda + \gamma^d V_{\max}^d$.

To obtain a probability bound in the root node, we require that the bound in (16) holds in all action nodes simultaneously. Applying the union bound as in Lemma 4 of Kearns et al. (2002) we conclude that with probability at least $1 - (|\mathcal{A}|C)^d \cdot 2e^{-2\lambda^2 C/(V_{\max}^d)^2}$, we have

$$\left|\hat{\mathcal{Q}}(H_0, a) - \mathcal{Q}_\alpha^*(H_0, a)\right| \leq \beta_\gamma(d)\lambda + \gamma^d V_{\max}^d \tag{17}$$

in the root state H_0 for all $a \in \mathcal{A}$. This bounds the error due to finite sampling.

To complete the proof, we combine (17) with Theorem 1. We have

$$\begin{aligned}\left|\hat{\mathcal{Q}}(H_0, a) - Q^*(h_0, a)\right| &\leq \left|\hat{\mathcal{Q}}(H_0, a) - \mathcal{Q}_\alpha^*(H_0, a)\right| + \left|\mathcal{Q}_\alpha^*(H_0, a) - Q^*(h_0, a)\right| \\ &\leq \beta_\gamma(d)\lambda + \gamma^d V_{\max}^d + \beta_\gamma(d)(p + \delta q) \\ &= \beta_\gamma(d)(\lambda + p + \delta q) + \gamma^d V_{\max}^d.\end{aligned}$$

By the same reasoning as in Corrolary 1, the above value estimation error bound implies the regret bound

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\Big[\beta_\gamma(d)(\lambda + p + \delta q) + \gamma^d V_{\max}^d\Big],$$

where $a^* = \arg\max_{a \in \mathcal{A}} \hat{\mathcal{Q}}(H_0, a)$.

□

A.3 Proof of Proposition 4

Proposition 4. *Consider a history MDP $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ and an abstraction $\alpha = \langle \chi, \mu^* \rangle$ of T composed of equivalence relation χ and the optimal weight function μ^*. Let π be a stochastic abstract policy $\pi : \mathcal{H}/\chi \times (\mathcal{A} \cup \{\omega\}) \mapsto [0, 1]$ whose action space is augmented with the "stop" action ω. Let H be a random variable such that $H \sim \mathcal{P}_{\mu^*}^\pi(\cdot|\{s_0\})$ and let h be a random variable such that $h \sim P^{\downarrow\pi}(\cdot|s_0)$. Then the random variable $[h]_\chi$ is equal in distribution to H.*

Proof. Recall that the probability of sampling an abstract history $H = S_0 a_0 S_1 \dots a_{d-1} S_d$ under abstraction α and abstract sampling policy π is

$$\mathcal{P}_{\mu^*}^\pi(H|S_0) = \mathbb{1}_{H_0 = S_0} \pi(H, \omega) \prod_{t=0}^{d-1} \pi(H_t, a_t) \mathcal{P}_{\mu^*}(H_{t+1}|H_t, a_t),$$

and the probability of sampling a ground history $h = s_0 a_0 s_1 \dots a_{d-1} s_d$ under the grounded sampling policy $\downarrow\pi$ is

$$P^{\downarrow\pi}(h|s_0) = \mathbb{1}_{h_0 = s_0} \downarrow\pi(h, \omega) \prod_{t=0}^{d-1} \downarrow\pi(h_t, a_t) P(h_{t+1}|h_t, a_t).$$

We need to show that for $h \sim P^{\downarrow\pi}(\cdot|s_0)$ and $H \sim \mathcal{P}_{\mu^*}^\pi(\cdot|\{s_0\})$, $[h]_\chi =^d H$. This is equivalent to the condition that

$$\sum_{h \in H} P^{\downarrow\pi}(h|s_0) = \mathcal{P}_{\mu^*}^\pi(H|\{s_0\}). \tag{18}$$

Because all ground histories $h \in H$ share the same action sequence, and because $\downarrow\pi$ is the grounded version of π, we have that for all $h, g \in H$,

$$\prod_{t=0}^{\ell(h)-1} \downarrow\pi(h_t, a_t) = \prod_{t=0}^{\ell(g)-1} \downarrow\pi(g_t, a_t) = \prod_{t=0}^{\ell(H)-1} \pi(H_t, a_t),$$

and similarly that $\downarrow\pi(h, \omega) = \downarrow\pi(g, \omega) = \pi(H, \omega)$. We can use these facts to further simplify (18) to the equivalent relationship

$$\sum_{h \in H} P(h|s_0) = \mathcal{P}_{\mu^*}(H|\{s_0\}),$$

where

$$P(h|s_0) = \mathbb{1}_{h_0 = s_0} \prod_{t=0}^{\ell(h)-1} P(h_{t+1}|h_t, a(h_t)),$$
$$\mathcal{P}_{\mu^*}(H|S_0) = \mathbb{1}_{H_0 = S_0} \prod_{t=0}^{\ell(H)-1} \mathcal{P}_{\mu^*}(H_{t+1}|H_t, a(H_t)).$$

We prove this relationship by induction on the length of the history.

Base case Since $H_0 = \{s_0\}$ and $h_0 = s_0$, we have $P(h_0|s_0) = 1 = \mathcal{P}_{\mu^*}(H_0|\{s_0\})$.

Inductive step Assume the inductive hypothesis $\sum_{h_{k-1} \in H_{k-1}} P(h_{k-1}|s_0) = \mathcal{P}_{\mu^*}(H_{k-1}|\{s_0\})$. We have

$$\begin{aligned}
\sum_{h_k \in H_k} P(h_k|s_0) &= \sum_{h_{k-1} \in H_{k-1}} P(h_{k-1}) \sum_{h_k \in H_k} P(h_k|h_{k-1}, a_{k-1}) \\
&= \frac{\sum_{g \in H_{k-1}} P(g|s_0)}{\sum_{g \in H_{k-1}} P(g|s_0)} \sum_{h_{k-1} \in H_{k-1}} P(h_{k-1}|s_0) \sum_{h_k \in H_k} P(h_k|h_{k-1}, a_{k-1}) \\
&= \mathcal{P}_{\mu^*}(H_{k-1}|\{s_0\}) \sum_{h_{k-1} \in H_{k-1}} \frac{P(h_{k-1}|s_0)}{\sum_{g \in H_{k-1}} P(g|s_0)} \sum_{h_k \in H_k} P(h_k|h_{k-1}, a_{k-1}) \qquad (*) \\
&= \mathcal{P}_{\mu^*}(H_{k-1}|\{s_0\}) \sum_{h_{k-1} \in H_{k-1}} \mu^*(H_{k-1}, h_{k-1}) \sum_{h_k \in H_k} P(h_k|h_{k-1}, a_k) \qquad (**) \\
&= \mathcal{P}_{\mu^*}(H_{k-1}|\{s_0\}) \mathcal{P}_{\mu^*}(H_k|H_{k-1}, a_k) \\
&= \mathcal{P}_{\mu^*}(H_k|\{s_0\}),
\end{aligned}$$

where in (*) we used the inductive hypothesis and in (**) we used the definition of μ^* (Definition 3).

This completes the inductive argument, and we conclude that for $H \sim \mathcal{P}^{\pi}_{\mu^*}(\cdot|\{s_0\})$ and $h \sim P^{\downarrow\pi}(\cdot|s_0)$, we have $[h]_\chi =^d H$.

□

Appendix B. Refining the Weight Function

The ordering of abstractions defined in Section 5.1 addresses only the abstraction relation portion of the state abstraction. A similarly natural relationship can be defined for the corresponding weight functions.

Definition 12. Consider two abstractions $\alpha = \langle \chi, \mu \rangle$ and $\beta = \langle \psi, \nu \rangle$ where $\psi \preceq \chi$. The *refinement of μ with respect to ψ*, denoted $\nu = \mu/\psi$, is the weight function $\nu : \mathcal{H}/\psi \times \mathcal{H} \mapsto [0, 1]$ such that for each $H \in \mathcal{H}/\chi$, for each $G \in \mathcal{H}/\psi$, for all $g \in \mathcal{H}$,

$$\nu(G, g) = \mathbb{1}_{g \in G} \frac{\mu(H, g)}{\sum_{h \in G} \mu(H, h)}.$$

Essentially, given abstraction $\alpha = \langle \chi, \mu \rangle$ and an abstraction relation $\psi \preceq \chi$, one constructs μ/ψ by re-normalizing the probability mass given by μ to the new, smaller equivalence classes with respect to ψ. Note that if μ is the set of empirical probability mass functions $\bar{\mu}$ with respect to χ, then μ/ψ is just the set of empirical probability mass functions with respect to ψ. The PARSS algorithm (Section 5.2) uses this weight refinement strategy implicitly.

The refinement of an optimal weight function is the optimal weight function in the refined state space.

Proposition 10. *Consider two abstractions $\alpha = \langle \chi, \mu \rangle$ and $\beta = \langle \psi, \nu \rangle$ where $\psi \preceq \chi$ and μ is the optimal weight function for T/α (ie. $\mu = W\mu$). Then $\nu = \mu/\psi$ is the optimal weight function for T/β (ie. $\nu = W\nu$).*

Proof. Let $H \in \mathcal{H}/\chi$ and let $G \in \mathcal{H}/\psi$ be such that $G \subseteq H$. Begin by noting that

$$\sum_{g \in G} \mu^*(H, g) \stackrel{\text{def}}{=} \sum_{g \in G} P(g|H) = P(G|H).$$

The definition of optimal weight functions in terms of probabilities allows us to simplify,

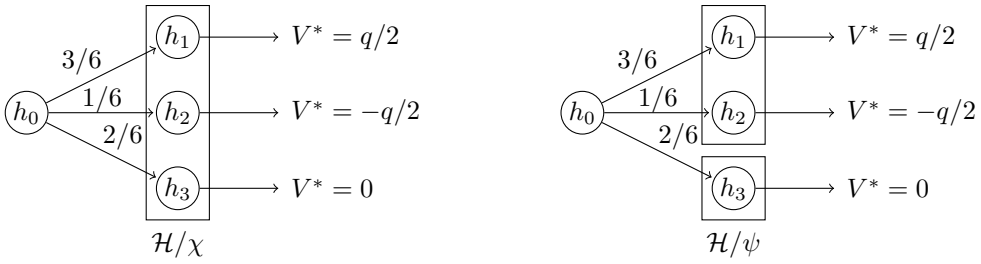
$$\nu(G, g) \stackrel{\text{def}}{=} \mathbb{1}_{g \in G} \frac{\mu^*(H, g)}{\sum_{h \in G} \mu^*(H, h)} = \frac{\mathbb{1}_{g \in G} P(g|H)}{P(G|H)} = \frac{\mathbb{1}_{g \in G} P(H, g)}{P(H)P(G|H)} = \frac{\mathbb{1}_{g \in G} \mathbb{1}_{g \in H} P(g)}{P(G)P(H|G)}$$

Now, since $G \subseteq H$, we have $P(H|G) = 1$ and $\mathbb{1}_{g \in G} \mathbb{1}_{g \in H} = \mathbb{1}_{g \in G}$. We conclude

$$\frac{\mathbb{1}_{g \in G} \mathbb{1}_{g \in H} P(g)}{P(G)P(H|G)} = \mathbb{1}_{g \in G} \frac{P(g)}{P(G)} \stackrel{\text{def}}{=} \nu^*(G, g).$$

□

For a non-optimal weight function $\mu \neq \mu^*$, $\nu = \mu/\psi$ may be such that $\delta_{T/\langle \psi, \nu \rangle} > \delta_{T/\langle \chi, \mu \rangle}$. Consider the following MDP, for which the edge labels denote transition probabilities:



The abstract state $H = \{h_1, h_2, h_3\}$ is $(0, q)$-consistent. If $\mu^*(H) = [3/6, 1/6, 2/6]$ but $\mu(H) = [1/3, 1/3, 1/3]$, then $\delta_{T/\langle\chi,\mu\rangle}(H) = \frac{1}{2}\|\mu(H) - \mu^*(H)\|_1 = 1/6$. We have an abstraction error of $q \cdot 1/6$ in H. Now consider $\mathcal{H}/\psi = \{\{h_1, h_2\}, \{h_3\}\}$ and let $\nu = \mu/\psi$. We have $\nu(\{h_1, h_2\}) = [1/2, 1/2]$ while $\nu^*(\{h_1, h_2\}) = [3/4, 1/4]$, thus $\delta_{T/\langle\psi,\nu\rangle}(\{h_1, h_2\}) = 1/4 > 1/6$. Since $\{h_1, h_2\}$ is $(0, q)$-consistent like H, the abstraction error in $\{h_1, h_2\}$ is $q \cdot 1/4 > q \cdot 1/6$.

From this example we see that abstraction refinement can increase error due to δq if the optimal weight function μ^* is not available, at least for the most obvious method of refining the weight function. Nevertheless, repeated abstraction refinement will eventually produce the ground abstraction $\langle\bot, \mu_\bot\rangle$, which is sound. Our algorithms rely on this eventual convergence to $\bot$ in order to provide performance guarantees.

References

Anand, A., Grover, A., Mausam, & Singla, P. (2015). ASAP-UCT: Abstraction of state-action pairs in UCT. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Anand, A., Noothigattu, R., Mausam, & Singla, P. (2016). OGA-UCT: On-the-go abstractions in UCT. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Bai, A., Srivastava, S., & Russell, S. (2015). Markovian state and action abstractions for MDPs via hierarchical MCTS. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Balla, R.-K., & Fern, A. (2009). UCT for tactical assault planning in real-time strategy games. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1-2), 81–138.

Baum, J., Nicholson, A. E., & Dix, T. I. (2012). Proximity-based non-uniform abstractions for approximate planning. *Journal of Artificial Intelligence Research*, *43*, 477–522.

Bertsekas, D. P., & Castañon, D. A. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, *5*(1), 89–108.

Bertsekas, D. P., & Ioffe, S. (1996). Temporal differences-based policy iteration and applications in neuro-dynamic programming. Tech. rep., Massachusetts Institute of Technology.

Binns, J. R., Bethwaite, F. W., & Saunders, N. R. (2002). Development of a more realistic sailing simulator. In *High Performance Yacht Design Conference*.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Bubeck, S., & Munos, R. (2010). Open loop optimistic planning. In *Conference on Learning Theory (COLT)*.

Calvo, B., & Santafe, G. (2015). scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, *Accepted for publication*.

Chang, H. S., Givan, R., & Chong, E. K. (2004). Parallel rollout for online solution of partially observable Markov decision processes. *Discrete Event Dynamic Systems*, *14*(3), 309–341.

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(3).

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research (JMLR)*, *7*, 1–30.

Edelkamp, S. (2001). Planning with pattern databases. In *European Conference on Planning (ECP)*.

Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *AAAI Conference on Artificial Intelligence*.

Ferns, N., Panangaden, P., & Precup, D. (2004). Metrics for finite Markov decision processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Gabillon, V., Ghavamzadeh, M., & Scherrer, B. (2013). Approximate dynamic programming finally performs well in the game of Tetris. In *Advances in Neural Information Processing Systems (NIPS)*.

Gelly, S., & Silver, D. (2007). Combining online and offline knowledge in UCT. In *International Conference on Machine Learning (ICML)*.

Givan, R., Dean, T., & Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, *147*(1), 163–223.

Guerin, J. T., Hanna, J. P., Ferland, L., Mattei, N., & Goldsmith, J. (2012). The academic advising planning domain. In *Workshop on the International Planning Competition (WS-IPC) at ICAPS*.

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems*.

Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Hauser, K. (2011). Randomized belief-space replanning in partially-observable continuous spaces. In *Algorithmic Foundations of Robotics IX*, pp. 193–209. Springer.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning.. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58*(301), 13–30.

Hostetler, J., Fern, A., & Dietterich, T. (2015). Progressive abstraction refinement for sparse sampling. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Hostetler, J., Fern, A., & Dietterich, T. (2014). State aggregation in Monte Carlo tree search. In *AAAI Conference on Artificial Intelligence*.

Hostetler, J. A. (2017). *Monte Carlo Tree Search with Fixed and Adaptive Abstractions*. Ph.D. thesis, Oregon State University.

Hutter, M. (2014). Extreme state aggregation beyond MDPs. In *International Conference on Algorithmic Learning Theory*.

Jiang, N., Singh, S., & Lewis, R. (2014). Improving UCT planning via approximate homomorphisms. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Kearns, M., Mansour, Y., & Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, *49*(2-3), 193–208.

Kearns, M. J., Mansour, Y., & Ng, A. Y. (1999). Approximate planning in large POMDPs via reusable trajectories. In *Advances in Neural Information Processing Systems (NIPS)*.

Keller, T., & Eyerich, P. (2012). PROST: Probabilistic planning based on UCT. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

King, B., Fern, A., & Hostetler, J. (2013). On adversarial policy switching with experiments in real-time strategy games.. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*.

Korf, R. E. (1993). Linear-space best-first search. *Artificial Intelligence*, *62*(1), 41–78.

Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics*.

McCallum, A. K. (1996). *Reinforcement learning with selective perception and hidden state*. Ph.D. thesis, University of Rochester.

McMahan, H. B., Likhachev, M., & Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *International Conference on Machine Learning (ICML)*.

Meuleau, N., Kim, K.-E., Kaelbling, L. P., & Cassandra, A. R. (1999). Solving POMDPs by searching the space of finite policies. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Moore, A. W., & Atkeson, C. G. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, *21*(3), 199–233.

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)*, *20*, 379–404.

Pinto, J., & Fern, A. (2014). Learning partial policies to speedup MDP tree search. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Poupart, P., & Boutilier, C. (2003). Bounded finite state controllers. In *Advances in Neural Information Processing Systems (NIPS)*.

Ravindran, B., & Barto, A. (2004). Approximate homomorphisms: A framework for nonexact minimization in Markov decision processes. In *International Conference on Knowledge-Based Computer Systems*.

Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach*. Prentice Hall.

Sanner, S. (2010). Relational dynamic influence diagram language (RDDL): Language description. Tech. rep., Australian National University.

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), 484–489.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1), 181–211.

Van den Broeck, G., & Driessens, K. (2011). Automatic discretization of actions and states in Monte-Carlo tree search. In *ECML/PKDD Workshop on Machine Learning and Data Mining in and around Games*.

Van Roy, B. (2006). Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, *31*(2), 234–244.

Walsh, T. J., Goschin, S., & Littman, M. L. (2010). Integrating sample-based planning and model-based reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Weinstein, A., & Littman, M. L. (2012). Bandit-based planning and learning in continuous-action Markov decision processes.. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Weinstein, A., & Littman, M. L. (2013). Open-loop planning in large-scale stochastic domains. In *AAAI Conference on Artificial Intelligence*.