AN ABSTRACT OF THE DISSERTATION OF

Jesse A. Hostetler for the degree of Doctor of Philosophy in Computer Science presented on March 20, 2017.

Title: Monte Carlo Tree Search with Fixed and Adaptive Abstractions

Abstract approved: _____

Thomas G. Dietterich Alan P. Fern

Monte Carlo tree search (MCTS) is a class of online planning algorithms for Markov decision processes (MDPs) and related models that has found success in challenging applications. In the online planning approach, the agent makes a decision in the current state by performing a limited forward search over possible futures and selecting the course of action that is expected to lead to the best outcomes. This thesis proposes a new approach to MCTS based on abstraction and progressive abstraction refinement that makes better use of a limited number of samples. Our first contribution is an analysis of state abstraction in the MCTS setting. We describe a class of state aggregation abstractions that generalizes previously-proposed abstraction criteria and show that the regret due to planning with such abstractions is bounded. We then adapt popular MCTS algorithms to use fixed state abstractions. Our second contribution is a novel approach to MCTS based on abstraction refinement. We propose the Progressive Abstraction Refinement for Sparse Sampling (PARSS) algorithm, which begins by performing sparse sampling with a coarse state abstraction and then refines the abstraction progressively to make it more accurate. The PARSS algorithm provides the same formal guarantees as ordinary sparse sampling, and we show experimentally that PARSS outperforms sparse sampling in the ground state space and with fixed uninformed abstractions. Our third contribution is an extension of the progressive refinement idea to incorporate other kinds of abstraction. For this purpose, we introduce the formalism of abstraction diagrams (ADs) and show that ADs can express diverse kinds of abstraction, including state abstraction, temporal abstraction, and action pruning. We then describe refinement

operators for ADs, extending the progressive refinement search framework to abstractions represented as ADs. Our fourth and final contribution is an application of online planning algorithms to the problem of controlling electrical transmission grids to mitigate the effects of equipment failures. Our work in this area is distinguished by the use of a full dynamical model of the power grid, which captures more mechanisms of cascading failure than simpler models. Because of the computational cost of the simulation, we choose simple online planning algorithms that require a small number of simulation trajectories. Our results demonstrate the superiority of the online planning approach to fixed expert policies, while also highlighting the need for faster simulators to enable more sophisticated solution algorithms.

Monte Carlo Tree Search with Fixed and Adaptive Abstractions

by

Jesse A. Hostetler

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented March 20, 2017
Commencement June 2017

Doctor of Philosophy dissertation of Jesse A. Hostetler presented on March 20, 2017.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Jesse A. Hostetler, Author

ACKNOWLEDGEMENTS

I have been fortunate to have had many mentors in my academic life. I could not have hoped for better PhD advisors than Thomas Dietterich and Alan Fern. In my time as a student both of you have always been generous with your knowledge and accomodating of my interests and eccentricities. I am grateful to Ashok Samal, Leen-Kiat Soh, and Berthe Choueiry for picquing my interest in computer science research as an undergraduate and preparing me for my graduate education. I also owe a great deal to Anne Cognard, from whom I learned intellectual rigor and how to write a research paper.

I am pleased to acknowledge the financial support of Caron and Larry Ogg for a portion of my graduate education. Thank you for many enjoyable conversations.

I thank my parents, Mary Sawicki and Karl Hostetler, for their constant support and for cultivating my love of science. Finally, I thank the members of Meatbomb – Aswin Raghavan, Gayathry Lakshminarasimhan, and Dale Lawson – for keeping me sane and giving me something to look forward to on the weekends.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1: Introduction

Sequential decision making under uncertainty is the problem of deciding what to do in a world that changes stochastically over time. Examples include choosing the fastest route for a taxi to take or operating a power plant to maximize generation and minimize fuel consumption. The goal is to obtain a controller or *policy* that prescribes the actions to take to optimize some performance criterion. A policy is a function π that maps a state of the world to an action. A taxi driving policy, for example, might take the destination and the next street intersection as input and produce "left", "right", or "straight" as output. An optimal taxi driving policy would reach the destination in the shortest time possible.

Policies fall roughly into two categories — *reactive* policies and *deliberative* policies — differentiated by whether evaluating the policy involves reasoning about the future. A reactive policy is a more-or-less direct mapping from states to actions, such as a hash table with states for keys and actions for values. A deliberative policy, by contrast, explicitly reasons about the likely outcomes of different actions and selects the action that produces the best outcomes. One can think of a deliberative policy as a *lazy* policy, since it delays computing the result of $\pi(s)$ until it is called for.

The two types of policies excel in different applications. Reactive policies are expensive to compute initially, because it is not known at which states the policy will be evaluated and thus a *complete* policy that is defined in every state is required. Once computed, however, reactive policies make decisions quickly, which makes them suitable for real-time decision-making. Deliberative policies, on the other hand, do not waste computation on states that are never actually encountered. There is no substantial up-front computation and no large policy to represent and store. The price of laziness is that each decision actually taken requires a significant amount of computation. Deliberative policies are needed when the problem is big enough that computing and storing the action choice for all of the states is impractical.

Deliberative policies are realized via the process of online planning (OP). Given the current state s, an OP algorithm evaluates $\pi(s)$ by constructing a partial policy $\tilde{\pi}$ that

is expected to be effective for the immediate future following s and returning the action choice $\tilde{\pi}(s)$. For efficiency, the planning problem is typically simplified in some way, such as by approximating the dynamics with a small number of samples or restricting the action choices available to $\tilde{\pi}$. Although this means that each partial policy $\tilde{\pi}$ is unlikely to be optimal, the process of continual re-planning allows the deliberative policy $\pi(s)$ to perform better than each partial policy.

Within the online planning family, Monte Carlo tree search (MCTS) algorithms [Browne et al., 2012] and particularly the UCT algorithm [Kocsis and Szepesvári, 2006] have risen to prominence, largely due to the success of UCT variants in the game of *go* [Gelly and Silver, 2007; Silver et al., 2016] and in other complex domains (e.g. [Balla and Fern, 2009; Guo et al., 2014]). MCTS algorithms estimate the values of the available actions in the current state s by sampling a tree of possible future trajectories rooted at s. The size of the 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. The key property of MCTS algorithms like UCT and sparse sampling (SS) [Kearns et al., 2002] is that they achieve bounded value estimation error in the root state with a number of samples that does not depend on the state space size. This property makes MCTS algorithms attractive choices for problems like *go* that have large state spaces.

The primary disadvantage of MCTS and other OP algorithms is that they interleave planning and execution. 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.

The main focus of this thesis is an *abstraction*-based approach to improving the practical performance of Monte Carlo tree search algorithms. An abstraction is a transformation of the problem that simplifies it in some way. For example, we could create a state abstraction by specifying a subset of the state variables that are "irrelevant" and

ignoring their values when making decisions. In effect, we treat states that differ only on the irrelevant state variables as equivalent. This creates an abstract state space that is smaller than the original. In the terminology of online planning, abstraction reduces the sets of candidate policies from which the partial policies $\tilde{\pi}$ are drawn. Our main contributions in this area are

1. A definition of a class of state abstractions that generalizes existing criteria for sound (lossless) state abstraction to allow unsound abstractions (which allow greater reductions in the state space),
2. A formal bound on the regret due to acting according to an abstract tree search using abstractions from this class,
3. Versions of popular MCTS algorithms that exploit fixed state abstractions,
4. A novel MCTS algorithm based on progressive abstraction refinement, which addresses the problem of specifying the correct abstraction for tree serach,
5. A framework for generalizing the progressive refinement approach to incorporate many additional abstraction modalities beyond state abstraction, such as temporal abstraction and action pruning.

In addition to this primary focus, we also present an application of online planning to the problem of controlling a power transmission grid to mitigate the detrimental effects of component failures. This problem is just the type of problem that abstract tree search is intended to solve: it has very large state and action spaces, a high degree of stochasticity, and potentially long delays between actions and their consequences. We employ a state-of-the-art power grid dynamics simulator designed specifically to capture the dynamics of cascading failure to evaluate online planning algorithms in comparison to fixed expert policies. Although the high computational cost of the simulations prevented us from applying our new MCTS algorithms to the problem, we show experimentally that simpler online planning algorithms are superior to expert policies.

1.1 Outline

We begin by introducing common notation for MDPs and MCTS algorithms in Chapter 2. We view MCTS 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. Chapter 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. Chapter 3 also describes how the two main categories of MCTS algorithms can be modified to exploit fixed state abstractions. We present and analyze the PARSS algorithm in Chapter 4, and describe several variations of PARSS based on different abstraction refinement strategies. Chapter 4 also presents experiments with abstract MCTS algorithms and their results. We generalize the PARSS algorithm framework to include other types of abstraction in Chapter 5 by introducing the *abstraction diagram* formalism. We then digress somewhat from the main topic in Chapter 6 to present an application of online planning techniques to the problem of controlling power transmission grids to mitigate blackouts following equipment failures. We conclude with a summary and directions for future work in Chapter 7.

Chapter 2: Background

Our work focuses on Monte Carlo tree search (MCTS) algorithms for anytime online planning in Markov decision processes (MDPs). We incorporate MDP state abstraction and progressive abstraction refinement into MCTS algorithms to improve their anytime performance. This section introduces notation and key concepts.

2.1 Markov Decision Processes

Markov decision processes are the standard model of decision making under uncertainty. 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, 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.

2.2 Solving MDPs

There are many approaches to solving MDPs. Key points of distinction between approaches include whether they compute a reactive or a deliberative policy and whether they require access to the transition probability function $P(\cdot|s,a)$ (*model-based* approaches) or merely the ability to sample from it (*model-free* approaches). This section briefly reviews some major categories of solution algorithms.

2.2.1 Dynamic Programming

Dynamic programming is a model-based approach that solves the problem globally. Value iteration [Sutton and Barto, 1998] is a prototypical DP algorithm. The value iteration algorithm consists of repeatedly applying the *Bellman backup* operator T to an

estimate of the value function to obtain an improved estimate

$$(TV)(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)V(s'). \tag{2.1}$$

The optimal value function is the unique fixed point of T, and thus $\lim_{n\to\infty} T^n V = V^*$.

The value iteration method has two major drawbacks. First, it requires access to $P(\cdot|s,a)$. In many problems, the transition function is unknown. Second, it solves the problem *in every state*, even though many of those states may not be reached from typical starting states under reasonable policies. These observations motivate the need for other approaches.

2.2.2 Reinforcement Learning

Reinforcement learning [Sutton and Barto, 1998] encompasses a diverse family of algorithms with the common theme that optimization occurs only along the states and actions in one or more particular trajectories. Reinforcement learning methods can often solve larger problems than DP because their computation is focused. Their most important drawback is the need to balance *exploration*, in which under-explored actions are tried in order to determine which action is best, with *exploitation*, in which the best actions are taken to obtain the best reward.

A prototypical RL algorithm is *Q-learning*. Q-learning learns the state-action value function with the iteration

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \eta\big[r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1},a) - Q(s_t,a_t)\big], \tag{2.2}$$

where η is the learning rate. Actions are chosen with an *exploration policy* π, which uses Q to choose its actions. To ensure convergence of (2.2), π must explore adequately in the long run. A simple way to achieve this is with an *ϵ-greedy* policy, which tries a random action instead of the estimated optimal one with probability ϵ.

Although RL methods may avoid computing the entire value function, they still calculate a fixed policy that must be represented. Even this may be problematic if the state and action spaces are very large. The next category of algorithms avoids this issue by computing the policy "lazily."

2.2.3 Online Planning

Online planning (OP) 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 and Castañon, 1999]. Policy rollout improves a fixed policy π by implementing a new policy

$$\pi_{\text{pr}}(s) = \arg\max_{a \in \mathcal{A}} \hat{Q}^{\pi}(s, a) \tag{2.3}$$

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 of Chang et al. [2004] replaces the single rollout policy with a set of rollout policies. Policy switching [Chang et al., 2004; King et al., 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 Monte Carlo Tree Search

Our focus in this thesis is on Monte Carlo tree search (MCTS) [Browne et al., 2012], which is an anytime online planning approach. MCTS has risen to prominence due largely to

the success of variations of the UCT algorithm [Kocsis and Szepesvári, 2006] in *go* [Gelly and Silver, 2007; Silver et al., 2016] and other complex domains (e.g. [Balla and Fern, 2009; Guo et al., 2014]). MCTS algorithms simulate state-action *histories* starting from the current state s_0 and gather statistics of those history samples into a search tree. The search tree is employed 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 MCTS 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 MCTS algorithms. We then introduce the two dominant paradigms for MCTS: 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 $\mathcal{H}^*(M, s_0) = \{s_0\} \times \mathcal{A} \times \mathcal{S} \times \cdots$. Note that $\mathcal{H}^*(M, s_0)$ 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 $h = s_0 a_1 s_1 \ldots a_t s_t$, we write $s(h) \stackrel{\text{def}}{=} s_t$ and $a(h) \stackrel{\text{def}}{=} a_t$ for the final state and action in the history, $\mathbf{pre}(h) \stackrel{\text{def}}{=} s_0 a_1 s_1 \ldots a_{t-1} s_{t-1}$ for the prefix of h, and $\mathbf{len}(h) \stackrel{\text{def}}{=} t$ for the length of h.

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 $\mathcal{H}^*(M, s_0)$ for the ground MDP M with the restriction that $h \in \mathcal{H} \Rightarrow \mathbf{pre}(h) \in \mathcal{H}$. The dynamics of T are given by overloading the P and R functions to apply to histories,

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

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 and 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 finite history MDPs in this thesis.

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 sampled successor state, and this is continued until the tree has uniform depth d. 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 et al., 2010] realizes this idea. FSSS constructs the SS tree incrementally by expanding nodes along one state-action trajectory at a time. The trajectories are 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 et al., 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.

2.3.3 Trajectory Sampling

Trajectory sampling (TS) algorithms [Keller and Helmert, 2013] 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. 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 3.5).

The most well-known TS algorithm is UCT [Kocsis and Szepesvári, 2006]. Keller and Helmert [2013] formalized the generic trajectory sampling framework that we have described here and of which UCT is a member. This basic TS algorithm structure as pioneered by UCT is so ubiquitous in the literature that some authors (e.g., Browne et al. [2012]) define Monte Carlo tree search to include only TS algorithms and not SS algorithms. We define MCTS more broadly to include any algorithm that builds a forward search tree through random sampling.

Chapter 3: State Aggregation Abstractions for Tree Search

3.1 Introduction

Our major focus in this thesis is on improving the anytime performance of MCTS algorithms through the use of abstraction. This chapter focuses on state abstraction. Broadly speaking, state abstraction includes any way of reducing the amount of information needed to describe the states of an MDP. We adopt the simplest form of MDP state abstraction, which is state aggregation [Li et al., 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 $\mathcal{H}$.

3.2 History Aggregation

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,

$$\begin{aligned} \mathcal{P}_\mu(H'|H,a) &= \sum_{h\in H} \mu(H,h) \sum_{h'\in H'} P(h'|h,a), \\ \mathcal{R}_\mu(H) &= \sum_{h\in H} \mu(H,h) R(h). \end{aligned} \tag{3.1}$$

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[1] $h \simeq_\chi g \Rightarrow [\mathbf{pre}(h) \simeq_\chi \mathbf{pre}(g) \wedge a(h) = a(g)]$ 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

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

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

$$\downarrow\pi(h) = \pi([h]_\chi). \tag{3.2}$$

Using the induced policy, we can define the *ground* value function of an abstract policy π as $V^{\downarrow\pi}$. We say that an abstraction α is *sound* if every optimal policy π^* for the

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

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. It may be the case that there are states h where $V^*(h) \neq \mathcal{V}^\pi_\alpha([h]_\chi)$ and yet $V^{\downarrow\pi} = V^*$.

3.3 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 $H \in \mathcal{H}/\chi$,

$$\exists a^* \in \mathcal{A} \;.\; \forall h \in H : \quad V^*(h) - Q^*(h, a^*) \leq p \tag{3.3}$$

$$\forall h, g \in H : \quad \left|V^*(h) - V^*(g)\right| \leq q. \tag{3.4}$$

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

The p condition (3.3) quantifies the "action homogeneity" of the partition. It 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 (3.4) quantifies the "value homogeneity." It 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 value estimation due to the different dynamics of the abstract and ground processes.

The (p, q)-consistency property is a generalization of the a^*-irrelevance and π^*-irrelevance properties identified by Li et al. [2006] in their taxonomy of sound state aggregation abstractions. The π^*-irrelevance property is equivalent to $(0, \infty)$-consistency, while a^*-irrelevance 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 abstractions satisfying π^*-irrelevance are sound, learning with these abstractions can be problematic because it may be that the optimal ground value of a state s

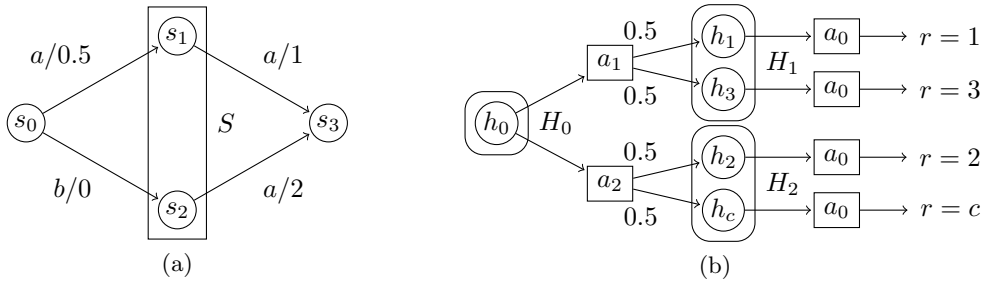


Figure 3.1: (a) An example of an MDP for which a $(0, \infty)$-consistent abstraction is unsound [Li et al., 2006]. The edge labels like "$a/0.5$" mean action a yields immediate reward 0.5. (b) A history MDP for which a $(0, \infty)$-consistent abstraction $\langle \chi, \mu \rangle$ is unsound for some weighting functions $\mu \neq \mu^*$. Edge labels denote transition probabilities.

is not equal to the optimal abstract value of its equivalence class, that is $V^*(s) \neq \mathcal{V}^*([s]_\chi)$. A simple example due to Li et al. [2006] illustrates the problem (Figure 3.1a). 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 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 the abstract value function is not optimal in the ground problem because it chooses a in s_0 due to its larger immediate reward.

History aggregation abstractions (Definition 1) cannot create the structure in Figure 3.1a 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 3.1b. If the weight function μ is

$$\begin{aligned} \mu(H_1, h_1) &= 0 & \mu(H_2, h_2) &= 1 \\ \mu(H_1, h_3) &= 1 & \mu(H_2, h_c) &= 0, \end{aligned}$$

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. The following definition formalizes this property.

Definition 3. The *optimal* weight function for history MDP T, denoted μ_T^* (or μ^* if T is clear from context), weights ground states by the conditional probability $\mathbb{P}(h|H)$ of occupying ground state h given that the process followed the abstract history H:

$$\begin{aligned}\mu_T^*(H,h) &= \frac{\mathbb{1}_{h\in H}\mathbb{P}(h)}{\mathbb{P}(H)} = \frac{\prod_{i=1}^{\ell(h)} P(h_i|h_{i-1},a(h_i))}{\sum_{g\in H}\prod_{i=1}^{\ell(g)} P(g_i|g_{i-1},a(g_i))} \\ &= \frac{\sum_{g\in\mathbf{pre}(H)} \mu_T^*(\mathbf{pre}(H),g)P(h|g,a(h))}{\mathcal{P}_{\mu_T^*}(H|\mathbf{pre}(H),a(H))}.\end{aligned} \tag{3.5}$$

The recursive form of the last expression in Definition 3 is especially natural in the tree search setting. We can view this recursive form as an operator $[\mu]^*$ acting on a weight function μ to give an *exact update* of μ,

$$[\mu]^*(H,h) = \frac{\sum_{g\in\mathbf{pre}(H)} \mu(\mathbf{pre}(H),g)P(h|g,a(h))}{\mathcal{P}_{\mu}(H|\mathbf{pre}(H),a(H))}. \tag{3.6}$$

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

$$\delta_{\mathcal{T}}(H) = \frac{1}{2}\Big\|\mu(H,\cdot) - [\mu]^*(H,\cdot)\Big\|_1, \tag{3.7}$$

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

$$\delta_{\mathcal{T}} = \max_{H\in\mathcal{H}/\chi} \delta_{\mathcal{T}}(H), \tag{3.8}$$

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

$$J_\alpha(h) = \max_{a\in\mathcal{A}} Q^*(h,a) - Q^*(h,{\downarrow}\pi^*(h)). \tag{3.9}$$

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,

$$\beta_\gamma(d) = \sum_{i=1}^{d} \gamma^d. \tag{3.10}$$

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 $d = \max_{h \in \mathcal{H}} \mathbf{len}(h)$ (which may be infinite). 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. 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*

$$J_\alpha(h) \leq 2\beta_\gamma(d)(p + \delta q). \tag{3.11}$$

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 if $a^* \neq \hat{a}$, 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, $J_{\langle \chi, \mu \rangle}(s_0) = 0$. 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 $\mathcal{Q}^*_\alpha(\{s_0\}, \cdot) = Q^*(s_0, \cdot)$. This is a stronger guarantee than soundness of the abstraction, since it ensures that we can act optimally in the ground problem by acting greedily with respect to an abstract value function. Contrast this result with the counterexample MDP in Figure 3.1a, for which no abstract value function has a greedy policy whose induced ground policy is optimal in the ground problem.

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 4.4.2.2.

3.4 Abstract MDPs as Partially Observable MDPs

Bai et al. [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 $z_0 a_0 z_1 \dots z_k$ to an action.

Consider a history MDP $T = \langle \mathcal{H}, \mathcal{A}, P, R, s_0, \gamma \rangle$ 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 3.6, which when the notation is expanded reads

$$[\mu]^*(H, h) = \frac{\sum_{g \in \mathbf{pre}(H)} \mu(\mathbf{pre}(H), g)P(h|g, a(h))}{\sum_{g \in \mathbf{pre}(H)} \mu(\mathbf{pre}(H), g) \sum_{h \in H} P(h|g, a(h))}.$$

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, $[\mu]^*$ 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 et al., 1999; Poupart and Boutilier, 2003]). One can view abstract MCTS 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 MCTS the policy is used only to evaluate the current state and thus need only be effective locally.

3.5 MCTS Algorithms using Fixed Abstractions

Adapting MCTS 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 func-

tion 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.

3.5.1 Framework and Notation

Algorithm 1 Abstract MCTS Framework

```
1: procedure AbstractMCTS(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
```

We will view abstract MCTS 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 MCTS 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.

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 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 and action $a \in \mathcal{A}$ to a set of successors

$$k_n(h, a) = \{h' \in n : \mathbf{pre}(h') = h, a(h') = a\}. \tag{3.12}$$

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

$$m_n(h, a) = \sum_{h' \in k_n(h,a)} n(h'). \tag{3.13}$$

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

The second product of abstract MCTS — 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

$$N(H) = \sum_{h \in H} n(h) \quad \forall H \in \mathcal{H}/\chi. \tag{3.14}$$

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,

$$K_N(H, a) = \{H' \in N : \mathbf{pre}(H') = H, a(H') = a\}, \tag{3.15}$$

$$M_N(H, a) = \sum_{H' \in K_N(H,a)} N(H'), \tag{3.16}$$

and as before we will usually omit the N subscripts.

We can now outline a generic abstract MCTS 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 TS and SS algorithms.

3.5.2 Representing Abstractions in Tree Search

In the tree search setting, it is natural to represent 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

Algorithm 2 Abstract Sparse Sampling

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

class of a ground history $h = s_0 a_0 s_1 \ldots a_{d-1} s_d$ is the set

$$[h]_\chi = S_0 a_0 S_1 \ldots a_{d-1} S_d, \qquad \text{where } S_i = [s_i]_{\chi(S_{i-1}, a_{i-1})},\ S_0 = \{s_0\}.$$

Any history aggregation abstraction 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 (Chapter 4).

3.5.3 Abstract Sparse Sampling

Sparse sampling [Kearns et al., 2002] is a systematic approach to MCTS. It is systematic 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 recursively for every action node ha in the tree with $\ell(h) < d$. The algorithm achieves small error with high probability with a sample complexity that does not depend on the size of the state space $|\mathcal{S}|$ (Theorem 1 of Kearns et al. [2002]). The AbstractSS algorithm (Algorithm 2) employs the same systematic sampling strategy, but it operates in the abstract state space.

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 sampling from $\mathcal{P}_{\hat{\mu}}$, where $\hat{\mu}$ is our estimate of μ^*. The obvious choice of $\hat{\mu}(H, \cdot)$ is the empirical probability of the ground histories $h \in H$,

$$\bar{\mu}(H, h) = \mathbb{1}_{h \in H} \frac{n(h)}{N(H)}. \tag{3.17}$$

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

$$\bar{\mu}^+(H, h) = \begin{cases} \bar{\mu}(H, h) & \text{if } H \in \mathbf{dom}(\bar{\mu}), \\ \mu^*(H, h) & \text{otherwise.} \end{cases} \tag{3.18}$$

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^*.

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 $\mathbf{len}(h) = d$. Since our rewards are bounded in $[0, 1]$, one possible definition of $V^d_{\max}$ is

$$V^d_{\max} = \begin{cases} \sum_{t=0}^{\infty} \gamma^t & \text{if } \gamma < 1, \\ D - d & \text{if } \gamma = 1, \end{cases} \tag{3.19}$$

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 $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ be a history MDP and let χ be a (p, q)-consistent history equivalence relation on $\mathcal{H}$. 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*

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\beta_\gamma(d)(\lambda + p + \delta q),$$

where δ is the divergence (3.7) 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.

3.5.4 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 subtrees. It provides the same performance guarantees as SS and often performs less computation.

Abstract FSSS (AFSSS; Algorithm 3) is a straightforward extension of FSSS. Its structure is similar to AbstractSS. In addition to the data structures required by AbstractSS, each abstract state node H in the AFSSS tree has associated upper and a lower value bounds $U(H)$ and $L(H)$, and each action node Ha has similar bounds $U(H,a)$ and $L(H,a)$. For state nodes, we maintain the Boolean value $expanded(H)$, which we use to identify non-terminal state nodes for which we have not sampled any successors.

The addition of upper and lower value bounds allows us to define an early stopping condition for the sampling procedure [Walsh et al., 2010]. We say that the search has *converged* if

$$L(H_0, a^*) \geq \max_{a \neq a^*} U(H_0, a), \tag{3.20}$$

where $a^* = \arg\max_{a \in \mathcal{A}} L(H_0, a)$. For this early stopping criterion to be sound, the value bounds L and U must bracket the value estimate that AbstractSS would compute. We now define a condition on L and U that ensures that this is the case. We call this condition *admissibility*, but note that our definition is somewhat different from the typical definition of admissibility employed in algorithms like A^* search.

Definition 4. Let $\alpha = \langle \chi, \mu \rangle$ be a state abstraction of a history MDP T inducing an abstract MDP $T/\alpha = \langle \mathcal{H}/\chi, \mathcal{A}, \mathcal{P}_\mu, \mathcal{R}_\mu, s_0, \gamma \rangle$. Let $\rho^\pi(H)$ be a random variable giving the return (sum-of-rewards) from following an abstract policy $\pi \in \Pi(T/\alpha)$ in T/α starting from H, and let $\rho^\pi(H,a)$ be a random variable giving the return from doing a and then following π. A pair of state value bounds $L, U : \mathcal{H}/\chi \mapsto \mathbb{R}$ on the state space $\mathcal{H}/\chi$ is *admissible with respect to* α if for any policy $\pi \in \Pi(T/\alpha)$, with probability 1,

$$L(H) \leq \rho^\pi(H) \leq U(H) \quad \text{for all } H \in \mathcal{H}/\chi.$$

A pair of state-action value bounds $L, U : \mathcal{H}/\chi \times \mathcal{A} \mapsto \mathbb{R}$ is *admissible with respect to* α if for any policy $\pi \in \Pi(T/\alpha)$, with probability 1,

$$L(H,a) \leq \rho^\pi(H,a) \leq U(H,a) \quad \text{for all } \langle H, a \rangle \in \mathcal{H}/\chi \times \mathcal{A}.$$

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. Figures 4.1a and 4.1c illustrate the structure of two abstract FSSS trees. Note that we continue to assume the use of the empirical weight function $\bar{\mu}$.

Definition 5. An *abstract FSSS tree with respect to* χ, or a χ*-FSSS 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 $M(H, a) \geq C \quad \forall a \in \mathcal{A}$;
3. L and U are admissible with respect to $\alpha = \langle \chi, \bar{\mu}^+ \rangle$ (Definition 4);
4. $\mathcal{F}$ satisfies the convergence criterion (3.20).

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 weighted by their empirical frequency. The additional parameters to AFSSS are the sparse sampling width and depth C and d, admissible value 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 4.3). 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 χ-FSSS tree is defined by

$$\begin{aligned}\mathcal{F}_0(s_0, V_{\min}, V_{\max}) &= \langle N_0, L, U, H_0, \chi \rangle \\ \text{where } H_0 &= \{s_0\}, N_0(H) = \mathbb{1}_{H=H_0}, \\ L(H_0) &= V_{\min}, U(H_0) = V_{\max}, \chi = \emptyset.\end{aligned} \tag{3.21}$$

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 $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ be a history MDP, and let χ be a (p, q)-consistent history equivalence relation. The procedure* AFSSS(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*

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\beta_\gamma(d)(\lambda + p + \delta q),$$

where δ is the divergence (3.7) of the completed empirical weight function $\bar{\mu}^+$ derived from the empirical weight function $\bar{\mu}$ computed by AFSSS*.*

Proof. 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 with respect to $\alpha = \langle \chi, \bar{\mu}^+ \rangle$ for all $a \in \mathcal{A}$. Since $\mathcal{F}$ satisfies the convergence criterion (3.20), 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 $\mathcal{Q}(H_0, a^*) \geq \mathcal{Q}(H_0, a)$ for all $a \neq a^*$, where $\mathcal{Q}$ is the abstract Q-function of an *arbitrary* AbstractSS(C, d, χ) search tree that contains the AFSSS tree as a subset. Thus AFSSS provides the same guarantees as AbstractSS (Proposition 2). □

3.5.5 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

Algorithm 3 Abstract Forward Search Sparse Sampling

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) **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**
    $L(H) \leftarrow \mathcal{R}(H)$, $U(H) \leftarrow \mathcal{R}(H)$
  else
    if not *expanded*(H) **then** EXPAND(H, χ_0)
    $a^* \leftarrow \arg\max_a U(H, a)$
    $H^* \leftarrow \arg\max_{H' \in K(H, a^*)} [U(H') - L(H')]$
    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, C)
  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)
  $L(H, a) \leftarrow \mathcal{R}(H) + \gamma \sum_{H' \in K(H,a)} \frac{N(H')}{M(H,a)} L(H')$
  $U(H, a) \leftarrow \mathcal{R}(H) + \gamma \sum_{H' \in K(H,a)} \frac{N(H')}{M(H,a)} U(H')$
procedure BACKUP(H)
  $L(H) \leftarrow \max_a L(H, a)$
  $U(H) \leftarrow \max_a U(H, a)$

a *weak simulator*, which is a generative model from which complete state-action histories can be sampled under a fixed sampling policy. We refer to the process of generating a single history sample as 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. Given a history $h = s_0 a_0 s_0 \dots a_{d-1} s_d$, the probability of sampling h under sampling policy π starting from h_0 is denoted $P^\pi(h|h_0)$ and given by

$$P^\pi(h|h_0) = \pi(h, \omega) \prod_{t=0}^{d-1} \pi(h_t, a_t) P(h_{t+1}|h_t, a_t). \tag{3.22}$$

To implement an abstract TS algorithm, we need to sample abstract histories $H = S_0 a_0 S_1 \dots a_{d-1} S_d$ from the probability distribution over trajectories in T/α when executing a fixed abstract policy ξ, which again uses the modified action space $\mathcal{A} \cup \{\omega\}$. We denote this probability by $\mathcal{P}^\xi_\alpha(H|h_0)$, where

$$\mathcal{P}^\xi_\alpha(H|h_0) = \xi(H, \omega) \prod_{t=0}^{d-1} \xi(H_t, a_t) \mathcal{P}_\mu(H_{t+1}|H_t, a_t). \tag{3.23}$$

We need to sample from $\mathcal{P}^\xi_\alpha$ but we have access only to P^π. The obvious way forward is to sample a ground history from P^π and then apply the abstraction relation χ to it. At the same time, we want the sampling process to be guided by the statistics of the *abstract* tree N. Thus π should be the *grounded* version of an abstract policy ξ, that is $\pi = \downarrow \xi$ (3.2), where $\xi = \xi(N)$ is parameterized by N. The following result shows that this approach in fact yields a sample from $\mathcal{P}^\xi_{\langle \chi, \mu^* \rangle}$, which is the abstract trajectory distribution with respect to χ and the *optimal* weight function μ^*.

Proposition 4. *Consider a history MDP T augmented with the special action ω and an abstraction $\alpha = \langle \chi, \mu^* \rangle$ of T composed of equivalence relation χ and the corresponding optimal weight function μ^*. Let H' be a random variable $H' \sim \mathcal{P}^\xi_\alpha(\cdot|h_0)$ for a fixed abstract policy $\xi \in \Pi(T/\alpha)$, and let h' be a random variable such that $h' \sim P^{\downarrow\xi}(\cdot|h_0)$. Then the random variable $[h']_\chi$ is equal in distribution to H'.*

Proof. Appendix A.3. □

Algorithm 4 Abstract Trajectory Sampling (with UCT Variation)

procedure AbstractTS(s_0)
  while time remains **do**
    Visit(s_0)
  return $\arg\max_{a \in \mathcal{A}} \mathcal{Q}(\{s_0\}, \cdot)$
procedure Visit(h)
  if h is terminal **then**
    return 0
  Let $H = [h]_\chi$
  if $N(H) = 0$ **then**
    Let $v =$ Evaluate(h)
  else
    Let $a =$ Select(H)
    Let $h' \sim P(\cdot|h, a)$
    Let $q =$ Visit(h')
    Let $v = R(h) + \gamma q$
  Update(H, a, v)
  $n(h) \leftarrow n(h) + 1$
  return v
procedure Select(H)
  if $\exists a \in \mathcal{A} : M(H, a) = 0$ **then**
    return a
  Let $\mathcal{U}(H, a) = \mathcal{Q}(H, a) + c\sqrt{\frac{\log N(H)}{M(H,a)}}$
  return $\arg\max_{a \in \mathcal{A}} \mathcal{U}(H, a)$
procedure Evaluate(h)
  if **len**$(H) = D$ **then**
    return 0
  else
    Let $a \sim$ Uniform($\mathcal{A}$)
    Let $h' \sim P(\cdot|h, a)$
    return $R(h) + \gamma$ Evaluate(h')
procedure Update(H, a, v)
  $\mathcal{Q}(H, a) \leftarrow \mathcal{Q}(H, a) + \frac{v - \mathcal{Q}(H,a)}{n(H,a)}$

Proposition 4 shows that we can sample from $\mathcal{P}^{\xi}_{\langle \chi, \mu^* \rangle}(\cdot|h_0)$ by sampling a history from the ground dynamics $P^{\downarrow \xi}(\cdot|h_0)$ and then abstracting the ground history with χ, without explicitly computing μ^*.

We give a generic abstract TS algorithm based on this approach in Algorithm 4. We also show the concrete implementations of Select, Evaluate, and Update that together create the abstract UCT algorithm. 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.

3.5.6 Handling Action Constraints

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. Model "illegal" actions as having no effect and possibly 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 (Chapter 4) more difficult, since refining the abstraction can expand the legal action sets for the aggregate states. Note that option 3 with a penalty of $-\infty$ for executing an illegal action has a similar effect to option 2.

The problem of action constraints is a fundamental obstacle to state abstraction. The framework of approximate MDP homomorphisms [Ravindran and 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 to determine that no useful one exists.

3.6 Related Work

Much of the theory of state abstraction in MDPs is based on the framework of stochastic bisimilarity [Givan et al., 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 et al., 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.

Most other work on abstraction in MCTS has focused on trajectory sampling algorithms. The AS-UCT algorithm proposed by Jiang et al. [2014] performs an abstract UCT search with an approximate MDP homomophism [Ravindran and Barto, 2004] computed from trajectory samples. The ASAP-UCT algorithm of Anand et al. [2015] extends ASAP-UCT to abstract $\langle h, a \rangle$ pairs, which enables the abstraction to take advantage of action symmetries. OGA-UCT [Anand et al., 2016] is an incremental version of ASAP-UCT that interleaves sampling and abstraction revision. We will revisit these algorithms in Chapter 4.

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 and Schaeffer, 1998; Edelkamp, 2001] and has been developed into methods such as merge-and-shrink heuristics [Helmert et al., 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.

In addition to state abstraction, action abstraction and temporal abstraction have also been applied in MCTS. The TLS algorithm [Van den Broeck and Driessens, 2011] builds a search tree over action equivalence classes. Pinto and Fern [2014] use action pruning to speed up UCT and are 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 et al., 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 et al., 1994; Nau et al., 2003].

3.7 Summary

In this chapter we have developed a framework for state abstraction in MCTS. We defined an abstraction criterion that adapts the sound state aggregation criteria of Li et al. [2006] to history MDPs and generalizes these critera to include unsound abstractions. We then showed that the regret due to using these unsound abstractions for decision making with tree search is bounded. Finally, we showed how the MCTS algorithms sparse sampling, forward search sparse sampling, and UCT / trajectory sampling, can be adapted to use state abstraction. In the next chapter, we address the problem of discovering appropriate state abstractions for tree search and evaluate abstract MCTS algorithms empirically.

Chapter 4: Progressive Abstraction Refinement for Sparse Sampling

4.1 Introduction

It is difficult to assess the quality of a state abstraction for tree search *a priori* because of the complex interaction of abstraction 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 decelerating 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.

4.2 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 $\psi \prec \chi$.

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

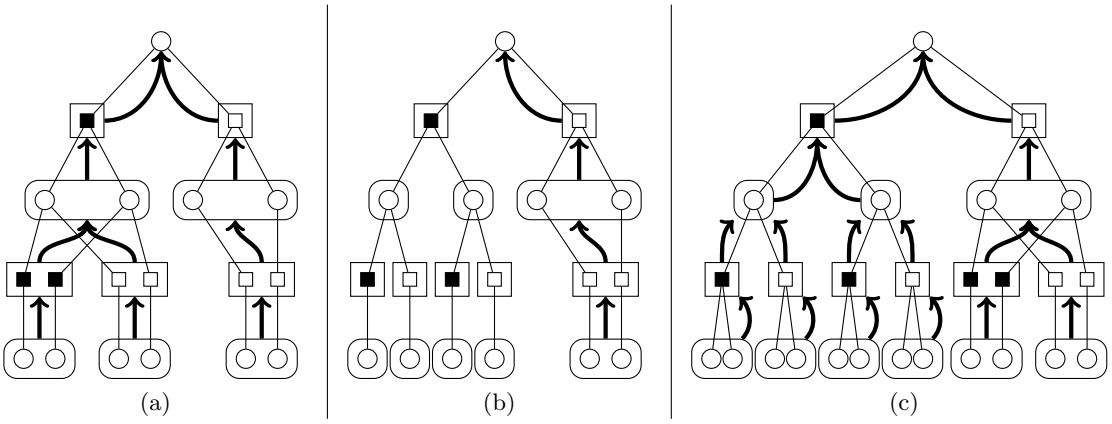


Figure 4.1: (a) An abstract FSSS tree of width $C = 2$ and depth $d = 2$. The small circles and squares represent ground state and action nodes, respectively. Ground nodes are aggregated into abstract nodes, but the structure of the ground tree is retained. The arrows show how value estimates propagate in the abstract tree. 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.

ground histories of the same length and containing the same action sequence to the same abstract history, $[h]_\top = \{g \in \mathcal{H}^{\mathbf{len}(h)} : a_i(h) = a_i(g), i = 1, \ldots, \mathbf{len}(h)\}$. Searching in the abstract problem $T/\top$ amounts to searching for the best open-loop policy in T, while searching in $T/\bot$ is equivalent to searching in the ground 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 $F^*(\chi) = \bot$. 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.

4.3 Progressive Abstraction Refinement for Sparse Sampling

The Progressive Abstraction Refinement for Sparse Sampling (PARSS) algorithm (Algorithm 7) is an adaptation of AFSSS that refines its abstraction during search. PARSS

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(pre(H), a(H)) ← REFINE(χ(pre(H), a(H)))
5:     SPLIT(pre(H), a(H), χ)
6:     UPDATETREE(pre(H), a(H))
```

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 PARSS 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 $\chi(\mathbf{pre}(H), a(H))$ 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 4.4.

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 such 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. PARSS uses a modified version of AFSSS in which the SAMPLE procedure is replaced

by the SampleModified procedure defined in Algorithm 6. In the modified version, when sampling an abstract action node Ha, rather than sampling ground states from $\bar{\mu}(H, \cdot)$ as in AFSSS, 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 $m(h, a) \leq C$ at all times for all ground action nodes ha in the sample tree, which guarantees that PARSS never draws more samples than $\perp$-SS (Section 4.3.1).

Algorithm 6 Modified Sample procedure for PARSS

```
1: procedure SampleModified(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
```

4.3.1 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 for all $h, g \in H$, $h = g$.

If a state node H is pure, then nothing is accomplished by further refining H. Note that this need not imply that $\chi(\mathbf{pre}(H), a(H)) = \perp$, 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 $\text{REFINE}(\chi) \prec \chi$.

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 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 search trees $(\mathcal{N}_0, \mathcal{N}_1, \ldots)$, each with respect to its corresponding χ_t. Proposition 5 establishes that $\mathcal{N}_t$ 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 $\mathcal{N}_\tau$ is a $\bot$-FSSS tree. Lemma 7 shows that $\bot$-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 $\mathcal{T}$ is a χ-*FSSS *tree, then after one iteration of the loop in Algorithm 7, Line 4, the resulting tree $\mathcal{T}'$ is a ψ-*FSSS *tree for some ψ such that $\psi \prec \chi$.*

Proof. By assumption, $\text{REFINE}(H)$ produces a new abstraction ψ with the property $\psi(\mathbf{pre}(H), a(H)) \prec \chi(\mathbf{pre}(H), a(H))$, and therefore $\psi \prec \chi$. The SPLIT operation partitions the subtree rooted at H according to ψ, establishing condition (5.1). The UPSAMPLE loop in UPDATETREE (Line 15) adds samples and performs backups in the subtree of H to establish (5.2) and (5.3) for the subtree. Then values are backed up from H to the root node (Line 17), which establishes (5.3) for the rest of the tree. Finally, the call to AFSSS (Line 6) establishes convergence (5.4). □

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 abstract tree* $\mathcal{T}$ *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 4.2), the abstraction relations $\chi(H, a)$ will be equal to $\perp$ 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 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 in 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 incurs 0 value estimation error from state aggregation (Theorem 1). 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 contains an error. 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 $\mathcal{T}$ with the same worst-case sample complexity as SS (ie. $O((|\mathcal{A}|C)^d)$). $\mathcal{T}$ is different from a ground FSSS tree in that states that are equal in the ground representation are aggregated in $\mathcal{T}$. Because the FSSS pruning mechanism is sound when L and U are admissible, $\mathcal{T}$ achieves the same error bounds as an SS tree in which identical states are aggregated. 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 as well as SS and FSSS in terms of worst case sample complexity and error bounds if both 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 4.6 and 4.7) that PARSS has an advantage compared to FSSS and AFSSS in this regard.

4.3.2 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 $h \in H$ 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 condition.

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 21). 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 $\mathcal{P}_{\bar{\mu}}(\cdot|H, a)$. Thus 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.

4.3.3 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 3.5), 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 obstacle 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 currently optimal subtree. 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 particular design choices. Nevertheless, most other work on abstract MCTS is based on TS algorithms, not SS algorithms (Section 4.5), and we feel that abstraction refinement in TS is an important area for further work.

4.4 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.

4.4.1 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.

4.4.1.1 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 once a state node becomes *pure*, it never becomes *impure* again since its ancestors have already been refined. This makes breadth-first selection the easiest to implement.

4.4.1.2 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 active 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 naivete.

4.4.1.3 Heuristic Guided Selection

Most generally, we can define a priority ordering over the set of active 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,

$$q(h,a) = R(h) + \frac{1}{m(h,a)} \sum_{h' \in k(h,a)} n(h') \max_{a' \in \mathcal{A}(h)} q(h', a'). \tag{4.1}$$

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,

$$f_{\sigma^2}(H) = \frac{1}{\sum_{a \in \mathcal{A}} M(H,a)} \sum_{a \in \mathcal{A}} M(H,a) \sigma^2(H,a). \tag{4.2}$$

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 effectively sound.

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

$$f_{\text{bf}}(H) = \frac{1}{\textbf{len}(H)}, \tag{4.3}$$

$$f_{\text{unif}}(H) = 1, \tag{4.4}$$

with ties being broken randomly.

4.4.2 State Abstraction Refinement

An abstraction relation is essentially a multilabel classifier, and many standard techniques in classification or clustering could serve as a basis for refinement strategies. In our experiments, we tried the following two approaches.

4.4.2.1 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 of approximately equal size to form the refined abstraction. 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 $N(H)$.

4.4.2.2 Decision Tree-based Refinement

If we have access to a set of features $\{\phi_i(h)\}$ for each state, we can take a more sophisticated approach. The DT refinement strategy represents abstractions as 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 (4.1), $u(h, a)$ 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 clustering algorithm.

4.5 Related Work

The idea of adaptive refinement or revision of an abstraction has been the basis for several MDP abstraction algorithms, including the G algorithm [Chapman and Kaelbling, 1991], the Parti-Game algorithm [Moore and Atkeson, 1995], and the UTree algorithm [McCallum, 1996]. Baum et al. [2012] propose 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 4.4). 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 MCTS 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] derive suboptimality bounds for their abstractions using the theory of MDP homomorphisms [Ravindran and Barto, 2004]. The ASAP-UCT algorithm of Anand et al. [2015] extends ASAP-UCT to abstract $\langle h, a \rangle$ pairs, which enables the abstraction to take advantage of action symmetries. OGA-UCT [Anand et al., 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 MCTS 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 POMDP view of abstraction 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 et al., 1999]. PARSS essentially enumerates and evaluates policies in a certain order deter-

mined 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].

4.6 Experiments

Our experiments compare multiple 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`.

4.6.1 Algorithms

We tested six different variations of PARSS which were obtained as the cross product of the three node selection strategies Breadth-First (BF), Uniform, and Variance (Section 4.4.1) and the two refinement strategies DT and Random (Section 4.4.2). We compared these PARSS variants to $\bot$-FSSS and $\top$-FSSS, and also to AFSSS with two random abstractions of different granularities (rand-FSSS).

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.

4.6.2 Domains

Our problem pool includes the domains used by Hostetler et al. [2015] as well as several additional problems.

4.6.2.1 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. SAVING is an episodic task in which 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 $T_i, T_b, T_m > 0$. Its state space consists of integers $\langle p, t_b, t_i, t_m \rangle$, where $p \in \{p_{\min}, p_{\max}\}$, $t_b \in \{0, T_b\}$, $t_i \in \{0, T_i\}$, and $t_m \in \{0, T_m\}$.

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 can have only one investment at a time.

We instantiate the SAVING problem with parameters $p_{\min} = -4$, $p_{\max} = 4$, $T_i = 4$, and $T_b = 4$. 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 $\mathbb{E}\big[p\big] = 0$. *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 $Q^*(s, \mathit{invest}) > Q^*(s, \mathit{save})$, but there are some policies π — in particular, the optimal policy $\pi^*_\top$ under abstraction $\top$ — for which $Q^\pi(s, \mathit{invest}) < Q^\pi(s, \mathit{save})$. Because $\pi^*_\top$ cannot discriminate between states, it estimates the future value of *sell* as $\mathbb{E}\big[p\big] = 0$. Thus the optimal policy under $\top$ is to always *save*. When we estimate Q-values using this policy, we find that $Q^{\pi^*_\top}(s, \mathit{invest}) < Q^{\pi^*_\top}(s, \mathit{save})$ 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 in [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 4.4.1). 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.

4.6.2.2 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 depends on the angle relative to the wind and the Euclidean distance to the neighboring location. The wind blows in one of the same 8 directions, and either stays the same or switches to a neighboring direction uniformly at random every step. 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 use the same random problem instances for all of the algorithms to reduce variance.

4.6.2.3 Racetrack

Racetrack is a classic domain introduced by Barto et al. [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, and crashing the car gives a penalty equal to the maximum cumulative step cost, which means that crashing is always worse than not achieving the goal. We used both the Small and Large grid topologies of Barto et al. [1995].

4.6.2.4 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.

In Spanish Blackjack, not all actions are legal in all states. We thus require the abstraction not to aggregate states with different legal action sets. This means that wherever we would otherwise use the top abstraction $\top$, we instead use the coarsest abstraction that respects the action constraints.

4.6.2.5 Academic Advising

Academic Advising ("Advising") is a modification of the IPC problem of the same name [Guerin et al., 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

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

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. In our experiments, we set $g = 4$, $g^* = 2$, $\eta = 0.2$, and $\eta_0 = 0.8$.

4.6.2.6 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. Getting hit prevents the agent from moving for the rest of the episode but gives no immediate penalty. Thus a planner cannot identify getting hit as a bad outcome unless it has already found a policy that reaches the goal with non-zero probability.

4.6.2.7 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, since further arrival events at those floors have no effect.

4.6.2.8 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 *empty*, occupied with a *native* plant, or occupied with a *tamarisk* plant. Plants spread stochastically to empty 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.

4.6.2.9 Tetris

Tetris is the classic videogame of stacking differently-shaped blocks. It has quite a long history in AI research (e.g. [Bertsekas and Ioffe, 1996; Gabillon et al., 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 and Ioffe, 1996] as the ground representation.

In Tetris, some types of blocks have more legal positions than others because they are longer in the horizontal direction in certain orientations. In order to have the same legal action set in every state, we map illegal positions to the nearest legal position, which

simply entails translating the block horizontally so that it is in bounds. This makes the planning problem slightly harder, since there are now redundant actions that have the same effect, and samples are wasted evaluating these actions. Similarly, some blocks have only two distinct orientations while others have four, resulting in further action redundancy.

4.6.3 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 average return of algorithm A running on problem M with per-decision budget b and parameters θ. 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 $b \in \mathcal{B}$. 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.

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 criterion $\rho^*_M(A, b)$ to be more realistic.

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 domains due to memory limits. The random branching factor B was varied over either $\{2, 3, 5\}$ or $\{2, 4\}$ depending on the domain. Early experiments used $\{2, 3, 5\}$, while in later experiments we reduced this to $\{2, 4\}$ to limit the parameter search space.

We chose not to perform any experiments with a time budget, and to instead use

sample budget as a proxy. This decision was based on earlier results indicating that the relative performance of the algorithms was similar for both time and sample budgets [Hostetler et al., 2015]. Time budget experiments take significantly longer to run due to the expensive system calls needed to measure execution time accurately in our environment. Focusing on sample budgets allowed us to examine more domains and to more thoroughly optimize the algorithm parameters.

4.7 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.

Note when interpreting the charts in Figures 4.2, 4.3, and 4.4 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.

4.7.1 Performance of PARSS

PARSS was the best algorithm overall in five problems: the two SAVING problems, the two RACETRACK problems, and ADVISING (Figure 4.2). In all other domains (Figures 4.3 and 4.4), 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 appar-

ent in RACETRACK SMALL if that experiment were to be continued to larger budgets. In ADVISING, all of the algorithms improve steadily with increasing budgets, but PARSS is consistently best.

4.7.2 Performance of ⊤-FSSS

There were five domains in which ⊤-FSSS outperformed ⊥-FSSS over most of the range of budgets (Figure 4.3). ⊤-FSSS also performed well in the two RACETRACK domains (Figure 4.2), although its performance began to plateau at larger budgets. We would expect this plateau to occur in most domains if we continued the experiments to sufficiently large budgets, since the optimal ground policy usually will not be representable in the ⊤-abstract state space.

⊤-FSSS was generally inferior to ⊥-FSSS on the SAVING and SAILING problems, and to a small extent also in ADVISING. This was the expected result in SAVING because ⊤-FSSS cannot estimate the value of the *invest* action correctly. Note however that ⊤-FSSS is superior to ⊥-FSSS for the smallest budgets because ⊥-FSSS estimates the value of *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.

4.7.3 Performance of ⊥-FSSS

⊥-FSSS performed well on the two SAILING problems and on ELEVATORS (Figure 4.4). 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 indicates that the latter, near-deterministic situation is much more common. Note that PARSS did equally as well as ⊥-FSSS in these domains.

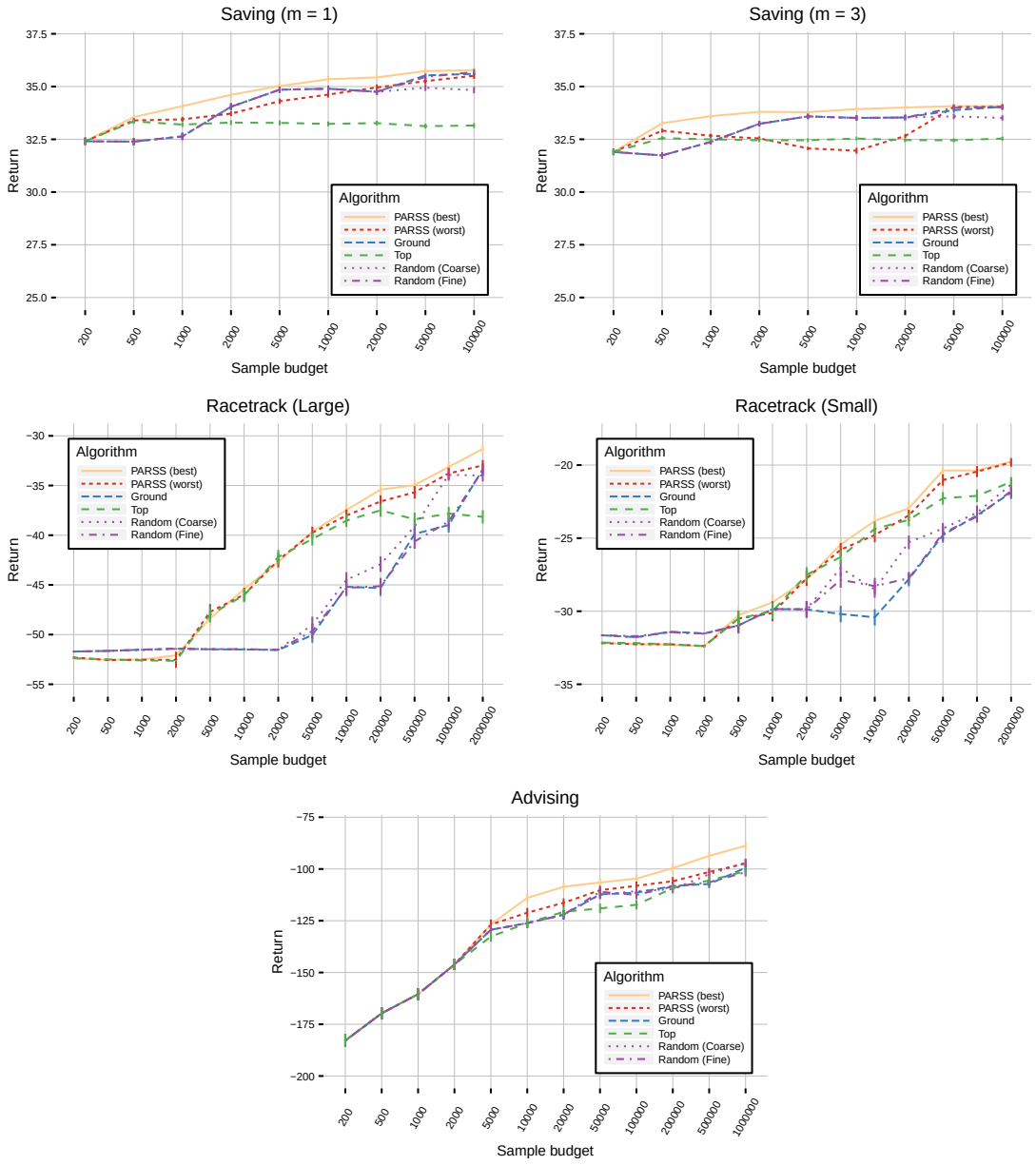


Figure 4.2: Domains where PARSS outperformed all other algorithms.

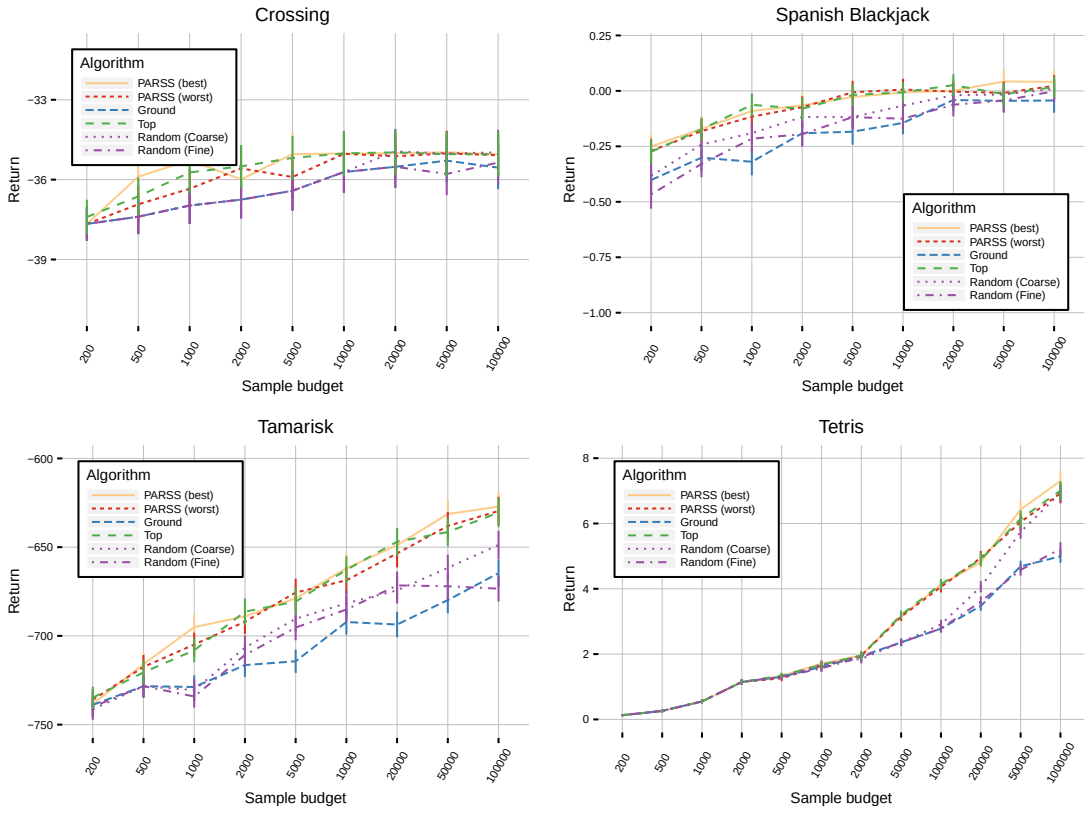


Figure 4.3: Domains where ⊤-FSSS outperformed GROUND and RANDOM. Note that all PARSS variants performed equally as well as ⊤-FSSS.

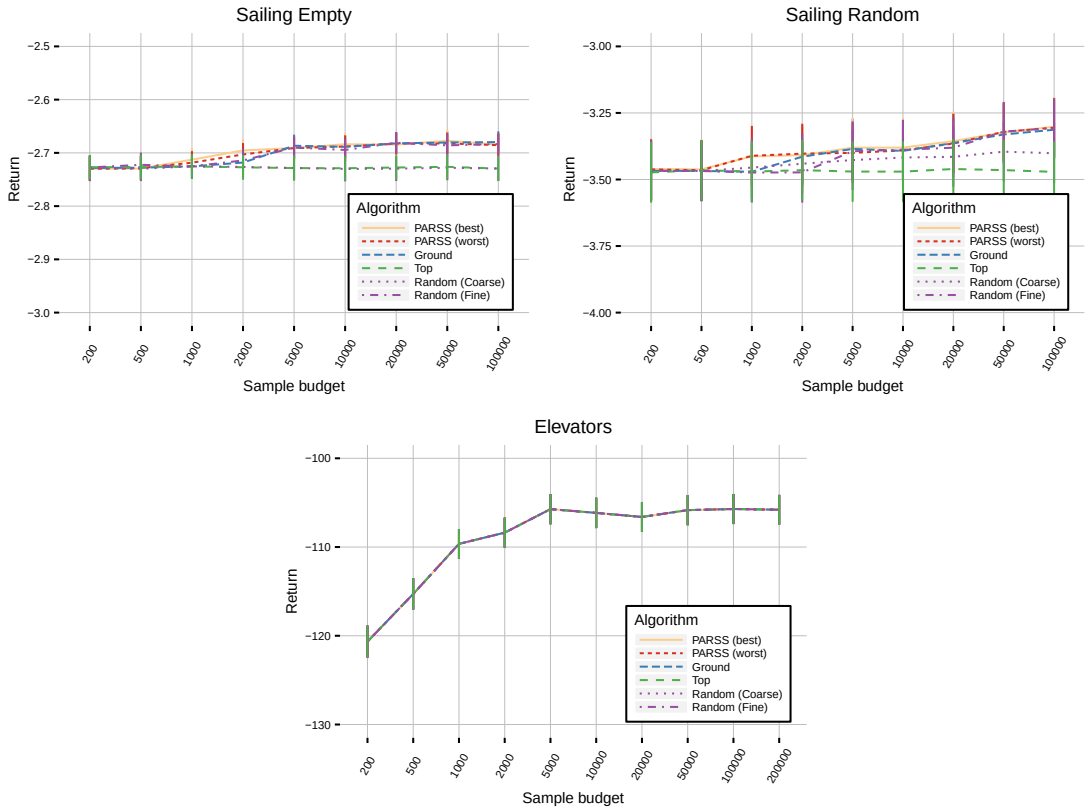


Figure 4.4: Domains where $\perp$-FSSS was best. Note that all PARSS variants performed equally as well as $\perp$-FSSS. In the ELEVATORS domain, the best performance occurred when the width parameter was $C = 1$. Since all the algorithms are equivalent if $C = 1$, the results shown are identical.

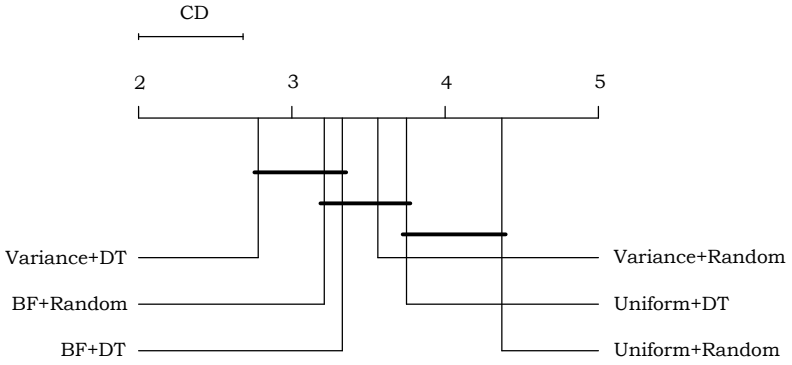


Figure 4.5: A critical difference plot [Demšar, 2006] showing the pairwise differences in performance among the PARSS variants. The horizontal scale shows the average rank of each algorithm, with smaller ranks indicating better performance. Algorithms connected by a dark line had statistically identical performance at the $p = 0.05$ level. This plot was produced by the R package `scmamp` [Calvo and Santafe, 2015].

4.7.4 Comparing PARSS Variations

We can see from Figures 4.2, 4.3, and 4.4 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 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 4.6).

We made a statistical comparison of the overall relative performance of the six PARSS variations using Friedman's test [Demšar, 2006], which detects an overall effect of the choice of algorithm on performance across multiple problems. We consider each combination of problem domain 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 problem domains. The test revealed 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] to correct for multiple comparisons. These pairwise comparisons are summarized

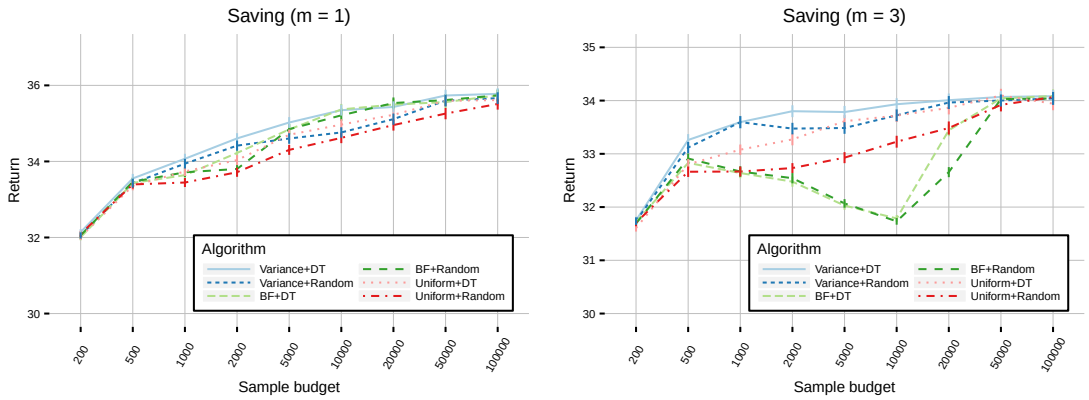


Figure 4.6: Comparing all PARSS variations on the SAVING domain. The BF order performs poorly when $T_m = 3$ because it refines many abstraction relations that are already sound.

in Figure 4.5 using a *critical difference plot* [Demšar, 2006]. While there was no single best or worst algorithm, 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 results in a larger number of state nodes becoming fully-refined, and in fully-refined state nodes the refinement mechanism is no longer relevant.

4.7.5 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 PARSS with the UNIFORM+RANDOM refinement strategy. The UNIFORM+RANDOM strategy may well produce intermediate trees with abstractions whose inaccuracies are similar to 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 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.

4.7.6 Stochastic Branching Factor vs. Best Algorithm

The four domains where $\top$-FSSS was (tied for) best were also the four domains with by far the largest minimum stochastic branching factors (Table 4.1). 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 domains, suggesting that the performance gap between PARSS and $\bot$-FSSS on some of these domains is due to other structural features of the problem in addition to the branching factor.

4.7.7 On 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 an open-loop policy will correctly rank the values of root actions. We can interpret $\top$-FSSS as a policy rollout algorithm (2.3) 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 most of these domains exhibit aspects of the Weinstein-

Table 4.1: 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.

	Branching		
Problem	Min	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	

Littman structure [Weinstein and 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. Two of the domains — SAVING(1) and SAVING(3) — 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 in a riskier (i.e., faster) way. 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 $\bot$-FSSS for the same budgets. While $\top$-FSSS incurs error due to abstraction (Theorem 1), $\bot$-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.

4.7.8 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 MCTS.

These difficulties also highlight the main disadvantage of PARSS compared to AFSSS with a fixed abstraction, which 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 4.3.2). 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.

4.7.9 Summary of Results

The experimental results indicate that PARSS is superior or equal to $\bot$-FSSS on a range of problem domains 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. Since PARSS provides the same bounded error guarantees as $\bot$-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.

4.8 Summary

This chapter described the Progressive Abstraction Refinement for Sparse Sampling (PARSS) algorithm [Hostetler et al., 2015], which addresses the problem of choosing the correct abstraction for MCTS by progressively refining an initially coarse abstraction during search. Our analysis of PARSS showed that it provides the same asymptotic 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 SS and FSSS. Drawbacks of PARSS include additional implementation complexity and sometimes a higher memory footprint.

Algorithm 7 Progressive Abstraction Refinement for SS

procedure PARSS(h_0, C, d)
  Let $\mathcal{F} = \mathcal{F}_0(s_0, V_{\min}, V_{\max})$ (3.21)
  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)
  if H is a leaf **then return**
  for all $H' \in K(H, a)$ **do**
    Let $\mathcal{G}' = H'/\chi(H, a)$ ▷ Refined partition
    for all $\langle G', a' \rangle \in \mathcal{G}' \times \mathcal{A}$ **do**
      $\chi(G', a') \leftarrow \chi(H', a')$ ▷ Copy old relation
      Split(G', a', χ)
procedure UpdateTree(H, a)
  for all $H' \in K(H, a)$ **do**
    UpSample(H')
  for t from 0 to **len**(H) **do** ▷ Backup path to root
    for all $a \in \mathcal{A}$ **do** Backup(H, a)
    Backup(H)
    Let $H = \mathbf{pre}(H)$
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)

Algorithm 8 Modified SAMPLE procedure for rand-FSSS

procedure SAMPLERAND(H, a)
  for all $h \in H$ **do**
    while $m(h, a) < \lceil \frac{C}{n(H)} \rceil$ **do**
      Let $h' \sim P(\cdot|h, a)$
      $n(h') \leftarrow n(h') + 1$
  for all $h' \in \bigcup_{h \in H} k(h, a)$ **do**
    if $\exists G \in K(H, a)$, $g \in G$ where $h' = g$ **then**
      continue
    else if $|K(H, a)| < B$ **then**
      Add new equivalence class $\{h'\}$ to $\chi(H, a)$
    else
      Let $G = \arg\min_{H' \in K(H,a)} N(H')$
      Modify $\chi(H, a)$ so that $[h']_{\chi(H,a)} = G$.

Chapter 5: Extending PARSS: Abstraction Diagrams and Progressive Abstract Tree Search

Joint work with Ankit Anand

5.1 Introduction

In the previous chapters we have focused on abstract MCTS algorithms that use state abstraction. State abstraction, though, is only one form of MDP abstraction. In this chapter, we introduce a more general framework for progressive abstract tree search. We describe a formalism for expressing MDP abstractions called an *abstraction diagram* and show how several forms of abstraction described in the literature can be expressed as an AD. We then define abstract MCTS algorithms that build search trees with respect to an arbitrary AD, which allows us to use all of these types of abstraction in tree search. Finally, we show how to define refinement operators on ADs, which facilitates the construction of MCTS algorithms based on abstraction refinement similar to the PARSS algorithm.

5.2 MDP Abstractions as Policy Set Constraints

To allow us to place multiple abstraction modalities within a common formalism, we require a slightly generalized setting. Let $M = \langle \mathcal{S}, \mathcal{A}, P, R, s_0 \rangle$ be an MDP with an identified start state s_0. We augment M with a set of *base policies* $B = \{b_i\}$, where each b_i is a stationary stochastic policy $b_i : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ over the ground state and action sets. Denote by $B(s) \subseteq B$ the set of base policies admissible in state $s \in \mathcal{S}$. These base policies will be treated as the primitive actions, and we define their dynamics as

$$P(s'|s, b) = \sum_{a \in \mathcal{A}} b(s, a) P(s'|s, a), \tag{5.1}$$

$$R(s, b, s') = \sum_{a \in \mathcal{A}} b(s, a) R(s, a, s'). \tag{5.2}$$

Note that the original action space corresponds to a set of base policies $B_{\mathcal{A}} = \{b_a : a \in \mathcal{A}\}$ where each b_a is a policy that only plays a, that is $b_a(s, a') = \mathbb{1}_{a'=a}$.

Solving an MDP entails finding the best policy in some set of candidate policies. An abstraction of an MDP reduces the cost of computing a solution by reducing the size of the candidate policy set. Only those policies that are *consistent* with the abstraction are considered by the policy search. As in previous chapters, we define abstractions over state-action histories. We will use the notation $\mathcal{H}(P, Q)$ to denote the set of histories over "state" set P and "action" set Q. As before, histories always begin and end with a state. Abstract histories will be denoted with the letter τ, rather than the capital H we used previously. We use $\mathbf{last}(h)$ to denote the final element of h. Otherwise we will continue to use the notation for histories defined in Chapter 2.3.1.

Definition 12 (MDP Abstraction)**.** An *abstraction* of an MDP $M = \langle \mathcal{S}, \mathcal{A}, B, P, R, \gamma \rangle$ is a tuple $\langle X, \alpha, \beta \rangle$ consisting of an abstract state set X, an abstraction function $\alpha : \mathcal{H}(\mathcal{S}, B) \mapsto \mathcal{H}(X, B)$, and a function $\beta : X \mapsto 2^B$ where $\beta(x)$ gives the set of legal actions in abstract state x.

Definition 13. Let $\pi : \mathcal{H}(\mathcal{S}, B) \mapsto B$ be a ground policy and let $A = \langle X, \alpha, \beta \rangle$ be an abstraction. Given an abstract history τ, let $H(\tau) = \{h \in \mathcal{H}(\mathcal{S}, B) : \alpha(h) = \tau\}$ be the set of histories that map to τ. Then π is *consistent with* A if and only if for all $\tau \in \mathcal{H}(X, B)$ both of the following hold:

1. For all $h \in H(\tau)$, $\pi(h) \in \beta(\mathbf{last}(\tau))$, and
2. For all $h, h' \in H(\tau)$, $\pi(h) = \pi(h')$.

The first condition requires that π plays only legal actions according to β. The second condition requires that π plays the same action in histories that are considered equivalent according to α.

The concept of a consistent policy is central to our approach to abstraction. Viewing abstraction as a restriction of the policy space allows many different abstraction modalities to be unified in one formalism.

5.3 Abstraction Diagrams

Our goal in this section is to unify the three main categories of abstraction — state abstraction, action abstraction, and temporal abstraction — within a single formalism called an *abstraction diagram*. We first define the formalism, then show how different forms of abstraction are cast within it.

Definition 14. An *abstraction diagram* over state set $\mathcal{S}$ and action set B is a bipartite deterministic finite automaton $G = \langle \mathcal{I}, X, Y, \mathcal{L}, \delta, y_0 \rangle$, where $\mathcal{I}$ is a set of vertices, $X \subset \mathcal{I}$ is the subset of *state vertices*, $Y = \mathcal{I} - X$ is the complimentary subset of *action vertices*, $\mathcal{L} = \mathcal{S} \cup B$ is the set of edge labels, $\delta \subseteq (X \times B \times Y) \cup (Y \times \mathcal{S} \times X)$ is the transition relation, and $y_0 \in Y$ is a designated start vertex. The transition relation δ must be such that in each action node $y \in Y$ the union of the labels of the outgoing edges is the entire state space $\mathcal{S}$, and in each state node $x \in X$ there is at least one outgoing edge.

Intuitively, an action node $y \in Y$ encodes a state equivalence relation in its outgoing edges. Two states $s, s' \in \mathcal{S}$ are considered equivalent in action node y if there are edges $y \xrightarrow{s} x$ and $y \xrightarrow{s'} x$ leading to the same state node x. Similarly, the outgoing edges of state nodes $x \in X$ encode the legal action sets in abstract states, and actions that lead to the same action node y share the same abstraction over futures.

Formally, an abstraction diagram $G = \langle \mathcal{I}, X, Y, \mathcal{L}, \delta, y_0 \rangle$ induces the MDP abstraction $\langle X, \alpha_G, \beta_G \rangle$. The abstraction function $\alpha_G(h) \stackrel{\text{def}}{=} \alpha_G(h, y_0)$ maps ground histories in $\mathcal{H}(\mathcal{S}, B)$ to abstract histories in $\mathcal{H}(X, B)$. It is defined recursively by

$$\begin{aligned} \alpha_G(s, y) &= \delta(y, s) \\ \alpha_G(s_j b_j s_{j+1} \ldots, y) &= x \ : \ b_j \ : \ \alpha_G(s_{j+1} \ldots, y') \\ &\qquad \text{where } x = \delta(y, s_j),\ y' = \delta(x, b_j), \end{aligned}$$

where the colon ":" denotes concatenation and $x = \delta(y, s_j)$ is shorthand for $x : \langle y, s_j, x \rangle \in \delta$. The legal action sets are defined by

$$\beta(x) = \{b \in B : \exists y. \langle x, b, y \rangle \in \delta\}.$$

We now show how a wide range of abstractions from the literature can be expressed in the AD framework. For this purpose we will employ a simple running example MDP,

in which the agent controls a heater and must keep the temperature near a set point. The states of the problem contain a state variable $t \in \mathbb{R}$ giving the current temperature, and we will refer to two different action spaces, either $\{off, on\}$ or $\{off, low, high\}$.

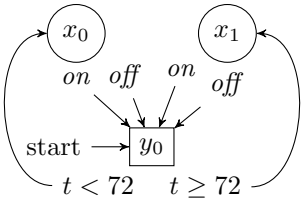
5.3.1 State Aggregation

State aggregation abstractions, as we have seen in previous chapters, group multiple ground states into a smaller number of equivalence classes. An abstraction diagram encodes state aggregation via the successors of its action nodes. Each action node y induces a state abstraction relation χ_y, which is an equivalence relation on $\mathcal{S}$ defined by $s_1 \simeq_{\chi_y} s_2 \iff \delta(y, s_1) = \delta(y, s_2)$. Figure 5.1a shows a state abstraction for our running example, with two abstract states corresponding to temperatures above and below 72 degrees. Notice that this abstraction is stationary with respect to the ground process, meaning that for $h = s_0 b_0 s_1 \ldots$, $a_G(h) = f(s_0) b_0 f(s_1) \ldots$ where $f = \delta(\cdot, y_0)$. Most work on MDP state abstraction deals with stationary abstractions.

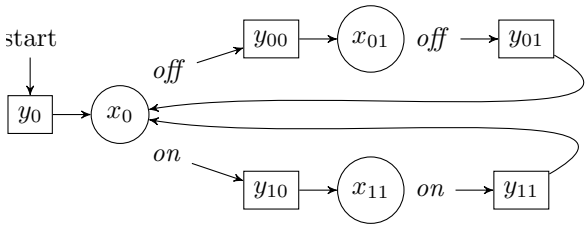
5.3.2 MDP Homomorphisms

MDP homomorphisms [Ravindran and Barto, 2004] generalize the notion of state equivalence to equivalence of state-action pairs. This allows for modeling of action symmetries. An MDP homomorphism consists of a state abstraction function f and action abstraction functions g_s for each state $s \in \mathcal{S}$, where $f : \mathcal{S} \mapsto \bar{\mathcal{S}}$ maps ground states to abstract states, and $g_s : \mathcal{A} \mapsto \bar{\mathcal{A}}$ is a contextual mapping of ground actions to abstract actions. After obtaining a policy π for the abstract problem, we act in ground state s by computing $\bar{a}^* = \pi(f(s))$ and then executing one of the actions in $g_s^{-1}(\bar{a}^*)$.

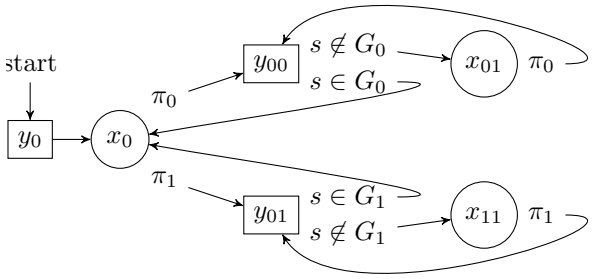
We can express the state abstraction function f in an AD as in the previous section. The action abstraction functions g_s are incorporated as base policies. For each abstract action $\bar{a}$, define the base policy $b_{\bar{a}} : \mathcal{S} \mapsto \mathcal{A}$, where $b_{\bar{a}}(s, \cdot)$ is an arbitrary probability measure over $g_s^{-1}(\bar{a})$. The base policy set is then $B = \{b_{\bar{a}} : \bar{a} \in \bar{\mathcal{A}}\}$.



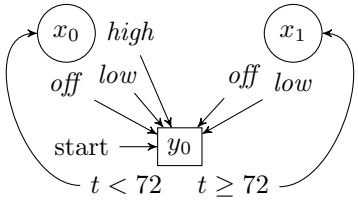
(a) A stationary state abstraction with two abstract states, corresponding to temperatures below and above 72 degrees.



(b) A temporal abstraction representing a recurring decision to set the heater to either *off* or *on* for the next two time steps.



(c) A recurring choice between two options $\langle \pi_0, \mathcal{S}, \omega_0 \rangle$ and $\langle \pi_1, \mathcal{S}, \omega_1 \rangle$, where $\omega_i(s) = \mathbb{1}_{s \in G_i}$ for some set of goal states $G_i \subseteq \mathcal{S}$.



(d) A version of (5.1a) for the 3-action HEATER domain, where the *high* action has been pruned in state node x_1.

Figure 5.1: Examples of abstraction diagrams

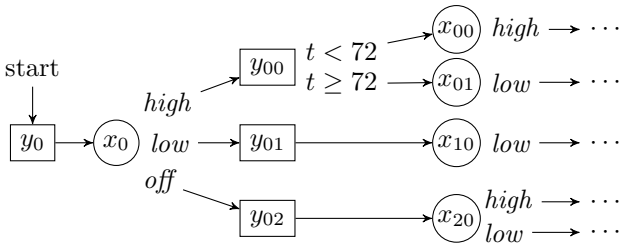


Figure 5.2: Part of a tree-structured AD incorporating several kinds of abstraction.

5.3.3 Temporal Abstraction

A temporal abstraction reduces the rate of decision-making by replacing atomic actions with policies that are followed for more than one time step. The simplest example is a *macro-action*, which is a sequence of actions $a_0 a_1 \dots a_k$ to be followed regardless of random outcomes. The AD in Figure 5.1b describes the set of open-loop policies that choose between two macro-actions.

The *options* formalism [Sutton et al., 1999] generalizes temporal abstraction beyond macro-actions. An option is a tuple $\langle \pi, I, \omega \rangle$, where π is a policy, $I \subseteq \mathcal{S}$ is the set of states where π can be initiated, and $\omega : \mathcal{S} \mapsto [0, 1]$ is the termination probability function. When the termination function is deterministic, that is $\omega(s) \in \{0, 1\}$ for all $s \in \mathcal{S}$, the option can be expressed in an AD directly, as in Figure 5.1c. Stochastic termination functions can be incorporated by augmenting the ground state s with a sample $t \sim \text{Bernoulli}(\omega(s))$ from the termination function, using the modified dynamics

$$P(\langle s', t \rangle | s, a) = \Big[t\omega(s') + (1 - t)(1 - \omega(s')) \Big] P(s' | s, a).$$

With this modification, the option fits into an AD analogous to Figure 5.1c, with $G_i = \{\langle s, t_i \rangle : t_i = 1\}$.

5.3.4 Action Pruning

Action pruning (e.g. Pinto and Fern [2014]) entails making the sets of permissible actions $B(s)$ smaller by removing actions deemed unlikely to be optimal. Action pruning is expressed in an AD by restricting the action set $\beta(x)$ in some state nodes x so that

$\beta(x) \subsetneq B$. Temporal abstraction is actually an extreme case of action pruning, in which $|\beta(x)| = 1$ while "committed" to following a policy (Figures 5.1b and 5.1c).

5.3.5 History Abstractions

Abstractions in the AD formalism can depend on the entire history, not only on the current state. For example, the abstraction induced by the AD in Figure 5.1b is history-dependent because given two states s, s', in general $\alpha(sb_0s') \neq \alpha(sb_1s')$. While temporal abstraction is inherently history-dependent, MDP state abstractions are often assumed to be stationary because MDPs have stationary optimal policies. History-dependent state abstractions are encountered more often in POMDPs (e.g. McCallum [1996]; Hansen [1998]; Meuleau et al. [1999]), because the belief state is a function of the entire observation history. In the extreme case, the AD may have a tree structure, in which case every path through the AD could induce a different abstraction (Figure 5.2). In the algorithms described in Chapters 3 and 4, abstraction relations were associated with nodes in the search tree, implicitly creating an AD with a tree structure. However, policies that are consistent with an AD need not have the same structure as the AD. We can use the abstract MCTS algorithms we describe next with any AD, regardless of whether the AD is a tree.

5.4 Tree Search with ADs

We represent a search tree over the ground problem as a multiset of ground histories. Let $n : \mathcal{H}(\mathcal{S}, B) \mapsto \mathbb{N}^{\geq 0}$ be the multiplicity function. We use the notation $h \in n \iff n(h) > 0$ to indicate membership. The tree structure implies that $h \in n \Rightarrow \mathbf{pre}(h) \in n$. An abstract search tree is a "view" of n in which nodes correspond to sets of histories. The nodes of the abstract tree are identified with abstract histories $\tau \in \mathcal{H}(X, B)$, and each abstract node τ has a corresponding set of ground histories $H(\tau)$. The abstract tree is also a multiset, this time of abstract histories, and we denote its multiplicity function by $N : \mathcal{H}(X, B) \mapsto \mathbb{N}^{\geq 0}$. We will describe these algorithms using notation similar to that introduced in Chapters 3 and 4, except that now everything will be defined in terms of n and H. From these two objects, we define the ground tree successor relation and action

counts

$$k(h,b) \stackrel{\text{def}}{=} \{h' \in n : \mathbf{pre}(h') = hb\}$$
$$m(h,b) \stackrel{\text{def}}{=} \sum_{h' \in k(h,b)} n(h'),$$

the abstract tree visit counts and successor relation

$$N(\tau) \stackrel{\text{def}}{=} \sum_{h \in H(\tau)} n(h)$$
$$K(\tau,b) \stackrel{\text{def}}{=} \{\tau' \in N : \mathbf{pre}(\tau') = \tau b\}$$
$$M(\tau,b) \stackrel{\text{def}}{=} \sum_{\tau' \in K(\tau,b)} N(\tau'),$$

and the empirical weight function and abstract reward function

$$\bar{\mu}(\tau,h) \stackrel{\text{def}}{=} \mathbb{1}_{h \in H(\tau)} \frac{n(h)}{N(\tau)}$$
$$\bar{R}(\tau) \stackrel{\text{def}}{=} \frac{1}{N(\tau)} \sum_{h \in H(\tau)} n(h) R(h).$$

Note that since all of these are determined by n and H, in our pseudocode we show updates to n and H only. Abstract SS with an AD (ADSS) is implemented in Algorithm 9, and Abstract TS with an AD (ADTS) is implemented in Algorithm 10. An ADFSSS algorithm can be implemented by adapting AFSSS (Algorithm 3) in a similar way.

5.5 Refinements

An abstraction refinement operation F maps an abstraction α to a new abstraction $F(\alpha)$ that is "closer" to the ground representation. In Chapter 4 we saw that progressive refinement of state abstractions shows promise as a design principle for new kinds of MCTS algorithms. Our primary objective in this chapter is to generalize abstraction refinement to other kinds of abstraction besides state abstraction in order to create new progressive abstract search algorithms. We begin by giving a slightly generalized definition of refinement.

Algorithm 9 Abstract Sparse Sampling with an Abstraction Diagram

procedure ADSS(G, s_0, C, d)
  Let $x_0 = \alpha_G(s_0)$
  $H(x_0) \leftarrow \{s_0\}$
  EXPAND(G, x_0, C, d)
  return $\arg\max_{b \in \beta_G(x_0)} \mathcal{Q}(x_0, b)$
procedure EXPAND(G, τ, C, d)
  Let $x = \mathbf{last}(\tau)$
  if τ is terminal **then**
    $\mathcal{Q}(\tau, b) \leftarrow 0$ for all $b \in \beta_G(x)$
    return
  for all $b \in \beta_G(x)$ **do**
    if $d = 0$ **then**
      $\mathcal{Q}(\tau, b) \leftarrow \bar{R}(\tau)$
      continue
    for C times **do**
      Let $h \sim \bar{\mu}(\tau, \cdot)$
      Let $h' \sim P(\cdot|h, b)$
      $n(h') \leftarrow n(h') + 1$
    Let $K \leftarrow [\bigcup_{h \in H} k(h, b)]/\alpha_G$
    for $H' \in K$ **do**
      Let h' be any element of H'
      Let $\tau' = \alpha_G(h')$
      $H(\tau') \leftarrow H'$
      EXPAND(G, τ', C, $d-1$)
    $\mathcal{Q}(\tau, b) \leftarrow \bar{R}(\tau) + \gamma \sum_{\tau' \in K(\tau,b)} \left[\frac{N(\tau')}{C} \max_{b' \in \beta_G(\mathbf{last}(\tau'))} \mathcal{Q}(\tau', b')\right]$

Definition 15. Let Π and Φ be policy sets. We say that Φ *refines* Π, denoted $\Phi \succeq \Pi$, if for all $s \in \mathcal{S}$,

$$\max_{\phi \in \Phi} V^{\phi}(s) \geq \max_{\pi \in \Pi} V^{\pi}(s). \tag{5.3}$$

This definition generalizes the intuitive idea that refinement expands the policy set.

Lemma 10. *If* $\Phi \supseteq \Pi$, *then* $\Phi \succeq \Pi$.

Proof. Since $\Pi \subseteq \Phi$, the optimal policy $\pi^* = \arg\max_{\pi \in \Pi}$ in Π is also an element of Φ. Thus $\max_{\phi \in \Phi} V^{\phi}(s) \geq V^{\pi^*}(s) = \max_{\pi \in \Pi} V^{\pi}(s)$ for all $s \in \mathcal{S}$. □

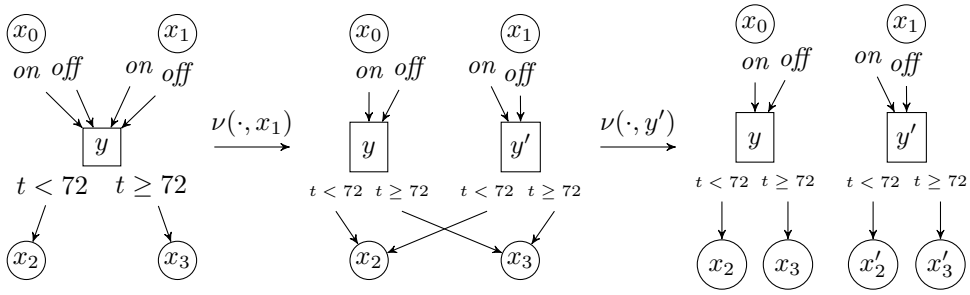


Figure 5.3: A sequence of two *unzip* operations applied to a fragment of an abstraction diagram.

The converse of Lemma 10 does not hold. For example, if π^* is the unique optimal policy in Π, then $\{\pi^*\} \not\supseteq \Pi \setminus \{\pi^*\}$ but clearly $\{\pi^*\} \succeq \Pi \setminus \{\pi^*\}$. As a more practical example, suppose that in a domain with continuous actions we simplify the action space by creating the base policy set $B = \{b\}$ where $b(s, \cdot) = \text{Uniform}(A)$ for some $A \subseteq \mathcal{A}$. We might now want to refine this abstraction by partitioning A into two parts C and D and adding the actions $c(s, \cdot) = \text{Uniform}(C)$ and $d(s, \cdot) = \text{Uniform}(D)$. Let $B' = \{c, d\}$ and let $U = B \cup B'$. Clearly $U \succeq B$ by Lemma 10. But it is also true that $B' \succeq B$, because either c or d must have a higher value than b. This is the motivation for Definition 15.

In the remainder of this section, we define some basic transformations of abstraction diagrams and show that they are refinements. We will use the following notation to describe operations on abstraction diagrams.

$$\begin{aligned}
\mathbf{in}(j) &= \{\langle i, \ell, j\rangle : i \in \mathcal{I}, \ell \in \mathcal{L}, \langle i, \ell, j\rangle \in \delta\} \\
\mathbf{out}(i) &= \{\langle i, \ell, j\rangle : j \in \mathcal{I}, \ell \in \mathcal{L}, \langle i, \ell, j\rangle \in \delta\} \\
\mathbf{cp.out}(i, i') &= \{\langle i', \ell, j\rangle : j \in \mathcal{I}, \ell \in \mathcal{L}, \langle i, \ell, j\rangle \in \delta\}.
\end{aligned}$$

The sets $\mathbf{in}(i)$ and $\mathbf{out}(i)$ contain the in- and out-edges of i, and the set $\mathbf{cp.out}(i, i')$ contains "copies" of the out-edges of i in which i is replaced by i'.

5.5.1 State Node Splitting

For a given action node y, the successors of y define a partition of the state space. Each successor x is identified with an equivalence class $S_x \in \mathcal{S}/\chi_y$. Splitting a successor

node x into two nodes x_1 and x_2 represents splitting the equivalence class S_x into two equivalence classes S_1 and S_2. The split operation is denoted by $\sigma(G, y, x, S_1, S_2)$, and it makes the following changes to the AD:

$$\begin{aligned} X &\to X \setminus \{x\} \cup \{x_1, x_2\} \\ \delta &\to \delta \setminus \mathbf{in}(x) \setminus \mathbf{out}(x) \cup \bigcup_{s \in S_1} \langle y, s, x_1 \rangle \cup \bigcup_{s \in S_2} \langle y, s, x_2 \rangle \\ &\quad \cup \mathbf{cp.out}(x, x_1) \cup \mathbf{cp.out}(x, x_2). \end{aligned}$$

Proposition 11. *Let $G = \langle \mathcal{I}, X, Y, \mathcal{L}, \delta, y_0 \rangle$ be an abstraction diagram, and let $G' = \sigma(G, y, x, S_1, S_2)$. Then $G' \succeq G$.*

Proof. Let x_1 and x_2 denote the two nodes that replace x in G'. By construction, $\mathbf{last}(\alpha_G(h)) = \mathbf{last}(\alpha_{G'}(h))$ except when $\mathbf{last}(\alpha_G(h)) = x$. If $\mathbf{last}(\alpha_G(h)) = x$, then either $\mathbf{last}(\alpha_{G'}(h)) = x_1$ or $\mathbf{last}(\alpha_{G'}(h)) = x_2$. A policy π that is consistent with G plays the same action in every h such that $\mathbf{last}(\alpha_G(h)) = x$ and therefore plays the same action in every h_1 s.t. $\mathbf{last}(\alpha_{G'}(h_1)) = x_1$ and every h_2 s.t. $\mathbf{last}(\alpha_{G'}(h_2)) = x_2$. Thus every policy that is consistent with G is also consistent with G', and we conclude $G' \succeq G$. □

Proposition 12. *Let $G = \langle \mathcal{I}, X, Y, \mathcal{L}, \delta, y_0 \rangle$ be an abstraction diagram, and let $G' = \sigma(G, y, x, S_1, S_2)$. If $|\beta(x)| > 1$, then $\Pi_{G'} \supset \Pi_G$.*

Proof. By the same reasoning as Proposition 11 we have $\Pi_{G'} \supseteq \Pi_G$. Consider some $\pi \in \Pi_G$. Let $b = \pi(h)$ where h is such that $\mathbf{last}(\alpha_G(h)) = x$. Let ϕ be a policy that is identical to π except that in all states h' s.t. $\mathbf{last}(\alpha_{G'}(h')) = x_2$, $\pi(h') = b'$ where $b' \in \beta(x)$ and $b' \neq b$. We have $\phi \in \Pi_{G'}$ but $\phi \notin \Pi_G$. Therefore $\Pi_{G'} \supset \Pi_G$. □

Note that splitting a choice node x in which $|\beta(x)| = 1$ does not expand the policy set because the single available action must be played in both resulting states.

5.5.2 Action Set Expansion

The action set expansion operator $\rho(G, x, b, y)$ makes a new action b available in node x. It adds an edge labeled with b to the AD,

$$\delta \to \delta \cup \{\langle x, b, y\rangle\}.$$

Clearly $\Pi_{\rho(G,x,b,y)} \supset \Pi_G$, since ρ adds a new action and removes nothing.

The action set expansion operation is sufficient to realize both temporal abstraction refinement and action unpruning. Recall that a temporal abstraction is represented by a chain of AD vertices in which there is only one action choice available in the state vertices. In Figure 5.1b, for example, the AD represents a temporal abstraction. Applying the action set expansion operators $\rho(\cdot, x_{01}, \mathit{on}, y_{01})$ and $\rho(\cdot, x_{11}, \mathit{off}, y_{11})$ to this AD produces a refined temporal abstraction in which the decision time scale has been reduced from 2 to 1.

5.5.3 Unzipping

Modifications such as the two just described alter the result of $\alpha_G(h)$ for any h that passes through the altered nodes. We might thus want to alter the structure of the AD to permit making localized changes that affect a smaller number of histories. The unzip operation $\nu(G, i)$ endows node i with its own exclusive copy of its successor nodes. Suppose the successor set of i is $\mathbf{succ}(i) = \{j_1, \ldots, j_k\}$. Let $\{j'_1, \ldots, j'_k\}$ be a set of new nodes of the same size and let $\mathbf{mv}(i, n) = \{\langle i, \ell, j'_n\rangle : \ell \in \mathcal{L}, \langle i, \ell, j_n\rangle \in \delta\}$ for $n \in \{1, \ldots, k\}$. Unzipping entails making the following changes to the AD.

$$\begin{aligned}
&\mathcal{I} \to \mathcal{I} \cup \{j'_1, \ldots, j'_k\} \\
&\delta \to \delta \setminus \mathbf{out}(i) \cup \bigcup_{n=1}^{k} \mathbf{mv}(i, n) \cup \bigcup_{n=1}^{k} \mathbf{cp.out}(j_n, j'_n)
\end{aligned}$$

The set X or Y is also updated as appropriate depending on the type of i.

The unzip operation does not alter the policy set, that is $\Pi_{\nu(G,i)} = \Pi_G$. However, it causes subsequent mutating operations to have a different effect. For example, in Figure 5.3, if we split the choice node x_2 in the left-most diagram, histories that pass

through x_0 or x_1 are affected by the split. After applying two unzip operations, we obtain the right-most diagram. Splitting x_2 in this diagram affects histories that pass through x_0, but not those that pass through x_1.

5.5.4 Composing Refinement Operations

Naturally, the refinement operations we have just described can be composed to create more complicated refinements. An important case of this is the tree structure-preserving state refinements used in the PARSS algorithm (Chapter 4). To perform PARSS-style refinements, we first apply the state splitting operator σ, and then recursively apply the unzip operator ν to restore the tree structure.

5.6 Progressive Abstract Tree Search

Progressive abstract tree search with an AD follows the same general pattern as the PARSS algorithm (Chapter 4). The main difference is that because the AD need not have the same structure as the search tree, making one refinement to the AD may affect more than one search tree node. To account for this possibility, we simply need to ensure that the UpdateTree procedure updates every affected tree node. Because UpdateTree recursively updates all descendants of the node it is called on, it is sufficient to call UpdateTree on some set of nodes A that contains an ancestor of every tree node affected by the refinement. This is the approach taken in Algorithm 11.

To fully realize the PATS framework, new refinement selection strategies need to be developed. State abstraction refinements can be chosen in a manner similar to how this is done in PARSS. New heuristics are needed for choosing between state splitting and action set expansion, and for choosing which actions to add when action expansion is chosen. Choosing an action to add is fundamentally different from choosing a state split because we have no information about the actions that are not present currently. Thus a heuristic for adding an action might consider the statistics of that action in other parts of the search tree, or even statistics from previous searches. We could also exploit domain knowledge to make the choice. For example, we might know that certain actions are more "specialized" than others and thus less likely to be good actions in an arbitrary state. One possible heuristic for choosing where to expand the action set is to look for a

state where the value abruptly decreases and add new actions in the states that precede it, the intuition being that the new actions might allow the agent to avoid the decrease in value.

5.7 Related Work

The abstraction diagram formalism is a small adaptation of the well-known idea of using a finite automaton to represent a structured policy. The POMDP solvers of Hansen [1998] and Meuleau et al. [1999], for example, search for policies represented as FAs. The HAM framework of Parr and Russell [1998] uses hierarchical FAs to encode temporal abstractions for reinforcement learning. The novelty of the progressive abstract tree search framework is in the interpretation of the FA as a policy constraint rather than as the policy itself. This allows the structure of the solution to be decoupled from the structure of the AD. The two are linked via the concept of policy consistency. The abstract MCTS algorithms that we consider build tree-shaped policies that are consistent with a given AD, but the AD need not also be tree-shaped. The MCTS component could be replaced by a different solution algorithm. For example, one could find an FA policy with the same topology as the AD using policy search methods, and such a policy would also be consistent with the AD.

The idea of incremental temporal refinement is inspired by the iterative refinement search algorithm of Neller [2002]. In this algorithm a search tree over durative actions is first constructed assuming that all actions are followed until the planning horizon T. Then, as time permits, further trees are built with decision intervals of $T/2$, $T/3$, and so on. Note, however, that these abstraction revisions are not necessarily refinements in our sense of the word. For example, the abstraction with $\Delta t = T/3$ is not a refinement of the abstraction with $\Delta t = T/2$ because each abstraction allows choices at times when the other abstraction does not. The abstraction with $\Delta t = T/4$, on the other hand, *is* a refinement of $\Delta t = T/2$.

Algorithm 10 Abstract Trajectory Sampling with an Abstraction Diagram

procedure ADTS(G, s_0)
  Let $x_0 = \alpha_G(s_0)$
  while time remains **do**
    VISIT(G, x_0, s_0)
  return $\arg\max_{b \in \beta_G(x_0)} \mathcal{Q}(x_0, b)$
procedure VISIT(G, τ, h)
  if τ is terminal **then**
    Let $v = 0$
  else if $n(\tau) = 0$ **then**
    Let $v =$ EVALUATE(h)
  else
    Let $b^* =$ SELECT(τ, G)
    Let $h' \sim P(\cdot|h, b^*)$
    Let $\tau' = \alpha_G(h')$
    Let $q =$ VISIT(τ', h', G)
    Let $v = R(h) + \gamma q$
    UPDATE(τ, b^*, v)
  $H(\tau) \leftarrow H(\tau) \cup \{h\}$
  $n(h) \leftarrow n(h) + 1$
  return v
procedure SELECT(G, τ)
  Let $x =$ **last**(τ)
  if $\exists b \in \beta_G(x) : M(\tau, b) = 0$ **then**
    return b
  Let $\mathcal{U}(\tau, b) = \mathcal{Q}(\tau, b) + c\sqrt{\frac{\log N(\tau)}{M(\tau,b)}}$
  return $\arg\max_{b \in \beta_G(x)} \mathcal{U}(\tau, b)$
procedure UPDATE(τ, b, v)
  $\mathcal{Q}(\tau, b) \leftarrow \mathcal{Q}(\tau, b) + \frac{v - \mathcal{Q}(\tau,b)}{M(\tau,b)}$

Algorithm 11 Progressive Abstract Sparse Sampling

1: **procedure** PASS(G, h_0, C, d)
2: Let $\mathcal{F} = \mathcal{F}_0(s_0, V_{\min}, V_{\max})$ (3.21)
3: ADFSSS(G, $\mathcal{F}$, C, d)
4: **while** time remains **do**
5: Refine the abstraction diagram G
6: Find a set of abstract tree state nodes A such that all tree nodes affected by the refinement are either in A or are a descendant of a node in A.
7: Rebuild the abstract subtrees rooted in A by applying the refined abstraction to the corresponding parts of the ground tree. Let A' be the nodes that replace the nodes in A in the rebuilt tree.
8: **for** $\langle H, a \rangle \in A' \times \mathcal{A}$ **do**
9: UpdateTree(H, a)
10: ADFSSS(G, $\mathcal{F}$, C, d)

Chapter 6: Applying Online Planning to Blackout Mitigation in Power Transmission Grids

6.1 Introduction

We now digress from the main topic of the thesis to examine an interesting application of OP algorithms to the problem of power grid control. Large failures of power transmission systems, such as the 2003 blackout in the Northeastern USA or the historic 2012 blackout in India, are often the result of a cascade of failures initiated by smaller events [Pahwa et al., 2013a]. The potential for localized failures to have such a dramatic widespread effect makes power transmission grids uniquely vulnerable to rare events such as natural disasters or terrorist attacks. Increasing the transmission grid's robustness to cascading failure is thus a pressing concern.

Robustness has been achieved historically through redundancy. Transmission grids are designed with sufficient redundancy to tolerate the loss of any single component. This property of the network is called $N-1$ security. Further robustness could be achieved through further redundancy, but this requires significant capital investment in equipment that will not be fully utilized under normal conditions.

Intelligent control of the system is an alternative approach to robustness. Prompt control response to a failure may allow the grid to recover and prevent a cascading failure [Amin and Wollenberg, 2005; Meier et al., 2014]. Yet emergency control is typically carried out largely by human operators taking manual actions at the time scale of minutes [Amin and Wollenberg, 2005]. A great deal of work has been done developing expert emergency control policies, based on strategies such as load shedding (e.g. [Pahwa et al., 2013a]) or islanding (e.g. [Pahwa et al., 2013b]), but these strategies tend to be fixed policies that are derived from heuristics or from solving optimization problems. Recent work has demonstrated the effectiveness of hybrid policies that combine several of these strategies [Meier et al., 2014]. We extend this line of work by fully automating the process of selecting among emergency response strategies using automated simulation-based online planning.

Validating an emergency control policy requires that we are able to simulate its effects in realistic emergency scenarios. Power system researchers commonly rely on simplified models of power grid dynamics when designing and testing control policies. One approach is to ignore the time-dependent aspects of the system's behavior to produce a quasi-steady state model; another approach neglects certain aspects of the AC power flow equations to produce the so-called "DC" power flow approximation [Frank and Rebennack, 2012]. While these simplified models have attractive numerical and computational properties, they do not capture all of the mechanisms of cascading failure [Song et al., 2016]. For example, only the full AC dynamical model captures the phenomenon of voltage collapse due to reactive power shortage.

The work presented in this chapter is a step toward fully automated emergency control planning based on high-fidelity simulation of the power grid's dynamics. We adapt the Cosmic power simulator [Song et al., 2016], which is specifically designed for simulating cascading failures, for use in simulation-based online planning. Cosmic simulates the full AC power flow dynamics of the system, and includes physical models of generators and their control systems, loads, and emergency protection relays. We apply policy rollout algorithms [Bertsekas and Castañon, 1999] to both deterministic and stochastic versions of emergency control problems in two standard benchmark transmission grid architectures, and compare them to typical examples of expert control policies. In deterministic simulations, we find that policy rollout outperforms our baseline policies in both grid architectures. In stochastic simulations, we find that the addition of randomness dramatically increases the computational cost of the simulation. For practical sample budgets, policy rollout has median performance similar to the best fixed policies, but with smaller variability. We conclude the chapter with an overview of outstanding obstacles to applications of automated planning to power system control on a large scale.

6.2 Background

The relevant background includes previous work on intelligent control of power systems, as well as Markov decision process (MDP) planning methods, specifically online planning.

6.2.1 Power Grid Simulation

The dynamics of the transmission grid are given by a system of differential algebraic equations [Song et al., 2016]. The state of the system at time t encompasses three vectors, $s(t) = \langle \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t) \rangle$. The vector $\mathbf{x}$ contains the dynamical variables, which evolve according to a non-linear differential equation

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(t, \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t)).$$

The vector $\mathbf{y}$ contains algebraic variables determined by

$$\mathbf{g}(t, \mathbf{x}, \mathbf{y}, \mathbf{z}) = 0.$$

Finally, $\mathbf{z}$ contains Boolean state variables that represent discrete events such as the triggering of protective relays. It is common for certain state variables such as the voltage magnitude to be specified in a *per-unit* system, so that a value of 1pu indicates that the variable is at its nominal value.

Solving or approximating these equations is a central task in the analysis of power systems. Models of power flow can be divided broadly into steady state models, which neglect the dynamics of the problem, and time-domain models. Time-domain models can be further subdivided into those that solve the full alternating current (AC) power flow, versus those that solve a linearized or "DC" power flow. In general, steady state models are simplest to compute, while full AC dynamical models allow for more detailed models of, for example, generator and load dynamics. The MATPOWER simulator [Zimmerman et al., 2011] is one example of a steady-state simulator. Examples of time-domain simulators include PowerWorld [PowerWorld Corporation, 2016], PSS/E [Siemens, 2015], RAMSES [Aristidou et al., 2014], and COSMIC [Song et al., 2016].

6.2.2 The Cosmic Power Simulator

Since we are concerned with controlling the power grid in an emergency situation, it is critical that we choose a model that properly captures both the behavior of the grid in highly perturbed states and the dynamics of cascading failure. We therefore use the Cosmic simulator [Song et al., 2016], which was designed specifically to model cascading

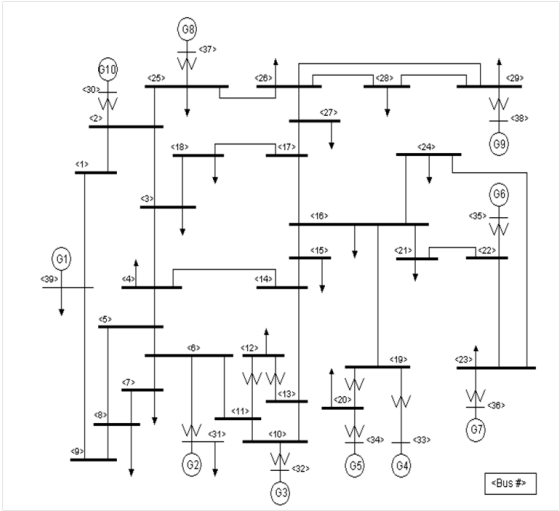


Figure 6.1: The IEEE39 grid topology [Athay et al., 1979]. The dark lines are *buses*, the lighter lines are *branches*, the arrows are *shunts* (which connect to loads), and the circles are *generators*.

failure accurately, for our experiments.

The power grid models used by COSMIC consist of 7 types of components: buses, branches, shunts, machines, exciters, governors, and generators [Song et al., 2016]. We will be concerned mainly with buses, branches, and shunts. We denote the set of buses by **bu**, the set of branches by **br**, and the set of shunts by **sh**. The power grid structure is an undirected graph (Figure 6.1). Buses are the nodes of the graph, and they may have associated shunts and generators. Branches are the edges of the graph, and each branch connects two buses. Shunts represent connections to loads, which draw power from the system. Power is supplied by the generators, which are driven by machines and controlled by exciters and governors. The COSMIC simulator implements dynamical models of all of these components in addition to the dynamics of AC power flow. Crucially for our purposes, COSMIC also implements discrete events such as the triggering of automatic protective relays. The cascading activation of protective systems is the actual mechanism behind cascading blackouts, and COSMIC is able to model this phenomenon.

Compared to steady state models, the distributions of cascading failure events produced by COSMIC are notably different [Song et al., 2016]. Specifically, large cascading failures are more common in the COSMIC model.

6.2.3 Emergency Control for Transmission Systems

There are two major strategies for stabilizing transmission grids in response to faults: load shedding and islanding [Meier et al., 2014]. Load shedding approaches respond to faults by disconnecting loads from the network. This reduces the flow of energy through the network, potentially preventing overloading of lines or voltage collapse due to excessive demand. Islanding approaches separate a failing network into multiple independent components in order to isolate the fault from the rest of the network. A cascading failure in an isolated component will not spread to the rest of the network.

Load shedding is a popular approach. Pahwa et al. [2013a] alone cite 14 examples of load shedding strategies. Proposed load shedding criteria tend to be heuristic. Some are based purely on domain knowledge (e.g., [Seethalekshmi et al., 2011]). Others are derived as solutions to various optimization problems, with objectives such as minimizing cost of shed load [Xu and Girgis, 2001; Aponte and Nelson, 2005; Faranda et al., 2007].

For islanding approaches, the key question is how the grid should be partitioned. Each connected component after islanding should be stable and self-sufficient, and there should be relatively few of them. Strategies for finding good partitions include clustering according to electrical distance [Cotilla-Sanchez et al., 2013], according to generator coherency in the islands [Yang et al., 2007; Sun et al., 2011], and according to generation-load imbalance in the resulting islands [Li et al., 2010]. Islanding schemes also differ in whether the islands to be formed are computed offline or online [Sun et al., 2011]. Islanding and load shedding can also be combined to create hybrid schemes [Dola and Chowdhury, 2006; Meier et al., 2014].

The proliferation of different load shedding and islanding strategies is evidence that no single approach is clearly best. This is one of the key advantages of online planning with policies as an approach to power grid control. Meier et al. [2014] demonstrated that load shedding and islanding policies both can be incorporated easily within this framework. If consensus emerges that certain heuristic load shedding or islanding strategies are superior, those policies can be added to the set of available policies from which the planner can choose.

6.3 Online Planning for Mitigating Blackouts

Our main contribution is an evaluation of simple online planning techniques for controlling power grids to prevent cascading outages. We formulate the control problem as an MDP in discrete time, and we keep the action space to a manageable size by using expert actions inspired by previous work on power grid control. Our work builds upon the work of Meier et al. [2014], which examined an "offline" policy switching approach to the power grid control problem.

6.3.1 MDP formulation

We construct an MDP in discrete time by adopting a fixed time step $\Delta t = 1$sec. We will denote the states at discrete times using subscripts $s_t = \langle \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t \rangle$. The actions make instantaneous changes to the algebraic variables $\mathbf{y}$. A successor state $s_{t+1} \sim P(\cdot|s_t, a)$ is obtained by applying action a in s_t and then integrating the dynamical equations from t to $t + \Delta t$ using Cosmic. We consider both deterministic and stochastic versions of P.

The choice of the action space $\mathcal{A}$ is an important design decision. We begin with two kinds of primitive control actions: open one branch, or decrease power consumption at one shunt by a specified percentage. In principle, these actions could be taken concurrently for every branch and shunt, giving an action space of size $O(2^{|\mathbf{br}|+|\mathbf{sh}|})$. Clearly this primitive action space is too large for exhaustive search. We reduce the action space by defining smaller sets of expert actions. These fall into the two categories of load shedding and islanding [Meier et al., 2014]. Load shedding strategies reduce demand by disconnecting some of the loads from the network. Islanding strategies partition the network into multiple connected components so that failures are isolated from the rest of the grid.

6.3.2 Optimization Objective

A control policy π running from initial state s_0 generates a distribution over trajectories through the state space. Among the state variables $s_t = \langle \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t \rangle$ are the variables $p_{i,t}$ giving the real power flow through shunt i at time t in state s_t. We define the optimization objective in terms of the real power flow. If there is stochasticity in the model, then the real power flow is a random variable, which we denote with a capital

letter $P_{i,t}$. The total power delivered at time t is the sum of the power at each shunt, $P_t = \sum_{i \in \mathbf{sh}} P_{i,t}$.

A control policy should satisfy as much power demand as possible, while avoiding large outages. Naturally, outages cause large losses of satisfied demand, so a natural figure of merit for a controller π is the expected total energy delivered,

$$E(\pi, s_0) = \mathbb{E}_{P,\pi}\big[\sum_{t=0}^{H} P_t\big].$$

This basic objective could be supplemented by constraints on acceptable operating ranges, by attaching different weights to different loads, or in many other ways. Since we are interested in cascading failure scenarios, we might also want to minimize the probability of total blackout,

$$B(\pi, s_0) = \mathbb{P}\Big(\min_t P_t = 0\Big).$$

We define our reward function as a modification of the total energy criterion,

$$R(s_t) = \sum_{i \in \mathbf{sh}} \min(p_{i,t}, p^*_{i,t}), \tag{6.1}$$

where $p^*_{i,t}$ is the real power demand at shunt i at time t when the voltage at shunt i is 1pu. Taking the minimum prevents the agent from being rewarded for artificially increasing power demand by driving the voltage away from 1pu.

6.3.3 Stochasticity

Although the dynamics of the transmission grid are deterministic, stochasticity is present in real control problems due to uncertainty in future demand, the possibility of equipment failure, and other factors. In the stochastic version of our experiments, we incorporated randomness from two sources:

Load fluctuations – For each load i, the real and reactive power demands P^i and Q^i

follow a bounded random walk with Gaussian increments,

$$\begin{aligned} P^i_{t+1} &\sim \mathbf{proj}_{[\ell,u]}(P^i_t + \mathcal{N}(0,\sigma^2)), \\ Q^i_{t+1} &\sim \mathbf{proj}_{[\ell,u]}(Q^i_t + \mathcal{N}(0,\sigma^2)), \end{aligned} \tag{6.2}$$

where $\mathbf{proj}_{[\ell,u]}(x) = \max(\ell, \min(u, x))$. We set $\ell = 0.8$ and $u = 1.2$ for all problems. We selected the variance σ^2 to produce a qualitative match of the whole-grid power fluctuation to the empirical results of Karlsson and Hill [1994]. This resulted in $\sigma = 0.01$ for the IEEE39 grid architecture that we consider in our experiments.

Random delays in relay activation – Protection relays in real systems may not operate exactly as expected. They could be mis-programmed or mis-calibrated, or the mechanical devices they control may not operate perfectly due to environmental conditions. A full model of the many possible failure modes is beyond the scope of this work, but we attempt to capture some of the variability that results by adding a random delay between the triggering of a protective relay and the actual application of the protective function that it controls. For example, consider a relay that disconnects a load when some parameter x exceeds a critical value x^*. If x first exceeds x^* at time t, then the load will be disconnected after a random delay,

$$t_{\text{disconnect}} \sim t_{\text{trigger}} + \text{Exp}(\lambda), \tag{6.3}$$

where λ is the exponential rate parameter. We set $\lambda = \frac{1}{2}$ in our experiments.

6.3.4 Baseline Policies

We evaluated a set of simple fixed policies to establish a baseline of performance. These included policies based on load shedding and on islanding.

Isolate : The Isolate policy immediately isolates any zones in which a failure occurred from the rest of the grid. We consider a failure to have occurred in a zone z if the bus at either end of the failed branch b was in z.

ShedGlobal : The ShedGlobal(p) policy immediately sheds a fixed proportion $p \in [0, 1]$ of the load at every shunt simultaneously.

HLS : We also designed a more sophisticated expert policy that we call "hysteretic load shedding" and denote by HLS(p, ℓ, u, δ). HLS prescribes load shedding near a bus $b \in \mathbf{bu}$ if the voltage magnitude $|V^b|$ at that bus dropped below ℓ at least δ units of time in the past and has stayed below u since then. More formally, for each bus $b \in \mathbf{bu}$, if there exists a time in the past $t' < t - \delta$ such that both 1) $|V_t^b| < \ell$ and 2) $|V_\tau^b| < u$ for all $\tau \in [t', t]$, then HLS prescribes a load shedding action that affects bus b. Since loads are associated with shunts and not all buses are connected directly to a shunt with a load, HLS finds the shunt with an active load that is nearest to each bus b in the sense of *electrical distance* [Cotilla-Sanchez et al., 2013]. At every unique shunt identified in this manner, a proportion p of the per-unit load is shed.

The load shedding policies assume that any proportion of the load can be shed. This assumption is not entirely realistic, since load shedding is implemented by opening discrete circuit breakers and thus some proportions may not be feasible. However, it should hold approximately since the loads on a transmission grid actually represent entire distribution grids serving many customers, and we can approximate shedding a proportion p of the load by disconnecting a proportion p of the customers. Analogous policies could be defined that respect any constraints on load shedding.

6.3.5 Policy Rollout

Due to the complexity of the problem domain and the large cost of generating samples, we focused on simple planning algorithms that can perform well with small sample budgets. The policy rollout algorithm [Bertsekas and Castañon, 1999] is one of the simplest. Policy rollout implements a control policy π_{pr} by estimating the action-value function Q^π of a rollout policy π and acting greedily according to this estimate,

$$\pi_{\text{pr}}(s) = \arg\max_{a \in \mathcal{A}} \hat{Q}^\pi(s, a). \tag{6.4}$$

Let $\tau = s_0 a_0 s_1 \dots s_H$ denote a trajectory of length H starting from state s_0. The probability of generating τ under policy π is given by

$$P^\pi(\tau) = \prod_{t=0}^{H-1} \mathbb{1}_{a_t=\pi(s_t)} P(s_{t+1}|s_t, a_t).$$

The return of a trajectory is the total reward received,

$$\rho(\tau) = \sum_{i=0}^{H} R(s_i).$$

The estimated Q-function $\hat{Q}_k^\pi$ after sampling k trajectories is

$$\hat{Q}_k^\pi(s,a) = \frac{1}{n_k(a)} \sum_{i=1}^{n_k(a)} \rho(\tau_a^i), \tag{6.5}$$

where $n_k(a)$ is the number of times that action a has been sampled and τ_a^i is the ith sampled trajectory that begins with sa. The policy rollout objective is to minimize

$$\mathcal{R}_{\text{pr}}(s) = \max_{a \in \mathcal{A}} Q^\pi(s,a) - Q^\pi(s, \pi_{\text{pr}}(s)). \tag{6.6}$$

Since the space of primitive actions is prohibitively large, we make the action search space $\mathcal{A}$ a small set of "expert" actions. These actions are of one of three parameterized types:

ShedGlobal(p) : Sheds a proportion $p \in [0,1]$ of the load at every shunt simultaneously.

ShedZone(z, p) : Sheds a proportion $p \in [0,1]$ of the load simultaneously at every shunt in zone z.

Island(z) : Opens (disconnects) all branches between a bus in zone z and a bus in any zone $z' \neq z$.

These actions are similar in spirit to the expert policies we use in policy switching. In

Domain	$N-2$ faults	$\lvert\mathbf{bu}\rvert$	$\lvert\mathbf{br}\rvert$	$\lvert\mathbf{sh}\rvert$	Blackouts
IEEE39	972	39	46	19	520 (53.5%)
RTS96	7140	73	120	51	2023 (28.3%)

Domain	V (kWh) mean (σ)	R_H (W) mean (σ)	t_{blackout} (sec) mean (σ)
IEEE39	303 (220)	2785 (2998)	183 (132)
RTS96	639 (247)	6434 (4046)	257 (99)

Table 6.1: Characteristics of the test domains under deterministic dynamics. The values V, R_H, and t_{blackout} are calculated for the uncontrolled grid.

our experiments we use the action set

$$\mathcal{A} = \{\textsc{ShedGlobal}(p = 0.05), \textsc{ShedZone}(z,\ p = 0.05), \textsc{Island}(z)\ :\ z \in \mathbf{zones}\}.$$

Policy rollout also requires one or more rollout policies. Due to the high cost of sampling trajectories from the Cosmic simulator, we use a single deterministic rollout to evaluate each action. We use this strategy in both the deterministic and stochastic versions of the problem. Thus the decisions made by policy rollout in the stochastic problem are based on simulating without stochsticity starting from the current state. We use the DoNothing policy for the rollout policy, and the rollout procedure follows it for a fixed number of steps d.

6.4 Experiments

Our experiments evaluated the performance of online planning approaches as well as fixed baseline control policies in comparison to performance without control.

6.4.1 Transmission Grid Architectures

We used two standard transmission grid architectures for our experiments. The first, called IEEE39 [Athay et al., 1979], was used for the earlier work of Meier et al. [2014]. We use the 4 zones defined for this problem by Meier et al. [2014]. The second architecture,

called RTS96 [Grigg et al., 1999], consists of a common subgrid replicated three times with different interconnections. We define 3 zones for this problem corresponding to these 3 subgrids.

6.4.2 Identifying failure cases

We first evaluated the effect of all $N-2$ contingencies when no control policy is used in deterministic simulation. We then excluded those $N-2$ contingencies for which a total blackout occurred immediately after the initial $N-2$ event before the agent has a chance to act. We call the remaining contingencies the *recoverable contingencies*. In the IEEE39 domain, we included all recoverable contingencies for which any load was lost under the DoNothing policy. This resulted in a set of 534 contingencies. For RTS96, due to the overall larger number of contingencies, we included only those recoverable contingencies that led to a total blackout. This resulted in a set of 1764 contingencies.

6.4.3 Common Random Numbers

To reduce variance in our stochastic experiment, we used identical random number streams during simulation when evaluating the policies (but not for making action choices within the policies). More precisely, when being evaluated on the ith trajectory, all of the control policies see the same sequence of random loads and the same sequence of random relay delays. This implies that control policies that choose the same sequence of actions will see exactly the same outcomes.

6.4.4 Baseline Policies

We selected parameterizations of the baseline ShedGlobal and HLS policies based on a small pilot experiment in the stochastic version of the IEEE39 domain. In all of our experiments we use the three baseline policies Isolate, ShedGlobal($p = 0.1$), and HLS($p = 0.05$, $\ell = 0.95$, $u = 0.98$, $\delta = 5$). In the context of the experiments, we will refer to these parameterizations as Isolate, ShedGlobal, and HLS.

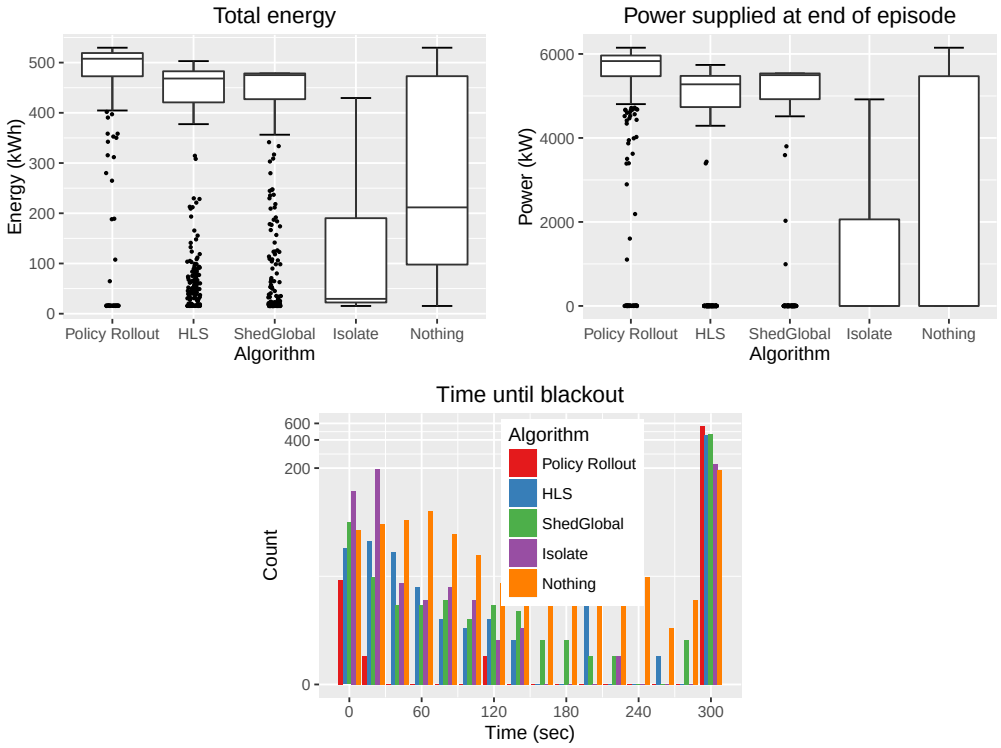


Figure 6.2: Comparison of policy rollout to baseline policies in IEEE39.

6.5 Results

The solution algorithms are evaluated using three criteria: total energy supplied, power supplied during the last time step of the simulation, and time until blackout. The total energy supplied is the primary figure of merit and the one most closely related to the optimization objective for the OP algorithms. The last step power criterion allows us to see how much demand has been lost over the course of the episode. If a blackout has occurred, the last step power will be equal to 0. The time until blackout criterion gauges the agent's ability to prolong stable operation of the grid, which is desireable even if a blackout eventually occurs. Results for the first two criteria are presented as box plots in which the "whiskers" show ± 1.5 times the inter-quartile range (IQR). Data points outside this range are shown as points in the charts and are considered outliers. Results for the time until blackout criterion are presented as histograms showing the number of

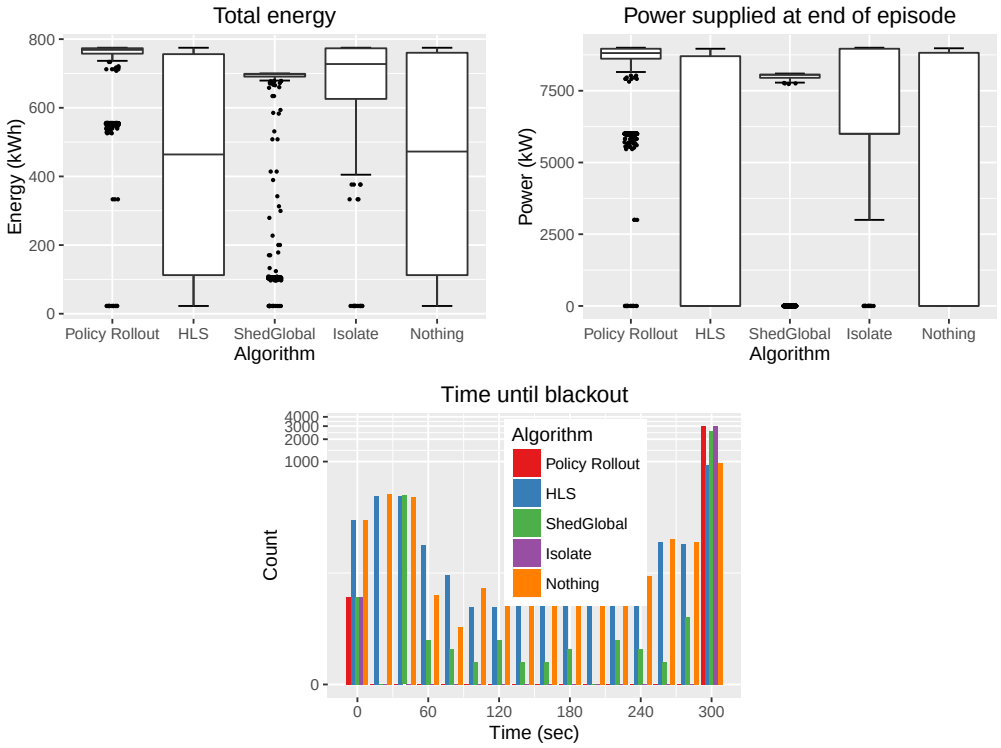


Figure 6.3: Comparison of policy rollout to baseline policies in RTS96.

failure cases for each range of times. Note the logarithmic scale on the vertical axis in these histograms.

In IEEE39 (Figure 6.2), policy rollout and both of the fixed load shedding policies performed well in comparison to the DoNothing policy. Policy rollout was the best algorithm overall, especially in terms of the number of total blackouts. The Isolate policy performed poorly, but was able to prevent a total blackout in a small number of cases.

In RTS96 (Figure 6.3), the pattern of results is very different. Here Isolate performs extremely well, often successfully responding to the contingency without losing any load. The ShedGlobal policy also prevents most blackouts, but apparently sheds more load than necessary. Policy rollout performed as well as Isolate but with smaller variability, and there is not much room for further improvement. HLS was clearly the worst policy; under HLS, 542 contingencies progressed to total blackouts, compared to

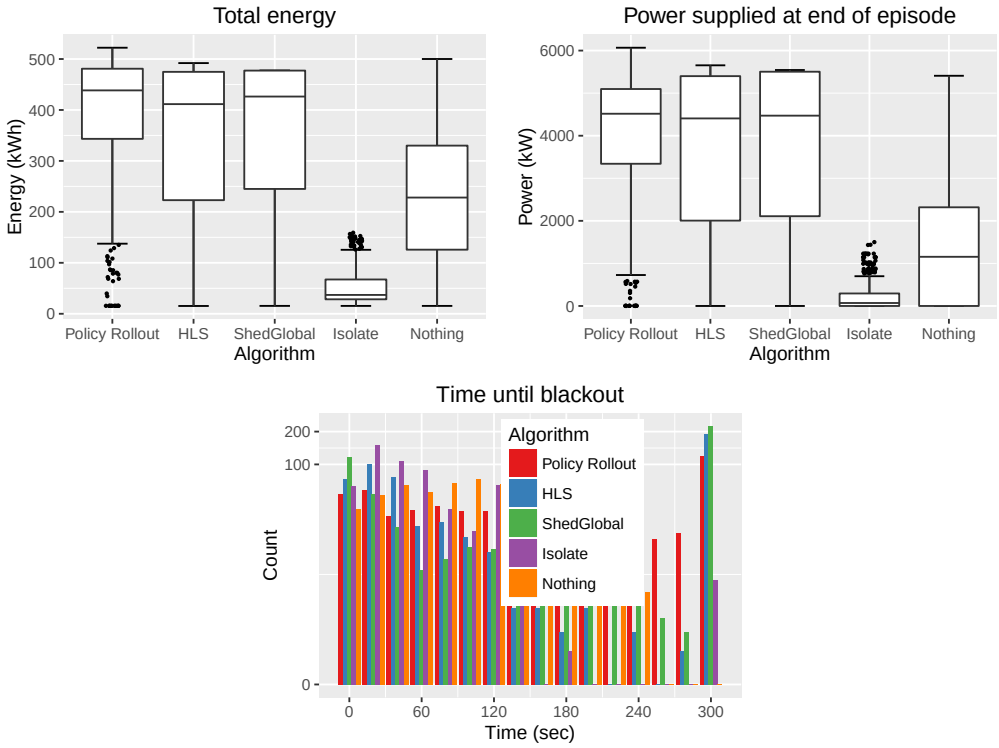


Figure 6.4: Comparison of policy rollout to baseline policies in the stochastic version of IEEE39. Note that there are 10 times as many data points in this experiment compared to deterministic IEEE39, because each failure case was replicated 10 times with different random numbers.

only 3 contingencies under each of the other three policies. It seems that the zones defined for the RTS96 architecture are better able to tolerate being isolated than the zones in IEEE39. It is unclear why the performance of HLS is so different between the two cases. Possibly the buses that tend to depart from their nominal voltages are not good candidates for load shedding in RTS96.

In the stochastic version of IEEE39 (Figure 6.4), ISOLATE continued to perform poorly while HLS and SHEDGLOBAL had comparable performance. Policy rollout achieves similar median energy and last-step power as these two policies, but with less variability. Strangely, however, policy rollout was less likely to completely avoid a blackout, as seen in the time-until-blackout chart (Figure 6.4). This could be due in part to the perfor-

mance criterion used by policy rollout (6.1), which does not directly penalize the agent for allowing a blackout. It could also be due to the use of deterministic simulations in policy rollout. Since half of all random futures will have greater power demand than the deterministic scenario, the estimates based on deterministic simulation may be too optimistic, resulting in insufficient corrective action.

6.6 Discussion and Future Work

Our results in simulation demonstrate the promise of OP algorithms for emergency control of transmission grids. Nevertheless, advances in several areas are required before similar approaches can be implemented in real systems. The most pressing need is for faster simulation of emergency scenarios. Greater exploitation of parallel computation is vital, such as applied in the RAMSES simulator of Aristidou et al. [2014]. Parallel execution of simulation trajectories is also easily implemented in OP algorithms, and this would provide an easy and effective speedup. It would also be worthwhile to experiment with the use of simplified simulations for decision-making. Although high-fidelity simulation is necessary for evaluating decision-making algorithms for this problem, it may be possible for OP algorithms to make good decisions using simpler models. We have explored this idea somewhat already, when we used a deterministic model for simulation even though the real model is stochastic.

Further work is also needed to create realistic models of the sensing and control systems of the transmission grid. The state variables of real networks are not fully observable, due to the limitations of the sensors deployed in the network. Incomplete knowledge of the system state means that OP algorithms would need to sample several possible "current states" from the state distribution and plan for all of them. This increases the computational burden significantly.

The high cost of simulation forced us to choose very simple OP algorithms for our experiments in order to reduce the simulation requirements as much as possible. If the efficiency of the simulator can be significantly improved, it will become feasible to apply more sophisticated OP algorithms such as the abstract MCTS algorithms we have described in this thesis to the power grid control problem.

Chapter 7: Conclusion and Future Work

This thesis has presented a new approach to Monte Carlo tree search based on progressive abstraction refinement. We first analyzed state abstraction in the tree search setting and derived a regret bound for decision making using abstract tree search with a class of state aggregation abstractions. We then presented the Progressive Abstraction Refinement for Sparse Sampling algorithm, which is the prototypical example of the progressive refinement framework for MCTS. PARSS was demonstrated experimentally to be superior to tree search with fixed abstractions – including the ground representation – for a range of problem domains. We then introduced abstraction diagrams, which unify several types of abstraction besides state abstraction in a single formalism, and used ADs to generalize the basic idea of PARSS to create the progressive abstract tree search framework. The ultimate goal of this work is to allow MCTS algorithms to scale up to problems with large state and action spaces.

In the final chapter, we applied online planning techniques to the problem of controlling an electrical transmission grid during abnormal conditions to mitigate the possibility of cascading failure and blackout. This is the type of problem for which progressive abstract tree search is intended – it has very large state and action spaces and the effects of actions unfold over relatively long time scales. Unfortunately the limitations of state-of-the-art simulators for this domain forced us to use simpler algorithms that require fewer samples to make a decision. Although even these simple algorithms were better than fixed expert policies, there is likely plenty of room for improvement.

There are several interesting directions for further work on progressive abstract tree search. One interesting possibility is to learn to control the abstraction refinement process to achieve better performance. This represents a new way of *learning to plan*, complimenting methods such as learning the leaf evaluation function. The space of refinement strategies would need to be describe in a formalism that is amenable to learning, such as by creating a stochastic grammar that generates refinement sequences and optimizing its parameters. Feedback for the learner could come from measuring the change in value estimates obtained from search with the old and new abstractions.

A second possibility is to use raw data about the effectiveness of different abstractions gathered from the search trees to inform the construction of a single, fixed abstraction, which could then be used for learning a reactive policy. Long-term learning of the sort that humans do seems to involve compiling behaviors that once required conscious thought into reflexive behaviors that are executed almost automatically. This could be realized in an agent architecture that builds up a library of reactive policies for solving common tasks and uses these policies as the primitive actions for higher-level planning. Examining the abstractions produced by progressive abstract tree search algorithms as they solve these subtasks could provide valuable guidance for choosing the correct abstractions to enable subtask policies to be learned effectively.

Bibliography

Amin, S. and Wollenberg, B. (2005). Toward a smart grid: Power delivery for the 21st century. *IEEE Power and Energy Magazine*, 3(5):34–41.

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

Aponte, E. and Nelson, J. (2005). Time optimal load shedding for distributed power systems. *IEEE Transactions on Power Systems*, 21(1):269–277.

Aristidou, P., Fabozzi, D., and Van Cutsem, T. (2014). Dynamic simulation of large-scale power systems using a parallel schur-complement-based decomposition method. *IEEE Transactions on Parallel and Distributed Systems*, 25(10):2561–2570.

Athay, T., Podmore, R., and Virmani, S. (1979). A practical method for the direct analysis of transient stability. *IEEE Transactions on Power Apparatus and Systems*, 98(2):573–584. See also: http://publish.illinois.edu/smartergrid/ieee-39-bus-system/ (Accessed 18-May-2015).

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

Balla, R.-K. and 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., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138.

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

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

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

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

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

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

Chang, H. S., Givan, R., and 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. and Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Cotilla-Sanchez, E., Hines, P., Barrows, C., Blumsack, S., and Patel, M. (2013). Multiattribute partitioning of power networks based on electrical distance. *IEEE Transactions on Power Systems*, 28(4):4978–4987.

Culberson, J. C. and 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.

Dola, H. and Chowdhury, B. (2006). Intentional islanding and adaptive load shedding to avoid cascading outages. In *IEEE Power Engineering Society General Meeting*.

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

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

Faranda, R., Pievatolo, A., and Tironi, E. (2007). Load shedding: A new proposal. *IEEE Transactions on Power Systems*, 22(4):2086–2093.

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

Frank, S. and Rebennack, S. (2012). A primer on optimal power flow: Theory, formulation, and practical examples. Technical Report 14, Colorado School of Mines.

Gabillon, V., Ghavamzadeh, M., and Scherrer, B. (2013). Approximate dynamic programming finally performs well in the game of Tetris. In *Advances in neural information processing systems (NIPS)*.

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

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

Grigg, C., Wong, P., Albrecht, P., Allan, R., Bhavaraju, M., Billinton, R., Chen, Q., Fong, C., Haddad, S., Kuruganty, S., et al. (1999). The IEEE reliability test system-1996: A report prepared by the reliability test system task force of the application of probability methods subcommittee. *IEEE Transactions on Power Systems*, 14(3):1010–1020.

Guerin, J. T., Hanna, J. P., Ferland, L., Mattei, N., and 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., and 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*, pages 193–209. Springer.

Helmert, M., Haslum, P., and 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., and Dietterich, T. (2014). State aggregation in Monte Carlo tree search. In *AAAI Conference on Artificial Intelligence*.

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

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

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

Karlsson, D. and Hill, D. J. (1994). Modelling and identification of nonlinear dynamic loads in power systems. *IEEE Transactions on Power Systems*, 9(1):157–166.

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

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

Keller, T. and Helmert, M. (2013). Trial-based heuristic tree search for finite horizon MDPs. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

King, B., Fern, A., and 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. and 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, J., Liu, C.-C., and Schneider, K. (2010). Controlled partitioning of a power network considering real and reactive power balance. *IEEE Transactions on Smart Grid*, 1(3):261–269.

Li, L., Walsh, T. J., and 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*. PhD thesis, University of Rochester.

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

Meier, R., Cotilla-Sanchez, E., and Fern, A. (2014). A policy switching approach to consolidating load shedding and islanding protection schemes. In *Power Systems Computation Conference (PSCC)*.

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

Moore, A. W. and 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., and Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404.

Neller, T. W. (2002). Iterative-refinement for action timing discretization. In *AAAI Conference on Artificial Intelligence*.

Pahwa, S., Scoglio, C., Das, S., and Schulz, N. (2013a). Load-shedding strategies for preventing cascading failures in power grid. *Electric Power Components and Systems*, 41:879–895.

Pahwa, S., Youssef, M., Schumm, P., Scoglio, C., and Schulz, N. (2013b). Optimal intentional islanding to enhance the robustness of power grid networks. *Physica A: Statistical Mechanics and its Applications*, 392(17):3741–3754.

Parr, R. and Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems (NIPS)*.

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

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

PowerWorld Corporation (2016). Power world simulator. [Online; accessed 21-Nov-2016].

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

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

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

Seethalekshmi, K., Singh, S. N., and Srivastava, S. C. (2011). A synchrophasor assisted frequency and voltage stability based load shedding scheme for self-healing of power system. *IEEE Transactions on Smart Grid*, 2(2):221–230.

Siemens (2015). Psse: Power transmission system planning. [Online; accessed 18-May-2015].

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.

Song, J., Cotilla-Sanchez, E., Ghanavati, G., and Hines, P. D. (2016). Dynamic modeling of cascading failure in power systems. *IEEE Transactions on Power Systems*, 31(3):2085–2095.

Sun, K., Hur, K., and Zhang, P. (2011). A new unified scheme for controlled power system separation using synchronized phasor measurements. *IEEE Transactions on Power Systems*, 26(3):1544–1554.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge University Press.

Sutton, R. S., Precup, D., and 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. and 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., and Littman, M. L. (2010). Integrating sample-based planning and model-based reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

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

Xu, D. and Girgis, A. A. (2001). Optimal load shedding strategy in power systems with distributed generation. In *IEEE Power Engineering Society Winter Meeting*, volume 2, pages 788–793.

Yang, B., Vittal, V., Heydt, G. T., and Sen, A. (2007). A novel slow coherency based graph theoretic islanding strategy. In *IEEE Power Engineering Society General Meeting*.

Zimmerman, R. D., Murillo-Sánchez, C. E., and Thomas, R. J. (2011). MATPOWER: Steady-state operations, planning and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19.

APPENDICES

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 $d = \max_{h \in \mathcal{H}} \mathbf{len}(h)$ (which may be infinite). 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 $E(\{s_0\}, a) = |\mathcal{Q}^*(\{s_0\}, a) - Q^*(s_0, a)|$.

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

$$E(H, a) = \left| \mathcal{Q}^*_\alpha(H, a) - \sum_{h \in H} \mu(H, h) Q^*(h, a) \right| = \left| \mathcal{R}_\mu(H) - \sum_{h \in H} \mu(H, h) R(H) \right| = 0. \tag{A.1}$$

Inductive step We now consider interior states $H \in \overline{\Omega}$ and assume the inductive hypothesis $E(H', a') \leq \beta_\gamma(k)(p + \delta q)$ 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 $E(H, a)$ (A.1), 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 $E(H,a) \leq \gamma(E_Q + E_\chi)$, 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

$$\sum_{h' \in H'} \mu(H',h') \max_{a' \in \mathcal{A}} Q^*(h',a') - \max_{a' \in \mathcal{A}} \sum_{h' \in H'} \mu(H',h') Q^*(h',a') \leq p. \tag{A.2}$$

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

$$\Big| \max_{a' \in \mathcal{A}} \mathcal{Q}^*_\alpha(H',a') - \max_{a' \in \mathcal{A}} \sum_{h' \in H'} \mu(H',h') Q^*(h',a') \Big| \leq \max_{a' \in \mathcal{A}} E(H',a'), \tag{A.3}$$

because of the general fact that $|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)|$ for real-valued functions f and g on the same domain. Combining (A.2) and (A.3) with the triangle inequality, we have

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

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 (3.6),

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

We can then express E_χ as

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

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

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

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 $V^*(h) - v(H) \leq q$ 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,

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

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,

$$E_Q + E_\chi \leq \beta_\gamma(k)(p + \delta q) + p + \delta q = (1 + \beta_\gamma(k))(p + \delta q). \tag{A.4}$$

Since $E(H, a) \leq \gamma(E_Q + E_\chi)$, we multiply (A.4) 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 $H \in \mathcal{H}/\chi$ and all $a \in \mathcal{A}$. This completes the inductive argument.

□

A.2 Proof of Proposition 2

Proposition 2. *Let $T = \langle \mathcal{H}, \mathcal{A}, P, R, \gamma, s_0 \rangle$ be a history MDP and let χ be a (p, q)-consistent history equivalence relation on $\mathcal{H}$. 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*

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\beta_\gamma(d)(\lambda + p + \delta q),$$

where δ is the divergence (3.7) 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 (3.18),

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

where $\mathbf{dom}(\bar{\mu}) \subseteq \mathcal{H}/\chi$ is the subset of the abstract history set on which $\bar{\mu}$ is defined.

The error due to finite sampling in AbstractSS is given by

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

Following the proof of Kearns et al. [2002], we introduce the quantity

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

The difference $|\mathcal{Q}^*_\alpha(H,a) - \mathcal{U}^*(H,a)|$ captures the error 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 an expectation and an empirical average of C iid samples. We can thus apply Hoeffding's inequality (Hoeffding [1963], Theorem 2) to

conclude that

$$\mathbb{P}\Big(\Big|\mathcal{Q}^*_\alpha(H,a) - \mathcal{U}^*(H,a)\Big| \le \lambda \le \frac{\lambda}{\gamma}\Big) \le 1 - 2e^{-2\lambda^2 C/(V^d_{\max})^2}. \tag{A.5}$$

Using this result, we decompose the sampling error in terms of U^* as

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

Now we need to bound the difference between $\mathcal{V}^*$ and the actual estimate $\hat{\mathcal{V}}$ that we obtain from search. We bound this difference by bounding the difference in action values,

$$F(H,a) = \Big|\mathcal{Q}^*_\alpha(H,a) - \hat{\mathcal{Q}}(H,a)\Big|.$$

We argue by induction that $F(H_0,a) \le \beta_\gamma(d)\lambda$.

Base case Consider an arbitrary leaf node $H \in \Omega$. Since H has no successors, $F(H,a) = 0 \le \lambda$ for all $a \in \mathcal{A}$.

Inductive step Now consider an arbitrary interior node $H \in \overline{\Omega}$ and action $a \in \mathcal{A}$ and assume the inductive hypothesis $F(H',a') \le \beta_\gamma(k)\lambda$ for all $H' \in K(H,a)$ and $a' \in \mathcal{A}$. We have

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

This completes the inductive argument.

To obtain a probability bound in the root node, we require that the bound in (A.5) 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

$$\left|\hat{\mathcal{Q}}(H_0, a) - \mathcal{Q}^*_\alpha(H_0, a)\right| \leq \beta_\gamma(d)\lambda \tag{A.6}$$

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 (A.6) with Theorem 1. We have

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

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

$$V^*(s_0) - Q^*(s_0, a^*) \leq 2\beta_\gamma(d)(\lambda + p + \delta q),$$

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 augmented with the special action ω and an abstraction $\alpha = \langle \chi, \mu^* \rangle$ of T composed of equivalence relation χ and the corresponding optimal weight function μ^*. Let H' be a random variable $H' \sim \mathcal{P}^\xi_\alpha(\cdot|h_0)$ for a fixed abstract policy $\xi \in \Pi(T/\alpha)$, and let h' be a random variable such that $h' \sim P^{\downarrow\xi}(\cdot|h_0)$. Then the random variable $[h']_\chi$ is equal in distribution to H'.*

Proof. Recall that the probability of an abstract history $H = H_0 a_0 H_1 \ldots a_d H_d$ under abstraction α and abstract sampling policy ξ is

$$\mathcal{P}^\xi_\alpha(H|h_0) = \xi(H, \omega) \prod_{t=0}^{d-1} \xi(H_t, a_t) \mathcal{P}_\mu(H_{t+1}|H_t, a_t),$$

and the probability of a ground history $h = h_0 a_1 h_1 \dots a_d h_d$ under sampling policy π is

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

We need to show that for $h \sim P^{\downarrow\xi}$ and $H \sim \mathcal{P}^\xi_\alpha$, $[h]_\chi =^d H$. This is equivalent to the condition that $\sum_{h \in H} P^{\downarrow\xi}(\cdot|h_0) = \mathcal{P}^\xi_\alpha(\cdot|h_0)$.

The proof is simpler when we consider a slightly different pair of distributions,

$$\mathcal{W}^\xi_\alpha(H|h_0) = \prod_{t=0}^{d-1} \xi(H_t, a_t) \mathcal{P}_\mu(H_{t+1}|H_t, a_t),$$
$$W^\pi(h|h_0) = \prod_{t=0}^{d-1} \pi(h_t, a_t) P(h_{t+1}|h_t, a_t).$$

We call these the *prefix distributions* because they give the probability of generating a history that *starts with* H or h. They are related to the history distributions by

$$\mathcal{P}^\xi_\alpha(H|h_0) = \xi(H, \omega) \mathcal{W}^\xi_\alpha(H|h_0),$$
$$P^\pi(h|h_0) = \pi(h, \omega) W^\pi(h|h_0).$$

By assumption, the ground policy is $\downarrow\xi$. For this choice of policy, we have that for all $h \in H$, $\downarrow\xi(h, \omega) = \xi([h]_\chi, \omega) = \xi(H, \omega)$. Thus $\sum_{h \in H} P^{\downarrow\xi}(h) = \mathcal{P}^\xi_\alpha(H)$ if and only if $\sum_{h \in H} W^{\downarrow\xi}(h) = \mathcal{W}^\xi_\alpha(H)$. We prove the latter fact by induction on the length of the history prefix.

Base case Let $H_0 = \{h_0\}$ be the initial state. Since H_0 is a singleton we have trivially that $\mathcal{W}^\xi_\alpha(H_0) = 1 = W^{\downarrow\xi}(h_0)$.

Inductive step Consider a history $H = H_0 a_1 H_1 \dots a_k H_k$. Assume the inductive hypothesis $\sum_{h_{k-1} \in H_{k-1}} W^{\downarrow\xi}(h_{k-1}|h_0) = \mathcal{W}^\xi_\alpha(H_{k-1}|h_0)$. Again using the definition of $\downarrow\xi$,

we have

$$\sum_{h_k \in H_k} W^{\downarrow\xi}(h_k) = \sum_{h_k \in H_k} \sum_{h_{k-1} \in H_{k-1}} W^{\downarrow\xi}(h_{k-1}) \xi([h_{k-1}]_\chi, a_k) P(h_k | h_{k-1}, a_k)$$
$$= \xi(H_{k-1}, a_k) \sum_{h_{k-1} \in H_{k-1}} W^{\downarrow\xi}(h_{k-1}) \sum_{h_k \in H_k} P(h_k | h_{k-1}, a_k).$$

Now we want to isolate a factor of $\sum_{h_{k-1} \in H_{k-1}} W^{\downarrow\xi}(h_{k-1})$ so that we can use the inductive hypothesis. We do this by multiplying by 1:

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

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

$$\mu^*(H, h) = \mathbb{P}(h|H) = \frac{\mathbb{P}(H, h)}{\mathbb{P}(H)} = \mathbb{1}_{h \in H} \frac{W^{\downarrow\xi}(h)}{\sum_{g \in H} W^{\downarrow\xi}(g)}.$$

This completes the inductive argument, and we conclude that for $H' \sim \mathcal{P}^\xi_\alpha(\cdot|h_0)$ and $h' \sim P^{\downarrow\xi}(\cdot|h_0)$, we have $[h']_\chi =^d H'$, where $\alpha = \langle \chi, \mu^* \rangle$.

□