Optimal Rewards versus Leaf-Evaluation Heuristics in Planning Agents

Jonathan Sorg
Computer Science and Engineering
University of Michigan
jdsorg@umich.edu

Satinder Singh
Computer Science and Engineering
University of Michigan
baveja@umich.edu

Richard L. Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Abstract

Planning agents often lack the computational resources needed to build full planning trees for their environments. Agent designers commonly overcome this finite-horizon approximation by applying an evaluation function at the leaf-states of the planning tree. Recent work has proposed an alternative approach for overcoming computational constraints on agent design: modify the reward function. In this work, we compare this reward design approach to the common leaf-evaluation heuristic approach for improving planning agents. We show that in many agents, the reward design approach strictly subsumes the leaf-evaluation approach, i.e., there exists a reward function for every leaf-evaluation heuristic that leads to equivalent behavior, but the converse is not true. We demonstrate that this generality leads to improved performance when an agent makes approximations in addition to the finite-horizon approximation. As part of our contribution, we extend PGRD, an online reward design algorithm, to develop reward design algorithms for Sparse Sampling and UCT, two algorithms capable of planning in large state spaces.

Introduction

In this work, we consider model-based planning agents which do not have sufficient computational resources (time, memory, or both) to build full planning trees. Thus, estimates of the expected sum of rewards, or return, from the current state will only be approximate. Agent designers often build in heuristics for at least partially overcoming these limitations. For example, in the case of limited planning depth, the *Leaf-Evaluation Heuristic* (LEH) (Shannon 1950) adds a heuristic value to the estimated return at the leaf states of the planning tree. If the expected return obtainable after the leaf state were known, then it could be used as the leaf-state return estimate to compensate for the missing subtree below the leaf. As this is not known in practice, some method must be employed to estimate the leaf-state values.

In recent work, Sorg et al. (2010b) proposed an alternative approach to mitigating computational limitations in planning agents: modify the agent's reward function. They defined the Optimal Reward Problem (ORP)—choose the reward function for a given agent architecture that maximizes the reward received by the agent designer. An optimal reward function can compensate for general computational limitations on agents, including limits on the depth of a planning tree. However, the optimal reward approach has not yet been compared to the more-common leaf-evaluation heuristic approach.

One method for modifying reward functions, Potential-Based Reward Shaping (PBRS) (Ng, Russell, and Harada 1999; Asmuth, Littman, and Zinkov 2008), modifies the agent's reward function in a particular constrained form. An optimal policy under a potential-based reward function is guaranteed to be optimal under the original reward function. However, a limited agent may exhibit different behavior when using a potential-based reward than when using the original.

In this work, we compare the ORP approach—which does not restrict its class of reward functions—to the LEH and PBRS approaches. The central claim of this work is that there exist environments and agents in which the ORP approach outperforms both the LEH and PBRS approaches. In fact, we show that the LEH approach and the PBRS approach are effectively equivalent: for every leaf-evaluation heuristic, there exists a potential-based reward which produces equivalent behavior, and vice versa. Furthermore, we show that the effects achievable through reward function design subsume those of the set of leaf-evaluation heuristics (and potential-based rewards). The key aspect of this generality is *path dependence*. Reward function design is able to modify the agent's return function in a way that depends on entire state-action trajectories, while a leaf-evaluation heuristic only applies to states at the ends of trajectories. As a result, in agents which make approximations in shallow portions of the tree, optimal reward functions can compensate for errors that the LEH and PBRS approaches cannot compensate for.

Of course, the nature of this benefit depends heavily on the planning algorithm chosen and we cannot consider all planning algorithms in this paper. We focus on UCT (Kocsis and Szepesvári 2006), a state-of-the art planning algorithm that has been used in many applications (Gelly and Silver 2008; Finnsson and Björnsson 2008). UCT makes several approximations that can be mitigated through solving the ORP: (1) it cannot generate unbounded-length trajectories, causing the *finite-horizon bias*; (2) it uses Monte-Carlo sampling to approximate return values, resulting in *sampling variance*; and (3) its early sample trajectories evaluate a suboptimal policy, a problem we refer to as *search control bias*.

To analyze the differing abilities of leaf-evaluation functions and optimal rewards to compensate for these approximations, we additionally consider two planning algorithms which share some of these approximations: Full Finite-Horizon Planning (FFHP), which only suffers from the finite-horizon, and Sparse Sampling (Kearns, Mansour, and Ng 1999), which makes the finite-horizon and sampling approximations. We extend an online reward function optimization method developed for the FFHP algorithm, PGRD (Sorg, Singh, and Lewis 2010a), to develop reward function optimization methods for Sparse Sampling and UCT, enabling reward functions to be optimized in agents whose computational complexity is independent of the size of the state space. We use these methods to demonstrate our claims.

Optimal Heuristics

Formally, we consider discrete-time environments M with a finite number of states $s \in S$ and actions $a \in A$. The dynamics are governed by a state-transition function $P(s'|s,a)$ that defines a distribution over next state s' conditioned on current state s and action a. The agent G has a reward function, $R(s,a,s')$ that maps state s, action a, and next-state s' tuples to scalar values and an accumulation function U which defines how rewards are summed over time. Together they define an agent's return function U_R. For example, the discounted return function is $U_R(h) = \lim_{H\to\infty} \sum_{i=0}^{H} \gamma^i R(s_i, a_i, s_{i+1})$ for some state-action trajectory $h = s_0, a_0, s_1, a_1, ...s_\infty$ and discount factor $\gamma \in [0,1)$. A policy μ maps state s to distributions over actions $\mu(a|s)$. A value function V^μ is the expected return of following the policy μ given that the agent starts in a particular state: $V^\mu(s) = \mathbb{E}[U_R(h)|\mu, s_0 = s]$. An optimal policy μ^* maximizes expected return from the current state, i.e. $\mu^* = \arg\max_\mu V^\mu(s)$. The optimal value function is the value function of an optimal policy $V^* = V^{\mu^*}$. Local planning agents repeatedly estimate these quantities from a current state by building planning trees using a given or learned model $\hat{P}(s'|s,a)$, and selecting the action corresponding to a locally optimal policy.

Optimal Reward Functions

In the standard view of Reinforcement Learning (RL), the agent uses the same reward function as the designer. Optimal reward theory allows for the agent's reward function to be specified separately from the agent designer's. The designer's goals are specified via an *objective reward function* R_O and an accumulation function U which defines how they are summed over time. Together, they define the *objective return function* U_{R_O}. The designer's goal is to design an agent whose behavior maximizes the expected objective return.

The optimal reward function R^* is defined by the optimal reward problem:

$$R^* = \arg\max_{R\in\mathcal{R}} \mathbb{E}\left[U_{R_O}(h)|G(U_R), M\right],$$

where the distribution over h is the result of agent G planning with return function U_R acting in environment M. The set $\mathcal{R}$ is the set of all mappings from state, action, next-state tuple to real number: $S \times A \times S \to \mathbb{R}$. In words, an optimal reward function leads to an agent with the greatest expected objective return for the designer.

Optimal Leaf-Evaluation Heuristics

Let $h[i,j] = s_i, a_i, s_{i+1}, a_{i+1}, ..., s_j$ denote the subsequence of trajectory h from time i to j. Given some finite horizon H, we define the truncated return function $U_{R_O}(h[0,H]) = \sum_{i=0}^{H-1} \gamma^i R_O(s_i, a_i, s_{i+1})$. The truncated return ignores rewards after the horizon H.

The leaf-evaluation heuristic compensates for this finite-horizon bias in an agent's return estimates. A leaf-evaluation heuristic is a function $L(s)$ defined over the state space $s \in S$. When planning with truncated trajectories, an agent adds L to its return estimates as a function of the final state in the trajectory. Define the leaf-evaluation return to be $U^L_{R_O}(h[0,H]) = U_{R_O}(h[0,H]) + \gamma^H L(s_H)$.

V^* LEH. The value function of an optimal policy, V^*, is the traditional notion of the ideal leaf-evaluation heuristic (hereafter called the V^*-leaf-evaluation heuristic). This corresponds to setting $L(s) = V^*(s)\,\forall s$. Using this function completely corrects for the finite-horizon bias when computing full expectations over the truncated return. We use the V^*-LEH as a baseline in our empirical work.

Optimal LEH. However, if the agent cannot exactly compute the full expectation over the truncated trajectories, the V^*-leaf-evaluation heuristic may not be optimal. This can happen if the agent makes approximations in addition to the finite-horizon approximation, as is usual in planning methods applicable to real domains. An optimal LEH is defined similarly to an optimal reward function:

$$L^* = \arg\max_{L\in\mathcal{L}} \mathbb{E}\left[U_{R_O}(h)\middle|G(U^L_{R_O}), M\right],$$

where the distribution over h is the result of agent G planning with leaf-evaluation return function $U^L_{R_O}$ acting in environment M, and $\mathcal{L}$ is the set of all mappings from S to $\mathbb{R}$. An optimal LEH leads to an agent with the best expected objective return for the designer. Below, we present a method to find approximately optimal leaf-evaluation heuristics.

Reward Design Subsumes Leaf-Heuristics

In this section, we prove that for any leaf-evaluation heuristic, there exists a reward function that produces identical behavior. Our analysis is most closely related to the work of Asmuth et al. (2008) on potential-based reward shaping in model-based agents. Specifically, applying an evaluation function at the leaf is equivalent to adding a potential-based shaping function to the reward, minus a constant offset depending on start state.

Define a potential-based reward to be $R_L(s,a,s') = R_O(s,a,s') - L(s) + \gamma L(s')$ for some function of state $L(s)$. Then, the following theorem holds:

Lemma 1. *For any leaf-evaluation heuristic L, there exists an effectively equivalent potential-based reward R_L and vice versa. Specifically, for any history h and for all finite horizons H, the leaf-evaluation return $U^L_{R_O}(h[0,H])$ and the potential-based return $U_{R_L}(h[0,H])$ differ by a constant dependent only on start state.*

Proof. The additive reward bonus creates a telescoping sum:

$$
\begin{aligned}
&U_{R_L}(h[0,H]) \\
&= \sum_{t=0}^{H-1} \gamma^t R_O(s_t, a_t, s_{t+1}) + \sum_{t=1}^{H} \gamma^t L(s_t) - \sum_{t=0}^{H-1} \gamma^t L(s_t) \\
&= U_{R_O}(h[0,H]) + \gamma^H L(s_H) - L(s_0) \\
&= U^L_{R_O}(h[0,H]) - L(s_0)
\end{aligned}
$$

□

The constant offset $L(s_0)$ will not affect an agent's preference between outcomes, but this does not guarantee that a particular approximate planning algorithm will return identical policies when planning with U_{R_L} versus planning with $U^L_{R_O}$. However, for the algorithms used in this paper, it can easily be shown that planning with U_{R_L} and $U^L_{R_O}$ do indeed lead to identical behavior.

Lemma 1 shows that the set of reward functions effectively subsumes the set of leaf-evaluation functions, in the sense that they are capable of specifying at least as broad a set of preference orderings over policies. In Theorem 1 below, we prove that the relationship is strict; there exist reward functions that express preference orderings over trajectories which cannot be expressed via a leaf-evaluation heuristic.

Theorem 1. *The set of potential-based return functions is a strict subset of the set of reward-based return functions.*

Proof. Lemma 1 proves non-strict subsumption. Here, we prove strictness. Consider two trajectories: $h_1 = (s_1, a_1, s_2, a_2, s_3)$ and $h_2 = (s_1, a_1, s_4, a_2, s_3)$, where the subscripts here indicate identity, not time. Let $U_{R_O}(h_1) = U_{R_O}(h_2) = 0$. Then, $U_{R_L}(h_1) = U_{R_L}(h_2) = -L(s_1) + \gamma^2 L(s_3)$ regardless of L. However, in general, $U_R(h_1) = R(s_1, a_1, s_2) + \gamma R(s_2, a_2, s_3) \neq U_R(h_2) = R(s_1, a_1, s_4) + \gamma R(s_4, a_2, s_3)$. □

The proof highlights the difference between general rewards and both leaf-evaluation heuristics and potential-based rewards. General rewards allow for *path dependent* modifications of objective return—they depend on the entire state-action sequence in the trajectory—while leaf-evaluation functions and potential-based rewards only modify the return as a function of the leaf state.

Optimal Rewards Outperform Leaf-Values

In this section, we discuss how the extra generality afforded by reward functions can be beneficial in UCT by first discussing two other planning algorithms: Full Finite-Horizon Planning and Sparse Sampling. This progression will enable us to isolate different approximations made by UCT. For each approximation, we discuss whether the generality of reward design can lead to better performance over leaf-evaluation heuristics.

Full Finite-Horizon Planning. FFHP is a simple example of a computationally limited planning agent. Each time step, FFHP computes a finite-horizon approximation of the full planning problem. Specifically, Q^μ_H is the expected truncated return of policy μ given that the agent follows μ after taking action a from state s: $Q^\mu_H(s,a) = \mathbb{E}[U_R(h[0,H])|\mu, s_0 = s, a_0 = a]$. FFHP computes the optimal action-value function under the truncated return $Q^*_H(s,a) = \max_\mu Q^\mu_H(s,a) \, \forall a \in A$ and acts greedily with respect to $Q^*_H(s,a)$. FFHP does this by building a finite-horizon, balanced planning tree rooted at the current state. For each state-action node, the tree contains a child node for all possible successor states and an entry describing the probability of reaching each state under the model. FFHP computes the true expectation of the truncated return; therefore, as discussed above, the V^*-leaf-evaluation heuristic perfectly corrects for the approximation. Thus, the optimal reward function R^* will do no better than the optimal leaf-evaluation heuristic L^* (or potential-based reward) in FFHP. In other words optimal reward functions do not outperform leaf-evaluation heuristics on an agent which suffers only from the finite-horizon bias.

Sparse Sampling. FFHP is intractable in practice because the number of next states from each node can be large or infinite and because the size of the tree is exponential in the horizon. To overcome the first concern, Kearns, Mansour, and Ng (1999) proposed Sparse Sampling, which approximates a FFHP planning tree by sampling the next-state dynamics. It builds a balanced, approximate FFHP planning tree by sampling C next-states for each state-action node.

The sample tree built by Sparse Sampling causes two types of errors not present in FFHP when estimating the action-value function at the root state: (1) the iterated sampling of C next states in building the tree introduces a variance into the action-value estimates; and (2) in the backing up of values from the leaves of the tree to the root, the max operation over noisy estimates introduces a positive bias into the action-value estimates. Crucially these errors occur throughout the tree, and indeed increasing depth compounds both sources of errors. Thus, leaf-evaluation functions, which are functions of the leaf state only, are less able to mitigate these errors relative to designed reward functions, which can have effects at all depths in the tree. To appreciate this difference, consider that the V^*-leaf-evaluation heuristic, in particular, is entirely about accounting for the expected value of the missing subtree *below* the leaf state. While effective at mitigating the finite-horizon bias common to FFHP and Sparse Sampling, it has little or no correlation to the factors inducing the additional errors in the tree *above* the leaf state. Our empirical results below confirm that not only can optimal rewards be more effective than the V^*-LEH, but they can also be more effective than optimal leaf-evaluations heuristics (L^*)—that is, even when allowing leaf evaluation heuristics to mitigate errors introduced by the sampling approximations to the greatest extent possible.

UCT. Even Sparse Sampling, however, is infeasible on practical problems, because the expense of building the tree is exponential in the horizon. UCT (Kocsis and Szepesvári 2006) does not build a balanced tree. Instead, it samples N, H-length trajectories from the current state, constructing an imbalanced tree from the sampled trajectories. This results in the challenge of choosing sample actions, a problem known as search control. UCT chooses actions using statistics from previous sample trajectories. It estimates the value of a state, action, depth tuple (s, a, d) as the average return obtained after experiencing the tuple:

$Q(s,a,d) = \sum_{i=1}^{N} \frac{I_i(s,a,d)}{n(s,a,d)} \sum_{k=d}^{H-1} \gamma^{k-d} R(s_k^i, a_k^i, s_{k+1}^i; \theta)$, where $n(s,a,d)$ is the number of times tuple (s,a,d) has been sampled, $I_i(s,a,d)$ is 1 if tuple (s,a,d) is in trajectory i and 0 otherwise, s_k^i is the k^{th} state in the i^{th} trajectory, and a_k^i is the k^{th} action in the i^{th} trajectory. UCT selects search control actions using Q and an additional bonus which encourages balanced sampling. Specifically, the agent chooses $a^* = \arg\max_a Q(s,a,d) + c\sqrt{\frac{\log n(s,d)}{n(s,a,d)}}$, where $n(s,d) = \sum_a n(s,a,d)$. With an appropriate choice of c, UCT's Q estimates at the root converge to FFHP's Q estimates as the number of trajectories N increases.

For finite N, UCT shares the finite depth and sample variance errors from Sparse Sampling, but it also introduces another source of error in the estimation of action values at the root. During initial trajectory generation, the algorithm incorporates sub-optimal actions into its evaluation of the optimal policy, producing an error in the value estimates that we refer to as the *search control bias*. This error in UCT occurs throughout the trajectories. Thus, we expect that optimal reward functions, which can modify the calculation of return based on all the state-action pairs along a trajectory, will be better at mitigating this error relative to leaf evaluation heuristics, which can modify the calculation of return based only on the leaf state of the trajectory. Our experimental results below confirm this intuition.

Reward Design in Sampling Planners

The agents we present in our experiments plan using the algorithms above, but execute actions according to a soft-max distribution at the root of the tree. Specifically, $\mu(a|s) \stackrel{\text{def}}{=} \frac{e^{\tau Q(s,a)}}{\sum_b e^{\tau Q(s,b)}}$, where τ is a temperature parameter controlling how deterministically the agent selects the action with the highest score and $Q(s,a)$ is the action-value computed by the planning tree rooted at state s. Using stochastic policies allows us to compute gradients. Also, from this section onward, we assume that the reward function is parameterized $R(s,a,s';\theta)$ via some set of parameters $\theta \in \Theta$.

Policy Gradient for Reward Design (PGRD) (Sorg, Singh, and Lewis 2010a) views these reward function parameters as policy parameters of the above stochastic policy, for the special case where Q in the soft-max above is calculated via FFHP. It optimizes the parameters using the standard policy gradient algorithm OLPOMDP (Bartlett and Baxter 2000). At each time step, OLPOMDP requires (1) the agent's distribution over actions in the current state $\mu(a|s_t)$, from which it samples an action a_t; (2) the gradient of the agent's policy with respect to the policy parameters $\nabla_\theta \mu(a_t|s_t)$ (3) the resulting next state, sampled from the environment; and (4) the resulting designer's objective reward. It uses these to optimize the agent's policy parameters θ while following the agent's policy μ. In PGRD, the agent's policy is computed via FFHP and the parameters are the reward function parameters. Because the policy is a soft-max on Q, the gradient is given by $\nabla_\theta \mu(a|s) = \tau \cdot \mu(a|s)[\nabla_\theta Q(s,a) - \sum_{b \in A} \mu(b|s) \nabla_\theta Q(s,b)]$. Although the Q function in FFHP is not differentiable everywhere (due to the max operation), the subgradient is computable and resembles planning itself (Neu and Szepesvári 2007). Its use in PGRD has been demonstrated to work well in practice for FFHP (Sorg, Singh, and Lewis 2010a). In order to extend PGRD to work for Sparse Sampling and UCT, we must develop similar subgradient computations for their planning algorithms. We refer to the resulting reward design algorithms as PGRD-SS and PGRD-UCT, respectively. In this section, we present these methods.

Reward Improvement in Sparse Sampling. Unlike in FFHP, there are two sources of stochasticity in Sparse Sampling. In addition to the stochasticity created by the soft-max action selection, the tree sampling creates stochasticity in the computed Q values. We denote the distribution over Q-values $p(Q|s;\theta)$. Combining these two sources of randomness produces the full distribution over next action: $\mu(a|s;\theta) = \sum_Q \mu(a|s,Q) p(Q|s;\theta)$. We do not have an analytical formula which computes $p(Q|s;\theta)$. Estimating it is prohibitively expensive, requiring many sample trees.

This problem can be solved by noting that the distribution over planning trees is independent of the reward parameters. Thus, the Sparse Sampling algorithm can be viewed as being composed of two stages. First, the algorithm samples a planning tree, then the reward function is applied to that tree to compute the Q-values. Let T denote a sampled planning tree. The stochastic policy can now be broken up as: $\mu(a|s;\theta) = \sum_Q \mu(a|Q) \sum_T p(Q|s,T;\theta) p(T|s)$.

We simplify this equation in two steps. First, note that the Q-function is deterministic given a fixed tree and return function. Second, note that the tree distribution, though dependent on root state, is independent of the reward distribution. In other words, the distribution over trees can be handled as if it were a property of the environment. Viewed this way, the agent acts according to $\mu(a|\langle s_t, T_t\rangle;\theta_t)$, where $\langle s_t, T_t\rangle$ is the joint environment–planning-tree state. The agent no longer needs to marginalize over planning trees; however, this adds variance to the stochastic gradient ascent by effectively adding variance to the environment.

PGRD-SS acts according to Sparse Sampling and computes the gradient using this simplification. It amounts to applying the FFHP subgradient computation to the sample tree. It can be shown that using this gradient computation results in a convergent method, as we state in the following theorem (proof omitted for lack of space). Generally, it will converge to a local optimum in the parameters.

Theorem 2. *PGRD-SS converges to a stable equilibrium, as in Theorem 3 in Sorg et al. (2010a).*

Reward Improvement in UCT. Like Sparse Sampling, PGRD-UCT has two sources of stochasticity when selecting actions. Unlike Sparse Sampling, however, the search control decisions depend on the reward function. Our approximate solution ignores the dependence on the reward function in the tree generation. This approximation is motivated by the proof of convergence of UCT. With an unlimited number of sample trajectories, the Q estimates converge to the FFHP estimates; in the limit, the search-control effects are negligible.

After this assumption is made, we can use the same trick used in Sparse Sampling—assume that the remain-

ing stochasticity is conditioned solely on the environment. The resulting gradient computation is $\nabla_\theta Q(s,a,0) = \sum_{i=1}^{N} \frac{I_i(s,a,0)}{n(s,a,0)} \sum_{k=0}^{H-1} \gamma^k \nabla_\theta R(s_k^i, a_k^i, s_{k+1}^i; \theta)$. Notice that the computation of the gradient resembles the computation of the Q estimates. These two computations can be performed simultaneously, as the trajectories are generated.

Experiments

In this section, we empirically validate our claim that optimal rewards outperform optimal leaf-evaluation heuristics (and therefore potential-based rewards) in some agents. The experiments, along with the observations above, support our claim that the optimal reward function R^* is often better than the optimal leaf-evaluation heuristic L^*. They also support the conclusion that the optimal leaf-evaluation heuristic L^* is sometimes better than the V^*-leaf-evaluation heuristic.

Reward Spaces. Recall that using a potential-based reward is equivalent to using a leaf-evaluation heuristic. Thus, by optimizing both the potential-based rewards and more general rewards, the PGRD algorithms can be used to compare approximately optimal leaf-evaluation heuristics and approximately optimal reward functions. The general reward space we use in our experiments, $R(s,a,s';\theta)$, is a look-up table over (s,a,s'). The potential-based reward space is $R_L(s,a,s';\theta) = R_O(s,a,s') - L(s;\theta) + \gamma L(s';\theta)$, where $L(s;\theta)$ is a look-up table over s.

Marble Maze Environment. We empirically validate our claims in the Marble Maze environment in Figure 1(a), a grid world based on a commonly-used set of noisy dynamics (Asmuth, Littman, and Zinkov 2008). The agent starts in the location marked S. The designer receives a reward of 1 when the agent reaches the goal state, marked G, after which the agent is transported back to the start. Otherwise, the designer receives 0 reward. If the agent moves into a pit (gray square), it is transported back to the start after 1 time step. The agent has movement actions corresponding to the four cardinal directions. Each action has a 0.2 probability of failing; if the action fails, the agent instead moves in a random direction perpendicular to the intended direction, each with probability 0.5. In each trial, an agent is placed in a different instantiation of the Marble Maze environment, sampled uniformly randomly from all solvable 5x5 Marble Maze worlds with 8 pits and the start and goal locations in opposite corners.

The properties of Marble Maze highlight the deficiencies in Sparse Sampling and UCT. When a sample trajectory falls into a pit, it does great damage to the value of a trajectory. This creates a large return variance and a large search control bias when the agent's search control policy is poor.

Experiment Methodology. For each planning algorithm, we plot PGRD learning curves for each reward space using various planning depths. For each reward-space and for each planning-depth, we optimized PGRD's learning rate parameter to a constant from a discrete set on a logarithmic scale in the range $[10^{-6}, 10^{-3}]$. Each learning curve uses the learning rate which yields the maximum expected cumulative objective reward. We used PGRD parameters $\beta = 0.9$ and $\lambda = 0$ and the soft-max temperature $\tau = 100$. The reward parameters for both general rewards and potential-based rewards are initialized such that the functions are equal to the objective reward R_O. The agents learned the transition models online using empirical counts: $\hat{P}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$, where $n(\cdot)$ is the number of times the agent has experienced each tuple.

We used an additional baseline algorithm for learning potential-based reward functions, SARSA(λ), for comparison. The SARSA(λ) agent learns a Q-function from the objective rewards and uses the maximum action-value in each state as the $L(s)$ value in the potential-based reward definition. The agent takes a random action with probability ϵ and the planning algorithm's action otherwise, planning with the learned potential-based reward function. We optimized over ϵ in the set $\{0, 0.01, 0.1, 0.2\}$. We optimized over learning rates from discrete values in the range $[0.01, 0.2]$ and used $\lambda = 0.95$. As opposed to the PGRD agent with potential-based rewards, which attempts to find the equivalent of the L^*-LEH, the SARSA(λ) agent learns a value function, and uses this as the LEH. In other words, the SARSA(λ) agent attempts to find the V^*-LEH. Finally, we also tested the V^*-leaf-evaluation heuristic, found via value iteration.

Sparse Sampling Experiment. In this section, we show that rewards are better than leaf-evaluation heuristics at improving performance in agents with sampling approximation errors. Figure 1(b) presents learning curves for PGRD-SS agents with a fixed sample count of C=2. We varied over planning depths 1 through 4, though we show only depths 1 and 3 for readability. The leftmost points on the curve represent agents which use the objective reward. The remainder of the curve plots the objective reward per step obtained by PGRD-SS during optimization, averaged over the previous 10^4 time steps. At the final points in the curve we plot 95% confidence intervals. Each curve is an average of 100 trials.

The general reward functions (top 2 curves) outperform leaf-evaluation heuristics at each depth, both for PGRD-optimized evaluation heuristics (bottom 2 curves) and SARSA-optimized evaluation heuristics (middle 2 curves). Although it might be difficult to see given the small graph size, the leftmost points on the graph demonstrate that increasing the planning depth improves the agent's performance for agents that use only the objective reward R_O. This is because increasing the planning depth reduces the finite-horizon approximation bias. Also as predicted, increasing the planning depth hinders the leaf-value heuristic's ability to compensate for the finite-horizon approximation. Thus, there is a crossover effect. As a leaf-evaluation heuristic becomes more accurate, the optimal planning depth decreases. This can be seen by the fact that the Depth-1 V^*-LEH agent outperforms the Depth-3 V^*-LEH agent. In contrast, the ability of optimal reward functions to induce path-dependent modifications of the objective return allows the reward-design agents to converge to near-optimal behavior (optimal behavior not shown) at each planning depth.

UCT Experiment. In this section, we demonstrate that optimal rewards are better than leaf-evaluation heuristics at mitigating the search-control bias. UCT has both the search control bias and sampling approximation errors (in addition to the finite-horizon bias). To separate the confounding sam-

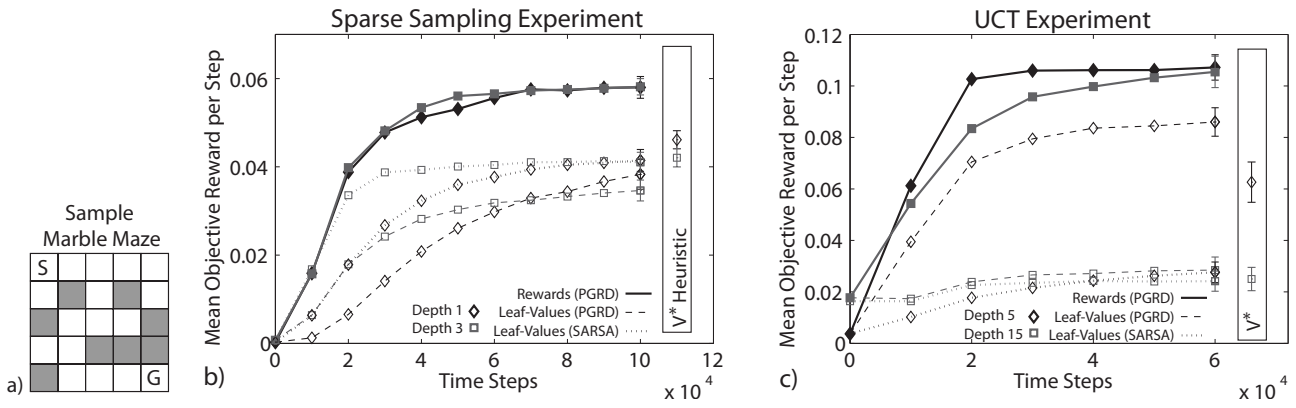

Figure 1: Marble Maze Experiment. Solid-lines/filled markers indicate general reward functions. Dashed or Dotted-lines/hollow markers indicate leaf-evaluation functions. Inset on right: performance of V^*-leaf-evaluation functions, indicated by marker.

pling error explanation from the search-control bias explanation for the reward's improvement over leaf-evaluation heuristics, we perform this experiment in a deterministic version of the Marble Maze—actions have a 0 probability of failing. We tested UCT agents with planning horizons 1 through 15, $N = 100$, and $c = 1$. All other experiment conditions were identical to the Sparse Sampling experiment.

Figure 1(c) presents the results for the UCT agents with depths 5 and 15. Each learning curve plots the average of 50 trials. Predictably, increasing the planning depth improves performance for agents which use the objective reward, as shown by the fact that the square marker is above the diamond marker at 0 time steps. Also as predicted, the ability of a leaf-evaluation heuristic to improve performance is reduced as the tree is deepened, as shown by the superior performance of the Depth-5 Leaf-Value(PGRD) agent (dashed line, diamond markers) compared to the Depth-15 Leaf-Value(PGRD) agent (dashed line, square markers) and the similar trend observed in the V^*-leaf-evaluation heuristic agents (inset on right).

In contrast, general reward functions are more resilient to this bias at large planning depths. The learned rewards (top 2 curves) converge to near-optimal behavior at all tested depths. Also, observe that the Depth-5 Leaf-Value(PGRD) agent outperforms the corresponding Depth-5 V^*-LEH agent shown in the inset, verifying our claim that the optimal LEH, L^*, can outperform the V^*-LEH when the agent is unable to exactly evaluate the expected return of the truncated trajectory.

Discussion

Although additional heuristics exist for overcoming some of the agent limitations presented here (Gelly and Silver 2008), solving the optimal reward problem is a general, principled approach which applies across agent algorithms. We have demonstrated that due to the added expressiveness of path dependence, the optimal reward function approach is better than the leaf-evaluation heuristic approach and the potential-based reward shaping approach at improving the performance of planning agents which make approximations in shallow portions of the planning tree. Specifically, we showed that reward design is preferred in agents affected by sampling errors and search-control bias. We presented convergent and approximate online reward optimization methods for Sparse Sampling and UCT, respectively, and used these methods to learn approximately optimal reward functions and leaf-evaluation heuristics, improving the performance of UCT, a state-of-the-art planning agent.

Acknowledgments: This work was supported by the Air Force Office of Scientific Research under grant FA9550-08-1-0418 as well as by NSF grant IIS 0905146. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

References

Asmuth, J.; Littman, M. L.; and Zinkov, R. 2008. Potential-based shaping in model-based reinforcement learning. In *Proceedings of the 23rd AAAI*, 604–609. AAAI Press.

Bartlett, P. L., and Baxter, J. 2000. Stochastic optimization of controlled partially observable Markov decision processes. In *Proceedings of the 39th IEEE Conference on Decision and Control*.

Finnsson, H., and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *Proceedings of the 23rd AAAI*, 259–264.

Gelly, S., and Silver, D. 2008. Achieving master level play in 9 x 9 computer Go. *Proceedings of the 23rd AAAI*.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 1999. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings of the 16th IJCAI*, 1324–1331.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the 17th ECML*, 282–293.

Neu, G., and Szepesvári, C. 2007. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the 23rd UAI*, 295–302.

Ng, A. Y.; Russell, S. J.; and Harada, D. 1999. Policy invariance under reward transformations: theory and application to reward shaping. In *Proceedings of the 16th ICML*, 278–287.

Shannon, C. E. 1950. Programming a computer for playing chess. In *Philosophical Magazine* Vol. 41, 256–275.

Sorg, J.; Singh, S.; and Lewis, R. L. 2010a. Gradient methods for internal reward optimization. In *Advances in NIPS 23*.

Sorg, J.; Singh, S.; and Lewis, R. L. 2010b. Internal rewards mitigate agent boundedness. In *Proceedings of the 27th ICML*.