On the Analysis of Complex Backup Strategies in Monte Carlo Tree Search

Piyush Khandelwal PIYUSHK@CS.UTEXAS.EDU
Elad Liebman ELADLIEB@CS.UTEXAS.EDU
Scott Niekum SNIEKUM@CS.UTEXAS.EDU
Peter Stone PSTONE@CS.UTEXAS.EDU
Department of Computer Science, University of Texas at Austin, 2317 Speedway, Stop D9500, Austin, TX 78712 USA

Abstract

Over the past decade, Monte Carlo Tree Search (MCTS) and specifically Upper Confidence Bound in Trees (UCT) have proven to be quite effective in large probabilistic planning domains. In this paper, we focus on how values are backpropagated in the MCTS tree, and apply complex return strategies from the Reinforcement Learning (RL) literature to MCTS, producing 4 new MCTS variants. We demonstrate that in some probabilistic planning benchmarks from the International Planning Competition (IPC), selecting a MCTS variant with a backup strategy different from Monte Carlo averaging can lead to substantially better results. We also propose a hypothesis for why different backup strategies lead to different performance in particular environments, and manipulate a carefully structured grid-world domain to provide empirical evidence supporting our hypothesis.

1. Introduction

Planning under uncertainty, or probabilistic planning, is an important part of many intelligent systems. Problems requiring such planning can be represented as Markov Decision Processes (MDPs) (Sutton and Barto, 1998), and solved using techniques such as Value Iteration (VI) (Bellman, 1957). More recent approaches such as RTDP (Barto *et al.*, 1995) and LAO* (Hansen and Zilberstein, 2001) can solve MDPs more efficiently than VI by restricting traversal of state-action space. While these approaches require access to a full declarative model of an MDP, Monte Carlo Tree Search (MCTS), or specifically Upper Confidence Bound in Trees (UCT) (Kocsis and Szepesvári, 2006), is an anytime planning algorithm designed for approximately solving MDPs that only requires generative samples from the MDP. UCT was first popularized due to excellent performance in Computer Go (Gelly and Wang, 2006; Chaslot *et al.*, 2008), and it has also performed well in large Partially Observable MDPs (POMDPs) (Silver and Veness, 2010). The PROST planner (Keller and Eyerich, 2012), based on UCT, won the probabilistic track of the International Planning Competition (IPC) in 2011 and 2014.

In this paper, we explore how complex backup techniques, a longstanding focus of the Reinforcement Learning (RL) literature that has not previously been considered in MCTS, can improve the performance of MCTS in some domains. In the original MCTS algorithm, the long-term expected return of taking an action is averaged using Monte Carlo backups. In contrast, it has been demonstrated that complex backup strategies can speed up convergence of value estimates in RL problems. Such complex backup strategies include the λ-return (Sutton, 1988), γ-return (Konidaris *et al.*, 2011), and the Ω-return (Thomas *et al.*, 2015), where the last two approaches are parameter free. Motivated especially by real-time applications that require finding good policies within limited per-step computation time (e.g. Khandelwal *et al.* (2015)), we evaluate these backup strategies using benchmarks with limited planning time.

The main contributions of this paper are four novel MCTS variants which utilize complex backup strategies for value backpropagation in MCTS. We find that one proposed parameter-free backup strategy, MaxMCTS$_\gamma$, performs equivalently to or better than Monte Carlo backups in all probabilistic planning benchmarks from the IPC. Another novel strategy, MaxMCTS(λ), performs better than all other backup strategies when an appropriate value of λ is selected. Apart from experiments on IPC domains, we also study how domain structure influences performance of backup strategies in a grid-world domain, shedding some light on the empirical results presented in this paper.

The remainder of this paper is organized as follows. We first present a general overview of MCTS in § 2, followed by how complex backup strategies can be applied to MCTS

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

in § 3. We then empirically evaluate and analyze different backup strategies in § 4. Next, we discuss related work in § 5 before concluding in § 6.

2. Monte Carlo Tree Search

Given a simulator or a generative model of the environment, Monte Carlo Tree Search (MCTS) is an anytime algorithm that runs simulations to estimate the action at the current state that produces the highest return, i.e. the cumulative reward over the planning horizon. MCTS maintains a tree structure to store information about state-actions it has encountered while performing simulations, with the root node of the tree representing the current MDP state. Algorithm 1 outlines the pseudocode for MCTS planning. Within every simulation, there are 4 separate phases:

- *Selection* - An action selection strategy is recursively applied at all nodes in the simulation trajectory starting from the root node (Line 7). Strategies aim to restrict search to more important regions of the state-action space, and are outlined in § 2.1. If a state has not been previously visited, then the action is chosen according to some default policy (Line 6), which can be randomized or designed using expert knowledge.
- *Simulation* - Once an action is selected, a generative model of the domain is used to generate a transition to the next state (Line 8). The process of *Selection* and *Simulation* are interleaved to generate a trajectory. A user-specified parameter called *planHorizon* controls the trajectory length (or MCTS tree depth). This horizon is useful in domains where termination is rare.
- *Expansion* - The search tree is expanded using one or many new states encountered while simulating. The number of new states added per simulation can be parametrized, and is typically set depending on memory constraints. In this paper, since there are few planning simulations, all new states are added to the search tree (Line 9) to speed up convergence.
- *Backpropagation* - The trajectory is stored via a stack (Line 10), and the return estimates of encountered state-actions are updated via a backup strategy (Line 12). We study different backup strategies in § 3.

Once planning is complete, the action that maximizes the return at the current state s is selected as:

$$a = \arg\max_a [rootNode.Q_{s,a}], \tag{1}$$

where $rootNode.Q_{s,a}$ represents the long-term expected value for action a, i.e. $Q(s,a)$. When the agent takes this action and reaches a new system state *ns*, the planning process is repeated before taking the next action. Additionally, the appropriate sub-tree from the previous search can be used to bootstrap search at this new system state.

Algorithm 1 MCTS starting at state s

```
1:  rootNode ← initNode(s)
2:  for sim ∈ {1, ..., numSimulations} do
3:      node ← rootNode
4:      trajectory ← new Stack
5:      while trajectory.size() < planHorizon and
                notTerminal(node) do
6:          if node.n_s = 0 then a ← defaultAction(node)
7:          else a ← selectAction(node)
8:          ⟨ns, reward⟩ ← simulate(node, a)
9:          nextNode ← getNode(node, a, ns)
10:         trajectory.push(node, a, reward)
11:         node ← nextNode
12:     BACKPROPAGATE(trajectory)
```

2.1. Action Selection Strategies

Two action selection strategies are used during evaluation:

2.1.1. UNIFORM

The planning action is selected randomly from the set of all K actions available at state s that have been tried the least. The probability of selecting action a is:

$$p_a = \begin{cases} 1/K & node.n_a \leq node.n_{a'} \quad \forall [a' \neq a] \\ 0 & \text{otherwise} \end{cases}.$$

Since uniform action selection is parameter free and does not depend on backpropagated values, it is well suited for comparing different backup strategies.

2.1.2. UCT

The UCB1 algorithm is popular for action selection in MCTS, and the resulting MCTS algorithm is known as Upper Confidence bounds for Trees (UCT) (Kocsis and Szepesvári, 2006). In UCT, uniform action selection is performed until each action at a given state node is selected once, and subsequent actions are selected as:

$$a = \arg\max_a \left(node.Q_{s,a} + c_p \sqrt{\frac{\ln(node.n_s)}{node.n_a}} \right),$$

where $node.n_s$ is the number of visits to $node$, $node.n_a$ is the number of times action a was previously selected during planning at this node, $node.Q_{s,a}$ is the current expected long term reward for taking action a, i.e. $Q(s,a)$, and $c_p = \sqrt{2}$ when the MDP rewards are scaled between 0 and 1. c_p can also be tuned empirically to yield better performance by better balancing exploration versus exploitation, especially when the number of simulations is small.

3. Complex Backup Strategies

In this section, we briefly revisit complex backup strategies as applied in RL problems (Sutton and Barto, 1998; Konidaris *et al.*, 2011). We then formally describe how these backup strategies can be applied to MCTS.

Monte Carlo RL algorithms use the full return from a sample trajectory to update a given state-action value estimate. A sample of of the state-action value estimate, i.e. the return sample, is computed as:

$$R^{MC}_{s_t,a_t} = \sum_{i=0}^{L-1} \gamma^i r_{t+i}, \tag{2}$$

where a_i is the action taken at state s_i to yield reward r_i and a transition to state s_{i+1}. State s_{t+L} is terminal, and γ is the MDP discount factor.

In contrast, SARSA (Sutton and Barto, 1998), a Temporal Difference (TD) learning approach, uses an on-policy backup that bootstraps a one step reward transition with existing state-action value estimates. In SARSA, this one-step return sample is computed as:

$$R^{SARSA}_{s_t,a_t} = r_t + \gamma Q_{s_{t+1},a_{t+1}}, \tag{3}$$

where $Q_{s,a}$ is the value estimate for $\langle s, a\rangle$, i.e. $Q(s, a)$.

Both the Monte Carlo and SARSA return samples can be described using a single generalized n-step return sample, assuming the Q-value at the terminal state $Q_{s_{t+L,a}} = 0$:

$$R^{(n)}_{s_t,a_t} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n Q_{s_{t+n},a_{t+n}}, \tag{4}$$

where $R^{(n)}_{s_t,a_t}$ is equivalent to the Monte Carlo return sample (2) when $n = L$, and it is equal to the SARSA return sample (3) when $n = 1$.

Instead of using either the Monte Carlo or the one-step return sample to estimate the state-action value estimate Q_{s_t,a_t}, learning algorithms that employ complex backup strategies update the estimate using a combination of all n-step returns. This combined complex return sample $R^C_{s_t,a_t}$ can be described as:

$$R^C_{s_t,a_t} = \sum_{i=1}^{L} w_{n,L} R^{(n)}_{s_t,a_t}, \tag{5}$$

where $w_{n,L}$ is the weight assigned to the n-step return sample, $\sum_{n=1}^{L} w_{n,L} = 1$, and L is the length of the trajectory. Different complex backup strategies use a different combination of weights, as follows.

SARSA(λ) (Rummery, 1995) is one such complex backup learning algorithm that uses the λ-return to update state-action value estimates. λ is a user-defined parameter ($0 \leq \lambda \leq 1$) that sets the different weights as:

$$w^{\lambda}_{n,L} = \begin{cases} (1-\lambda)\lambda^{n-1} & 1 \leq n < L \\ \lambda^L & n = L \end{cases}. \tag{6}$$

TD$_\gamma$ (Konidaris *et al.*, 2011) is another complex backup learning algorithm which uses the parameter-free γ-return to update state value estimates, and defines the following weights to compute the complex return in (5):

$$w^{\gamma}_{n,L} = \frac{(\sum_{i=1}^{n} \gamma^{2(i-1)})^{-1}}{\sum_{n=1}^{L} (\sum_{i=1}^{n} \gamma^{2(i-1)})^{-1}}, \tag{7}$$

where γ is the MDP discount factor. While TD$_\gamma$ was originally designed to estimate state values ($V(s)$), the weights used in the complex return can also be applied to estimate state-action values ($Q(s, a)$).

It should be noted that the weights used by the λ-return are not statistically sound, since weights should depend on the variance of each n-step return, whereas the γ-return is more statistically principled (Konidaris *et al.*, 2011). However, the λ-return is mathematically more convenient to implement than the γ-return and is thus more commonly used.

The returns defined in (4) and (5) describe on-policy value estimation, i.e the state-action value estimates reflect the planning/exploration policy. Since action selection after planning is greedy [see (1)], an off-policy approach such as one based on Q-Learning (Watkins, 1989) may produce better results. In contrast to SARSA, Q-Learning uses the following one-step off-policy bootstrapped return:

$$R^{Q}_{s_t,a_t} = r_t + \gamma \max_a Q_{s_{t+1},a}. \tag{8}$$

Similar to (4) and (5), we can construct a generalized off-policy n-step return $MaxR^{(n)}_{s_t,a_t}$ and complex return $MaxR^C_{s_t,a_t}$ as:

$$MaxR^{(n)}_{s_t,a_t} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \max_a Q_{s_{t+n},a}, \text{and} \tag{9}$$

$$MaxR^C_{s_t,a_t} = \sum_{i=1}^{L} w_{n,L} MaxR^{(n)}_{s_t,a_t}. \tag{10}$$

Using the weights for the λ-return from (6) with the off-policy complex return described in (10) leads exactly to Peng's Q(λ) off-policy learning algorithm (Peng and Williams, 1996). Similarly, the weights for the γ-return from (7) with the off-policy complex return yields an off-policy γ-return based approach.

We have now described 4 different backup strategies: λ-return, off-policy λ-return, γ-return, and off-policy γ return. Next, we describe how these strategies can be incorporated within the MCTS framework to produce 4 different variants called: MCTS(λ), MaxMCTS(λ), MCTS$_\gamma$, and MaxMCTS$_\gamma$, respectively. These algorithms combine the tree structure and planning horizon of MCTS with different value backup strategies. By doing so, these proposed algorithms bridge a gap between the general problem of value estimation in reinforcement learning, and the advantages of accelerated exploration and planning in MCTS.

3.1. MCTS(λ) and MaxMCTS(λ)

After each planning trajectory in the MCTS tree is carried out to termination, the value of each state-action encountered within the trajectory needs to be updated (Alg. 1 Line 12). This process, using the λ-return, is described in Algorithm 2. Due to the nature of the λ-return, the update rule for every state can make use of a single cumulative return q backpropagated up the tree (Line 5), where q is equivalent to $(Max)R^C_{s_t,a_t}$. The number of state and state-action visits are stored for UCT action selection (Lines 3-4). The state-action value is updated via averaging (Line 6).

Algorithm 2 BACKPROPAGATE(*trajectory*) for λ-return.

1: $q \leftarrow 0$ # Backup Value/Complex Return $(Max)R^C_{s_t,a_t}$.
2: **for** $\langle node, a, r\rangle = trajectory.pop()$ **do**
3: $node.n_s \leftarrow node.n_s + 1$
4: $node.n_a \leftarrow node.n_a + 1$
5: $q \leftarrow q + r$, $\delta_Q \leftarrow q - node.Q_{s,a}$
6: $node.Q_{s,a} \leftarrow node.Q_{s,a} + (\delta_Q / node.n_a)$
7: **if** MCTS(λ) **then** $q \leftarrow (1-\lambda) node.Q_{s,a} + \lambda q$
8: **else** # MaxMCTS(λ)
$q \leftarrow (1-\lambda)\max_{a'|node.n_{a'}\neq 0}[node.Q_{s,a'}] + \lambda q$

In MCTS(λ), λ is used to interpolate between the current cumulative return and the existing Q-value of that state-action (Line 7), effectively updating each state-action using Eqns. (4), (5), and (6). In contrast, MaxMCTS(λ) interpolates the current cumulative return with the max Q-value at that state (Line 8), effectively updating each state-action in an off-policy manner using Eqns. (9), (10), and (6). When $\lambda = 1$, backpropagation in both algorithms is the same and equivalent to Monte Carlo backups.

Comparing these two algorithms, we expect that the performance of MCTS(λ) would not vary much with λ. Since the number of planning simulations is limited, the number of visits to a state-action pair further down the tree can often be fairly low, and the state-action value estimate $Q(s,a)$ may not be informative. In contrast, the performance of MaxMCTS(λ) is likely to vary more with λ if another action at a tree node has significantly higher value.

In MaxMCTS(λ), the eligibility parameter λ balances Monte Carlo ($\lambda = 1$) and Max Monte Carlo ($\lambda = 0$) updates. At $\lambda = 0$, with the action selection strategy as UCT, MaxMCTS(λ) is equivalent to MaxUCT (Keller and Helmert, 2013). In Max Monte Carlo backpropagation, even if an exploratory action is taken at a given state node, the value of the action with the highest expected reward is propagated higher up in the tree minimizing the effect of exploratory actions taken further down in the tree on value estimates. The backup diagrams for both MCTS(λ) and MaxMCTS(λ) are illustrated in Figure 1 to show how λ affects value estimation in the MCTS tree.

3.2. MCTS$_\gamma$ and MaxMCTS$_\gamma$

The MCTS$_\gamma$ and MaxMCTS$_\gamma$ algorithms mirror MCTS(λ) and MaxMCTS(λ), respectively, with two main differences. First, the rule updating state-action value estimation uses the γ-return (7) instead of λ-return (6) while computing the complex combined return. Second, unlike the λ-return approaches, the γ-return cannot be implemented using a single cumulative return that is backpropagated up the tree, and all n-step returns must be computed.

Pseudocode for implementing both MCTS$_\gamma$ and MaxMCTS$_\gamma$ is presented in Algorithm 3. The algorithm keeps track of all n-step returns (Lines 8, 12, and 13), and then uses the weights for the γ-return to compute the combined return sample (Line 9). The Q-value estimate is then updated using sample averaging (Lines 10-11). Lines 12-13 differentiate between on-policy (MCTS$_\gamma$) and off-policy (MaxMCTS$_\gamma$) updates by varying whether (i) the state-action value estimate, or (ii) the max state-action value estimate at that state, respectively, is used while computing the n-step return.

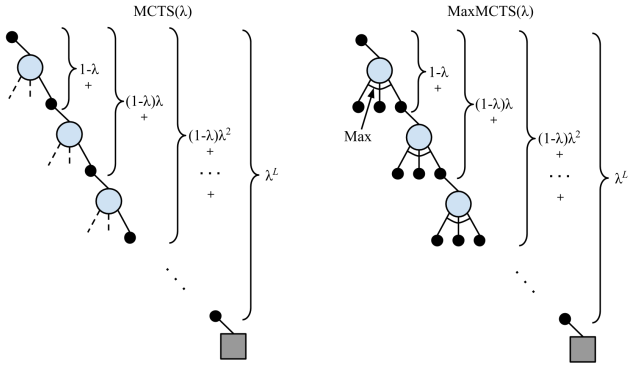


Figure 1. Backup diagram for MCTS(λ) and MaxMCTS(λ) (in the style of (Sutton and Barto, 1998)). The figure demonstrates how the complex return $(Max)R^C_{s_t,a_t}$ is computed in the MCTS tree using the weights for the λ-return (6). Each individual term represents an n-step return using (4) and (9) for on-policy and off-policy update, respectively. MCTS$_\gamma$ and MaxMCTS$_\gamma$ have similar backup diagrams, except weights for the γ return (7) are used.

Algorithm 3 BACKPROPAGATE(*trajectory*) for γ-return.

nreturns $\leftarrow$ **new** List # List of $(Max)R^{(n)}_{s_t,a_t}$.
$L \leftarrow 1$ # Trajectory length from current node.
for $\langle node, a, r \rangle = trajectory.pop()$ **do**
  $node.n_s \leftarrow node.n_s + 1$
  $node.n_a \leftarrow node.n_a + 1$
  $q \leftarrow 0$ # Complex Return $(Max)R^{C}_{s_t,a_t}$
  for $i \in \{1, \ldots, L\}$ **do**
    $nreturns[i] \leftarrow nreturns[i] + r$
    $q \leftarrow q + w^{\gamma}_{i,L} \times nreturns[i]$
  $\delta_Q \leftarrow q - node.Q_{s,a}$
  $node.Q_{s,a} \leftarrow node.Q_{s,a} + (\delta_Q / node.n_a)$
  if MCTS$_\gamma$ **then** *nreturns.insertStart*$(node.Q_{s,a})$
  else # MaxMCTS$_\gamma$
    nreturns.insertStart$(\max_{a' | node.n_{a'} \neq 0} [node.Q_{s,a'}])$
  $L \leftarrow L + 1$

4. Experiments

We now empirically demonstrate that complex backup strategies introduced in § 3 can often outperform Monte Carlo backups (i.e. MCTS(1)) in MCTS, using benchmarks from the probabilistic track of the International Planning Competition (IPC). We then perform experiments in a grid-world environment to better understand the relationship between domain structure and backup strategy (§ 4.2).

4.1. IPC Domains

IPC domains are described using RDDL (Sanner, 2010)[1], and parsed using the RDDL parser available with the PROST planner (Keller and Eyerich, 2012). Each IPC domain includes 10 problems with different complexities and reward structures. For ease of analysis, only one problem of medium difficulty, problem 5 in all domains, is tested.

All the IPC domains have an intrinsic built-in horizon of 40 steps, i.e. states are terminal if and only if 40 actions have been taken from the start, and MCTS planning is performed until this horizon. Before taking each action, the agent plans for 10,000 simulations (trajectories) while reusing the appropriate sub-tree from the previous search. To evaluate the effect of backup strategies in isolation, planning actions are selected using uniform action selection (§ 2.1.1).

Performance results for all domains are illustrated in Figure 2, and a subset of these results are tabulated in Table 2. Domains in Figure 2 have been ordered such that in those on the left, MaxMCTS(λ) performs best with a low to mid-range value of λ, and in domains on the right, a λ close to 1, i.e. the Monte Carlo backup, performs best. All results are averaged over 1,000 independent trials, and Welch's t-test (with $p < 0.05$) is used for significance testing.

There are several key observations we can make regarding the experimental results across the IPC domains. MaxMCTS(λ) with an appropriately selected λ typically outperforms all other approaches, but typically also performs the worst if λ is incorrectly selected. Optimal values of λ range from 0 (Elevators) to 0.3 (Recon) to 0.6 (Triangle Tireworld) to 0.7 (Academic Advising) to 1 (Skill Teaching). As Table 2 shows, while the Monte Carlo (MCTS(1)) backup performs well in many domains, MaxMCTS(λ) performs better with an intermediate value of λ in 3 domains, and $\lambda = 0$ in one domain.

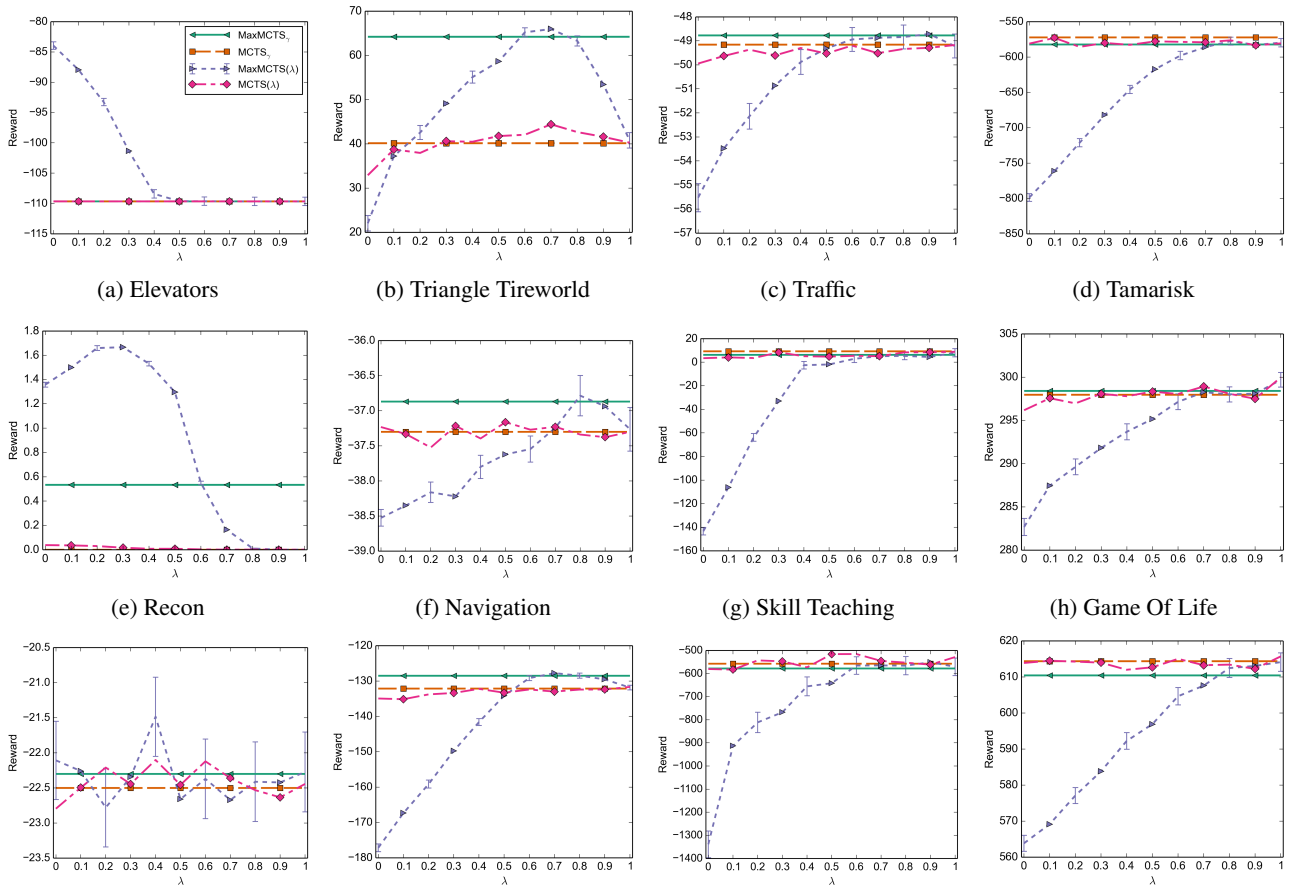
As Figure 2 shows, and as hypothesized in § 3.1, MCTS(λ) does not show substantial difference in performance when λ is varied. In Table 2, neither on-policy approaches, MCTS(λ) and MCTS$_\gamma$, perform significantly better than Monte Carlo.

It is also important to note that MaxMCTS$_\gamma$ performs reasonably well when compared to Monte Carlo, especially in domains where MaxMCTS(λ) performs best with $\lambda > 0.5$. As shown in Table 2, MaxMCTS$_\gamma$ always performs equivalently to or significantly better than Monte Carlo (in Recon, Triangle Tireworld, and Academic Advising), and should always be preferred. This fact is useful, since MaxMCTS$_\gamma$ is parameter free, and unlike MaxMCTS(λ) does not require selecting an appropriate λ value. As a result, in situations when it may be computationally intractable to try out many different backup strategies, i.e. many different values of λ, simply evaluating MaxMCTS$_\gamma$ and MaxMCTS(0) is likely sufficient for getting near-optimal performance between one of these two proposed backup approaches.

The results thus far clearly establish that the type of backup strategy used in MCTS can significantly affect performance. One interesting question this result raises is how the magnitude of the backup strategy's effect compares to that of action selection. In fact, whereas conventional wisdom has been that using UCT (§ 2.1.2) for action selection is the critical factor for improving performance compared to uniform action selection, we find that at least in some domains, altering the backup strategy from the traditional MCTS(1) can improve performance significantly more than changing the action selection strategy.

Figure 3 illustrates the performance of UCT, where the exploration/exploitation tradeoff parameter in UCT (c_p) is tuned via grid search, and uniform action selection in Elevators and Recon. In Elevators (Fig. 3a), the action selection strategy has no impact on performance with a pure Monte Carlo backup (MCTS(1)), whereas MaxMCTS(0) performs much better with either uniform or UCT ac-

[1] Domains are available at `https://github.com/ssanner/rddlsim`. This paper uses updated definitions for *Recon* and *Skill Learning*, as explained in `rddlsim/files/final_comp/ERRATA.txt`. In order to reduce the state-action space, domain constraints have been added to collapse all actions which are equivalent to the *null* action in that domain into to a single null action. These superfluous actions can also be removed analytically using a declarative model of the domain (Keller and Eyerich, 2012).

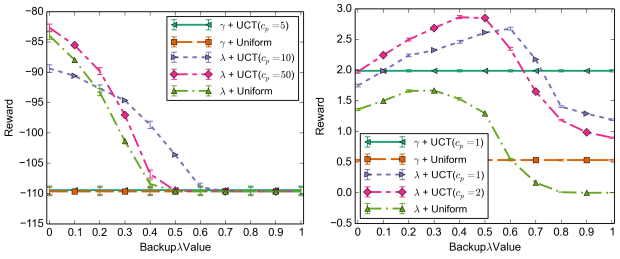


(a) Elevators (b) Triangle Tireworld (c) Traffic (d) Tamarisk

(e) Recon (f) Navigation (g) Skill Teaching (h) Game Of Life

(i) Crossing Traffic (j) Academic Advising (k) Wildfire (l) Sysadmin

Figure 2. Results for each of the 12 IPC domains show the performance of different backup strategies when the action selection strategy was Uniform. Only the standard error for MaxMCTS(λ) is shown as error bars for clarity; all methods have comparable error bars.



(a) Elevators (b) Recon

Figure 3. Results for 2 representative IPC domains show domain performance of MaxMCTS(λ) and MaxMCTS$_\gamma$ with both uniform and UCT action selection. Only the best performing UCT variants are illustrated.

tion selection. Similarly, in Recon (Fig. 3b), uniform action selection with MaxMCTS(0.3) significantly outperforms Monte Carlo (MCTS(1)) backups with UCT. In other words, in domains similar to these two, selecting the appropriate backup strategy is more important than tuning the action selection strategy with Monte Carlo backups.

Now that we have demonstrated that the choice of backup strategy can significantly impact performance, we also compare the computational costs of different backup strategies. Table 1 compares the average time per episode across 6 representative domains. Since each IPC domain has 40 action-selection steps per trial, and 10,000 simulations are performed per selection step, these results represent the cumulative time difference across 400,000 planning simulations. All the code is implemented in C++ and all backup strategies have been implemented efficiently. In general, backpropagation takes between 15-25% of total planning time. The overall difference in computational cost among different backup strategies is less than 10%.

	Elev	**Nav**	**Recon**	**ST**	**TT**	**Wildfire**
Monte Carlo	196	95	246	225	176	638
MaxMCTS(λ)	202	107	252	230	192	676
MaxMCTS$_\gamma$	202	98	250	224	190	694

Table 1. Average time per episode in seconds for different backup strategies given a fixed number of planning simulations.

	AA	CT	Elev	GOL	Nav	Recon	ST	Sysadm	Tam	Traffic	TT	Wildfire
Monte Carlo	-131.88	-22.27	-109.67	299.69	-37.27	0.0	8.07	614.09	-579.73	-49.22	40.78	-571.13
MaxMCTS(λ)	**-127.78**	-21.49	***-83.95***	299.69	-36.79	***1.67***	8.07	614.09	-577.62	-48.71	**65.94**	-551.25
at λ-value	0.7	0.4	0.0	1.0	0.8	0.3	1.0	1.0	0.8	0.9	0.7	0.9
MaxMCTS$_\gamma$	**-128.53**	-22.3	-109.67	298.4	-36.87	**0.53**	6.16	610.43	-582.33	-48.77	**64.19**	-578.05
MCTS(λ)	-131.69	-22.1	-109.67	299.97	-37.16	**0.04**	8.83	615.68	-573.07	-49.17	44.4	-515.8
at λ-value	1.0	0.4	0.0	1.0	0.5	0.0	1.0	1.0	0.1	1.0	0.7	0.6
MCTS$_\gamma$	-132.19	-22.5	-109.67	297.97	-37.3	0.0	9.29	614.33	-572.2	-49.16	40.11	-557.74

Table 2. Mean performance of different backup strategies for all 12 IPC domains. A value is bold if it is significantly better than Monte Carlo, i.e. MaxMCTS($\lambda = 1$). A value is italicized if it is significantly better than MaxMCTS$_\gamma$.

Next, we study the connection between domain structure and choice of backup strategy using a grid-world domain.

4.2. Grid World

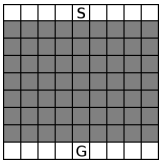
It is evident from the results across the IPC domains that the same complex backup strategy can lead to vastly different performance across particular domains. How is domain structure linked to different performance for different backup strategies, and can we use this structure to inform our choice of backup strategy? To shed more light on these questions, we perform multiple tests on a controlled grid-world environment, and obtain a deeper understanding of the MCTS mechanics given different conditions.

All of our tests are conducted on a grid-world environment of size 9x9 which has to be navigated by an agent, as depicted in Figure 4a. The agent's start and goal locations are fixed and labeled as S and G in the figure, respectively. The domain supports 4 non-deterministic actions: *up*, *down*, *left*, and *right* with the following transitions:

$$p(ns_a|a,s) = 0.925, \text{ and } p(ns_{a'}|a,s) = 0.025 \mid a' \neq a,$$

where $a, a' \in \{up, down, left, right\}$, s is the agent's current cell, ns_a is the adjacent cell in direction a. If the agent is next to an edge, the probability mass of moving into the edge is mapped into the current cell of the agent. When the agent reaches the goal state, it is given a reward of +100, and all other actions result in a reward of -1. The MCTS search depth parameter, *planHorizon*, is set to 100. Planning is done using 10,000 simulations with uniform action selection, and the previous search tree is reused.

Our main hypothesis regarding the performance of different backup strategies revolves around value estimation during exploration. More exactly, we believe it is a matter of how many optimal or close-to-optimal trajectories exist in the tree. In the grid-world environment, there are cases when there are a lot of paths in the search tree that the agent can use to get to the goal. Consequently, using a Monte Carlo backup produces the best results (Figure 4b, $\lambda = 1$ with no 0-reward terminal states), as other backup approaches have a greater likelihood of getting stuck in suboptimal trajectories due to an off-policy update.



(a) Grid

#0-Term	0	3	6	9	12	15	18
$\lambda = 1$	***90.4***	11.3	0.9	-1.3	-2.1	-2.2	-2.2
$\lambda = 0.8$	90.2	28.0	10.7	5.9	0.3	-1.4	-2.0
$\lambda = 0.6$	89.5	62.8	45.3	30.6	17.5	8.5	3.3
$\lambda = 0.4$	88.7	***85.1***	77.6	62.2	41.7	24.1	10.1
$\lambda = 0.2$	87.7	82.6	**78.1**	***69.9***	**51.3**	28.4	13.2
$\lambda = 0$	84.5	79.8	74.1	67.0	50.2	**31.8**	**15.75**
γ	90.1	25.7	15.3	13.2	8.2	4.7	2.3

(b) Varying 0-Reward Terminal States

Figure 4. The gray cells in (a) indicate the cells that can become 0-reward terminal states. (b) tabulates average episodic reward as the number of 0-reward terminal states is varied. The highest performing backup strategy for each domain setting is bold, and also italicized if significantly better than all other column entries.

On the other hand, if there are very few successful paths to the goal, then averaging out multiple rollouts can easily drown value backups from rare close-to-optimal trajectories. We test this hypothesis by introducing a number of additional terminal states, or barriers, between the start and goal locations in the grid formulation. These barriers are chosen at random from the grey cells in Figure 4a. If the agent transitions to one of these cells, it receives a reward of 0, and the episode terminates. The average episodic reward obtained by MaxMCTS(λ) and MaxMCTS$_\gamma$ as the number of terminal states varied is tabulated in Figure 4b.

When the number of obstacles is 0, Monte Carlo or MaxMCTS(1) significantly outperforms all other approaches. However, with only three obstacles, the performance of MaxMCTS$_\gamma$ and MaxMCTS($\lambda \geq 0.6$) deteriorates sharply, and MaxMCTS(0.4) significantly outperforms all other approaches. The deterioration in performance for

MaxMCTS(λ) is inversely related with the number of obstacles. At 9 obstacles, MaxMCTS(0.2) significantly outperforms all other approaches, and at 15 MaxMCTS(0) performs best (with a p-value of 0.075). The fact that MaxMCTS$_\gamma$ behaves similarly to MaxMCTS(λ) with high λ conforms with our observations from the IPC domains.

These results support our hypothesis. As reaching the goal becomes increasingly harder, it becomes necessary to retain more information from the few instances where the goal is reached. Off-policy complex backup strategies can do so by repeatedly backpropagating the best subtree at a node instead of the Monte Carlo backup value. In the general case, when there are few paths in a domain that lead to good long term reward values, it may be more suitable to use MaxMCTS(λ) with a lower value of λ. We can further corroborate this observation by looking at the specific IPC domains where MaxMCTS$_\lambda$ with low λ values did well, such as Elevators and Recon, where only a few successful trajectories exist. In other words, these are domains where it is easier, given limited simulation experience, to learn a successful yet suboptimal trajectory fast, than to try and obtain a more accurate statistical estimate by averaging noisy returns from multiple trajectories.

5. Related Work

Since its modern introduction in 2006 (Kocsis and Szepesvári, 2006; Coulom, 2007; Chaslot *et al.*, 2008), many different versions and variations of the Monte Carlo Tree Search (MCTS) architecture have been proposed and studied empirically. For example, variants such as UCB-tuned have modified the upper confidence bound of UCB1 to account for variance, in order to perform better exploration (Auer *et al.*, 2002). Tesauru et al. proposed a Bayesian version of UCT, which engenders better estimations of node values and uncertainties given limited experience, but is more computation-intensive (Tesauro *et al.*, 2012). Various modifications have been proposed to improve the exploration strategy under UCT (Gelly and Wang, 2006; Wang and Gelly, 2007) and its value estimation (Gelly and Silver, 2007; 2011; Helmbold and Parker-Wood, 2009; Feldman and Domshlak, 2014b). Additional work has combined temporal-difference-like offline learning with online learning in Monte Carlo Tree Search (Gelly and Silver, 2007; Silver *et al.*, 2008; Osaki *et al.*, 2008).

By and large, the majority of previous work related to MCTS has focused on the action selection strategy. Nonetheless, The idea of applying more sophisticated backup strategies in planning is not new. Hester presented a variation on UCT similar to MaxMCTS(λ) as part of the general TEXPLORE framework, and applied it to various domains without analyzing the effect of varying the λ parameter (Hester, 2012). Silver et al. applied the idea of eligibility traces to planning in Dyna2 (Silver *et al.*, 2008). While it may be possible to adapt Dyna2's approach to MCTS, this approach is unsuitable for deterministic action selection algorithms such as UCT, since it is possible that values of nodes closer to the root may not change when $\lambda < 1$ in Dyna2, which would cause UCT to under-explore by selecting the same action at these nodes.

Keller and Helmert also proposed several variants to UCT apart from MaxUCT, such as DP-UCT, which performs partial backpropagation proportionately to successor probabilities, and UCT*, which combines DP-UCT with heuristic search (Keller and Helmert, 2013). These algorithms require access to a declarative model of the MDP, whereas variants presented in this paper do not. Feldman and Domshlak have compared Monte Carlo to Bellman updates in MCTS, and were able to show formal regret bounds for both (Feldman and Domshlak, 2014a).

Several previous papers have comparatively studied different components of the MCTS architecture. As in the general literature, most of these surveys focused on the action selection strategy component of MCTS. Helmbold and Parker-Wood studied different AMAF heuristics for Monte Carlo Go (Helmbold and Parker-Wood, 2009). Perick et al. compared different exploration strategies in Monte Carlo Tree Search for the game of Tron (Perick *et al.*, 2012). More closely related to this paper, Coulom analyzed several domain-specific and hand designed backup strategies in MCTS, tailored exclusively for Crazy Stone, an agent for 9X9 GO (Coulom, 2007). To the best of our knowledge, we are the first to rigorously analyze the usage of principled backpropagation strategies in MCTS.

6. Conclusion

The main contribution of this paper has been the introduction of complex backup strategies to the MCTS algorithm. Specifically, 4 novel MCTS variants are proposed: MCTS(λ), MaxMCTS(λ), MCTS$_\gamma$, and MaxMCTS$_\gamma$. Using various benchmarks from the IPC, we've demonstrated that the choice of backup strategy can have significant impact on performance. One of the proposed approaches, MaxMCTS$_\gamma$, is parameter-free and performs as well or better than Monte Carlo backups in all domains tested. Another proposed approach, MaxMCTS(λ), outperforms all other proposed and existing MCTS variants. In many domains, an intermediate value of λ between 0 and 1 is necessary to obtain best performance, whereas existing algorithms are equivalent to using one of the extreme values. We have hypothesized that the backup strategy MaxMCTS(λ) with a low value of λ outperforms other backup strategies in domains where trajectories with high rewards are rare, and provided empirical evidence to support this hypothesis.

Acknowledgments

The authors would like to thank Todd Hester and Samuel Barrett for numerous discussions and their work on UCT that has culminated in the publication of this paper.

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-1330072, CNS-1305287), ONR (21C184-01), AFRL (FA8750-14-1-0070), and AFOSR (FA9550-14-1-0087).

References

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

Richard Bellman. Dynamic programming, 1957.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo tree search: A new framework for game AI. In *Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE)*, 2008.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, pages 72–83. Springer, 2007.

Zohar Feldman and Carmel Domshlak. Monte-Carlo tree search: To MC or to DP? In *European Conference on Artificial Intelligence (ECAI)*, volume 263, page 321. IOS Press, 2014.

Zohar Feldman and Carmel Domshlak. Simple regret optimization in online planning for Markov decision processes. *Journal of Artificial Intelligence Research*, 51:165–205, 2014.

Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *International Conference on Machine Learning (ICML)*, pages 273–280. ACM, 2007.

Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *Conference on Neural Information Processing Systems (NIPS)*, 2006.

Eric A Hansen and Shlomo Zilberstein. LAO$*$: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1):35–62, 2001.

David P Helmbold and Aleatha Parker-Wood. All-moves-as-first heuristics in Monte-Carlo Go. In *International Conference on Artificial Intelligence (ICAI)*, pages 605–610, 2009.

Todd Hester. *TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains.* PhD thesis, The University of Texas at Austin, Austin, Texas, USA, December 2012.

Thomas Keller and Patrick Eyerich. PROST: Probabilistic planning based on UCT. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 119–127. AAAI Press, June 2012.

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

Piyush Khandelwal, Samuel Barrett, and Peter Stone. Leading the way: An efficient multi-robot guidance system. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2015.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, 2006.

George Konidaris, Scott Niekum, and Philip S Thomas. Td_gamma: Re-evaluating complex backups in temporal difference learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2402–2410, 2011.

Yasuhiro Osaki, Kazutomo Shibahara, Yasuhiro Tajima, and Yoshiyuki Kotani. An Othello evaluation function based on temporal difference learning using probability of winning. In *Computational Intelligence and Games (CIG)*, pages 205–211. IEEE, 2008.

Jing Peng and Ronald J Williams. Incremental multi-step Q-learning. *Machine Learning*, 22(1-3):283–290, 1996.

Pierre Perick, David L St-Pierre, Francis Maes, and Damien Ernst. Comparison of different selection strategies in Monte-Carlo tree search for the game of Tron. In *Computational Intelligence and Games (CIG)*, pages 242–249. IEEE, 2012.

Gavin Adrian Rummery. *Problem solving with reinforcement learning*. PhD thesis, University of Cambridge, 1995.

Scott Sanner. Relational dynamic influence diagram language RDDL: Language description. *Unpublished. Australian National University*, 2010.

David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Conference on Neural Information Processing Systems (NIPS)*. 2010.

David Silver, Richard S Sutton, and Martin Müller. Sample-based learning and search with permanent and transient memories. In *International Conference on Machine Learning (ICML)*, pages 968–975. ACM, 2008.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Cambridge University Press, 1998.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

Gerald Tesauro, VT Rajan, and Richard Segal. Bayesian inference in Monte-Carlo tree search. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.

Philip S Thomas, Scott Niekum, Georgios Theocharous, and George Konidaris. Policy evaluation using the Ω-return. In *Advances in Neural Information Processing Systems (NIPS)*, pages 334–342. 2015.

Yizao Wang and Sylvain Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *Computational Intelligence and Games (CIG)*, volume 7, pages 175–182, 2007.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.