Confidence Backup Updates for Aggregating MDP State Values in Monte-Carlo Tree Search

Zahy Bnaya
Center for Neural Science
New York University
zahy.bnaya@gmail.com

Alon Palombo
Department of Information Systems Engineering
Ben-Gurion University
alonpalombo@gmail.com

Rami Puzis
Department of Information Systems Engineering
Ben-Gurion University
faramir.p@gmail.com

Ariel Felner
Department of Information Systems Engineering
Ben-Gurion University
felner@bgu.ac.il

Abstract

Monte-Carlo Tree Search (MCTS) algorithms estimate the value of MDP states based on rewards received by performing multiple random simulations. MCTS algorithms can use different strategies to aggregate these rewards and provide an estimation for the states' values. The most common aggregation method is to store the mean reward of all simulations. Another common approach stores the best observed reward from each state. Both of these methods have complementary benefits and drawbacks. In this paper, we show that both of these methods are biased estimators for the real expected value of MDP states. We propose an hybrid approach that uses the best reward for states with low noise, and otherwise uses the mean. Experimental results on the *Sailing* MDP domain show that our method has a considerable advantage when the rewards are drawn from a noisy distribution.

Introduction and background

The Monte-Carlo Tree Search (MCTS) framework (Browne et al. 2012) was shown to be a successful solving approach for both *Markov Decision Processes* (MDP) and adversarial domains (Gelly and Silver 2007; Ramanujan and Selman 2011; Kocsis and Szepesvári 2006). MCTS algorithms grow asymmetric search trees by repeatedly adding a single node to the frontier of the tree. Nodes are evaluated by aggregating multiple rewards received by sampling the state space. These samples are performed according to a predefined *default policy* which is often noisy and sometimes even completely random. This is common to MCTS applications since MCTS is especially successful with domains lacking strong domain knowledge (e.g., heuristic function). When a sample terminates, a reward is observed and propagated up to the root of the tree. This propagation procedure requires a predefined *backup operator* which determines how a node value should be updated given a new reward.

Recent work (Domshlak and Feldman 2013; Ramanujan and Selman 2011; Keller and Helmert 2013) has discussed two commonly used backup operators. The *Monte-Carlo* (MC) backup operator, as used by the UCT algorithm (Kocsis and Szepesvári 2006), which averages all rewards received through a node and the *Dynamic programming* (DP) backup operator which updates a node value according to the value of their best successor. Both methods are widely used but are known to have drawbacks.

In this paper we suggest a new approach to perform aggregation of rewards. Our method examines the variance in the overall series of rewards observed from a given node and compares it to the individual variances of rewards obtained from each of the successors. We then decide which of these rewards should be used to estimate the value of the node. We discuss our method in the context of MDPs.

Markov Decision Process (MDP) (Puterman 2009) is a well-studied model for sequential decision making under uncertainty. An MDP instance is a tuple $\langle S, A, T, R \rangle$, where S is the set of all states, and $A(s)$ are the sets of actions available from state $s \in S$. The transition model $T(s, a, s')$ represents the probability of moving from state s to state s' by performing action a. An immediate reward $R(s, a)$ is received when performing an action a from state s. The term *horizon* of an MDP denotes the number of consecutive decisions possible until reaching a terminal state. A solution for an MDP is a *policy* that maps each state $s \in S$ to an action $a \in A(s)$ such that a high *expected* reward is received.

The Monte-Carlo Tree Search (MCTS) framework (Browne et al. 2012) was shown to be a successful solving approach for MDPs. The search tree built by MCTS contains two types of nodes, *frontier* nodes and *internal* nodes. Frontier nodes were not sampled yet and their successors are not part of the tree. *Internal* nodes were already sampled and expanded. Initially, the tree only includes the root node as a single frontier node.

Each node m in the tree maintains two variables which are updated in each iteration; (1) current value estimate $v(m)$ and (2) number of observations, $ob(m)$, which is the number of samples that were performed from node m. We denote $v^*(m)$ as the *optimal* expected value of node m.

In MDP, the tree contains interleaved layers of *chance* and *decision* nodes. *Decision* nodes represent states where decision is required and *chance* nodes represent a chosen action.

An iteration of MCTS consists of four steps. **(1) Selection:** a single frontier node in the search tree is selected to be sampled next. This is usually performed by a *tree-descent* procedure starting from the root and ending when the first frontier node is reached and then selected. Implementations of this stage vary. For instance, the UCT algorithm (Kocsis and Szepesvári 2006) uses the UCB for-

mula (Auer, Cesa-Bianchi, and Fischer 2002) (designed for the multi-armed bandit problem) to balance exploration and exploitation when deciding on the next sample to perform. For every internal *decision* node m, UCT selects the successor node n that maximizes the following formula:

$$C\sqrt{\frac{\log\ ob(m)}{ob(n)}} + R(m,a) + v(n)$$

where a is the action that transitions from node m to node n and C is an *exploration constant*. On each *chance* node, UCT selects the next node by drawing from the distribution defined by the transition function T.

(2) Expansion: the selected frontier node n is *expanded*. The successor nodes (denoted as $N(n)$) are generated and added as new frontier nodes to the tree and n becomes an internal node.

(3) Simulation: a *default policy* for selecting moves is executed starting from n, usually until a terminal node t is reached. This is called a *playout*. A value r (the *reward*) associated with the complete *playout* is observed.

(4) Backup: the observed reward r propagates up to n and then up the tree using a *backup operator*. During this phase the values of all internal nodes on the relevant trajectory are updated. This stage is the main focus of our paper.

Backup Operators

The most common backup operator updates the current value estimate, $v(m)$, to be the mean of the entire set of rewards received from within the subtree rooted at m. This value is referred to as the Monte-Carlo mean (*MC*) and it is computed recursively as follows:

$$v(m) = \begin{cases} 0 & \text{m is terminal} \\ \sum\limits_{n \in N(m)} \frac{ob(n)}{ob(m)}[v(n) + \eta(m,n)] & \text{otherwise} \end{cases}$$

Equation 1: MC update

where $\eta(m,n)$ is the reward received in the transition from m to n. If m is a *decision* node, this value is the immediate reward of performing the action a that leads from m to successor node n, i.e., $\eta(m,n) = R(m,a)$. If m is a *chance* node, this term equals 0.

The MC backup operator has drawbacks also presented by others (Bubeck, Munos, and Stoltz 2011; Feldman and Domshlak 2013). Let m be a *decision* node and let a^* be the action that produces the optimal value for m. The optimal value of m is $v^*(m) = R(m,a^*) + v^*(n)$ where $v^*(n)$ is the expected optimal value of the *chance* node to which a^* leads. The MC backup estimates $v(m)$ by also averaging rewards from performing actions other than a^*. These rewards are not drawn from the distribution for which $v^*(m)$ is the expected value and in some cases they could be entirely unrelated. Therefore, although MC averages these rewards, ideally they should be ignored. Usually, MCTS algorithms such as UCT explore moves even when they are probably not optimal. These explorations provide valuable information regarding what is the best action. However, including these rewards in the average value of the node weakens the estimation of the node's value. Note that using the MC backup operator still guarantees that $v(m)$ converges to $v^*(m)$ when used as part of the UCT algorithm since as the number of observations of m increases, the best move (a^*) is visited most of the time. However, convergence could require an extremely large number of iterations, which is the major drawback of using the MC operator.

Bubeck et al. (2011) describes a related problem by showing that the *simple regret* (i.e., expected loss or rewards in a single decision) and *cumulative regret* (i.e., expected loss of rewards over time) are somewhat competing objectives. Feldman et al. (2013) describe a similar idea called "separation of concerns". That is, in MCTS there are two similar (but no identical) goals; learning the best action to perform from a node, and learning the correct value of this node.

An alternative backup operator uses the principle of *Dynamic Programming* (DP). On each update, the DP operator updates $v(m)$ to be the value of its best successor. Given an MDP *decision* node (the value of MDP *chance* nodes are calculated as in MC), DP performs the following update:

$$v(m) = \begin{cases} 0 & \text{m is terminal} \\ \max_{n \in N(m)} v(n) + \eta(m,n) & \text{otherwise} \end{cases}$$

Equation 2: DP update

The DP operator might update the value estimates of the nodes based on promising but "under-sampled" successors, whose value estimations are very far from their actual optimal expected value. This could be extremely misleading, especially when the simulation is based on a random policy and the branching factor is large, which makes sampling even less informative. As with MC, the DP operator is also guaranteed to converge to the optimal solution eventually.

Backup operators as biased estimators

MCTS aims for a minimal difference between the estimated value $v(m)$ and the optimal expected value of the state $v^*(m)$. The value $v(m)$ acts as a statistical estimator for $v^*(m)$. We show that both MC and DP act as *biased* statistical estimators and thus they are, for a large extent, wrong. The *estimation bias* of an estimator $\hat{s}$ is defined as the difference between the expected value of $\hat{s}$ and the expected value it estimates. The bias of an estimator $\hat{s}$ for the value of a MCTS node m is defined as $B[\hat{s}] = E[\hat{s}] - v^*(m)$. When an estimator has a bias of 0 it is called an *unbiased estimator*.

Theorem 1 *Given a set of rewards observed from a decision node m. The MC backup operator $MC(m)$ is a biased estimator for the value of m.*

proof: The value of $MC(m)$ can be expressed as $\sum\limits_{n \in N(m)} \frac{ob(n)}{ob(m)}[MC(n) + \eta(n,m)]$.

The expected value $E[MC(m)]$ is calculated as: [1]

$$\sum_{n \in N(m)} \int \int Pr_{(ob(n))} \frac{ob(n)}{ob(m)} Pr_{(MC(n))}[MC(n) + \eta(n,m)]$$

For simplicity, we assume that *chance* nodes' estimates are unbiased such that $E[MC(n)] = v^*(n)$.

The optimal value of m is $v^*(m) = R(m, a^*) + v^*(n^*)$, where a^* is the optimal action and n^* is the optimal successor node. Since each action needs to be sampled at least once, the probability $Pr(ob(n))Pr(MC(n))$ for any non-optimal actions and successor is nonzero and thus: $B[v(m)] = E[MC(m)] - v^*(n) \neq 0$ Thus, decision nodes calculated via the MC operators are biased estimators. We now show the same for DP operator:

Theorem 2 *Given a set of rewards observed from a decision node m. The DP backup operator $DP(m)$ is a biased estimator for the value of m.*

Proof: $DP(m)$ is calculated as $max_{n \in N(m)}[MC(n) + \eta(m,n)]$.

Again we assume that chance nodes estimators $MC(n)$ are unbiased for all chance nodes values $v^*(n)$. The expected value of $DP(m)$ is calculated as:

$$E[DP(m)] = \sum_{n \in N(m)} Pr_{max}(n)[MC(n) + \eta(m,n)]$$

where $Pr_{max}(n)$ is defined as

$$\prod_{l \in N(m)/\{n\}} Pr[MC(n) > MC(l)]$$

Since there are no guarantees that $Pr_{max}(MC(n))$ is 0 for every non-optimal successor n, $B[DP(d)] = E[DP(d)] - v^*(m) \neq 0$

Confidence DP Backup Operator

We introduce a novel backup operator called the *Confidence Dynamic Programming* backup operator (denoted CDP) which aims to overcome some of the drawbacks of MC and DP backup operators by reducing the estimation bias. On each call CDP first selects a subset of successors with reliable value estimates (relative to the MC value estimate of the parent). Then CDP performs the aggregation only on these reliable values and excludes the rest.

Stable nodes We define the *stable* successors set $N^+(m) \subseteq N(m)$ to be the subset of successors whose rewards are included in the CDP aggregation. Successor nodes not in $N^+(m)$ are called *unstable*.

If $N^+(m) = \emptyset$, none of the successors' estimates are more reliable than $MC(m)$. If, however, *stable* successors exist ($N^+(m) \neq \emptyset$), CDP switches to the DP backup operator and updates $v(m)$ to the value of the *best stable node*.

To keep track of which nodes should go into the *stable* set, we maintain an additional variable, $o(m) \geq 0$ for each node m in the tree, which represents the *level of noise* observed in $v(m)$. Low noise means high confidence. Both $v(m)$ and $o(m)$ are updated by the backup operator in a recursive manner. Terminal nodes and new frontier nodes both have $o(m) = 0$ since in both cases there is only one sample and therefor no noise. $o(m)$ is defined as follows:

$$o(m) = \begin{cases} 0 & ob(m) = 1 \vee terminal(m) \\ Var(m) & otherwise \end{cases}$$

Where $Var(m) = \frac{1}{ob(m)-1} \sum_{i=1}^{ob(m)} [r_i - MC(m)]^2$ is the *sample variance* of rewards observed from node m. In our implementation, we use the *sample variance* as our measure of $o(m)$. However, we expect other forms of $o(m)$ to be successful, but leave that for future work. Calculating the variance of a series of rewards can be done *online* without storing the entire history of rewards (Welford 1962). This makes our method memory efficient (requiring us to store only a single additional variable per node). In fact, in our experiments, the execution time difference between our method and CDP/MC was insignificant.

Stable set selection criteria On each call to back propagation, CDP builds the *stable* successor set. *Stable* nodes $N^+(m)$ are defined as nodes $n \in N(m)$ that satisfy the following two conditions: **(1)** $v(n) + \eta(m,n) > MC(m)$ and **(2)** $o(n) < Var(m)$

Satisfying condition (2) means that the estimated value of the successor $v(n)$ is generally less noisy than that of its parent $v(m)$.

In most cases, at least some *stable* nodes exist. Sampling variance in the individual successors' rewards tend to decrease compared to the variance of the parent's, since the parent's rewards originate from many actions. Additionally, before any update is done, at least one successor has a better average value than the average value of the parent $MC(m)$.

If however, none of the successor nodes are *stable*, we update the estimate of m exactly as in the MC operator. In our experiments we observed that only 10% to 30% of back propagation calls produced an empty *stable* set.

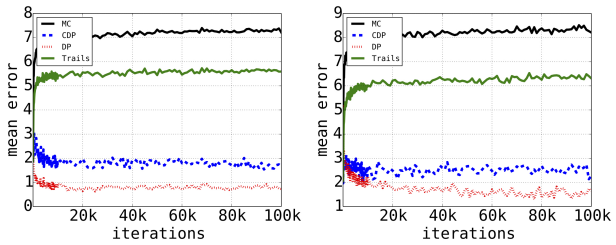
If $N^+(m) \neq \emptyset$, we propagate the values of the best stable node and modify both $v(m)$ and $o(m)$ as follows:

$$v(m) = \max_{n \in N^+(m)} v(n)$$

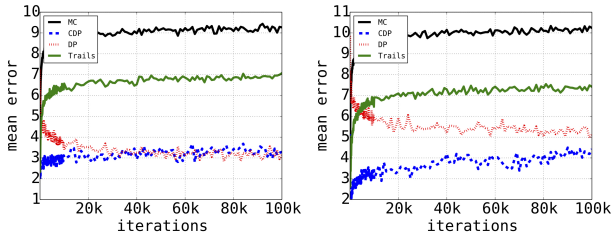
$$o(m) = o(argmax_{n \in N^+(m)} v(n))$$

Related work Perhaps the closest work to ours is by Coulom (2007), who showed an improvement over MC in 9 x 9 *Go*. His suggested backup operator updates nodes to the value of the successor move with the maximum number of visits if it also has the best value. Otherwise, it uses the MC mean. In CDP, we use a more general criteria to decide which successors are to be considered. We base our decision on the noise level instead of on the number of visits. In addition to considering the best move, we consider all moves that have a better value than the MC mean. We implemented Coulom's method (which we denote as *Trails* in the results) and we show that on noisy MDP's, this method

[1] We abbreviate $Pr[ob(n)|ob(m)]$ to $Pr_{(ob(n))}$ for clarity.

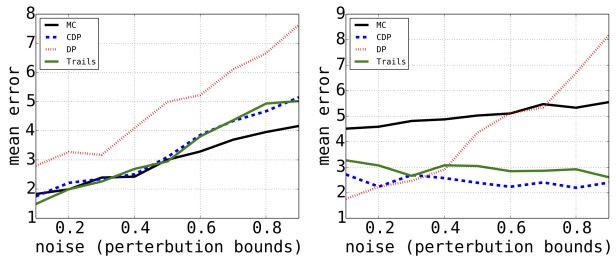


(a) $\epsilon \sim U[0.2, -0.2]$ (b) $\epsilon \sim U[0.5, -0.5]$

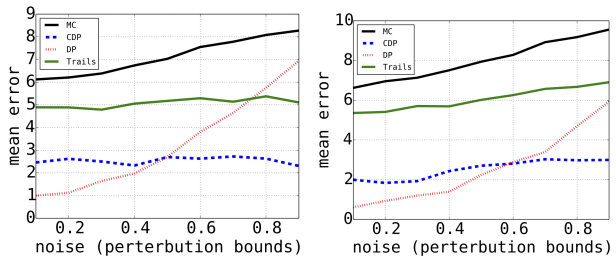


(c) $\epsilon \sim U[0.7, -0.7]$ (d) $\epsilon \sim U[0.9, -0.9]$

Figure 1: Mean error as a function of iterations.



(a) $iterations = 10$ (b) $iterations = 100$



(c) $iterations = 1000$ (d) $iterations = 10000$

Figure 2: Mean error as a function of perturbation bounds (b)

is not as successful as in *Go*, although it also reduces some of the estimation bias when compared to MC.

Experimental results

We performed empirical experiments demonstrating that our CDP operator reduces some of the estimation bias compared to MC, DP and Coulom's *Trails* backup operators. We construct multiple MCTS trees (using the UCT algorithm) and then estimate the root node value using one of the four backup operators. We also "off-line" calculated the exact expected value of each built tree. We report the mean error achieved by each propagation method.

We performed our experiments on the *sailing* domain, a simple MDP where a sailboat needs to find the shortest expected path between two points in a grid under fluctuating wind conditions. We used the same settings as described in (Kocsis and Szepesvári 2006). We tried different grid sizes and got similar results. The results reported in this paper are for 10×10 grids.

We found that the success of CDP depends on the level of "randomness" in the observed *playout* rewards. Thus, we experiment with several *default policy* settings that generate different series of rewards, with different noise levels. We define a general *playout* default policy Π_k^b as follows. First, Π_k^b performs k random steps, where k is drawn from a geometric distribution. Then, the optimal expected value of the k^{th} node, $v^*(n_k)$ is calculated "off-line" using the *value iteration* algorithm (Bellman 1957). A *perturbation rate* ϵ is drawn from a uniform distribution $\epsilon \sim U[-b, +b]$. Finally Π_k^b returns the reward $(1+\epsilon)v^*(n_k)$. We experiment with different uniform distribution bounds $\pm b$ from which ϵ is drawn. This allows us to examine CDP performance with different levels of noise in the rewards series. [2]

We execute each method on 300 instances for each data point we report. Figure 1 shows the mean error on different perturbation rates. On low noise levels, MC and DP generate less average error. However, when the level of noise increases ($b \geq \pm\ 0.7$), our CDP method is able to withstand it better, and produces more accurate results. Note that although Figure 1 shows a temporary increase in error levels, all backup operators in our experiments eventually converged. Figure 2 demonstrates CDP mean error levels in relation to noise level. On considerably small number of iterations, CDP does not have any advantage. However, when the number of iterations increases, CDP eliminates some of the estimation bias in comparison to the other methods. CDP requires additional book-keeping and comparisons. Thus, our analysis would not be complete without timing the different operators. We timed the execution of the four backup operators and we could not see a significant difference in the execution time for our implementations.

Conclusion and future work

We proposed a novel backup operator for MCTS that performs dynamic programming updates on a limited set of successors. We believe that our method has the potential of combining the advantages of DP and MC and reducing estimation bias to some extent. Empirical evaluation on finite-horizon MDP shows the advantage of our method over both DP and MC backup operators. Our method shows considerable advantage when the *playout* default policy generates noisy rewards, which is a very common condition for many

[2] We tried three strategies to update the level of noise. (1)*Full-update:* which performs $o(m) = o(argmax_{n \in N^+(m)} v(n))$. (2) *Partial-update* which performs $o(m) = \delta \times o(argmax_{n \in N^+(m)} v(n))$ where δ is a function of the depth of node m and (3)*no-update* where $o(m) = Var(m)$. All three strategies performed similarly and we only report results for *no-update* strategy because of its simplicity.

MCTS applications. Our results look promising and there are many possible future work directions. One direction is to explore other measurements for the noise level $o(m)$ and other definitions for the *stable* set. Another direction is to apply CDP on game domains and especially on *General Game Playing* (GGP), which seems natural to this approach because of the lack in domain knowledge and noisy rewards.

Acknowledgements

The research was supported by Israel Science Foundation (ISF) under grant #417/13 to Ariel Felner.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* 47(2-3):235–256.

Bellman, R. 1957. A markovian decision process. Technical report, DTIC Document.

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. *Computational Intelligence and AI in Games, IEEE Transactions on* 4(1):1–43.

Bubeck, S.; Munos, R.; and Stoltz, G. 2011. Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science* 412(19):1832–1852.

Coulom, R. 2007. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*. Springer. 72–83.

Domshlak, C., and Feldman, Z. 2013. To uct, or not to uct?(position paper).

Feldman, Z., and Domshlak, C. 2013. Monte-carlo planning: Theoretically fast convergence meets practical efficiency. *arXiv preprint arXiv:1309.6828*.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, 273–280. ACM.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon mdps. In *ICAPS*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Puterman, M. L. 2009. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. John Wiley & Sons.

Ramanujan, R., and Selman, B. 2011. Trade-offs in sampling-based adversarial planning. In *ICAPS*.

Welford, B. 1962. Note on a method for calculating corrected sums of squares and products. *Technometrics* 4(3):419–420.