Feedback-Based Tree Search for Reinforcement Learning

Daniel R. Jiang [1] **Emmanuel Ekwedike** [2 3] **Han Liu** [2 4]

Abstract

Inspired by recent successes of Monte-Carlo tree search (MCTS) in a number of artificial intelligence (AI) application domains, we propose a reinforcement learning (RL) technique that iteratively applies MCTS on batches of small, finite-horizon versions of the original infinite-horizon Markov decision process. The terminal condition of the finite-horizon problems, or the *leaf-node evaluator* of the decision tree generated by MCTS, is specified using a combination of an estimated value function and an estimated policy function. The recommendations generated by the MCTS procedure are then provided as feedback in order to refine, through classification and regression, the leaf-node evaluator for the next iteration. We provide the first sample complexity bounds for a tree search-based RL algorithm. In addition, we show that a deep neural network implementation of the technique can create a competitive AI agent for the popular multi-player online battle arena (MOBA) game *King of Glory*.

1. Introduction

Monte-Carlo tree search (MCTS), introduced in Coulom (2006) and surveyed in detail by Browne et al. (2012), has received attention in recent years for its successes in gameplay artificial intelligence (AI), culminating in the Go-playing AI AlphaGo (Silver et al., 2016). MCTS seeks to iteratively build the decision tree associated with a given Markov decision process (MDP) so that attention is focused on "important" areas of the state space, assuming a given initial state (or *root node* of the decision tree). The intuition behind MCTS is that if rough estimates of state or action values are given, then it is only necessary to expand the decision tree in the direction of states and actions with high estimated value. To accomplish this, MCTS utilizes the guidance of *leaf-node evaluators* (either a policy function (Chaslot et al., 2006) rollout, a value function evaluation (Campbell et al., 2002; Enzenberger, 2004), or a mixture of both (Silver et al., 2016)) to produce estimates of downstream values once the tree has reached a certain depth (Browne et al., 2012). The information from the leaf-nodes are then *backpropagated* up the tree. The performance of MCTS depends heavily on the quality of the policy/value approximations (Gelly & Silver, 2007), and at the same time, the successes of MCTS in Go show that MCTS *improves upon a given policy* when the policy is used for leaf evaluation, and in fact, it can be viewed as a policy improvement operator (Silver et al., 2017). In this paper, we study a new feedback-based framework, wherein MCTS updates its own leaf-node evaluators using observations generated at the root node.

MCTS is typically viewed as an *online planner*, where a decision tree is built starting from the current state as the root node (Chaslot et al., 2006; 2008; Hingston & Masek, 2007; Maîtrepierre et al., 2008; Cazenave, 2009; Méhat & Cazenave, 2010; Gelly & Silver, 2011; Gelly et al., 2012; Silver et al., 2016). The standard goal of MCTS is to recommend an action for the *root node only*. After the action is taken, the system moves forward and a new tree is created from the next state (statistics from the old tree may be partially saved or completely discarded). MCTS is thus a "local" procedure (in that it only returns an action for a given state) and is inherently different from value function approximation or policy function approximation approaches where a "global" policy (one that contains policy information about all states) is built. In real-time decision-making applications, it is more difficult to build an adequate "on-the-fly" local approximation than it is to use pre-trained global policy in the short amount of time available for decision-making. For games like Chess or Go, online planning using MCTS may be appropriate, but in games where fast decisions are necessary (e.g., Atari or MOBA video games), tree search methods are too slow (Guo et al., 2014). The proposed algorithm is intended to be used in an *off-policy* fashion during the reinforcement learning (RL) *training phase*. Once the training is complete, the policies associated with leaf-node evaluation can be implemented to make fast, real-time decisions without any further need for tree search.

Main Contributions. These characteristics of MCTS motivate our proposed method, which attempts to leverage the

[1]University of Pittsburgh [2]Tencent AI Lab [3]Princeton University [4]Northwestern University. Correspondence to: Daniel R. Jiang <drjiang@pitt.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

local properties of MCTS into a training procedure to iteratively build *global* policy across all states. The idea is to apply MCTS on batches of small, finite-horizon versions of the original infinite-horizon Markov decision process (MDP). A rough summary is as follows: (1) initialize an arbitrary value function and a policy function; (2) start (possibly in parallel) a batch of MCTS instances, limited in search-depth, initialized from a set of sampled states, while incorporating a combination of the value and policy function as leaf-node evaluators; (3) update both the value and policy functions using the latest MCTS root node observations; (4) Repeat starting from step (2). This method exploits the idea that an MCTS policy is better than either of the leaf-node evaluator policies alone (Silver et al., 2016), yet improved leaf-node evaluators also improve the quality of MCTS (Gelly & Silver, 2007). The primary contributions of this paper are summarized below.

1. We propose a batch, MCTS-based RL method that operates on continuous state, finite action MDPs and exploits the idea that leaf-evaluators can be updated to produce a stronger tree search using *previous tree search results*. Function approximators are used to track policy and value function approximations, where the latter is used to reduce the length of the tree search rollout (oftentimes, the rollout of the policy becomes a computational bottle-neck in complex environments).
2. We provide a full sample complexity analysis of the method and show that with large enough sample sizes and sufficiently large tree search effort, the performance of the estimated policies can be made close to optimal, up to some unavoidable approximation error. To our knowledge, batch MCTS-based RL methods have not been theoretically analyzed.
3. An implementation of the feedback-based tree search algorithm using deep neural networks is tested on the recently popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`). The result is a competitive AI agent for the 1v1 mode of the game.

2. Related Work

The idea of leveraging tree search during training was first explored by Guo et al. (2014) in the context of Atari games, where MCTS was used to generate offline training data for a supervised learning (classification) procedure. The authors showed that by using the power of tree search offline, the resulting policy was able to outperform the deep Q-network (DQN) approach of (Mnih et al., 2013). A natural next step is to repeatedly apply the procedure of Guo et al. (2014). In building AlphaGo Zero, Silver et al. (2017) extends the ideas of Guo et al. (2014) into an iterative procedure, where the neural network policy is updated after every episode and then reincorporated into tree search. The technique was able to produce a superhuman Go-playing AI (and improves upon the previous AlphaGo versions) without any human replay data.

Our proposed algorithm is a *provably near-optimal* variant (and in some respects, generalization) of the AlphaGo Zero algorithm. The key differences are the following: (1) our theoretical results cover a continuous, rather than finite, state space setting, (2) the environment is a stochastic MDP rather than a sequential deterministic two player game, (3) we use batch updates, (4) the feedback of previous results to the leaf-evaluator manifests as both policy and value updates rather than just the value (as Silver et al. (2017) does not use policy rollouts).

Anthony et al. (2017) proposes a general framework called *expert iteration* that combines supervised learning with tree search-based planning. The methods described in Guo et al. (2014), Silver et al. (2017), and the current paper can all be (at least loosely) expressed under the expert iteration framework. However, no theoretical insights were given in any of these previous works and our paper intends to fill this gap by providing a full theoretical analysis of an iterative, MCTS-based RL algorithm. Our analysis relies on the *concentrability coefficient* idea of Munos (2007) for approximate value iteration and builds upon the work on classification based policy iteration (Lazaric et al., 2016), approximate modified policy iteration (Scherrer et al., 2015), and fitted value iteration (Munos & Szepesvári, 2008).

Sample complexity results for MCTS are relatively sparse. Teraoka et al. (2014) gives a high probability upper bound on the number of playouts needed to achieve ϵ-accuracy at the root node for a stylized version of MCTS called `FindTopWinner`. More recently, Kaufmann & Koolen (2017) provided high probability bounds on the sample complexity of two other variants of MCTS called `UGapE-MCTS` and `LUCB-MCTS`. In this paper, we do not require any particular implementation of MCTS, but make a generic assumption on its accuracy that is inspired by these results.

3. Problem Formulation

Consider a discounted, infinite-horizon MDP with a continuous state space $\mathcal{S}$ and finite action space $\mathcal{A}$. For all $(s,a) \in \mathcal{S} \times \mathcal{A}$, the *reward function* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ satisfies $r(s,a) \in [0, R_{\max}]$. The *transition kernel*, which describes transitions to the next state given current state s and action a, is written $p(\cdot \,|\, s,a)$ — a probability measure over $\mathcal{S}$. Given a *discount factor* $\gamma \in [0,1)$, the value function V^π of a *policy* $\pi : \mathcal{S} \to \mathcal{A}$ starting in $s = s_0 \in \mathcal{S}$ is given by

$$V^\pi(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, \pi_t(s_t))\right], \qquad (1)$$

where s_t is the state visited at time t. Let Π be the set of all stationary, deterministic policies (i.e., mappings from state to action). The *optimal value function* is obtained by maximizing over all policies: $V^*(s) = \sup_{\pi \in \Pi} V^\pi(s)$.

Both V^π and V^* are bounded by $V_{\max} = R_{\max}/(1-\gamma)$. We let $\mathcal{F}$ be the set of bounded, real-valued functions mapping $\mathcal{S}$ to $[0, V_{\max}]$. We frequently make use of the shorthand operator $T_\pi : \mathcal{F} \to \mathcal{F}$, where the quantity $(T_\pi V)(s)$ is be interpreted as the reward gained by taking an action according to π, receiving the reward $r(s, \pi(s))$, and then receiving an expected terminal reward according to the argument V:

$$(T_\pi V)(s) = r(s, \pi(s)) + \gamma \int_{\mathcal{S}} V(\tilde{s})\, p(d\tilde{s}\,|\,s, \pi(s)).$$

It is well-known that V^π is the unique fixed-point of T_π, meaning $T_\pi V^\pi = V^\pi$ (Puterman, 2014). The *Bellman operator* $T : \mathcal{F} \to \mathcal{F}$ is similarly defined using the maximizing action:

$$(TV)(s) = \max_{a \in \mathcal{A}} \Big[r(s,a) + \gamma \int_{\mathcal{S}} V(\tilde{s})\, p(d\tilde{s}\,|\,s, a) \Big].$$

It is also known that V^* is the unique fixed-point of T (Puterman, 2014) and that acting greedily with respect to the optimal value function V^* produces an *optimal policy*:

$$\pi^*(s) \in \arg\max_{a \in A} \Big[r(s,a) + \gamma \int_{\mathcal{S}} V^*(\tilde{s})\, p(d\tilde{s}\,|\,s, a) \Big].$$

We use the notation T^d to mean the d compositions of the mapping T, e.g., $T^2 V = T(TV)$. Lastly, let $V \in \mathcal{F}$ and let ν be a distribution over $\mathcal{S}$. We define left and right versions of an operator P_π:

$$(P_\pi V)(s) = \int_{\mathcal{S}} V(\tilde{s})\, p(d\tilde{s}\,|\,s, \pi(s)),$$

$$(\nu P_\pi)(d\tilde{s}) = \int_{\mathcal{S}} p(d\tilde{s}\,|\,s, \pi(s))\, \nu(ds).$$

Note that $P_\pi V \in \mathcal{F}$ and μP_π is another distribution over $\mathcal{S}$.

4. Feedback-Based Tree Search Algorithm

We now formally describe the proposed algorithm. The parameters are as follows. Let $\bar{\Pi} \subseteq \Pi$ be a space of approximate policies and $\bar{\mathcal{F}} \subseteq \mathcal{F}$ be a space of approximate value functions (e.g., classes of neural network architectures). We let $\pi_k \in \bar{\Pi}$ be the policy function approximation (PFA) and $V_k \in \bar{\mathcal{F}}$ be the value function approximation (VFA) at iteration k of the algorithm. Parameters subscripted with '0' are used in the value function approximation (regression) phase and parameters subscripted with '1' are used in the tree search phase. The full description of the procedure is given in Figure 1, using the notation $T_a = T_{\pi_a}$, where π_a maps all states to the action $a \in \mathcal{A}$. We now summarize the two phases, VFA (Steps 2 and 3) and MCTS (Steps 4, 5, and 6).

VFA Phase. Given a policy π_k, we wish to approximate its value by fitting a function using subroutine `Regress` on N_0 states sampled from a distribution ρ_0. Each call to `MCTS` requires repeatedly performing rollouts that are initiated from leaf-nodes of the decision tree. Because repeating full rollouts during tree search is expensive, the idea is that a VFA obtained from a one-time regression on a single set of rollouts can drastically reduce the computation needed for `MCTS`. For each sampled state s, we estimate its value using M_0 full rollouts, which can be obtained using the absorption time formulation of an infinite horizon MDP (Puterman, 2014, Proposition 5.3.1).

MCTS Phase. On every iteration k, we sample a set of N_1 i.i.d. states from a distribution ρ_1 over $\mathcal{S}$. From each state, a tree search algorithm, denoted `MCTS`, is executed for M_1 iterations on a search tree of maximum depth d. We assume here that the leaf evaluator is a general function of the PFA and VFA from the previous iteration, π_k and V_k, and it is denoted as a "subroutine" `LeafEval`. The results of the `MCTS` procedure are piped into a subroutine `Classify`, which fits a new policy π_{k+1} using classification (from continuous states to discrete actions) on the new data. As discussed more in Assumption 4, `Classify` uses L_1 observations (one-step rollouts) to compute a loss function.

1. Sample a set of N_0 i.i.d. states $\mathcal{S}_{0,k}$ from ρ_0 and N_1 i.i.d. states $\mathcal{S}_{1,k}$ from ρ_1.
2. Compute a sample average $\hat{Y}_k(s)$ of M_0 independent rollouts of π_k for each $s \in \mathcal{S}_{0,k}$. See Assumption 1.
3. Use `Regress` on the set $\{\hat{Y}_k(s) : s \in \mathcal{S}_{0,k}\}$ to obtain a value function $V_k \in \bar{\mathcal{F}}$. See Assumption 1.
4. From each $s \in \mathcal{S}_{1,k}$, run `MCTS` with parameters M_1, d, and evaluator `LeafEval`. Return estimated value of each s, denoted $\hat{U}_k(s)$. See Assumptions 2 and 3.
5. For each $s \in \mathcal{S}_{1,k}$ and $a \in \mathcal{A}$, create estimate $\hat{Q}_k(s,a) \approx (T_a V_k)(s)$ by averaging L_1 transitions from $p(\cdot\,|\,s,a)$. See Assumption 4.
6. Use `Classify` to solve a cost-sensitive classification problem and obtain the next policy $\pi_{k+1} \in \bar{\Pi}$. Costs are measured using $\{\hat{U}_k(s) : s \in \mathcal{S}_{1,k}\}$ and $\{\hat{Q}_k(s, \pi_{k+1}(s)) : s \in \mathcal{S}_{1,k}\}$. See Assumption 4. Increment k and return to Step 1.

Figure 1. Feedback-Based Tree Search Algorithm

The illustration given in Figure 2 shows the interactions (and feedback loop) of the basic components of the algorithm: (1) a set of tree search runs initiated from a batch of sampled states (triangles), (2) leaf evaluation using π_k and V_k is used during tree search, and (3) updated PFA and VFA π_{k+1} and

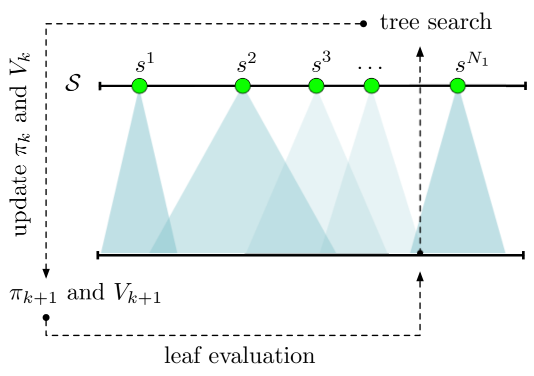

Figure 2. Illustration of the Feedback Loop

V_{k+1} using tree search results.

5. Assumptions

Figure 1 shows the algorithm written with general subroutines `Regress`, `MCTS`, `LeafEval`, and `Classify`, allowing for variations in implementation suited for different problems. However, our analysis assumes specific choices and properties of these subroutines, which we describe now. The regression step solves a least absolute deviation problem to minimize an empirical version of

$$\|f - V^{\pi_k}\|_{1,\rho_0} = \int_{\mathcal{S}} |f(s) - V^{\pi_k}(s)|\rho_0(ds),$$

as described in the first assumption.

Assumption 1 (`Regress` Subroutine)**.** *For each $s^i \in \mathcal{S}_{0,k}$, define $s^i = s_0^{ij}$ for all j and for each t, the state s_{t+1}^{ij} is drawn from $p(\cdot\,|\,s_t^{ij}, \pi_k(s_t^{ij}))$. Let $\hat{Y}_k(s^i)$ be an estimate of $V^{\pi_k}(s^i)$ using M_0 rollouts and V_k, the VFA resulting from* `Regress`*, obtained via least absolute deviation regression:*

$$\hat{Y}_k(s_0^i) = \frac{1}{M_0} \sum_{j=1}^{M_0} \sum_{t=0}^{\infty} \gamma^t\, r(s_t^{ij}, \pi_k(s_t^{ij})), \tag{2}$$

$$V_k \in \operatorname*{arg\,min}_{f \in \bar{\mathcal{F}}} \frac{1}{N_0} \sum_{i=1}^{N_0} \big|f(s^i) - \hat{Y}_k(s^i)\big|. \tag{3}$$

There are many ways that `LeafEval` may be defined. The standard leaf evaluator for MCTS is to simulate a default or "rollout" policy (Browne et al., 2012) until the end of the game, though in related tree search techniques, authors have also opted for a value function approximation (Campbell et al., 2002; Enzenberger, 2004). It is also possible to combine the two approximations: Silver et al. (2016) uses a weighted combination of a full rollout from a pre-trained policy and a pre-trained value function approximation.

Assumption 2 (`LeafEval` Subroutine)**.** *Our approach uses a partial rollout of length $h \geq 0$ and a value estimation at the end.* `LeafEval` *produces unbiased observations of*

$$J_k(s) = \mathbf{E}\left[\sum_{t=0}^{h-1} \gamma^t\, r(\tilde{s}_t, \pi_k(\tilde{s}_t)) + \gamma^h\, V_k(\tilde{s}_h)\right], \tag{4}$$

where $\tilde{s}_0 = s$.

Assumption 2 is motivated by our MOBA game, on which we observed that even short rollouts (as opposed to simply using a VFA) are immensely helpful in determining local outcomes (e.g., dodging attacks, eliminating minions, health regeneration). At the same time, we found that numerous full rollouts simulated using the relatively slow and complex game engine is far too time-consuming within tree search.

We also need to make an assumption on the sample complexity of `MCTS`, of which there are many possible variations (Chaslot et al., 2006; Coulom, 2006; Kocsis & Szepesvári, 2006; Gelly & Silver, 2007; Couëtoux et al., 2011a;b; Al-Kanj et al., 2016; Jiang et al., 2017). Particularly relevant to our continuous-state setting are tree expansion techniques called *progressive widening* and *double progressive widening*, proposed in Couëtoux et al. (2011a), which have proven successful in problems with continuous state/action spaces. To our knowledge, analysis of the sample complexity is only available for stylized versions of MCTS on finite problems, like Teraoka et al. (2014) and Kaufmann & Koolen (2017). Theorems from these papers show upper bounds on the number of iterations needed so that with high probability (greater than $1-\delta$), the value at the root node is accurate within a tolerance of ϵ. Fortunately, there are ways to discretize continuous state MDPs that enjoy error guarantees, such as Bertsekas (1975), Dufour & Prieto-Rumeau (2012), or Saldi et al. (2017). These error bounds can be combined with the MCTS guarantees of Teraoka et al. (2014) and Kaufmann & Koolen (2017) to produce a sample complexity bound for MCTS on continuous problems. The next assumption captures the essence of these results (and if desired, can be made precise for specific implementations through the references above).

Assumption 3 (`MCTS` Subroutine)**.** *Consider a d-stage, finite-horizon subproblem of (1) with terminal value function J and initial state is s. Let the result of* `MCTS` *be denoted $\hat{U}(s)$. We assume that there exists a function $m(\epsilon, \delta)$, such that if $m(\epsilon, \delta)$ iterations of* `MCTS` *are used, the inequality $|\hat{U}(s) - (T^d J)(s)| \leq \epsilon$ holds with probability at least $1-\delta$.*

Now, we are ready to discuss the `Classify` subroutine. Our goal is to select a policy $\pi \in \bar{\Pi}$ that *closely mimics* the performance of the `MCTS` result, similar to practical implementations in existing work (Guo et al., 2014; Silver et al., 2017; Anthony et al., 2017). The question is: given a candidate π, how do we measure "closeness" to the `MCTS` policy? We take inspiration from previous work in classification-based RL and use a cost-based penalization of classification errors (Langford & Zadrozny, 2005; Li et al., 2007; Lazaric et al., 2016). Since $\hat{U}(s^i)$ is an approximation of the performance of the MCTS policy, we should try to select a policy π with similar performance. To estimate the performance of some candidate policy π, we use a one-step rollout and evaluate the downstream cost using V_k.

Assumption 4 (Classify Subroutine)**.** *For each $s^i \in \mathcal{S}_{1,k}$ and $a \in \mathcal{A}$, let $\hat{Q}_k(s^i, a)$ be an estimate of the value of state-action pair (s^i, a) using L_1 samples.*

$$\hat{Q}_k(s^i, a) = \frac{1}{L_1} \sum_{j=1}^{L_1} \bigl[r(s^i, a) + \gamma V_k(\tilde{s}^j(a))\bigr].$$

Let π_{k+1}, the result of Classify*, be obtained by minimizing the discrepancy between the* MCTS *result $\hat{U}_k$ and the estimated value of the policy under approximations $\hat{Q}_k$:*

$$\pi_{k+1} \in \operatorname*{arg\,min}_{\pi \in \bar{\Pi}} \frac{1}{N_1} \sum_{i=1}^{N_1} \bigl|\hat{U}_k(s^i) - \hat{Q}_k(s^i, \pi(s^i))\bigr|,$$

where $\tilde{s}^j(a)$ are i.i.d. samples from $p(\cdot \,|\, s^i, a)$.

An issue that arises during the analysis is that even though we can control the distribution from which states are sampled, this distribution is transformed by the transition kernel of the policies used for rollout/lookahead. Let us now introduce the *concentrability coefficient* idea of Munos (2007) (and used subsequently by many authors, including Munos & Szepesvári (2008), Lazaric et al. (2016), Scherrer et al. (2015), and Haskell et al. (2016)).

Assumption 5 (Concentrability)**.** *Consider any sequence of m policies $\mu_1, \mu_2, \ldots, \mu_m \in \Pi$. Suppose we start in distribution ν and that the state distribution attained after applying the m policies in succession, $\nu P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}$, is absolutely continuous with respect to ρ_1. We define an m-step concentrability coefficient*

$$A_m = \sup_{\mu_1, \ldots, \mu_m} \left\| \frac{d\nu P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}}{d\rho_1} \right\|_\infty,$$

and assume that $\sum_{i,j=0}^{\infty} \gamma^{i+j} A_{i+j} < \infty$. Similarly, we assume $\rho_1 P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}$, is absolutely continuous with respect to ρ_0 and assume that

$$A'_m = \sup_{\mu_1, \ldots, \mu_m} \left\| \frac{d\rho_1 P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}}{d\rho_0} \right\|_\infty$$

is finite for any m.

The concentrability coefficient describes how the state distribution changes after m steps of arbitrary policies and how it relates to a given reference distribution. Assumptions 1-5 are used for the remainder of the paper.

6. Sample Complexity Analysis

Before presenting the sample complexity analysis, let us consider an algorithm that generates a sequence of policies $\{\pi_0, \pi_1, \pi_2, \ldots\}$ satisfying $T_{\pi_{k+1}} T^{d-1} V^{\pi_k} = T^d V^{\pi_k}$ with no error. It is proved in Bertsekas & Tsitsiklis (1996, pp. 30-31) that $\pi_k \to \pi^*$ in the finite state and action setting. Our proposed algorithm in Figure 1 can be viewed as *approximately satisfying* this iteration in a continuous state space setting, where MCTS plays the role of T^d and evaluation of π_k uses a combination of accurate rollouts (due to Classify) and fast VFA evaluations (due to Regress). The sample complexity analysis requires the effects of all errors to be systematically analyzed.

For some $K \geq 0$, our goal is to develop a high probability upper bound on the *expected suboptimality, over an initial state distribution* ν, of the performance of policy π_K, written as $\|V^* - V^{\pi_K}\|_{1,\nu}$. Because there is no requirement to control errors with probability one, bounds in $\|\cdot\|_{1,\nu}$ tend to be much more useful in practice than ones in the traditional $\|\cdot\|_\infty$. Notice that:

$$\begin{aligned} \frac{1}{N_1} \sum_{i=1}^{N_1} \bigl|\hat{U}_k(s^i) - \hat{Q}_k(s^i, \pi_{k+1}(s^i))\bigr| \\ \approx \bigl\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\bigr\|_{1,\rho_1}, \end{aligned} \tag{5}$$

where the left-hand-side is the loss function used in the classification step from Assumption 4. It turns out that we can relate the right-hand-side (albeit under a different distribution) to the expected suboptimality after K iterations $\|V^* - V^{\pi_K}\|_{1,\nu}$, as shown in the following lemma. Full proofs of all results are given in the supplementary material.

Lemma 1 (Loss to Performance Relationship)**.** *The expected suboptimality of π_K can be bounded as follows:*

$$\begin{aligned} \|V^* - V^{\pi_K}\|_{1,\nu} \leq \; & \gamma^{Kd} \, \|V^* - V^{\pi_0}\|_\infty \\ & + \sum_{k=1}^{K} \gamma^{(K-k)d} \, \bigl\|T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}}\bigr\|_{1,\Lambda_{\nu,k}} \end{aligned}$$

where $\Lambda_{\nu,k} = \nu\,(P_{\pi^*})^{(K-k)d}\,\bigl[I - (\gamma P_{\pi_k})\bigr]^{-1}$.

From Lemma 1, we see that the expected suboptimality at iteration K can be upper bounded by the suboptimality of the initial policy π_0 (in maximum norm) plus a discounted and re-weighted version of $\|T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}}\|_{1,\rho_1}$ accumulated over prior iterations. Hypothetically, if $(T^d V^{\pi_{k-1}})(s) - (T_{\pi_k} V^{\pi_{k-1}})(s)$ were small for all iterations k and all states s, then the suboptimality of π_K converges linearly to zero. Hence, we may refer to $\|T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}}\|_{1,\rho_1}$ as the "true loss," the target term to be minimized at iteration k. We now have a starting point for the analysis: if (5) can be made precise, then the result can be combined with Lemma 1 to provide an explicit

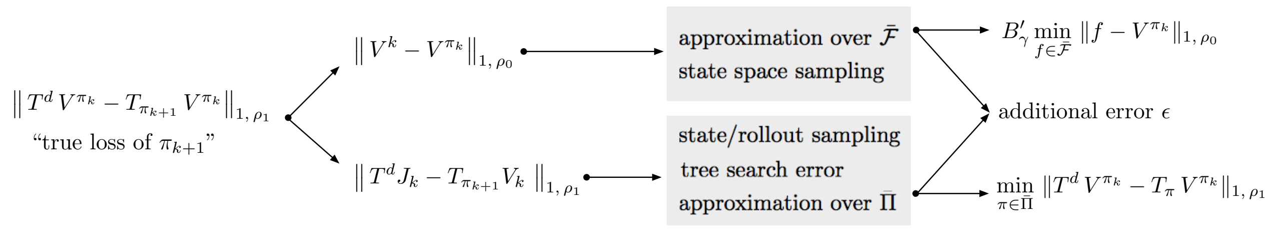
$\| T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k} \|_{1,\rho_1}$
"true loss of π_{k+1}"
$\| V^k - V^{\pi_k} \|_{1,\rho_0}$
$\| T^d J_k - T_{\pi_{k+1}} V_k \|_{1,\rho_1}$
approximation over $\bar{\mathcal{F}}$
state space sampling
state/rollout sampling
tree search error
approximation over $\bar{\Pi}$
$B'_\gamma \min_{f \in \bar{\mathcal{F}}} \|f - V^{\pi_k}\|_{1,\rho_0}$
additional error ϵ
$\min_{\pi \in \bar{\Pi}} \|T^d V^{\pi_k} - T_\pi V^{\pi_k}\|_{1,\rho_1}$

Figure 3. Various Errors Analyzed in Lemma 3

bound on $\|V^* - V^{\pi_K}\|_{1,\nu}$. The various errors that we incur when relating the objective of `Classify` to the true loss include the error due to regression using functions in $\bar{\mathcal{F}}$; the error due to sampling the state space according to ρ_1; the error of estimating $(T_\pi V_k)(s)$ using the sample average of one-step rollouts $\hat{Q}_k(s, \pi(s))$; and of course, the error due to `MCTS`.

We now give a series of lemmas that help us carry out the analysis. In the algorithmic setting, the policy π_k is a random quantity that depends on the samples collected in previous iterations; however, for simplicity, the lemmas that follow are stated from the perspective of a fixed policy μ or fixed value function approximation V rather than π_k or V_k. Conditioning arguments will be used when invoking these lemmas (see supplementary material).

Lemma 2 (Propagation of VFA Error)**.** *Consider a policy $\mu \in \Pi$ and value function $V \in \mathcal{F}$. Analogous to (4), let $J = T^h_\mu V$. Then, under Assumption 5, we have the bounds:*

(a) $\sup_{\pi \in \bar{\Pi}} \|T_\pi V - T_\pi V^\mu\|_{1,\rho_1} \le \gamma A'_1 \|V - V^\mu\|_{1,\rho_0}$,

(b) $\|T^d J - T^d V^\mu\|_{1,\rho_1} \le \gamma^{d+h} A'_{d+h} \|V - V^\mu\|_{1,\rho_0}$.

The lemma above addresses the fact that instead of using V^{π_k} directly, `Classify` and `MCTS` only have access to the estimates V_k and $J_k = T^h_{\pi_k} V_k$ (h steps of rollout with an evaluation of V_k at the end), respectively. Note that propagation of the error in V_k is discounted by γ or γ^{d+h} and since the lemma converts between $\|\cdot\|_{1,\rho_1}$ and $\|\cdot\|_{1,\rho_0}$, it is also impacted by the concentrability coefficients A'_1 and A'_{d+h}.

Let $d_{\bar{\Pi}}$ be the *VC-dimension* of the class of binary classifiers $\bar{\Pi}$ and let $d_{\bar{\mathcal{F}}}$ be the *pseudo-dimension* of the function class $\bar{\mathcal{F}}$. The VC-dimension is a measure of the *capacity* of $\bar{\Pi}$ and the notion of a pseudo-dimension is a generalization of the VC-dimension to real-valued functions (see, e.g., Pollard (1990), Haussler (1992), Mohri et al. (2012) for definitions of both). Similar to Lazaric et al. (2016) and Scherrer et al. (2015), we will present results for the case of two actions, i.e., $|\mathcal{A}| = 2$. The extension to multiple actions is possible by performing an analysis along the lines of Lazaric et al. (2016, Section 6). We now quantify the error illustrated in Figure 3. Define the quantity $B'_\gamma = \gamma A'_1 + \gamma^{d+h} A'_{d+h}$, the sum of the coefficients from Lemma 2.

Lemma 3. *Suppose the regression sample size N_0 is*

$$\mathcal{O}\left((V_{\max} B'_\gamma)^2 \, \epsilon^{-2} \left[\log(1/\delta) + d_{\bar{\mathcal{F}}} \log(V_{\max} B'_\gamma/\epsilon)\right]\right)$$

and the sample size M_0, for estimating the regression targets, is

$$\mathcal{O}\left((V_{\max} B'_\gamma)^2 \, \epsilon^{-2} \left[\log(N_0/\delta)\right]\right).$$

Furthermore, there exist constants C_1, C_2, C_3, and C_4, such that if N_1 and L_1 are large enough to satisfy

$$N_1 \ge C_1 V_{\max}^2 \, \epsilon^{-2} \left[\log(C_2/\delta) + d_{\bar{\Pi}} \log(e N_1/d_{\bar{\Pi}})\right],$$
$$L_1 \ge C_1 V_{\max}^2 \, \epsilon^{-2} \left[\log(C_2 N_1/\delta) + d_{\bar{\Pi}} \log(e L_1/d_{\bar{\Pi}})\right],$$

and if $M_1 \ge m(C_3 \epsilon, C_4 \delta/N_1)$, then

$$\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} \le B'_\gamma \min_{f \in \bar{\mathcal{F}}} \|f - V^{\pi_k}\|_{1,\rho_0} + \min_{\pi \in \bar{\Pi}} \|T^d V^{\pi_k} - T_\pi V^{\pi_k}\|_{1,\rho_1} + \epsilon$$

with probability at least $1 - \delta$.

Sketch of Proof. By adding and subtracting terms, applying the triangle inequality, and invoking Lemma 2, we see that:

$$\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} \le B'_\gamma \|V_k - V^{\pi_k}\|_{1,\rho_0} + \|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1},$$

Here, the error is split into two terms. The first depends on the sample $\mathcal{S}_{0,k}$ and the history through π_k while the second term depends on the sample $\mathcal{S}_{1,k}$ and the history through V_k. We can thus view π_k as fixed when analyzing the first term and V_k as fixed when analyzing the second term (details in the supplementary material). The first term $\|V_k - V^{\pi_k}\|_{1,\rho_0}$ contributes the quantity $\min_{f \in \bar{\mathcal{F}}} \|f - V^{\pi_k}\|_{1,\rho_0}$ in the final bound with additional estimation error contained within ϵ. The second term $\|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1}$ contributes the rest. See Figure 3 for an illustration of the main proof steps. □

The first two terms on the right-hand-side are related to the approximation power of $\bar{\mathcal{F}}$ and $\bar{\Pi}$ and can be considered unavoidable. We upper-bound these terms by maximizing over $\bar{\Pi}$, in effect removing the dependence on the random process π_k in the analysis of the next theorem. We define:

$$\mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) = \max_{\pi \in \bar{\Pi}} \min_{f \in \bar{\mathcal{F}}} \|f - V^\pi\|_{1,\rho_0},$$
$$\mathbb{D}_1^d(\bar{\Pi}) = \max_{\pi \in \bar{\Pi}} \min_{\pi' \in \bar{\Pi}} \|T^d V^\pi - T_{\pi'} V^\pi\|_{1,\rho_1},$$

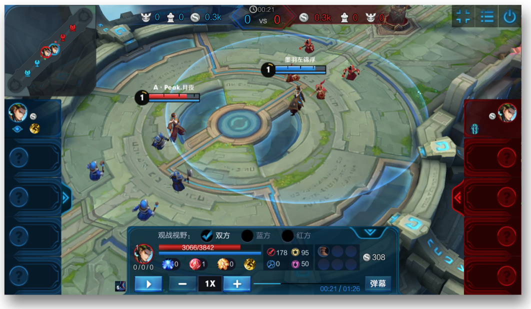

Figure 4. Screenshot from 1v1 *King of Glory*

two terms that are closely related to the notion of *inherent Bellman error* (Antos et al., 2008; Munos & Szepesvári, 2008; Lazaric et al., 2016; Scherrer et al., 2015; Haskell et al., 2017). Also, let $B_\gamma = \sum_{i,j=0}^{\infty} \gamma^{i+j} A_{i+j}$, which was assumed to be finite in Assumption 5.

Theorem 1. *Suppose the sample size requirements of Lemma 3 are satisfied with ϵ/B_γ and δ/K replacing ϵ and δ, respectively. Then, the suboptimality of the policy π_K can be bounded as follows:*

$$\|V^* - V^{\pi_K}\|_{1,\nu} \leq B_\gamma [B'_\gamma \, \mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) + \mathbb{D}_1^d(\bar{\Pi})] + \gamma^{Kd} \|V^* - V^{\pi_0}\|_\infty + \epsilon,$$

with probability at least $1-\delta$.

Search Depth. How should the search depth d be chosen? Theorem 1 shows that as d increases, fewer iterations K are needed to achieve a given accuracy; however, the effort required of tree search (i.e., the function $m(\epsilon, \delta)$) grows exponentially in d. At the other extreme ($d = 1$), more iterations K are needed and the "fixed cost" of each iteration of the algorithm (i.e., sampling, regression, and classification — all of the steps that do not depend on d) becomes more prominent. For a given problem and algorithm parameters, these computational costs can each be estimated and Theorem 1 can serve as a guide to selecting an optimal d.

7. Case Study: King of Glory MOBA AI

We implemented *Feedback-Based Tree Search* within a new and challenging environment, the recently popular MOBA game *King of Glory* by Tencent (the game is also known as *Honor of Kings* and a North American release of the game is titled *Arena of Valor*). Our implementation of the algorithm is one of the first attempts to design an AI for the 1v1 version of this game.

Game Description. In the *King of Glory*, players are divided into two opposing teams and each team has a base located on the opposite corners of the game map (similar to other MOBA games, like *League of Legends* or *Dota 2*). The bases are guarded by towers, which can attack the enemies when they are within a certain attack range. The goal of each team is to overcome the towers and eventually destroy the opposing team's "crystal," located at the enemy's base. For this paper, we only consider the 1v1 mode, where each player controls a primary "hero" alongside less powerful game-controlled characters called "minions." These units guard the path to the crystal and will automatically fire (weak) attacks at enemies within range. Figure 4 shows the two heroes and their minions; the upper-left corner shows the map, with the blue and red markers pinpointing the towers and crystals.

Experimental Setup. The state variable of the system is taken to be a 41-dimensional vector containing information obtained directly from the game engine, including *hero locations*, *hero health*, *minion health*, *hero skill states*, and *relative locations to various structures*. There are 22 actions, including move, attack, heal, and special skill actions, some of which are associated with (discretized) directions. The reward function is designed to mimic *reward shaping* (Ng et al., 1999) and uses a combination of signals including *health*, *kills*, *damage dealt*, and *proximity to crystal*. We trained five King of Glory agents, using the hero *DiRenJie*:

1. The "FBTS" agent is trained using our feedback-based tree search algorithm for $K = 7$ iterations of 50 games each. The search depth is $d = 7$ and rollout length is $h = 5$. Each call to `MCTS` ran for 400 iterations.

2. The second agent is labeled "NR" for *no rollouts*. It uses the same parameters as the FBTS agent except no rollouts are used. At a high level, this bears some similarity to the AlphaGo Zero algorithm (Silver et al., 2017) in a batch setting.

3. The "DPI" agent uses the *direct policy iteration* technique of (Lazaric et al., 2016) for $K = 10$ iterations. There is no value function and no tree search (due to computational limitations, more iterations are possible when tree search is not used).

4. We then have the "AVI" agent, which implements *approximate value iteration* (De Farias & Van Roy, 2000; Van Roy, 2006; Munos, 2007; Munos & Szepesvári, 2008) for $K = 10$ iterations. This algorithm can be considered a batch version of DQN (Mnih et al., 2013).

5. Lastly, we consider an "SL" agent trained via *supervised learning* on a dataset of approximately 100,000 state/action pairs of human gameplay data. Notably, the policy architecture used here is consistent with the previous agents.

In fact, both the policy and value function approximations are consistent across all agents; they use fully-connected neural networks with five and two hidden layers, respectively, and SELU (scaled exponential linear unit) activation (Klambauer et al., 2017). The initial policy π_0 takes random actions: move (w.p. 0.5), directional attack (w.p. 0.2), or a special skill (w.p. 0.3). Besides biasing the move direction toward the forward direction, no other heuristic information is used by π_0. `MCTS` was chosen to be a variant of UCT (Kocsis & Szepesvári, 2006) that is more amenable toward parallel simulations: instead of using the argmax of the UCB scores, we sample actions according to the distribution obtained by applying softmax to the UCB scores.

In the practical implementation of the algorithm, `Regress` uses a mean squared error loss while `Classify` combines a negative log-likelihood loss with a cosine proximity loss (due to continuous action parameters; see supplementary material), differing from the theoretical specifications. Due to the inability to "rewind" or "fast-forward" the game environment to arbitrary states, the sampling distribution ρ_0 is implemented by first taking random actions (for a random number of steps) to arrive at an initial state and then following π_k until the end of the game. To reduce correlation during value approximation, we discard $2/3$ of the states encountered in these trajectories. For ρ_1, we follow the `MCTS` policy while occasionally injecting noise (in the form of random actions and random switches to the default policy) to reduce correlation. During rollouts, we use the internal AI for the hero *DiRenJie* as the opponent.

Results. As the game is nearly deterministic, our main test methodology is to compare the agents' effectiveness against a common set of opponents chosen from the internal AIs. We also added the internal DiRenJie AI as a "sanity check" baseline agent. To select the test opponents, we played the internal DiRenJie AI against other internal AIs (i.e., other heroes) and selected six heroes of the *marksman* type that the internal DiRenJie AI is able to defeat. Each of our agents, including the internal DiRenJie AI, was then played against every test opponent. Figure 5 shows the length of time, measured in frames, for each agent to defeat the test opponents (a value of 20,000 frames is assigned if the opponent won).

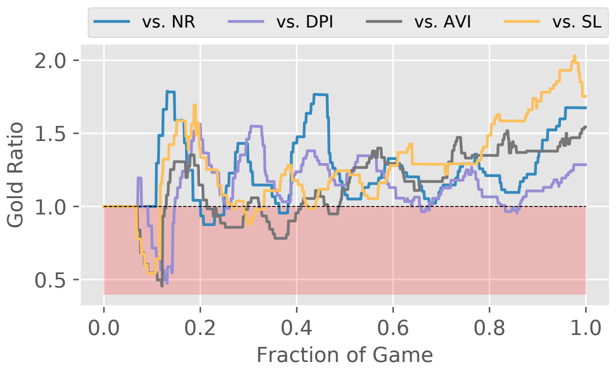

Figure 6. In-game Behavior

Against the set of common opponents, FBTS significantly outperforms DPI, AVI, SL, and the internal AI. However, FBTS only slightly outperforms NR on average (which is perhaps not surprising as NR is the only other agent that also uses MCTS). Our second set of results help to visualize head-to-head battles played between FBTS and the four baselines (all of which are won by FBTS): Figure 6 shows the ratio of the FBTS agent's gold to its opponent's gold as a function of time. Gold is collected throughout the game as heroes deal damage and defeat enemies, so a ratio above 1.0 (above the red region) indicates good relative performance by FBTS. As the figure shows, each game ends with FBTS achieving a gold ratio in the range of $[1.25, 1.75]$.

8. Conclusion & Future Work

In this paper, we provide a sample complexity analysis for feedback-based tree search, an RL algorithm based on repeatedly solving finite-horizon subproblems using MCTS. Our primary methodological avenues for future work are (1) to analyze a self-play variant of the algorithm and (2) to consider related techniques in multi-agent domains (see, e.g., Hu & Wellman (2003)). The implementation of the algorithm in the 1v1 MOBA game *King of Glory* provided us encouraging results against several related algorithms; however, significant work remains for the agent to become competitive with humans.

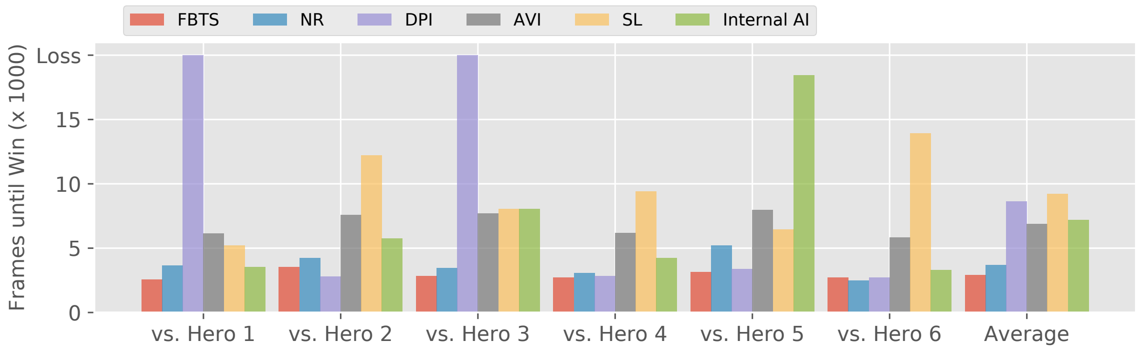

Figure 5. Number of Frames to Defeat Marksman Heroes

Acknowledgements

We wish to thank four anonymous reviewers, whose feedback helped to significantly improve the paper. We also thank our colleagues at Tencent AI Lab, particularly Carson Eisenach and Xiangru Lian, for technical help. Daniel Jiang is grateful for the support from Tencent AI Lab through a faculty award. The research of Han Liu was supported by NSF CAREER Award DMS1454377, NSF IIS1408910, and NSF IIS1332109. This material is also based upon work supported by the National Science Foundation under grant no. 1740762 “Collaborative Research: TRIPODS Institute for Optimization and Learning.”

References

Al-Kanj, L., Powell, W. B., and Bouzaiene-Ayari, B. The information-collecting vehicle routing problem: Stochastic optimization for emergency storm response. *arXiv preprint arXiv:1605.05711*, 2016.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pp. 5366–5376, 2017.

Antos, A., Szepesvári, C., and Munos, R. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

Bertsekas, D. P. Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, 1975.

Bertsekas, D. P. and Tsitsiklis, J. N. *Neuro-dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

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

Campbell, M., Hoane Jr, A. J., and Hsu, F.-h. Deep blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

Cazenave, T. Nested Monte-Carlo search. In *International Joint Conference on Artificial Intelligence*, pp. 456–461, 2009.

Chaslot, G., Saito, J.-T., Uiterwijk, J. W., Bouzy, B., and van den Herik, H. J. Monte-Carlo strategies for computer Go. In *18th Belgian-Dutch Conference on Artificial Intelligence*, pp. 83–90, 2006.

Chaslot, G., Bakkes, S., Szita, I., and Spronck, P. Monte-carlo tree search: A new framework for game AI. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2008.

Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., and Bonnard, N. Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, pp. 433–445. Springer, 2011a.

Couëtoux, A., Milone, M., Brendel, M., Doghmen, H., Sebag, M., and Teytaud, O. Continuous rapid action value estimates. In *Asian Conference on Machine Learning*, pp. 19–31, 2011b.

Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pp. 72–83, 2006.

De Farias, D. P. and Van Roy, B. On the existence of fixed points for approximate value iteration and temporal-difference learning. *Journal of Optimization theory and Applications*, 105(3):589–608, 2000.

Dufour, F. and Prieto-Rumeau, T. Approximation of markov decision processes with general state space. *Journal of Mathematical Analysis and Applications*, 388(2):1254–1267, 2012.

Enzenberger, M. Evaluation in go by a neural network using soft segmentation. In *Advances in Computer Games*, pp. 97–108. Springer, 2004.

Gelly, S. and Silver, D. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine learning*, pp. 273–280, 2007.

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

Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., and Teytaud, O. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems*, pp. 3338–3346, 2014.

Haskell, W. B., Jain, R., and Kalathil, D. Empirical dynamic programming. *Mathematics of Operations Research*, 41 (2):402–429, 2016.

Haskell, W. B., Jain, R., Sharma, H., and Yu, P. An empirical dynamic programming algorithm for continuous MDPs. *arXiv preprint arXiv:1709.07506*, 2017.

Haussler, D. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

Hingston, P. and Masek, M. Experiments with Monte Carlo Othello. In *IEEE Congress on Evolutionary Computation*, pp. 4059–4064. IEEE, 2007.

Hu, J. and Wellman, M. P. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4(Nov):1039–1069, 2003.

Jiang, D. R., Al-Kanj, L., and Powell, W. B. Monte carlo tree search with sampled information relaxation dual bounds. *arXiv preprint arXiv:1704.05963*, 2017.

Kaufmann, E. and Koolen, W. Monte-Carlo tree search by best arm identification. In *Advances in Neural Information Processing Systems*, pp. 4904–4913, 2017.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 972–981, 2017.

Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pp. 282–293, 2006.

Langford, J. and Zadrozny, B. Relating reinforcement learning performance to classification performance. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 473–480, 2005.

Lazaric, A., Ghavamzadeh, M., and Munos, R. Analysis of classification-based policy iteration algorithms. *Journal of Machine Learning Research*, 17(19):1–30, 2016.

Li, L., Bulitko, V., and Greiner, R. Focus of attention in reinforcement learning. *Journal of Universal Computer Science*, 13(9):1246–1269, 2007.

Maîtrepierre, R., Mary, J., and Munos, R. Adaptative play in Texas hold'em poker. In *European Conference on Artificial Intelligence*, 2008.

Méhat, J. and Cazenave, T. Combining UCT and nested Monte Carlo search for single-player general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):271–277, 2010.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning*. MIT Press, 2012.

Munos, R. Performance bounds in l_p-norm for approximate value iteration. *SIAM Journal on Control and Optimization*, 46(2):541–561, 2007.

Munos, R. and Szepesvári, C. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9 (May):815–857, 2008.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pp. 278–287, 1999.

Pollard, D. Empirical processes: Theory and applications. In *NSF-CBMS Regional Conference Series in Probability and Statistics*, pp. i–86. JSTOR, 1990.

Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

Saldi, N., Yüksel, S., and Linder, T. On the asymptotic optimality of finite approximations to markov decision processes with Borel spaces. *Mathematics of Operations Research*, 42(4):945–978, 2017.

Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B., and Geist, M. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16(Aug):1629–1676, 2015.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Teraoka, K., Hatano, K., and Takimoto, E. Efficient sampling method for Monte Carlo tree search problem. *IEICE Transactions on Information and Systems*, 97(3):392–398, 2014.

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