Verification-Guided Tree Search

Extended Abstract

Alvaro Velasquez
Air Force Research Laboratory
Rome, New York
alvaro.velasquez.1@us.af.mil

Daniel Melcer
Northeastern University
Boston, Massachusetts
melcer.d@northeastern.edu

ABSTRACT

Monte-Carlo Tree Search (MCTS) algorithms have been adopted by the artificial intelligence planning community for decades due to their ability to reason over long time horizons while providing guarantees on the convergence of the solution policy. In recent years, such algorithms have been modernized through the integration of Convolutional Neural Networks (CNNs) as part of the state evaluation process. However, both traditional and modern MCTS algorithms suffer from poor performance when the underlying reward signal of the environment is sparse. In this paper, we propose a verification-guided tree search solution which incorporates a reward shaping function within modern MCTS implementations. This function leverages the mathematical representation of the underlying objective by leveraging techniques from formal verification.

KEYWORDS

Monte Carlo search, zero learning, reward shaping, sparse rewards

ACM Reference Format:
Alvaro Velasquez and Daniel Melcer. 2020. Verification-Guided Tree Search. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 3 pages.

1 INTRODUCTION

In recent years, powerful MCTS variants using deep CNNs have been proposed for the game of Go and other board games [1] [8] [10] [9] [7]. However, these modern MCTS implementations, sometimes referred to as zero learning [3], are not yet capable of handling arbitrary objectives with sparse reward signals. We seek to mitigate this by collecting statistics over the objective representation (i.e. the automaton which encodes the objective) as opposed to the implicit statistics collected in traditional reinforcement learning approaches over a state-action representation. These statistics capture the utility of individual transitions in the automaton that were particularly conducive to accomplishing the underlying objective. This is useful as a complement to existing MCTS approaches by reasoning over both the representation of agent-environment dynamics through deep convolutional neural networks as well as the representation of the underlying objective via automata. We call this Verification-Guided Tree Search (VGTS) and argue that this is useful due to the low dimensionality of the automaton that represents the objective. This means that individual transitions within the automaton correspond to many transitions within the Monte Carlo tree.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

2 METHODOLOGY

We assume that the agent-environment dynamics are modeled by a Markov Decision Process (MDP) $\mathcal{M} = (S, s_1, A, T, R)$, where S is a set of states, s_1 is the initial state, A is a set of actions, $T(s'|s,a) \in [0,1]$ denotes the probability of transitioning from state s to state s' when action a is taken, and $R : S \times A \times S \mapsto \mathbb{R}$ is a reward observed when action a is taken in state s leading to state s'. We denote by $L : S \mapsto 2^{AP}$ a labeling function mapping a state in the MDP to a set of atomic propositions AP which hold for that given state.

An LTL_f [5] formula ϕ is defined inductively by the logical connectives and operators $p, !\phi, \phi|\psi, \mathbf{X}\phi, \mathbf{F}\phi, \mathbf{G}\phi, \phi\mathbf{U}\psi$, where $p \in AP$ is an atomic proposition, !, | denote logical negation and disjunction, and ϕ, ψ are LTL_f formulas. The operator semantics are defined as follows: $\mathbf{X}\phi$ holds iff ϕ holds in the next time step, $\mathbf{F}\phi$ holds iff ϕ holds at some point in the future, $\mathbf{G}\phi$ holds iff ϕ holds in all time steps in the future, $\phi\mathbf{U}\psi$ holds iff ψ holds in some time step in the future and ϕ holds until ψ holds. Any LTL_f specification can be equivalently represented as a deterministic finite automaton $\mathcal{A} = (\Omega, \omega_1, \Sigma, \delta, F)$ [4], where Ω is the set of nodes with initial node $\omega_1 \in \Omega$, $\Sigma = 2^{AP}$ is an alphabet defined over a given set of atomic propositions AP, $\delta : \Omega \times \Sigma \mapsto \Omega$ is the transition function, and $F \subseteq \Omega$ is the set of accepting nodes. We use $(\omega, \sigma, \omega') \in \delta$ to denote that $\sigma \in \Sigma$ causes a transition from node ω to ω'.

A trace is defined as a sequence of nodes $\omega_1, \omega_{i_1}, \omega_{i_2}, \ldots$ starting in the initial node ω_1 such that, for each transition, we have $(\omega_{i_k}, \cdot, \omega_{i_{k+1}}) \in \delta$. An accepting trace is one which ends in some node belonging to the accepting set F. Such a trace is said to satisfy the LTL_f specification being represented by the automaton. Our focus is on deriving a policy for a given MDP such that it is conducive to satisfying a given LTL_f specification ϕ, where both of these share the same set of atomic propositions. As a result, transitions in the MDP can cause transitions in the automaton representation of ϕ.

At a high-level, MCTS functions by (i) simulating experience in a given state s_t and (ii) taking an action in said state based on the value estimates obtained from the simulated experience. See Figure 1. In order to accelerate convergence and mitigate the problem of sparse rewards, we extend the tree policy within MCTS by integrating information about $\mathcal{A}$. In particular, we keep track of how often each transition in $\mathcal{A}$ has been part of a trace which satisfies the underlying LTL_f objective of the agent.

$$\pi_{\text{tree}}(s_t, \omega_t) = \text{argmax}_{a \in A} \left(Q(s_t, a) + U(s_t, a) + Y(s_t, a, \omega_t) \right) \quad (1)$$

$$U(s_t, a) = c_{\text{UCB}} \pi_{\text{CNN}}(a|s_t) \frac{\sqrt{\sum_{b \in A(s_t)} N(s_t, b)}}{1 + N(s_t, a)} \quad (2)$$

$$Y(s_t, a, \omega_t) = \max_{s \in \text{subtree}(s_{t+1})} \left(\max_{\omega \to \omega' \in \text{traces}(\omega_{t+1})} V_{\text{LTL}}(\omega, \omega') \right) \quad (3)$$

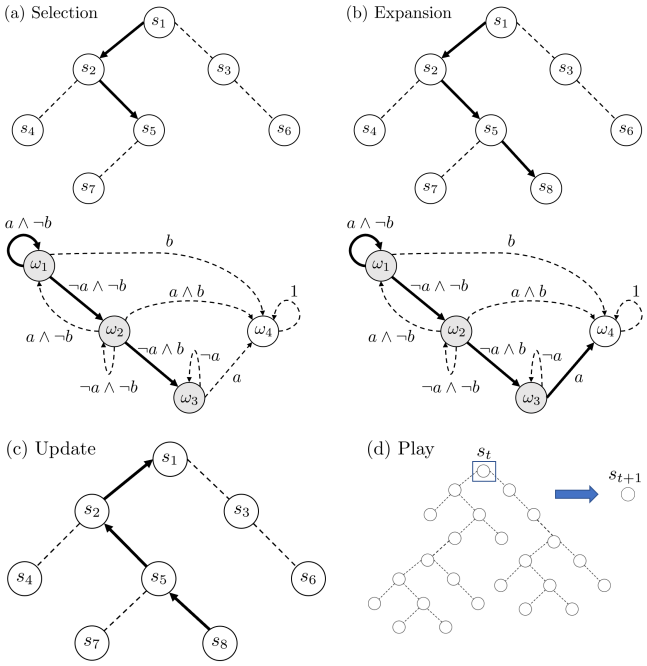

Figure 1: Assume the automaton $\mathcal{A} = (\Omega, \omega_1, \Sigma = 2^{\{\mathbf{a},\mathbf{b}\}}, \delta, F = \{\omega_4\})$ encodes the objective. (a) Given the current state s_1 with $L(s_1) = \{a\}$, a transition $(\omega_1, \{a\}, \omega_1) \in \delta$ in the automaton occurs and an action is selected according to the tree policy (1), where $U(s, a)$ (2) is an exploration function known as the Upper Confidence Bound [2]; $Q(s, a)$ denotes the ratio of (i) the number of times the state-action pair s, a has led to a trajectory which satisfies the underlying objective to (ii) the total number of times $N(s, a)$ that s, a have been encountered; and $Y(s, a, \omega)$ searches through the tree (automaton) starting in state s (node ω) and returns the greatest V_{LTL} value observed in the automaton. Here, $V_{\text{LTL}}(\omega, \omega')$ is the ratio of (i) the number of times the automaton transition $(\omega, \cdot, \omega')$ has led to a trace which satisfies the underlying objective to (ii) the total number of times $(\omega, \cdot, \omega')$ has been encountered. Another state s_2 and its label $L(s_2) = \emptyset$ are observed and the transition $(\omega_1, \emptyset, \omega_2) \in \delta$ occurs. (b) If a new leaf node is added to the tree, its expected value is determined via a rollout policy or neural network inference. We adopt the latter and use a deep convolutional neural network to obtain the values $\pi_{\text{CNN}}(a|s)$ and $V_{\text{CNN}}(s)$ denoting move probabilities and predicted value of the input state, respectively. A new state is observed and a transition in the automaton may occur. If the automaton reaches the accepting node ω_4, the objective has been satisfied and a positive reward is observed. (c) The value estimates are propagated up the tree in order to update previous value estimates for existing states. After some number of expansions have occurred, (d) the agent takes an action in the real environment following the play policy $\pi_{\text{play}}(a|s) \propto N(s, a)/\sum_b N(s, b)$. Samples of the form $(s_t, \pi_{\text{play}}(\cdot|s_t), r)$ are stored in order to train the CNN, where $r = 1$ if and only if the trajectory corresponds to a satisfying trace of the automaton of the given LTL_f specification and $r = 0$ otherwise.

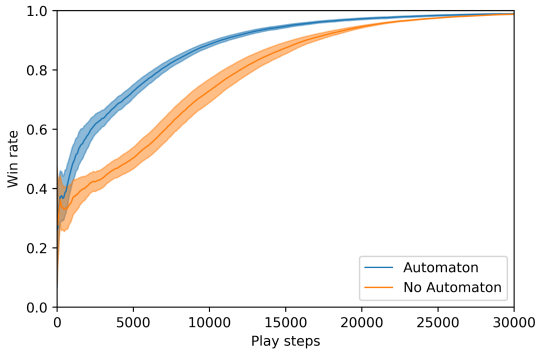

Figure 2: Performance of VGTS and MCTS as a function of the number of play steps executed in the environment described in Section 3. Both VGTS and MCTS are implemented with tree uncertainty backups ([6], Section 3.1) to improve exploration. Average rate and variance with which play steps were part of a trajectory that satisfied the LTL_f objective are reported for 100 random instances of the environment.

3 EXPERIMENTAL SETUP

We evaluate VGTS using a 10x10 minecraft-like environment similar to [11]. The environment contains a home base, 3 randomly placed wood resources, and a randomly placed factory. The agent has 6 possible actions corresponding to moves in any of the cardinal directions, an action to interact with the environment, or no-op. If the agent is standing on a wood tile and chooses the interact action, the wood inventory is increased by 1 and the tile disappears. If the agent is on top of a factory tile and interacts, the wood inventory decreases by 1 and the finished product, or tool, inventory increases by 1. Otherwise, the interact action does nothing. Each instance has an object layout that is randomly generated at the start of training.

The goal of the agent is to return home with three tools in order to build a house. We define the objective of the agent as: *eventually satisfy the goal of building a house, and every time a wood is collected, eventually go to a factory.* This is denoted by the LTL_f formula $\mathbf{G}(wood \implies \mathbf{F} factory) \,\&\, \mathbf{F}(tools \geq 3 \,\&\, house)$. The agent can only carry 2 woods at a time. Thus, it must alternate between collecting wood and going to the factory. A reward is observed upon completion of the task, thereby ending the episode. Alternatively, the episode ends with no reward after 100 play steps.

The CNN used during the expansion phase of VGTS to obtain $\pi_{\text{CNN}}, V_{\text{CNN}}$ consists of a shared trunk with separate policy and value heads. The input to the neural network is a number of stacked 2D layers that are the size of the board. One layer contains a one-hot representation of the position of the agent. There is a layer for every tile type (i.e. wood, factory, etc.) with a value of 1 in the positions where that tile is placed in the environment and 0 otherwise. Lastly, there is a constant-valued layer for each inventory item type (i.e. wood, tool) with a value between 0 and 1 as determined by the current number of held item type divided by the maximum capacity of that item. These maximum capacities are 2 for the wood items and 3 for the tools. These layers are all stacked together and used as input to the neural network. In a similar manner to [10], we optimize the loss function $L = (r - V_{\text{CNN}})^2 - \pi_{\text{play}}^T \log \pi_{\text{CNN}}$. Figure 2 demonstrates the benefit of VGTS over MCTS in this case.

REFERENCES

[1] Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*. 5360–5370.

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

[3] Tristan Cazenave, Yen-Chi Chen, Guan-Wei Chen, Shi-Yu Chen, Xian-Dong Chiu, Julien Dehos, Maria Elsa, Qucheng Gong, Hengyuan Hu, Vasil Khalidov, et al. 2020. Polygames: Improved Zero Learning. *arXiv preprint arXiv:2001.09832* (2020).

[4] Giuseppe De Giacomo and Moshe Vardi. 2015. Synthesis for LTL and LDL on finite traces. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[5] Giuseppe De Giacomo and Moshe Y Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

[6] Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. 2018. Monte Carlo Tree Search for Asymmetric Trees. *arXiv preprint arXiv:1805.09218* (2018).

[7] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2019. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265* (2019).

[8] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.

[11] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2018. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 452–461.