Monte Carlo tree search with state merging for reinforcement learning in Regular Decision Processes

Gabriel Paludo Licks

[1]*Sapienza University of Rome, Department of Computer, Control and Management Engineering, Via Ariosto 25, Rome, 00185, Italy*

Abstract

This research focuses on a novel algorithm for Reinforcement Learning (RL) in Regular Decision Processes (RDPs), a model of non-Markovian decision processes where dynamics and rewards depend on regular properties of the history. Our algorithm is inspired by Monte Carlo tree search (MCTS), yet it is improved with state merging capabilities. Performing merges allows us to evolve the tree model into a graph over time, as we periodically perform similarity tests borrowed from automata learning theory to learn states that are equivalent to one another. This results in improved efficiency and scalability over standard MCTS. Empirical results demonstrate orders of magnitude performance improvement over the state-of-the-art RL algorithms for RDPs. We discuss limitations both in concerning the MCTS implementation and similarity tests from probabilistic automata.

Keywords

monte carlo tree search, automata learning, reinforcement learning, regular decision processes

1. Introduction

The use of Monte Carlo tree search (MCTS) has shown to be highly successful in a variety of applications in the field of reinforcement learning, with outstanding performance when dealing with large state spaces [1, 2]. The study of techniques for state merging, state abstraction, or state aggregation, is widely present in the literature of reinforcement learning, and a number of these have applied it to the case of MCTS [3, 4, 5]. A common observation is that these techniques assume that observations returned by environment represent a complete representation of the current state of the world. However, most real-world applications and domains are naturally non-Markovian. That is, the dynamics and rewards are functions of the entire history [6]. Therefore, complete histories of interactions with the environment have to be considered in order to behave optimally.

A recently introduced formalism, called Regular Decision Processes (RDPs) [6], captures a well-behaved class of Non-Markov Decision Processes (NMDPs). While the dynamics of NMDPs can show an arbitrary dependency on the history of past observations, RDPs work under the assumption that the reward and dynamics are regular functions of the history, and therefore representable by finite-state machines. This implies the existence of an underlying state space where states are determined by histories. This is a key property that makes them amenable to a range of solution techniques.

This research is studying how to devise a Monte Carlo tree search with state merging algorithm for reinforcement learning in RDPs. More specifically, we focus on employing state merging techniques from probabilistic automata [7] learning and seamlessly combine it within MCTS. Previous algorithms for reinforcement in Regular Decision Processes have also used automata learning techniques to learn the model of the decision process, however solved it using planning algorithms such as policy iteration and showed low scalability. The motivation for MCTS is exactly its ability for dealing with larger state spaces, which we show is greatly improved when combined with state merging. Since automata learning is in itself a hard problem, we further adapt MCTS to generate states on demand for every history such that, even with poor estimates of its values, the agent is still able to behave in a best-effort fashion in states it has not visited often enough. Our contributions include the introduction of a novel

22nd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2023) Doctoral Consortium, November 06–07, 2023, Rome, Italy

licks@diag.uniroma1.it (G. P. Licks)

0000-0003-4330-498X (G. P. Licks)

solution that combines MCTS and state merging, closely designed to tackle the current drawbacks in recent works in RDPs, and an empirical evaluation of the algorithm in comparison to state-of-the-art techniques in the literature of RDPs.

2. Related Work

There exists in the literature a number of related studies to this one with the goal of improving the performance of MCTS by means of state merging, state abstraction, or state aggregation. Here we highlight and point out the differences between existing work and our approach. [8] propose a value-based state aggregation, and studies how aggregating states according to their expected value can improve the performance of MCTS in MDPs. [9] and [3] show how abstracting the state space to a smaller dimension helps in the performance of MCTS by exploiting approximate state homomorphisms in MDPs. [4] provide a theoretical regret analysis on the benefit of merging states in Monte Carlo tree search and operating on a graph instead of a tree, however it does not provide a practical algorithm and merge criteria for doing so. [5] show a method for state abstractions based on the geometry of the state space, aggregating newly expanded nodes if similar to a neighbour node in the same depth of the tree. The common characteristic of the studies mentioned above are that none of them considers the setting of NMDPs. That is, state merging, abstraction, or aggregation, are preformed with the assumption that each state holds enough information in order to be aggregated, and not considering the whole history. In contrast, our techniques aim to improve the efficiency of the above algorithms in the non-Markov settings, taking into consideration that both the reward and the dynamics function are history-dependant.

The closest approaches that take into consideration both rewards and dynamics of the decision process are [10, 11, 12]. These approaches focus on Regular Decision Processes, which we introduce in the next section, that assume that the reward and dynamics are representable by finite automata. Given this characteristic, these approaches rely on automata learning techniques to effectively cluster histories into one representative state.

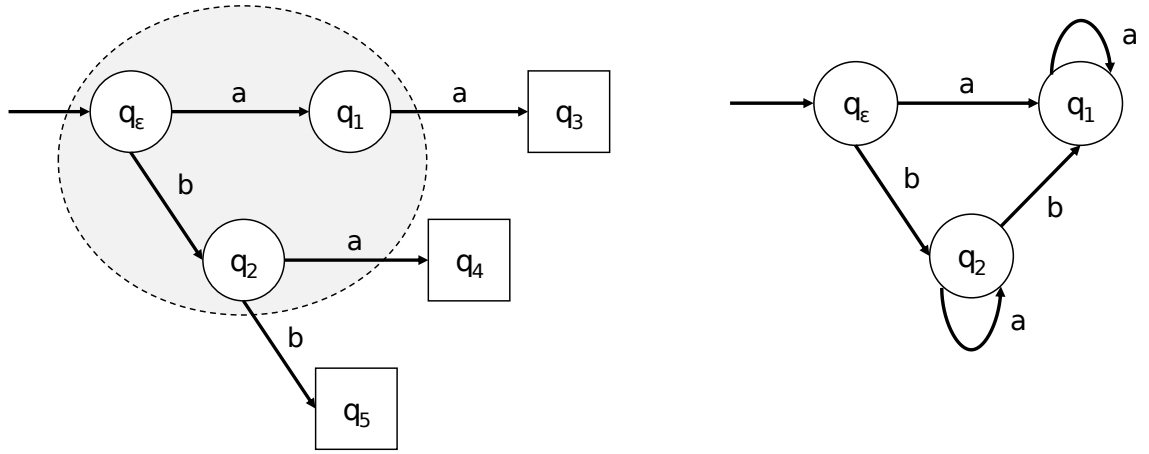
3. MCTS with state merging from probabilistic automata

3.1. Probabilistic Automata Learning

We present a general description of a PDFA-learning algorithm, highliting the concepts and features of algorithms in [13, 14] that are borrowed for developing our algorithm. These algorithms work by building a *hypothesis* automaton that approximates the true *target* automaton. Let us take Figure 1 as an example. Figure 1b shows the graph of a target automaton, and Figure 1a shows the graph of a hypothesis automaton learned so far. States in hypothesis automaton are classified as *safe* (circles) or *candidate* states (squares). Safe states in the hypothesis are one-to-one with states in the target automaton, while candidate states are the frontier of the current hypothesis. As a state is promoted to safe, the frontier is then expanded by adding new candidate states for every possible transition from the recently promoted state. The core of the algorithm are statistical similarity tests, which are periodically performed to determine when a candidate state can be considered safe if distinct from all other states, or merged to another state if equivalent to another state. With enough data collected for each state, the statistical tests we apply provide probably approximately correctly (PAC) guarantees, i.e. high confidence and accuracy, learn a hypothesis that approximates the true target.

3.2. Combining automata learning and MCTS

Now we describe how MCTS and probabilistic automata learning can seamlessly be put together as one. Let us start with the first MCTS procedure, *selection.* All nodes in the tree that are traversed in this phase will correspond to *safe* states, except for leaf nodes that will correspond to *candidate* states. It is easy to see that the concept of nodes in the tree that have already been expanded is closely aligned to the



(a) Hypothesis. (b) Target.

Figure 1: Incremental learning of hypothesis automaton with safe states as circles and candidates as squares, and target automaton.

one of safe nodes. That is, the collected data and statistics each of node/state is enough to confidently represent its possible outcomes.

Following the selection, the *expansion* procedure will correspond to the promotion of candidate states (leaf nodes) to safe states. As a state is promoted to safe, new candidate states are created. That is, the newly promoted safe state is expanded and new candidate states, i.e. child nodes that represent its direct outcome of applying an action, are added to the tree as leaves.

The *simulation* procedure will correspond to the execution of an exploration policy (the *rollout* policy) after a candidate state is reached. Notice that each history and its prefixes generated from simulations past the candidate state are added to the tree as a node. In other words, each candidate state holds a prefix tree of possible outcomes from the rollout policy.

Finally, the *backup* procedure will use the total return of the complete history to backpropagate the value of all nodes in the tree in which the history traverses. We keep for each node, including nodes in the prefix tree past a candidate state, an average return of rollouts from such node. This is an important aspect of the algorithm, since it allows us to maintain a rough statistic of the return, though not accurate in early iterations, for every history that maps to nodes at a not yet expanded part of the tree.

As mentioned in the previous subsection, statistical similarity tests will be performed periodically in order to merge or promote candidate states to safe. The results from these statistical tests will determine when a node will be promoted or merged. It is simple how promoting a candidate state to safe affects the tree by expanding it. However, merging states remains a more elaborate process. Assume that a similarity test returns that a candidate state and a safe state are in fact the same, i.e. their empirical distributions are similar enough such that the test considers them equal. Merging a candidate to safe state will amount to redirecting to the safe state all transitions that once pointed to the candidate state, and propagating the prefix tree from the candidate state to the safe state. Note that merges can also introduce cycles to the graph when a candidate is merged to its parent safe state. With subsequent applications of merges, it is easy to see that a tree eventually converges to a cyclic graph. With the use of MCTS as heuristic to guide the search, we manage to test and learn states that expands the model towards the goal, as data is collected towards higher rewarding nodes.

4. Considerations and future directions

While we apply this algorithm to Regular Decision Processes, which have a one-to-one correspondence to PDFA, we assume it can also be applied to larger classes of non-Markov decision processes. Our approach builds an automaton incrementally, but we expand the fringe of candidate states by considering a new candidate state for every visited history. This allows us to take complete advantage of the sampled

data and compute (initially rough) value estimates for every history, and therefore have a policy with higher average return earlier on. As a drawback, we initially face a large tree, which motivates our decision on building an algorithm based on MCTS, in conjunction with the fact that MCTS does not need a complete and accurate model of the environment. The tree eventually converges to a cyclic graph as states are merged and promoted to safe, and the value estimate of each node gets refined over time. Following the similarity tests we employ from PDFA learning literature, we do have PAC guarantees for the learned model of the dynamics [14, 11]. Regarding the policy, we have guarantees that it converges in the limit as it progressively approximates the value function's expected return (UCB1 never stops exploring)[15].

Notice that we need to collect a significant amount of data (suffixes of histories) for each state needs in order to correctly represent the distribution of possible futures from a given state. Therefore, a major limitation we try to overcome with this algorithm is that similarity tests can be very sample-inefficient in domains where states are harder to learn (we need to look ahead much further to be able to distinguish states). Longer histories have lower probability of being sampled from the environment, and sometimes this is the case for sampling the distinguishing history (a witness that tells two distributions apart). Since this may slow down the process of learning new states given the amount of data necessary, we try to overcome it by considering a new candidate state for every visited history instead of waiting for more data to either promote it to safe or merge.

Furthermore, we limited our base implementation on classic MCTS, which employs the UCB1 selection policy and a uniform rollout policy. We observed that, in domains with high stochasticity and in deeper levels of the tree, the expected return of a node is difficult to estimate and a simple averaged return of rollouts takes long to converge to the true expectation. However, we believe recent improvements in MCTS and policies can further benefit the performance of our algorithm with trivial adaptations.

In other directions, techniques to prune the search could be applied, including safety and advice properties that redirects the search when the agent visits histories that violate such properties [16, 17]. Another limitation concerns the statistical tests, which require a large amount of data in some domains before testing accurately with regard to the PAC parameters (confidence and accuracy). While the final hypothesis graph is minimal and PAC, one could give up on the guarantees and risk less accurate models for sample efficiency. It remains a challenge to improve sample efficiency and still provide guarantees.

Acknowledgments

This work has been partially supported by the EU H2020 project AIPlan4EU (No. 101016442), the ERC-ADG WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR project FAIR (No. PE0000013).

References

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

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, Science 362 (2018) 1140–1144.

[3] L. Xu, J. Hurtado-Grueso, D. Jeurissen, D. P. Liebana, A. Dockhorn, Elastic monte carlo tree search with state abstraction for strategy game playing, in: 2022 IEEE Conference on Games (CoG), IEEE, 2022, pp. 369–376.

[4] E. Leurent, O.-A. Maillard, Monte-carlo graph search: the value of merging similar states, in: Asian Conference on Machine Learning, PMLR, 2020, pp. 577–592.

[5] S. Sokota, C. Y. Ho, Z. Ahmad, J. Z. Kolter, Monte carlo tree search with iteratively refining state abstractions, Advances in Neural Information Processing Systems 34 (2021) 18698–18709.

[6] R. I. Brafman, G. De Giacomo, Regular decision processes: A model for non-Markovian domains, in: IJCAI, 2019, pp. 5516–5522.
[7] B. Balle, J. Castro, R. Gavaldà, Adaptively learning probabilistic deterministic automata from data streams, Mach. Learn. 96 (2014) 99–127.
[8] J. Hostetler, A. Fern, T. Dietterich, State aggregation in monte carlo tree search, in: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI'14, AAAI Press, 2014, p. 2446–2452.
[9] N. Jiang, S. Singh, R. Lewis, Improving uct planning via approximate homomorphisms, in: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, 2014, pp. 1289–1296.
[10] E. Abadi, R. I. Brafman, Learning and solving regular decision processes, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2020, pp. 1948–1954.
[11] A. Ronca, G. De Giacomo, Efficient PAC reinforcement learning in regular decision processes, in: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, International Joint Conferences on Artificial Intelligence Organization, 2021, pp. 2026–2032.
[12] A. Ronca, G. Paludo Licks, G. De Giacomo, Markov abstractions for PAC reinforcement learning in non-markov decision processes, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 3408–3415.
[13] A. Clark, F. Thollard, PAC-learnability of probabilistic deterministic finite state automata, J. Mach. Learn. Res. 5 (2004) 473–497.
[14] B. Balle, J. Castro, R. Gavaldà, Learning probabilistic automata: A study in state distinguishability, Theor. Comput. Sci. 473 (2013) 46–60.
[15] L. Kocsis, C. Szepesvári, Bandit based monte-carlo planning, in: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 282–293.
[16] D. Neider, J.-R. Gaglione, I. Gavran, U. Topcu, B. Wu, Z. Xu, Advice-guided reinforcement learning in a non-markovian environment, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021, pp. 9073–9080.
[17] G. De Giacomo, M. Favorito, L. Iocchi, F. Patrizi, A. Ronca, Temporal Logic Monitoring Rewards via Transducers, in: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, 2020, pp. 860–870. URL: https://doi.org/10.24963/kr.2020/89. doi:10.24963/kr.2020/89.