Tabular Model Learning in Monte Carlo Tree Search

Alberto Castellini[1,*], Davide Bragantini[1,†], Davide Rossignolo[1,†], Federico Segala[1,†] and Alessandro Farinelli[1]

[1]*University of Verona, Department of Computer Science, Strada Le Grazie 15, 37134, Verona, Italy*

Abstract

We present Monte Carlo Tree Search with Tabular Model Learning (MCTS-TML), an extension of MCTS that does not require to know the transition model of the environment, since it learns/adapts the model while interacting with the environment. MCTS-TML assumes discrete states and actions, hence it uses a tabular representation of the transition model. The model update strategy is inspired by that of Dyna-Q but the sample efficiency of MCTS-TML is higher, therefore it requires less interactions with the environment to learn a good policy. Furthermore, MCTS-TML can scale to much larger state spaces (i.e., environments) since it computes the policy online, focusing only on the current state of the system, instead of on all possible states. We also show that MCTS-TML outperforms Q-learning, a popular model-free RL algorithm equivalent to Dyna-Q with no planning steps. Empirical evaluation of MCTS-TML is performed on both deterministic and stochastic environments showing that its sample efficiency is higher than that of Dyna-Q and Q-learning.

Keywords

Monte Carlo Tree Search, Model Learning, Model-based reinforcement learning, Dyna-Q

1. Introduction

Monte Carlo Tree Search (MCTS) [2, 3] is an online probabilistic planning method that has attracted a lot of interest in the last decade because of the impressive results it has contributed to achieve in board games, such as, Go and Chess [4, 5]. A major effort is underway to develop techniques that exploit the potential of MCTS in real-world domains. Interesting research problems in this direction are, for instance, the extension of MCTS to continuous state and action domains [6, 7, 8, 9], the introduction of safety constraints in MCTS-based policies [10, 11] and the safe improvement of policies via MCTS [12].

A key issue for the use of MCTS in real-world applications such as robot navigation [13, 14, 15, 16] and autonomous driving, is the definition of the environment model [17] that the algorithm uses to perform simulations. This model is in fact very complex in real-world domains and usually it is impossible to know it precisely in advance. In this perspective an interesting

IPS-RCRA-SPIRIT 2023: Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy. November 7-9th, 2023, Rome, Italy [1]

*Corresponding author.

†These authors contributed equally.

alberto.castellini@univr.it (A. Castellini); davide.bragantini02@studenti.univr.it (D. Bragantini); davide.rossignolo@studenti.univr.it (D. Rossignolo); federico.segala@studenti.univr.it (F. Segala); alessandro.farinelli@univr.it (A. Farinelli)

trend aims at estimating the environment model used by MCTS [18, 19]. In this paper we tackle this problem. Namely, we assume the model of the environment to be unknown and we propose a methodology to learn it from data acquired online by the agent. We assume discrete state and action spaces, in which the model can be represented in a tabular way. We collect counts of observed transactions between states, and we use these counts to update transaction probabilities. Then we use these probabilities, together with minimal prior knowledge about the environment, as an estimated transition model to perform MCTS simulations.

We compare the performance of the proposed MCTS algorithm with learned model (in the following called MCTS-TML, for brevity) with that of two popular state-of-the-art tabular reinforcement learning (RL) algorithms, namely, the model-free *Q-learning* [20] and the model-based *Dyna-Q* [21, 20]. As expected, MCTS-TML outperforms Q-learning in terms of sample efficiency on a standard benchmark domain. More interestingly, the sample efficiency of MCTS-TML results higher than that of Dyna-Q, hence MCTS-TML reaches optimality earlier than Dyna-Q. In our empirical test we first analyze the behaviour of the two algorithms on deterministic environments, then we investigate the performance on stochastic environments, and finally we identify and explain the reasons of this difference in sample efficiency.

Although preliminary, this work provides new insight on model learning in MCTS, a topic which has so far only been addressed from a Bayesian RL perspective. In Bayesian Adaptive Monte Carlo Planning (BAMCP) [18] and Bayesian Adaptive Partially Observable Monte Carlo Planning (BAPOMCP) [19], model learning is seen as a part of planning under uncertainty, since model parameters are inserted in the state of the system and actions are performed to decrease model parameter uncertainty, until an increasingly certain dynamical model is found. In contrast, our algorithm follows a different perspective building on the basic idea proposed by Dyna-Q. Essentially, the approach stores in a tabular model the knowledge about the dynamics of the environment. This knowledge is collected using a policy which is initially random and then becomes more and more efficient using the dynamics model for planning (i.e., generating rollouts and computing Q-values from them). The main contributions of this work are therefore threefold:

- we propose a model-based RL and Dyna-inspired algorithm called MCTS-TML that introduces model learning in MCTS;
- we compare the performance of MCTS-TML with that of Dyna-Q and Q-learning in deterministic and stochastic environments;
- we investigate the motivations of the improvement achieved by MCTS-TML with respect to Dyna-Q highlighting the key elements for the superior performance of our approach and paving the way for future research directions in this are.

2. Background

2.1. Markov Decision Processes

A Markov Decision Process (MDP) [22] is a tuple $M = \langle S, A, T, R, \gamma \rangle$, where S is a finite set of *states*, A is a finite set of *actions* (we represent each action with its index, i.e., $A = \{1, \dots, |A|\}$), $T : S \times A \to \mathscr{P}(S)$ is a stochastic *transition function*, where $\mathscr{P}(E)$ denotes the space of probability distributions over the finite set E, therefore $T(s, a, s')$ indicates the probability of reaching the state $s' \in S$ after executing $a \in A$ in $s \in S$, $R : S \times A \rightarrow [-R_{max}, R_{max}]$ is a bounded stochastic *reward function*, and $\gamma \in [0, 1)$ is a *discount factor*. The set of stochastic policies for M is $\Pi = \{\pi : S \rightarrow \mathscr{P}(A)\}$.

Given an MDP M and a policy π we can compute state values $V^{\pi}_{M}(s), s \in S$, namely, the expected value acquired by π from s; and action values $Q_M(s, a), s \in S, a \in A$, namely, the expected value acquired by π when action a is performed from state s. To evaluate the performance of a policy π in an MDP M, i.e., $\rho(\pi, M)$, we compute its expected return (i.e., its value) in the initial state s_0, namely, $\rho(\pi, M) = V^{\pi}_{M}(s_0)$. The goal of MDP solvers, such as value iteration and policy iteration [20], is to compute optimal policies, namely, policies having maximal values (i.e., expected return) in all their states. We use V_{max} to denote the known upper bound of the return's absolute value, i.e., $V_{max} \leq \frac{R_{max}}{1-\gamma}$.

2.2. Monte Carlo Tree Search

MCTS [23, 3] is an online solver, namely, it computes the optimal policy only for the current state of the agent, instead of computing it for all possible states as value iteration, policy iteration and other offline solvers do. This feature of MCTS allows it to scale to large state spaces, which are typical of real-world domains. Given the current state of the agent, MCTS first generates a Monte Carlo tree rooted in the state to estimate in a sample-efficient way the Q-values for that state. Then, it uses these estimates to select the best action. A certain number $m \in \mathbb{N}$ of simulations is performed using, at each step, Upper Confidence Bound applied to Trees [24, 25] (inside the tree) or a rollout policy (from a leaf to the end of the simulation) to select the action, and the *known transition model* (or an equivalent simulator) to perform the step from one state to the next. Simulations allow to update two node statistics, namely, the average discounted return $Q(s, a)$ obtained selecting action a and the number of times $N(s, a)$ action a was selected from node (state) s. UCT extends UCB1 [24] to sequential decisions and allows to balance exploration and exploitation in the simulation steps performed inside the tree, and to find the optimal action as m tends to infinity. Given the average return $\bar{X}_{a,T_a(t)}$ of each action $a \in A$ of a node, where $T_a(t)$ is the number of times action a has been selected up to simulation t from that node, UCT selects the action with the best upper confidence bound. In other words, the index of the action selected at the t-th visit of a node is $I_t = \operatorname{argmax}_{a \in 1,\ldots,|A|} \ \bar{X}_{a,T_a(t)} + 2C_p\sqrt{\frac{\ln(t-1)}{T_a(t-1)}}$, with appropriate constant $C_p > 0$. When all m simulations are performed the action a with maximum average return $\bar{X}_{a,T_a(t)}$ in the root is executed in the real environment.

2.3. Problem definition, goal and research questions

We assume the transition model (or equivalent simulator) used by MCTS (to perform the steps of the Monte Carlo simulations) to be unknown. Our goal is to provide a method for learning this transition model from data acquired from the environment as the agent acts. We want to learn the model as quickly as possible and to use it efficiently to provide a high-performance policy. Sample efficiency is key in this context. Consolidated planning methods are available to solve the planning problem when the transition model is known but the model learning and adaptation problem in sampling-based (i.e., scalable) planning methods, such as MCTS, is still not completely explored. The research questions we want to answer are the following: *Q1* - "How efficient is, in terms of policy performance, to learn the transition model in the context of MCTS compared to learning it in the context of Q-learning, as in Dyna-Q?"; *Q2* - "How efficient is, in terms of policy performance, to learn the policy by explicitly learning the transition model and using it in the context of MCTS compared to learning directly the policy as in model-free RL methods, e.g., Q-learning?".

3. Method

We propose a method to learn the transition model inspired by that used in Dyna-Q. The technique assumes both the state space and the action space to be discrete. The environment dynamics can be deterministic or stochastic. The transition model, called M in the following, is tabular and it is implemented as a dictionary.

For each transition performed in the environment from a state $s \in S$ to a state $s' \in S$ applying an action $a \in A$ we collect a triplet $\langle s, a, s' \rangle$ (dictionary key) and update a related set of information about the transition $\langle c(s, a, s'), p(s, a, s'), d(s'), r(s, a, s') \rangle$ (dictionary value), namely, a count $c(s, a, s')$ of the number of times the transition has been observed until the current step (considering also previous episodes), the estimated probability $p(s, a, s') = \frac{c(s,a,s')}{\sum_{s'' \in S} c(s,a,s'')}$ to reach state s' performing action a from state s according to the current counts $c(s, a, \cdot)$, a boolean $d(s') \in \{0, 1\}$ which is 1 if the transition ends the episode (i.e., s' is a terminal state), 0 otherwise, and the reward $r(s, a, s') \in \mathbb{R}$ achieved performing the transition. For instance, if we perform in the real environment a transition from state $s = s_2$ to state $s' = s_5$ by action a_3, and if this is the first time the transition was performed, than we set $c(s_2, a_3, s_5) = 1$. If applying action a_3 from state s_2 we previously also reached state s_0 two times, then we set $p(s_2, a_3, s_5) = 1/3$ (and update $p(s_2, a_3, s_0)$ to 2/3). If s_5 is not terminal, then we set $d(s_5) = 0$. Finally, if the reward achieved in the transition from state s_2 to state s_5 by action a_3 is -1, we set $r(s_2, a_3, s_5) = -1$. The model of the environment is then updated by adding the key-value element $\langle s_2, a_3, s_5 \rangle \rightarrow \langle 1, 1/3, 0, -1 \rangle$ since the transition from s_2 to s_5 with action a_3 was performed for the first time.

The integration between the MCTS action-selection strategy and the proposed model-learning strategy is formalized in Algorithm 1. At the beginning the model does not contain any information hence M is an empty dictionary (line 2 of Algorithm 1). For each episode the algorithm initializes the state to s_0 (line 4), then for each step of the episode it: *i)* performs MCTS to select the action (line 6), *ii)* performs the selected action in the real environment (line 7), *iii)* updates the model (lines 9-16) by adding a new key-value entry (lines 9-13) if the transition has been observed for the first time, *iv)* updates the current state (line 17), *v)* starts a new episode if the new state is terminal (lines 18-20).

Notice that the action selection function *SELECT_ACTION_MCTS*(M_{prior}, M, s) performs MCTS using the current estimation of the transition model (i.e., model M) and also the prior knowledge about the environment (model M_{prior}). M_{prior} is used to perform random but realistic steps of simulations in states never observed before (for which no entry is present in M). For instance, in a GridWorld environment M_{prior} says that if the agent is in a cell (i.e., state s) and performs an action a (e.g., move right), it can reach randomly one of the four neighbour cells (i.e., s'), get a random reward (i.e., r) among those compatible with the transition, and randomly reach/not reach a final state (i.e., d) compatibly with the transition and reward previously selected. This prior knowledge is available in all domains and it does not provide any unfair advantage to MCTS-TML since it only describes *realistic* transitions in the specific domain. In the worst (i.e., less informative) case M_{prior} is completely random, namely, it allows transitions to any possible next state, with completely random reward and random terminal states. M_{prior} is used only if no entry is available in M for the current state s. When at least k[1] entries are available the model M is used to perform simulation steps from that state.

Algorithm 1 MCTS-TML

```
Require: EP: total number of episodes; ST: maximum number of steps per episode; s_0: initial
    state; M_prior: prior knowledge about the environment
 1: // Model initialization
 2: $M = \emptyset$
 3: for $ep = 1, \cdots, EP$ do
 4:     $s = s_0$
 5:     for $st = 1, \cdots, ST$ do
 6:         $a = SELECT\_ACTION\_MCTS(M_{prior}, M, s)$
 7:         $\langle s', r, d \rangle = PERFORM\_ACTION(s, a)$
 8:         // Model update
 9:         if $\langle s, a, s' \rangle \notin M$ then
10:             $M(s, a, s').c(s, a, s') = 0$
11:             $M(s, a, s').p(s, a, s') = 0$
12:             $M(s, a, s').d(s, a, s') = d$
13:         end if
14:         $M(s, a, s').c(s, a, s') = M(s, a, s').c(s, a, s') + 1$
15:         $M(s, a, s').p(s, a, s') = \frac{c(s,a,s')}{\sum_{s'' \in S} c(s,a,s'')}$
16:         $M(s, a, s').r(s, a, s') = r$
17:         $s = s'$
18:         if d=1 then
19:             Break
20:         end if
21:     end for
22: end for
```

4. Empirical evaluation

The performance of the proposed approach is here evaluated and compared with that of state-of-the-art methodologies.

[1]In this work we use $k = 1$ but future work will be dedicated to develop methods that best tune this parameter since this parameter is important for guaranteeing model precision (see Section 5).

4.1. Baseline algorithms

We compare our MCTS-TML with three other algorithms:

- MCTS_ORACLE [3]: it is the standard MCTS algorithm using the true transition model. This algorithm is used only as an oracle to estimate the best performance reachable using an exact model;
- Dyna-Q [21]: it is a model-based tabular RL algorithm from which we took inspiration to implement the model learning strategy. Basically, Dyna-Q and MCTS-TML use the same tabular representation of the transition model and they update it in the same way as new observations are collected. Therefore, the only difference between Dyna-Q and MCTS-TML is the way in which the two algorithms use the learned transition model to update the policy. MCTS-TML uses this model to performs Monte Carlo simulations (according to the UCT action selection strategy) and estimates the Q-values of all actions in the current state according to the MCTS strategy. Dyna-Q uses the model to generate single virtual steps from several observed states and updates the Q-values of all considered state-action pairs according to the Q-learning (i.e., temporal difference) strategy.
- Q-learning [20]: is a model-free tabular RL algorithm that corresponds to Dyna-Q with no planning steps. Namely, it updates the Q-values of only the state-action pairs that the agent actually visits using the Q-learning (i.e., temporal difference) strategy. We use this algorithm to analyze the performance of an approach that does not explicitly learn the transition model.

4.2. Domain

The domain used in our tests is the Gymnasium implementation of Frozen Lake[2]. We selected this domain because it has discrete state and action spaces, hence its model can be represented by a table. This is a requirement to use MCTS-TML. Furthermore, the domain has a deterministic and a stochastic version. Both versions can be solved by MCTS-TML. An agent moves in a 4x4 grid starting from the top-left corner and aiming to reach the goal in the bottom-right corner. Four holes in the grid must be avoided by the agent since they end the episode (without reaching the goal). The states are the 16 possible positions of the agent in the grid. Terminal states are those in which the agent reaches a hole or the goal. The actions are the four movements (left, down, right, up) the agent can do. The transition model can be set as deterministic (each action always moves the agent in the selected direction) or stochastic (each action moves the agent in the selected direction with probability 0.9 and in directions perpendicular to the chosen one with probability 0.05 for each perpendicular direction). If the agent tries to move out of the board it stays in the cell where it is. The reward function returns +1 when the agent reaches the goal state, 0 when the agent reaches any other cell (standard or hole).

4.3. Experimental setting

We first perform our tests on the deterministic environment, in which the transition model is simpler to learn because each transition must be observed only once to learn its probability.

[2]https://gymnasium.farama.org/environments/toy_text/frozen_lake/

Then, we switch to the stochastic environment, in which transition probabilities require several observations to be estimated precisely. We run each algorithm 50 times. Each time it performs 200 episodes and in each episode it performs at most 100 steps (less steps are performed if the agent reaches the goal or a hole in advance). MCTS-TML and MCTS-ORACLE perform at each step $m = 1000$ simulations starting from the current state to evaluate the action Q-values. This parameter has been tuned in advance on MCTS-ORACLE to be sure the number of simulations are enough to reach almost optimal performance in the Frozen Lake environment. Constant C_p in MCTS (see UCT algorithm) is set to 11. In Dyna-Q we used $\alpha = 0.7$, $\epsilon = 1$, ϵ-decay-rate= 0.7 and 25 planning steps (see [20] page 164 for details). All parameters were tuned manually to get the best performance. The discount factor is $\gamma = 1$ in all tests. Experiments are performed with a laptop with processor Intel Core i7 - 6500 CPU 2.50 GHz x 4, RAM 16 GB and operating system Ubuntu 20.04.5 LTS. The code is implemented in Python.

4.4. Performance measures

Performance is computed by averaging at each episode the return of the episode across the 50 repeats. For instance, at the end of episode 1 we compute the average return across the 50 repeats of that episode. After episode 2 the model is improved because it contains observations of both episode 1 and 2, hence we expect the average return after episode 2 is higher than that after episode 1, and so on until episode 200. We call this measure *return per episode*.

To deepen our analysis we also investigate the reason why the performance of an algorithm are higher than that of another algorithm. To this end we compute the distance of the estimated model from the true model (which is known in our synthetic domain). This distance is computed as the sum of the absolute values of the differences between the transition probabilities of the true model and those of the estimated model. In other words, the probability of each transition of the real model is subtracted to the probability of the same transition in the estimated model. We compute the absolute values of all these differences and we sum up all these absolute values. In this way, if the estimated model is equal to the real model the distance is zero and we expect that the distance decrease as the episodes go on, since the model should become more precise.

4.5. Results

The results of our experiments on the *deterministic environment* are reported in Figures 1.a and 1.b. In Figure 1.a MCTS-TML (blue line) reaches the performance of MCTS-ORACLE[3] in about

[3]The performance of MCTS-TML is slightly higher than that of MCTS-ORACLE because of the (small) number of simulations used in our experiments. With this number of simulations MCTS-ORACLE cannot completely converge to the optimal policy, hence it happens that it selects suboptimal actions by (wrongly) estimating their Q-values. On the other hand, MCTS-TML with partial models (i.e., considering only some transitions, because others still have to be learned) can be more simulation-efficient by focusing on fewer actions while computing Q-values. In particular, this happens when in MCTS-TML the transition associated to the best action has been learned but several transitions associated to suboptimal actions are still unknown. In these cases, the Q-value of the optimal action, computed by MCTS, tends to be higher than that of other actions related to unknown transitions (because of the lack of knowledge about those actions). This reduces the error-rate made by MCTS-TML while selecting the optimal action. In the deterministic Frozen Lake environment this situation occurs quite often, bringing MCTS-TML to show slightly better performance than MCTS-ORACLE with small number of simulations, however it is not a general property.

15 episodes, while the performance of Dyna-Q (orange line) grow more slowly and after 200 episodes it still has sub-optimal performance. Q-learning (green line) which does not learn any model has a still slower increase of performance but after episode 140 it surpasses Dyna-Q. The difference in performance between MCTS-TML and Dyna-Q can be explained by the chart in Figure 1.b. It shows that the distance between the model estimated by MCTS-TML and the true model (blue line) decreases much faster than the distance between the model estimated by Dyna-Q and the true model (orange line). Furthermore, the model estimated by Dyna-Q after about 125 episodes tends to stabilize to a value close to 0.15. This means that Dyna-Q almost stops to learn after episode 125 and this is the reason why also the performance of Dyna-Q tends to stabilize at a sub-optimal value.

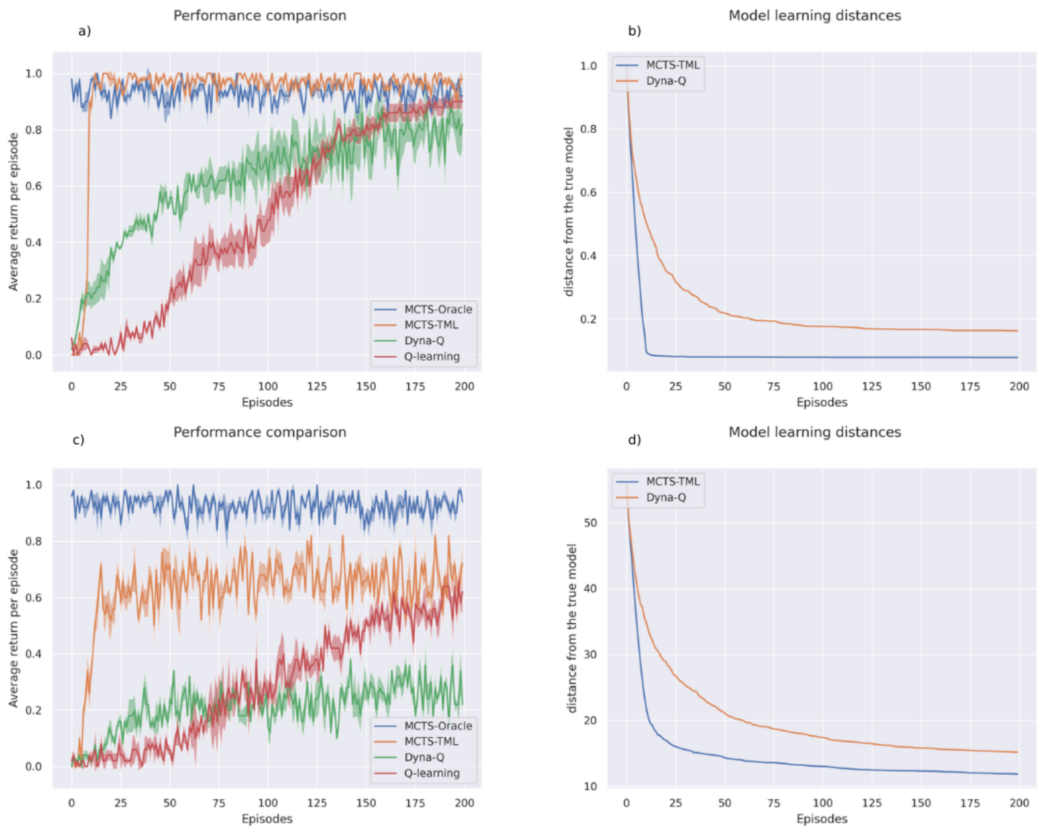

Figure 1: Experimental results. *a.* Evolution of the performance of the four algorithms during episodes in the deterministic environment. *b.* Evolution of the distance between true and estimated transition model of MCTS-TML and Dyna-Q during episodes in the deterministic environment. *c.* As *a* but in the stochastic environment. *d.* As *b* but in the stochastic environment.

The results for the *stochastic environment* are reported in Figures 1.c (algorithm performance) and 1.d (distance from the true model). In this case the performance variability increases because the learning strategy needs several observations to precisely estimate transition probabilities. The performance improvement also of MCTS-TML is a bit slower than that of the deterministic case but we mainly observe that it tends to stabilize to a value which is lower than the optimal one reached by MCTS-ORACLE. Interestingly, Dyna-Q performance increase even more slowly and they stabilize to a much lower value than that of MCTS-TML. Q-learning has an even slower increase of performance but its performance surpasses that of Dyna-Q at epoch 75 and it almost reaches that of MCTS-TML at epoch 200. The reason why performance of MCTS-TML and Dyna-Q stabilize at a sub-optimal value seem to depend on the fact that both of them tend to reduce their distance to the true model slowly from a certain episode. However, the distance of the model learned by MCTS-TML is much lower than that achieved by Dyna-Q.

The reason of these differences could be found in different factors. One is that MCTS-TML performs long simulations to estimate the Q-values, and in this simulations it uses information from the estimated model M where it is available and information from the model containing prior knowledge M_{prior} in transitions that have never been seen before. To investigate this possibility we tried to modify Dyna-Q [26] allowing it to perform several steps (instead of a single one) starting from an already observed transition and continuing in known or unknown transitions, also using model M in the first case and model M_{prior} in the second, but also in that case MCTS-TML outperforms Dyna-Q. We are still investigating the theoretical reason why MCTS-TML outperforms Dyna-Q, but our most accredited hypothesis is that this reason is related to the use of UCT made by MCTS-TML which is more efficient in estimating Q-values than the strategy used by Dyna-Q.

5. Conclusions and future work

We presented MCTS-TML, a model-based RL algorithm based on MCTS. The algorithm learns the model of the environment used to perform Monte Carlo simulations from data acquired while interacting with the environment. The first results achieved by MCTS-TML in the FrozenLake domain are encouraging since it shows better performance and mainly higher sample efficiency than Dyna-Q and Q-learning. Our future work will focus on evaluating the algorithm on other domains and comparing its performance with other model-based and model-free RL algorithms, such as Bayesian Adaptive Monte Carlo Planning (BAMCP) [18], Model Based Policy Optimization (MBPO) [27] and PPO [28]. Moreover, we want to better understand the theoretical reason behind the better performance achieved by MCTS-TML compared to Dyna-Q, and the reason why the distance between the model estimated by MCTS-TML and the true model tends to stabilize to a value different to zero in cases of stochastic environments. Furthermore, we want to implement strategies based on the confidence interval of the transition probabilities to guarantee the safety of the transitions in the simulation process (since imprecise transition probabilities estimated using only few observations can bias the policy towards suboptimal behaviours). Finally, we would like to consider non-tabular representations of the model to allow for greater scaling capabilities.

References

[1] R. De Benedictis, M. Castiglioni, D. Ferraioli, V. Malvone, M. Maratea, E. Scala, L. Serafini, I. Serina, E. Tosello, A. Umbrico, M. Vallati, Preface to the Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (IPS-RCRA-SPIRIT 2023), in: Proceedings of the Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (IPS-RCRA-SPIRIT 2023) co-located with 22th International Conference of the Italian Association for Artificial Intelligence (AI* IA 2023), 2023.

[2] R. Coulom, Efficient selectivity and backup operators in Monte-Carlo Tree Search, in: Computers and Games, Springer Berlin Heidelberg, 2007, pp. 72–83.

[3] 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.

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

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

[6] A. Couetoux, Monte Carlo Tree Search for Continuous and Stochastic Sequential Decision Making Problems, Theses, Université Paris Sud - Paris XI, 2013.

[7] Z. N. Sunberg, M. J. Kochenderfer, Online algorithms for pomdps with continuous state, action, and observation spaces, in: Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018), 2018, pp. 1–13.

[8] M. H. Lim, C. J. Tomlin, Z. N. Sunberg, Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation POMDPs, in: 2021 60th IEEE Conference on Decision and Control (CDC), IEEE Press, 2021, p. 4493–4500.

[9] F. Bianchi, L. Bonanni, A. Castellini, A. Farinelli, Monte Carlo Tree Search planning for continuous action and state spaces, in: Proceedings of the 9th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2023), AI*IA 2022, Udine, Italy, November 30, 2022, volume 3417 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 38–47.

[10] G. Mazzi, A. Castellini, A. Farinelli, Risk-aware shielding of Partially Observable Monte Carlo Planning policies, Artificial Intelligence 324 (2023) 103987.

[11] G. Mazzi, D. Meli, A. Castellini, A. Farinelli, Learning logic specifications for soft policy guidance in POMCP, in: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS '23, IFAAMAS, 2023, p. 373–381.

[12] A. Castellini, F. Bianchi, E. Zorzi, T. D. Simão, A. Farinelli, M. T. J. Spaan, Scalable safe

policy improvement via Monte Carlo tree search, in: Proceedings of the 40th International Conference on Machine Learning (ICML 2023), PMLR, 2023, pp. 3732–3756.

[13] M. Zuccotto, M. Piccinelli, A. Castellini, E. Marchesini, A. Farinelli, Learning state-variable relationships in POMCP: A framework for mobile robots, Frontiers in Robotics and AI 9 (2022).

[14] A. Castellini, E. Marchesini, A. Farinelli, Partially Observable Monte Carlo Planning with state variable constraints for mobile robot navigation, Engineering Applications of Artificial Intelligence 104 (2021) 104382.

[15] Y. Wang, F. Giuliari, R. Berra, A. Castellini, A. D. Bue, A. Farinelli, M. Cristani, F. Setti, POMP: pomcp-based online motion planning for active visual search in indoor environments, in: 31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020, BMVA Press, 2020.

[16] A. Castellini, G. Chalkiadakis, A. Farinelli, Influence of state-variable constraints on partially observable monte carlo planning, in: Proc. 28th International Joint Conference on Artificial Intelligence, IJCAI-19, ijcai.org, 2019, pp. 5540–5546.

[17] M. Zuccotto, A. Castellini, A. Farinelli, Learning state-variable relationships for improving POMCP performance, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, SAC '22, Association for Computing Machinery, 2022, p. 739–747.

[18] A. Guez, D. Silver, P. Dayan, Scalable and efficient bayes-adaptive reinforcement learning based on Monte-Carlo Tree Search, Journal of Artificial Intelligence Research 48 (2013) 841–883.

[19] S. Katt, F. A. Oliehoek, C. Amato, Learning in POMDPs with Monte Carlo tree search, in: Proceedings of the 34th International Conference on Machine Learning, PMLR, 2017, pp. 1819–1827.

[20] R. Sutton, A. Barto, Reinforcement Learning, An Introduction, 2nd ed., MIT Press, 2018.

[21] R. S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, SIGART Bull. 2 (1991) 160–163.

[22] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.

[23] G. Chaslot, S. Bakkes, I. Szita, P. Spronck, Monte-Carlo Tree Search: A new framework for game AI, in: Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'08, AAAI Press, 2008, p. 216–217.

[24] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning 47 (2002) 235–256.

[25] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: Machine Learning: ECML 2006. 17th European Conference on Machine Learning, volume 4212 of *LNCS*, Springer-Verlag, 2006, pp. 282–293.

[26] G. Z. Holland, E. Talvitie, M. H. Bowling, The effect of planning shape on dyna-style planning in high-dimensional state spaces, ArXiv abs/1806.01825 (2018).

[27] M. Janner, J. Fu, M. Zhang, S. Levine, When to trust your model: Model-based policy optimization, in: Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019, p. 12519–12530.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms., CoRR abs/1707.06347 (2017).