Monte Carlo Tree Search Planning for continuous action and state spaces

Federico Bianchi[1], Lorenzo Bonanni[1], Alberto Castellini[1] and Alessandro Farinelli[1]

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

Abstract

Sequential decision-making in real-world environments is an important problem of artificial intelligence and robotics. In the last decade reinforcement learning has provided effective solutions in small and simulated environments but it has also shown some limits on large and real-world domains characterized by continuous state and action spaces. In this work, we aim to evaluate some state-of-the-art algorithms based on Monte Carlo Tree Search planning in continuous state/action spaces and propose a first version of a new algorithm based on action widening. Algorithms are evaluated on a synthetic domain in which the agent aims to control a car through a narrow curve for reaching the goal in the shortest possible time and avoiding the car going off the road. We show that the proposed method outperforms the state-of-the-art techniques.

Keywords

Reinforcement Learning, Monte Carlo Tree Search, Planning in continuous space, Progressive widening

1. Introduction

Planning for sequential decision-making in real-world environments is an important problem in artificial intelligence and robotics. The interest in this topic has rapidly grown and recently Reinforcement Learning (RL) [1] methods have been applied to it. Several of these approaches have however shown to be effective on small or simulated environments but unable to scale to real-world problems. A feature that characterizes several real-world planning problems is the usage of continuous actions and states. Some examples are autonomous driving [2, 3], search and rescue [4] or dynamic resource allocation [5]. In these domains finding an optimal policy is often unfeasible due to the large or multi-dimensional action space, in which the number of possible actions is infinite hence making the exploration of all available actions impossible. In particular, large high-dimensional spaces make often space discretization unfeasible due to computational constraints. Even when such discretization is computationally feasible, the exploratory ability is limited and action accuracy is low.

Monte Carlo Tree Search (MCTS) [6, 7] based planning is an online method widely used to allow planning in Markov Decision Processes (MDP) on large domains. The methodology reached very good results on discrete state/action domains [8] and partially observable domains [9]. It was also extended to consider prior knowledge about the environment [10, 11] and to

The 9th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2022)

federico.bianchi@univr.it (F. Bianchi); lorenzo.bonanni@studenti.univr.it (L. Bonanni); alberto.castellini@univr.it (A. Castellini); alessandro.farinelli@univr.it (A. Farinelli)

improve the trustworthiness of the policy [12, 13, 14], and it was applied to real-world domains [15, 16]. However, only a few works focus on the extension of MCTS to continuous state and action spaces. Some techniques for MDPs were proposed in 2008. In particular, a method called MCTS Action Progressive Widening (MCTS-APW) that controls the widening of actions for continuous action space was introduced in [17] and theoretically analyzed in [18]. In 2013 a methodology based on [17] that extends the widening to actions and states nodes was proposed, namely, the method which is called MTCS-Double Progressive Widening (MCTS-DPW) controls the widening of the tree search both for the states and actions, balancing the exploration of new states and actions or the exploitation of visited nodes [19]. In 2018 the idea of [19] was extended for Partially Observable Markov Decision Processes (POMDPs). The extension introduces a method called Partially-Observable Monte Carlo Planning Observation Widening (POMCPOW) that controls the widening of actions, states, and observations [20]. Finally, in 2019 a technique based on Voronoi diagrams [21] was proposed and subsequently extended to progressive widening for POMDPs in 2021 [22]. This extension uses Voronoi diagrams to partition the action space in cells in which centroids represent the action with the highest state-action value and perform a global search in order to identify the cell and then the best action by sampling new actions around it.

In this work, we aim to *i)* evaluate the most prominent state-of-the-art methodologies for MCTS-based planning in continuous state and action spaces, *ii)* propose a new method that outperforms state-of-the-art techniques. We present, in particular, the results achieved so far comparing standard MCTS-planning [7] using discretized actions, MCTS-APW [17, 19] and an extended version of MCTS-APW, that we call MCTS-APW2, which performs widening by sampling new actions in an efficient way. The methods are compared on a synthetic domain in which the agent aims to control a car through a narrow curve for reaching the goal in the shortest possible time and avoiding the car going off the road. We show that the proposed methodology outperforms the other approaches.

2. Background

2.1. Markov Decision Processes

Markov Decision Processes (MDPs) [23, 24] can formalize a broad range of sequential decision-making problems. Formally, an MDP models problems in fully observable environments using a 5-tuple $M = \langle S, A, T, R, \gamma \rangle$, where S is a set of *states*, A is a set of *actions*, $T : S \times A \to \mathcal{P}(S)$ is a stochastic *transition function*, in which $T(s, a, s')$ indicates the probability of landing on state $s' \in S$ after executing action $a \in A$ in state $s \in S$, $R : S \times A \to \mathbb{R}$ is a *reward function* with $R(s, a) \in [-R_{max}, R_{max}]$, and $\gamma \in [0, 1)$ is a *discount factor*. When T cannot be implicitly defined, a *generative model* $G(s, a)$ can be used to stochastically generate a new state s' and reward r for MDP. The set of stochastic policies for M is $\Pi = \{\pi : S \to \mathcal{P}(A)\}$ and the goal is to find a control policy $\pi(s, a) = P(a|s)$ that maximizes the *expected discounted cumulative reward* defined as $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$.

2.2. Monte Carlo Tree Search Planning for MDPs with discrete states/actions

MCTS is a simulation-based search algorithm for online decision-making [25, 7] that works by incrementally creating a policy tree until some predefined computational budget is reached. The building of a policy tree consists of alternating layers of state nodes generated by a generative model G and action nodes estimated by state-action value function $Q(s, a)$. Each state node in the search tree represents a state of the domain, and direct links to child action nodes represent actions leading to the next states.

A search iteration starts from the root node and four steps are applied: i) *Selection:* An action node is recursively selected to descend through the tree until a node is expandable, i.e., it represents a non-terminal state and has unvisited children. The most common selection function for MCTS is the Upper Confidence Tree (UCT) which expands the tree by selecting nodes and balancing exploration and exploitation in order to maximize the upper confidence bound UCB [26, 27]; ii) *Expansion:* One (or more) child nodes are added to expand the tree according to the available actions; iii) *Simulation:* A simulation is run from the expanded nodes to the leaf nodes according to the default policy to produce an outcome. The default policy selects sequences of actions randomly; iv) *Backpropagation:* The result of the simulation is backpropagated through the selected nodes to update their statistics, which are the average reward obtained from that node onwards and the number of times the node has been visited. At the end of simulations, the action with the maximum Q-value is selected and executed in the environment. This method assumes that the states and actions are discrete.

3. Methods

We first summarize some of the main methods in the literature for MCTS-based planning in continuous state/action spaces and then we describe a new technique aiming to outperform state-of-the-art methods.

3.1. MCTS Planning with discretized actions

We refer here to the standard version of MCTS that discretizes the action space by splitting it into intervals and maps each interval to a discrete value. This technique poses the problem of tuning the number of actions for maximizing the performance, which is often complicated in real-world domains with continuous and multi-dimensional action space. When the number of possible actions is infinite it is impossible to explore all the space and it is often difficult to achieve a good trade-off between accuracy and MC tree width. The curse of dimensionality is a problem for this approach because a large number of actions/spaces makes the tree width explode. In other words, the tree gets wider and never develops in depth obtaining policies optimized only in the first step and using random (rollout) policies in all following steps. This type of policies are not optimal but very close to random.

3.2. MCTS-APW

MCTS Action Progressive Widening (MCTS-APW) was first proposed in [17, 19]. It explicitly controls the branching factor of the action nodes of the tree search. MCTS-APW is necessary

when the action space is continuous or so large that cannot be fully explored. During the search, the number of actions for each state is controlled by two parameters, namely, $k \in \mathbb{R}^+$, $\alpha \in [0, 1]$ that control the number of action nodes for each state. In particular, the technique samples a new random action from the action space if $||A(s)|| < kN(s)^\alpha$, where $||A(s)||$ is the set of actions added to the node with state s and $N(s)$ is the number of visits of state s. The strategy used by this algorithm to generate new actions is to randomly sample from all the action space. If no new action has to be added to the current set, then the algorithm selects an action using UCT [27].

3.3. MCTS-APW2

We propose an extension of MCTS-APW, that substitutes the random sampling of new actions from all the action space with a new sampling strategy. The first three actions selected by MCTS-APW2 are always the minimum, median and maximum points of the action space (in multi-dimensional action spaces the minimum, median, and maximum are obtained by selecting, respectively, the minimum, median and maximum values for each action). From the fourth action, with probability $p < \epsilon$ we select the two actions a_1, a_2, in the current set of selected actions, having the highest Q-value (this is why the name of the method has a "2" at the end), and we compute the new action as $a =$*mean*(a_1, a_2); with probability $1 - \epsilon$ we sample a new action randomly in all the action space. The algorithm is described in more detail in the Supplementary Material. After a new action is added to the state node, its statistics are initialized and UCT is used to select actions in the new action set.

4. Results

This section presents experiments comparing standard MCTS, MCTS-APW and MCTS-APW2. We first describe the domain, then we introduce the experimental setting and parameter tuning performed to determine good sets of parameters for each method. Finally, we show the global performance in terms of average cumulative reward and analyze the best and average trajectories generated to explain the reasons why MCTS-APW2 outperforms MCTS-APW.

4.1. Domain

We evaluate the algorithms in an environment representing a curved road with a bottleneck in which the agent has to reach the other side of the bottleneck in the minimum number of steps (i.e., in the shortest possible time or with the highest possible average speed). The environment is displayed in Figure 1, where the blue and red lines represent the edges of the road and the green dashed line is the trajectory executed by the agent. The trajectory starts in $S0$ with an initial speed 10 m/s with angle $90°$ ($0°$ is a horizontal movement to the right) and it ends when the agent crosses the vertical dashed green line on the right (or if it goes off the road). The state and action spaces are continuous. Namely, the agent can position itself at any point inside the road and it can control the acceleration $a \in [-5, 5]$ and the steering $\phi \in [-30°, 30°]$. The speed of the vehicle must be in the range $v \in [0, 20]$. The reward function returns a score $r = -\frac{\delta}{n_{steps}}$ at each step, where δ is a reward proportional to the distance between the agent and the target

and n_{steps} is the number of steps taken until the current point. The episode is terminated in three cases: if the agent reaches the target (with reward $r = \frac{10000}{n_{steps}}$), if the agent goes off-road (with reward $r = -1000$), or if it reaches 100 steps (with reward $r = -1000$). The transition model is a simple dynamical model that allows to move the agent deterministically according to the current speed, acceleration, and steering.

4.2. Parameter optimization

We perform parameter tuning to select the most suitable parameters for each method. Specifically, we perform a grid search running each combination of parameters for 10 episodes and then identify the set of parameters that maximize the average discounted reward. For standard MCTS (called MCTS in the following) the main parameter to tune is the number of intervals in which each action is discretized. We perform a grid search over each combination of number of accelerations $\omega \in [7, 11, 15, 21, 50, 70, 120, 150]$ and angles $\phi \in [3, 5, 7, 11, 15]$. For MCTS-APW and MCTS-APW2 we perform a grid search to determine the parameters k and α that control the action widening. We tested, in particular, all combinations of $k \in [30, 40, 50, 60, 70, 80, 90]$ and $\alpha \in [0, 0.1, 0.4, 0.5, 0.6, 0.8, 0.9]$. With MCTS-APW2 we also considered all $\epsilon \in [0.4, 0.6, 0.9]$.

The best parameters for the methods are the following: i) MCTS: $\omega = 7$ and $\phi = 7$; ii) MCTS-APW: $k = 40$ and $\alpha = 0$; iii) MCTS-APW2: $k = 40$, $\alpha = 0$ and $\epsilon = 0.4$. In all methods we use 100 simulations to generate the tree, the exploration factor is $c = 11$ and the discount factor is $\gamma = 0.99$. After selecting the best parameters for each method we computed their cumulative reward on 100 episodes using those parameters. Tests are run on two laptops, the first with processor Intel Core i7 - 6500 CPU 2.50 GHz x 4, RAM 16 GB and operating system Ubuntu 20.04.5 LTS, the second with processor Intel Core i7-8550U CPU 1.80GHz × 8, RAM 16 GB and operating system Ubuntu 22.04 LTS.

4.3. Performance comparison

Table 1 shows the performance of the three methods using the best parameters. The parameters used are reported in brackets close to the name of the method in the first column. The second column shows the average performance (over 100 runs), which is the main measure we use to evaluate the algorithms. MCTS-APW2 has the highest cumulative reward (ρ_{avg}), namely 48.3, then there is MCTS with -309.2 and finally MCTS-APW with -909.8. We notice that the algorithm that we propose, i.e., MCTS-APW2, provides a large improvement in terms of average performance. The third and fourth columns of Table 1 show, respectively, the maximum and minimum reward (over 100 runs). It is interesting to notice that also on this measure MCTS-APW2 wins, with a value of 1774.7, but in this case, MCTS reaches a similar performance, i.e., 1773.2. The fifth column shows the number of times the agent went off-road (over 100 runs). MCTS-APW2 does it 42 times, MCTS 66, and MCTS-APW 91. The average reward is of course influenced by the number of times the agent goes off-road and also by the average reward of runs in which the agent reaches the goal. The last column of Table 1 shows the number of actions used on average by each method. It is clear that the two methods based on widening, namely MCTS-APW and MCTS-APW2 use fewer actions (i.e., 41) than standard MCTS with discretized actions (which uses 49 actions). This reduction of actions allows MCTS-APW and

Method	ρ_{avg}	ρ_{max}	ρ_{min}	#offroad	#actions
MCTS (ω=7, ϕ=7)	-309.2	1773.2	$\mathbf{-953.2}$	66/100	49
MCTS-APW ($k = 40$, $\alpha = 0$)	-809.8	1371.8	-956.8	91/100	41
MCTS-APW2 ($k = 40$, $\alpha = 0$, $\epsilon = 0.4$)	**48.3**	**1774.7**	-955	**42/100**	**41**

Table 1
Summary of performance among MCTS, MCTS-APW, and MCTS-APW2.

MCTS-APW2 to perform more simulations for each action on average, developing the tree more in depth than standard MCTS. MCTS-APW2 is also more precise in the selection of the actions, managing to pass through the bottleneck better than other methods.

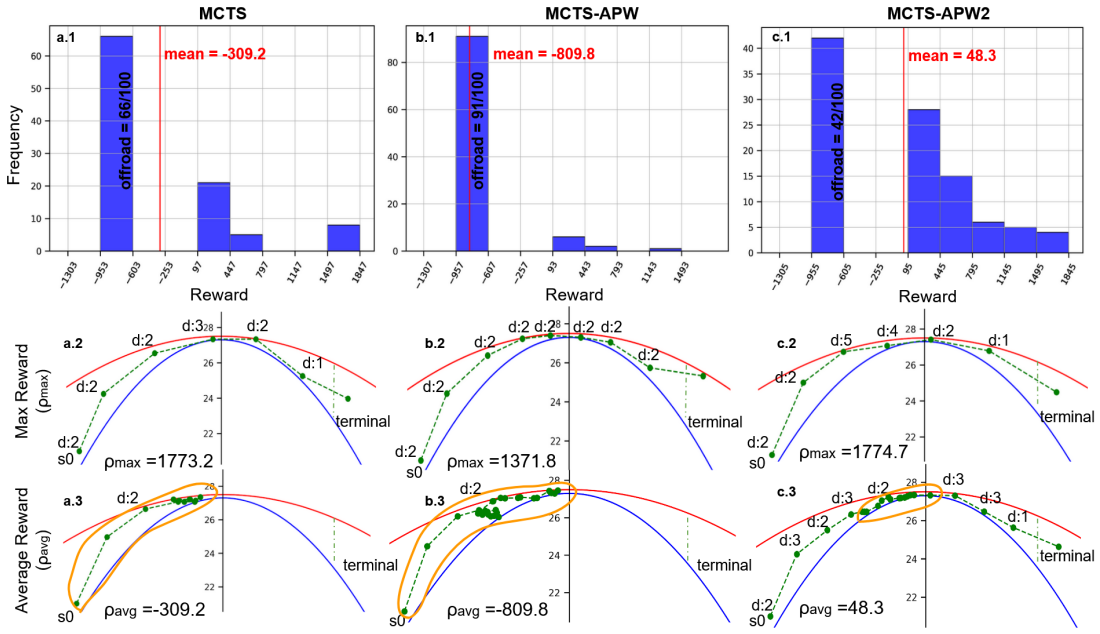

Figure 1: Distributions of reward and analysis of the trajectories in the best and in the average case for each method.

Figure 1 shows the distributions of rewards for all the methods, in particular, the 58% of the mass for MCTS-APW2 is positive, which means that the episodes are completed and the agent is able to pass the curve and reaches the target without getting off the road. For MCTS and MCTS-APW the mass of positive rewards decreases respectively to 34% and 9%. Below the performance distributions, the bottom part of Figure 1 shows, for each method, the trajectory with maximum reward (first row) and the trajectory with reward closer to the average reward (second row). For each scenario, we show the number of steps that methods take to complete the episode and the mean depth (d) of the tree for each step of the episode. In the best case, MCTS-APW2 and MCTS reach the target in 6 steps, while MCTS-APW takes 8 steps, but MCTS-APW2 has a deeper search tree than the other methods and this results in a longer planning horizon. For instance, we can notice the first step of trajectory in Figure 1.c.3 in which

MCTS-APW2 computes a wider trajectory than MCTS to pass faster through the bottleneck. In the average case, we can notice that the number of steps for completing the episode increases and that MCTS-APW2 has better accuracy in choosing the actions and is the only able to compute a trajectory that reaches the goal (see Figure 1.c.3), even if the performance of the proposed method are improvable and far from the optimal. MCTS (see Figure 1.a.3) reaches the bottleneck while reducing the velocity and remains stuck. Analyzing the trajectories we notice that the discretization of the action space introduced by MCTS reduces the accuracy of actions preventing the robot to pass the bottleneck (and the curve) in several cases. In MCTS-APW the random sampling of new actions in the continuous space implies that the exploration of action space is often inefficient and yields a performance decrease, on average, with respect to standard MCTS. The new sampling strategy proposed in MCTS-APW2 manages instead to solve the problem of action space exploration and to increase the average return from -309.2 to 48.3.

5. Conclusions and future work

In this study, we consider different MCTS-based planning methods that work on discretized action space and continuous state/action spaces. We propose a new method that extends the state-of-the-art technique on action progressive widening in continuous domains and we show that the method we proposed outperforms the state-of-the-art. Future work will focus on the improvement of the performance of our method and on extending the analysis to other progressive widening techniques. Future work will also propose a Python framework that implements all the algorithms for MCTS-based planning in continuous action/state spaces and some environments for evaluation.

References

[1] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, second ed., The MIT Press, 2018.

[2] T. Bandyopadhyay, K. S. Won, E. Frazzoli, D. Hsu, W. S. Lee, D. Rus, Intention-aware motion planning, in: Algorithmic foundations of robotics X, Springer, 2013, pp. 475–491.

[3] H. Gupta, B. Hayes, Z. Sunberg, Intention-aware navigation in crowds with extended-space POMDP planning, in: 21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022, Auckland, New Zealand, May 9-13, 2022, 2022, pp. 562–570.

[4] Y. Liu, G. Nejat, Robotic urban search and rescue: A survey from the control perspective, Journal of Intelligent & Robotic Systems 72 (2013) 147–165.

[5] D. Bertsimas, S. Gupta, G. Lulli, Dynamic resource allocation: A flexible and tractable modeling framework, European Journal of Operational Research 236 (2014) 14–26.

[6] R. Coulom, Efficient selectivity and backup operators in monte-carlo tree search, in: H. J. van den Herik, P. Ciancarini, H. H. L. M. J. Donkers (Eds.), Computers and Games, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 72–83.

[7] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen,

S. Tavener, D. P. Liebana, 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.
[8] 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. 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.
[9] D. Silver, J. Veness, Monte-Carlo planning in large POMDPs, in: Advances in Neural Information Processing Systems, NeurIPS 2010, 2010, pp. 2164–2172.
[10] A. Castellini, G. Chalkiadakis, A. Farinelli, Influence of State-Variable Constraints on Partially Observable Monte Carlo Planning, in: IJCAI 2019, Macao, China, August 10-16, 2019, ijcai.org, 2019, pp. 5540–5546.
[11] 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, New York, NY, USA, 2022, p. 739–747.
[12] G. Mazzi, A. Castellini, A. Farinelli, Identification of unexpected decisions in partially observable monte-carlo planning: A rule-based approach, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), International Foundation for Autonomous Agents and Multiagent Systems, 2021, p. 889–897.
[13] G. Mazzi, A. Castellini, A. Farinelli, Rule-based shielding for partially observable monte-carlo planning, in: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), volume 31, 2021, pp. 243–251.
[14] G. Mazzi, D. Meli, A. Castellini, A. Farinelli, Learning logic specifications for soft policy guidance in pomcp, in: Proceedings of the 22nd Conference on Autonomous Agents and MultiAgent Systems, 2023, p. in publication.
[15] A. Castellini, E. Marchesini, A. Farinelli, Partially observable monte carlo planning with state variable constraints for mobile robot navigation, Eng. Appl. Artif. Intell. 104 (2021) 104382.
[16] M. Zuccotto, M. Piccinelli, A. Castellini, E. Marchesini, A. Farinelli, Learning state-variable relationships in POMCP: A framework for mobile robots, Frontiers Robotics AI 9 (2022).
[17] G. M. J. Chaslot, M. H. Winands, H. J. v. d. Herik, J. W. Uiterwijk, B. Bouzy, Progressive strategies for monte-carlo tree search, New Mathematics and Natural Computation 4 (2008) 343–357.
[18] Y. Wang, J.-Y. Audibert, R. Munos, Algorithms for infinitely many-armed bandits, 2008, pp. 1729–1736.
[19] A. Couetoux, Monte carlo tree search for continuous and stochastic sequential decision making problems, 2013.
[20] 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, 2018.
[21] M. H. Lim, C. J. Tomlin, Z. N. Sunberg, Sparse tree search optimality guarantees in pomdps with continuous observation spaces, arXiv preprint arXiv:1910.04332 (2019).
[22] M. H. Lim, C. J. Tomlin, Z. N. Sunberg, Voronoi progressive widening: Efficient on-

line solvers for continuous state, action, and observation pomdps, in: 2021 60th IEEE Conference on Decision and Control (CDC), IEEE Press, 2021, p. 4493–4500.
[23] R. Bellman, A markovian decision process, Journal of Mathematics and Mechanics 6 (1957) 679–684.
[24] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.
[25] 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.
[26] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2002) 235–256.
[27] L. Kocsis, C. Szepesvári, Bandit based monte-carlo planning, in: Proceedings of the 17th European Conference on Machine Learning, ECML'06, Springer-Verlag, 2006, p. 282–293.

A. Supplementary material

In this section, we provide the pseudo-code of procedures that are in common among the methods we compared, namely *PLAN* and *ROLLOUT*. For simplicity, we provide only the pseudo-code of MCTS-APW2. For the pseudo-code of MCTS-APW please refer to this paper [19].

A.1. Pseudo-code of common procedures and MCTS-APW2

Algorithm 1 Common procedures

procedure ROLLOUT(s, d)
  if $d = 0$ **then**
    return 0
  end if
  $a = \pi_0(S)$
  return $r = r + \gamma$Rollout($s', a, d - 1$)
end procedure

procedure PLAN(r)
  for $i \in n$ **do**
    Simulate(r, d_{max})
  end for
  return $\underset{a}{\operatorname{argmax}}\, Q(s)$
end procedure

Algorithm 2 MCTS-APW2

procedure NewActionSelection($A(s)$)
  $p = \text{Unif}[0, 1]$
  if $p < \epsilon$ **then**
    $a_1, a_2 =$ select two actions with the two highest $Q(s)$
    a = mean(a_1, a_2)
  else
    a = randomAction()
  end if
  return a
end procedure

procedure NewActionProgressiveWidening( )
  if $||A(s)|| < kN(s)^{\alpha}$ **then**
    if $N(s) = 0$ **then**
      $a = \text{median}(A(s))$
    else if $N(s) = 1$ **then**
      $a = \text{min}(A(s))$
    else if $N(s) = 2$ **then**
      $a = \text{max}(A(s))$
    else
      $a = \text{NewActionSelection}(A(s))$
    end if
    $N(s, a), Q(s, a), V(s, a) = N_0(s, a), Q_0(s, a), V(s, a)$
    $A(s) = A(s) \cup \{a\}$
  end if
  return $\underset{a}{\text{argmax}}\, Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$
end procedure

procedure Simulate(s, d)
  if $d = 0$ **then**
    return 0
  end if
  if $s \notin T$ **then**
    $T = T \cup \{s\}$
    $N(s) = N_0(s)$
    return Rollout(s, d)
  end if
  a = NewActionProgressiveWidening()
  $s', r = G(s, a)$
  $q = r + \gamma$SIMULATE($s', d - 1$)
  $N(s, a) = N(s, a) + 1$
  $Q(s, a) = Q(s, a) + \frac{q - Q(s,a)}{N(s,a)}$
  return q
end procedure