ELEKTROTEHNIŠKI VESTNIK 82(3): 85–92, 2015
ORIGINAL SCIENTIFIC PAPER

Forgetting Early Estimates in Monte Carlo Control Methods

Tom Vodopivec and Branko Šter

University of Ljubljana, Faculty of Computer and Information Science
Večna pot 113, 1000, Ljubljana, Slovenia

E-mail: tom.vodopivec@fri.uni-lj.si, branko.ster@fri.uni-lj.si

Abstract.
Monte Carlo algorithms are one of the three main reinforcement learning paradigms that are capable of efficiently solving control and decision problems in dynamic environments. Through sampling they shape the values of states in the search space. Based on these values they develop an exploration policy that is in turn used to guide the future direction of sampling. Studies confirm the convergence of this interleaving iterative approach to an optimal solution; however, when a learning agent lacks prior knowledge of the problem domain, the convergence rate may be extremely slow in case of an erroneous staring policy that causes far-from-optimal value estimates. In this paper we present a brief overview of Monte Carlo control algorithms in the scope of reinforcement learning and propose a method to improve the convergence by gradually forgetting early estimates. Our method keeps track of the state values with a moving average that gives a higher weight to the recent rewards and discounts the weight of the previous rewards, while assuming that the policy is improving over time. We apply it to the general on-policy Monte Carlo control algorithm and to the popular upper confidence bounds for trees algorithm in the Monte Carlo tree search framework. The evaluation on several decision problems confirms that our method regularly improves the convergence rate of both algorithms and in some cases also their final policy.

Keywords: Monte Carlo control, reinforcement learning, decision problem, on-line learning, on-policy Monte Carlo control, Monte Carlo tree search, upper confidence bounds for trees

POZABLJANJE PRETEKLIH VZORCEV PRI ODLOČITVENIH METODAH MONTE CARLO

Algoritmi na podlagi vzorčenja Monte Carlo predstavljajo enega od temeljnih načinov za reševanje odločitvenih problemov v domeni spodbujevalnega učenja. S preizkušanjem ocenjujejo vrednosti delov prostora stanj in akcij ter na podlagi teh vrednosti sprotno spreminjajo strategijo preiskovanja, kar vpliva na nadaljnje vrednotenje. Opisan ponavljajoči proces dokazano konvergira k optimalni odločitveni strategiji. Na hitrost konvergence vpliva predznanje ali, ko predznanja ni, ocene pridobljene iz začetnih vzorcev. Toda, če je začetna strategija slaba, potem so lahko tudi začetna vrednotenja daleč od optimalnih, kar bistveno upočasni konvergenco. V tem prispevku na kratko predstavljamo metode Monte Carlo na področju spodbujevalnega učenja in predlagamo način, kako pohitriti konvergenco s pomočjo postopnega pozabljanja preteklih vzorcev. Naša metoda uporablja za vrednotenje uteženo drseče povprečje, ob predpostavki, da so zaradi izboljšanja strategije na zadnje pridobljeni vzorci bolj zanesljivi od predhodnih. Metodo smo preizkusili na dveh uveljavljenih algoritmih: *on-policy Monte Carlo control* in *upper confidence bounds for trees*. Eksperimentalna evalvacija na petih odločitvenih problemih potrjuje, da naša metoda izboljša hitrost konvergence obeh algoritmov in pod določenimi pogoji tudi izboljša končno doseženo strategijo.

Received 26 January 2015
Accepted 5 February 2015

1 INTRODUCTION

Machine learning is roughly divided into three main branches: supervised, unsupervised and reinforcement learning. The first is very useful for regression and classification. The second has the ability to find hidden patterns, which is applicable for clustering and autocorrelation. The third is most suitable in solving control and decision problems.

Reinforcement learning is regarded as one of the most general known learning paradigms, because of the fact that it can effectively learn from a simple feedback in the form of a scalar signal [1]. In such a setting, Monte Carlo methods are used to gather experience by sampling interactions with the environment. Monte Carlo concepts are present in many state-of-the-art reinforcement learning techniques; however, one of their main issues is that the initial sample interactions may not be adequate estimates of the optimal solution. Since these estimates may slow down convergence, we propose to ignore them to a certain extent. We alter the state-evaluation mechanism to discount the impact of past experience. For rare events, we give more weight to the current rewards and less to the past rewards, while for the frequent events the contrary. We integrate this concept into the general on-policy Monte Carlo control algorithm [2] and into a Monte Carlo tree search algorithm [3] – the upper confidence bounds for trees algorithm [4]. We prove the soundness of our idea with experimental evaluations on an artificial decision problem and on four classical board games. The results show a faster convergence to a closer near-optimal value and a higher level of play in certain games.

2 Background

We provide an overview of the basic principles of reinforcement learning and Monte Carlo control methods, and describe the original variants of the two evaluated algorithms.

2.1 *Reinforcement Learning*

Reinforcement learning (RL) is usually thought of as a class of problems rather than a class of methods. It characterizes a specific set of learning problems from multiple domains, including control theory, game theory, optimal control, etc. The methods that can solve such problems are called RL methods [2].

The goal of an RL method is to teach a *learning agent*, i.e., the controller, of behaving well in a dynamic *environment*. The environment feeds back what is "good" in the form of a reward or penalty, which is called positive or negative *reinforcement*. By definition, reinforcement is a single scalar signal and is usually delayed in time. The agents behaviour is defined by the *actions* it takes. The environment *state* can be fully or partially observable by the agent. This agent-environment framework is shown in Fig. 1.

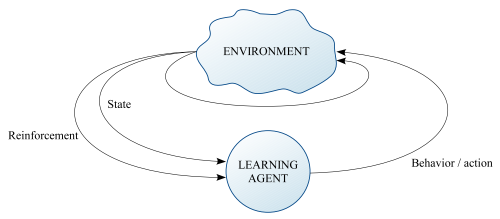


Figure 1. Standard reinforcement learning model [1].

An RL agent tries to learn an action-choosing *policy* in order to *maximize* the sum of all future reinforcements. For assistance in which actions to select, it estimates the *values* of the known states. A value is defined as the expected sum of all received future reinforcements, if continuing from a certain state to the *terminal* state. The *value function* defines the values of all states. An optimal value function is such that it enables an optimal policy in terms of reinforcement maximization.

The solution of any RL problem is to compute the optimal value function and the optimal policy. The theory of *Dynamic Programming* provides the means to iteratively compute the optimal value function using the *Bellman equation* [2]. However, the main issue of dynamic programming is that a complete knowledge of the model is required. This is where *Monte Carlo* methods prove useful as they are capable of learning from on-line or simulated sampling interactions with the environment without the knowledge of its dynamics. They gather *experience* with an increasing number of samples. Their essence is in averaging *returns* – the sum of all reinforcements until a terminal state is reached. The described common ground between the RL and MC theory was defined by Barto, Sutton, and many other researchers only after decades of separate treatment of the two fields.

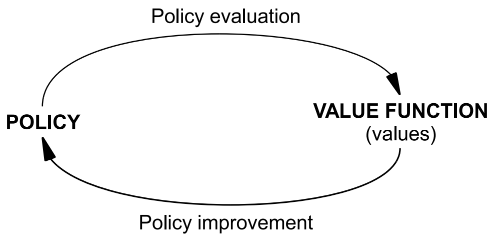


Figure 2. Policy iteration: in each iteration, the policy is evaluated with the the estimation of the value function and then, upon this value function, the policy is greedily improved.

2.2 *Monte Carlo Methods*

In the RL context, Monte Carlo (MC) methods are usually defined only for episodic (not infinite) tasks, so experience is divided into *episodes* and their performance rises in an episode-by-episode (i.e., iterative) fashion. Each occurrence of state s in a single episode is called a *visit* to s. There are two main MC classes: one averages the returns following *every visit* to s; and one averages just the returns following the *first visit*. These averages are used to estimate the value function, based on the current policy. This is called *policy evaluation*. In turn, a greedy *policy improvement* can be made based on the value function. This iterative procedure is called *policy iteration* and is a feature of many RL methods (Fig. 2).

When there is no knowledge of the model dynamics, usually it is more efficient to use *action-values* instead of values. They reflect the same properties as values, but may speed up convergence for the price of extra memory. These values are called Q-values and instead of representing the value of state s, they represent the value of selecting action a in state s – they evaluate the state-action pair (s, a). The condition for convergence is that all (s, a) pairs are visited an infinite number of times, that is why the policy must always keep choosing each of the actions in a state with a probability greater than zero. This is called the problem of exploration or the *exploration-exploitation dilemma*. The trade-off parameter is introduced as the probability ε of not choosing the best action, but rather a sub-optimal one. Such a policy is called an *epsilon-soft policy*.

The problem of exploration when using MC methods is addressed with two approaches, named *off-policy* and *on-policy*. Off-policy MC allows that the policy used for control is separated from the policy being evaluated. There are a behaviour policy and an estimation policy; only the former must be epsilon-soft, while the latter can be deterministic and can fully greedily improve [2]. The second approach, on-policy MC, is described below.

2.3 On-policy Monte Carlo Control

The on-policy algorithm serves as the basis for our improved method. Although policy iteration originally assumes that the policy improvement is completely greedy, it is enough if the improvement is only moved towards a greedy policy, which is the case with epsilon-soft policies [2]. Such policies can then be used to control the agents behaviour and be evaluated and improved at the same time. The basic on-policy MC control is described in Algorithm 1. Decreasing the exploration parameter ϵ in time often yields a higher performance. The decrease is usually linear or exponential, depending on whether the total number of iterations is known in advance or if the task is finite or infinite.

2.4 Monte Carlo Tree Search

Monte Carlo tree search (MCTS) methods [3] are currently the state-of-the-art choice for a wide range of game-playing algorithms [5]. The main difference with ordinary MC methods is that, based on the gathered experience, MCTS methods build a state-action tree. In the MCTS framework, the MC concept of an episode is often referred to as an *MCTS iteration*. An iteration is typically composed of four steps: (1) descend through the tree (i.e., memorized experience) by selecting nodes according to a *tree policy*, (2) expand the tree with one or more nodes visited in the current iteration, (3) choose actions according to a *default policy* until a terminal state is reached, and (4) backpropagate the gathered returns up through the tree.

2.5 Upper Confidence Bounds for Trees

Upper confidence bounds for trees (UCT) [4] was one of the first developed MCTS algorithms and is still the main representative of the field. Furthermore, it is also one of the most studied and generally effective MCTS algorithms. However, its convergence rate can be very low in practice, especially when the algorithm is not aided by domain-specific enhancements. Improving its performance without losing generality is a current research challenge [5].

Algorithm 1. On-policy Monte Carlo Control [2].

```
LEARNING PARAMETER:
 epsilon <- value between 0.0 and 1.0
            //exploring rate
INITIALIZATION:
 a.  arbitrary epsilon-soft policy
 b.  arbitrary Q-value for each
     state-action pair (s,a)
 c.  sum of returns for each (s,a) set to
     zero
 d.  number of visits for each (s,a) set to
     zero
INFINITE LOOP:
 (1) Generate an episode (iteration) by the
     current policy and remember visited
     states, actions, and reinforcement
 (2) Policy evaluation - for each (s,a) that
     appears in the episode (iteration):
   a.  R <- sum of all rewards following
          the first occurrence
   b.  Add R to the sum of returns
   c.  Increase number of visits by one
   d.  Calculate new Q-value from
       sum of rewards and number of visits
 (3) Policy improvement - for each state (s)
     in the episode (iteration):
   a.  best_action <- action with maximal
                      Q-value(s,a)
   b.  change probability of all actions
       (s,a) to epsilon
   c.  change probability of (s,best_action)
       to
       (1-epsilon+epsilon/number_of_actions)
```

The UCT algorithm expands the tree by one node per iteration, uses a random default policy, and backpropagates the reward through all visited nodes in the tree from the leaf node up to the root. It evaluates states with the bandit algorithm UCB1 [6], which has been proven to optimally balance exploration of the space and exploitation of the gathered experience; it theoretically solves the exploration-exploitation dilemma. The tree policy selects children nodes with the highest value of

$$Q_{\mathrm{UCT}} = Q_{\mathrm{MC}} + c_{n_{\mathrm{p}},n} \ , \tag{1}$$

where

$$Q_{\mathrm{MC}} = \frac{\sum R_i}{n} \tag{2}$$

is the average reward received from a node (i.e., state) until the terminal state and

$$c_{n_{\mathrm{p}},n} = 2C_p\sqrt{\frac{2\log n_{\mathrm{p}}}{n}} \tag{3}$$

is the exploration bias defined by the number of visits n of a node, number of visits of its parent node n_{p}, and weighting parameter C_p. Rewards R_i are gathered from MCTS iterations i.

3 Related Work

Our idea may be classified as a weighted Monte Carlo approach. Avellaneda et al. [7] developed a similar approach by assigning probability weights to samples to reduce the variance of results. Their method is applicatively used for building and calibrating asset-pricing market models [8]. Haghighat et al. [9] also developed a method for variance reduction by introducing deterministic importance functions as a weighting mechanism. A review and improvement of certain variance reduction techniques, including importance sampling, was presented in Saltas dissertation [10]. Another approach similar to ours, by using adaptation of the step-size parameter, was researched in the domain of temporal difference learning by Noda [11], but to our knowledge, it has not been applied to MC methods yet.

In the context of MCTS, Xie and Liu [12] studied the idea of weighting MCTS iterations according to their recency. They assume that early iterations contain less reliable value estimates, thus can be given a lower weight. In the backpropagation step, they partition MCTS iterations in groups with different weights. Their results confirm that later partitions deserve a higher weight, since they generally contain shorter and more accurate iterations. We rely on the same assumptions; however, we employ a simpler weighting function that uses a smoother weighting distribution and introduces less parameters to the enhanced algorithms.

4 Discounting Early Estimates With Moving Average Values

Both on-policy MC control and UCT estimate state-action values $Q_{(s,a)}$ as a statistical mean of gathered rewards R_i through several episodes or iterations i. This way, a single state-action value is iteratively updated by the rule

$$Q_{(s,a)} \leftarrow Q_{(s,a)} + \frac{1}{n_{(s,a)}}(R_i - Q_{(s,a)}) \ , \qquad (4)$$

where $n_{(s,a)}$ represents how many times action a was selected in state s (i.e., a visit count of the state-action pair that increases by 1 with each visit).

In improving the original MC algorithms we rely on the idea that initial Q-values may be unreliable and could corrupt the optimal value estimates at a later time. Driven by this intuition, and by the results presented by Xie and Liu [12], instead of using (4), we diminish the importance of past Q-values with an exponential moving average

$$Q_{(s,a)} \leftarrow (1 - \alpha_{(s,a)})\ Q_{(s,a)} + \alpha_{(s,a)} R_i \ , \qquad (5)$$

where the discount rate is controlled by the weighting factor $\alpha_{(s,a)}$ in the range of $[0, 1]$. The higher its value,

Algorithm 2. Extension of On-policy Monte Carlo Control with Moving Average Values

```
LEARNING PARAMETERS:
 beta <- value between 0.0 and 1.0
  //discount rate of previous rewards
...
 (2) Policy evaluation - for each (s,a) that
     appears in the episode (iteration):
...
   d1. alpha <- beta * current_episode /
                number_of_visits
   d2. if(alpha > 1)
          then alpha <-1.0
          //threshold or sigmoid function
   d3. Calculate new Q-value for (s,a) by
       exponential moving average:
       (1-alpha) * Q-value + (alpha) * R
...
```

the higher weight is given to current reinforcement and lower to memorized Q-values, and vice versa. Its value is adjusted in each iteration i for each state-action pair (s, a) by the equation

$$\alpha_{(s,a)} = \beta * \frac{i}{n_{(s,a)}} \ , \qquad (6)$$

where β is a constant parameter in the range of $[0, 1]$ that defines the discount rate of previous rewards. A higher value of β and a lower number of visits to (s, a) give more importance to recent rewards and less importance to past experience. Parameter β should have a value less or equal to the inverse of the maximum number of visits of a (s, a) pair, otherwise $\alpha_{(s,a)}$ can reach values greater than 1 for less visited states and divergence may occur. However, the maximum number of visits may not be predictable, due to the possibility of multiple visits in a single episode, or due to the lack of knowledge of the total number of episodes or iterations. A threshold or sigmoid function $f_{\mathrm{T}}(x)$ over $\alpha_{(s,a)}$ solves this issue. Note the resemblance with (4) when reformulating (5) and (6) into the following update rule

$$Q_{(s,a)} \leftarrow Q_{(s,a)} + f_{\mathrm{T}}\left(\beta * \frac{i}{n_{(s,a)}}\right)(R_i - Q_{(s,a)}) \ . \qquad (7)$$

We name the method described by this equation as the Moving Average Values (MAV) for discounting early MC estimates. Modification of on-policy MC control with MAV is straightforward and is shown in Algorithm 2.

MAV-UCT

We integrate MAV into the UCT algorithm by modifying its tree descent and backpropagation steps. In the tree descent we extend (1) to evaluate a node with both the default statistical mean Q_{MC} and the new moving average value Q_{MAV} by

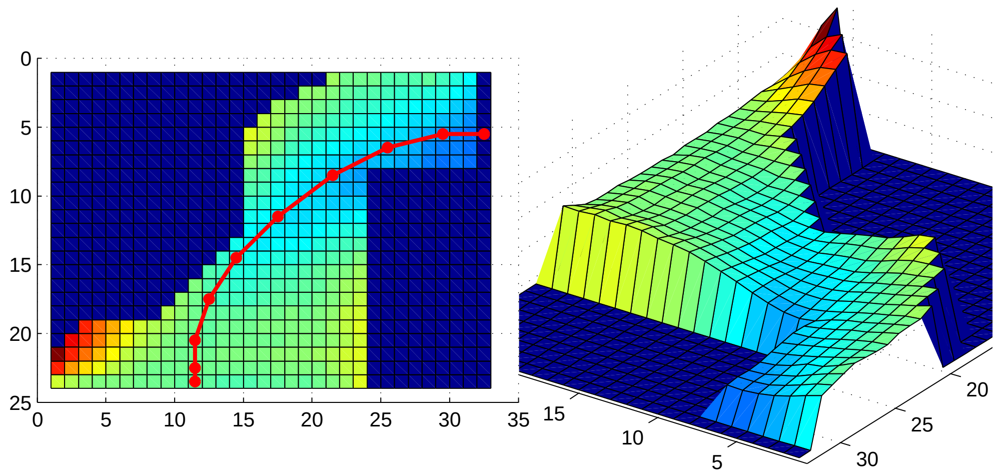


Figure 3. Optimal state-value estimates for the Racetrack problem and an example optimal policy after a million iterations of the on-policy MC control algorithm.

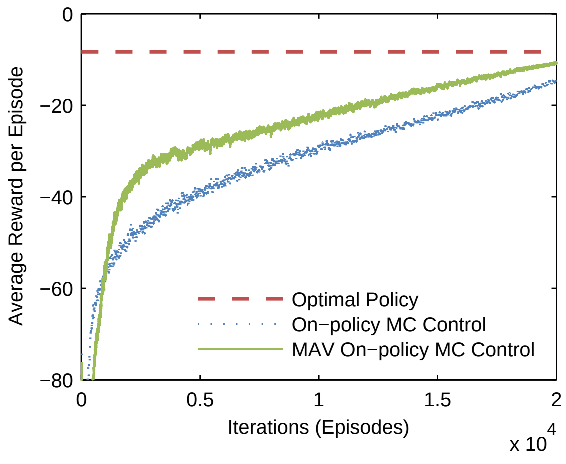


Figure 4. Performance of the original and MAV-enhanced on-policy MC control algorithms on the Racetrack problem.

$$Q_{\text{MAV-UCT}} = w_{\text{MAV}}\ Q_{\text{MAV}} + (1 - w_{\text{MAV}})\ Q_{\text{MC}} + c_{n_{\text{p}},n}\ , \quad (8)$$

where w_{MAV} is an additional parameter that defines the weight of the MAV contribution to a node value. Several MCTS enhancements also use such weighting as it allows a linear combination (i.e., a smooth transition) of two or more evaluators. The Q_{MAV} values are computed in the UCT backpropagation step by (5).

Since the MCTS tree memorizes the experience gained from the previous batches of iterations, started from several different root nodes, i.e., from different locations in the state space, Equ. (6) must be adjusted to consider iterations i' only in the current batch. The same applies to visits n'. This requires the implementation of an additional visit counter for each tree node, whose value is zeroed after each batch of iterations or every time the tree root changes. Here, the discount rate of previous rewards is calculated by

$$\alpha_{(s,a)} = f_{\text{T}}\left(\beta * \frac{i'}{n'_{(s,a)}}\right)\ . \quad (9)$$

In our experiments we implemented $f_{\text{T}}(x)$ as an adjusted sigmoid function by

$$f_{\text{T}}(x) = \left(\frac{1}{1+e^{-x}} - 0.5\right) * 2.0\ . \quad (10)$$

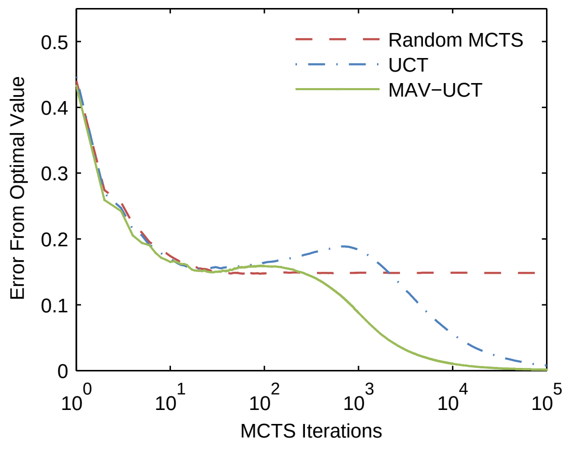
Since discount rate $\alpha_{(s,a)}$ is at least higher or equal to zero for any state-action pair, the function will never return a negative value.

5 Experimental Results

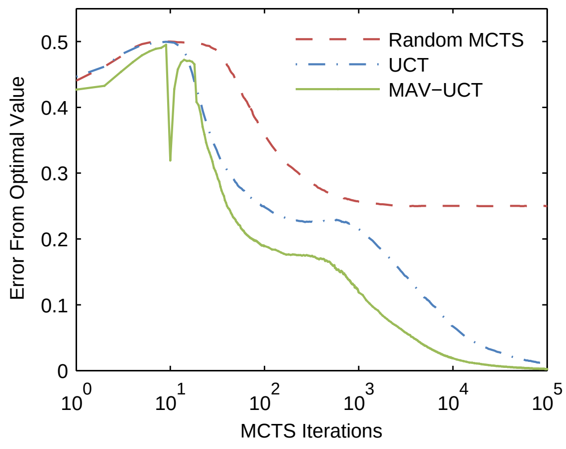
We evaluate the MAV on-policy MC control algorithm and the MAV-UCT algorithm on several benchmark problems and compare their performances with the original variants presented in Section 2. The first batch of experiments presents on-policy MC on a simple artificial problem, while the second batch presents UCT on four benchmark games of a different complexity.

5.1 *On-policy MC on the Racetrack Problem*

As a benchmark for the on-policy MC algorithm we use the Racetrack problem presented by Sutton and Barto [2]. It consists of a two-dimensional grid world with movement constraints in the shape of a right curve and a moveable car. The car has a vertical and a horizontal integer velocity in the range of $[0, 4]$. The controller may increase, decrease, or keep the same velocity, which yields nine actions per state. The car starts stationary at a random bottom position. The goal is to reach one of



(a) Root node error from the optimal value.



(b) Root best child node error from the optimal value.

Figure 5. Performance of MCTS algorithms on Tic Tac Toe expressed as error from the optimal value.

the far right positions in the minimum number of steps without hitting the edge. The environment returns a -1 reward in each time step and -4 if the car hits the edge of the track. Furthermore, the car is randomly moved by an additional tile up or right with a 50% probability in each time step. Figure 3 shows how on-policy MC control learns the optimal state value estimates and an optimal policy after a million episodes (the values are negated for better visualization). Positions near borders have overall lower values (i.e., higher penalty) than those in the middle.

We measured the performance of the original and MAV-enhanced on-policy algorithms as the average gathered reward per episode, which is analogous to the required number of steps to reach the terminal position. These measurements are presented as a function of the amount of available learning episodes (Fig. 4). We linearly decreased exploring rate ϵ from 0.2 to 0 as the total number of episodes was known. MAV parameter β was set to 0.0012. The parameter values were hand-picked based on repeated experiments. The results were averaged from 100 repeats.

The best reachable performance for the given problem is -8.3. After 20000 episodes, the original on-policy MC control achieves -14.6, while with MAV, it achieves -10.7, which is substantially closer to the optimal value. On the other hand, in the first few hundred episodes, the MAV-variant performs worse than the original algorithm.

5.2 *UCT on Games*

First we evaluate the MCTS algorithms on the game of Tic Tac Toe. This simple game is known to always end in a draw if both players play optimally. In our case, the environment feedbacks 1 for a win, 0.5 for a draw, and 0 for a loss. Consequently, the optimal values of the first two moves, i.e., the root node in the MCTS tree and its best child (i.e., the best action from the root), are both 0.5.

Table 1. Playing strength of UCT and MAV-UCT on four games. The opponent uses the original UCT algorithm with $C_p = 0.2$ and always plays first. The learning parameters of the two evaluated algorithms were optimized offline.

	Tic Tac Toe	Connect Four	Gomoku 7x7	Hex 7x7
Iterations/Move	10	100	100	100
Win Rates [%]				
Original UCT	33.3	44.4	55.2	57.8
MAV UCT	34.0	44.7	59.1	59.9
95% conf. interval	±0.3	±0.3	±1.0	±1.0
MAV UCT Optimal Parameter Values				
C_p	.040	.176	.015	.090
w_{MAV}	.385	.003	.236	.359
β	.043	.119	.004	.007

We comparatively evaluate three algorithms: MCTS with a random tree policy, UCT, and MAV-UCT. We generate a large amount of MCTS iterations from the initial game state (i.e., an empty board) and measure the absolute error of the root node (Fig. 5a) and its best child (Fig. 5b). In both cases, MAV-UCT performs best as it approximates the optimal values in much less iterations (note the exponential x-axis) than the other two algorithms, and performs at least as well as the others at a lower number of iterations. The results are averaged from 10000 repeats.

Finally, we evaluate the playing strength of UCT and MAV-UCT on four games: Tic Tac Toe, Connect Four, Gomoku, and Hex. Such games are common benchmarks in the MCTS domain [5]. These are more complex than the Racetrack problem and are more suitable for a general evaluation of new methods. The performance is expressed as the win rate against a fixed opponent, i.e., an UCT algorithm with $C_p = 0.2$ that always plays as the first player. The evaluated algorithms had their learning parameters C_p, w_{MAV}, and β optimized prior to the evaluation with a Linear Reward-Penalty Learning Automata [13] on 10000 games.

The results in Table 1 show an evident improvement of MAV-UCT over UCT at Gomoku, a slight improvement at Hex and Tic Tac Toe, and no improvement at Connect Four. In the latter, the weight of MAV contributions w_{MAV} was set to nearly zero by the optimization procedure. This further confirms that MAV may be ineffective at Connect Four. We experimented also at higher numbers of iterations per move, but there was no improvement in win rates. The results are averaged from several thousand repeats. We did not extensively test the robustness to changing the parameter values; however, preliminary experiments show a very high sensibility to discount rate β.

6 Discussion and Conclusion

In this paper we briefly presented the field of reinforcement learning [2], mechanics of learning with Monte Carlo (MC) control methods, and Monte Carlo Tree Search (MCTS) methods [3]. We proposed a new weighted MC algorithm, i.e., the moving average values (MAV), to alleviate the issue of unreliable early MC estimates due to a sub-optimal initial action policy. Our MAV method lowers the impact of previous estimates by using a moving average that is weighted according to the frequency of visits of individual parts of the state space. We applied MAV to the on-policy MC control algorithm [2] and the upper confidence bounds for trees (UCT) algorithm [4] and experimentally validated its performance on five problems with different dynamics.

Results indicate that MAV increases the convergence rate of both on-policy MC control and UCT on the Racetrack problem and at the game of Tic Tac Toe. However, when assessing the level of play expressed as the win rate of the MAV-UCT algorithm, it achieved a noticeable improvement only at one of the four benchmark games, i.e., Gomoku, and only at a lower number of iterations per move. Preliminary results also revealed that MAV's performance is critically sensitive to the value of its newly-introduced parameter β for discounting past rewards, so more experiments need to be done in order to fully analyse its robustness.

Considering that MAV-UCT introduces two new parameters and does not seem to generally increase performance, its use may be limited only to specific problems at this stage. We have just started to explore why the playing strength does not improve despite the faster convergence to the optimal value estimates. When progressing through the game, the dynamics of the environment changes considerably as the player is moving closer to the terminal position. We presume this may critically affect the optimal value of discount parameter β, which causes MAV to become ineffective in such a setting. For the future work, we propose to dynamically adapt β through the game duration and to identify the general features of the state space and of the MCTS tree structure that affect its optimal value.

References

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* The MIT Press, 1998.

[3] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proceedings of the 5th international conference on Computers and games*, ser. CG'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 72–83. [Online]. Available: http://dl.acm.org/citation.cfm?id=1777826.1777833

[4] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Proceedings of the Seventeenth European Conference on Machine Learning*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Berlin/Heidelberg, Germany: Springer, 2006, pp. 282–293. [Online]. Available: http://www.sztaki.hu/ szcsaba/papers/ecml06.pdf

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[6] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1013689704352

[7] M. Avellaneda and P. Jäckel, "Weighted Monte Carlo," *Encyclopedia of Quantitative Finance*, 2010.

[8] M. Avellaneda, R. Buff, C. Friedman, N. Grandchamp, N. Gr, L. Kruk, and J. Newman, "Weighted Monte Carlo: A New Technique for Calibrating Asset-Pricing Models," *International Journal of Theoretical and Applied Finance*, vol. 4, pp. 1–29, 2001.

[9] A. Haghighat and J. C. Wagner, "Monte Carlo Variance Reduction with Deterministic Importance Functions," *Progress in Nuclear Energy*, vol. 42, no. 1, pp. 25–53, 2003.

[10] E. R. Salta, "Variance Reduction Techniques in Pricing Financial Derivatives," Ph.D. dissertation, Florida State University, 2008.

[11] I. Noda, "Recursive Adaptation of Stepsize Parameter for Non-stationary Environments," in *Proceedings of the 12th International Conference on Principles of Practice in Multi-Agent Systems*, ser. PRIMA '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 525–533. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11161-7_38

[12] F. Xie and Z. Liu, "Backpropagation Modification in Monte-Carlo Game Tree Search," in *Proceedings of the Third International Symposium on Intelligent Information Technology Application - Volume 02*, ser. IITA '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 125–128. [Online]. Available: http://dx.doi.org/10.1109/IITA.2009.331

[13] K. S. Narendra and M. A. L. Thathachar, *Learning automata - an introduction.* Prentice Hall, 1989.

Tom Vodopivec received his pre-Bolgona B.Sc. degree (comparable to the Bologna M.Sc. degree) from the Faculty of Computer and Information Science of the University of Ljubljana in 2011. He is a Ph.D. student, teaching assistant, and a member of the Laboratory for Adaptive Systems and Parallel Processing at the same faculty. His Ph.D. thesis is focused on Monte Carlo Tree Search algorithms, while his research interests also comprise Reinforcement Learning, Evolutionary Computation, Neural Networks, Game Theory, Parallel Processing, and Machine Learning approaches in general.

Branko Šter is an associate professor at the Faculty of Computer and Information Science, University of Ljubljana. He achieved his M.Sc. degree in Electrical Engineering and Ph.D. degree in Computer and information science both from the University of Ljubljana, in 1996 and 1999, respectively. His research interests include neural networks, pattern recognition, reinforcement learning, and robotics.