Multi-Objective Monte Carlo Tree Search for Real-Time Games

Diego Perez, *Student Member, IEEE*, Sanaz Mostaghim, Spyridon Samothrakis, *Student Member, IEEE*, Simon M. Lucas, *Senior Member, IEEE*

***Abstract*—Abstract...**

I. Introduction

Here it goes, the introduction.

II. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a tree search algorithm that was originally applied to board games, concretely to the two-players game of Go. This game is played in a square grid board, with a size of 19×19 in the original game, and 9×9 in its reduced version. The game is played in turns, and the objective is to surround the opponent's stones by placing stones in any available position in the board. Due to the very large branching factor of Go, this game is considered the drosophila of Game AI, and MCTS players have reached professional level play in the reduced board size version [1]. After its success in Go, MCTS has been used extensively by many researchers in this and different domains. An extensive survey of MCTS methods, variations and applications, has been written by Browne et al. [12].

MCTS is considered to be an *anytime* algorithm, as it is able to provide a a valid next move to choose at any moment in time. This is true independently from how many iterations the algorithm was able to make (although, in general, more iterations usually produce better results). This differs from other algorithms (such A*) that normally provide the next ply only after they have finished. This makes MCTS a suitable candidate for real-time domains, where the decision time budget is limited, affecting the number of iterations that can be performed.

MCTS is an algorithm that builds a tree in memory. Each node in the tree maintains statistics that indicate how often a move is played from a given state ($N(s,a)$), how many times each move is played from there ($N(s)$) and the average reward ($Q(s,a)$) obtained after applying move a in state s. The tree is built iteratively by simulating actions in the game, making move choices based on the statistics store in the nodes.

Each iteration of MCTS can be divided into several steps [10]: *Tree selection*, *Expansion*, *Monte Carlo simulation* and *Back-propagation* (all summarized in Figure 1). When the algorithm starts, the tree is formed only by the root node, which holds the current state of the game. During the *selection* step, the tree is navigated from the root until a maximum depth or the end of the game has been reached.

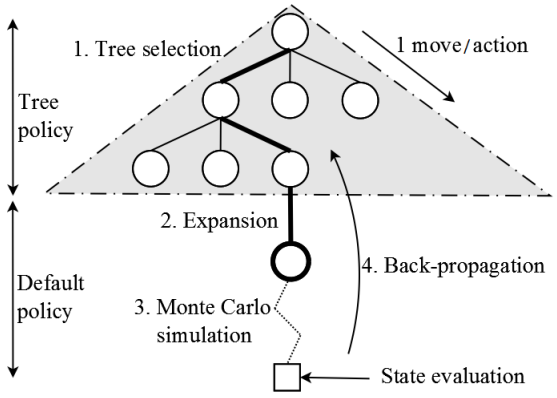

Fig. 1. MCTS algorithm steps.

In every one of this action decisions, MCTS balances between exploitation and exploration. In other words, this chooses between taking an action that leads to states with the best outcome found so far, and performing a move to go to less explored game states, respectively. In order to achieve this, MCTS uses Upper Confidence Bound (UCB1, see Equation 1) as a *Tree Policy*.

$$a^* = \underset{a \in A(s)}{\arg\max} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \qquad (1)$$

The balance between exploration and exploitation is achieved by setting the value of C. Higher values of C weight more the second term of the UCB1 Equation 1, giving preference to those actions that have been explored less, at the expense of taking actions with the highest average reward $Q(s,a)$. A commonly used value is $\sqrt{2}$, as it balances both facets of the search when the rewards are normalized between 0 and 1. It is worth noting that MCTS, when combined with UCB1 reaches asymptotically logarithmic regret [11].

If, during the *tree selection* phase, a node has less children than the available number of actions from a given position, a new node is added as a child of the current one (*expansion* phase) and the *simulation* step starts. At this point, MCTS executes a Monte Carlo simulation (or roll-out; *default policy*) from the expanded node. This is performed by choosing random (either uniformly random, or biased) actions until the game end or a pre-defined depth is reached, where the state of the game is evaluated.

Finally, during the *back-propagation* step, the statistics $N(s)$, $N(s,a)$ and $Q(s,a)$ are updated for each node visited,

Diego Perez, Spyridon Samothrakis, Simon M. Lucas (School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK; email: {dperez,ssamot,sml}@essex.ac.uk);
Sanaz Mostaghim (Department of Knowledge and Language Engineering, Otto-von-Guericke-Universitt Magdeburg, Magdeburg, Germany; email: sanaz.mostaghim@ovgu.de)

using the reward obtained in the evaluation of the state. These steps are executed in a loop until a termination criteria is met (such as number of iterations).

MCTS has been employed extensively in real-time games in the literature. A clear example of this is the popular real-time game *Ms. PacMan*. The objective of this game is to control Ms. PacMan to clear the maze by eating all pills, without being captured by the ghosts. An important feature of this game is that it is *open-ended*, as an end game situation is, most of the time, far ahead in the future and can not be devised by the algorithm during its iterations. The consequence of this is that MCTS, in its vanilla form, it is not able to know if a given ply will lead to a win or a loss game end state. Robles et al [2] solved this problem by including hand-coded heuristics that guided MCTS simulations towards more promising portions of the search space. Other authors also included domain knowledge to bias the search in MCTS, such as in [3], [4].

MCTS has also been applied to single-player games, like SameGame [5], where the player's goal is to destroy contiguous tiles of the same colour, distributed in a rectangular grid. Another use of MCTS is in the popular puzzle Morpion Solitaire [6], a connection game where the goal is to link nodes of a graph with straight lines that must contain at least five vertices. Finally, the PTSP has also been addressed with MCTS, both in the single-objective [7], [8] and the multi-objective versions [9]. These papers describe the entries that won both editions of the PTSP Competition.

It is worthwhile mentioning that in most cases found in the literature, MCTS techniques have been used with some kind of heuristic that guides the Monte Carlo simulations or the tree selection policy. In the algorithm described in this paper, simulations are purely random, as the objective is to compare the search abilities of the different algorithms. The intention is therefore to keep the heuristics to a minimum, and the existing pieces of domain knowledge are shared by all the algorithms presented (as in the case of the score function for MO-PTSP, described later).

III. Multi-Objective Optimization

IV. Multi-Objective Monte Carlo Tree Search

V. Benchmarks

VI. Experimentation

VII. Conclusions and Future Work

Acknowledgments

This work was supported by EPSRC grants EP/H048588/1, under the project entitled "UCT for Games and Beyond".

References

[1] C.-S. Lee, M.-H. Wang, G. M. J.-B. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 73–89, 2009. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4804732

[2] D. Robles and S. M. Lucas, "A Simple Tree Search Method for Playing Ms. Pac-Man," in *Proceedings of the IEEE Conference of Computational Intelligence in Games*, Milan, Italy, 2009, pp. 249–255.

[3] S. Samothrakis, D. Robles, and S. M. Lucas, "Fast Approximate Max-n Monte-Carlo Tree Search for Ms Pac-Man," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 2, pp. 142–154, 2011.

[4] N. Ikehata and T. Ito, "Monte Carlo Tree Search in Ms. Pac-Man," in *Proc. 15th Game Progrramming Workshop*, Kanagawa, Japan, 2010, pp. 1–8.

[5] S. Matsumoto, N. Hirosue, K. Itonaga, K. Yokoo, and H. Futahashi, "Evaluation of Simulation Strategy on Single-Player Monte-Carlo Tree Search and its Discussion for a Practical Scheduling Problem," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 3, Hong Kong, 2010, pp. 2086–2091. [Online]. Available: http://www.iaeng.org/publication/IMECS2010/IMECS2010_pp2086-2091.pdf

[6] S. Edelkamp, P. Kissmann, D. Sulewski, and H. Messerschmidt, "Finding the Needle in the Haystack with Heuristically Guided Swarm Tree Search," in *Multikonf. Wirtschaftsinform.*, Gottingen, Germany, 2010, pp. 2295–2308. [Online]. Available: http://webdoc.sub.gwdg.de/univerlag/2010/mkwi/03_anwendungen/planen_scheduling/05_finding_the_needle_in_the_haystack.pdf

[7] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the Physical Travelling Salesman Problem: Tree Search and Macro-Actions," *IEEE Trans. Comp. Intell. AI Games (submitted)*, 2013.

[8] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 234–241.

[9] ——, "Monte Carlo Tree Search with Macro-Actions and Heuristic Route Planning for the Multiobjective Physical Travelling Salesman Problem," in *Proc. IEEE Conf. Comput. Intell. Games*, 2013, pp. 73–80.

[10] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," Inst. Nat. Rech. Inform. Auto. (INRIA), Paris, Tech. Rep., 2006.

[11] P.-A. Coquelin, R. Munos *et al.*, "Bandit Algorithms for Tree Search," in *Uncertainty in Artificial Intelligence*, 2007.

[12] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. 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. 4:1, pp. 1–43, 2012.

Diego Perez received a B.Sc. and a M.Sc. in Computer Science from University Carlos III, Madrid, in 2007. He is currently pursuing a Ph.D. in Artificial Intelligence applied to games at the University of Essex, Colchester. He has published in the domain of Game AI, participated in several Game AI competitions and organized the Physical Travelling Salesman Problem competition, held in IEEE conferences during 2012. He also has programming experience in the videogames industry with titles published for game consoles and PC.

Sanaz Mostaghim BIO HERE.

PHOTO HERE

Spyridon Samothrakis is currently pursuing a PhD in Computational Intelligence and Games at the University of Essex. His interests include game theory, computational neuroscience, evolutionary algorithms and consciousness.

Simon Lucas (SMIEEE) is a professor of Computer Science at the University of Essex (UK) where he leads the Game Intelligence Group. His main research interests are games, evolutionary computation, and machine learning, and he has published widely in these fields with over 160 peer-reviewed papers. He is the inventor of the scanning n-tuple classifier, and is the founding Editor-in-Chief of the IEEE Transactions on Computational Intelligence and AI in Games.