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. [2].

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 [3]: *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.

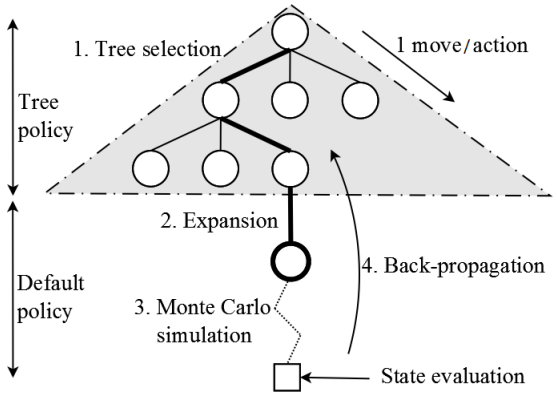

Fig. 1: MCTS algorithm steps.

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 [4].

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 [5] 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 [6], [7].

MCTS has also been applied to single-player games, like SameGame [8], 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 [9], 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 [10], [11] and the multi-objective versions [12]. 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 proposed 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

A multi-objective optimization problem (MOOP) represents a scenario where two or more objective functions are to be optimized (either maximized or minimized). The general form of a MOOP is formally described as a maximization function $F_m(x)$, that transforms points in the decision space (X) to points in the solution space (F). The elements of the decision space are vectors of n variables of the form $x = (x_1, x_2, \ldots, x_n)$, while elements in the solution space are vectors with a dimension m: $F_m(x) = (f_1(x), f_2(x), \ldots, f_m(x))$. Therefore, each solution provides m different scores (or rewards, or fitness) that are meant to be optimized. Without loss of generality, it is assumed from now on that all objectives must be maximized.

It is said that a solution $F_m(x)$ *dominates* another solution $F_m(y)$ if:

1) $F_m(x)$ is not worse than $F_m(y)$ in all objectives for all $i = 1, 2, \ldots, m$.

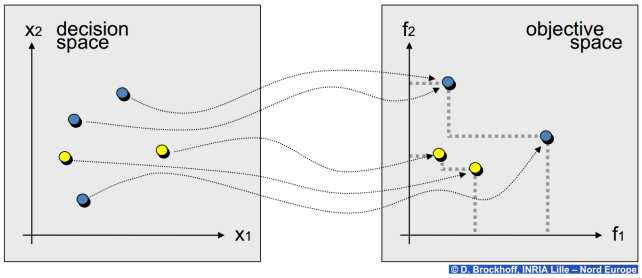

Fig. 2: Decision and Solution spaces in a MOOP with two variables (x_1 and x_2) and two objectives (f_1 and f_2). In the objective space, yellow dots are non-optimal objective vectors, while blue dots form the Pareto-optimal front.

2) At least one objective of $F_m(x)$ is better than its analogous counterpart in $F_m(y)$.

When this two conditions apply, it is said that $F_m(x) \preceq F_m(y)$ ($F_m(x)$ dominates $F_m(y)$, and $F_m(x)$ is non-dominated by $F_m(y)$). The *dominance* condition provides a partial ordering between points in the solution space: if $F_m(x) \preceq F_m(y)$, then $F_m(x)$ is considered to be better than $F_m(y)$.

However, there are some cases where it cannot be said that $F_m(x) \preceq F_m(y)$ or $F_m(y) \preceq F_m(x)$. This situation occurs when one of the objectives is better in $F_m(x)$ but a different objective is better in $F_m(y)$ (for instance, when $F_1(x) < F_1(y)$ but $F_2(x) > F_2(y)$). In this case, it is said that these solutions are non-dominated with respect to each other. Solutions that are not dominated by each other are grouped in a *non-dominated set*. Given a non-dominated set P, it is said that P is the *optimal Pareto front* if there is no other solution in the solution space that dominates any member of P. The relation between decision and objective space, dominance and Pareto fronts is depicted in Figure 2.

As many Pareto front can be formed by several points in the solution space, it is important to devise a mechanism to assess the quality of a given front. A possibility is to use the Hypervolume Indicator (HV): given a Pareto front P, $HV(P)$ is defined as the volume of the objective space dominated by P. More formally, $HV(P) = \mu(x \in \mathbb{R}^d : \exists r \in P \;\; s.t \;\; r \preceq x)$, where μ is the de Lebesgue measure on $\mathbb{R}^d$. If the objectives are to be maximized, the higher the $HV(P)$, the better the front calculated. Figure 3 shows an example of $HV(P)$ where the objective dimension space is 2.

For more extensive descriptions, definitions, properties and multi-objective optimization in general, the reader can consult the work by K. Deb [13].

Many different algorithms have been proposed to tackle multi-objective optimization problems in the literature. One of the most widely known and used methods is the weighted-sum approach. The procedure consists of giving a weight to each one of the objective and produce a single result as the linear combination of objectives and weights. By varying the weights provided, it is possible to converge to different solutions of the optimal Pareto front, if this is convex. However, K. Deb [13] explains how linear scalarization approaches fail in

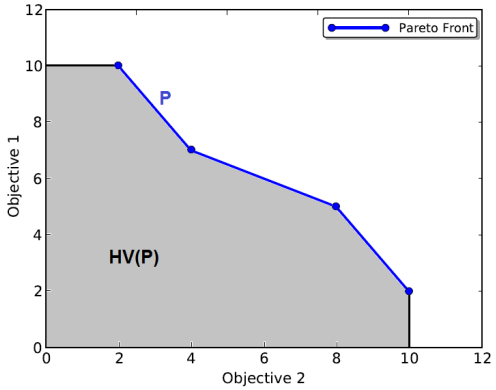

Fig. 3: $HV(P)$ of a given pareto front P.

Algorithm 1 NSGA-2 Algorithm.

function NSGA-2
  $P = NewRandomPopulation$
  while Termination criteria not met **do**
    $R = P \cup Q$
    $F = \text{FASTNONDOMINATEDSORT}(R)$
    while $|P| < N$ **do**
      $\text{CROWDINGDISTANCEASSIGNMENT}(F_i)$
      $P = P \cup F_i$
    $\text{SORT}(P)$
    $P = P[0 : N]$
    $Q = breed(P)$

those scenarios where the optimal Pareto front is non-convex.

A popular choice for multi-objective optimization problems are evolutionary multi-objective optimization (EMOA) algorithms [14], [15]. One of the most well known algorithms in the literature is the Non-dominated Sorting Evolutionary Algorithm 2 (NSGA-2), which pseudocode is shown in Algorithm 1. As in any evolutionary algorithm, NSGA-2 evolves a set of individuals or solutions to the problem, with the difference that here they are ranked according to dominance criteria and crowding distance (distances between members of the pareto fronts). A full description of the algorithm can be found in [16].

The three main pillars of the NSGA-2 algorithm are:

- A *fast non-dominated sorting* algorithm, that ranks the individuals of the population and groups them in Pareto fronts.
- A *crowding distance*, assigned to each one of the individuals, that measures how close it is to its neighbours. The selection genetic operator chooses individuals based on the ranks of the individuals and their crowding distance.
- *Elitism*, implemented so the algorithm automatically promotes the best N individuals to the next generation.

A more recent approach, developed by Q. Zhang, is the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) [17], that decomposes the problem into several single optimization sub-problems and an evolutionary algorithm optimizes them all simultaneously. Information is shared between neighbouring sub-problems in order to guide evolution. The authors show that MOEA/D performs similarly than, and sometimes even outperforms, NSGA-II in the scenarios tested.

Reinforcement Learning (RL) algorithms have also tackled Multi-objective optimization in some scenarios. RL [18] is a broad field in Machine Learning that studies real-time planning and control situations where an agent has to find out the actions (or sequences of actions) that should be applied in order to maximize the reward from the environment.

An RL problem can be defined as a tuple (S, A, T, R, π). S is the set of possible states in the problem (or game), and s_0 is the initial state. A is the set of available actions the agent can make at any given time, and the transition model $T(s_i, a_i, s_{i+1})$ determines the probability of reaching the state s_{i+1} when action a_i is applied in state s_i. The reward function $R(s_i)$ provides a single value (*reward*) that the agent must optimize, representing the desirability of the state s_i reached. Finally, a decision policy $\pi(s_i) = a_i$ determines which actions a_i must be chosen from each state $s_i \in S$. One of the most important challenges in RL, as shown in Section II, is the trade-off between exploration and exploitation. The decision policy can choose between following actions that provided good rewards in the past and exploring new parts of the search space by selecting new actions.

Multi-objective Reinforcement Learning (MORL) [19] changes this definition by using instead a vector $R = r_0, r_1, \ldots, r_n$ as rewards of the problem. Thus, MORL problems differ from RL in having more than one objective objectives (n) that must be maximized. If the objectives are independent or they do not oppose each other, scalarization technique approaches, as described above, could be suitable to tackle the problem. Essentially, this would mean to use a conventional RL algorithm on a single objective obtained by a weighted-sum of the multiple rewards. However, this is not always the case, as it is usual that the objectives are in conflict and the policy (π) must balance among them.

According to how π approaches this problem, Vamplew et al. [19] proposed the following distinction for MORL: *single-policy* algorithms are those that provide a preference order in the objectives available (given by the user or by the nature of the problem). An example of this type of algorithm can be found at [20], where the authors introduce an order of preference in the objectives treated and constraint the value of the rewards desired. Scalarization approaches would also fit in this category, as the work performed by S. Natarajan et al. [21].

The second type of algorithms, *multiple-policy*, target to approximate the optimal Pareto front of the problem. An example of this type of algorithm is the one given by L. Barrett [22], who propose the Convex Hull Iteration Algorithm. This algorithm provides the optimal policy for any linear preference function, by learning all policies that define the convex hull of the Pareto front.

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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] ——, "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.

[13] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.

[14] C. Coello, "An Updated Survey of Evolutionary Multiobjective Optimization Techniques: State of the Art and Future Trends," in *Proc. of the Congress on Evolutionary Computation*, 1999, pp. 3–13.

[15] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective Evolutionary Algorithms: A Aurvey of the State of the Art," *Swarm and Evolutionary Computation*, vol. 1, pp. 32–49, 2011.

[16] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II," in *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, 2000, pp. 1–11.

[17] Q. Zhang and H. Li, "Moea/d: A multi-objective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[18] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, 1998.

[19] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical Evaluation Methods for Multiobjective Reinforcement Learning Algorithms," *Machine Learning*, vol. 84, pp. 51–80, 2010.

[20] Z. Gabor, Z. Kalmar, and C. Szepesvari, "Multi-criteria reinforcement learning," in *The fifteenth international conference on machine learning*, 1998, pp. 197–205.

[21] S. Natarajan and P. Tadepalli, "Dynamic preferences in multi-criteria reinforcement learning," in *In Proceedings of International Conference of Machine Learning*, 2005, pp. 601–608.

[22] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proceedings of the international conference on machine learning*, 2008.

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.