Monte Carlo Tree Search with Metaheuristics

Jacek Mańdziuk[1][0000−0003−0947−028X] and Patryk Walczak[1]

Warsaw University of Technology, Warsaw, Poland,
mandziuk@mini.pw.edu.pl, patryk.walczak.stud@pw.edu.pl

Abstract. Monte Carlo Tree Search/Upper Confidence bounds applied to Trees (MCTS/UCT) is a popular and powerful search technique applicable to many domains, most frequently to searching game trees. Even though the algorithm has been widely researched, there is still room for its improvement, especially when combined with metaheuristics or machine learning methods. In this paper, we revise and experimentally evaluate the idea of enhancing MCTS/UCT with game-specific heuristics that guide the playout (simulation) phase. MCTS/UCT with the proposed guiding mechanism is tested on two popular board games: Othello and Hex. The enhanced method clearly defeats the well-known Alpha-beta pruning algorithm in both games, and for the more complex game (Othello) is highly competitive to the vanilla MCTS/UCT formulation.

Keywords: Monte Carlo Tree Search · Upper Confidence bounds applied to Trees · Two-player games · Metaheuristics.

1 Introduction

Since its introduction, the Monte Carlo Tree Search/Upper Confidence bounds applied to Trees (MCTS/UCT) algorithm [9] has been widely used in games and other combinatorial domains [14]. In recent years, it has been part of the superhuman Go playing program [12] that defeated the Go world champion. It has also been applied to Security Games [7, 8], chemical synthesis [11], or operation research [16], among others. In each of the above-mentioned cases, the baseline MCTS/UCT formulation has been significantly extended and tuned, usually by adding domain-knowledge.

Monte Carlo Tree Search. MCTS is an iterative algorithm that searches the state space and builds statistical evidence about the decisions available in particular states. In non-trivial problems, the state space (e.g. a game tree) is too large to be fully searched. Hence, in practical applications, MCTS is allotted some computational budget (a number of iterations) available for making a decision. Once this budget is fulfilled, MCTS returns the calculated best action/decision according to Eq. 1:

$$a^* = \arg\max_{a \in A(s)} Q(s, a) \tag{1}$$

where $A(s)$ is a set of actions available in state s, in which decision is to be made, and $Q(s, a)$ denotes the empirical average result of playing action a in state s. Naturally, the higher the number of iterations, the more confident the statistics, and the higher the chances that a finally recommended action is indeed the optimal one.

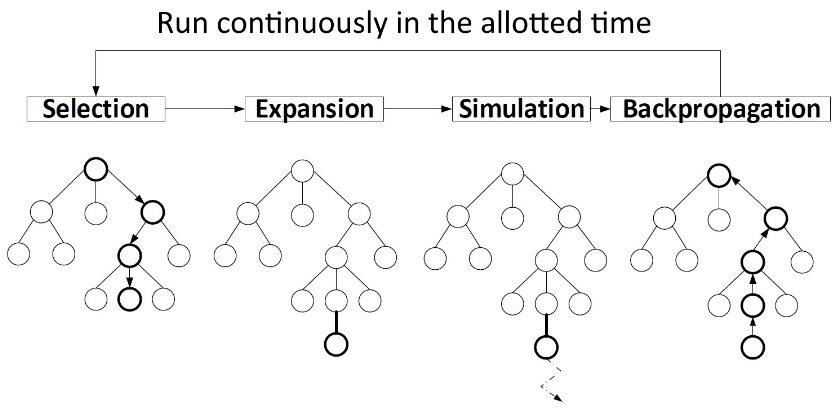


Fig. 1: Monte Carlo Tree Search phases.

Building a game tree starts with a single node (the root) representing the current position. Next, multiple iterations are performed, each of them consisting of four phases, depicted in Fig. 1.

1. **Selection** - In the first phase, the algorithm searches the portion of the tree that has already been represented in memory. Selection always starts from the root node and, at each level, selects the next node according to the *selection policy*. *Selection* terminates when a leaf node is reached.
2. **Expansion** - Unless the *selection* process reached a terminal state, the *expansion* phase takes place, which adds a new child node to the currently represented tree. The new node is a child of the node finally reached in the *selection* phase.
3. **Simulation** - In this phase, a **fully random** simulation of the game/problem is performed from the newly-added node, which reaches a terminal state of the game and calculates the players' scores/payoffs.
4. **Backpropagation** - This phase propagates back the payoff of the player represented in the root node along the path from the newly-added leaf node in the tree (the *expansion* node) to the root, and updates the nodes' statistics $(Q(s, a), N(s, a), N(a)$, described below).

Upper Confidence bounds applied to Trees. UCT [9] is typically used as the basic *selection policy* in MCTS. The algorithm attempts to balance the exploration (of yet not well tested actions) and the exploitation (of the best actions found so far). In a given node, UCT advises to first check each possible action once and then follow Eq. 2:

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,(N(s))}{N(s,a)}} \right\} \tag{2}$$

where $A(s)$ is a set of actions available in state s, $Q(s,a)$ denotes the average result of playing action a in state s in the simulations performed so far, $N(s)$ is the number of times state s has been visited in previous iterations, and $N(s,a)$ - the number of times action a has been sampled in state s. C is a game-dependent parameter that controls the balance between exploration and exploitation.

Due to the UCT formula, MCTS searches the game tree in an asymmetric manner, i.e. the promising lines of play are searched more thoroughly. Please consult [14] for a more detailed description of the MCTS/UCT algorithm.

Subsequent developments. In the years following the onset of MCTS/UCT, programs implementing this algorithm won recognized competitions in several popular games, e.g. General Game Playing (GGP) [13], Hex [1], or Go [4]. Over time, methods were also developed to modify the baseline formulation of the algorithm, e.g. RAVE [6] or MAST [3]. There have also been ideas related to combining the MCTS/UCT algorithm with neural networks [2], or its parallel implementation [14].

Contribution. In large search spaces (e.g. for games with high branching factor) MCTS/UCT may prove inefficient (assuming reasonable time limits) as it requires many simulations to assess the relevance of each move. For this reason, instead of using random simulations (*playouts*) one may consider using more focused, game-dependent, guided playouts that could lead to more meaningful information about the payoff structure than random playouts.

One particular approach is the *strategy switching mechanism* (SSM) [15] proposed in the context of GGP [13], which consists in utilizing a bunch of general or domain-specific heuristics that guide the MCTS/UCT *simulation* phase.

The main contribution of this paper is fourfold.

- We simplify the SSM method (and name it *strategy switching* (SS)), by excluding the possibility of making random moves, with certain probability, in the playout phase (see Section 2 for a detailed explanation).
- We thoroughly examine the efficacy of MCTS/UCT/SS approach in two popular board games of very different characteristics: Hex and Othello.
- Based on the conducted experiments, we analyse the strengths and weaknesses of MCTS/UCT/SS in particular setups.
- We compare MCTS/UCT/SS with the well-known Alpha-beta pruning algorithm, the vanilla MCTS/UCT, and several other enhancements to MCTS/UCT.

2 MCTS/UCT with *Strategy Switching*

The proposed extension of the MCTS/UCT algorithm utilizes a set of provided heuristics to modify the *simulation (playout)* phase of MCTS according to Algorithm 1.

Algorithm 1: *Simulation* phase of MCTS/UCT algorithm with SS

```
1 policy = strategy switching algorithm chooses the strategy;
2 while node is not a leaf do
3     node = a child of node chosen according to policy;
```

A critical component of SS is proper selection of a playout strategy (*policy*) in the current simulation. We follow a solution proposed in [15] in the context of GGP [13]. Namely, a selection is performed according to the UCB1 formula (Eq. 3), i.e. using the same exploration-exploitation balancing mechanism as the one used in MCTS/UCT in the *Selection* phase:

$$s^* = \arg\max_{s \in S} \left\{ Q(s,n) + b\sqrt{\frac{ln(n)}{T(s,n)}} \right\} \tag{3}$$

In Eq. 3, s^* is the strategy selected in the current simulation, n is the number of completed simulations, $Q(s,n)$ and $T(s,n)$ are the average score of strategy s in n simulations and the number of simulations that have used s in n performed simulations, respectively. b is the exploration-exploitation balancing factor.

For each strategy, a value in the parenthesis $\{\}$ is calculated based on the balance between the average score $Q(s,n)$ obtained in the previous simulations and the exploration factor (the part multiplied by b). A strategy with the maximum value is selected.

Please observe that in a broader perspective, SS implements a general idea of heterogeneous optimization, where particular components of the method (e.g. populations in a multi-population approach [17] or evolutionary operators in a multi-operator settings [18]) are dynamically selected from a certain pool, to maintain a proper exploration-exploitation balance in the optimization search.

3 Experiments

3.1 Tested games and heuristics

Hex. Hex [10] owes its name to the hexagonal tiles that form a rhombus-shaped board. Its rules are very simple. Players take turns placing their pieces on the empty fields of the board, trying to create a path connecting parallel sides of the board. The player who first manages to chain their edges of the board wins. Hex can have multiple board sizes. One can find small 7×7 boards, but also definitely larger ones with sides consisting of as many as 19 tiles. The most popular size is 11×11, and this is the one used in this work.

The following four general heuristics have been used in the experiments, to guide the move selection process in the playout phase: (1) random; (2) player's mobility (i.e. the number of possible moves/actions available to the player); (3) the difference between the player's and the opponent's mobilities; (4) simple game state evaluation function.

Othello. Othello (also known as Reversi) is a two-player game played on an 8×8 square board. The goal of the game is to possess on the board as many player's pieces as possible. Players take turns placing their pieces, and the game ends when one of the following conditions occurs: (a) all squares are filled with pieces; (b) none of the players can make a valid move; (c) one of the players lost all their pieces.

Initially, four pieces are placed on the board, two of each colour (black and white). The player with black pieces starts the game by placing their piece on a square that causes the "surrounding" of some number of white pieces. In effect, the surrounded pieces are captured, i.e. flipped to become black pieces. Next, the player with white pieces takes their turn and places their piece in a way that surrounds some number of black pieces. The surrounded pieces are captures and turned into white pieces. And so on.

Due to the above rule of changing the ownership of surrounded pieces, the difference between black and white pieces changes rapidly, which makes proper assessment of the game situation a non-trivial task.

In MCTS/UCT/SS, four general heuristics, the same as in Hex, have been tested: (1) random; (2) player's mobility; (3) the difference between the player's and the opponent's mobilities; (4) simple game state evaluation (a difference in the numbers of players' pieces). Furthermore, an additional Othello-specific heuristic, such as (5) weighted piece counter (WPC), has been considered. WPC assigns a certain weight w_i to the ith board square, $i = 1, \ldots, 64$, and computes a linear combination: $WPC = \sum_{i=1}^{64} w_i \cdot x_i$, where $x_i = 1, 0, -1$ if square i is occupied by the player's piece, is empty, is occupied by the opponent's piece, respectively.

3.2 Benchmark approaches

Alpha-beta pruning. The main method MCTS/UCT/SS has been compared with is the Alpha-beta algorithm, well-known in the game domain. In brief, the method improves the Minimax algorithm by means of adding two parameters: α and β, which store respectively the minimum and maximum value of the part of the tree that has already been checked. The values of α and β allow verifying the validity of further search. The method requires an evaluation function to work properly. In Hex it is the well-known Dijkstra shortest path algorithm, whereas in Othello it is a difference between the number of pieces of the player and the opponent.

Rapid Action Value Estimation (RAVE). RAVE algorithm [5, 6] optimizes the MCTS/UCT selection phase by offering an alternative way to calculate the node score that determines the selection of the child node. In short, in MCTS/UCT/RAVE, besides the $Q(s, a)$ value (cf. Eq. 2) an additional measure $Q_{RAVE}(s, a)$ is maintained for all state-action pairs (s, a). Suppose the current simulation is composed of actions $a_1 a_2 \dots a_n$ and states $s_1 s_2 \dots s_{n+1}$, i.e. action a_i is played in state $s_i, i = 1, \dots, n$. For each action a_i, if it is valid in any of the states $s_j, j = 1, \dots n+1, j \neq i$ (but was not played in that state in this simulation), the value $Q_{RAVE}(s_j, a_i)$ is updated based on the simulation score, in the same manner as $Q(s_i, a_i)$. The underlying assumption is the following: since action a_i has been played in state s_i and the simulation led to win/loss of a game, the impact of a_i is generally positive/negative also in other states of the simulated path. Using Q_{RAVE} statistics speeds up the MCTS/UCT state-action estimations. $Q(s, a)$ in Eq. 2 is redefined as follows:

$$Q(s,a) := \beta(s,a) \times Q_{RAVE}(s,a) + (1 - \beta(s,a)) \times Q(s,a); \;\; \beta(s,a) = \sqrt{\frac{k}{3N(s,a)+k}} \tag{4}$$

where k is a parameter that defines the number of simulations with action a played in state s at which $Q_{RAVE}(s, a)$ and $Q(s, a)$ are given equal weight, i.e. $\beta(s, a) = 1/2$.

3.3 Experimental setup

In the experiments, two versions of the SS algorithm have been tested. The first one had access to all heuristics considered for a given game (Hex and Othello, respectively). The other one was limited to the top 3 most effective heuristics devised in some number of preliminary tests, devoted to the methods' tuning. The former will be denoted as **MCTS/UCT/SS(all)**, the latter as **MCTS/UCT/SS(top3)**. The following selection of methods was used in comparative tests to assess both MCTS/UCT/SS versions:
1. **MCTS/UCT** – vanilla approach;
2. **MCTS/UCT/E** – the playout phase is guided by the evaluation function used in Alpha-beta – the bigger its value, the higher the assessment of a given node;
3. **MCTS/UCT/Mp** – the playout phase is guided by the player's mobility – the more actions are available, the higher the assessment of a given node;
4. **MCTS/UCT/Md** – the playout phase is guided by the difference between player's and opponent's mobilities – the bigger the difference, the higher the assessment of a given node;
5.**MCTS/UCT/RAVE** – in the selection phase, UCT is replaced by UCT/RAVE (cf. Eq. 4);
6.-7. **Alpha-beta(10)**, **Alpha-beta(12)** – Alpha-beta pruning algorithm with depth 10 and 12, respectively and simple evaluation function (see Section 3.2);
In the last three tested methods there is no tree search involved, and the move

is chosen directly based on:
8. **E** – the evaluation function used in Alpha-beta;
9. **R** – uniform distribution;
10. **WPC** – the WPC evaluation – only in Othello;

Tests were carried out with the following parameters: Hex game board size - 11×11, $k = 1000$ in RAVE (Eq. 4), $b = 4$ in the switching mechanism (Eq. 3), 10 000 simulations per move in MCTS/UCT algorithms. Parameter values were selected based on a limited number of preliminary tests and reflected a tradeoff between the quality of obtained results and computational time requirements. The methods have been compared based on a round-robin tournament in which every pair of methods was pitted 100 times, with different seeds and with sides swapped after each game.

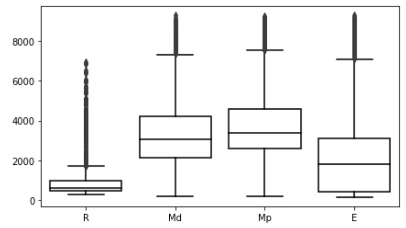
4 Results

Table 1: Hex results. The top 3 heuristics used in the MCTS/UCT/SS(top3) setting were player's mobility, the difference between the player's and the opponent's mobilities and simple game state evaluation. In rows 7 and 9 the evaluation function is Dijkstra's algorithm (the same as in the Alpha-beta algorithm).

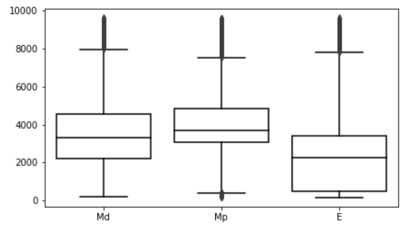
No	**Algorithm**	**Win**	**Loss**	**Avg. Time (h)**
1	MCTS/UCT	88,42%	11,58%	4,85
2	MCTS/UCT/SS(all)	72,73%	27,27%	17,52
3	MCTS/UCT/SS(top3)	64,72%	35,28%	21,95
4	MCTS/UCT/Mp	64,67%	35,33%	18,10
5	Alpha-beta(12)	64,61%	35,39%	6,64
6	Alpha-beta(10)	64,61%	35,39%	6,95
7	MCTS/UCT/E	52,42%	47,58%	12,89
8	MCTS/UCT/Md	52,17%	47,83%	31,66
9	E	37,08%	62,92%	6,69
10	R	21,50%	78,50%	7,50
11	MCTS/UCT/RAVE	12,00%	88,00%	9,89

Hex. Table 1 presents the results of by the algorithms in Hex. It can be seen that vanilla MCTS/UCT algorithm dominates not only in the win/loss scores, but also in the average computation time. Second and third on the podium is MCTS/UCT/SS algorithm, with all 4 and top 3 heuristics to choose from, respectively (without random choice in the latter case). Note, however, that the differences between positions 3 to 6 are practically negligible.

Most probably, the reason for vanilla MCTS/UCT leadership is the relative simplicity of Hex. In simpler games, standard MCTS/UCT proved very effective and the selection of additional heuristics to guide the playout simulations is, in some sense, a waste of time. Nevertheless, leaving aside MCTS/UCT, the MCTS/UCT/SS approach outperformed all the remaining competitors, including other MCTS/UCT implementations, as well as all non-MCTS/UCT approaches. A possible reason for the poor MCTS/UTC/RAVE performance is the aforementioned simplicity of Hex, which makes the more sophisticated versions of MCTS/UCT actually overcomplex.



(a) All 4 strategies are available



(b) Only top 3 strategies are available

Fig. 2: Statistics of the MCTS/UCT/SS strategy selection in Hex

Figure 2 shows statistics of strategy selection by the switching mechanism in case all strategies are available for selection (Fig. 2a) and when the algorithm chooses among the 3 most effective strategies only (Fig. 2b). The favourite strategy of SS having four policies to choose from was the player's mobility. In case MCTS/UCT/SS could only choose among the top 3 policies (excluding the random one), all quartiles, together with the mean, clearly show that the player's mobility strategy was again the most frequently used in the simulations, which is in line with the results presented in Table 1. This strategy was followed by the difference between player's and opponent's mobilities strategy, and then by the evaluation function based strategy.

Othello. The game of Othello is much more complex than Hex in terms of both the rules and tactical complexity. In this game MCTS/UCT/SS(top3) turned out to be the strongest approach, followed by MCTS/UCT/Md and MCTS/UCT/WPC (see Table 2). Both runner-up approaches utilized the well known Othello strategies, widely considered in the game literature. Due to very "unstable" pieces configurations in Othello, the strategies that refer to the numbers of pieces possessed by the players are generally weaker than the mobility-based strategies. In this respect, the second place of MCTS/UCT/Md is not surprising. Likewise, WPC in known to be the strong evaluation measure, as it considers critical aspects of the game, e.g. highly weights placing pieces on the corner squares, and penalizes placing them next to the corners. The fourth

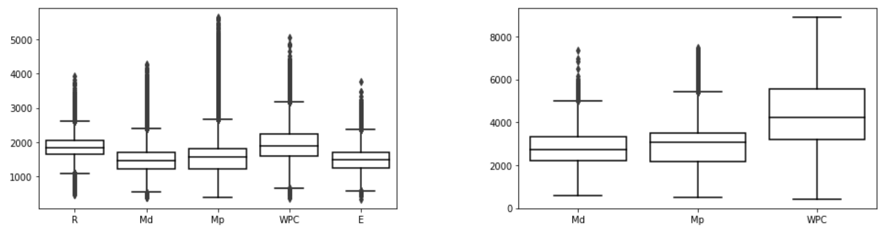
Table 2: Othello results. The top 3 heuristics used in the restricted MCTS/UCT/SS approach were player's mobility, the difference between the player's and the opponent's mobilities, WPC evaluation. In rows 2 and 10 the evaluation function is the difference between the numbers of the player's and opponent's pieces (the same as in the Alpha-beta algorithm).

No	**Algorithm**	**Win**	**Draw**	**Loss**	**Avg. Time (s)**
1	MCTS/UCT/SS(top3)	81,58%	1,75%	16,67%	1127,28
2	MCTS/UCT/E	79,67%	2,00%	18,33%	2061,46
3	MCTS/UCT/WPC	77,25%	2,25%	20,50%	504,98
4	MCTS/UCT/SS(all)	71,00%	1,17%	27,83%	1203,60
5	MCTS/UCT/Mp	69,67%	1,92%	28,42%	1316,78
6	MCTS/UCT	55,67%	1,92%	42,42%	524,69
7	MCTS/UCT/RAVE	42,33%	1,33%	56,33%	506,90
8	Alpha-beta(12)	19,67%	0,25%	80,08%	374,12
9	Alpha-beta(10)	19,67%	0,25%	80,08%	381,77
10	E	17,00%	0,25%	82,75%	390,81
11	R	9,58 %	1,08%	89,33%	421,08

and the fifth were MCTS/UCT/SS(all) and the MCTS/UCT with the strategy based on the player's mobility, respectively.

Apparently, in a more complex game like Othello, the SS idea proved useful and showed its superiority over all other methods. The MCTS/UCT algorithm with the switching mechanism was first and fourth in the overall ranking.

Detailed summaries of the average number of SS choices are shown in Fig. 3, respectively in case all strategies are available for selection (Fig. 3a) and when only top 3 strategies are considered (Fig. 3b). In both cases, the analysis of



(a) All 4 strategies are available (b) Only top 3 strategies are available

Fig. 3: Statistics of the MCTS/UCT/SS strategy selection in Othello

statistics of the switching mechanism, clearly points to the WPC heuristics as the most popular, hence the most advantageous among all. In the case of 5 possible choices, i.e. MCTS/UCT/SS(all), the advantage of WPC is not as striking as in the top 3 case.

5 Conclusions and Future Work

In this paper, we revise and experimentally evaluate the strategy switching mechanism combined with the popular MCTS/UCT search algorithm. The major findings from the presented research can be summarized as follows.

- In simple games, such as Hex, enhancing MCTS/UCT with SS does not seem to have much reason. Since MCTS/UCT is sufficiently strong method for this game, making the approach more complex, actually deteriorates the results.
- In more complex problems, e.g. the game of Othello, MCTS/UCT/SS shows its advantages, as the switching mechanism guides the MCTS playouts in a focused manner, hence saving time and resources.
- The choice of switching strategies is crucial for the SS efficacy. This issue was clearly observed in Hex which, despite being a game simple enough to be effectively approached with the vanilla MCTS/UCT, also did not seem to have the best choice of the supporting strategies. Apparently, heuristics based on mobility are not so strong and relevant in this game as in Othello, where along with WPC they constitute the core set of the effective strategies.
- The more effective of the two MCTS/UCT/SS setups is the one based on top 3 (generally top n) heuristics, as opposed to using all of them. This observation supports the intuition that selection among a smaller set of stronger strategies is generally more effective (easier) than the selection from a larger set, which also contains relatively weaker strategies.
- MCTS/UCT supported with a proper set of heuristic strategies outperforms the well-known Alpha-beta pruning algorithm, with the evaluation function relying on basic material assessment.
- MCTS/UCT with a lightweight evaluation function (either assigned directly or via the SS mechanism) is clearly a more powerful approach than a direct application of the same heuristics to move selection.

Our future research plans include verification of the MCTS/UCT/SS approach in more complex games, such as Hive and in other problem domains, e.g. project scheduling. We also plan to compare the SSM approach proposed in [15] with our simplified SS version and discuss the similarities and differences between these two implementations.

Acknowledgements. This research was carried out with the support of the Laboratory of Bioinformatics and Computational Genomics and the HPC Center of the Faculty of Mathematics and Information Science, Warsaw University of Technology under computational grant number A-22-03.

References

1. Arneson, B., Hayward, R., Henderson, P.: Mohex wins hex tournament. ICGA **32**, 114–116 (09 2013). https://doi.org/10.3233/ICG-2009-32218
2. Barratt, J., Pan, C.: Playing Go without Game Tree Search Using Convolutional Neural Networks. ArXiv **abs/1907.04658** (2019)
3. Björnsson, Y., Finnsson, H.: Cadiaplayer: A simulation-based general game player. Computational Intelligence and AI in Games, IEEE Transactions on **1**, 4 – 15 (04 2009). https://doi.org/10.1109/TCIAIG.2009.2018702
4. Enzenberger, M., Müller, M., Arneson, B., Segal, R.: Fuego—An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search. IEEE Transactions on Computational Intelligence and AI in Games **2**(4), 259–270 (2010). https://doi.org/10.1109/TCIAIG.2010.2083662
5. Gelly, S., Silver, D.: Combining Online and Offline Knowledge in UCT. In: Proceedings of the 24th International Conference on Machine Learning. p. 273–280 (2007), https://doi.org/10.1145/1273496.1273531
6. Gelly, S., Silver, D.: Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. Artif. Intell. **175**(11), 1856–1875 (jul 2011). https://doi.org/10.1016/j.artint.2011.03.007
7. Karwowski, J., Mańdziuk, J.: A new approach to security games. In: Artificial Intelligence and Soft Computing - 14th International Conference, ICAISC 2015, Zakopane, Poland, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9120, pp. 402–411. Springer (2015)
8. Karwowski, J., Mańdziuk, J.: A Monte Carlo Tree Search approach to finding efficient patrolling schemes on graphs. European Journal of Operational Research **277**(1), 255–268 (2019)
9. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Machine Learning: ECML 2006. pp. 282–293. Springer Berlin Heidelberg (2006)
10. Maarup, T.: Everything You Always Wanted to Know About HexBut Were Afraid to Ask. Ph.D. thesis (2005), https://maarup.net/thomas/hex/hex3.pdf
11. Segler, M.H., Preuss, M., Waller, M.P.: Planning chemical syntheses with deep neural networks and symbolic AI. Nature **555**(7698), 604–610 (2018)
12. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of Go without human knowledge. Nature **550**(7676), 354–359 (2017)
13. Świechowski, M., Park, H.S., Mańdziuk, J., Kim, K.J.: Recent Advances in General Game Playing. The Scientific World Journal **2015**, Article ID: 986262 (2015)
14. Świechowski, M., Godlewski, K., Sawicki, B., Mańdziuk, J.: Monte Carlo Tree Search: a review of recent modifications and applications. Artificial Intelligence Review **56**, 2497–2562 (2023). https://doi.org/10.1007/s10462-022-10228-y
15. Świechowski, M., Mańdziuk, J.: Self-adaptation of playing strategies in general game playing. IEEE Transactions on Computational Intelligence and AI in Games **6**(4), 367–381 (2014). https://doi.org/10.1109/TCIAIG.2013.2275163
16. Walędzik, K., Mańdziuk, J.: Applying hybrid Monte Carlo Tree Search methods to risk-aware project scheduling problem. Information Sciences **460-461**, 450–468 (2018)
17. Łapa, K., Cpałka, K., Kisiel-Dorohinicki, M., Paszkowski, J., Dębski, M., Le, V.H.: Multi-population-based algorithm with an exchange of training plans based on population evaluation. Journal of Artificial Intelligence and Soft Computing Research **12**(4), 239–253 (2022). https://doi.org/doi:10.2478/jaiscr-2022-0016

18. Łapa, K., Cpałka, K., Laskowski, Ł., Cader, A., Zeng, Z.: Evolutionary algorithm with a configurable search mechanism. Journal of Artificial Intelligence and Soft Computing Research **10**(3), 151–171 (2020). https://doi.org/doi:10.2478/jaiscr-2020-0011