Comparison of Rapid Action Value Estimation Variants for General Game Playing

Chiara F. Sironi and Mark H. M. Winands
Games & AI Group, Department of Data Science and Knowledge Engineering
Maastricht University, Maastricht, The Netherlands
Email: {c.sironi,m.winands}@maastrichtuniversity.nl

***Abstract*—General Game Playing (GGP) aims at creating computer programs able to play any arbitrary game at an expert level given only its rules. The lack of game-specific knowledge and the necessity of learning a strategy online have made Monte-Carlo Tree Search (MCTS) a suitable method to tackle the challenges of GGP. An efficient search-control mechanism can substantially increase the performance of MCTS. The RAVE strategy and its more recent variant, GRAVE, have been proposed for this reason. In this paper we further investigate the use of GRAVE for GGP and compare its performance with the more established RAVE strategy and with a new variant, called HRAVE, that uses more global information. Experiments show that for some games GRAVE and HRAVE perform better than RAVE, with GRAVE being the most promising one overall.**

I. Introduction

The aim of General Game Playing (GGP) is to develop agents that are able to play many arbitrary games at an expert level, only by being given their rules. As opposed to traditional game playing, GGP agents cannot rely on pre-coded game-specific knowledge because the game to be played is not known in advance. For the same reason, it is not possible to predetermine which search method is more suited for the game. The search method must be able to cope with a possibly infinite number of games. Moreover, the rules of the game are given to the agent only few seconds before the game starts, thus the agent has to learn the best playing strategy online. An extra challenge is posed by the fact that the agent usually has only few seconds per turn to choose a move.

A search technique that proved successful in GGP is Monte-Carlo Tree Search (MCTS) [1]–[3]. In its basic form MCTS is *aheuristic*, it does not require any game-specific knowledge, *anytime*, it can choose the move to be played within any time budget, and *selective*, it favours regions of the search tree that have the most promising moves, growing the tree asymmetrically [4]. Nowadays, all the best GGP agents are MCTS-based [5]–[9].

Other than GGP, MCTS has been successfully applied in many other domains. The most popular is the game of Go [1], for which MCTS represented a substantial step forward. Other examples are Hex [10], Havannah [11] and Lines of Action [12]. Moreover, the application of MCTS has not been limited to games, but also to other domains like combinatorial optimization problems, constraint satisfaction problems, scheduling problems, sample-based planning and procedural content generation [4].

Previous work [3], [13]–[19] has shown that good search-control mechanisms can consistently improve the overall performance of MCTS. Many enhancements have been proposed to improve different phases of the search. Some have been proposed for particular games as they rely on game-specific knowledge [3]. This makes them less interesting for GGP. Others, instead, are intrinsically domain-independent [13], [15]–[18] or are domain-independent modifications of game-specific methods [14], and are thus suitable for GGP.

Among domain-independent enhancements for the selection phase of MCTS we can find the Rapid Action Value Estimation technique (RAVE) [15], [20], [21]. RAVE has been successfully applied in different domains, like the game of Go [15], [20], and General Game Playing [21]. Recently, a generalization of RAVE has been proposed, the Generalized Rapid Action Value Estimation (GRAVE) [22]. This strategy has been shown to perform better than RAVE on some variants of Go and some other games. This and the fact that it does not necessarily need game-specific knowledge make GRAVE interesting to investigate further in the context of GGP.

The aim of this paper is to compare the performance and the robustness of GRAVE and RAVE for GGP. Moreover, we introduce another variant of GRAVE, called HRAVE, that uses the root history statistics. This enables to verify how performance is influenced by the use of information at different levels (from more local in RAVE to more global in HRAVE, with GRAVE being in between). In addition, we test how the performance of these RAVE variants is influenced by using a more informed play-out strategy instead of the one that chooses random moves. We do so by combining all the three strategies with MAST [13].

This paper is structured as follows. Section II gives and overview of MCTS and the MAST search-control mechanism. Section III describes the RAVE strategy and the variants that we are evaluating. Sections IV and V discuss the experimental setup and the obtained results, respectively. Finally, Section VI gives the conclusions and mentions possible future research.

II. Background

A. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a simulation-based search method that incrementally builds a tree representation of the search space for a game [1]–[3]. Each iteration of the algorithm performs a complete simulation of the game from the root state to a terminal state, adding nodes to the tree after each simulation and collecting information about the game in every node. More precisely, each iteration of the MCTS algorithm consists of the following four phases:

- *Selection:* the algorithm descends the tree built so far until it reaches a node that needs expansion. At each node it uses a *selection strategy* to determine which move to visit next. The standard MCTS selection strategy is the Upper Confidence bounds applied to Trees (UCT) [2]. Given a state s and the set $A(s)$ of all legal moves in s, it selects the most promising move a^* as follows:
$$a^* = \operatorname*{argmax}_{a \in A(s)} \left\{ Q(s,a) + C \times \sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \quad (1)$$
$N(s)$ is the number of times node s has been visited during the search, $N(s,a)$ is the number of times move a has been selected whenever node s was visited and $Q(s,a)$ is the average result obtained for all the simulations in which move a was played in state s. The second term of the formula is used to balance the exploitation of the estimated best move and the exploration of less visited moves. The constant C controls this balance.
- *Expansion:* this phase controls the expansion of the tree. An *expansion strategy* chooses which node(s) must be added to the tree and when. A common strategy is the one that expands the first encountered node that has at least one unexplored move. The node corresponding to the state reached by playing this move is added to the tree. If there is more than one unvisited move, one of them is chosen randomly. Other strategies might add more than one node at a time or prefer the selection of a promising visited move even if there are unvisited moves in the node. Other expansion strategies are discussed in [23].
- *Play-out:* starting from the last node added to the tree, the algorithm plays the game until a terminal state is reached. In each state the algorithm uses a *play-out* strategy to choose the move to play. One of the basic play-out strategies consists in selecting a move uniformly at random among the legal moves in the considered state.
- *Backpropagation:* after reaching a terminal state, the result of the simulation is propagated back through all the nodes traversed in the tree. The information memorized in the nodes depends on what the simulation and play-out strategies need. Usually, for UCT, each node s memorizes the values used in Formula (1).

After a certain number of iterations, the best move in the root node is chosen to be played in the real game. The meaning of *best move* depends on the implementation. It could be, for example, the one with the highest number of visits or the one with the highest average score. In this paper we consider the one with the highest average score.

B. The MAST play-out strategy in GGP

The Move Average Sampling Technique (MAST) [13], [16] is among the successful domain-independent search-control mechanisms proposed to guide the search during the play-out phase of MCTS. The main idea behind MAST is that a move that is good in one state is highly likely to be good also in other states. During the search this strategy memorizes for each move a a global average value $Q_{MAST}(a)$ based on the results of all the simulations in which move a was played. The original version of MAST was using $Q_{MAST}(a)$ in the Gibbs measure to compute a probability distribution over all the moves in a state and then select one of them according to this distribution. Later research [17], [18] has shown that an ϵ-greedy strategy that chooses the move with highest $Q_{MAST}(a)$ with probability $(1-\epsilon)$ and a random move with probability ϵ performs significantly better in most of the tested games.

A variant of MAST, called NST, has been proposed by Tak *et al.* [17]. The NST play-out strategy keeps also track of statistics of sequences of moves. This strategy has been shown to outperform MAST in most of the tested games.

A characteristic of both MAST and NST is that they keep track of the collected statistics throughout all the game. Further gain in performance has been achieved by decaying such statistics [24]. As the game progresses old statistics might not be as reliable as they were before, they might refer to moves that are strong in some parts of the game but weak in others. Thus, it is desirable to reduce their influence over time.

III. RAVE, GRAVE and HRAVE

A. RAVE

The RAVE selection strategy has been proposed in order to speed up the learning process inside the MCTS tree [16], [20], [21]. The UCT algorithm bases the selection of a move in a node on the estimated value obtained by sampling this move in the node multiple times. However, especially when the state space is large, the algorithm needs many simulations before it can sample all the moves in a node and more simulations before it can accumulate enough samples for the moves to reduce the variance of their estimated scores. To overcome this issue, RAVE keeps track of other statistics, also known as *All Moves As First (AMAF)* values [25], [26]. In every node it memorizes for all legal moves the following values:

- The average result $Q(s,a)$, obtained from all the simulations in which move a is performed in state s (the same value used in Formula (1)).
- The average result $AMAF(s,a)$, obtained from all the simulations in which move a is performed further down the path that passes by node s.

This means that, when backpropagating the result of a simulation in a certain node s of the tree, the value $Q(s,a)$ is updated for the move a that was directly played in the state, and the value $AMAF(s,a')$ is updated for all the legal moves a' in s that have been encountered at a later stage of the simulation. In this way RAVE can collect more samples and use them to reduce the variance of the moves values estimates for the nodes that do not have many visits. Using the AMAF scores enables to gather more information faster, however this information is more global than the local $Q(a,s)$ scores.

AMAF scores are useful for less visited nodes, but when the number of visits increases, the $Q(s,a)$ scores become more reliable and the influence of the AMAF scores should progressively decrease. This is why the RAVE algorithm keeps track of the two scores separately and uses a weight β to reduce the importance of the AMAF score over time.

Different variants for the RAVE move evaluation formula and for the β parameter computation have been proposed [11], [15], [20]. This paper uses the same formula that has been first used in GGP by CadiaPlayer [21]. RAVE selects a move according to (1), where the term $Q(s,a)$ is substituted by:

$$(1-\beta(s)) \times Q(s,a) + \beta(s) \times AMAF(s,a) \qquad (2)$$

and the term $\beta(s)$ is computed as follows:

$$\beta(s) = \sqrt{\frac{K}{3 \times N(s) + K}} \qquad (3)$$

where $N(s)$ is the number of times node s has been visited and K is the *equivalence parameter*, that indicates for how many simulations the two scores are weighted equal.

B. GRAVE

GRAVE [22] is a modification of RAVE that has been proposed to overcome one of its drawbacks. A problem of RAVE is that for the nodes close to the leaves of the tree not only the $Q(s,a)$ scores are based on a low number of samples, but also the AMAF scores. In these nodes the estimates of the moves values have less accuracy.

To solve this problem, for the nodes that have a number of visits lower than a given ref value GRAVE uses the AMAF scores of an ancestor node. Each node in the tree memorizes its own AMAF scores, but keeps also a reference to the closest ancestor that has a sufficient number of visits for its AMAF scores to be considered reliable. When a node s has sufficient visits ($N(s) > ref$), it starts using its own AMAF values instead of the ones of an ancestor, and the algorithm in that node starts behaving like RAVE. Note that, if $ref = 0$, GRAVE behaves exactly like RAVE from the beginning of the search. The GRAVE strategy enables to increase the accuracy of the estimates for the less visited nodes. However, the AMAF scores of an ancestor might be less relevant for its descendants, because these scores refer to a different game state.

Another aspect to be mentioned is the increased memory consumption of GRAVE with respect to RAVE. The latter needs only to store an extra statistic for each legal move in the node. With GRAVE, instead, the AMAF scores in a node might be used for other nodes lower in the tree that have a different set of legal moves. Therefore each node has to memorize the AMAF scores for all the moves that can be encountered at any lower level in the tree.

C. HRAVE

HRAVE is exactly the same as GRAVE, except that it always uses the AMAF scores of the current root of the tree (i.e. the ref parameter is set to infinity).

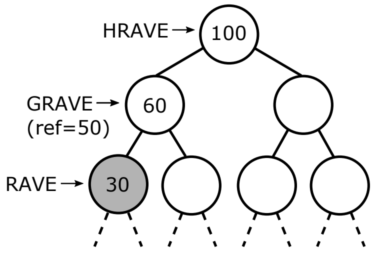

Fig. 1. Information used by RAVE, GRAVE and HRAVE for move selection in the highlighted tree node.

HRAVE shares similarities with the domain-independent selection strategy known as Progressive History [14]. This strategy adds to the UCT formula a bonus that depends on the *relative history* of the move being evaluated. This relative history is defined as the average result of all the simulations where the move was played. The influence of this bonus decreases over time as the number of visits of the node increases and the UCT estimate becomes more reliable.

In the case of HRAVE, the AMAF score of a move that is included in the UCT formula as shown in (2) can be compared to the Progressive History bonus. This is because both the AMAF score and the bonus are computed using the same statistics. In each turn of the game, the AMAF scores of the root of the tree correspond exactly to the history heuristic scores used by Progressive History. Moreover, like in Progressive History, the influence of the AMAF score decreases over time and makes the move evaluation formula converge to a pure UCT strategy.

A difference between HRAVE and Progressive History is that, for HRAVE, at the beginning of the search for a given turn, the root node already contains some statistics collected during previous turns. Progressive History, instead, starts each turn with an empty table. We decided to collect these statistics also during the previous turns to have a fair comparison of HRAVE with GRAVE and RAVE, because both of them, at every turn except the first, start the search already having some statistics in the AMAF tables of the nodes in the tree.

HRAVE can also be seen as the opposite of RAVE. While the latter uses the most local AMAF information, the former uses the most global one. GRAVE can be placed in between, it starts with more global AMAF statistics and then converges to the most local ones. Fig. 1 gives an example when these three heuristics are applied in MCTS. The number reported in each node is the number of node visits. For the selection of a move in the highlighted node, the figure shows in which node each algorithm looks for the AMAF statistics to use.

IV. Experimental setup

A. Games

The discussed algorithms have been tested on 15 different games: *3D Tic Tac Toe*, *Breakthrough*, *Knightthrough*, *Skirmish*, *Battle*, *Chinook*, *Chinese Checkers* with three players, *Checkers*, *Connect 5*, *Othello*, *Quad* (the version played on a 7×7 board), *Sheep and Wolf*, *Tic Tac Chess Checkers*

TABLE I
CHARACTERISTICS OF THE GAMES USED FOR THE EXPERIMENTS

Game	Players	Simult.	Constant-Sum
GROUP I			
3D Tic Tac Toe	2	No	Yes
Breakthrough	2	No	Yes
Knightthrough	2	No	Yes
Skirmish	2	No	No
Battle	2	Yes	No
Chinook	2	Yes	No
Chinese Checkers 3P	3	No	No
GROUP II			
Checkers	2	No	Yes
Connect 5	2	No	Yes
Othello	2	No	Yes
Quad	2	No	Yes
Sheep and Wolf	2	No	Yes
TTCC4 2P	2	No	No
Zhadu	2	No	Yes
TTCC4 3P	3	No	No

Four (*TTCC4*) with two and three players, and *Zhadu*. Table I gives an overview of the main characteristics of these games specifying the number of players, if they are sequential or simultaneous move games, and if they are constant or variable sum games. This set of games has been chosen because it is heterogeneous and because most of the games have been used in previous experiments that applied RAVE to GGP [16], [21].

In the following experiments, the games in Group I of Table I have also been used for tuning the *equivalence parameter* K for the algorithms. The games in Group II, instead, have only been used for comparing the strengths of RAVE, GRAVE and HRAVE. The GDL description of all the considered games can be found on the GGP-Base repository [27].

B. Setup

The aforementioned RAVE variants were implemented in the General Game Playing code base provided by the open-source GGP-Base project [7]. The code is implemented in Java and each agent tested in the experiments uses a reasoner based on Propositional Networks (PropNet, cfr. [28]).

In all the series of experiments, two agent types at a time are matched against each other. For each match, the PropNet of the game is generated in advance and both agents use the same so that none of them has any advantage for having a faster structure. Play clock and start clock are set to 1s, except for the experiments presented in Subsection V-B that are repeated also with start clock and play clock set to 10s.

For each game, if r is the number of roles in the game, there are 2^r different ways in which 2 types of agents can be assigned to the roles [29]. Two of the configurations involve only the same agent type assigned to all the roles, thus are not interesting and excluded from the experiments. Each configuration is run the same amount of times until the desired number of matches have been played.

For each of the performed experiments, we report as results the average winning percentage of one of the two involved agents with a 95% confidence interval. For each match the agent that achieved the highest score is considered the winner. When both agent types achieve the same score, the outcome of the match is considered a draw. In the first case, the winning player gets 1 point (full win) and the other player 0 points. In case of a draw both agent types get 0.5 points (half win).

As baseline to compare the different selection policies we have used an agent implementing the MCTS algorithm with UCT selection and random play-out strategy (P_{UCT}) and an agent implementing the MCTS algorithm with UCT selection and MAST play-out strategy ($P_{\text{UCT-MAST}}$). The UCT selection uses the formula given in (1), with $C = 0.7$. For the MAST strategy ϵ is set to 0.4, because it is the value that overall performed better in [17]. Moreover, the MAST statistics are decayed after playing every move with a factor $\gamma = 0.2$ (i.e. 20% of the statistics is kept for the next turn). This value is set lower than the one that was found to be the best in [24] because for each turn we have a higher number of simulations. This means that the number of collected statistics is higher and their influence needs to be decreased more strongly.

The aim of the first series of experiments is to tune the *equivalence parameter* K used to compute the weight $\beta(s)$ in (3). The tested values for K are 10, 50, 100, 250, 500, 750, 1000 and 2000 and the parameter is tuned using the games in Group I shown in Table I. The agents P_{RAVE}, P_{GRAVE} and P_{HRAVE} have been implemented and matched singularly against P_{UCT} for each value of K for at least 500 matches per game. As selection strategy they use the RAVE, GRAVE and HRAVE algorithm, respectively. They all use the random play-out strategy. All of them use the value 0.2 for the C constant because a lower value than the one used for the plain UCT algorithm empirically showed to achieve a better performance. For P_{GRAVE} the ref parameter is set to 50. For each of the three agents, the value of K that performed overall best in these series of experiments is also used in subsequent experiments.

In the second series of experiments, the agents P_{RAVE}, P_{GRAVE} and P_{HRAVE} with the best value of K are matched against P_{UCT} on all the games in Table I. Testing the agents on a wider set of games enables to detect a potential over-fitting of the K value to the games used for tuning. Moreover, it enables to check whether the tuned value works well also on other games. These experiments are performed with a start clock and play clock of 1s and then repeated with a start clock and play clock of 10s. This is to verify how an increased amount of time, and thus of simulations, influences the performance of the three RAVE variants. The minimum number of played matches per game is increased to 1000. This provides a more precise estimate of the average winning percentage of the agents, detecting with a higher confidence which of the algorithms performs best.

The aim of the third series of experiments is to verify the effect that the addition of the MAST play-out strategy has on the three variants of RAVE. For this series of experiments the random play-out strategy has been replaced with MAST to obtain the agents $P_{\text{RAVE-MAST}}$, $P_{\text{GRAVE-MAST}}$ and $P_{\text{HRAVE-MAST}}$. These agents have been matched only for the best value of K against $P_{\text{UCT-MAST}}$ on all the games in Table I with 1000 matches per game. Each of these agents has the same

TABLE II
WIN% OF P_{RAVE}, P_{GRAVE} AND P_{HRAVE} AGAINST P_{UCT} FOR DIFFERENT VALUES OF K FOR THE GAMES IN GROUP I

Game	$K = 10$	$K = 50$	$K = 100$	$K = 250$	$K = 500$	$K = 750$	$K = 1000$	$K = 2000$
P_{RAVE} vs P_{UCT}								
3D Tic Tac Toe	68.7(±4.06)	70.2(±4.00)	72.3(±3.91)	80.3(±3.47)	75.0(±3.78)	81.9(±3.36)	79.5(±3.53)	74.4(±3.81)
Breakthrough	58.2(±4.33)	60.6(±4.29)	63.8(±4.22)	65.8(±4.16)	72.2(±3.93)	72.0(±3.94)	71.6(±3.96)	65.8(±4.16)
Knightthrough	68.4(±4.08)	70.8(±3.99)	71.4(±3.96)	73.6(±3.87)	70.6(±4.00)	71.2(±3.97)	71.2(±3.97)	70.8(±3.99)
Skirmish	64.7(±4.16)	57.1(±4.28)	53.5(±4.34)	49.3(±4.35)	41.2(±4.26)	41.9(±4.27)	40.8(±4.27)	39.7(±4.23)
Battle	58.0(±3.83)	60.2(±3.78)	54.3(±3.86)	57.2(±3.81)	55.8(±3.88)	58.0(±3.85)	54.0(±3.94)	52.5(±4.00)
Chinook	45.2(±4.04)	51.5(±4.06)	55.2(±4.10)	59.2(±4.05)	57.2(±4.09)	59.3(±4.01)	55.9(±4.12)	52.4(±4.11)
Chinese Checkers 3P	63.9(±4.20)	61.1(±4.26)	63.9(±4.20)	58.7(±4.30)	64.3(±4.19)	64.9(±4.17)	59.6(±4.28)	58.5(±4.31)
Robustness	5	**6**	**6**	**6**	5	5	5	3
Avg Win%	61.0	61.6	62.1	63.4	62.3	**64.2**	61.8	59.2
P_{GRAVE} vs P_{UCT}								
3D Tic Tac Toe	63.5(±4.21)	71.4(±3.96)	75.0(±3.78)	75.3(±3.78)	80.1(±3.49)	80.7(±3.45)	79.9(±3.51)	77.9(±3.61)
Breakthrough	52.8(±4.38)	58.4(±4.32)	61.2(±4.28)	65.0(±4.19)	67.8(±4.10)	68.6(±4.07)	65.2(±4.18)	62.2(±4.25)
Knightthrough	72.6(±3.91)	72.0(±3.94)	74.0(±3.85)	71.2(±3.97)	70.6(±4.00)	74.4(±3.83)	68.0(±4.09)	68.6(±4.07)
Skirmish	62.2(±4.20)	57.0(±4.28)	51.2(±4.34)	55.7(±4.30)	46.1(±4.31)	44.6(±4.27)	42.0(±4.28)	42.2(±4.28)
Battle	68.7(±3.38)	72.7(±3.33)	71.7(±3.29)	69.6(±3.36)	71.6(±3.31)	72.6(±3.25)	69.6(±3.46)	67.5(±3.46)
Chinook	55.0(±4.08)	64.4(±4.00)	66.6(±3.89)	67.3(±3.80)	69.6(±3.80)	70.5(±3.70)	66.5(±3.87)	64.2(±3.94)
Chinese Checkers 3P	63.9(±4.20)	67.5(±4.09)	63.3(±4.21)	63.6(±4.20)	60.0(±4.28)	64.9(±4.17)	62.5(±4.23)	57.7(±4.32)
Robustness	6	**7**	6	**7**	6	5	5	5
Avg Win%	62.7	66.2	66.1	66.8	66.5	**68.0**	64.8	62.9
P_{HRAVE} vs P_{UCT}								
3D Tic Tac Toe	66.5(±4.12)	63.1(±4.22)	71.9(±3.93)	76.1(±3.74)	76.1(±3.71)	75.4(±3.75)	77.0(±3.67)	68.5(±4.04)
Breakthrough	53.6(±4.38)	57.0(±4.34)	62.6(±4.25)	65.2(±4.18)	65.4(±4.17)	60.4(±4.29)	63.2(±4.23)	59.2(±4.31)
Knightthrough	74.0(±3.85)	72.4(±3.92)	74.0(±3.85)	77.4(±3.67)	74.0(±3.85)	73.8(±3.86)	70.4(±4.01)	68.2(±4.09)
Skirmish	62.6(±4.21)	57.4(±4.31)	52.0(±4.33)	48.9(±4.33)	46.2(±4.32)	41.8(±4.26)	44.2(±4.30)	37.0(±4.18)
Battle	72.3(±3.21)	75.7(±3.25)	73.2(±3.17)	69.2(±3.37)	70.9(±3.34)	67.4(±3.44)	73.5(±3.26)	69.4(±3.48)
Chinook	54.9(±4.10)	66.0(±3.89)	66.4(±3.94)	74.9(±3.54)	72.9(±3.58)	75.4(±3.55)	73.4(±3.63)	73.8(±3.61)
Chinese Checkers 3P	67.9(±4.08)	64.1(±4.19)	66.3(±4.13)	65.5(±4.16)	62.7(±4.23)	60.3(±4.28)	61.3(±4.26)	60.2(±4.27)
Robustness	6	**7**	6	6	6	5	5	5
Avg Win%	64.5	65.1	66.6	**68.2**	66.9	64.9	66.1	62.3

settings of the corresponding version without MAST and for the MAST strategy the settings are the same as $P_{UCT\text{-}MAST}$.

As a validation of the results obtained in the previous series of experiments, the last series of experiments matches P_{RAVE}, P_{GRAVE} and P_{HRAVE} against each other two at a time and $P_{RAVE\text{-}MAST}$, $P_{GRAVE\text{-}MAST}$ and $P_{HRAVE\text{-}MAST}$ against each other two at a time. A total of at least 1000 matches per game have been played. All the experiments presented in the next sections were performed on a Linux server consisting of 64 AMD Opteron 6174 2.2-GHz cores.

V. EMPIRICAL EVALUATION

A. *Parameter tuning*

Table II shows the performance of P_{RAVE}, P_{GRAVE} and P_{HRAVE} against P_{UCT} for different values of K. For each agent, the value of K that achieves the highest robustness is selected to be used in subsequent experiments. We compute the robustness of a certain K for an agent by summing 1 point for each game in which the agent with such K achieved a statistically significant improvement over P_{UCT} and subtracting 1 point for each game in which it obtained a statistically significant worsening of the performance. In case more values of K have the same robustness, we chose the one with highest average win percentage over all the games.

For P_{RAVE} none of the values of K reaches the maximum robustness, however, for more than one value the agent achieves a statistically significant improvement in all games but one.

TABLE III
SIMULATIONS PER SECOND OF P_{UCT}, P_{RAVE}, P_{GRAVE} AND P_{HRAVE}

Game	P_{UCT}	P_{RAVE}	P_{GRAVE}	P_{HRAVE}
3D Tic Tac Toe	3093	2831	2920	2877
Breakthrough	1453	1378	1430	1435
Knightthrough	2285	2100	2146	2210
Skirmish	106	105	104	106
Battle	2149	2001	1898	1916
Chinook	2178	2085	2150	2144
Chinese Checkers 3P	4995	4108	4235	4229
Checkers	532	518	511	518
Connect 5	1191	1160	1144	1148
Othello	39	39	39	38
Quad	2767	2617	2627	2684
Sheep And Wolf	2110	2071	2063	2097
TTCC4 2P	1124	1277	1321	1368
Zhadu	494	484	477	480
TTCC4 3P	2058	2207	2220	2257

Among these values, $K = 250$ is chosen because it is the one with the highest average win percentage. For P_{GRAVE} the value $K = 250$ is selected because among the values with highest robustness is also the one with highest average win percentage. Finally, for P_{HRAVE} the value $K = 50$ is selected because it is the only one that reaches the highest robustness.

B. *Comparison of P_{RAVE}, P_{GRAVE} and P_{HRAVE} with P_{UCT}*

In this series of experiments, P_{RAVE}, P_{GRAVE} and P_{HRAVE} are matched against P_{UCT}, both with 1s and 10s play clock.

TABLE IV
WIN% OF P_{RAVE}, P_{GRAVE} AND P_{HRAVE} WITH BEST K AGAINST P_{UCT} WITH 1S PLAY CLOCK AND START CLOCK

Game	P_{RAVE}	P_{GRAVE}	P_{HRAVE}
3D Tic Tac Toe	**78.4**($\pm$2.54)	74.3($\pm$2.70)	64.0($\pm$2.97)
Breakthrough	66.6($\pm$2.92)	**67.6**($\pm$2.90)	57.8($\pm$3.06)
Knightthrough	73.0($\pm$2.75)	**73.6**($\pm$2.73)	71.3($\pm$2.81)
Skirmish	47.5($\pm$3.07)	54.5($\pm$3.05)	**59.3**($\pm$3.02)
Battle	57.0($\pm$2.69)	69.7($\pm$2.34)	**73.7**($\pm$2.29)
Chinook	59.6($\pm$2.84)	**68.3**($\pm$2.71)	65.1($\pm$2.74)
Chinese Checkers 3P	61.9($\pm$3.00)	63.2($\pm$2.98)	**64.4**($\pm$2.96)
Checkers	63.5($\pm$2.83)	**70.8**($\pm$2.65)	60.7($\pm$2.84)
Connect 5	70.8($\pm$2.76)	**75.5**($\pm$2.62)	66.8($\pm$2.89)
Othello	36.9($\pm$2.96)	42.9($\pm$2.99)	**57.4**($\pm$3.02)
Quad	**75.1**($\pm$2.67)	73.6($\pm$2.72)	73.3($\pm$2.73)
Sheep And Wolf	**66.0**($\pm$2.94)	62.9($\pm$3.00)	56.7($\pm$3.07)
TTCC4 2P	**72.9**($\pm$2.73)	71.2($\pm$2.77)	62.3($\pm$3.00)
Zhadu	69.3($\pm$2.86)	67.4($\pm$2.91)	**71.3**($\pm$2.80)
TTCC4 3P	52.1($\pm$3.03)	52.6($\pm$3.03)	**53.4**($\pm$3.05)
Robustness	11	12	**15**
Avg Win%	63.4	**65.9**	63.8

TABLE V
WIN% OF P_{RAVE}, P_{GRAVE} AND P_{HRAVE} WITH $K = 250$ AGAINST P_{UCT} WITH 10S PLAY CLOCK AND START CLOCK

Game	P_{RAVE}	P_{GRAVE}	P_{HRAVE}
3D Tic Tac Toe	**69.7**($\pm$2.81)	68.2($\pm$2.86)	60.7($\pm$2.99)
Breakthrough	**67.5**($\pm$2.90)	65.1($\pm$2.96)	60.4($\pm$3.03)
Knightthrough	**84.8**($\pm$2.23)	84.1($\pm$2.27)	84.4($\pm$2.25)
Skirmish	**60.2**($\pm$3.01)	60.0($\pm$3.00)	55.8($\pm$3.07)
Battle	**59.1**($\pm$2.19)	57.2($\pm$2.29)	54.6($\pm$2.35)
Chinook	39.8($\pm$2.78)	56.6($\pm$2.84)	**71.8**($\pm$2.52)
Chinese Checkers 3P	54.0($\pm$3.08)	**54.7**($\pm$3.07)	49.7($\pm$3.09)
Checkers	52.2($\pm$2.77)	56.3($\pm$2.73)	**61.8**($\pm$2.69)
Connect 5	**66.9**($\pm$2.36)	59.9($\pm$2.50)	53.3($\pm$2.47)
Othello	61.8($\pm$2.97)	**62.0**($\pm$2.97)	60.6($\pm$2.97)
Quad	**10.7**($\pm$1.87)	8.5($\pm$1.68)	7.9($\pm$1.64)
Sheep And Wolf	**69.6**($\pm$2.85)	69.0($\pm$2.87)	67.2($\pm$2.91)
TTCC4 2P	61.1($\pm$2.90)	**66.4**($\pm$2.80)	65.7($\pm$2.80)
Zhadu	63.4($\pm$2.94)	66.5($\pm$2.86)	**68.5**($\pm$2.82)
TTCC4 3P	54.4($\pm$2.97)	**58.5**($\pm$2.95)	50.3($\pm$3.02)
Robustness	10	**13**	11
Avg Win%	58.3	**59.5**	58.2

Table III reports for each game the average median number of simulations per second that each of the agents can perform.

Table IV shows the performance of P_{RAVE}, P_{GRAVE} and P_{HRAVE} with the best K against P_{UCT} with 1s play clock and start clock. P_{HRAVE} is the only one that achieves a significant improvement over P_{UCT} in all games, despite not being the one with the highest average win percentage. P_{RAVE} and P_{GRAVE} still obtain a significant improvement in most of the games, only in *Othello* they are significantly outperformed by P_{UCT}.

Table V shows the results obtained by repeating the experiment with 10s play clock and start clock. The results of P_{HRAVE} with $K = 50$ were noticeably lower (robustness = 7, average win percentage = 53.0) than the ones of P_{RAVE} and P_{GRAVE} with their best K. For this reason, the experiment for P_{HRAVE} was repeated with the value that produced the highest average win percentage in Table II, $K = 250$. Such results, being better than the ones of $K = 50$, are reported in Table V.

As can be seen, in most of the games the longer search time reduces the performance increase of P_{RAVE}, P_{GRAVE} and P_{HRAVE} against P_{UCT}. In *Quad* it even makes the use of RAVE, GRAVE and HRAVE detrimental, substantially reducing the win percentage around 10%. In *Knightthrough* and *Othello* instead, it seems that more search time increases the performance of all the three RAVE variants.

C. Comparison of $P_{\text{RAVE-MAST}}$, $P_{\text{GRAVE-MAST}}$ and $P_{\text{HRAVE-MAST}}$ with $P_{\text{UCT-MAST}}$

Table VI shows the performance of $P_{\text{RAVE-MAST}}$, $P_{\text{GRAVE-MAST}}$ and $P_{\text{HRAVE-MAST}}$ with the best K against $P_{\text{UCT-MAST}}$. For most of the games the addition of MAST as play-out strategy seems to benefit more $P_{\text{UCT-MAST}}$. $P_{\text{RAVE-MAST}}$, $P_{\text{GRAVE-MAST}}$ and $P_{\text{HRAVE-MAST}}$ perform significantly better than $P_{\text{UCT-MAST}}$ in most of the games. However, for many of these games the difference in performance achieved by $P_{\text{RAVE-MAST}}$, $P_{\text{GRAVE-MAST}}$ and $P_{\text{HRAVE-MAST}}$ against $P_{\text{UCT-MAST}}$ is not as high as the difference in performance achieved by P_{RAVE}, P_{GRAVE} and P_{HRAVE} against P_{UCT}. Some examples are the games *3D Tic Tac Toe*, *Connect 5* and *TTCC4* with 2 players.

The game for which MAST has the highest benefit on P_{UCT} is *Quad*. In this game P_{RAVE}, P_{GRAVE} and P_{HRAVE} were previously obtaining an improvement over P_{UCT}, while the corresponding agents with MAST are realizing a decrease in performance with respect to $P_{\text{UCT-MAST}}$.

Among the RAVE variants, the one that seems to benefit the most (in about half of the games) from the use of MAST is RAVE. This could be explained by considering that the AMAF scores used by RAVE in the nodes with a low number of visits only have a small number of samples. MAST can compensate the lack of local information near the leaf nodes of the tree. Using its global statistics, MAST steers the simulations towards more promising parts of the state space during the play-out improving its quality. The quality of a simulation for GRAVE and HRAVE, instead, is already improved near the leaf nodes by the use of the AMAF statistics of an ancestor.

TABLE VI
WIN% OF $P_{\text{RAVE-MAST}}$, $P_{\text{GRAVE-MAST}}$ AND $P_{\text{HRAVE-MAST}}$ WITH BEST K AGAINST $P_{\text{UCT-MAST}}$

Game	$P_{\text{RAVE-MAST}}$	$P_{\text{GRAVE-MAST}}$	$P_{\text{HRAVE-MAST}}$
3D Tic Tac Toe	64.9($\pm$2.76)	**65.3**($\pm$2.75)	57.3($\pm$2.89)
Breakthrough	**78.5**($\pm$2.55)	74.6($\pm$2.70)	72.3($\pm$2.78)
Knightthrough	**81.9**($\pm$2.39)	74.7($\pm$2.70)	75.6($\pm$2.66)
Skirmish	56.1($\pm$3.04)	53.6($\pm$3.04)	**64.9**($\pm$2.92)
Battle	72.5($\pm$2.32)	76.9($\pm$2.20)	**80.8**($\pm$2.03)
Chinook	32.2($\pm$2.60)	**61.3**($\pm$2.85)	58.3($\pm$2.91)
Chinese Checkers 3P	**58.7**($\pm$3.04)	57.5($\pm$3.05)	56.1($\pm$3.07)
Checkers	65.1($\pm$2.80)	**67.1**($\pm$2.74)	59.1($\pm$2.85)
Connect 5	**60.2**($\pm$2.25)	58.4($\pm$2.29)	46.6($\pm$2.41)
Othello	36.8($\pm$2.94)	42.6($\pm$3.00)	**50.1**($\pm$3.06)
Quad	**34.5**($\pm$2.80)	29.2($\pm$2.65)	29.8($\pm$2.67)
Sheep And Wolf	56.3($\pm$3.08)	56.6($\pm$3.07)	**57.3**($\pm$3.07)
TTCC4 2P	63.3($\pm$2.92)	**66.2**($\pm$2.85)	46.6($\pm$3.04)
Zhadu	**73.8**($\pm$2.73)	64.8($\pm$2.96)	65.1($\pm$2.96)
TTCC4 3P	**56.0**($\pm$2.98)	55.6($\pm$2.98)	55.9($\pm$3.01)
Robustness	9	**11**	8
Avg Win%	59.4	**60.3**	58.4

TABLE VII

WIN% OF ALL POSSIBLE COMBINATIONS OF AGENTS WITH AND WITHOUT MAST. THE WIN% ALWAYS REFERS TO THE FIRST OF THE TWO PLAYERS.

Game	P_{GRAVE} vs P_{RAVE}	$P_{GRAVE\text{-}MAST}$ vs $P_{RAVE\text{-}MAST}$	P_{HRAVE} vs P_{RAVE}	$P_{HRAVE\text{-}MAST}$ vs $P_{RAVE\text{-}MAST}$	P_{GRAVE} vs P_{HRAVE}	$P_{GRAVE\text{-}MAST}$ vs $P_{HRAVE\text{-}MAST}$
3D Tic Tac Toe	50.4(±3.09)	51.1(±2.93)	40.1(±3.01)	41.9(±2.89)	63.0(±2.97)	57.1(±2.91)
Breakthrough	46.9(±3.09)	46.1(±3.09)	38.8(±3.02)	44.6(±3.08)	57.1(±3.07)	54.6(±3.09)
Knightthrough	52.8(±3.10)	38.7(±3.02)	49.6(±3.10)	40.3(±3.04)	48.7(±3.10)	45.6(±3.09)
Skirmish	52.3(±3.04)	54.0(±3.04)	62.6(±2.97)	59.5(±3.02)	40.7(±3.02)	44.2(±3.01)
Battle	66.8(±2.34)	54.5(±2.65)	68.4(±2.38)	62.6(±2.52)	50.0(±2.46)	48.4(±2.67)
Chinook	58.3(±2.76)	70.1(±2.41)	55.4(±2.76)	70.8(±2.38)	54.0(±2.79)	49.7(±2.84)
Chinese Checkers 3P	55.1(±3.07)	50.1(±3.09)	55.3(±3.08)	49.4(±3.10)	46.2(±3.09)	50.1(±3.10)
Checkers	53.4(±2.91)	53.1(±2.91)	46.5(±2.94)	43.0(±2.91)	54.3(±2.90)	58.5(±2.89)
Connect 5	57.9(±2.97)	47.3(±2.19)	51.0(±3.04)	39.6(±2.17)	58.7(±2.99)	57.8(±2.22)
Othello	52.8(±3.04)	54.5(±3.04)	65.0(±2.91)	65.0(±2.90)	37.5(±2.95)	36.4(±2.93)
Quad	50.5(±3.09)	42.4(±2.90)	48.9(±3.07)	44.4(±2.89)	52.5(±3.06)	53.4(±2.93)
Sheep And Wolf	51.0(±3.10)	49.2(±3.10)	43.8(±3.08)	48.8(±3.10)	57.2(±3.07)	49.8(±3.10)
TTCC4 2P	52.3(±3.03)	54.2(±2.96)	43.7(±3.03)	37.7(±2.92)	60.9(±2.96)	66.0(±2.85)
Zhadu	50.1(±3.10)	42.0(±3.06)	52.3(±3.10)	40.2(±3.04)	46.7(±3.09)	49.5(±3.10)
TTCC4 3P	48.6(±3.02)	50.0(±2.98)	52.8(±3.04)	50.4(±3.01)	48.7(±3.03)	48.5(±3.01)
Robustness	4	1	0	−4	3	3
Avg Win%	53.3	50.5	51.6	49.2	51.7	51.3

Thus, the addition of MAST in the play-out has less added benefit to the overall simulation quality.

D. Matching RAVE variants against each other

As a validation of previous results the agents based on the three RAVE variants have been matched two at a time against each other. Table VII shows the obtained results. For each pair of algorithms the table reports a column of results without MAST and a column with MAST.

These results are in line to what has been observed in previous experiments. P_{GRAVE} performs better than P_{RAVE} in some games and equally in others. The performance of P_{RAVE} and P_{HRAVE} is more game-dependent. In some games they perform equally, in games like *3D Tic Tac Toe* and *Breakthrough* P_{RAVE} performs best and in games like *Skirmish* and *Battle* P_{HRAVE} performs best. A similar game-dependent performance can be observed for P_{GRAVE} and P_{HRAVE}, but in this case there are more games in which P_{GRAVE} performs best. When MAST is added to all the agents, the difference in their performance diminishes. $P_{GRAVE\text{-}MAST}$ and $P_{RAVE\text{-}MAST}$ perform similarly, one outperforming the other in a few games and vice-versa. MAST also benefits both $P_{RAVE\text{-}MAST}$ and $P_{GRAVE\text{-}MAST}$ against $P_{HRAVE\text{-}MAST}$.

Finally, we can compare the results obtained for *Knightthrough* by P_{GRAVE} against P_{RAVE} with the ones in [22]. It can be noticed that we did not achieve the same performance increase. In [22] the player based on GRAVE achieves a win rate of 67.8% against the one based on RAVE when both players have a limit of $1,000$ simulations per turn and a win rate of 67.2% when the limit is $10,000$ simulations per turn. However, this might be due to the different formula that we use for β and to the fact that we do not limit the number of simulations per turn but the amount of time. Moreover, our implementation of RAVE is achieving a higher win rate against UCT than in [22], where the win rate of RAVE is 69.4% for $1,000$ simulations per turn and 56.2% for $10,000$. This, therefore, reduces the potential gain by GRAVE.

TABLE VIII

AVERAGE NUMBER OF MOVE STATISTICS PER NODE OF P_{RAVE} AND P_{GRAVE}

Game	P_{RAVE}	P_{GRAVE}
3D Tic Tac Toe	4.58	9.11
Breakthrough	3.49	21.14
Knightthrough	3.16	13.44
Skirmish	4.02	54.38
Battle	8.40	19.92
Chinook	2.46	13.81
Chinese Checkers 3P	2.59	16.01
Checkers	2.58	41.94
Connect 5	4.47	9.69
Othello	2.41	14.87
Quad	3.66	7.57
Sheep and Wolf	2.95	32.04
TTCC4 2P	2.27	28.82
Zhadu	2.73	23.12
TTCC4 3P	2.47	13.32

E. Memory usage

As mentioned in Section III-B, GRAVE needs to memorize in each node the AMAF statistics for all the actions that are encountered during every simulation that passes through the node. The RAVE algorithm, instead, only needs to memorize in each node the AMAF statistics for the moves that are legal in the corresponding game state.

Table VIII shows for RAVE and GRAVE the average number of AMAF move statistics that are memorized in each node for every game. These results give an idea of the difference between the algorithms in memory usage. The space required by GRAVE ranges between 2 (in *3D Tic Tac Toe*) to 16 (in *Checkers*) times the space required by RAVE.

VI. CONCLUSION

In this paper the performance of the GRAVE strategy was compared to the one of the RAVE and the HRAVE strategies. GRAVE was also tested on a larger set of games than the one used in [22] to verify its applicability in the context of GGP.

When combined with a random play-out strategy, we may conclude that the performance of GRAVE is, in the worst case, comparable with the one of RAVE both when using 1s or 10s play clock. Not for all the tested games GRAVE was better than RAVE, but it never had an inferior performance, except in *Connect 5* when using a 10s play clock.

Regarding HRAVE, we may conclude that its performance is more game dependent when a random play-out strategy is used. In some games HRAVE is either better or comparable to RAVE and GRAVE, but there are some games where it performs worse. Moreover, when looking at the average win percentage, in none of the experiments its overall performance proved to be better than both RAVE and GRAVE.

When combined with the MAST play-out strategy, GRAVE still seems to be overall better than RAVE. However, it does not have the same advantage over RAVE that it has when both strategies are combined with the random play-out. MAST, apparently, compensates the lack of information near the leaf nodes for RAVE, closing the performance gap between RAVE and GRAVE. There are also a few games where the combination GRAVE-MAST actually performs worse than RAVE-MAST. Moreover, when using MAST, HRAVE is the strategy that appears to be the least beneficial among the three strategies.

As seen in the experiments, the difference in performance between RAVE, GRAVE and HRAVE is not large. Future research could investigate further the strengths of GRAVE over RAVE and HRAVE by tuning also its ref parameter. Moreover, the formula proposed more recently in [15] to compute the β parameter could be tested. According to their findings, with this formula the performance of the three RAVE variants could improve further. Moreover, in this paper we only tested the combination of these strategies with MAST. Other play-out policies might influence them in a different way. Testing the combination with the NST play-out strategy could be an idea for future research.

Acknowledgment

This work is funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project GoGeneral, grant number 612.001.121.

References

[1] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Computers and Games (CG 2006)*, ser. LNCS. Springer, 2007, vol. 4630, pp. 72–83.

[2] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Machine Learning: ECML 2006*, ser. LNCS. Springer, 2006, vol. 4212, pp. 282–293.

[3] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008.

[4] 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," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[5] Y. Björnsson and H. Finnsson, "CadiaPlayer: A simulation-based general game player," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 1, pp. 4–15, 2009.

[6] J. Méhat and T. Cazenave, "Ary, a General Game Playing program," in *Board Games Studies Colloquium*, 2010.

[7] S. Schreiber, "The General Game Playing base package," https://github.com/ggp-org/ggp-base.

[8] S. Darper and A. Rose, "Sancho GGP player," http://sanchoggp.blogspot.nl/2014/07/sancho-is-ggp-champion-2014.html.

[9] R. Emsile, "Galvanise," https://bitbucket.org/rxe/galvanise_v2.

[10] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo tree search in Hex," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 4, pp. 251–258, 2010.

[11] F. Teytaud and O. Teytaud, "Creating an upper-confidence-tree program for Havannah," in *Advances in Computer Games*, ser. LNCS. Springer, 2010, vol. 6048, pp. 65–74.

[12] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, "Monte Carlo tree search in Lines of Action," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 4, pp. 239–250, 2010.

[13] H. Finnsson and Y. Björnsson, "Simulation-based approach to General Game Playing." in *AAAI*, vol. 8, 2008, pp. 259–264.

[14] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for multi-player Monte-Carlo tree search," in *Computers and Games*, ser. LNCS, H. J. van den Herik, H. Ida, and A. Plaat, Eds. Springer, 2011, vol. 6515, pp. 238–249.

[15] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[16] H. Finnsson, "Simulation-based General Game Playing," Ph.D. dissertation, School of Computer Science, Reykjavk University, Reykjavk, Iceland, 2012.

[17] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-grams and the Last-Good-Reply policy applied in General Game Playing," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 2, pp. 73–83, 2012.

[18] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Bandits all the way down: UCB1 as a simulation policy in Monte Carlo tree search," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 81–88.

[19] E. J. Powley, P. I. Cowling, and D. Whitehouse, "Information capture and reuse strategies in Monte Carlo tree search, with applications to games of hidden information," *Artificial Intelligence*, vol. 217, pp. 92–116, 2014.

[20] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *Proceedings of the 24th International Conference on Machine Learning*. ACM, 2007, pp. 273–280.

[21] H. Finnsson and Y. Björnsson, "Learning simulation control in general game-playing agents." in *AAAI*, vol. 10, 2010, pp. 954–959.

[22] T. Cazenave, "Generalized rapid action value estimation," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 754–760.

[23] T. Yajima, T. Hashimoto, T. Matsui, J. Hashimoto, and K. Spoerer, "Node-expansion operators for the UCT algorithm," in *Computers and Games*, ser. LNCS, H. J. van den Herik, H. Ida, and A. Plaat, Eds. Springer, 2011, vol. 6515, pp. 116–123.

[24] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "Decaying simulation strategies," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 4, pp. 395–406, 2014.

[25] B. Brügmann, "Monte Carlo Go," Max Planck Institute of Physics, München, Germany, Tech. Rep., 1993.

[26] B. Bouzy and B. Helmstetter, "Monte-Carlo Go developments," in *Advances in Computer Games Many Games, Many Challenges*, ser. IFIP, vol. 263. Kluwer Academic, 2003, pp. 159–174.

[27] S. Schreiber, "Games - Base repository," http://games.ggp.org/base/, 2016.

[28] C. F. Sironi and M. H. M. Winands, "Optimizing propositional networks," in *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, 2016, pp. 7–14.

[29] N. Sturtevant, "A comparison of algorithms for multi-player games," in *Computers and Games (CG 2002)*, ser. LNCS. Springer Berlin Heidelberg, 2003, vol. 2883, pp. 108–122.