Analysis of Self-Adaptive Monte Carlo Tree Search in General Video 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
{c.sironi, m.winands}@maastrichtuniversity.nl

***Abstract*—A purpose of General Video Game Playing (GVGP) is to create agents capable of playing many different real-time video games. Instead of using a fixed general strategy, a challenging aspect is devising strategies that adapt the search to each video game being played. Recent work showed that on-line parameter tuning can be used to adapt Monte-Carlo Tree Search (MCTS) in real-time. This paper extends prior work on Self-adaptive Monte-Carlo Tree Search (SA-MCTS) by further testing one of the previously proposed on-line parameter tuning strategies, based on the N-Tuple Bandit Evolutionary Algorithm (NTBEA). Results show that, both for a simple and a more advanced MCTS agent, on-line parameter tuning has impact on performance only for a few GVGP games. Moreover, an informed strategy as NTBEA shows a significant performance increase only in one case. In a real-time domain as GVGP, advanced parameter tuning does not seem very promising. Randomizing pre-selected parameters for each simulation appears to be a robust approach.**

***Index Terms*—Monte-Carlo tree search, self-adaptive search, general video game playing, on-line parameter tuning**

I. Introduction

In GVGP [1] there is the need to devise general search approaches able to deal in real-time with many heterogeneous video games. Moreover, such approaches cannot exploit game-specific and prior knowledge. MCTS with its variations and enhancements [2] is one of the most commonly used techniques in GVGP. MCTS is often controlled by many parameters and the best parameter settings vary across games. A problem when using MCTS in GVGP is that parameters cannot be tuned in advance for the game at hand and have to be set to values that are generally good (i.e. perform overall well on a reference set of games, possibly heterogeneous).

Recently, methods to tune search parameters on-line have been investigated. Sironi and Winands [3] propose on-line parameter tuning for General Game Playing and show that on-line tuned agents almost reach the same performance of off-line tuned agents. Sironi *et al.* [4] apply on-line parameter tuning to obtain a Self-adaptive MCTS (SA-MCTS) strategy for GVGP and show that, when SA-MCTS is implemented in the SampleMCTS agent of the General Video Game AI framework (GVGAI) the win rate in a few games is increased.

This paper extends prior work on SA-MCTS. It considers one of the allocation strategies that performed best in [4], the one based on the N-Tuple Bandit Evolutionary Algorithm (NTBEA). This is compared with a less informed allocation strategy based on a Multi-Armed Bandit (MAB) and with a random allocation strategy. These allocation strategies are tested both on a simple and a more advanced MCTS agent. The effect of increasing the search budget is also investigated.

The paper is structured as follows. Section II describes the SA-MCTS approach and the tested allocation strategies. Section III discusses the results obtained by the experiments. Conclusion and future work are presented in Section IV.

II. Self-adaptive Monte-Carlo Tree Search

This section describes how parameters can be tuned on-line to obtain a self-adaptive behavior of the search. Subsection II-A describes how on-line parameter tuning is integrated with MCTS, while Subsection II-B describes three strategies that decide how to allocate the available samples to evaluate different parameter value combinations.

A. Integration of On-line Parameter Tuning with MCTS

Figure 1 gives an overview of SA-MCTS. The central box shows the four phases of standard MCTS: *selection*, *expansion*, *play-out* and *backpropagation*. To tune parameters on-line two more phases are added to the search: (i) an initial phase where an allocation strategy chooses which combination of parameter values will control the next simulation, and (ii) a final phase where the reward obtained by the simulation is used to update statistics about the chosen combination of parameters.

B. Allocation Strategies

An allocation strategy decides how to allocate the available samples to the feasible parameter values that need to be evaluated. This section describes the three considered allocation strategies: *Random*, *Multi-Armed Bandit* (MAB) [5], *and N-Tuple Bandit Evolutionary Algorithm* (NTBEA) [6].

1) Random: before each MCTS simulation this allocation strategy selects a parameter combination uniformly at random from a set of manually pre-selected feasible values. This means that it does not need to collect any statistics about the performance of the combinations. This study includes a random strategy to verify how the other more informed allocation strategies would compare to one that uses no information collected on-line to select parameter values. Note that by pre-selecting the set of values we are still giving information to the strategy, and its performance will depend on how good these values are.

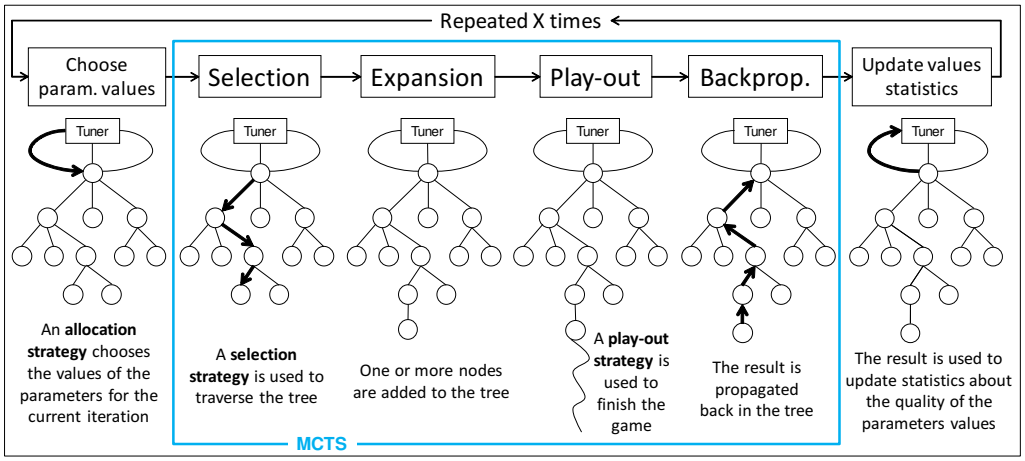

Fig. 1. Interleaving on-line tuning with MCTS

2) MAB: this allocation strategy considers the problem as a Multi-Armed Bandit [5], where each arm corresponds to one of the combinations of parameter values. Before each simulation a combination of parameter values $\vec{p}^* = \langle p_1, ..., p_d \rangle$ is selected with UCB1 as follows:

$$\vec{p}^* = \underset{\vec{p} \in \mathcal{P}}{\operatorname{argmax}} \left\{ \overline{Q}(\vec{p}) + C_{\text{MAB}} \times \sqrt{\frac{\ln N}{N_{\vec{p}}}} \right\} .$$

Here, $\mathcal{P}$ is the set of all combinations of parameters, $\overline{Q}(\vec{p})$ is the average reward obtained by all simulations controlled by combination $\vec{p}$ (normalized in $[0, 1]$), C_{MAB} is the *exploration constant*, N is the total number of simulations performed so far, and $N_{\vec{p}}$ is the number of simulations performed so far that were controlled by the parameter combination $\vec{p}$. When selecting the combination that maximizes the UCB1 value, unexplored combinations are assigned a predefined value fpu (i.e. *first play urgency*). For the experiments presented in Section III C_{MAB} is set to 0.7 and fpu is set to 1.0.

A limitation of this strategy is that it ignores the combinatorial structure of the search space. When choosing parameter values, it does not consider that good values in a combination might be good in general or in other combinations.

3) NTBEA: this allocation strategy is based on the algorithm proposed by Lucas *et al.* [6]. Two main components of NTBEA can be distinguished: an *evolutionary algorithm*, and an *N-Tuple fitness landscape model* (LModel). The evolutionary algorithm considers each combination of parameters as an individual and each single parameter as a gene. It starts with a randomly generated parameter combination and evolves it over time using statistics collected in LModel (e.g. average reward and number of visits of tuples of parameters) to decide which combination should be evaluated next. More precisely, the evolutionary algorithm repeats the following steps:

1) Use the current combination $\vec{p}$ to control an MCTS simulation.
2) Use the reward obtained by the MCTS simulation to update the statistics for parameter tuples in *LModel*.
3) Generate x neighbors of $\vec{p}$, each by mutating the value of a randomly selected parameter in $\vec{p}$.
4) Evaluate each of the x neighbors using *LModel* to compute an estimate of their UCB1 value.
5) Set the neighbor with the highest estimated UCB1 value as the current combination.

More details on how to use LModel to compute the estimate of the UCB1 values can be found in [6]. For the experiments presented in Section III x is set to 5 and the exploration constant C_{NTBEA} used to compute UCB1 values is set to 0.7.

III. Experiments

This section presents an analysis of SA-MCTS under different conditions. Subsection III-A describes the experimental setup and Subsection III-B reports and analyses the results.

A. Setup

SA-MCTS is evaluated in the single-player track of the GVGAI framework [1]. The allocation strategies tune feasible preselected parameters on-line for the following MCTS agents:

- SampleMCTS: the MCTS agent provided in the framework. This agent implements a simple version of MCTS that uses UCB1 as selection strategy and a random play-out strategy. For this agent we tune the UCB1 exploration constant C with values in $\{0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0\}$, and the maximum search depth D with values in $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. When this agent is not tuned on-line it uses the following fixed parameter setting: $C = 1.4$, $D = 10$.
- MaastCTS2: the winner of the 2016 GVGAI Single-Player Planing Championship [7]. It implements MCTS with UCB1 selection strategy and a series of enhancements, among which the use of *Progressive History* in the selection strategy and the *N-Gram Selection Technique* (NST) as play-out strategy. For this agent we tune the UCB1 exploration constant C with values in $\{0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0\}$,

the *Progressive History* weight W with values in $\{0.1, 0.25, 0.5, 1, 3, 5, 7.5, 10, 20, 50\}$, the NST probability of playing a random action ϵ with values in $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, the minimum number of visits for an N-gram to be considered in the computation K with values in $\{1, 3, 5, 7, 10, 15, 20, 30, 50\}$, and the maximum used N-gram length N with values in $\{1, 2, 3, 4, 5\}$. When this agent is not tuned on-line it uses the following fixed parameter setting: $C = 0.6$, $W = 1$, $\epsilon = 0.5$, $K = 7$, $N = 3$.

All agents do not keep any knowledge between game runs, so both the game tree built by MCTS and the parameter statistics collected by the allocation strategies are reset.

Four series of experiments were performed. The first series of experiments tests the SAMPLEMCTS agent with fixed parameter values, and three SAMPLEMCTS agents with values tuned on-line by each of the allocation strategies. The game tick is set to $40ms$ as in the GVGAI competition, and the agents are tested on 20 single-player games. The second series of experiments tests the MAASTCTS2 agent with fixed parameter values, and three MAASTCTS2 agents with values tuned on-line by each of the allocation strategies. The game tick is set to $40ms$ and the agents are tested on the same 20 single-player games. The third series of experiments is the same as the second, but the game tick is set to $100ms$ and only 10 of the initial 20 games are used. Games for which MAASTCTS2 is performing close to 100% are excluded. Only games for which there is more room for improvement when given more search time are kept. The fourth series of experiments tests the performance of SAMPLEMCTS and MAASTCTS2 when their parameters are set to fixed values expected to be sub-optimal for the games. These values are $C = 0$, $D = 1$ for SAMPLEMCTS, and $C = 0$, $W = 0$, $\epsilon = 0$, $K = \infty$, $N = 1$ for MAASTCTS2.

Results presented below always report the win percentage of the agent with 95% confidence interval. The win percentage is computed by playing all 5 levels of each game for 100 times, obtaining a total of 500 samples per game per agent.

B. Results

Tables I, II, III and IV show results obtained with the first, second, third and fourth series of experiments, respectively.

What is more interesting to observe for the first two series of experiments is that the random strategy seems to achieve in most of the games the same performance as the agent with fixed parameters. In addition, for almost all the games neither the MAB nor the NTBEA allocation strategy seem to perform better than the random strategy.

A combination of three factors might explain these results. Firstly, the manually constructed sets of parameter values might be mostly reasonable for all the games and might not contain particularly bad values. This means that for games for which the fixed parameter settings are sub-optimal, randomization will instead be able to control most of the search with optimal values. This is probably what happens in Table I for *Chase* and *Crossfire*, for which SAMPLEMCTS with the random allocation strategy seems to have a better performance than SAMPLEMCTS with fixed parameters. Secondly, the number of simulations that the agents can perform might be too small (usually a few dozen of simulations per tick, with a $40ms$ tick duration). A small number of simulation might have two implications: (i) the allocation strategies cannot find optimal values early enough in the game to make a difference in the performance, and (ii) even if there are sub-optimal values among the feasible ones, the number of simulations controlled by them is not high enough to be detrimental. Thirdly, the continuous change of parameter values selected by the allocation strategies, especially by the random one, might cause more diversity in the simulations. A more diversified search might actually be beneficial to tackle GVGP games.

From the third series of experiments (Table III) we can see that a longer search time significantly increases the performance of MAASTCTS2 in many of the games. With more search time there are a few games (*Bait*, *Chase* and *Survive Zombies*) for which the performance of the agent tuned with the random strategy seems to decrease. Also the overall performance of the agent tuned with MAB seems inferior to the other agents, as we would expect. The MAB allocation strategy

TABLE I
WIN PERCENTAGE OF SAMPLEMCTS WITH FIXED PARAMETERS AND SAMPLEMCTS WITH PARAMETERS TUNED ON-LINE BY EACH OF THE ALLOCATION STRATEGIES. GAME TICK IS SET TO *40ms*

Game	SAMPLEMCTS			
	Fixed parameters	Random	MAB	NTBEA
Aliens	**100.0**(±0.00)	**100.0**(±0.00)	99.6(±0.55)	**100.0**(±0.00)
Bait	6.6(±2.18)	**8.0**(±2.38)	7.6(±2.33)	6.8(±2.21)
Butterflies	95.2(±1.88)	94.4(±2.02)	95.2(±1.88)	**95.6**(±1.80)
CamelRace	4.2(±1.76)	5.6(±2.02)	**5.8**(±2.05)	3.8(±1.68)
Chase	3.2(±1.54)	**6.6**(±2.18)	**6.6**(±2.18)	6.2(±2.12)
Chopper	**91.4**(±2.46)	86.2(±3.03)	86.0(±3.04)	89.2(±2.72)
Crossfire	4.2(±1.76)	8.8(±2.49)	8.6(±2.46)	**14.2**(±3.06)
DigDug	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
Escape	0.2(±0.39)	0.8(±0.78)	**4.2**(±1.76)	0.2(±0.39)
HungryBirds	5.4(±1.98)	4.6(±1.84)	**5.6**(±2.02)	4.4(±1.80)
Infection	97.0(±1.50)	97.6(±1.34)	**98.0**(±1.23)	97.8(±1.29)
Intersection	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
MissileCommand	60.4(±4.29)	**63.0**(±4.24)	61.6(±4.27)	60.0(±4.30)
Modality	27.0(±3.90)	27.0(±3.90)	27.4(±3.91)	**28.2**(±3.95)
PlaqueAttack	91.8(±2.41)	**93.4**(±2.18)	87.6(±2.89)	92.8(±2.27)
Roguelike	0.0(±0.00)	0.0(±0.00)	**0.2**(±0.39)	0.0(±0.00)
SeaQuest	**55.0**(±4.37)	50.4(±4.39)	49.6(±4.39)	51.6(±4.38)
SurviveZombies	41.0(±4.32)	**42.6**(±4.34)	37.6(±4.25)	42.2(±4.33)
WaitForBreakfast	15.4(±3.17)	**17.6**(±3.34)	15.8(±3.20)	13.2(±2.97)
Avg Win%	39.9(±0.96)	**40.3**(±0.96)	39.9(±0.96)	**40.3**(±0.96)

TABLE II
WIN PERCENTAGE OF MAASTCTS2 WITH FIXED PARAMETERS AND MAASTCTS2 WITH PARAMETERS TUNED ON-LINE BY EACH OF THE ALLOCATION STRATEGIES. GAME TICK IS SET TO *40ms*

Game	MAASTCTS2			
	Fixed parameters	Random	MAB	NTBEA
Aliens	**100.0**(±0.00)	**100.0**(±0.00)	99.6(±0.55)	**100.0**(±0.00)
Bait	**31.8**(±4.09)	30.4(±4.04)	22.0(±3.63)	31.6(±4.08)
Butterflies	98.6(±1.03)	99.2(±0.78)	99.4(±0.68)	**100.0**(±0.00)
CamelRace	**44.4**(±4.36)	41.0(±4.32)	39.2(±4.28)	42.4(±4.34)
Chase	28.0(±3.94)	**30.4**(±4.04)	20.2(±3.52)	26.8(±3.89)
Chopper	**99.8**(±0.39)	**99.8**(±0.39)	99.6(±0.55)	99.6(±0.55)
Crossfire	**31.8**(±4.09)	27.4(±3.91)	17.2(±3.31)	28.4(±3.96)
DigDug	1.6(±1.10)	1.2(±0.96)	**1.8**(±1.17)	1.2(±0.96)
Escape	93.4(±2.18)	**94.6**(±1.98)	80.4(±3.48)	92.2(±2.35)
HungryBirds	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)
Infection	**100.0**(±0.00)	99.8(±0.39)	99.8(±0.39)	**100.0**(±0.00)
Intersection	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
MissileCommand	**96.8**(±1.54)	96.6(±1.59)	92.6(±2.30)	94.6(±1.98)
Modality	25.6(±3.83)	24.6(±3.78)	32.0(±4.09)	**41.0**(±4.32)
PlaqueAttack	94.8(±1.95)	95.0(±1.91)	94.2(±2.05)	**95.8**(±1.76)
Roguelike	4.6(±1.84)	**4.8**(±1.88)	0.8(±0.78)	3.2(±1.54)
SeaQuest	**58.4**(±4.32)	53.6(±4.38)	53.8(±4.37)	54.6(±4.37)
SurviveZombies	42.4(±4.34)	**42.8**(±4.34)	38.2(±4.26)	40.8(±4.31)
WaitForBreakfast	**99.0**(±0.87)	98.4(±1.10)	98.0(±1.23)	98.0(±1.23)
Avg Win%	**62.6**(±0.95)	62.0(±0.95)	59.4(±0.96)	62.5(±0.95)

TABLE III
WIN PERCENTAGE OF MAASTCTS2 WITH FIXED PARAMETERS AND MAASTCTS2 WITH PARAMETERS TUNED ON-LINE BY EACH OF THE ALLOCATION STRATEGIES. GAME TICK IS SET TO *100ms*

Game	MAASTCTS2			
	Fixed parameters	Random	MAB	NTBEA
Bait	**51.8**(±4.38)	40.4(±4.31)	31.2(±4.07)	36.6(±4.23)
CamelRace	**95.8**(±1.76)	92.2(±2.35)	85.4(±3.10)	90.8(±2.54)
Chase	**56.2**(±4.35)	50.4(±4.39)	43.0(±4.34)	51.6(±4.38)
Crossfire	**84.8**(±3.15)	83.2(±3.28)	68.2(±4.09)	81.8(±3.39)
DigDug	0.0(±0.00)	0.2(±0.39)	**0.4**(±0.55)	0.0(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
Modality	26.2(±3.86)	25.6(±3.83)	38.8(±4.28)	**40.4**(±4.31)
Roguelike	**32.6**(±4.11)	30.6(±4.04)	28.4(±3.96)	32.0(±4.09)
SeaQuest	58.6(±4.32)	56.0(±4.36)	**61.8**(±4.26)	56.2(±4.35)
SurviveZombies	**49.0**(±4.39)	46.0(±4.37)	44.8(±4.36)	45.4(±4.37)
Avg Win%	**45.5**(±1.38)	42.5(±1.37)	40.2(±1.36)	43.5(±1.37)

TABLE IV
WIN PERCENTAGE OF SAMPLEMCTS AND MAASTCTS2 WITH SUB-OPTIMAL FIXED PARAMETER VALUES. GAME TICK IS SET TO *40ms*

Game	SAMPLEMCTS$_{SUB}$	MAASTCTS2$_{SUB}$
Aliens	67.0(±4.13)	100.0(±0.00)
Bait	6.0(±2.08)	23.2(±3.70)
Butterflies	66.8(±4.13)	98.8(±0.96)
CamelRace	3.0(±1.50)	32.0(±4.09)
Chase	3.6(±1.63)	29.0(±3.98)
Chopper	0.0(±0.00)	98.4(±1.10)
Crossfire	10.2(±2.66)	20.2(±3.52)
DigDug	0.0(±0.00)	1.0(±0.87)
Escape	29.6(±4.01)	92.4(±2.33)
HungryBirds	1.8(±1.17)	99.4(±0.68)
Infection	95.0(±1.91)	100.0(±0.00)
Intersection	100.0(±0.00)	100.0(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)
MissileCommand	32.4(±4.11)	94.2(±2.05)
Modality	17.0(±3.30)	47.0(±4.38)
PlaqueAttack	24.6(±3.78)	88.0(±2.85)
Roguelike	0.0(±0.00)	1.6(±1.10)
SeaQuest	25.0(±3.80)	58.8(±4.32)
SurviveZombies	27.4(±3.91)	36.6(±4.23)
WaitForBreakfast	58.2(±4.33)	84.8(±3.15)
Avg Win%	28.4(±0.88)	60.3(±0.96)

suffers from the high overhead of computing the UCB1 value for all parameter combinations before each simulation, and in addition does not exploit information about the combinatorial structure of the parameter space. However, even if we increase the search time to $100ms$, the results are still in line with what is observed for $40ms$. The random allocation strategy is still quite robust, and MAB and NTBEA do not seem much better.

Over all the first three series of experiments, it is still interesting to notice that there are a few games for which on-line parameter tuning seems beneficial. For the SAMPLEMCTS agent, on-line tuning with NTBEA significantly increases the performance in *Crossfire*, and on-line tuning with MAB significantly increases the performance in *Escape* (Table I). Moreover, for the MAASTCTS2 agent the performance in *Modality* is significantly increased by NTBEA when using $40ms$ per tick (Table II), and by both NTBEA and MAB when using $100ms$ per tick III. This suggest that the allocation strategies that make informed decisions have the potential to be useful even when decisions have to be made fast.

The fourth series of experiments (Table IV) seems to confirm that on-line parameter adaptation (whether randomly or with an informed strategy) might still have some benefits. We can see that the sub-optimal values for these experiments cause a decrease in performance for many of the games if compared with the fixed values used in previous experiments. Nevertheless, there are games for which the performance is higher (if not the highest over all experiments), like *Crossfire*, *Escape* and *Wait For Breakfast* for SAMPLEMCTS, and *Modality* for MAASTCTS2. This suggests that for those games the fixed values used in the first three series of experiments were non-optimal. Moreover, some good values for those games might have been left out of the pre-selected sets of values.

IV. CONCLUSION AND FUTURE WORK

This paper extended the analysis of SA-MCTS in GVGP by testing on-line parameter tuning both on a simple and a more advanced MCTS agent. The effect of increasing the search time has also been evaluated. In addition, the performance of NTBEA, one of the allocation strategies that performed best in previous work on on-line parameter tuning, has been compared with a less informed allocation strategy (MAB) and a completely random allocation strategy.

Given the obtained results, we may conclude that the use of on-line parameter tuning might be more suitable for domains where a higher number of simulations can be reached, or for domains that are more sensitive to changes in the search parameter values. Moreover, we may conclude that in a real-time context like GVGP, randomization of parameter values usually gives a robust setting for a small number of parameters, especially if we pre-select small sets of feasible values.

For future work it would be interesting to see if increasing time constraints to achieve a few thousands simulations per tick would make a difference. It would also be interesting to investigate other ways of on-line self-adaptation of the search that do not necessarily involve changing search parameters.

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] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[2] 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.

[3] C. F. Sironi and M. H. M. Winands, "On-line parameter tuning for Monte-Carlo tree search in general game playing," in *Workshop on Computer Games*. Springer, 2017, pp. 75–95.

[4] C. F. Sironi, J. Liu, D. Perez-Liebana, R. D. Gaina, I. Bravi, S. M. Lucas, and M. H. M. Winands, "Self-adaptive MCTS for general video game playing," in *Applications of Evolutionary Computation. EvoApplications 2018*, ser. Lecture Notes in Computer Science, K. Sim and P. Kaufmann, Eds., vol. 10784. Springer, 2018, pp. 358–375.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[6] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The n-tuple bandit evolutionary algorithm for game agent optimisation," *arXiv preprint arXiv:1802.05991*, 2018.

[7] D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands, "Enhancements for real-time Monte-Carlo tree search in general video game playing," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.