Comparing Randomization Strategies for Search-Control Parameters in Monte-Carlo Tree Search

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

***Abstract*—Monte-Carlo Tree Search (MCTS) has been applied successfully in many domains. Previous research has shown that adding randomization to certain components of MCTS might increase the diversification of the search and improve the performance. In a domain that tackles many games with different characteristics, like General Game Playing (GGP), trying to diversify the search might be a good strategy. This paper investigates the effect of randomizing search-control parameters for MCTS in GGP. Four different randomization strategies are compared and results show that randomizing parameter values before each simulation has a positive effect on the search in some of the tested games. Moreover, parameter randomization is compared with on-line parameter tuning.**

***Index Terms*—Monte-Carlo tree search, search-control parameter randomization, General Game Playing**

I. Introduction

The goal of General Game Playing (GGP) is to create agents that are capable of playing a wide variety of abstract games by only being given their rules [1]. Agents have a limited amount of time (usually a few seconds) to prepare to play and to select a move in-between game turns. In this setting, no prior or game-specific knowledge can be exploited by the agents, which have to acquire such knowledge on-line.

A search strategy that has been successfully applied to GGP is Monte-Carlo tree search (MCTS) [2], [3]. Numerous domain-independent enhancements for the different phases of MCTS have also been proposed, and shown to perform well in GGP [4]. The performance of MCTS and its enhancements is usually controlled by a certain number of parameters, the optimal values of which are in general game-dependent. However, in GGP it is not possible to tune the parameters for a specific game in advance, because the game the agent is going to play is not known. Therefore, parameters are usually tuned off-line by selecting a set of values that perform overall best on a predefined set of games.

Sironi *et al.* [5] proposed different strategies to tune search-control parameters on-line per game. Using on-line parameter tuning strategies enables the agent to adapt the search to each new game being played (i.e. self-adaptive MCTS). The on-line tuning strategy that is shown to perform best is the one based on the N-Tuple Bandit Evolutionary Algorithm (NTBEA) [6]. The on-line tuning agent that uses this strategy performs almost as good as the agent that uses off-line tuned parameters, being even better in a few games.

It has also been shown that in a general game playing domain where time setting are shorter, like the 40ms per move of the real-time domain of the General Video Game AI (GVG-AI) competition [7], tuning parameters on-line is, for most of the games, comparable to just randomizing them before each MCTS simulation [8]. In addition, on-line randomization of a small number of parameters seems to give robust settings for most of the tested games when a small predefined set of feasible values is considered for each parameter.

Although the time settings of GGP are less tight than in GVG-AI, a few second of search time per move is still quite short. Therefore, it is interesting to see how parameter randomization performs in GGP and how it compares to on-line parameter tuning. This paper compares four different strategies to randomize search-control parameters for MCTS in GGP. These strategies randomize parameters once per game run, once per turn, once per simulation and once per visited state, respectively. The randomization strategy that performs best is further analyzed to verify how it influences the performance with respect to fixed parameter values. Moreover, this randomization strategy is compared with on-line parameter tuning both directly and by evaluating the performance against an agent with fixed sub-optimal parameter values and against a benchmark GGP agent, CadiaPlayer [4].

The paper is organized as follows. Section II discusses related work. Section III gives background knowledge on MCTS and on-line parameter tuning. The parameter randomization strategies are described in Section IV. Section V reports the results of the performed experiments, and in Section VI the conclusion is given and future work is discussed.

II. Related Work

Previous research has shown how tree search might benefit from adding randomization to some of its aspects. A first example can be found in the work of Beal *et al.* [9], which shows the effects of using random numbers as intermediate state evaluations when performing minimax search in Chess.

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

An agent using only random evaluations for intermediate states is shown to outperform the same agent that uses 0 as heuristic instead. Adding a random term to existing heuristic functions is shown to be beneficial as well. This effect is explained by considering that random evaluations are able to capture some aspects of the structure of the tree.

Bošanskỳ *et al.* [10] proposed the use of a random tie-breaking rule when MCTS has to select among multiple actions of a player that have the same UCT value. They test an UCT agent that, for each role, selects an action randomly among those that have the UCT value within a predefined small offset from the highest UCT value. Results on a set of simultaneous move games show that this agent converges to a better approximation of the optimal strategy with respect to the agent that uses a deterministic tie-breaking rule.

Chen [11] proposed two randomization techniques for MCTS in the game of Go. The first technique consists in randomizing a set of parameters that control the selection phase. During a simulation, each of these parameters is randomized in a predefined range of values before selecting a move in each of the visited tree nodes. The second technique is used to add randomization to the play-out phase of MCTS by hierarchically randomizing the order of a set of predefined move generators before selecting a move in each state visited during the play-out. His results show that these randomization techniques improve the performance of the MCTS agent for different search budgets and sizes of the Go board.

III. Background

This section provides background on MCTS and on on-line tuning of search-control parameters for MCTS.

A. Monte-Carlo Tree Search

MCTS [2], [3] is a simulation based-search strategy that incrementally builds a tree representation of the state space of a game. It performs game simulations by repeating the following four phases until a search budget expires:

Selection: the tree built so far is traversed until a node that has not been fully expanded is reached. A node is fully expanded when all its successor nodes have been added to the tree. A *selection strategy* is used in each visited node to select the next move to visit.

Expansion: one or more nodes are added to the tree.

Play-out: starting from a node that was added to the tree during expansion, the game is simulated until a terminal state or a fixed depth is reached. A *play-out strategy* is used in each state to select which move to visit.

Backpropagation: the game result obtained at the end of the simulation is propagated back through all the visited tree nodes, and used to update statistics about the moves.

When the search budget expires, MCTS returns one of the moves in the root node to be played in the real game.

The standard MCTS selection strategy is UCT [2] (Upper Confidence bounds applied to Trees). Given a state s, UCT chooses the action a^* for a player using the UCB1 sampling strategy [12] as follows:

$$a^* = \operatorname*{argmax}_{a \in A(s)} \left\{ Q(s,a) + C \times \sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \quad . \qquad (1)$$

$A(s)$ is the set of legal moves in s, $Q(s,a)$ is the average result of all simulations in which move a has been selected in s, $N(s)$ is the number of times state s has been visited during the search and $N(s,a)$ is the number of times move a has been selected whenever state s was visited. The C constant is used to control the balance between exploitation of good moves and exploration of less visited ones.

Other successful selection strategies are the Rapid Action Value Estimation strategy [13], and its generalization, GRAVE [14]. GRAVE selects a move using the UCB1 formula (Eq. 1), where the term $Q(s,a)$ is substituted by the following quantity:

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

with

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

Here, $AMAF(s',a)$ is the average result of all the simulations in which move a was selected at any moment after s' was visited, and s' is the ancestor of s that has been visited at least *Ref* times. *Ref* is a threshold that specifies the minimum number of visits that a state should have for its AMAF statistics to be considered reliable. The *equivalence parameter* K specifies the number of simulations for which the two move estimates (i.e. $Q(s,a)$ and $AMAF(s',a)$) are weighted equal.

A successful play-out strategy is the Move Average Sampling Technique (MAST) [4]. During the play-out, MAST selects a random move with probability ϵ, and the move with highest $Q(a)$ with probability $(1-\epsilon)$. $Q(a)$ is the average result obtained by all the simulations in which move a was played at any point in the game.

B. On-line Parameter Tuning

To implement on-line parameter tuning the standard MCTS iteration is modified by adding two extra phases, therefore the search for each MCTS simulation is performed as follows:

- For each role in the game, for each parameter that is being tuned, an *allocation strategy* chooses a value in the predefined finite set of feasible values for the parameter.
- The standard MCTS phases are performed (selection, expansion, play-out and backpropagation), controlled by the parameter values chosen in the previous phase.
- The result obtained by the simulation is used to update statistics about the parameter values that were used to control the search during such simulation.

Different allocation strategies have been proposed for the first phase [5], [15]. In this paper, the one that seemed to perform best, the one based on the N-Tuple Bandit Evolutionary Algorithm (NTBEA) [6], is considered to compare on-line

TABLE I
PARAMETERS CONSIDERED IN THE EXPERIMENTS WITH THEIR DESCRIPTION, DEFAULT VALUE, SUB-OPTIMAL VALUE AND SETS OF FEASIBLE VALUES

Param.	Description	Default value	Sub-opt. value	Set of feasible values
C	Exploration constant for the UCT selection	0.2	0.9	$\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$
ϵ	Probability of selecting a random action with MAST	0.4	0	$\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
K	*Equivalence parameter* of GRAVE	250	∞	$\{0, 10, 50, 100, 250, 500, 750, 1\,000, 2\,000, \infty\}$
Ref	Visit threshold used by GRAVE	50	∞	$\{0, 50, 100, 250, 500, 1\,000, 10\,000, \infty\}$
VO	When using GRAVE selection in state s, select a random move a among the ones with value $V(a, s) \in [max_a(V(a, s)) - VO, max_a(V(a, s))]$	0.01	0.025	$\{0.001, 0.005, 0.01, 0.015, 0.02, 0.025\}$
T	During selection at s, if $N(s) < T$, select next action with play-out strategy	0	200	$\{0, 5, 10, 20, 30, 40, 50, 100, 200, \infty\}$

parameter tuning with parameter randomization. The NTBEA allocation strategy uses an *evolutionary algorithm*, which considers each combination of parameters as an individual and each single parameter as a gene. The evolutionary algorithm starts with a randomly generated combination of parameter values and evolves it over time using the statistics (i.e. average reward and number of visits of tuples of parameters of different lengths) collected in an *N-Tuple fitness landscape model* (LModel). The following are the steps that NTBEA repeats until the search budget expires:

1) Use the current combination of parameter values (i.e. individual) to control an MCTS simulation.
2) Use the reward obtained by the MCTS simulation to update the statistics for the parameter tuples in *LModel* that correspond to the evaluated combination.
3) Generate x neighbors of the evaluated combination, each by mutating the value of a randomly selected parameter in the combination.
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 the NTBEA allocation strategy and on how to use LModel to compute the estimate of the UCB1 values can be found in [5].

IV. SEARCH PARAMETER RANDOMIZATION

This section describes four different strategies to randomize search-control parameters: *per game*, *per turn*, *per simulation* and *per state*. They consider a finite set of parameters, each of which is associated to a finite fixed set of possible values, and they randomize the values of such parameters for each role in the game separately.

Per run: before the start of the search for an entire run of the game, this strategy sets for each role each considered parameter to a random value in its set of feasible values. The combination of parameters for each role are then kept fixed until the run of the game is over.

Per turn: for each role in the game, this strategy sets a random combination of feasible values for the considered parameters before starting the search of each game turn.

Per simulation: this strategy sets for each role a new random combination of feasible parameter values before the start of each new MCTS simulation. This strategy is the one that most resembles on-line parameter tuning, which is also modifying the combination of parameter values for each role before each simulation. The difference is that on-line parameter tuning uses the previously learned information to bias the selection of parameter values.

Per state: every time a state is visited during a simulation, this strategy randomizes for each role the values of the parameters used to perform the search in the state. This strategy is similar to the one proposed in [11], although in [11] the same random parameter values are assigned to all the roles.

V. EMPIRICAL EVALUATION

This section presents an analysis of parameter randomization for MCTS, by performing multiple series of experiments. Subsection V-A describes the experimental setup, while the results are given in Subsections V-B, V-C, V-D and V-E.

A. Setup

The baseline GGP agent used in the experiment implements MCTS with the GRAVE selection strategy [14] and the MAST play-out strategy [4]. When NTBEA is used to tune search parameters on-line, the settings are the same as in [5]. Thus, the number of neighbors x generated when evolving the considered parameter combination is 5, the exploration constant C_{NTBEA} used to compute the UCB1 value of parameter combinations with LModel is 0.2, and LModel considers only 1- and d-tuples, where d is the number of tuned parameters. No tuples of intermediate lengths are considered.

Table I summarizes all the parameters that can be randomized/tuned. The table reports their default value, the sub-optimal value they are set to in one of the series of experiments presented in Subsection V-E, and the set of feasible values used when randomizing or tuning the parameters. These values are the same as in [5]. The default values were obtained by off-line tuning over a predefined set of games, therefore are expected to perform generally well over the games tested in this paper.

All the experiments presented in the next subsections are performed on a set of 14 games [16]: *3D Tic Tac Toe*, *Breakthrough*, *Knightthrough*, *Chinook*, *Chinese Checkers* with 3 players, *Checkers*, *Connect 5*, *Quad* (the version played on a 7×7 board), *Sheep and Wolf*, *Tic Tac Chess Checkers Four* (*TTCC4*) with 2 and 3 players, *Connect 4*, *Pentago* and *Reversi*. Each experiment matches two agent types at a time against each other, ensuring that each of them is assigned to each role in the game the same amount of times over all the game runs. For 3-player games, all possible assignments of agent types to the roles are considered, except the two configurations that assign the same type to each role. All

TABLE II
COMPARISON OF PARAMETER RANDOMIZATION STRATEGIES AGAINST EACH OTHER FOR DIFFERENT NUMBERS OF RANDOMIZED PARAMETERS

	K, Ref						
	3D Tic Tac Toe	Breakthrough	Knightthrough	Chinook	Chinese Checkers 3P	Checkers	Connect 5
GAMERND	46.8(±2.38)	49.7(±2.53)	**53.3**(±2.53)	44.9(±2.33)	49.5(±2.52)	38.7(±2.35)	45.3(±1.92)
TURNRND	46.8(±2.39)	48.1(±2.53)	51.5(±2.53)	40.7(±2.29)	**51.1**(±2.52)	43.4(±2.40)	42.2(±1.90)
SIMRND	**59.5**(±2.31)	**52.9**(±2.53)	51.1(±2.53)	**58.1**(±2.33)	49.9(±2.52)	58.8(±2.36)	54.6(±1.82)
STATERND	46.9(±2.35)	49.4(±2.53)	44.1(±2.51)	56.4(±2.33)	49.5(±2.52)	**59.1**(±2.35)	**57.8**(±1.85)
	Quad	Sheep And Wolf	TTCC4 2P	TTCC4 3P	Connect 4	Pentago	Reversi
GAMERND	38.4(±2.32)	47.3(±2.53)	41.5(±2.42)	49.6(±2.45)	41.3(±2.38)	48.0(±2.49)	44.4(±2.40)
TURNRND	37.5(±2.31)	49.0(±2.53)	45.6(±2.46)	48.5(±2.46)	45.8(±2.41)	48.1(±2.48)	46.0(±2.41)
SIMRND	56.2(±2.37)	**53.5**(±2.52)	**60.5**(±2.39)	50.0(±2.45)	**60.8**(±2.34)	**52.2**(±2.49)	**57.0**(±2.39)
STATERND	**67.9**(±2.20)	50.1(±2.53)	52.3(±2.45)	**51.9**(±2.46)	52.1(±2.42)	51.7(±2.49)	52.5(±2.39)
	K, Ref, C, ϵ						
	3D Tic Tac Toe	Breakthrough	Knightthrough	Chinook	Chinese Checkers 3P	Checkers	Connect 5
GAMERND	43.9(±2.40)	49.5(±2.53)	**55.9**(±2.51)	50.3(±2.41)	50.7(±2.52)	52.4(±2.40)	46.4(±2.17)
TURNRND	34.9(±2.33)	45.5(±2.52)	50.7(±2.53)	42.7(±2.38)	41.4(±2.48)	30.4(±2.21)	32.6(±2.02)
SIMRND	**68.2**(±2.22)	**61.1**(±2.47)	51.7(±2.53)	**63.9**(±2.32)	**59.0**(±2.48)	**59.8**(±2.36)	54.9(±2.11)
STATERND	53.0(±2.39)	43.9(±2.51)	41.7(±2.50)	43.1(±2.41)	48.9(±2.52)	57.4(±2.40)	**66.1**(±2.02)
	Quad	SheepAndWolf	TTCC4 2P	TTCC4 3P	Connect 4	Pentago	Reversi
GAMERND	44.3(±2.45)	52.6(±2.53)	48.9(±2.51)	49.2(±2.48)	47.2(±2.44)	51.5(±2.45)	49.3(±2.49)
TURNRND	40.3(±2.42)	50.1(±2.53)	46.8(±2.51)	46.1(±2.48)	46.1(±2.45)	39.9(±2.40)	48.1(±2.49)
SIMRND	**72.3**(±2.19)	**52.9**(±2.53)	**58.1**(±2.47)	**53.5**(±2.48)	**61.5**(±2.38)	**62.4**(±2.36)	**51.4**(±2.49)
STATERND	43.0(±2.45)	44.5(±2.52)	46.2(±2.51)	51.2(±2.50)	45.1(±2.45)	46.1(±2.45)	51.3(±2.50)
	K, Ref, C, ϵ, T, VO						
	3D Tic Tac Toe	Breakthrough	Knightthrough	Chinook	Chinese Checkers 3P	Checkers	Connect 5
GAMERND	44.3(±2.41)	52.5(±2.53)	44.5(±2.52)	52.6(±2.42)	45.4(±2.51)	43.2(±2.40)	40.7(±2.16)
TURNRND	28.0(±2.21)	45.1(±2.52)	41.3(±2.49)	37.3(±2.35)	40.0(±2.47)	25.3(±2.10)	30.4(±2.06)
SIMRND	**71.8**(±2.15)	**53.6**(±2.52)	54.1(±2.52)	54.7(±2.41)	**57.8**(±2.49)	**67.2**(±2.26)	**67.9**(±2.01)
STATERND	55.9(±2.38)	48.7(±2.53)	**60.1**(±2.48)	**55.4**(±2.43)	56.8(±2.50)	64.3(±2.33)	61.0(±2.14)
	Quad	Sheep And Wolf	TTCC4 2P	TTCC4 3P	Connect 4	Pentago	Reversi
GAMERND	38.6(±2.40)	49.8(±2.53)	49.5(±2.52)	48.1(±2.49)	42.5(±2.43)	46.4(±2.46)	43.2(±2.48)
TURNRND	34.1(±2.34)	46.5(±2.53)	39.0(±2.46)	45.3(±2.48)	37.8(±2.39)	36.1(±2.37)	42.2(±2.48)
SIMRND	**75.3**(±2.11)	**55.6**(±2.52)	**60.4**(±2.47)	53.1(±2.50)	**67.2**(±2.28)	**64.0**(±2.37)	57.2(±2.47)
STATERND	52.0(±2.47)	48.1(±2.53)	51.1(±2.53)	**53.5**(±2.50)	52.5(±2.46)	53.5(±2.47)	**57.5**(±2.47)

configurations are run the same number of times until each agent type has played at least 500 games in total. All agents are given $1s$ start- and play-clock, except CADIAPLAYER, which uses $10s$ start- and play-clock in order to reach a number of simulations similar to the other agents. Experimental results always report the average win percentage of one of the two agent types with a 95% confidence interval. The average win percentage for a game is computed by splitting 1 point among the agents that achieved the highest score and assigning 0 points to all other agents. Bold results indicate the agent type with the highest win rate for the corresponding game and number of parameters. Experiments were performed on a Linux server consisting of 64 AMD Opteron 6274 2.2-GHz cores, except the ones presented in Table IV and V, which were performed on a Linux server consisting of 48 AMD Opteron 6344 2.6-GHz cores.

B. Comparison of Parameter Randomization Strategies

This series of experiments evaluates the randomization strategies, both by directly matching the agents that implement them against each other, and by comparing them when matched against the agent with fixed default parameters.

Table II shows the results obtained by performing a round-robin tournament with all the randomizing agents (i.e. STATERND, SIMRND, TURNRND, GAMERND). Each block of the table corresponds to a different number of parameters being randomized, each row corresponds to an agent, and each column to a game. Result in the table represent the average win percentage of the row agent against all other agents. As can be seen, for most of the games, independently of the number of parameters being considered, the agent that randomizes parameters per simulation seems to be the best performing one. This does not contradict the findings in [11]. Although the author mentions that randomizing parameters per simulation did not perform as well as per state, he tested only in the game of Go. When tested on more games, there are still a few games where randomization per state performs better (e.g. Quad for 2 parameters, Connect 5 for 4, Knightthrough for 6), but randomizing per simulation seems overall best.

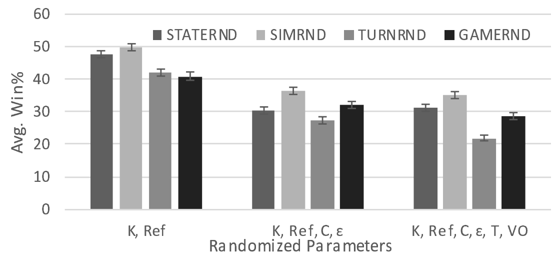

Fig. 1. Win percentage of the agents with the parameter randomization strategies against the agent with fixed default parameter values.

The performance of the randomizing agents against the agent with default fixed values is shown in Fig. 1. For different number of parameters, the figure reports the average win percentage of each of the randomizing agents over all the tested games. Once again, the agent that randomizes parameters per simulation is the one showing the best performance overall when compared to the other randomizing agents. However, none of the agents seems to be overall better than the agent that uses fixed default values. Moreover, the results suggest that the performance of the randomizing agents drops with

TABLE III
COMPARING ALL FEASIBLE VALUES OF C WITH VALUE RANDOMIZATION PER SIMULATION

	3D Tic Tac Toe	Breakthrough	Knightthrough	Chinook	Chinese Checkers 3P	Checkers	Connect 5
$C = 0.1$	51.5(±1.38)	66.0(±1.38)	63.9(±1.40)	42.2(±1.35)	48.2(±1.45)	54.2(±1.37)	45.7(±1.07)
$C = 0.2$	**55.6**(±1.36)	**78.2**(±1.21)	**69.0**(±1.35)	63.8(±1.31)	**58.9**(±1.43)	**65.8**(±1.31)	**53.0**(±1.05)
$C = 0.3$	54.9(±1.36)	71.0(±1.33)	62.5(±1.41)	**66.9**(±1.30)	57.8(±1.44)	61.3(±1.34)	51.9(±1.04)
$C = 0.4$	52.5(±1.35)	58.3(±1.44)	54.0(±1.46)	63.4(±1.32)	54.7(±1.45)	56.4(±1.37)	52.7(±1.05)
$C = 0.5$	50.8(±1.35)	45.6(±1.46)	47.6(±1.46)	56.5(±1.36)	51.5(±1.45)	50.7(±1.38)	52.7(±1.04)
$C = 0.6$	50.0(±1.37)	39.0(±1.43)	42.4(±1.44)	47.6(±1.37)	48.5(±1.45)	47.2(±1.38)	51.9(±1.06)
$C = 0.7$	47.5(±1.36)	35.8(±1.40)	38.6(±1.42)	39.3(±1.34)	45.1(±1.45)	41.3(±1.36)	49.6(±1.10)
$C = 0.8$	46.8(±1.36)	33.2(±1.38)	37.9(±1.42)	35.2(±1.30)	44.8(±1.45)	38.7(±1.35)	47.1(±1.09)
$C = 0.9$	43.7(±1.35)	30.0(±1.34)	41.0(±1.44)	33.0(±1.28)	41.5(±1.43)	36.9(±1.34)	46.1(±1.09)
SIMRND$_C$	46.8(±1.35)	42.8(±1.45)	43.0(±1.45)	52.1(±1.38)	49.0(±1.45)	47.6(±1.38)	49.3(±1.07)
	Quad	Sheep And Wolf	TTCC4 2P	TTCC4 3P	Connect 4	Pentago	Reversi
$C = 0.1$	24.0(±1.16)	**59.6**(±1.43)	39.1(±1.39)	49.6(±1.42)	29.5(±1.26)	34.6(±1.35)	**61.2**(±1.40)
$C = 0.2$	37.5(±1.34)	55.3(±1.45)	61.5(±1.39)	**52.1**(±1.41)	48.5(±1.40)	51.6(±1.41)	59.9(±1.41)
$C = 0.3$	53.2(±1.39)	51.7(±1.46)	**62.7**(±1.38)	51.1(±1.42)	54.5(±1.39)	**54.6**(±1.39)	54.1(±1.43)
$C = 0.4$	59.9(±1.36)	49.7(±1.46)	58.3(±1.40)	51.0(±1.42)	**56.4**(±1.39)	53.5(±1.38)	50.1(±1.44)
$C = 0.5$	59.5(±1.37)	47.8(±1.46)	52.0(±1.43)	51.2(±1.42)	54.5(±1.40)	50.9(±1.39)	49.0(±1.44)
$C = 0.6$	59.2(±1.38)	48.2(±1.46)	48.9(±1.43)	48.8(±1.42)	54.2(±1.40)	51.1(±1.39)	46.4(±1.44)
$C = 0.7$	55.5(±1.40)	46.4(±1.46)	45.7(±1.43)	49.7(±1.42)	50.7(±1.41)	51.7(±1.39)	44.0(±1.43)
$C = 0.8$	48.5(±1.41)	46.9(±1.46)	42.0(±1.42)	49.7(±1.42)	49.5(±1.41)	51.2(±1.39)	44.2(±1.43)
$C = 0.9$	40.7(±1.39)	45.3(±1.45)	40.8(±1.41)	46.8(±1.42)	49.7(±1.41)	46.4(±1.39)	45.3(±1.43)
SIMRND$_C$	**62.0**(±1.35)	49.0(±1.46)	49.1(±1.43)	50.0(±1.43)	52.5(±1.40)	54.5(±1.38)	45.8(±1.43)

TABLE IV
COMPARING ALL FEASIBLE VALUES OF ϵ WITH VALUE RANDOMIZATION PER SIMULATION

	3D Tic Tac Toe	Breakthrough	Knightthrough	Chinook	Chinese Checkers 3P	Checkers	Connect 5
$\epsilon = 0.0$	43.0(±1.25)	**64.2**(±1.27)	**73.7**(±1.16)	38.8(±1.17)	32.3(±1.23)	42.2(±1.22)	37.8(±1.00)
$\epsilon = 0.1$	47.0(±1.25)	59.3(±1.30)	67.4(±1.24)	58.8(±1.20)	45.8(±1.31)	58.4(±1.23)	46.6(±0.98)
$\epsilon = 0.2$	46.3(±1.25)	57.3(±1.31)	63.2(±1.27)	**64.7**(±1.16)	50.8(±1.32)	64.2(±1.19)	50.4(±0.98)
$\epsilon = 0.3$	48.2(±1.25)	57.4(±1.31)	57.3(±1.31)	63.4(±1.17)	55.1(±1.31)	**64.8**(±1.19)	51.1(±0.97)
$\epsilon = 0.4$	48.4(±1.25)	56.6(±1.31)	54.7(±1.32)	61.1(±1.19)	55.5(±1.31)	61.2(±1.21)	54.1(±0.96)
$\epsilon = 0.5$	49.8(±1.26)	55.2(±1.31)	53.7(±1.32)	58.8(±1.20)	**57.4**(±1.30)	56.7(±1.23)	56.3(±0.97)
$\epsilon = 0.6$	53.1(±1.25)	53.0(±1.32)	50.9(±1.32)	54.3(±1.22)	54.5(±1.31)	51.1(±1.24)	57.6(±0.95)
$\epsilon = 0.7$	55.4(±1.25)	51.3(±1.32)	47.5(±1.32)	50.2(±1.22)	53.9(±1.31)	46.5(±1.25)	**58.5**(±0.97)
$\epsilon = 0.8$	**59.7**(±1.23)	46.8(±1.32)	40.9(±1.30)	43.2(±1.21)	50.9(±1.32)	42.8(±1.23)	58.1(±1.00)
$\epsilon = 0.9$	57.4(±1.25)	37.3(±1.28)	31.5(±1.23)	36.3(±1.18)	48.2(±1.32)	39.3(±1.22)	53.9(±1.05)
$\epsilon = 1.0$	44.0(±1.29)	22.9(±1.11)	16.7(±0.98)	25.4(±1.06)	42.5(±1.30)	35.8(±1.19)	36.2(±1.13)
SIMRND$_\epsilon$	47.7(±1.26)	38.7(±1.29)	42.6(±1.31)	44.9(±1.20)	53.0(±1.31)	36.8(±1.20)	39.3(±1.01)
	Quad	Sheep And Wolf	TTCC4 2P	TTCC4 3P	Connect 4	Pentago	Reversi
$\epsilon = 0.0$	18.6(±0.98)	38.7(±1.29)	40.5(±1.26)	41.6(±1.25)	26.9(±1.11)	46.1(±1.27)	43.2(±1.28)
$\epsilon = 0.1$	26.2(±1.10)	41.4(±1.30)	60.7(±1.25)	47.3(±1.27)	29.9(±1.15)	49.2(±1.27)	54.0(±1.30)
$\epsilon = 0.2$	33.7(±1.19)	42.2(±1.31)	67.0(±1.20)	50.2(±1.27)	35.6(±1.20)	51.0(±1.28)	60.5(±1.27)
$\epsilon = 0.3$	39.7(±1.23)	44.7(±1.31)	**67.3**(±1.19)	52.2(±1.27)	39.5(±1.23)	52.1(±1.28)	**61.3**(±1.27)
$\epsilon = 0.4$	47.6(±1.26)	47.1(±1.32)	62.9(±1.23)	53.2(±1.27)	45.4(±1.25)	**52.5**(±1.28)	59.1(±1.28)
$\epsilon = 0.5$	53.8(±1.26)	48.8(±1.32)	58.2(±1.26)	**54.0**(±1.27)	51.7(±1.25)	51.4(±1.28)	56.9(±1.29)
$\epsilon = 0.6$	61.5(±1.23)	51.9(±1.32)	50.4(±1.28)	52.7(±1.27)	56.8(±1.25)	49.3(±1.27)	52.9(±1.30)
$\epsilon = 0.7$	67.0(±1.18)	55.1(±1.31)	43.1(±1.27)	51.5(±1.27)	62.3(±1.22)	49.7(±1.27)	49.7(±1.30)
$\epsilon = 0.8$	**69.4**(±1.17)	56.3(±1.31)	37.9(±1.25)	49.2(±1.27)	65.0(±1.20)	48.9(±1.28)	45.4(±1.29)
$\epsilon = 0.9$	68.4(±1.19)	60.4(±1.29)	30.8(±1.19)	48.4(±1.28)	**67.7**(±1.18)	49.2(±1.28)	38.2(±1.26)
$\epsilon = 1.0$	57.2(±1.29)	**60.8**(±1.29)	25.8(±1.13)	45.6(±1.27)	66.5(±1.19)	48.1(±1.28)	29.2(±1.18)
SIMRND$_\epsilon$	56.8(±1.25)	52.4(±1.32)	55.2(±1.27)	**54.0**(±1.27)	52.9(±1.26)	**52.5**(±1.27)	49.6(±1.30)

the increase in the number of randomized parameters.

C. Randomization per Simulation vs Fixed Parameters

These series of experiments further analyze the randomization strategy that performed overall best in the previous series of experiments: randomization per simulation. The purpose is to verify if randomization of parameter values during the search brings any contribution to the performance of the MCTS agent with respect to keeping parameter values fixed for the whole game. Each series of experiments focuses on one single parameter, considering the agents that keep the parameter fixed to one of its feasible values and the agent that randomizes such parameter per simulation (SIMRND). All other parameters for SIMRND are set to their default values. These agents are matched against each other in a round-robin tournament and, for each game, the average win percentage of each agent against all other agents is reported in the results.

Tables III, IV and V show the results of such experiments for the parameter C, ϵ and K, respectively. Being quite time-consuming, these series of experiments have been performed only for the parameters that seemed to be more relevant for the search. First of all, it is interesting to notice that for a few games randomizing the parameter values during the search achieves one of the highest win percentages. This can be observed in Quad and Pentago for the parameter C, in TTCC4 with 3 players and Pentago for the parameter ϵ and in about half of the games for the parameter K. Randomization seems to be particularly effective for this parameter.

In general, for many of the games randomization per simulation seems to perform better than a few of the fixed values of the parameter. Moreover, there are a few games where randomizing parameter values, even if not the best choice, still performs better than the default fixed value. For some games, this happens because the default value, despite being optimal

TABLE V
COMPARING ALL FEASIBLE VALUES OF K WITH VALUE RANDOMIZATION PER SIMULATION

	3D Tic Tac Toe	Breakthrough	Knightthrough	Chinook	Chinese Checkers 3P	Checkers	Connect 5
$K = 0$	47.8(±1.32)	46.2(±1.38)	57.8(±1.37)	40.1(±1.26)	51.1(±1.38)	33.1(±1.26)	48.6(±1.06)
$K = 10$	51.2(±1.33)	50.0(±1.39)	**59.9**(±1.36)	40.9(±1.27)	51.4(±1.38)	41.5(±1.31)	52.6(±1.04)
$K = 50$	53.2(±1.31)	53.6(±1.38)	56.8(±1.37)	45.3(±1.28)	**52.4**(±1.38)	55.7(±1.31)	54.5(±1.01)
$K = 100$	53.2(±1.31)	54.2(±1.38)	54.5(±1.38)	51.9(±1.28)	52.2(±1.38)	61.7(±1.28)	55.0(±1.01)
$K = 250$	56.4(±1.30)	56.8(±1.37)	53.8(±1.38)	55.8(±1.27)	52.0(±1.38)	**66.6**(±1.24)	**58.0**(±0.98)
$K = 500$	56.5(±1.30)	**56.9**(±1.37)	51.9(±1.39)	57.4(±1.26)	**52.4**(±1.38)	63.5(±1.26)	57.8(±0.98)
$K = 750$	**57.1**(±1.30)	55.2(±1.38)	51.0(±1.39)	57.2(±1.26)	52.0(±1.38)	58.7(±1.30)	55.2(±0.98)
$K = 1000$	56.5(±1.30)	54.0(±1.38)	50.5(±1.39)	55.5(±1.27)	51.9(±1.38)	53.9(±1.31)	53.3(±0.99)
$K = 2000$	53.6(±1.31)	52.7(±1.38)	50.4(±1.39)	54.6(±1.27)	49.7(±1.38)	42.7(±1.31)	47.9(±0.99)
$K = \infty$	8.9(±0.77)	16.7(±1.03)	16.6(±1.03)	29.3(±1.16)	34.4(±1.31)	8.4(±0.73)	10.2(±0.62)
SIMRND$_K$	55.7(±1.30)	53.6(±1.38)	46.8(±1.38)	**62.0**(±1.24)	50.3(±1.38)	64.3(±1.26)	56.9(±0.98)

	Quad	Sheep And Wolf	TTCC4 2P	TTCC4 3P	Connect 4	Pentago	Reversi
$K = 0$	49.0(±1.31)	**51.6**(±1.39)	28.4(±1.23)	49.9(±1.34)	41.6(±1.30)	37.4(±1.31)	53.4(±1.36)
$K = 10$	50.2(±1.32)	49.9(±1.39)	33.0(±1.27)	51.5(±1.34)	44.2(±1.31)	38.9(±1.32)	58.3(±1.34)
$K = 50$	50.8(±1.31)	50.8(±1.39)	43.6(±1.34)	**53.3**(±1.33)	48.4(±1.32)	44.6(±1.34)	61.1(±1.33)
$K = 100$	50.8(±1.31)	50.4(±1.39)	50.7(±1.34)	50.6(±1.33)	50.0(±1.31)	48.7(±1.35)	**61.5**(±1.32)
$K = 250$	51.7(±1.31)	50.6(±1.39)	59.5(±1.31)	51.7(±1.33)	54.3(±1.31)	55.7(±1.33)	57.6(±1.35)
$K = 500$	53.4(±1.31)	50.2(±1.39)	63.2(±1.28)	52.2(±1.33)	55.9(±1.31)	58.0(±1.32)	52.0(±1.36)
$K = 750$	54.6(±1.31)	50.0(±1.39)	**63.9**(±1.28)	51.1(±1.33)	56.8(±1.31)	60.1(±1.30)	48.9(±1.36)
$K = 1000$	55.7(±1.30)	49.6(±1.39)	63.4(±1.28)	50.9(±1.33)	57.1(±1.31)	60.5(±1.31)	44.0(±1.36)
$K = 2000$	57.3(±1.30)	49.8(±1.39)	59.0(±1.32)	49.2(±1.34)	56.7(±1.30)	60.2(±1.31)	38.7(±1.33)
$K = \infty$	10.6(±0.83)	45.5(±1.38)	21.9(±1.13)	39.1(±1.31)	25.3(±1.15)	23.9(±1.15)	19.5(±1.08)
SIMRND$_K$	**65.9**(±1.25)	**51.6**(±1.39)	63.5(±1.29)	50.4(±1.34)	**59.8**(±1.30)	**62.0**(±1.30)	55.0(±1.36)

TABLE VI
PARAMETER RANDOMIZATION AGAINST PARAMETER TUNING

Game	SIMRND		
	K, Ref	K, Ref, C, ϵ	K, Ref, C, ϵ, T, VO
3D Tic Tac Toe	47.1(±4.01)	52.1(±4.07)	51.7(±4.16)
Breakthrough	39.8(±4.29)	8.8(±2.49)	12.6(±2.91)
Knightthrough	40.8(±4.31)	10.6(±2.70)	15.6(±3.18)
Chinook	50.8(±4.08)	19.7(±3.24)	36.9(±4.11)
Chinese Checkers 3P	44.6(±4.34)	47.2(±4.36)	56.3(±4.33)
Checkers	49.7(±4.10)	25.9(±3.63)	62.5(±4.04)
Connect 5	48.1(±3.15)	66.1(±3.32)	66.7(±3.53)
Quad	55.2(±4.17)	65.8(±3.99)	93.0(±2.06)
Sheep And Wolf	48.8(±4.39)	52.2(±4.38)	51.4(±4.39)
TTCC4 2P	50.3(±4.22)	22.7(±3.61)	34.5(±4.12)
TTCC4 3P	50.9(±4.27)	53.9(±4.26)	59.1(±4.22)
Connect 4	49.1(±4.21)	59.7(±4.19)	67.6(±3.92)
Pentago	47.7(±4.15)	54.7(±4.04)	57.5(±4.12)
Reversi	48.1(±4.31)	42.1(±4.26)	57.4(±4.27)
Avg. Win%	47.9(±1.11)	41.5(±1.11)	51.6(±1.14)

over all the set of games on which it was tuned, is actually not optimal for the specific game. Examples are Quad, Connect 4 and Pentago for C, Quad, Sheep and Wolf, TTCC4 with 3 players and Connect 4 for ϵ, and Quad, TTCC4 with 2 players, Connect 4 and Pentago for K.

D. Randomization per Simulation vs Parameter Tuning

This series of experiments compares the agent that uses parameter randomization per simulation (SIMRND) with the one that tunes parameter values on-line with the NTBEA allocation strategy (NTBEA). Table VI shows the results obtained by matching the two agents against each other for 2, 4 and 6 randomized/tuned parameters. Regarding the overall performance, for 2 and 4 parameters on-line tuning seems to perform better than randomization, while when the tuned parameters increase to 6, randomization performs in general as well as on-line tuning. This is probably because with 6 parameters the number of possible value combinations becomes too high. With the short time settings of GGP the agent does not have sufficient time to converge to good combinations, therefore evaluating combinations almost randomly. Among the tested ones, 4 seems to be the most interesting number of parameters to compare randomization and on-line tuning against the default values. With 2 and 6 parameters the performance of the two agents is close for many of the games, while with 4 there is a clearer distinction between games for which tuning performs best and games for which randomization performs best.

Looking at specific games, interesting results are the ones for Knightthrough and Breakthrough. For these two games, for each number of considered parameters, on-line tuning seems more effective than randomization. In addition, for Chinook and TTCC4 with 2 players on-line tuning performs much better than randomization when the number of tuned parameters is 4 or 6. On the contrary, in Quad randomization achieves a much higher performance than on-line tuning for all number of parameters. Moreover, in Connect 5 and Connect 4, parameter randomization shows a better performance than online tuning for 4 and 6 parameters, and in Checkers its performance drops when going from 2 to 4 parameters, but becomes better than on-line tuning with 6 parameters.

E. Comparison of Default Parameter Values, Parameter Randomization and On-line Parameter Tuning

These series of experiments compare parameter randomization and parameter tuning with the fixed default values. First of all, the agent that randomizes parameter values (SIMRND) and the one that tunes them on-line (NTBEA) are matched directly against the agent that uses the fixed default parameter values (DEFAULT). Subsequently, all three agents are compared by matching them against two different types of opponents: an agent that uses fixed sub-optimal values for the parameters and a successful GGP agent, CADIAPLAYER [4], which won three editions of the GGP competition.

Results obtained by matching SIMRND and NTBEA against DEFAULT are shown in Table VII. Results are reported for 2, 4 and 6 randomized/tuned parameters. These results are in line to the results presented in Subsection V-D. When matched

TABLE VII

COMPARISON OF SIMRND AND NTBEA AGAINST THE AGENT WITH DEFAULT PARAMETER VALUES

Game	K, Ref		K, Ref, C, ϵ		$K, Ref, C, \epsilon, T, VO$	
	SIMRND	NTBEA	SIMRND	NTBEA	SIMRND	NTBEA
3D Tic Tac Toe	**46.4**(±4.12)	46.0(±4.11)	39.4(±4.00)	**39.5**(±4.08)	**40.6**(±4.01)	38.6(±4.05)
Breakthrough	40.4(±4.31)	**48.6**(±4.39)	11.0(±2.75)	**55.2**(±4.36)	9.4(±2.56)	**31.6**(±4.08)
Knightthrough	44.2(±4.36)	**46.8**(±4.38)	20.6(±3.55)	**68.2**(±4.09)	19.8(±3.50)	**50.2**(±4.39)
Chinook	61.8(±3.99)	**63.5**(±3.95)	23.6(±3.61)	**51.0**(±4.05)	19.8(±3.34)	**31.6**(±3.94)
Chinese Checkers 3P	45.6(±4.35)	**51.0**(±4.37)	34.7(±4.16)	**42.7**(±4.32)	**33.7**(±4.13)	28.0(±3.92)
Checkers	**48.8**(±4.08)	47.6(±4.13)	17.9(±3.17)	**40.4**(±4.06)	20.4(±3.29)	**20.9**(±3.42)
Connect 5	43.9(±3.12)	**45.7**(±3.05)	**36.5**(±3.22)	28.6(±3.07)	**45.7**(±3.21)	33.2(±3.26)
Quad	**65.0**(±3.93)	60.1(±4.05)	**72.8**(±3.71)	51.9(±4.21)	**72.4**(±3.67)	17.2(±3.13)
Sheep And Wolf	52.0(±4.38)	**52.2**(±4.38)	**49.8**(±4.39)	44.8(±4.36)	**50.0**(±4.39)	46.2(±4.37)
TTCC4 2P	49.5(±4.27)	**51.5**(±4.19)	20.6(±3.47)	**49.7**(±4.20)	19.0(±3.40)	**33.6**(±4.03)
TTCC4 3P	**48.7**(±4.26)	48.4(±4.26)	**46.1**(±4.28)	43.1(±4.18)	**40.8**(±4.23)	39.4(±4.15)
Connect 4	50.9(±4.17)	**55.6**(±4.18)	**55.4**(±4.13)	46.8(±4.20)	**48.0**(±4.23)	30.6(±3.89)
Pentago	53.7(±4.19)	**55.3**(±4.21)	**50.2**(±4.22)	43.5(±4.15)	**42.6**(±4.13)	42.1(±4.18)
Reversi	42.8(±4.29)	**46.9**(±4.33)	31.0(±3.99)	**45.1**(±4.33)	28.7(±3.92)	**33.5**(±4.07)
Avg. Win%	49.6(±1.12)	**51.4**(±1.12)	36.4(±1.08)	**46.5**(±1.12)	**35.1**(±1.07)	34.0(±1.07)

TABLE VIII

COMPARISON OF DEFAULT, SIMRND AND NTBEA AGAINST SUB-OPTIMAL FIXED PARAMETER VALUES

Game	K, Ref			K, Ref, C, ϵ			$K, Ref, C, \epsilon, T, VO$		
	DEFAULT	SIMRND	NTBEA	DEFAULT	SIMRND	NTBEA	DEFAULT	SIMRND	NTBEA
3D Tic Tac Toe	**94.9**(±1.90)	92.4(±2.22)	93.4(±2.03)	**87.1**(±2.67)	80.5(±3.21)	85.3(±2.86)	**98.8**(±0.91)	97.3(±1.38)	96.4(±1.61)
Breakthrough	**97.8**(±1.29)	95.6(±1.80)	97.0(±1.50)	96.4(±1.63)	73.2(±3.89)	**97.8**(±1.29)	96.8(±1.54)	83.2(±3.28)	**97.0**(±1.50)
Knightthrough	96.6(±1.59)	92.4(±2.33)	**96.8**(±1.54)	93.0(±2.24)	63.0(±4.24)	**99.0**(±0.87)	95.2(±1.88)	70.2(±4.01)	**95.6**(±1.80)
Chinook	73.1(±3.55)	**83.5**(±3.07)	80.9(±3.21)	83.3(±3.08)	63.8(±4.06)	**86.0**(±2.85)	**93.5**(±2.01)	86.6(±2.82)	89.1(±2.51)
Chinese Checkers 3P	67.9(±4.08)	68.5(±4.06)	**72.0**(±3.92)	**79.6**(±3.52)	70.8(±3.97)	70.6(±3.98)	**89.7**(±2.66)	85.3(±3.09)	76.4(±3.71)
Checkers	94.8(±1.80)	95.3(±1.76)	**95.7**(±1.68)	**88.4**(±2.55)	67.4(±3.85)	84.1(±2.95)	**99.3**(±0.70)	98.8(±0.83)	96.5(±1.39)
Connect 5	**94.7**(±1.54)	92.8(±1.66)	94.5(±1.40)	**85.5**(±2.55)	83.9(±2.77)	67.0(±3.63)	96.1(±1.36)	**97.2**(±1.37)	90.1(±2.32)
Quad	91.0(±2.40)	**96.8**(±1.33)	95.9(±1.64)	79.0(±3.52)	**91.3**(±2.42)	81.8(±3.29)	98.3(±1.08)	**99.4**(±0.68)	92.6(±2.21)
Sheep And Wolf	**58.4**(±4.32)	56.6(±4.35)	57.0(±4.34)	**65.4**(±4.17)	**65.4**(±4.17)	62.2(±4.25)	**77.0**(±3.69)	70.8(±3.99)	67.4(±4.11)
TTCC4 2P	90.0(±2.60)	92.0(±2.36)	**93.5**(±2.15)	89.4(±2.67)	71.3(±3.95)	**89.5**(±2.68)	**98.2**(±1.17)	91.8(±2.41)	94.2(±2.05)
TTCC4 3P	**68.4**(±3.97)	65.0(±4.08)	67.7(±3.96)	69.7(±3.95)	**70.0**(±3.95)	67.1(±4.02)	**85.0**(±3.05)	81.7(±3.34)	79.4(±3.47)
Connect 4	93.1(±2.09)	**95.7**(±1.70)	94.2(±1.94)	90.8(±2.40)	**94.3**(±1.99)	87.8(±2.79)	97.3(±1.35)	**98.1**(±1.08)	95.1(±1.84)
Pentago	84.1(±3.15)	**89.4**(±2.64)	87.7(±2.83)	**87.7**(±2.83)	85.6(±3.02)	81.6(±3.32)	**93.4**(±2.11)	**93.4**(±2.07)	90.3(±2.56)
Reversi	**85.3**(±3.05)	81.2(±3.35)	83.4(±3.23)	**83.8**(±3.20)	67.8(±4.04)	69.5(±4.00)	**95.4**(±1.80)	93.8(±2.10)	92.9(±2.19)
Avg. Win%	85.0(±0.81)	85.5(±0.80)	**86.4**(±0.78)	**84.2**(±0.83)	74.9(±0.99)	80.7(±0.89)	**93.8(**±0.55)	89.1(±0.72)	89.5(±0.70)

TABLE IX

COMPARISON OF DEFAULT, SIMRND AND NTBEA AGAINST CADIAPLAYER

Game	DEFAULT	K, Ref		K, Ref, C, ϵ		$K, Ref, C, \epsilon, T, VO$	
		SIMRND	NTBEA	SIMRND	NTBEA	SIMRND	NTBEA
3D Tic Tac Toe	92.1(±2.36)	**92.3**(±2.26)	91.9(±2.34)	**91.4**(±2.41)	90.4(±2.55)	**91.9**(±2.35)	86.7(±2.89)
Breakthrough	63.2(±4.23)	50.6(±4.39)	**61.8**(±4.26)	23.2(±3.70)	**68.0**(±4.09)	19.4(±3.47)	**45.8**(±4.37)
Knightthrough	50.8(±4.39)	35.8(±4.21)	**52.2**(±4.38)	19.6(±3.48)	**74.8**(±3.81)	16.0(±3.22)	**45.0**(±4.37)
Chinook	82.8(±3.22)	86.6(±2.92)	**88.0**(±2.74)	54.1(±4.22)	**81.3**(±3.28)	55.1(±4.24)	**63.4**(±4.10)
Checkers	90.6(±2.32)	86.5(±2.72)	**91.2**(±2.28)	63.2(±3.97)	**87.6**(±2.71)	**65.8**(±3.92)	52.6(±4.12)
Connect 5	70.4(±3.18)	66.8(±3.33)	**68.2**(±3.29)	**61.2**(±3.73)	45.5(±3.78)	**70.4**(±3.54)	51.9(±3.95)
Quad	98.8(±0.96)	**99.6**(±0.55)	99.2(±0.78)	98.8(±0.96)	**99.4**(±0.68)	**99.4**(±0.68)	93.0(±2.24)
Sheep And Wolf	56.8(±4.35)	56.8(±4.35)	**60.4**(±4.29)	**51.6**(±4.38)	**51.6**(±4.38)	**55.6**(±4.36)	50.0(±4.39)
Connect 4	68.2(±3.90)	65.1(±4.04)	**69.7**(±3.92)	**68.5**(±3.98)	63.2(±4.06)	**65.4**(±4.04)	48.0(±4.24)
Pentago	73.0(±3.80)	75.0(±3.62)	**78.1**(±3.52)	69.7(±3.95)	**71.3**(±3.80)	**64.7**(±4.11)	62.6(±4.10)
Avg. Win%	74.7(±1.16)	71.5(±1.21)	**76.1**(±1.14)	60.1(±1.32)	**73.3**(±1.19)	**60.4**(±1.32)	59.9(±1.32)

against an opponent that uses generally good fixed default values, the difference in the overall performance between SIMRND and NTBEA is similar to the difference in performance that was observed when the two agents were matched against each other directly. Moreover, for 2 parameters both agents seem to have at least the same performance of DEFAULT in many of the tested games. For 4 parameters the performance of NTBEA is still quite close to the one of DEFAULT, while the performance of SIMRND drops for most of the games. For 6 parameters both tuning and randomizing parameter values does not seem to provide any benefit in most of the games. Once again, Quad is an interesting game, because it seems to significantly benefit from parameter randomization for any tested number of randomized parameters.

The results obtained by matching the DEFAULT, SIMRND and NTBEA agents against the agent that uses sub-optimal fixed values for the parameters are presented in Table VIII. These agents are compared for 2, 4 and 6 randomized/tuned parameters. Note that for the sub-optimal agent only the parameters that are being randomized/tuned by the opponent are set to sub-optimal values, other parameters are set to their default values. Once again, results show that for 2 and 6 parameters SIMRND and NTBEA have a very close performance, while for 4 parameters NTBEA performs better. In general, all agents have a much better performance than the agent with sub-optimal values. This supports the claim that parameter values have a strong influence on the search and that it is worth investigating which are the best values for each game. Moreover, in a situation where the optimal parameter values for a specific game are not known in advance, tuning or randomizing them seems to be a valid approach, rather than setting an arbitrary combination that might be sub-optimal.

Results of the final series of experiments are shown in Table IX. This table reports the results obtained by matching DEFAULT, SIMRND and NTBEA against CADIAPLAYER. All the agents tested so far implement the same MCTS strategy with the same enhancements. The purpose of this series of experiments is to verify how the agents perform against an opponent with a different implementation of MCTS. For most of the games, independently of the number of considered parameters, all agents show a better performance than CADIAPLAYER, with NTBEA that tunes 2 parameters being the best. It is confirmed that on-line parameter tuning is successful in Breakthrough and Knightthrough for 2 tuned parameters and in particular for 4, while parameter randomization performs the worst on these two games for 4 and 6 parameters.

Next, for 2 parameters, in all previous series of experiments the performance of SIMRND was close to the one of NTBEA. However, against CADIAPLAYER the difference in performance is higher. This means that the relative performance of these two agent does not only depend the number of parameters that they are considering, but also by the type of opponent. From previous experiments, tuning only 2 parameters might have seemed not very interesting, because it was only slightly improving the performance over randomization. However, against CADIAPLAYER on-line tuning is shown to be better than just randomizing the parameters. Therefore, for the agent to be more robust against different types of opponents, 2 parameters are still worth tuning. The biggest gap in the performance of SIMRND with respect to NTBEA is visible for 4 parameters. Moreover, by going from 2 to 4 parameters the performance of SIMRND drops much more than the one of NTBEA (more than 9 points for SIMRND and less than 3 for NTBEA). This suggest that, for different opponents, on-line tuning might be more robust than parameter randomization.

VI. CONCLUSION AND FUTURE WORK

This paper evaluated four different strategies that randomize search control parameters for MCTS in GGP: per game, per turn, per simulation and per state. Moreover, randomization per simulation has been compared with on-line parameter tuning, giving more insights on the performance of both approaches.

Results showed that, when compared to each other, the randomization strategy that performs best is the one that randomizes parameter values before each simulation. Moreover, for single parameters, it has been shown that for some games it is better to randomize the value per simulation rather than keeping it fixed for the whole game. This suggests that MCTS might benefit from diversifying the search not only by using strategies like UCT and MAST that try to balance exploration and exploitation of moves, but also by changing the strategy itself while searching. Randomizing parameter values allows to explore not only the search space for the moves, but also the search space of the strategies, diversifying the search even more. The fact that the search is further diversified by constantly changing the parameter values while searching for the best ones, might also be one of the reasons behind the success of on-line parameter tuning in some games.

Results have also shown that the effect of parameter randomization depends on multiple factors. Whether it is beneficial to randomize parameter values per simulation rather than keeping them fixed or rather than tuning them on-line does not only depend on the game. It also depends on which and how many parameters are being randomized and on the type of opponent the agent is facing. Future work could look into devising a mechanism to automatically detect for each new game if it is worth using parameter randomization, and if so, which parameters are worth randomizing. This mechanism could further be extended to detect on-line if, for a given game, it is better to randomize or tune the parameters.

It may also be concluded that, although not always the best solution for all games, randomization is still beneficial in GGP for games where the fixed settings that are generally optimal are actually performing poorly. Moreover, parameter randomization might be a valid alternative when the number of parameter to tune is high and time settings are limited, because the problem of tuning them on-line becomes too hard due to the increased combinatorial complexity.

REFERENCES

[1] M. Genesereth and M. Thielscher, "General game playing," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 2, pp. 1–229, 2014.

[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] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proceedings of the 5th International Conference on Computer and Games*, ser. Lecture Notes in Computer Science (LNCS), J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Heidelberg, Germany: Springer-Verlag, 2007, pp. 72–83.

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

[5] C. F. Sironi, J. Liu, and M. H. M. Winands, "Self-adaptive Monte-Carlo tree search in general game playing," *IEEE Transactions on Games*, 2019, in press.

[6] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The n-tuple bandit evolutionary algorithm for game agent optimisation," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 221–229.

[7] 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.

[8] C. F. Sironi and M. H. M. Winands, "Analysis of self-adaptive monte carlo tree search in general video game playing," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 397–400.

[9] D. Beal and M. C. Smith, "Random evaluations in chess," *ICGA Journal*, vol. 17, no. 1, pp. 3–9, 1994.

[10] B. Bošanskỳ, V. Lisỳ, M. Lanctot, J. Čermák, and M. H. M. Winands, "Algorithms for computing strategies in two-player simultaneous move games," *Artificial Intelligence*, vol. 237, pp. 1–40, 2016.

[11] K.-H. Chen, "Dynamic randomization and domain knowledge in monte-carlo tree search for go knowledge-based systems," *Knowledge-Based Systems*, vol. 34, pp. 21–25, 2012.

[12] 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.

[13] 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.

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

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

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