Journal of Artificial Intelligence Research 72 (2021) 717-757 Submitted 06/2021; published 11/2021

Analysis of the Impact of Randomization of Search-Control Parameters in Monte-Carlo Tree Search

Chiara F. Sironi C.SIRONI@MAASTRICHTUNIVERSITY.NL
Mark H. M. Winands M.WINANDS@MAASTRICHTUNIVERSITY.NL
Games Group,
Department of Data Science and Knowledge Engineering,
Maastricht University, P.O. Box 616, 6200 MD,
Maastricht, The Netherlands

Abstract

Monte-Carlo Tree Search (MCTS) has been applied successfully in many domains, including games. However, its performance is not uniform on all domains, and it also depends on how parameters that control the search are set. Parameter values that are optimal for a task might be sub-optimal for another. In a domain that tackles many games with different characteristics, like general game playing (GGP), selecting appropriate parameter settings is not a trivial task. Games are unknown to the player, thus, finding optimal parameters for a given game in advance is not feasible. Previous work has looked into tuning parameter values online, while the game is being played, showing some promising results. This tuning approach looks for optimal parameter values, balancing exploitation of values that performed well so far in the search and exploration of less sampled values. Continuously changing parameter values while performing the search, combined also with exploration of multiple values, introduces some randomization in the process. In addition, previous research indicates that adding randomization to certain components of MCTS might increase the diversification of the search and improve the performance. Therefore, this article investigates the effect of randomly selecting values for MCTS search-control parameters online among predefined sets of reasonable values. For the GGP domain, this article evaluates four different online parameter randomization strategies by comparing them with other methods to set parameter values: online parameter tuning, offline parameter tuning and sub-optimal parameter choices. Results on a set of 14 heterogeneous abstract games show that randomizing parameter values before each simulation has a positive effect on the search in some of the tested games, with respect to using fixed offline-tuned parameters. Moreover, results show a clear distinction between games for which online parameter tuning works best and games for which online randomization works best. In addition, the overall performance of online parameter randomization is closer to the one of online parameter turning than the one of sub-optimal parameter values, showing that online randomization is a reasonable parameter selection strategy. When analyzing the structure of the search trees generated by agents that use the different parameters selection strategies, it is clear that randomization causes MCTS to become more explorative, which is helpful for alignment games that present many winning paths in their trees. Online parameter tuning, instead, seems more suitable for games that present narrow winning paths and many losing paths.

1. Introduction

Monte-Carlo Tree Search (MCTS) is a best-first algorithm that has been applied successfully to many domains (games) (Browne et al., 2012). Previous research has shown that MCTS

does not perform equally well on all games (Finnsson & Björnsson, 2011; Ramanujan, Sabharwal, & Selman, 2010). Its performance is influenced by various game characteristics, such as the depth and branching factor of the tree, the level of game progression, the presence of many terminal states and shallow traps in the tree, or the presence of optimistic moves (i.e., moves that lead to a (definitive) advantage for the player, if the opponent does not realize that there is an immediate counter-move). An environment where this aspect is particularly relevant is general game playing[1] (GGP), which tackles many games with different characteristics, assuming that their rules are not known in advance. Therefore, there might be games that have challenging characteristics for MCTS.

Moreover, the search settings used for MCTS in GGP are usually selected to perform generally well on a wide set of games, but might not perform optimally for all of them. Setting search parameters in GGP is not a trivial task. They could be set in advance to values that the programmer deems sensible, but not knowing which games the algorithm is going to address, they might be generally *sub-optimal*. They could also be *tuned offline* on a predefined set of games to find the values that perform on average best on such set (Finnsson, 2012). However, there might still be games for which the chosen values are not optimal. In addition, offline parameter tuning might be computationally expensive when there is a large number of parameters to tune, especially if they are tuned in combination. An approach to select parameter values that tries to overcome these limitations consists in *tuning parameters online* for each new game that the algorithm has to tackle (Sironi, Liu, & Winands, 2020). However, the limited search budget usually available in GGP might not be sufficient to find optimal values, especially when tuning a large number of parameters.

Previous work has shown that the performance of search algorithms can be influenced by adding randomization to certain components of the search. Randomization might increase the diversification of the search, which might be beneficial in certain domains. Different approaches have been proposed to add randomization to the search, such as adding a random term to the state evaluation function used by the search algorithm (Beal & Smith, 1994), or adding some randomness when choosing actions during the selection or the play-out phase of MCTS (Chen, 2012; Bošanskỳ, Lisỳ, Lanctot, Čermák, & Winands, 2016). In GGP, trying to diversify the search adding some randomization in the online selection of parameter values might be a good strategy for some games. It might be beneficial in games that have characteristics that might hinder the performance of MCTS. This is also suggested by the results presented by Sironi et al. (2020) for online parameter tuning in the Stanford General Game Playing (Stanford GGP) framework, which tackles abstract games (Genesereth & Thielscher, 2014). In their work, the authors present different strategies to tune MCTS parameters for each new game being played, while playing the game for the first time. Their results showed that online parameter tuning is beneficial for MCTS, especially if the number of tuned parameters is low and if the agents can perform a sufficient number of simulations to evaluate the combinations of values. Part of the explanation for the performance of online parameter tuning on some games might be that repeatedly changing parameter settings is adding a random component to the search, helping the agents diversify the search process. Therefore, they explore more parts of the tree, which would not be explored much when

1. In this article, the term *general game playing* with lower case initials is used to indicate the broad field that encompasses general game playing for any type of game, such as abstract games (General Game Playing), video games (General Video Game Playing), etc.

parameter values are fixed. There might be games for which this diversification is beneficial on its own. A way to verify whether this is true would be trying to randomize parameter values online.

Other results that support the relevance of research in search-control parameter randomization in a general game-playing environment are the ones obtained in General Video Game Playing (GVGP) (Perez-Liebana, Liu, Khalifa, Gaina, Togelius, & Lucas, 2019). Search-control parameter randomization has already been tested for MCTS agents in GVGP (Sironi & Winands, 2018a). In this environment, where time settings are shorter than in the Stanford GGP framework, online parameter tuning is shown to be comparable to randomizing parameters before each MCTS simulation. In addition, online 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.

This article extends on the authors' previous work (Sironi & Winands, 2019). The main contribution of this article consist in giving further insights on the effect that randomizing search-control parameters online has on MCTS. Moreover, it analyzes how online parameter randomization compares with online parameter tuning, and how both these online strategies compare with selecting parameter values offline, either manually, with the risk of incurring in a sub-optimal choice, or by offline tuning. The purpose for this analysis is to verify whether there are some games for which parameter randomization might be beneficial, and how the performance of the different parameter selection methods is related to certain game characteristics. First, four different search-control parameter randomization strategies are evaluated, to verify how they influence the performance of the search, and whether they bring any benefit for any of the tested games. Such strategies randomize parameter values once per game run, once per turn, once per simulation and once per visited state, respectively. In addition to the authors' previous work (Sironi & Winands, 2019), this article compares the results of the experiments with results reported in previous publications, confirming their conclusions and giving more insights both on parameter randomization and online parameter tuning. Moreover, it presents an analysis of the structure of the search trees built by agents that use offline-tuned, online-tuned or randomized parameters. This will verify if there is a relation between the way each agent explores the tree and the characteristics of the games on which each agent performs best.

The article is organized as follows. First, Section 2 discusses related work. Next, background knowledge on MCTS and online parameter tuning is given in Section 3. The parameter randomization strategies are described in Section 4. Subsequently, Section 5 reports the results of the performed experiments, the analysis of the search tree structure and of the selection of parameter values in the domain of abstract games. Section 6 discusses the most relevant results obtained by the application of parameter randomization and online parameter tuning to the real-time environment of GVGP arcade-style video games. Finally, in Section 7 the conclusion is given and future work is discussed.

2. 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 and Smith (1994), which shows the effects of using random numbers as intermediate state evaluations

when performing minimax search in Chess. An agent using only random evaluations for intermediate states is shown to outperform the same agent that uses 0 as default evaluation 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, biasing the search toward states where the player has more options to choose from.

Furthermore, Chen (2012) proposes 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. These randomization techniques diversify the sampling of the actions and are shown to improve the performance of the MCTS agent for different search budgets and sizes of the Go board.

Finally, Bošanskỳ et al. (2016) propose the use of a random tie-breaking rule for an MCTS agent that is selecting actions using the Upper Confidence Bound (UCB) value. More precisely, the agent selects for each role a random action among those that have the UCB value within a predefined small offset from the highest UCB 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 instead. This happens because with a tie-breaking rule the agent is mixing the strategies that it uses to sample the actions, instead of sampling always according to the same strategy.

3. Background

This section provides background on MCTS (Subsection 3.1) and on online tuning of search-control parameters for MCTS (Subsection 3.2).

3.1 Monte-Carlo Tree Search

MCTS (Kocsis & Szepesvári, 2006; Coulom, 2007) is a simulation based-search strategy that incrementally builds a tree representation of the state space of a game. Figure 1 shows how MCTS works. It performs game simulations by repeating the following four phases until a search budget expires.

1. **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.

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

3. **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.

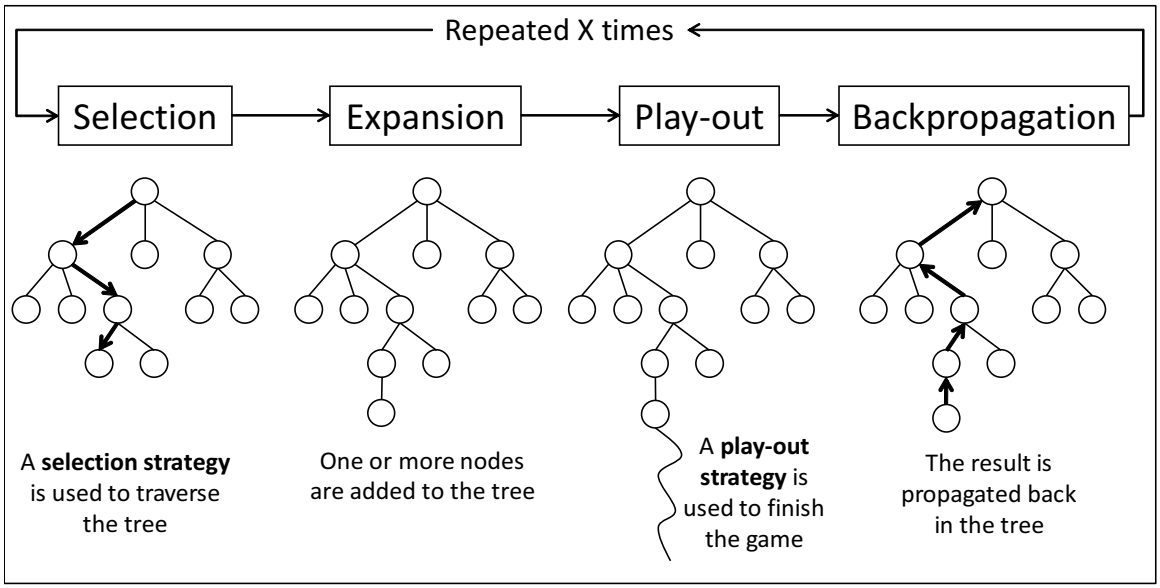


Figure 1: Outline of Monte-Carlo Tree Search (inspired by Chaslot et al., 2008).

4. **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. Commonly, the move that is returned is either the one with the highest number of visits or the one with the highest average payoff.

The standard MCTS selection strategy is UCT (Upper Confidence bounds applied to Trees, (Kocsis & Szepesvári, 2006)). Given a state s, UCT chooses the action a^* for a player using the UCB1 sampling strategy (Auer, Cesa-Bianchi, & Fischer, 2002) as follows:

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

Here, $A(s)$ 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 (Gelly & Silver, 2007), and its generalization, GRAVE (Cazenave, 2015). GRAVE selects a move using the UCB1 formula (Equation 1), where the term $Q(s,a)$ is substituted by the following quantity:

$$\beta(s) \times Q(s,a) + (1 - \beta(s)) \times AMAF(s',a) \ , \tag{2}$$

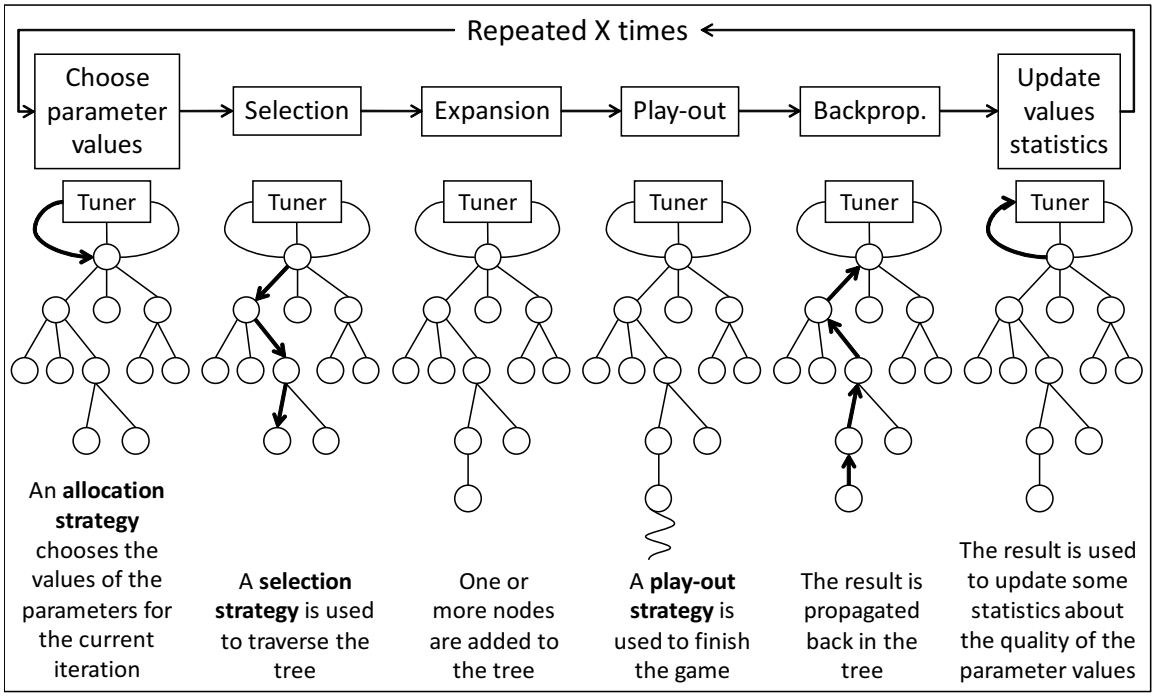


Figure 2: Interleaving online tuning with MCTS (inspired by Chaslot et al., 2008).

with

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

Here, $AMAF(s', a)$ is part of the All-Moves-As-First (AMAF) statistic and corresponds to the average result of all the simulations in which move a was selected at any moment after state s' was visited. The state s' is the ancestor of s that has been visited at least *Ref* times, and *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) (Björnsson & Finnsson, 2009). 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.

3.2 Online Parameter Tuning

As already mentioned, when applying MCTS in GGP it is hard to find a single configuration of the algorithm and its enhancements that performs optimally on every possible game. Moreover, the agent has to assume that it has never played the game at hand before. Thus, it cannot use prior knowledge to decide which parameter configuration might work best. Usually, search-control parameters are set to values that perform generally well, but might

still be sub-optimal for some games. For this reason, previous work proposed to tune the parameters for each game online (Sironi et al., 2020) (i.e., while the current instance of the game is being played). To implement online parameter tuning for MCTS, the standard algorithm iteration is modified by adding two extra phases, as shown in Figure 2. Therefore, the search for each MCTS simulation is performed as follows:

1. 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. This choice is based on statistics about the performance of parameter values collected in previous iterations.

2. The standard MCTS phases are performed (selection, expansion, play-out and back-propagation), controlled by the parameter values chosen in the previous phase.

3. 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 approaches can be used as allocation strategy. For example, popular hyper-parameter optimization algorithms like SMAC (Hutter, Hoos, & Leyton-Brown, 2011) or PIAC (Belkhir, Dréo, Savéant, & Schoenauer, 2017) can be used to select which combinations to evaluate in each MCTS iteration. However, they would require first some modification to be applied in the online setting of MCTS, where a single simulation is considered as an instance of the problem with its own reward. They might also be too computationally expensive to be used in GGP, where only a few seconds per move are available to search for a good combination of parameters, and only a limited number of parameter combinations can be sampled. The allocation strategy considered in this article to compare online parameter tuning with parameter randomization is the one based on the N-Tuple Bandit Evolutionary Algorithm (NTBEA) (Lucas, Liu, & Perez-Liebana, 2018). The main reason for this choice is that NTBEA has already been evaluated in previous work on online parameter tuning for MCTS, and it was the best performing one among the allocation strategies evaluated in GGP (Sironi & Winands, 2018b; Sironi et al., 2020).

The NTBEA allocation strategy uses an *evolutionary algorithm*, which considers each combination of parameters as an individual and each single parameter as a gene (Lucas et al., 2018). The evolutionary algorithm starts with a randomly generated combination of parameter values and evolves it over time using the statistics 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 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 the statistics collected in *LModel*.

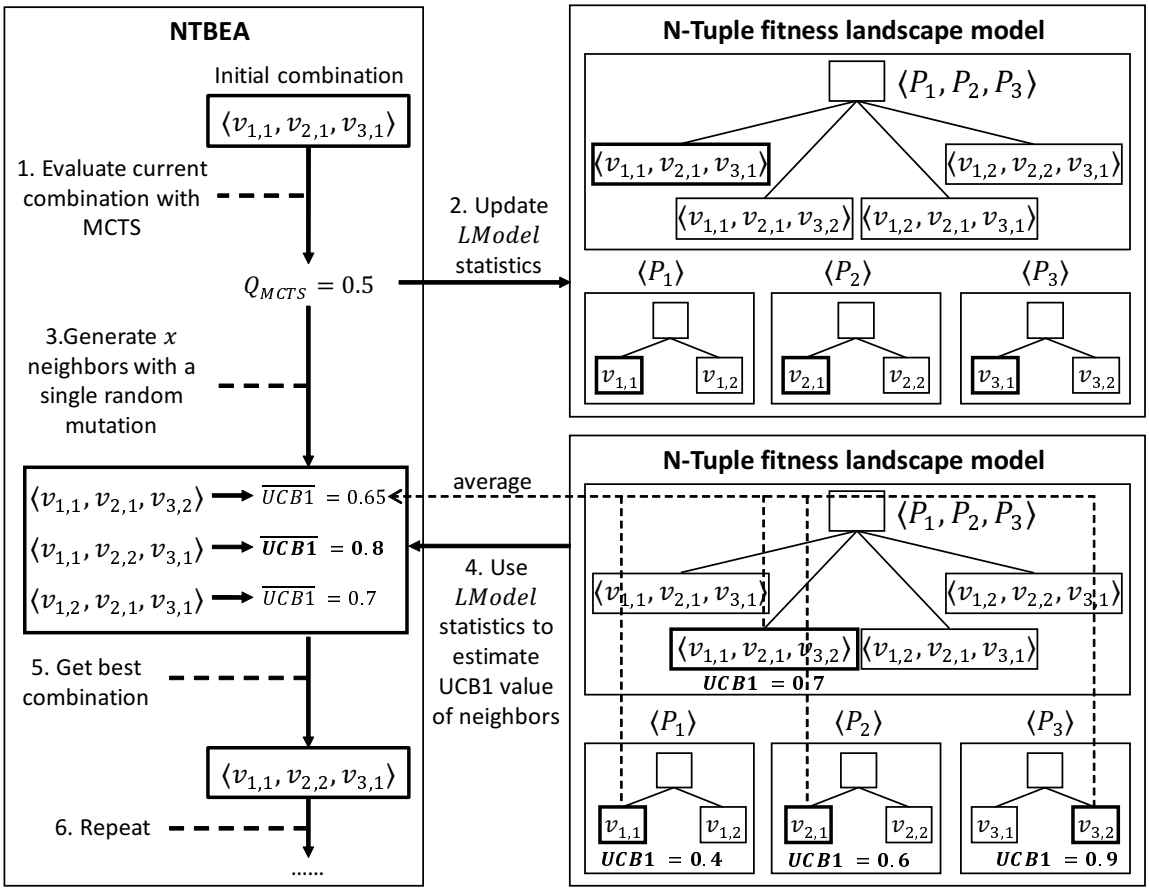


Figure 3: Overview of the execution of NTBEA.

5. Set the neighbor with the highest evaluation as the current combination.

To evaluate parameter combinations, LModel considers n-tuples of parameters formed by subsets of all the available d parameters (including the d 1-tuples, and the single d-tuple). For each tuple of parameters, it considers a Multi-Armed Bandit (MAB) where each arm corresponds to one of the possible assignment of values for the tuple. The evaluation of a parameter combination is computed using such MABs. For each MAB, NTBEA computes the UCB1 value of the arm that corresponds to the value assignment of the associated tuple in the considered combination. The average of the UCB1 values computed over all the MABs is used as the evaluation of the combination. This evaluation has the advantage of being faster than an entire MCTS simulation to evaluate a parameter combination. Therefore, it is used by the NTBEA allocation strategy to speed up the evolutionary process, while balancing exploration and exploitation of the various parameter combinations.

Figure 3 shows an example of the execution of NTBEA when three parameters are being tuned, P_1, P_2 and P_3, each of which has two possible values, $v_{i,1}$ and $v_{i,2}$, with $i \in \{1, 2, 3\}$. Note that not all parameter tuples might be used by LModel. In this example, LModel is using only the three 1-tuples (i.e., $\langle P_1\rangle$, $\langle P_2\rangle$ and $\langle P_3\rangle$) and the single 3-tuple

(i.e., $\langle P_1, P_2, P_3 \rangle$, but none of the possible 2-tuples (i.e., $\langle P_1, P_2 \rangle$, $\langle P_1, P_3 \rangle$ and $\langle P_2, P_3 \rangle$). More details on the NTBEA allocation strategy are given by Sironi et al. (2020).

4. Search-Control Parameter Randomization

In order to implement search-control parameter randomization for a tree search algorithm, the following three steps are performed:

- Identify a finite set of d parameters, $P = \{P_1, ..., P_d\}$, that will be randomized.
- For each parameter $P_i \in P$, define the finite set of m_i different values that the parameter can assume, $\mathcal{V}_i = \{v_{i,1}, ..., v_{i,m_i}\}$.
- Design a *randomization strategy* that decides when and how to randomize the parameters.

Note that also a continuous domain for the parameters could be considered. However, results presented by Sironi et al. (2020) for online parameter tuning showed that considering a continuous domain was not adding any benefit to the performance. Therefore, this article focuses only on a discrete domain also for parameter randomization.

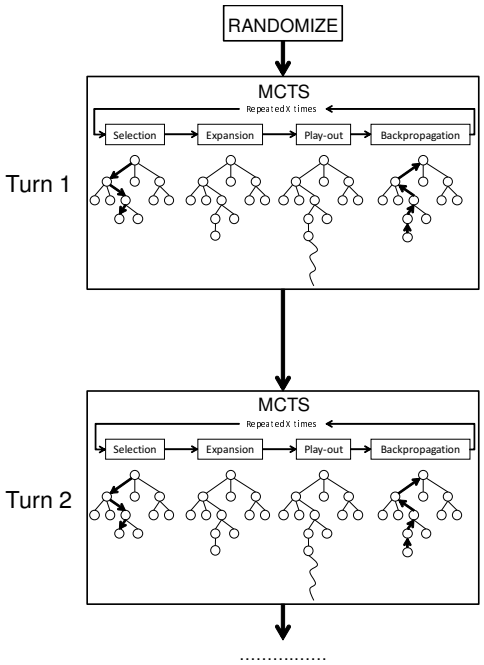
Four different strategies to randomize search-control parameters are considered: *per game run*, *per turn*, *per simulation* and *per state*. All of them randomize the values of the selected parameters for each role in the game separately. This choice has been made to be consistent with the online parameter-tuning mechanism with which parameter randomization is compared, which was designed to tune parameters independently for each role. The randomization strategies are described below and an overview of when randomization takes place for each of them is given in Figure 4.

Per game 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 combinations of parameters for each role are then kept fixed for the search performed in each turn, until the run of the game is over.

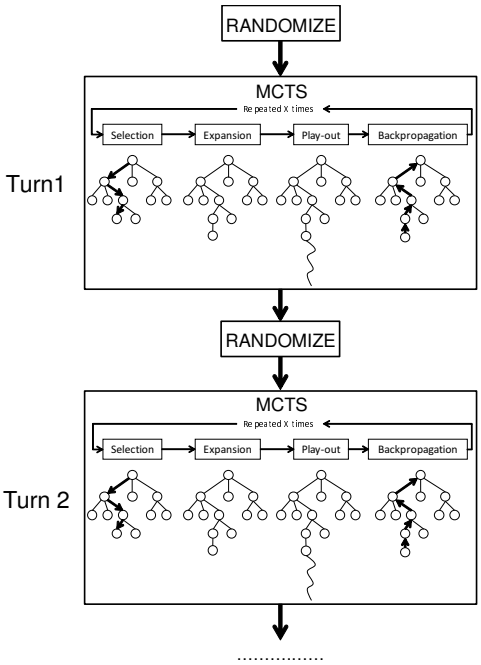
Per turn: for each role in the game, this strategy sets a random combination of feasible values for the considered parameters before starting the search for each game turn.

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

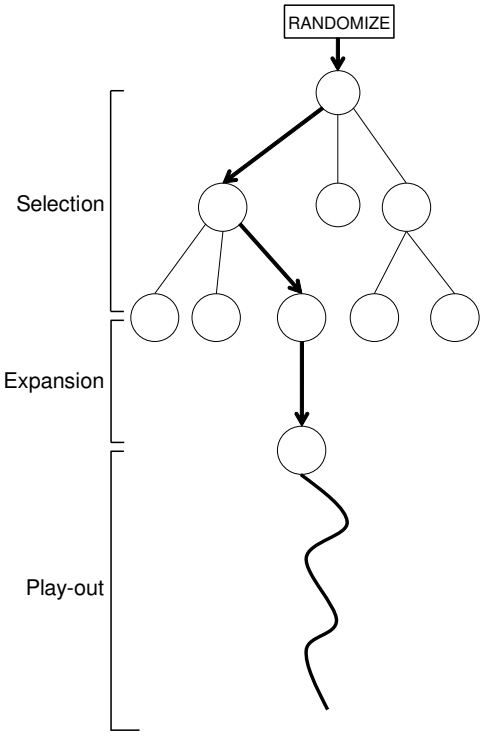
Per state: every time a state is visited during a simulation, both during the selection and the play-out phase of MCTS, this strategy randomizes for each role the values of the parameters used to perform the search in the state. This strategy is similar to the one proposed by Chen (2012), although he assigns the same random parameter values to all the roles. As previously mentioned, this article considers parameter randomization for all roles independently to be consistent with the online parameter-tuning mechanism analyzed by Sironi et al. (2020).



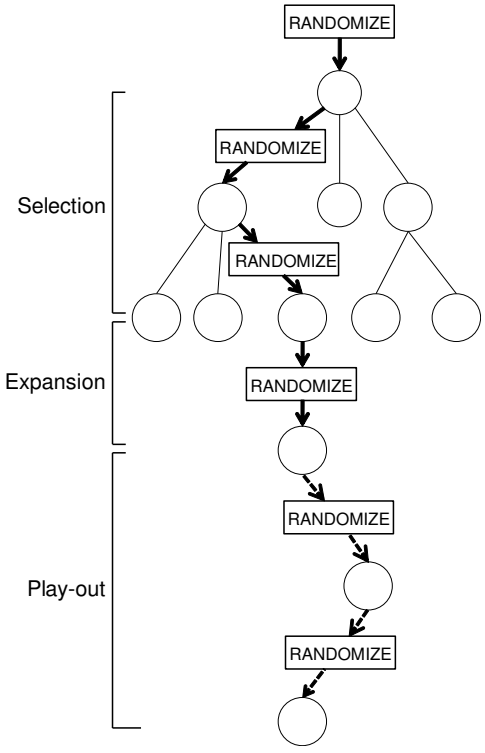
(a) Randomization per game run.



(b) Randomization per turn.



(c) Randomization per simulation.



(d) Randomization per state.

Figure 4: Randomization strategies.

Param.	Offline-tuned value	Sub-opt. value	Set of feasible values
C	0.2	0.9	$\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$
ϵ	0.4	0.0	$\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
K	250	∞	$\{0, 10, 50, 100, 250, 500, 750, 1\,000, 2\,000, \infty\}$
Ref	50	∞	$\{0, 50, 100, 250, 500, 1\,000, 10\,000, \infty\}$
VO	0.01	0.025	$\{0.001, 0.005, 0.01, 0.015, 0.02, 0.025\}$
T	0	200	$\{0, 5, 10, 20, 30, 40, 50, 100, 200, \infty\}$

Table 1: Parameters considered in the experiments with their offline-tuned value, sub-optimal value and sets of feasible values.

5. Empirical Evaluation on Abstract Games

This section presents an analysis of parameter randomization for MCTS in GGP, by performing multiple series of experiments. First, Subsection 5.1 describes the experimental setup. Subsequently, results obtained by performing the experiments on the abstract games of the Stanford GGP framework are given in Subsections 5.2, 5.3, 5.4 and 5.5. Next, an analysis of the search trees built by different agents is presented in Subsection 5.7. Finally, Subsection 5.8 discusses the importance of an accurate selection of feasible parameter values when tuning or randomizing parameters online.

5.1 Setup

The experiments presented in this article on the Stanford GGP framework use as baseline an MCTS agent with the GRAVE selection strategy and the MAST play-out strategy. The following six parameters are considered to be randomized or tuned online:

- $\boldsymbol{C}$**:** exploration constant for the UCT selection.
- $\boldsymbol{\epsilon}$**:** probability of selecting a random action with MAST.
- $\boldsymbol{K}$**:** *equivalence parameter* of GRAVE.
- $\boldsymbol{Ref}$**:** visit threshold used by GRAVE.
- $\boldsymbol{VO}$**:** when using the GRAVE selection in state s, select a random move a among the ones with value $V(a,s) \in [max_a(V(a,s)) - VO, max_a(V(a,s))]$.
- $\boldsymbol{T}$**:** during selection at state s, if $N(s) < T$, select the next action using the play-out strategy instead of the selection strategy.

When randomizing/tuning two parameters, the agents consider K and Ref. When randomizing/tuning four parameters, the agents consider K, Ref, C and ϵ. When randomizing/tuning six parameters, the agents consider K, Ref, C, ϵ, VO and T.

For each of these parameters Table 1 reports the offline-tuned and sub-optimal values, and their discrete sets of feasible values that are used when randomizing or tuning. The offline-tuned values have been obtained by tuning parameters individually and not in combination. The sub-optimal parameter values are used in some of the series of experiments to

give an indication of the performance of an agent for which values that could be reasonable, are actually not optimal for the game (i.e., the worst-case scenario). Note that the sub-optimal values are also included in the corresponding sets of feasible values. Therefore, the online randomizing and tuning strategies have the possibility to choose them as well. When reporting the results, the agents are identified by the type of parameters they are using. The baseline agent with offline-tuned values is indicated as OFFLINE, while the one with sub-optimal values as SUB-OPT. When randomizing or tuning the parameters online, the agents are identified with the type of randomization or tuning strategy used, respectively. The four randomization strategies are identified, in order, as GAME-RND, TURN-RND, SIM-RND and STATE-RND, while the online tuning strategy is identified as NTBEA.

The settings for NTBEA are the same used by Sironi et al. (2020). 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.

In one of the series of experiments, the last available version of CadiaPlayer[2] (Finnsson, 2012) is used as a benchmark to compare the performance of the best parameter randomization strategy with the performance of online tuning and of the fixed parameters.

For comparison with the work presented by Sironi et al. (2020), the experiments are performed on the same set of 14 games (Schreiber, 2017): 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. For the experiments with CadiaPlayer, four games (Chinese Checkers with 3 players, TTCC4 with 2 and 3 players, and Reversi) are excluded because this agent encountered errors while playing them. 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 number 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 configurations are run the same number of times until each agent type has played at least 500 games in total. For each agent, start- and play-clock are set to $1s$, except for CadiaPlayer, which uses $10s$ start- and play-clock in order to reach a number of simulations similar to the other agents.

Different series of experiments have been performed, one that compares all the randomization strategies, one that compares randomization of single parameters with offline-tuned parameter values, one that compares parameter randomization with online parameter tuning directly, one that compares parameter randomization and online parameter tuning with offline-tuned parameter values, both directly and against different types of opponents (i.e., an agent with sub-optimal values and CadiaPlayer), and one that compares the performance of randomization and online parameter tuning for different sets of feasible values for the parameters. Experimental results always report the average win percentage of one of the two involved agent types with a 95%-confidence interval. The average win percentage of an agent type for a game is computed by assigning 1 point to the agent type that achieved the highest score and 0 to the other. If they both achieve the same score, they are given 0.5

2. Version of 18-11-2012. Downloaded from `http://cadia.ru.is/wiki/public:cadiaplayer:main`

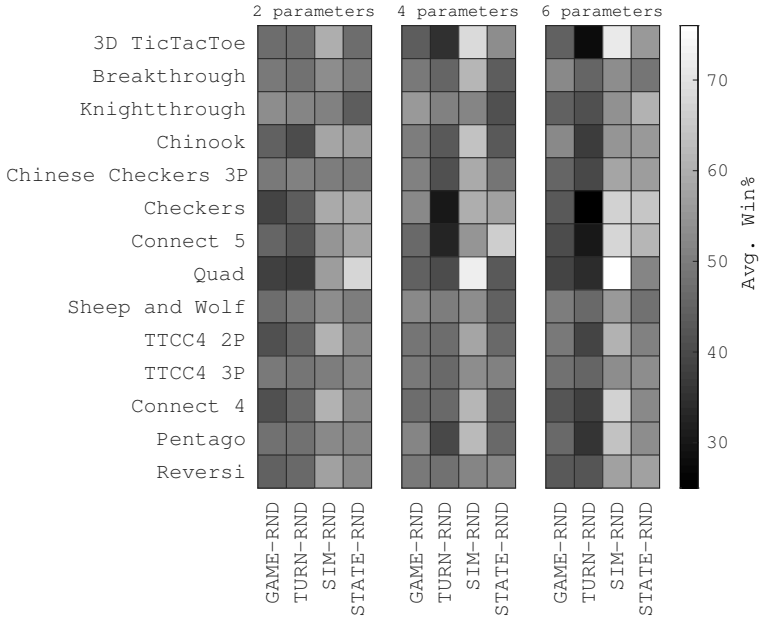


Figure 5: Performance of parameter randomization strategies against each other for different numbers of randomized parameters.

points each. If relevant, bold results indicate the agent type with the highest win rate for the corresponding game and number of randomized/tuned parameters. Experiments were performed on a Linux server consisting of 64 AMD Opteron 6274 2.2-GHz cores, except the ones presented in Tables 4 and 5, which were performed on a Linux server consisting of 48 AMD Opteron 6344 2.6-GHz cores.

5.2 Evaluation of Parameter Randomization Strategies

These series of experiments evaluate the randomization strategies by matching the agents that implement them against the agent with offline-tuned parameters, and by directly matching them against each other to see which one has the best performance. Table 2 shows the results obtained by the randomizing agents against the agent with offline-tuned parameter values. For different number of parameters, the table reports the average win percentage of each of the randomizing agents over each game. The agents consider K and *Ref* when randomizing two parameters, K, *Ref*, C and ϵ when randomizing four parameters and K, *Ref*, C, ϵ, VO and T when randomizing six.

Looking at the overall performance, it is not surprising that none of the randomizing agents is better than the agent that uses offline-tuned values. The offline-tuned values have been selected because they perform generally well on many games, while the selection of a random combination of values is more likely to find a sub-optimal setting. However, it is interesting to look at the performance of the randomizing agents on individual games. In Quad, for example, randomization per simulation performs significantly better than the offline-tuned values, for each number of randomized parameters. Significantly better results are obtained by SIM-RND also in Chinook for two parameters and in Connect 4 for four.

Game	2 parameters			
	GAME-RND	**TURN-RND**	**SIM-RND**	**STATE-RND**
3D TicTacToe	36.8(±3.97)	35.9(±4.00)	**46.4**(±4.12)	35.4(±3.94)
Breakthrough	34.8(±4.18)	36.6(±4.23)	**40.4**(±4.31)	35.0(±4.19)
Knightthrough	41.8(±4.33)	43.2(±4.35)	**44.2**(±4.36)	43.4(±4.35)
Chinook	47.3(±4.06)	47.6(±4.02)	**61.8**(±3.99)	57.9(±4.08)
Chinese Checkers 3P	44.4(±4.34)	**47.8**(±4.37)	45.6(±4.35)	47.4(±4.36)
Checkers	36.9(±4.01)	35.3(±3.97)	**48.8**(±4.08)	47.0(±4.11)
Connect 5	36.3(±3.14)	35.6(±3.11)	43.9(±3.12)	**45.2**(±3.20)
Quad	46.8(±4.12)	48.4(±4.24)	65.0(±3.93)	**67.4**(±3.80)
Sheep and Wolf	46.6(±4.38)	48.4(±4.38)	**52.0**(±4.38)	49.4(±4.39)
TTCC4 2P	35.4(±4.06)	40.1(±4.16)	**49.5**(±4.27)	46.4(±4.21)
TTCC4 3P	44.6(±4.25)	46.4(±4.24)	**48.7**(±4.26)	48.6(±4.24)
Connect 4	40.1(±4.10)	40.5(±4.07)	**50.9**(±4.17)	45.5(±4.24)
Pentago	42.6(±4.20)	45.5(±4.19)	**53.7**(±4.19)	50.9(±4.20)
Reversi	38.4(±4.18)	35.5(±4.13)	42.8(±4.29)	**45.5**(±4.28)
Avg. Win%	40.9(±1.10)	41.9(±1.11)	**49.6**(±1.12)	47.5(±1.12)

Game	4 parameters			
	GAME-RND	**TURN-RND**	**SIM-RND**	**STATE-RND**
3D TicTacToe	31.7(±3.85)	18.2(±3.22)	**39.4**(±4.00)	28.6(±3.66)
Breakthrough	**15.6**(±3.18)	10.0(±2.63)	11.0(±2.75)	9.0(±2.51)
Knightthrough	**22.2**(±3.65)	20.8(±3.56)	20.6(±3.55)	15.8(±3.20)
Chinook	22.5(±3.40)	16.4(±2.98)	**23.6**(±3.61)	18.9(±3.28)
Chinese Checkers 3P	**35.7**(±4.19)	29.2(±3.97)	34.7(±4.16)	31.3(±4.05)
Checkers	18.5(±3.19)	9.0(±2.35)	17.9(±3.17)	**18.6**(±3.20)
Connect 5	30.7(±3.15)	25.8(±2.96)	36.5(±3.22)	**41.8**(±3.49)
Quad	47.3(±4.20)	46.7(±4.18)	**72.8**(±3.71)	53.6(±4.23)
Sheep and Wolf	48.6(±4.39)	44.2(±4.36)	**49.8**(±4.39)	42.0(±4.33)
TTCC4 2P	**23.5**(±3.62)	14.5(±3.05)	20.6(±3.47)	17.2(±3.28)
TTCC4 3P	41.9(±4.23)	38.4(±4.16)	**46.1**(±4.28)	45.8(±4.28)
Connect 4	46.3(±4.17)	45.5(±4.22)	**55.4**(±4.13)	42.6(±4.16)
Pentago	40.2(±4.17)	32.4(±4.01)	**50.2**(±4.22)	30.0(±3.90)
Reversi	25.7(±3.75)	31.0(±4.01)	31.0(±3.99)	**31.3**(±3.99)
Avg. Win%	32.2(±1.05)	27.3(±1.01)	**36.4**(±1.08)	30.5(±1.04)

Game	6 parameters			
	GAME-RND	**TURN-RND**	**SIM-RND**	**STATE-RND**
3D TicTacToe	26.6(±3.67)	15.2(±3.03)	**40.6**(±4.01)	27.8(±3.55)
Breakthrough	**13.2**(±2.97)	6.2(±2.12)	9.4(±2.56)	6.8(±2.21)
Knightthrough	19.4(±3.47)	11.4(±2.79)	19.8(±3.50)	**23.0**(±3.69)
Chinook	**27.2**(±3.71)	14.4(±2.93)	19.8(±3.34)	25.9(±3.72)
Chinese Checkers 3P	30.6(±4.03)	25.6(±3.81)	**33.7**(±4.13)	29.4(±3.98)
Checkers	14.4(±2.91)	7.8(±2.24)	**20.4**(±3.29)	**20.4**(±3.37)
Connect 5	30.6(±3.02)	19.1(±2.77)	**45.7**(±3.21)	38.5(±3.38)
Quad	40.6(±4.14)	34.8(±4.00)	**72.4**(±3.67)	54.1(±4.25)
Sheep and Wolf	43.6(±4.35)	41.2(±4.32)	**50.0**(±4.39)	42.8(±4.34)
TTCC4 2P	**20.2**(±3.46)	12.0(±2.84)	19.0(±3.40)	15.6(±3.17)
TTCC4 3P	40.4(±4.20)	37.1(±4.13)	40.8(±4.23)	**41.6**(±4.23)
Connect 4	33.5(±3.99)	33.4(±3.98)	**48.0**(±4.23)	40.5(±4.14)
Pentago	31.2(±3.94)	25.9(±3.75)	**42.6**(±4.13)	29.3(±3.85)
Reversi	27.1(±3.85)	21.0(±3.52)	28.7(±3.92)	**34.7**(±4.10)
Avg. Win%	28.5(±1.02)	21.8(±0.94)	**35.1**(±1.07)	30.7(±1.04)

Table 2: Win percentage of GAME-RND, TURN-RND, SIM-RND and STATE-RND against OFFLINE.

Moreover, also STATE-RND with two randomized parameters is significantly better than the agent with offline-tuned values in Chinook and Quad. This suggest that there are some games that might benefit from parameter randomization. This could be because a diversified search manages to explore more interesting parts of the game tree that would not be explored otherwise. It could also be because, for such games, the offline-tuned parameter values are sub-optimal, and selecting random combinations of values increases the frequency of using optimal ones. These possibilities will be further explored in subsequent experiments. Results in the table also show that in general the performance of all the randomizing agents drops with the increase in the number of randomized parameters. This might indicate that the bigger the parameters search space becomes, the more the relative number of bad combinations increases. Thus, it becomes more likely for the agent to sample them.

Looking at the performance of the randomizing agents only, SIM-RND seems to be the best performing on average over all games, with STATE-RND being comparable only for two randomized parameters. For most of the games, independently of the number of parameters being considered, SIM-RND is never significantly inferior to the other agents. This is confirmed also by the results reported in Figure 5, that were obtained by performing a round-robin tournament with all the agents that use one of the parameter randomization strategies. Each grid in the figure corresponds to a different number of parameters being randomized, each row in the grid corresponds to a game, and each column to an agent. Results represent the average win percentage of the column agent against all other agents that randomize the same number of parameters. For most of the games, independently of the number of parameters being considered, the agent that randomizes parameters per simulation performs at least as well and sometimes better than other randomizing agents. This does not contradict the findings of Chen (2012). Although the author mentions that randomizing parameters per simulation did not perform as well as per state, the comparison was performed only in the game of Go. When tested on more games, there are still a few games where randomization per state performs significantly better (e.g., Quad for two parameters, Connect 5 for four, and Knightthrough for six).

It is interesting to notice that, for games for which parameter randomization sometimes improves the performance over offline-tuned values (i.e., Quad, Chinook, Connect 4), it seems better to randomize parameter values often. Randomization per simulation and per state perform usually better on these games than the other two strategies, probably because they cause more exploration within the same search. Moreover, among the two strategies, randomization per simulation might have an advantage because, while exploring more paths in the tree, it is still keeping each simulation consistent regarding its action selection.

5.3 Randomization per Simulation of Individual Parameters

These series of experiments further analyze parameter randomization, focusing on its effect on single parameters. Only the strategy that seemed to have the most influence on MCTS in the previous series of experiments is further tested: randomization per simulation. The purpose of these experiments is to verify how the performance of MCTS is influenced by each parameter, and check if and for which parameters randomization brings any benefits. Each series of experiments focuses on one single parameter, considering the agents that keep it fixed to its offline-tuned value and the agent that randomizes it per simulation (SIM-RND).

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

Table 3: Comparing all feasible values of C with value randomization of C per simulation.

All other parameters for SIM-RND are set to their offline-tuned values. All these agents are matched against each other in a round-robin tournament. For each game, the average win percentage of each agent against all other agents is reported in the results.

Tables 3, 4 and 5 show the results of such experiments for the parameters 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 (i.e., not significantly worse than the highest percentage). This can be observed in Quad, TTCC4 with 3 players and Pentago for the parameter C, in TTCC4 with 3 players and Pentago for the parameter ϵ and in more than half of the games for the parameter K. This means that, for these games, randomizing the parameter value is at least as good as using the best parameter value(s) for the game. This could be due to the fact that many of the considered values of the parameter are good for the game, thus the agent with randomization is still using good values for most of the simulations. However, randomization of the parameter K in Chinook, Quad and Connect 4 gives a significantly better performance than keeping the parameter fixed to any of the considered values. This suggests that randomization in itself might have a positive influence, and for these games changing this parameter value during the search brings a benefit.

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

Table 4: Comparing all feasible values of ϵ with value randomization of ϵ per simulation.

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 offline-tuned value (see Table 1). For some games, this happens because the offline-tuned value, despite being optimal 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, and Connect 4 for ϵ, and Chinook, Quad, TTCC4 with 2 players, Connect 4 and Pentago for K. For an environment like GGP, where optimal parameters cannot easily be set in advance for every game, this is an interesting result. It opens the possibility for parameter randomization to be a viable alternative to keeping fixed parameter values when those risk to be sub-optimal. In this way, the agent will alternate between using good and bad parameter values, which guarantees that at least part of the search is controlled by good parameter values.

	3D TicTacToe	Breakthrough	Knightthrough	Chinook	Chinese Checkers 3P
$K = 0$	47.8(±1.32)	46.2(±1.38)	57.8(±1.37)	40.1(±1.26)	51.1(±1.38)
$K = 10$	51.2(±1.33)	50.0(±1.39)	**59.9**(±1.36)	40.9(±1.27)	51.4(±1.38)
$K = 50$	53.2(±1.31)	53.6(±1.38)	56.8(±1.37)	45.3(±1.28)	**52.4**(±1.38)
$K = 100$	53.2(±1.31)	54.2(±1.38)	54.5(±1.38)	51.9(±1.28)	52.2(±1.38)
$K = 250$	56.4(±1.30)	56.8(±1.37)	53.8(±1.38)	55.8(±1.27)	52.0(±1.38)
$K = 500$	56.5(±1.30)	**56.9**(±1.37)	51.9(±1.39)	57.4(±1.26)	**52.4**(±1.38)
$K = 750$	**57.1**(±1.30)	55.2(±1.38)	51.0(±1.39)	57.2(±1.26)	52.0(±1.38)
$K = 1000$	56.5(±1.30)	54.0(±1.38)	50.5(±1.39)	55.5(±1.27)	51.9(±1.38)
$K = 2000$	53.6(±1.31)	52.7(±1.38)	50.4(±1.39)	54.6(±1.27)	49.7(±1.38)
$K = \infty$	8.9(±0.77)	16.7(±1.03)	16.6(±1.03)	29.3(±1.16)	34.4(±1.31)
SIM-RND$_K$	55.7(±1.30)	53.6(±1.38)	46.8(±1.38)	**62.0**(±1.24)	50.3(±1.38)
	Checkers	Connect 5	Quad	Sheep and Wolf	TTCC4 2P
$K = 0$	33.1(±1.26)	48.6(±1.06)	49.0(±1.31)	**51.6**(±1.39)	28.4(±1.23)
$K = 10$	41.5(±1.31)	52.6(±1.04)	50.2(±1.32)	49.9(±1.39)	33.0(±1.27)
$K = 50$	55.7(±1.31)	54.5(±1.01)	50.8(±1.31)	50.8(±1.39)	43.6(±1.34)
$K = 100$	61.7(±1.28)	55.0(±1.01)	50.8(±1.31)	50.4(±1.39)	50.7(±1.34)
$K = 250$	**66.6**(±1.24)	**58.0**(±0.98)	51.7(±1.31)	50.6(±1.39)	59.5(±1.31)
$K = 500$	63.5(±1.26)	57.8(±0.98)	53.4(±1.31)	50.2(±1.39)	63.2(±1.28)
$K = 750$	58.7(±1.30)	55.2(±0.98)	54.6(±1.31)	50.0(±1.39)	**63.9**(±1.28)
$K = 1000$	53.9(±1.31)	53.3(±0.99)	55.7(±1.30)	49.6(±1.39)	63.4(±1.28)
$K = 2000$	42.7(±1.31)	47.9(±0.99)	57.3(±1.30)	49.8(±1.39)	59.0(±1.32)
$K = \infty$	8.4(±0.73)	10.2(±0.62)	10.6(±0.83)	45.5(±1.38)	21.9(±1.13)
SIM-RND$_K$	64.3(±1.26)	56.9(±0.98)	**65.9**(±1.25)	**51.6**(±1.39)	63.5(±1.29)
	TTCC4 3P	Connect 4	Pentago	Reversi	
$K = 0$	49.9(±1.34)	41.6(±1.30)	37.4(±1.31)	53.4(±1.36)	
$K = 10$	51.5(±1.34)	44.2(±1.31)	38.9(±1.32)	58.3(±1.34)	
$K = 50$	**53.3**(±1.33)	48.4(±1.32)	44.6(±1.34)	61.1(±1.33)	
$K = 100$	50.6(±1.33)	50.0(±1.31)	48.7(±1.35)	**61.5**(±1.32)	
$K = 250$	51.7(±1.33)	54.3(±1.31)	55.7(±1.33)	57.6(±1.35)	
$K = 500$	52.2(±1.33)	55.9(±1.31)	58.0(±1.32)	52.0(±1.36)	
$K = 750$	51.1(±1.33)	56.8(±1.31)	60.1(±1.30)	48.9(±1.36)	
$K = 1000$	50.9(±1.33)	57.1(±1.31)	60.5(±1.31)	44.0(±1.36)	
$K = 2000$	49.2(±1.34)	56.7(±1.30)	60.2(±1.31)	38.7(±1.33)	
$K = \infty$	39.1(±1.31)	25.3(±1.15)	23.9(±1.15)	19.5(±1.08)	
SIM-RND$_K$	50.4(±1.34)	**59.8**(±1.30)	**62.0**(±1.30)	55.0(±1.36)	

Table 5: Comparing all feasible values of K with value randomization of K per simulation.

It is also interesting to notice that part of the results presented in this subsection can be compared to some of the results presented in previous works of the authors to verify their consistency, and, in some cases, give more insights. For instance, results in Table 4 seem to confirm that combining the GRAVE selection strategy with the MAST play-out strategy has in general a positive effect on the search, as concluded by Sironi and Winands (2016). Setting $\epsilon = 1.0$ for MAST means that the MAST score has no influence on the search and the agent is using a random play-out strategy. As can be seen from the table, the agent that uses random play-outs is one of the worst-performing agents for most of the games. Using MAST with intermediate values for ϵ usually increases the performance.

Another comparison with the work of Sironi and Winands (2016) can be performed for the agents that use $K = 0$ and $K = 250$ in Table 5. Setting $K = 0$ means deactivating the GRAVE strategy, thus, this agent is using only the UCT selection strategy combined with the MAST play-out strategy. The agent with $K = 250$, instead, is using the GRAVE selection strategy combined with MAST. These two types of agents have been compared also by Sironi and Winands (2016), where they are identified as $\text{P}_{\text{UCT-MAST}}$ and $\text{P}_{\text{GRAVE-MAST}}$, respectively. The only difference is that all agents tested in Table 5 have the C constant

set to 0.2, while the C constant for $\text{P}_{\text{UCT-MAST}}$ by Sironi and Winands (2016) is set to 0.7. For most of the games that are present in both articles, results reported in Table 5 seem to confirm the ones reported by Sironi and Winands (2016), showing that the GRAVE selection strategy with $K = 250$ has at least an equal performance to the UCT selection strategy when combined with MAST. An exception are the games of Knightthrough and Quad, for which results seem to differ. For Knightthrough, Table 5 suggests that, when combined with MAST, UCT (i.e., $K = 0$) is better than GRAVE with $K = 250$, but Sironi and Winands (2016) show $\text{P}_{\text{GRAVE-MAST}}$ to be significantly better than $\text{P}_{\text{UCT-MAST}}$. For Quad, instead, Table 5 suggests that there is not much difference between combining with MAST the UCT selection strategy or the GRAVE selection strategy with $K = 250$, but Sironi and Winands (2016) show $\text{P}_{\text{GRAVE-MAST}}$ to perform significantly worse than $\text{P}_{\text{UCT-MAST}}$. This difference is caused by the fact that the agent that uses the UCT selection strategy has a different value for the C constant used by Sironi and Winands (2016). Results suggest that, for Knightthrough, $C = 0.7$ is a sub-optimal value for the agent that uses UCT and MAST, therefore the agent that uses GRAVE performs better against it than against the UCT-MAST agent with $C = 0.2$. Conversely, $C = 0.2$ is sub-optimal for the UCT-MAST agent in Quad, causing the performance of this agent to worsen against the agent that uses GRAVE-MAST.

Table 5 also suggests that, while being among the best values of K for most of the games, the offline-tuned value of $K = 250$ is actually sub-optimal for the game of Knightthrough. This value is the same used by Sironi et al. (2020) for the baseline agent used to test various online parameter-tuning strategies and an analysis of its performance can give more insights on online parameter tuning. The fact that for Knightthrough this value is sub-optimal would contribute to explain why many instances of the online parameter-tuning agents that are tested by Sironi et al. (2020) show to benefit the agent for this game more than for most of the other games. Online parameter tuning is able to find better values for the parameter than the sub-optimal one. For further confirmation, we considered the agent that tunes four parameters with the NTBEA allocation strategy. This agent has been matched on Knightthrough against two non-tuning agents, one using the value of $K = 10$ (i.e., the one that in Table 5 shows the best performance for the game) and one using the offline-tuned value of $K = 250$. A total of 1 000 game runs were performed for each pair of agents. Results show that the win percentage of the NTBEA online tuning agent decreases from $66.6(\pm 2.92)$ when the opponent's K is set to a sub-optimal value (i.e., $K = 250$), to $59.6(\pm 3.04)$, when the opponent's K is set to the best among the tested values (i.e., $K = 10$). The significant difference in win percentage shows that it is beneficial to tune parameters online when they might otherwise be set to a sub-optimal fixed value. However, the fact that the online parameter-tuning agent is still performing better than the non-tuning agent could indicate that there are other factors at play. For example, for Knightthrough optimal parameter values might depend on the phase of the game, and online parameter tuning might be able to find the best parameter values for each game phase.

5.4 Randomization per Simulation vs Online Parameter Tuning

These series of experiments compare the agent that uses parameter randomization per simulation, SIM-RND, with the one that tunes parameter values online with the NTBEA

Game	SIM-RND		
	2 parameters	**4 parameters**	**6 parameters**
3D TicTacToe	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)

Table 6: Win percentage of SIM-RND against NTBEA.

allocation strategy. Table 6 shows the results obtained by matching the two agents against each other for two (K and Ref), four (K, Ref, C and ϵ) and six (K, Ref, C, ϵ, VO and T) randomized/tuned parameters. Regarding the overall performance, for two and four parameters online tuning seems to perform better than randomization, while when the tuned parameters increase to six, randomization performs in general at the same level of online tuning. This is probably because with six parameters the number of possible value combinations becomes too high. With short time settings, the online tuning agent does not have sufficient time to converge to good combinations, therefore evaluating combinations almost randomly. Among the tested ones, four seems to be the most interesting number of parameters to compare randomization and online tuning against the offline-tuned and values. With two parameters the performance of the two agents is close for many of the games, and with six parameters the search space is too large for online tuning to be competitive. With four parameters, instead, 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, online tuning seems more effective than randomization. In addition, for Chinook and TTCC4 with 2 players online tuning performs much better than randomization when the number of tuned parameters is four or six. On the contrary, in Quad randomization achieves a much higher performance than online tuning for all number of parameters. Moreover, in Connect 5, Connect 4 and Pentago, parameter randomization shows a better performance than online tuning for four and six parameters. In general, it seems that online parameter tuning performs better than parameter randomization on games that present only narrow winning paths and many losing paths, like Knightthrough and Breakthrough. On such games, if the search is too diversified by randomized parameter values, many losing paths will be encountered, making it more difficult to focus on winning paths. Parameter randomization, instead, seems to work better on games with more winning paths. For example, in Quad,

Game	SUB-OPT		
	2 parameters	**4 parameters**	**6 parameters**
3D TicTacToe	5.1(±1.90)	12.9(±2.67)	1.2(±0.91)
Breakthrough	2.2(±1.29)	3.6(±1.63)	3.2(±1.54)
Knightthrough	3.4(±1.59)	7.0(±2.24)	4.8(±1.88)
Chinook	26.9(±3.55)	16.7(±3.08)	6.5(±2.01)
Chinese Checkers 3P	32.1(±4.08)	20.4(±3.52)	10.3(±2.66)
Checkers	5.2(±1.80)	11.6(±2.55)	0.7(±0.70)
Connect 5	5.3(±1.54)	14.5(±2.55)	3.9(±1.36)
Quad	9.0(±2.40)	21.0(±3.52)	1.7(±1.08)
Sheep and Wolf	41.6(±4.32)	34.6(±4.17)	23.0(±3.69)
TTCC4 2P	10.0(±2.60)	10.6(±2.67)	1.8(±1.17)
TTCC4 3P	31.6(±3.97)	30.3(±3.95)	15.0(±3.05)
Connect 4	6.9(±2.09)	9.2(±2.40)	2.7(±1.35)
Pentago	15.9(±3.15)	12.3(±2.83)	6.6(±2.11)
Reversi	14.7(±3.05)	16.2(±3.20)	4.6(±1.80)
Avg Win%	15.0(±0.81)	15.8(±0.83)	6.2(±0.55)

Table 7: Win percentage of SUB-OPT against OFFLINE.

Connect 4, Connect 5 and Pentago the aim of the player is to place the pieces to form a certain shape (i.e., squares or lines), and there are usually many ways of doing so.

5.5 Comparison of Parameter Randomization and Online Parameter Tuning vs Offline-Tuned and Sub-optimal Parameter Values

These series of experiments compare parameter randomization and online parameter tuning with the offline-tuned and sub-optimal parameter values. The agent that randomizes parameter values, SIM-RND, and the one that tunes them online, NTBEA, are matched directly against the agents that use the offline-tuned and the sub-optimal parameter values, respectively (OFFLINE and SUB-OPT). Note that three versions of the sub-optimal agent are tested, which set sub-optimal values for two (K and Ref), four (K, Ref, C and ϵ) and six (K, Ref, C, ϵ, VO and T) parameters. The values of the other parameters are set to the offline-tuned values. The agents consider K and Ref when randomizing/tuning two parameters, K, Ref, C and ϵ when randomizing/tuning four parameters and K, Ref, C, ϵ, VO and T when randomizing/tuning six parameters.

To give an indication of how much influence sub-optimal parameters might have on the performance, the results obtained by matching the OFFLINE and SUB-OPT agents against each other are given in Table 7. As expected, when parameters are set to values that are sub-optimal for a game the loss in performance is generally quite high with respect to offline-tuned parameters. This confirms how important it is to reduce the risk of setting sub-optimal values.

Results obtained by matching SIM-RND and NTBEA against OFFLINE are shown in Table 8. Results are reported for two, four and six randomized/tuned parameters. These results are in line with the results presented in Subsection 5.4. When matched against an opponent that uses generally good fixed parameter values, the difference in the overall performance between SIM-RND and NTBEA is similar to the difference in performance that

Game	2 parameters		4 parameters		6 parameters	
	SIM-RND	**NTBEA**	**SIM-RND**	**NTBEA**	**SIM-RND**	**NTBEA**
3D TicTacToe	**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 8: Win percentage of SIM-RND and NTBEA against OFFLINE.

was observed when the two agents were matched against each other directly. Moreover, for two parameters both agents seem to have at least the same performance of OFFLINE in many of the tested games. For four parameters, the performance of NTBEA is still quite close to the one of OFFLINE, while the performance of SIM-RND drops for most of the games. For six parameters, both tuning and randomizing parameter values does not seem to provide any benefit in most of the games. With more parameters, the search space increases, and it is likely that the number of sub-optimal parameter combinations becomes higher. Thus, SIM-RND is probably using most of the time a sub-optimal combination, while for NTBEA it becomes harder to find good combinations to converge to in a limited amount of time. Once again, Quad is an interesting game, because it seems to significantly benefit from parameter randomization for any tested number of randomized parameters. What is interesting to observe is that among the online tuning strategies evaluated by Sironi et al. (2020) there was one, LSI, which showed a similarly good performance on Quad. A characteristic of LSI is that during its first phase, called *generation*, the allocation strategy is evaluating random parameter combinations, in a similar way to the SIM-RND randomization strategy. This can be seen as a confirmation that the search for the game of Quad particularly benefits from randomizing its control parameters.

A few other games show to benefit from parameter randomization. The performance of SIM-RND is significantly higher than the one of OFFLINE in Chinook when randomizing two parameters, and in Connect 4 when randomizing four. Results also confirm that online parameter tuning is better than randomization for Breakthrough and Knightthrough. However, only for four parameters it significantly benefits the search with respect to the offline-tuned parameter values. This means that, when tuning parameters online, it is important to select carefully which parameters to tune. First, it seems that for these games adding the parameters C and ϵ to the tuned ones influences the search more than only tuning K and Ref. Second, adding T and VO to the tuned parameters seems to increase the complexity of the parameter space too much, making online tuning detrimental for the search.

The results obtained by matching the SIM-RND and NTBEA agents against the agent that uses sub-optimal fixed values for the parameters are presented in Table 9. These

Game	2 parameters		4 parameters		6 parameters	
	SIM-RND	**NTBEA**	**SIM-RND**	**NTBEA**	**SIM-RND**	**NTBEA**
3D TicTacToe	92.4(±2.22)	93.4(±2.03)	80.5(±3.21)	85.3(±2.86)	97.3(±1.38)	96.4(±1.61)
Breakthrough	95.6(±1.80)	97.0(±1.50)	73.2(±3.89)	**97.8**(±1.29)	83.2(±3.28)	**97.0**(±1.50)
Knightthrough	92.4(±2.33)	**96.8**(±1.54)	63.0(±4.24)	**99.0**(±0.87)	70.2(±4.01)	**95.6**(±1.80)
Chinook	**83.5**(±3.07)	80.9(±3.21)	63.8(±4.06)	**86.0**(±2.85)	86.6(±2.82)	89.1(±2.51)
Chinese Checkers 3P	68.5(±4.06)	**72.0**(±3.92)	70.8(±3.97)	70.6(±3.98)	85.3(±3.09)	76.4(±3.71)
Checkers	95.3(±1.76)	**95.7**(±1.68)	67.4(±3.85)	84.1(±2.95)	98.8(±0.83)	96.5(±1.39)
Connect 5	92.8(±1.66)	94.5(±1.40)	83.9(±2.77)	67.0(±3.63)	**97.2**(±1.37)	90.1(±2.32)
Quad	**96.8**(±1.33)	95.9(±1.64)	**91.3**(±2.42)	81.8(±3.29)	**99.4**(±0.68)	92.6(±2.21)
Sheep and Wolf	56.6(±4.35)	57.0(±4.34)	**65.4**(±4.17)	62.2(±4.25)	70.8(±3.99)	67.4(±4.11)
TTCC4 2P	92.0(±2.36)	**93.5**(±2.15)	71.3(±3.95)	**89.5**(±2.68)	91.8(±2.41)	94.2(±2.05)
TTCC4 3P	65.0(±4.08)	67.7(±3.96)	**70.0**(±3.95)	67.1(±4.02)	81.7(±3.34)	79.4(±3.47)
Connect 4	**95.7**(±1.70)	94.2(±1.94)	**94.3**(±1.99)	87.8(±2.79)	**98.1**(±1.08)	95.1(±1.84)
Pentago	**89.4**(±2.64)	87.7(±2.83)	85.6(±3.02)	81.6(±3.32)	**93.4**(±2.07)	90.3(±2.56)
Reversi	81.2(±3.35)	83.4(±3.23)	67.8(±4.04)	69.5(±4.00)	93.8(±2.10)	92.9(±2.19)
Avg. Win%	85.5(±0.80)	**86.4**(±0.78)	74.9(±0.99)	80.7(±0.89)	89.1(±0.72)	89.5(±0.70)

Table 9: Win percentage of SIM-RND and NTBEA against SUB-OPT.

agents are compared for two, four and six 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 offline-tuned values. Once again, results show that for two and six parameters SIM-RND and NTBEA are close in performance, while for four parameters NTBEA performs better than SIM-RND. In general, both 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, the performance of SIM-RND is much closer to the one of NTBEA then it is to the one of SUB-OPT. Thus, in a situation where the optimal parameter values for a specific game are not known in advance, both tuning and randomizing them online seem to be valid approaches, rather than setting an arbitrary combination that might be sub-optimal.

5.6 Comparison with CadiaPlayer

These series of experiments compare OFFLINE, SUB-OPT, SIM-RND and NTBEA by matching them against a different type of opponent, a successful GGP agent for abstract games, CadiaPlayer (Finnsson, 2012). Results of these series of experiments are shown in Table 10. This table reports the results obtained by matching OFFLINE, SUB-OPT, SIM-RND and NTBEA against CadiaPlayer. All the agents tested so far implement the same MCTS strategy with the same enhancements. The purpose of these 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 except SUB-OPT show a better performance than CadiaPlayer, with NTBEA that tunes two parameters being the best. It is confirmed that online parameter tuning is successful in Breakthrough and Knightthrough for two tuned parameters and in particular for four, while parameter randomization performs the worst on these two games for four and six parameters. The use of arbitrary parameter values is confirmed not to be a good idea, because if they turn out to be sub-optimal for a game, the performance drop is quite high. Interesting results for SUB-OPT are the ones in Quad for two and four randomized

Game	OFFLINE	2 parameters		
		SUB-OPT	SIM-RND	NTBEA
3D TicTacToe	92.1(±2.36)	30.2(±4.01)	**92.3**(±2.26)	91.9(±2.34)
Breakthrough	**63.2**(±4.23)	6.6(±2.18)	50.6(±4.39)	61.8(±4.26)
Knightthrough	50.8(±4.39)	6.2(±2.12)	35.8(±4.21)	**52.2**(±4.38)
Chinook	82.8(±3.22)	49.2(±4.29)	86.6(±2.92)	**88.0**(±2.74)
Checkers	90.6(±2.32)	27.2(±3.73)	86.5(±2.72)	**91.2**(±2.28)
Connect 5	**70.4**(±3.18)	5.9(±1.76)	66.8(±3.33)	68.2(±3.29)
Quad	98.8(±0.96)	68.0(±4.09)	**99.6**(±0.55)	99.2(±0.78)
Sheep and Wolf	56.8(±4.35)	45.6(±4.37)	56.8(±4.35)	**60.4**(±4.29)
Connect 4	68.2(±3.90)	11.3(±2.70)	65.1(±4.04)	**69.7**(±3.92)
Pentago	73.0(±3.80)	29.3(±3.91)	75.0(±3.62)	**78.1**(±3.52)
Avg. Win%	74.7(±1.16)	28.0(±1.23)	71.5(±1.21)	**76.1**(±1.14)

Game	OFFLINE	4 parameters		
		SUB-OPT	SIM-RND	NTBEA
3D TicTacToe	**92.1**(±2.36)	66.3(±4.11)	91.4(±2.41)	90.4(±2.55)
Breakthrough	63.2(±4.23)	9.2(±2.54)	23.2(±3.70)	**68.0**(±4.09)
Knightthrough	50.8(±4.39)	4.6(±1.84)	19.6(±3.48)	**74.8**(±3.81)
Chinook	**82.8**(±3.22)	41.0(±4.18)	54.1(±4.22)	81.3(±3.28)
Checkers	**90.6**(±2.32)	40.5(±3.95)	63.2(±3.97)	87.6(±2.71)
Connect 5	**70.4**(±3.18)	22.6(±3.39)	61.2(±3.73)	45.5(±3.78)
Quad	98.8(±0.96)	71.6(±3.96)	98.8(±0.96)	**99.4**(±0.68)
Sheep and Wolf	**56.8**(±4.35)	39.0(±4.28)	51.6(±4.38)	51.6(±4.38)
Connect 4	68.2(±3.90)	17.5(±3.20)	**68.5**(±3.98)	63.2(±4.06)
Pentago	**73.0**(±3.80)	25.5(±3.77)	69.7(±3.95)	71.3(±3.80)
Avg. Win%	**74.7**(±1.16)	33.8(±1.28)	60.1(±1.32)	73.3(±1.19)

Game	OFFLINE	6 parameters		
		SUB-OPT	SIM-RND	NTBEA
3D TicTacToe	**92.1**(±2.36)	27.3(±3.90)	91.9(±2.35)	86.7(±2.89)
Breakthrough	**63.2**(±4.23)	4.4(±1.80)	19.4(±3.47)	45.8(±4.37)
Knightthrough	**50.8**(±4.39)	1.2(±0.96)	16.0(±3.22)	45.0(±4.37)
Chinook	**82.8**(±3.22)	19.8(±3.37)	55.1(±4.24)	63.4(±4.10)
Checkers	**90.6**(±2.32)	4.3(±1.61)	65.8(±3.92)	52.6(±4.12)
Connect 5	**70.4**(±3.18)	5.6(±1.88)	**70.4**(±3.54)	51.9(±3.95)
Quad	98.8(±0.96)	47.3(±4.38)	**99.4**(±0.68)	93.0(±2.24)
Sheep and Wolf	**56.8**(±4.35)	29.0(±3.98)	55.6(±4.36)	50.0(±4.39)
Connect 4	**68.2**(±3.90)	6.1(±2.05)	65.4(±4.04)	48.0(±4.24)
Pentago	**73.0**(±3.80)	20.5(±3.48)	64.7(±4.11)	62.6(±4.10)
Avg. Win%	**74.7**(±1.16)	16.6(±1.02)	60.4(±1.32)	59.9(±1.32)

Table 10: Win percentage of OFFLINE, SUB-OPT, SIM-RND and NTBEA (1s start- and play-clock) against CadiaPlayer (10s start- and play-clock).

parameters, and in 3D Tic Tac Toe for two randomized parameters. In these cases, its performance is significantly better than the one of CadiaPlayer, which might mean that the setting of SUB-OPT might not be the worst for this game, or that CadiaPlayer might also have sub-optimal settings for this game.

Also worth noticing is that in all previous series of experiments the performance of SIM-RND was close to the one of NTBEA for two parameters. However, against CadiaPlayer the difference in performance is higher. This means that the relative performance of these two agents does not only depend on the number of parameters that they are considering, but also on the type of opponent. From previous experiments, tuning only two parameters might have seemed not interesting, because it was only slightly improving the performance over randomization. However, against CadiaPlayer online tuning is shown to be better than just randomizing the parameters. Therefore, for the agent to be more robust against different types of opponents, two parameters are still worth tuning.

The biggest gap in the performance of SIM-RND with respect to NTBEA is visible for four parameters. Moreover, by going from two to four parameters, the performance of SIM-RND drops much more than the one of NTBEA (more than 9 points for SIM-RND and less than 3 for NTBEA). This suggest that, for different opponents, online tuning might be more robust than parameter randomization. Finally, for six parameters the overall performance of SIM-RND and NTBEA is rather similar. This is in line with the results shown for the two agents in Tables 8 and 9. Also against other opponents, SIM-RND and NTBEA are close in performance when randomizing/tuning six parameters.

5.7 Search Tree Analysis

To get more insights about the behavior of MCTS when parameters are tuned or randomized, for some of the games considered in previous experiments, statistics about the search trees built by the agents are collected for each turn. Moreover, to give an example of the structure of the trees built by the different agents, sample trees are plotted for each game.[3] The considered games are Breakthrough, Knightthrough, Quad and Connect 4, and have been selected among the games for which either online parameter tuning or parameter randomization shows to perform best in many of the performed experiments. The considered agents are OFFLINE, SIM-RND and NTBEA, with the last two agents randomizing/tuning four parameters, C, ϵ, K and Ref.

For the considered games, Figure 6 reports the maximum depth reached by the tree built by MCTS (considering the current root as having depth 0) during each turn of the game. Moreover, Figure 7 reports the average effective branching factor of the tree built by MCTS for each game turn. Note that, while the branching factor is computed considering all the actions that are legal in a node, the effective branching factor is computed considering only the actions that have actually been visited by the search algorithm. Each point in the presented plots has been computed averaging the considered value (i.e., maximum depth or average effective branching factor) over 500 runs for each game. Note that not all runs reach the last game turns, thus data points towards the end of the series are based on fewer samples.

3. The plots of the trees are inspired by the tree visualizations used by Wimmenauer (2019).

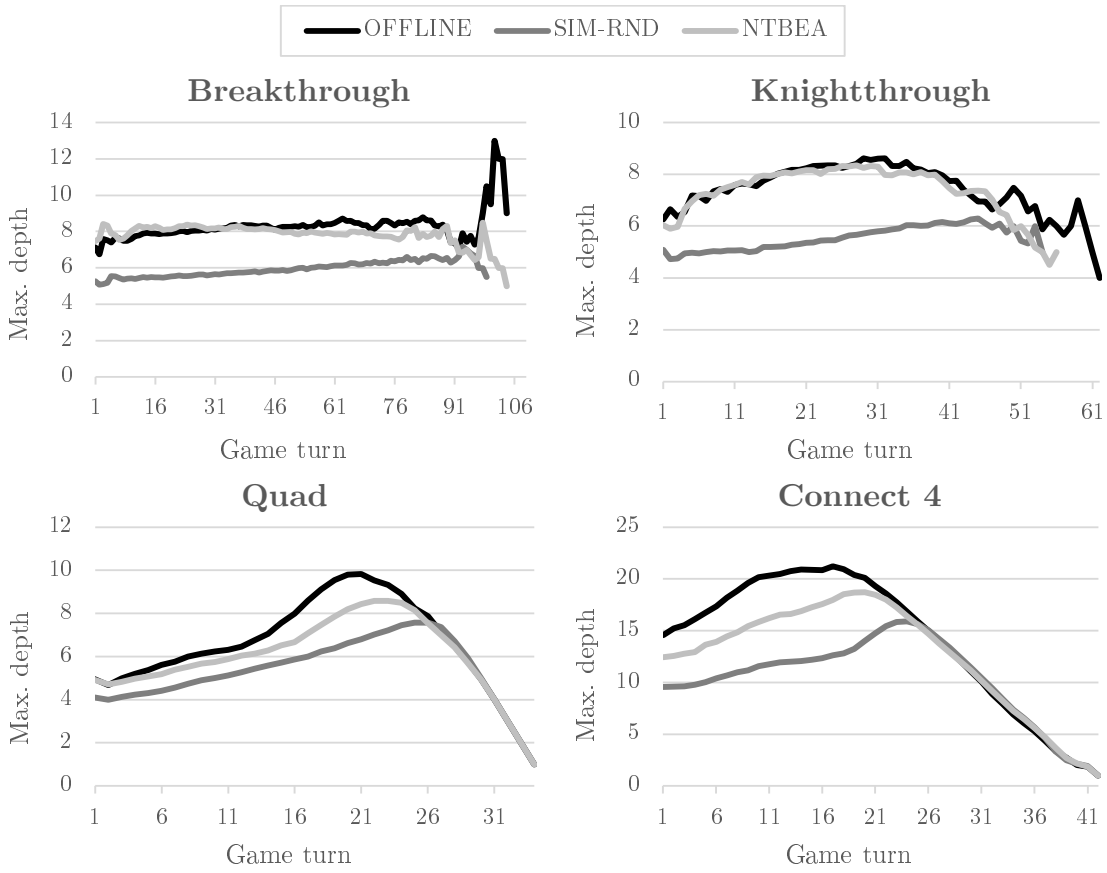


Figure 6: Maximum MCTS tree depth over game turns of the agents OFFLINE, SIM-RND and NTBEA.

As can be seen, for all games OFFLINE is the agent that in each turn builds a tree with the highest maximum depth and highest average effective branching factor. SIM-RND is the one that, in most turns, builds a tree for which these statistics are the lowest, while for the trees built by NTBEA these statistics are in-between the ones of OFFLINE and SIM-RND. Note that maximum depth and average effective branching factor only give information about how the tree is built by each of the strategies, but not on how nodes are actually visited during the search (i.e., how many times and in which order).

The statistics for the depth seem to confirm that the more randomization is introduced by changing the parameters the less the agent tends to exploit promising paths. With less or no randomization the agent focuses more on a few paths in the tree in order to visit them deeper (i.e., it is more exploitative). Results for the effective branching factor show that the more randomization is introduced the more the agent tends to explore, on average, fewer actions in each node. This might seem counterintuitive at first, because the randomization introduced by changing parameter values online would be expected to diversify the search and explore more branches of the tree. These results can be explained by considering that the effective branching factor of a search tree depends on its actual branching factor, and the tree visited by OFFLINE likely has a higher average branching factor than the one of

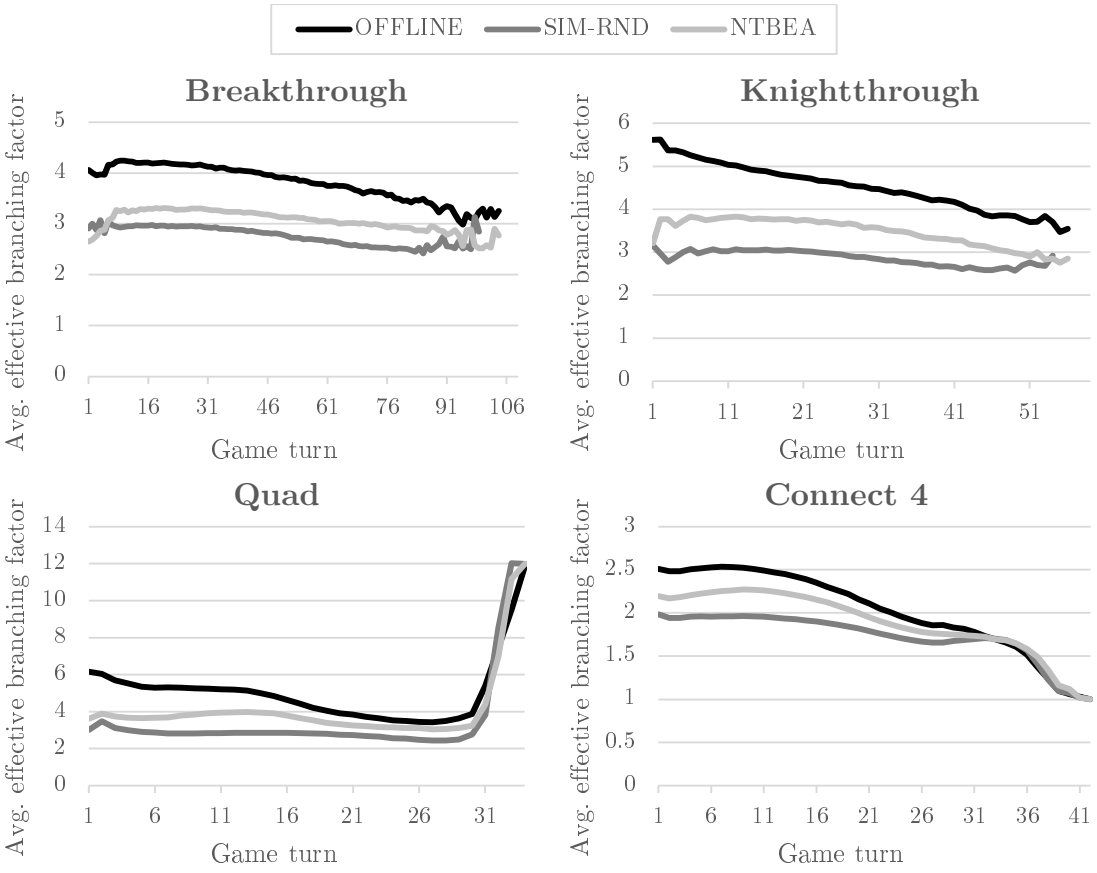


Figure 7: Average effective branching factor over game turns of the agents OFFLINE, SIM-RND and NTBEA.

SIM-RND and NTBEA. The OFFLINE agent is more likely to focus the search on realistic lines of play, which usually consist of states with higher mobility, and therefore a higher number of legal actions. SIM-RND and NTBEA, instead, try also to explore less realistic moves that might lead to parts of the tree where the branching factor is lower. For example, in games like Knightthrough and Breakthrough OFFLINE tends to explore less the actions that lead to the opponent capturing the player's pieces. On the contrary, with a more diversified search SIM-RND and NTBEA explore such actions more often. This means that the average branching factor of the search tree visited by SIM-RND and NTBEA is overall lower than the one of the tree visited by OFFLINE.

A clear difference between games like Breakthrough and Knightthrough, and games like Quad and Connect 4 emerges by looking at the depth plots. Breakthrough and Knightthrough are race games, i.e., games where the player that is the first to reach a goal wins. Quad and Connect 4, instead, are alignment games, where the winning player is the one that first aligns a number of its pieces to form a certain shape. For the latter games, it is common to have a limited number of pieces that each player can place on the board, thus the game usually has a forced end at a predefined depth, when no pieces are left to play for any of the players. This explains why the average visited depth decreases over turns for

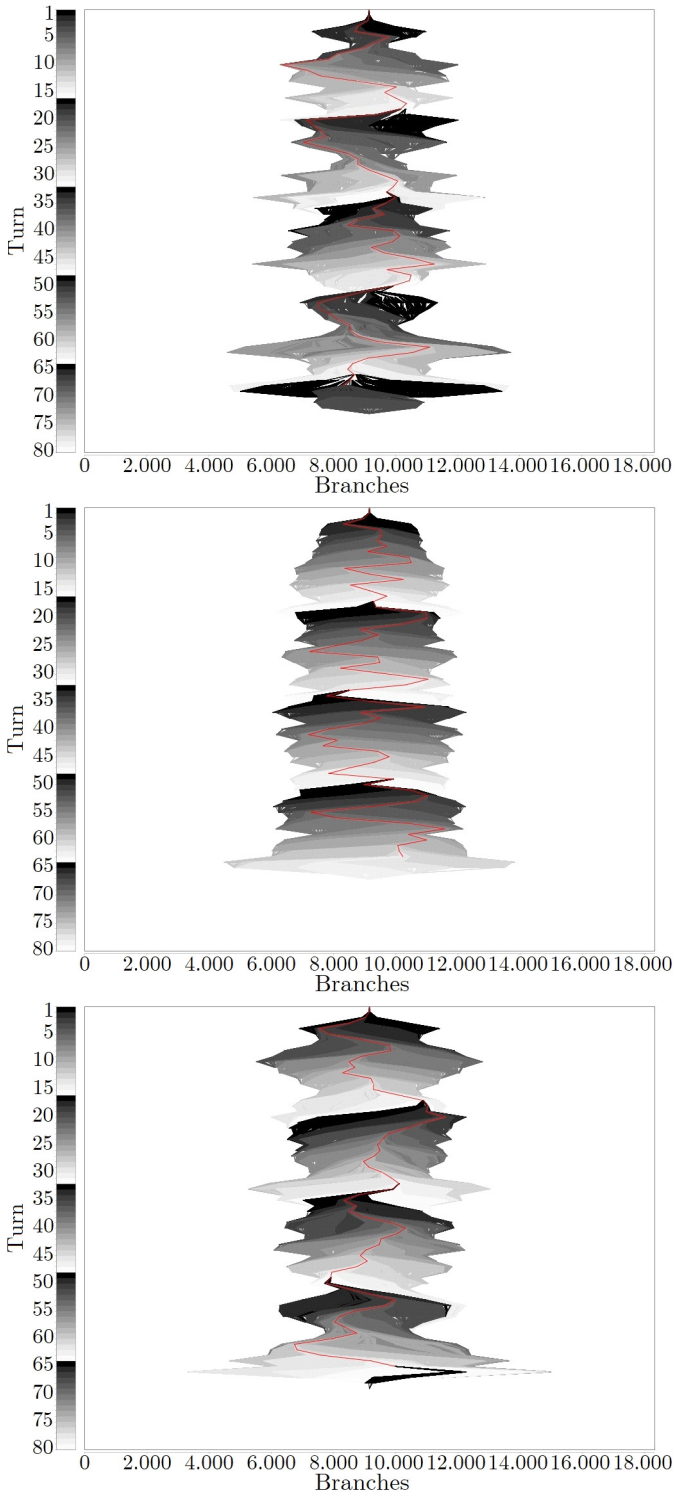


Figure 8: Trees built for Breakthrough by the OFFLINE, SIM-RND and NTBEA agents, respectively.

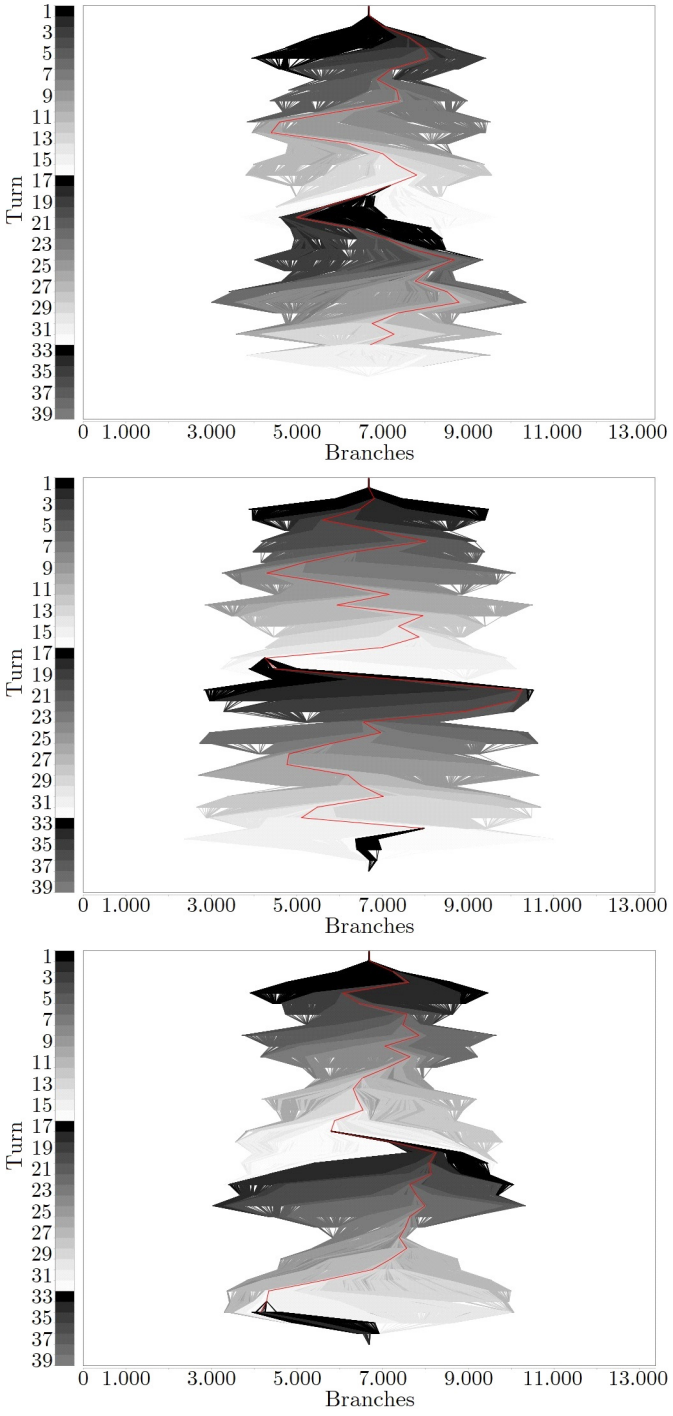


Figure 9: Trees built for Knightthrough by the OFFLINE, SIM-RND and NTBEA agents, respectively.

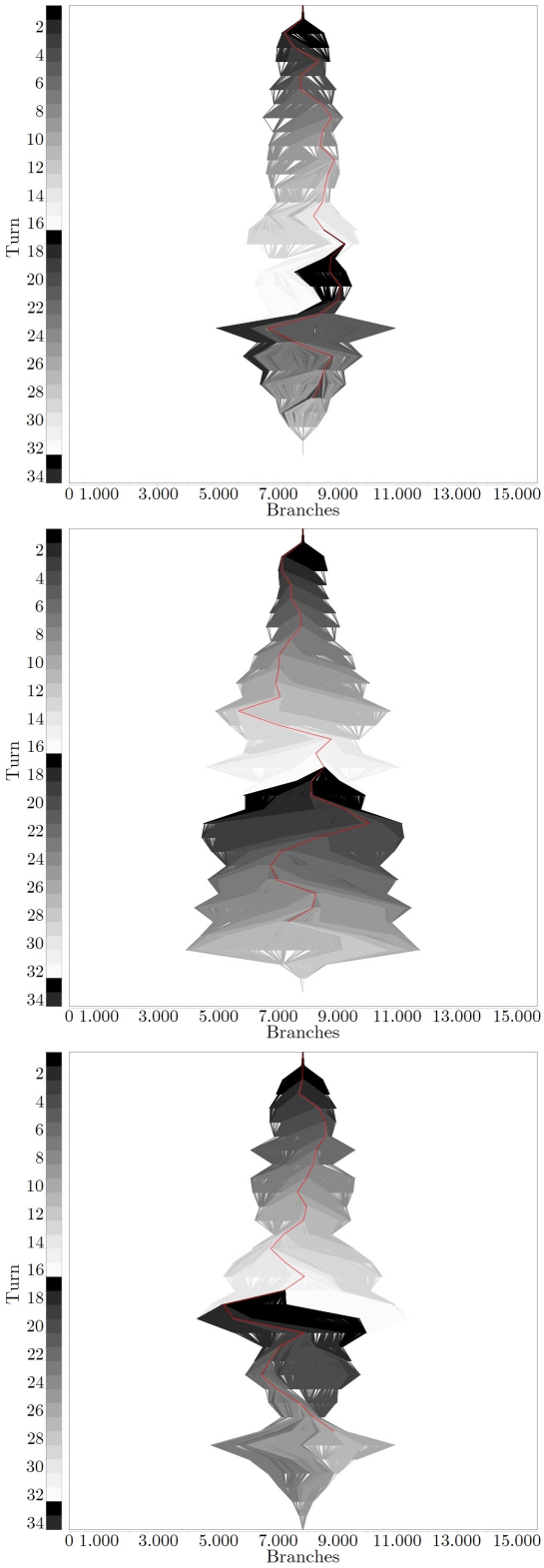


Figure 10: Trees built for Quad by the OFFLINE, SIM-RND and NTBEA agents, respectively.

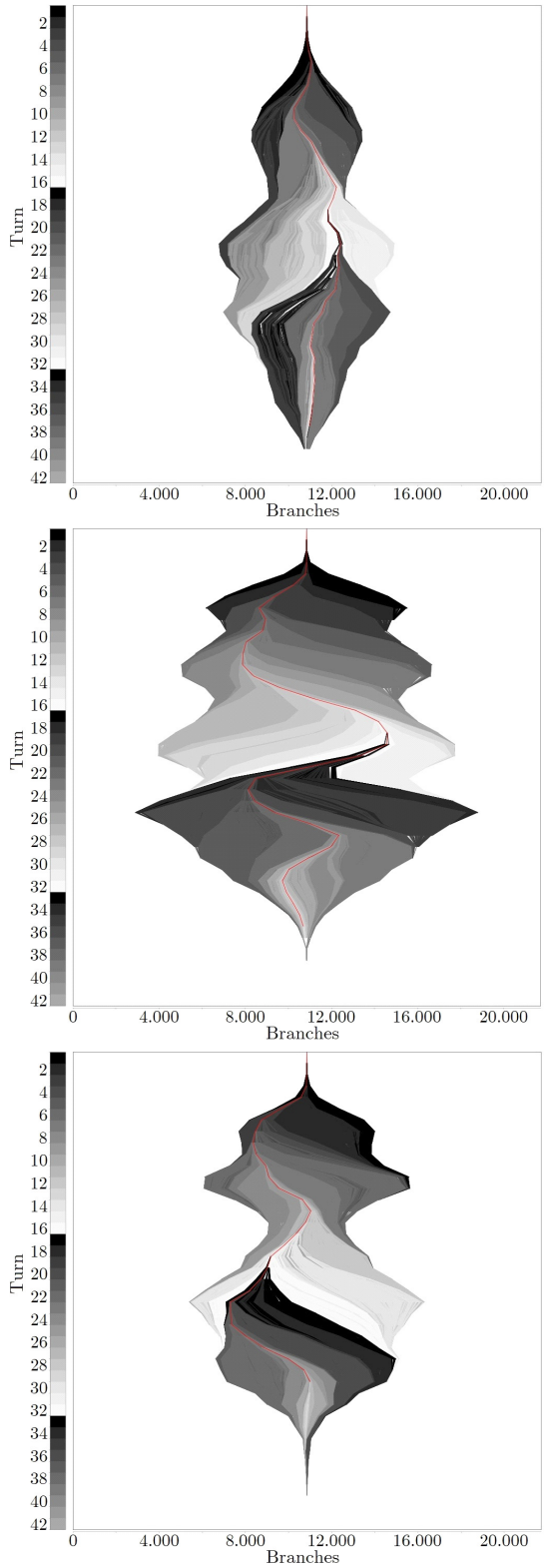


Figure 11: Trees built for Connect 4 by the OFFLINE, SIM-RND and NTBEA agents, respectively.

Quad and Connect 4, while it does not for Breakthrough and Knightthrough, for which the game might continue for a long time.

The decreasing depth of the search tree for Quad also explains what is observed in the plots with the branching factor. When the tree depth decreases, simulations are shorter and agents can perform more of them in the given time budget. In Quad, even if the number of pieces for each player decreases over time, the places on the board where a piece can be placed are still many. Thus, the actual branching factor of the search tree is still quite high. This, together with the fact that more simulations are performed in the last game turns, explains why the average effective branching factor increases. On the contrary, in Connect 4 a higher number of simulations towards the last turns does not increase the effective branching factor, because the actual branching factor of the search tree is already small. The columns on the board are getting full, thus the player can only drop its pegs in columns that still have free spaces.

More insights about the characteristics of the trees built by the three considered agents are visible in Figures 8, 9, 10 and 11. These figures show an example of the structure of the tree built by each considered agent for the games of Breakthrough, Knightthrough, Quad and Connect 4, respectively. In each plot, the x-axis reports the branches of the tree, while the y-axis reports the game turns, each corresponding to a different shade of gray. In the figure, edges added during a certain game turn are plotted with the shade of gray corresponding to the turn in which they were added. In each figure, the trees have been built, in order, by the agents OFFLINE, SIM-RND and NTBEA. Looking at the trees it is clear that each agent builds the search tree in a different way. OFFLINE seems the one that builds the most focused and deepest tree in each game turn. On the contrary, SIM-RND seems to be the agent that builds more shallow trees, visiting many different paths in each game turn, confirming the intuition that randomizing parameter values diversifies the search. The behavior of NTBEA, instead, seems to be in-between. It seems that in a few game turns the search is more focused and deep, while in others it is more shallow. This suggest that NTBEA, by tuning parameters, might also be able to detect when a more diversified search is better than a more focused one.

It is evident from the presented plots that OFFLINE, SIM-RND and NTBEA are always consistent in the way they explore a game tree (i.e., how deep or how shallow). This means that the difference in performance that is observed on the games is likely depending on some characteristics of the games themselves. As already mentioned, Breakthrough and Knighttrough present narrow winning paths and many losing paths. For these games, it seems best to have more exploration only at the start of the search, not only of the tree but also of the parameter search space. The fact that soon in the game NTBEA focuses most often on using the optimal parameter values, likely causes the exploration-exploitation trade-off of the search to stabilize. This means that MCTS will focus on the few available winning paths, investigating them more often.

On the contrary, for games like Quad and Connect 4, it seems better to have more variation in the search, using generally more exploration and less exploitation. These games have more winning paths, that can be found by a more diversified search. Moreover, what might make parameter randomization helpful for these games is that it can help recognize the presence of many optimistic moves. An optimistic move is one that gives a very good result for the player, assuming that the opponent does not realize that there is an immediate

	Larger set of feasible values
C	$\{0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9\}$
ϵ	$\{0, 0.5, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1\}$
K	$\{0, 5, 10, 25, 50, 75, 100, 250, 500, 750, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, \infty\}$
Ref	$\{0, 25, 50, 75, 100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000, 15000, 20000, \infty\}$
	Different range of feasible values
C	$\{0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8\}$
ϵ	$\{0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75\}$
K	$\{50, 100, 250, 500, 750, 1000, 1500, 2000, 2500, 3000\}$
Ref	$\{500, 750, 1000, 2500, 5000, 7500, 10000, \infty\}$

Table 11: Larger set of feasible values and different range of feasible values for each of the parameters considered in these series of experiments.

response that can nullify the effect of this move. In Quad and Connect 4 it often happens that a player places a piece to almost complete its alignment of pieces, but the opponent can immediately respond by blocking the alignment with one of its pieces. Previous research has shown that MCTS is usually slow at detecting optimistic moves, thinking for a large part of the search that they are promising (Finnsson & Björnsson, 2011). Parameter randomization might help MCTS visit other moves earlier than it would with fixed parameters, figuring out earlier which moves are too optimistic or focusing earlier on other moves that are really promising.

5.8 Importance of Accurate Selection of Parameter Values

One last aspect that should be evaluated when examining online parameter randomization and online parameter tuning is the choice of the feasible values for the parameters. The success of these two strategies that has been seen on a few of the games depends also on the fact that feasible values for the parameters have been carefully chosen to be a small number and in a reasonable range.

These series of experiments confirm this by evaluating the performance of SIM-RND and NTBEA that randomize/tune four parameters (C, ϵ, K and Ref) when the number of feasible values is increased for each parameter, and when the range in which parameter values are selected is modified. In the first case, values are selected in the same range as the original values, but a higher number of them is selected. In the second case, the number of values stays the same, but they are selected in a different range. Table 11 shows the sets of parameter values used in these series of experiments. The agents SIM-RND and NTBEA with the different sets of parameter values are matched against OFFLINE.

Table 12 reports the obtained results, together with the results obtained by the two agents using the original sets of feasible values. Once again, only results for Breakthrough, Knightthrough, Quad and Connect 4 are given, because on these games either parameter randomization or online tuning performs best, and thus are the most interesting. Finally, the table reports the win rate of all the agents on all the 14 games listed in Subsection 5.1. As expected, increasing the number of parameter values does not influence parameter randomization. As long as the values are in a range with generally good values the effect of randomization on the performance does not change that much. Changing the range in

Game	SIM-RND			NTBEA		
	Original	Values	Range	Original	Values	Range
Breakthrough	11.0(±2.75)	11.2(±2.77)	7.2(±2.27)	55.2(±4.36)	39.8(±4.29)	22.0(±3.63)
Knightthrough	20.6(±3.55)	16.4(±3.25)	8.2(±2.41)	68.2(±4.09)	53.2(±4.38)	34.0(±4.16)
Quad	72.8(±3.71)	70.3(±3.74)	58.1(±4.17)	51.9(±4.21)	52.2(±4.12)	50.7(±4.13)
Connect 4	55.4(±4.13)	52.1(±4.25)	47.5(±4.19)	46.8(±4.20)	44.9(±4.12)	49.4(±4.21)
Avg Win% (all 14 games)	36.4(±1.08)	34.9(±1.07)	30.6(±1.04)	46.5(±1.12)	43.1(±1.11)	39.9(±1.10)

Table 12: Performance of SIM-RND and NTBEA that randomize/tune four parameters. The agents consider the original sets of feasible parameter values (Original), the set with increased number of feasible parameter values (Values) and the set with a different range of parameter values (Range).

which values can be selected decreases the performance of randomization, confirming the importance of pre-selecting a range of values that are expected to be generally good on most games.

Results for NTBEA also give interesting insights. As expected, increasing the number of feasible parameter values negatively influences the overall performance of NTBEA on Breakthrough and Knightthrough. With more values, NTBEA has to search in a larger space of parameter combinations, not having sufficient time to converge to good parameter values for these games. The performance on Quad and Connect 4, on the contrary, is not significantly affected. These games might be less sensitive to different parameter values, thus the quality of the search is not reduced. However, the performance is also not increasing to the same level of parameter randomization, even if online tuning should be exploring the large search space almost randomly for a large part of the search. This might be explained by considering that online parameter tuning is still trying to focus on a subset of parameter values over time, thus not diversifying the search as much as parameter randomization does. Also modifying the range of parameter values reduces the overall performance of NTBEA, even more than increasing the number of feasible parameter values does. The decrease in performance is particularly visible in Breakthrough and Knightthrough. Interesting is to look at the results for Quad and Connect 4. The performance of NTBEA with a different range of values stays more or less the same on these games, while the performance of SIM-RND drops when a different range of values is used. This might be further confirmation that, when randomizing parameters, it is important to select an appropriate range. It could be that many of the values in the modified range are sub-optimal for these games. Therefore, while SIM-RND loses performance, NTBEA is able to focus on the use only of good values, reducing the use of sub-optimal ones.

6. Empirical Evaluation in Real-Time Settings

This section discusses the most relevant results obtained by comparing parameter randomization and online parameter tuning in the real-time environment of GVGP. Extending on the experiments performed by Sironi and Winands (2018a), the comparison is performed on the General Video Game AI (GVG-AI) framework (Perez-Liebana et al., 2019), using as baseline the MCTS-based agent MaastCTS2 (Soemers, Sironi, Schuster, & Winands, 2016).

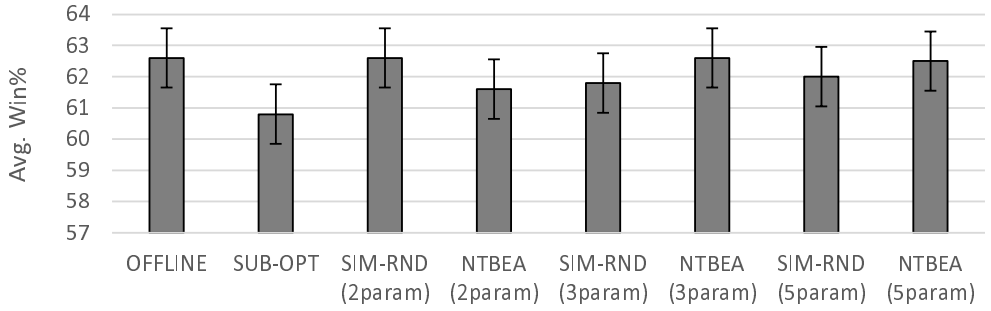


Figure 12: Win percentage of OFFLINE, SUB-OPT, SIM-RND and NTBEA on a set of 20 GVG-AI games.

A total of five parameters of the agent are randomized/tuned. These parameters influence how actions are chosen either during the selection phase or during the play-out phase of MCTS. The following heterogeneous set of 20 arcade-style video games of the single-player planning track of the GVG-AI competition is used for the experiments: Aliens, Bait, Butterflies, Camel Race, Chase, Chopper, Crossfire, Dig Dug, Escape, Hungry Birds, Infection, Intersection, Lemmings, Missile Command, Modality, Plaque Attack, Roguelike, Sea Quest, Survive Zombies, Wait For Breakfast.

In the performed series of experiments, parameter randomization per simulation and online parameter tuning with the NTBEA allocation strategy are compared with offline-tuned parameter values and sub-optimal fixed parameter values. The online tuning and randomizing agents have been tested for two, three and five tuned/randomized parameters. Figure 12 shows the average win percentage over all the games of each tested agent with a 95% confidence interval. The game tick duration is set to 40ms (i.e., the default time settings of the GVG-AI competition). Looking at the overall win percentage, all agents seem quite close in performance, although SUB-OPT seems to have a slightly lower performance. For both NTBEA and SIM-RND it seems that increasing the number of considered parameters does not influence the performance much. Results per game are omitted from the article because for most of them no significant difference in performance is observed among all the agents, even for SUB-OPT, which would be expected to perform badly in many games because of its sub-optimal parameters. One of the reasons for no particular difference in the performance could be the low number of simulations performed by the agents in each tick, which is on average only a few dozens. For the tested games, the performed simulations might not be sufficient for the parameter values, whether fixed, randomized or tuned, to influence the quality of the search. Another reason could be that for the tested games the parameters have only a limited effect on the search and changing their values does not vary the search meaningfully.

It is noted, however, that for the games Camel Race, Missile Command, Plaque Attack, Roguelike and Wait For Breakfast, SUB-OPT performs significantly worse than OFFLINE, while SIM-RND and NTBEA manage to reach the same performance of OFFLINE, suggesting that changing parameter values online is better than using sub-optimal fixed values. For these games, however, there is no significant difference in performance between SIM-RND

and NTBEA, indicating that probably the manually constructed sets of parameter values from which randomization per simulation can select contains mostly reasonable values. If this is the case, there seems to be no advantage in using an informed tuning strategy to find optimal values and exclude bad ones, randomization is already enough to control most of the search with optimal values.

The result that is most worth noticing is the one in the game Modality. For this game SUB-OPT has a win percentage of 43.4(±4.35), while OFFLINE has the significantly worse win percentage of 25.6(±3.83). It seems that the offline-tuned parameters, expected to be overall optimal, are actually sub-optimal for Modality, and the values used by SUB-OPT perform much better. This game also highlights the difference between parameter randomization and online parameter tuning, showing that there are games where the latter, using a more informed strategy to select parameter values, is able to outperform the former. In Modality, when three or five parameters are considered, NTBEA is able to reach the same performance of SUB-OPT (win percentage of 41.4(±4.32) when tuning three parameters and 41.0(±4.32) when tuning five). On the contrary, the performance of SIM-RND (win percentage of 26.6(±3.88) when randomizing three parameters and 24.6(±3.78) when randomizing five) stays close to the one of OFFLINE, which has sub-optimal parameter values for this game.

To verify whether more simulations influence the performance, another series of experiments has been performed with the same agents, but increasing the tick duration to 100ms. Although for many of the games a longer search time substantially increases the win rate of all the agents, they all seem to be quite close to each other in the overall performance, as was the case when using a tick duration of 40ms. Once again, for most of the games there seems to be no significant difference in the performance of all the agents, including the one with sub-optimal parameter values, SUB-OPT. The most interesting results are the ones for Modality, which confirm the ones obtained with a tick of 40ms. Both SIM-RND and NTBEA are at least performing as well as OFFLINE, which reaches a win rate of 26.2 (±3.86). However, when the considered parameters are three or five, the win percentage of NTBEA increases significantly to 40.6(±4.31)and 40.4(±4.31), respectively, confirming that for this game online parameter tuning is superior to parameter randomization.

7. Conclusion and Future Work

This article investigated the effect of randomizing search-control parameters for MCTS in GGP, and how this compares with using parameters that are tuned online, offline or set manually in advance with values that might be sub-optimal. The aim of this analysis was to verify whether different types of games might benefit from different strategies to set parameters, and whether there are some games that particularly benefit from the diversification of the search that parameter randomization induces. Four strategies to randomize parameter values have been evaluated: randomization per game, per turn, per simulation and per state. Moreover, randomization per simulation has been compared with online parameter tuning, giving more insights on the performance of both approaches and confirming results previously reported in the literature. A comparison of the two approaches with the performance of offline-tuned and sub-optimal parameter values has also been presented, together with a comparison of all strategies against a different MCTS-based opponent, CadiaPlayer.

Finally, an analysis of the search trees built when using randomized, online-tuned, or offline-tuned parameters was presented, together with a discussion of the importance of selecting reasonable values from which to choose when modifying parameters online.

Given the results obtained in the abstract games of the Stanford GGP framework, it may be concluded that, among the tested randomization strategies, the one that performs overall best is the one that randomizes parameter values before each simulation. This shows how this strategy might still be promising for some games, despite having been dismissed for the game of Go (Chen, 2012). Moreover, results indicate that, when parameters are randomized often within the same search (e.g., before each simulation, or before visiting each state) there are a few games for which the performance of MCTS improves over using offline-tuned parameter values. This holds especially when the number of randomized parameters is low. Moreover, analyzing randomization of single parameters, it may be concluded that for some games it is better to randomize the value per simulation rather than keeping it fixed for the whole game. This holds not only when the fixed value is sub-optimal, but for some games and parameters also when the fixed value is among the best ones. This suggests that, for some games, 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 also allows to explore the search space of the strategies, diversifying the search even more.

Results obtained by comparing parameter randomization and online parameter tuning with each other and with offline-tuned parameter values show that there is a clear distinction among games for which the first works best, games for which the second works best and games where none of the two improves the performance. The analysis of the game trees built by each of these strategies also confirms this. Online parameter tuning seems to increase the exploitation of MCTS, visiting mostly promising paths in the tree. It may be concluded that this type of search is more suitable for games that present narrow winning paths, with many losing paths. Parameter randomization, on the contrary, seems to increase the exploration of MCTS. It may be concluded that this type of search is more suitable for games that present many winning paths, possibly with hard-to-detect optimistic moves, and with a natural progression towards the end of the game (e.g., due to limited number of pieces that can be placed on the board). From the results, this seems to be the case for Quad and Connect 4. However, there are other alignment games in the considered set that might present the same characteristics, but for which parameter randomization did not have the same significant impact. This means that there might be other game characteristics that influence the performance of parameter randomization.

Results have also shown that the performance of parameter randomization and online parameter tuning depends on multiple factors. Whether it is beneficial to randomize parameter values per simulation rather than keeping them fixed to offline-tuned values or rather than tuning them online does not only depend on the game. It also depends on which and how many parameters are being randomized, on which and how many values are considered for each parameter, and on the type of opponent the agent is facing. Generally, it may be concluded that online tuning works better when the number of tuned parameters is low. Its performance generally worsens for a higher number of tuned parameters, making online tuning equivalent to selecting parameters randomly. It may also be concluded that pre-selecting a limited number of feasible values in a reasonable range is one of the requirements for both

parameter randomization and online parameter tuning to work well. Moreover, both randomization and online tuning are recommended as valid approaches for games for which optimal parameter settings are not known in advance, because the agent would otherwise be at risk of using sub-optimal fixed parameters for the entire search. Online parameter tuning, however, seems more robust against different types of opponents.

Results for the arcade-style video games of the GVG-AI framework show that parameter randomization performs more or less the same as online parameter tuning in real-time settings. This suggest that online 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-control parameter values.

Given the results presented in this article, it may be concluded that there are a few games for which randomizing parameters online improves the performance of MCTS. Moreover, although not always the best solution for all games, randomization within an appropriately selected set of values is still beneficial in GGP when parameters cannot be tuned offline and for games where the offline-tuned parameters are actually performing poorly. Parameter randomization might also be a valid alternative to online parameter tuning when the number of parameters to tune is high and time settings are limited, because the problem of tuning them online becomes too hard due to the increased combinatorial complexity. Moreover, the performance of parameter randomization is generally close to the one of online parameter tuning. This, together with the fact that it improves MCTS performance with respect to possibly sub-optimal predefined values, makes it a feasible approach for GGP. Finally, parameter randomization does not require any problem-specific knowledge. This means that, as long as the set of parameters is kept small and reasonable ranges of values for the parameters are known, it can be applied to MCTS in any domain. Some well-known MCTS application examples for which it might be used are the *Ms Pac-Man* game (Pepels, Winands, & Lanctot, 2014) and the Physical Traveling Salesman Problem (Perez, Powley, Whitehouse, Rohlfshagen, Samothrakis, Cowling, & Lucas, 2013). In addition, the proposed approach for parameter randomization might be useful for domains where MCTS has to deal with multiple heterogeneous tasks that might require variation in the search. For instance, it could be applied to job scheduling (Wimmenauer, 2019) to diversify the search for each new scheduling problem.

As future work, it would be interesting to analyze more properties of search trees than the ones analyzed here. This article focused on analyzing branching factor, depth and general shape of the search tree in each turn. It would also be interesting to look at other aspects of the search, for instance, how samples are distributed over the actions in each node. Furthermore, it would be interesting to extract game features (e.g., what type of moves are available in a game, which type of pieces are used, what type of goal should a player achieve to win, etc.), and see if there is a correlation between such features and the performance of different algorithms on the games.

Finally, it would be interesting to evaluate further the strategies presented in this article in the GVG-AI environment. Future work could investigate whether increasing time constraints to achieve a few thousands simulations per tick in GVG-AI would make a difference on the performance of parameter randomization and online parameter tuning. Furthermore, the results obtained on the game Modality show that it might be worth to further investigate online parameter tuning in real-time settings.

Acknowledgements

This work was supported by The Netherlands Organisation for Scientific Research (NWO) in the framework of the GoGeneral Project (Grant No. 612.001.121).

References

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the Multiarmed Bandit problem. *Machine Learning*, *47*(2-3), 235–256.

Beal, D. F., & Smith, M. C. (1994). Random evaluations in Chess. *ICCA Journal*, *17*(1), 3–9.

Belkhir, N., Dréo, J., Savéant, P., & Schoenauer, M. (2017). Per instance algorithm configuration of CMA-ES with limited budget. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 681–688.

Björnsson, Y., & Finnsson, H. (2009). CadiaPlayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, *1*(1), 4–15.

Bošanskỳ, B., Lisỳ, V., Lanctot, M., Čermák, J., & Winands, M. H. M. (2016). Algorithms for computing strategies in two-player simultaneous move games. *Artificial Intelligence*, *237*, 1–40.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transaction on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Cazenave, T. (2015). Generalized rapid action value estimation. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pp. 754–760.

Chaslot, G. M. J.-B., Winands, M. H. M., van den Herik, H. J., Uiterwijk, J. W. H. M., & Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, *4*(3), 343–357.

Chen, K.-H. (2012). Dynamic randomization and domain knowledge in Monte-Carlo tree search for Go knowledge-based systems. *Knowledge-Based Systems*, *34*, 21–25.

Coulom, R. (2007). Efficient selectivity and backup operators in Monte-Carlo tree search. In van den Herik, H. J., Ciancarini, P., & Donkers, H. H. L. M. (Eds.), *Computer Games*, Vol. 4630 of *LNCS*, pp. 72–83, Berlin, Germany. Springer.

Finnsson, H. (2012). *Simulation-Based General Game Playing*. Ph.D. thesis, School of Computer Science, Reykjavík University, Reykjavík, Iceland.

Finnsson, H., & Björnsson, Y. (2011). Game-tree properties and MCTS performance. In *IJCAI Workshop on General Intelligence in Game Playing Agents (GIGA 11)*, Vol. 11, pp. 23–30.

Gelly, S., & Silver, D. (2007). Combining online and offline knowledge in UCT. In Ghahramani, Z. (Ed.), *Proceedings of the 24th International Conference on Machine Learning*, pp. 273–280.

Genesereth, M., & Thielscher, M. (2014). *General Game Playing*, Vol. 8 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pp. 507–523. Springer.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In Fürnkranz, J., Scheffer, T., & Spiliopoulou, M. (Eds.), *Proceedings of the European Conference on Machine Learning*, Vol. 4212 of *LNCS*, pp. 282–293, Berlin, Germany. Springer.

Lucas, S. M., Liu, J., & Perez-Liebana, D. (2018). The N-tuple bandit evolutionary algorithm for game agent optimisation. In *Congress on Evolutionary Computation*. IEEE.

Pepels, T., Winands, M. H. M., & Lanctot, M. (2014). Real-time Monte Carlo tree search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*(3), 245–257.

Perez, D., Powley, E. J., Whitehouse, D., Rohlfshagen, P., Samothrakis, S., Cowling, P. I., & Lucas, S. M. (2013). Solving the physical traveling salesman problem: Tree search and macro actions. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*(1), 31–45.

Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J., & Lucas, S. M. (2019). General Video Game AI: a multi-track framework for evaluating agents, games and content generation algorithms. *IEEE Transactions on Games*, *11*(3), 195–214.

Ramanujan, R., Sabharwal, A., & Selman, B. (2010). On adversarial search spaces and sampling-based planning. In Brafman, R. I., Geffner, H., Hoffmann, J., & Kautz, H. A. (Eds.), *Twentieth International Conference on Automated Planning and Scheduling*, pp. 242–245.

Schreiber, S. (2017). Games - base repository. `http://games.ggp.org/base/`.

Sironi, C. F., Liu, J., & Winands, M. H. M. (2020). Self-adaptive Monte-Carlo tree search in general game playing. *IEEE Transactions on Games*, *12*(2), 132–144.

Sironi, C. F., & Winands, M. H. M. (2016). Comparison of rapid action value estimation variants for general game playing. In *2016 IEEE Conference on Computational Intelligence and Games*, pp. 309–316.

Sironi, C. F., & Winands, M. H. M. (2018a). Analysis of self-adaptive Monte Carlo tree search in general video game playing. In *2018 IEEE Conference on Computational Intelligence and Games*, pp. 397–400. IEEE.

Sironi, C. F., & Winands, M. H. M. (2018b). On-line parameter tuning for Monte-Carlo tree search in general game playing. In *6th Workshop on Computer Games*, pp. 75–95.

Sironi, C. F., & Winands, M. H. M. (2019). Comparing randomization strategies for search-control parameters in MCTS. In *2019 IEEE Conference on Games*, pp. 353–360. IEEE.

Soemers, D. J., Sironi, C. F., Schuster, T., & Winands, M. H. M. (2016). Enhancements for real-time Monte-Carlo tree search in general video game playing. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 436–443. IEEE.

Wimmenauer, F. (2019). Monte-Carlo search for leveraging performance of unknown job shop scheduling heuristics. Master's thesis, Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, The Netherlands.