

Combining Tree Search with Value Prediction Models Capable of Estimating Their Uncertainty

Maciej Świechowski *QED Games and Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw*
Warsaw, Poland

maciej.swiechowski@qed.pl
ORCID: 0000-0002-8941-3199

Abstract—The aim of this paper is twofold. Firstly, we propose a method that combines tree search – specifically, Monte Carlo Tree Search – with value prediction models conditionally based on the model’s uncertainty in a given state. If, during the search, this uncertainty is within the allowed margin, the current path is terminated, thereby saving simulation budget. This method, overlooked in literature, results in a non-uniform frequency of model usage, in contrast to typical methods applied to game-playing agents, which we summarize in the paper. Secondly, we investigate the effects of using models with various frequencies and prediction accuracies. Rich empirical results demonstrate the impact of 110 combinations of these parameters across various search budget scenarios. Generally, weaker players, for example, those performing fewer search iterations per move, benefit more from a broad selection of prediction models. For stronger baseline MCTS players, the model should only be used when its predictions achieve an accuracy greater than 0.9, especially when the objective is to develop an agent with high efficacy. Practical applicability of the proposed approach has been demonstrated in additional experiments involving chess and Othello, employing different approaches to prediction models. In both of them, it outperformed the baseline approaches.

Index Terms—Monte Carlo Tree Search, simulations, reinforcement learning, games, combinatorial optimization.

I. INTRODUCTION

MANY Artificial Intelligence (AI) researchers have devoted their skills and efforts to creating agents capable of playing mind games. The short-term goals typically include developing the strongest artificial player for a given game or achieving human-level efficacy. Longer-term goals aim to surpass top human players [1] or even solve the game completely [2]. Games have acted as a *litmus test* measuring progress in the field for several reasons [3]. Some of them are: the fact that games require intelligent strategic decision-making to achieve good results, each game provides a formal feedback signal in the form of a score (at least a *win* or *loss*), and they offer repeatable, constrained environments for controlled testing. Moreover, games are fun for humans, providing not only human opponents for comparison but also making progress exciting within the community.

Modern research trends in game AI encompass tackling increasingly complex games [4], developing programs capable of playing multiple games [5] (termed multi-game playing), and using algorithms originally designed for games in non-game contexts, such as combinatorial optimization [6], computational chemistry [7], scheduling [8], logistics [9] and many other. One of the state-of-the-art algorithms utilized for games

and other sequential decision-making problems is Monte Carlo Tree Search (MCTS). Initially proposed for *Go* [10], [11], it facilitated a quantum leap in playing efficacy. It later became a core component of the renowned *AlphaGo* [12] and *AlphaZero* [1] programs, which marked milestone achievements not only in *Go* but also in the general field of AI. Unlike traditional tree search algorithms like *min-max*, MCTS does not require a heuristic evaluation function, although it can benefit from one when available. Traditionally, these functions were hand-crafted by experts, sometimes written as a linear combination of features, with feature weights optimized using methods such as evolutionary algorithms. Currently, with machine learning (ML) leading broad AI research, the evaluation function is often represented as an ML model.

We have thoroughly analyzed literature on MCTS, which incidentally led to writing a survey paper on the subject [13]. Our findings indicate that existing combinations of MCTS with ML do not fully capitalize on dynamic estimations of uncertainty in the encountered states during the search. For more details, please refer to the next Section, where we enumerate related approaches. Our proposed approach is a self-balancing mechanism, which allocates the majority of the simulation budget to either search or prediction based on the confidence levels in particular states. Throughout this paper, we refer to *confidence* as the inverse of *uncertainty*. In addition to proposing a novel method, this paper’s contributions include an analysis of the effects of incorporating prediction models with varying frequencies and accuracies. Many insights from this analysis are presented in Section V.

This paper is organized as follows. The next section presents related work, including all commonly applied methods of combining tree search with machine learning. In Section III, we introduce our approach, followed by descriptions of two empirical experiments. The first experiment is devoted to a fundamental analysis of the properties of our approach and employs a custom setup introduced in Section IV. The corresponding empirical results are presented in Section V. The second experiment, detailed in Section VI, reports on a study conducted using real-world games, specifically chess and Othello, to further support the analysis. The final section is devoted to conclusions. In the Appendix, we discuss and reference the dataset obtained during our research, which has been made publicly available.

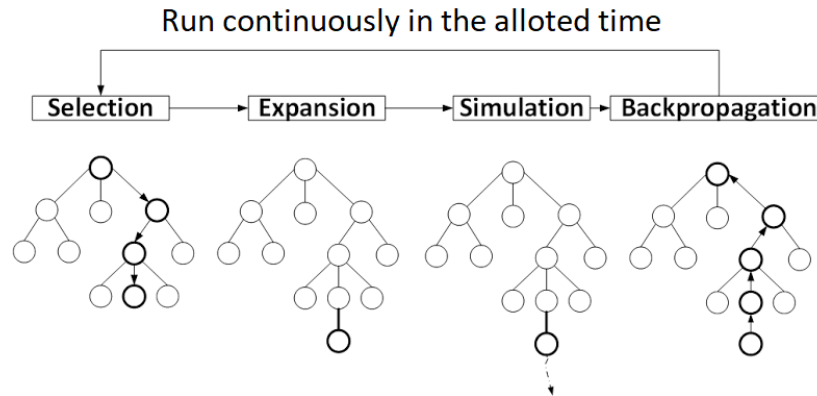


Fig. 1. Four phases in each iteration of the MCTS algorithm. The algorithm typically is executed for a given running time or a fixed number of iterations.

II. RELATED WORK

A. Monte Carlo Tree Search Background

MCTS is an iterative algorithm. Each iteration consists of four phases as depicted in Figure 1.

1) *Selection*: this phase always starts at the root node of the MCTS tree stored in memory. The goal is to explore the tree using the so-called *selection policy* until a leaf node is reached, i.e., when the next action would fall outside the tree. The most commonly applied selection policy is called Upper Confidence Bounds Applied for Trees [14]:

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{\ln[N(s)]}{N(s, a)}} \right\} \quad (1)$$

where $A(s)$ is the set of actions available in state s , $Q(s, a)$ is the observed average result of playing action a in state s , $N(s)$ is the number of times state s has been visited in previous iterations, and $N(s, a)$ is the number of times action a has been sampled in state s . The constant C controls the balance between exploration and exploitation.

2) *Expansion*: if this phase was not skipped due to visiting a leaf that contains the terminal state of the game, the state that occurs after taking the action that falls outside the tree is added as a new node to the MCTS tree.

3) *Simulation (Monte Carlo phase)*: From the last visited node in *Selection*, the algorithm performs a full ployout of the game using the so-called *default policy* to sample the results (game scores). The baseline default policy involves choosing actions at random with uniform probability.

4) *Backpropagation*: The game results are propagated back to all nodes that have been visited in the current iteration of MCTS. The number of visits, total scores, and mean scores are updated accordingly.

When the stop condition occurs, it is time to make a move in the actual game (or any decision problem the algorithm is used for). Typically, as in our experiments, it is the highest evaluated action, i.e., $\arg \max_{a \in A(s)} \{Q(s, a)\}$ from

the actions available in the root. The second most common approach is to choose the most visited action in the iterations so far.

B. Combining Search with External Models

Combining state-space search with heuristics dates back to the first *checkers* and *chess* programs. In 1959, Arthur Samuel, one of the participants of the famous Dartmouth Workshop where the term *AI* was coined, published a pioneering work titled “Some studies in machine learning using the game of checkers” [15]. The paper introduced utilizing a heuristic evaluation function with the *min-max* algorithm [16]. The author proposed a context-dependent method for determining when the tree-search algorithm should stop and assign the heuristic evaluation of the state corresponding to the node in the tree where it stopped. For instance, in Samuel’s program, the search stopped at a depth of 3-*ply*¹ if no special conditions occurred, such as the next move being a jump, the last move being a jump, or an exchange being possible. Such an obtained evaluation would be propagated to the root according to the *min-max* rule. For many years, the state-of-the-art programs for combinatorial games combined heuristic evaluation functions with a variant or extension of a *min-max* search algorithm such as *alpha-beta* pruning [17]. A notable example is *quiescence search* [18], which was one of the search techniques of *Deep Blue* [19]. It has a similar intuition behind the method we are proposing in this paper. Herein, the idea is to only evaluate “quiet” (stable) game configurations during the tree search. If the current node does not represent a “quiet” state, the search should continue.

Currently, the strongest game-playing programs often combine MCTS with ML models that can predict the quality of a state or a move in the game. However, from the MCTS perspective, this integration with external models is invariant to the technical realization of those models - whether they are ML-based, hand-crafted by experts, or created in another way.

In the literature, there are the following ways of integrating external models with the MCTS search algorithm:

- In the Monte Carlo phase:

¹A *ply* is a turn taken by one of the players. In two-player sequential games, typically a whole turn consists of two *plies*.

- Early termination: a very common approach (e.g., in [20], [21], [22]) is to terminate a simulation with a small probability or at a fixed depth and return the model’s evaluation instead of the simulation outcome.
 - Complete replacement: as a special case of the above, the evaluation may completely replace the Monte Carlo phase, as reported in [23], [24].
 - Epsilon-Greedy: during simulation, the agent chooses the highest-evaluated action by the heuristic with probability ϵ and otherwise a uniform random one. This technique has seen numerous applications [25], [26], for example to the so-called *history heuristic* [27].
 - Altering policy distribution: in a more general case, the evaluations from the model affect the probabilities of actions chosen in the simulation. The most common approaches are based on *Soft-max* [28] or *Boltzmann/Gibbs* distributions [29]. The policy for the Monte Carlo phase can be fully represented by a model, as demonstrated by the rollout policy network in *AlphaGo* [12].
 - Action pruning: based on a heuristic, certain actions may be removed from the available options [30]. This approach can again be considered a special case of altering policy distribution, wherein some actions are assigned zero probability.
- In the Selection phase:
 - Weighted evaluation: the heuristic state evaluation can be a combination of the standard MCTS estimation of the Q value [31]. For instance, in *AlphaGo* [12], the evaluation of a state was a linear combination of the scores obtained by random play-outs in MCTS and the estimations from the value network v_θ - trained using reinforcement learning:

$$V(s) = (1 - \lambda)v_\theta(s) + \lambda \cdot \text{simulation}(p_\pi) \quad (2)$$

- Weighted exploration: both *AlphaGo* and *AlphaZero* use a variant of the pUCT algorithm (first introduced in [32]), which leverages the learned policy to adjust the exploration term C in the UCT equation (see Eq. 1). Initially, the exploration favors actions with high prior probability and low visit count, but asymptotically it shifts focus toward higher Q values.
- Progressive widening: as a more general case, the heuristic estimation can be used to “kick-start” the process. The weight denoting its influence is reduced as the number of iterations grows [33].
- Move sorting: the model’s evaluation can be used to determine the order in which MCTS selects the actions for the first time [23], [34], [35]. Although the algorithm will eventually select each action in a node at least once, certain nodes down the tree are less likely to be revisited - therefore the impact of the first visited action is significant.

C. Including Uncertainty in MCTS

The following works focus on managing uncertainty within the MCTS process. They are loosely related to our approach in that they address uncertainty. However, a distinctive feature of our method is its emphasis on the uncertainty estimation of an external model. While we do not have the ability to learn and refine the estimates during the search, we also avoid errors, e.g., related to a lack of samples. Both approaches can be used together to complement each other.

Let us start with UCB1-Tuned, introduced in [36], which is a modification of the UCB formula (upon which the UCT formula is based). It introduces an additional term to the exploration component to account for the variance of the rewards for each action. The level of exploration for actions with lower variance can be reduced. The exploration component in UCB1-Tuned is presented in Eq. 3 below:

$$\sqrt{\frac{\log N(s)}{N(s, a)}} \cdot \min\left(\frac{1}{4}, V(s, a) + \sqrt{\frac{2 \log N(s)}{N(s, a)}}\right) \quad (3)$$

where $V(s, a)$ is the empirical variance estimate of the rewards received from taking action a in state s and the remaining symbols are as in Equation 1.

In a study focused on comparing 11 different selection strategies in the game of Tron, conducted in [37], UCB1-Tuned was the best-performing strategy. Building on UCB1-Tuned, the authors of [38] proposed UCBT, which integrates a formal Student t-test to detect high and low variance rewards [38]. This modification automatically adjusts the exploration constant (C) in the UCT formula (making it potentially parameter-less). This way, it directs the search budget to parts of the tree with higher uncertainty.

In a paper titled “The Second Type of Uncertainty in Monte Carlo Tree Search” [39], the authors present a method named MCTS-T+, which introduces a third term to the UCT formula in addition to existing action quality and exploration factor. The added term represents the namesake uncertainty and is based on the variation in subtree size. Larger subtrees are generally associated with higher variance. Furthermore, MCTS-T+ accounts for loops in simulations, where the same state is encountered at least two times during the same simulation episode. The authors underline the differences between loops and transpositions. In particular, MCTS-T+ detects situations where loops should not be explored at all.

Another approach is called Bayesian UCT, which utilizes Bayesian principles to represent uncertainties in node values [40]. By combining prior reward information with stochastic trial results from leaf nodes, it updates the node values using posterior distributions. These distributions are propagated upward in the tree using an inference model. The primary method for representing these probability distributions is through Gaussian approximations. A similar idea of modeling scores of states and actions as Gaussian distributions was presented in [41]. Here, the authors introduce a new back-propagation scheme (that propagates uncertainty from the leaf nodes to the root) based on the Wasserstein barycenter and α -divergence of all children nodes along the path.

III. THE METHOD

The proposed method extends the baseline MCTS algorithm introduced in Sec. II-A. The idea is to use a model together with MCTS that satisfies the following conditions:

- 1) Given a state, it estimates (predicts) the result of the game for the participating players. Such models, approximating the so-called *value function*, are very popular in Machine Learning research for games. They can be trained using historical results from the game. They are less popular when the state evaluation model is created manually by experts - in such cases, the evaluation function can be used to compare the quality of various states, but the absolute values do not often represent the game results, only correlating with them (e.g., the material-based evaluation in chess engines).
- 2) It also returns the confidence estimation of its prediction, i.e., how certain or uncertain it is. Such information is not utilized enough in game-playing programs as discussed in *Introduction* and *Related Work*. Certain machine learning models, such as Random Forests or Gaussian Processes, already have a built-in quantification of uncertainty. When such an inherent mechanism is unavailable, there are external techniques to measure uncertainty such as Monte Carlo Dropout or Bayesian Active Learning. Finally, a separate model can be trained with the objective to quantify the uncertainty attributed to the output of the given prediction model.

The proposed algorithm introduces two modifications:

- In the *Selection* phase: upon the creation of a node, the algorithm checks if the prediction model’s confidence exceeds a certain predefined threshold considered sufficient. The exact value is game-specific and may require experimentation. In our main experiment, we use a probability parameter for this. If the confidence threshold is reached, the *Selection* phase immediately ends and the algorithm proceeds to *Backpropagation*, which follows the standard protocol of the algorithm.
- In the *Simulation (Monte-Carlo phase)*: similarly, at each simulation step, the model’s confidence in the current state is assessed. If the confidence exceeds the *threshold*, the simulation is terminated early, and the model’s prediction about the game result is sampled instead. This idea is illustrated in Figure 2.

In the proposed method, it is more appropriate to use steps instead of iterations as a stop condition to control the running budget of the algorithm. A *step* corresponds to one action and one change of the currently visited state, which can be performed in the *Selection* or *Backpropagation* phase. In the literature, a step is often referred to as a forward model call in this context. The rationale for using this setup is that our iterations have unpredictable length due to the context-dependent test whether the model is confident about the current state. We can make the most out of the simulation budget by increasing the granularity of its measurement. This idea is presented in Figure 3.

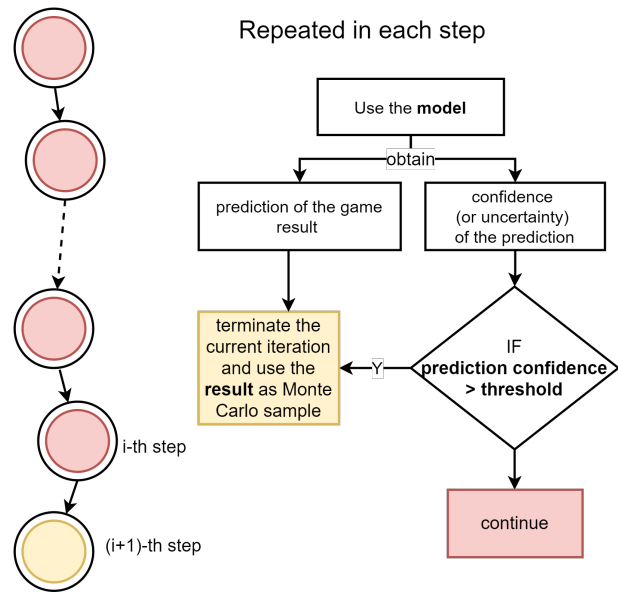


Fig. 2. In each step of the search algorithm, there is a check to determine whether to use a model based on its confidence estimation. This approach can be applied to any search algorithm that operates with the same type of values (e.g., game outcomes) that the model predicts. We employ this method with MCTS, one of the leading techniques for addressing combinatorial games.

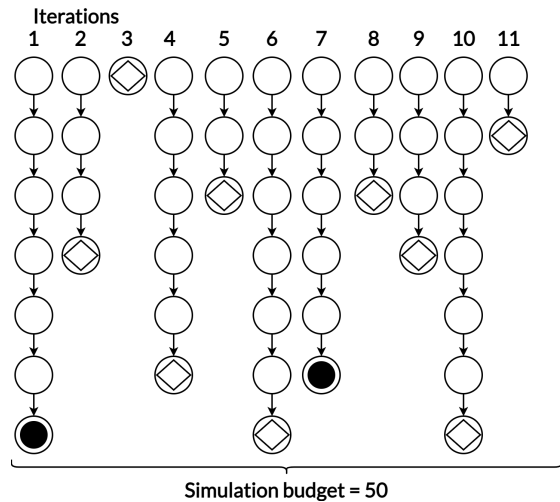


Fig. 3. An illustration of non-uniform-length iterations that are dynamically terminated. The simulation budget is equal to the number of steps performed in all iterations. Nodes with diamonds inside represent simulations terminated due to the uncertainty of the prediction model being low enough. Nodes with smaller black filled circles inside represent natural game end-states

IV. EXPERIMENTAL SETUP FOR FUNDAMENTAL ANALYSIS

A. The Environment

The first test environment has been inspired by research presented in the article [42] by Galvan and Ameneyro, which concerns optimizing the UCT formula used in MCTS.

Herein, the authors define “games of function optimization”. These are single-player games. Each game (variant) is associated with a function $f : \mathbb{R} \rightarrow \mathbb{R}$. A state in the game is defined by an interval of x values: $[A, B]$. The root state corresponds to the entire allowed domain. In our experiments, we set $A = 0$ and $B = 1000$, therefore, $x \in [0, 1000]$.

In each state, the player has two actions:

- a_1 : *go left*, which results in a state that corresponds to the first half of the parent's state interval:

$$\text{next}([A, B], a_1) = [A, \frac{A+B}{2}] \quad (4)$$

- a_2 : *go right*, which results in a state that corresponds to the second half of the parent's state interval:

$$\text{next}([A, B], a_2) = \left[\frac{A+B}{2}, B \right] \quad (5)$$

The goal of the game is to **find the function maximum**. In the referenced paper [42], the game state was terminal when $|B - A| < 10^{-5}$ for this state. We define a **terminal state** as having a depth of 26, where the root has a depth of 0. This is equivalent to reducing the original interval $2^{26} \approx 6 \times 10^8$ times. Since our starting interval is $[0, 10^3]$, after 26 bisections, the terminal interval will be of length $< 10^{-5}$. In a terminal state, the game result (propagated score) is determined by the function value at the midpoint of the interval.

In the experiments, we use three functions: f_1 , f_2 , and f_3 , presented in equations 6, 7, 8 and with plots shown in Figure 4.

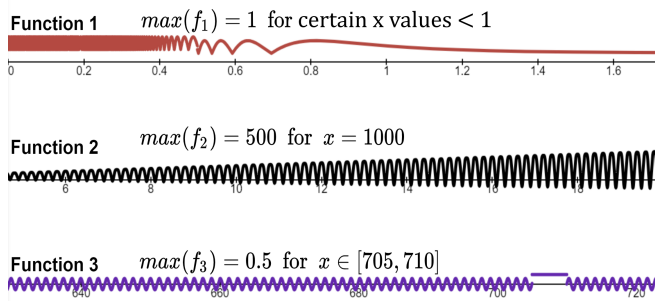


Fig. 4. Three functions used for the experiments. Only the portion of the $[0, 1000]$ domain is shown. Function 1 does not have a maximum past 0.4. Function 2 is generally increasing (with oscillations). Function 3 has the maximum value on a short sub-interval.

The first function is directly taken from [42]. The maximum, equal to 1, is achieved for several x values in the $[0, 1)$ part of the interval. It is interesting due to its ruggedness - the maxima lie in the vicinity of small values. Moreover, in the remaining part, i.e., $[1, 1000]$, there is not a single maximum. Therefore, the first actions taken in the game have to be perfect.

$$f_1(x) = \begin{cases} 0.5 + 0.5 \left| \sin\left(\frac{1}{(x+0.1)^5}\right) \right| & x < 0.5 \\ \frac{7}{20} + 0.5 \left| \sin\left(\frac{1}{x^5}\right) \right| & x \geq 0.5 \end{cases} \quad (6)$$

The second function is also taken from [42], but in that paper, it was confined only to the $[0, 1]$ domain, whereas we use $[0, 1000]$ as mentioned before. In contrast to the first function, here the maximum, equal to 500, is achieved in the very last section of the domain.

$$f_2(x) = 0.5x + (-0.7x + 1) \sin(5\pi x)^4 \quad (7)$$

Finally, the third function is particularly difficult because the maximum, equal to 0.5, is found in a small section when $x \in [705, 710]$.

$$f_3(x) = \begin{cases} 0.5 & x \in [705, 710] \\ 0.3 * \sin(0.05x) & \text{otherwise} \end{cases} \quad (8)$$

The motivation behind using this experimental setup is three-fold:

- 1) Single-player games are convenient to present the idea and analyze the state-space search capabilities of the respective approaches.
- 2) The results will stabilize more quickly - there is no variance associated with testing against various opponents. Moreover, the results are absolute, not relative to other players.
- 3) Our experiments require prediction models - it is easy to simulate them in the proposed environment.
- 4) We refer to an existing body of research (article [42]).

B. The Tested Approaches (Player Variants)

The aim of the experiments is to:

- 1) Validate the concept of using prediction models in a context-dependent way. We want to compare the results of such approaches to the regular MCTS with various simulation budgets as well as against one alternative method for integrating search with prediction models.
- 2) Analyze how often the models need to be certain in order for them to be useful compared to vanilla MCTS.
- 3) Analyze the effects of “false positive” firing, i.e., when the models falsely evaluate confidence but their output is, in fact, inaccurate.

All these together will determine the circumstances under which it is worth using the proposed method.

Therefore, we have prepared a family of MCTS-based players, all following the proposed method according to the description in Section III. The exploration parameter C (see Equation 1) was set to $\sqrt{\max_{x \in [0, 1000]}(f_i(x))}$ for each of the game variants, respectively. They differ by the following parameters:

- 1) **Steps** - the number of simulation steps they are allowed to perform per each move (decision state). When this limit is achieved, the player makes a move. Typically, MCTS-based players have a time for move allotted or an iteration budget. The time limit is hardware-dependent, so we want to avoid it in research, if possible. The step limit allows for more granularity than the iteration limit as each iteration consists of steps. We experimented with three values: $Steps \in [100, 1000, 10000]$.
- 2) **Frequency of estimated certainty** (of the model) - this is the probability that in a given state, the model will say it is confident (certain) about its prediction, and therefore, this prediction will be used (see Figure 2). For example, if the frequency is equal to 1.0, the model's prediction will be used in every step. We wanted this to be consistent for a state, so we store the result of the random draw in a given state. Whenever the same state is visited again, the model will or will not be used exactly the same as it was during the first visit to the same state. This parameter was tested with a whole range of values: $frequency\text{-of-}EC \in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$. It is worthwhile noticing that the value of 0 reduces to the

baseline MCTS algorithm. This way, we do not need a separate vanilla MCTS player. This parameter simulates a joint property of the game environment and the model. In a specific game, the frequency of estimated certainty would depend on the model’s quality, the difficulty of game states for the model’s assessment, and the chosen threshold for certainty.

- 3) **Model accuracy** - this is the probability that a model will not make a mistake and return the correct prediction. The correct prediction is defined as:

$$\text{predict}([A, B]) = \max_{x \in [A, B]} f(x) \quad (9)$$

The incorrect prediction is sampled with a uniform random distribution from the interval around the correct prediction. The range R around the correct prediction is inversely proportional to the model’s accuracy:

$$\text{predict}([A, B]) = f(\mathcal{U}[x^* - R, x^* + R]) \quad (10)$$

where:

- $x^* = \arg \max_{x \in [A, B]} f(x)$
- $R = \frac{B-A}{2} * (1 - \text{accuracy})$
- $\mathcal{U}[x_1, x_2]$ denotes a uniform random distribution over the interval $[x_1, x_2]$

Because the method advises to check for the model’s confidence and then use its prediction, incorrect decisions which are possible if the accuracy is lower than 1.00, effectively simulate the case of *false positive* confidence estimation.

The following values were prepared for experiments: $\text{accuracy} \in [0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0]$. Less accurate models would not be worth using at all in the game.

Additionally, we introduce a player named *AZ-Ins-LE*, an MCTS-based player that employs a simple AlphaZero-inspired leaf evaluation method. Given a prediction model, the first time a node is created, the player backpropagates the model’s prediction for that node. On subsequent visits to the same node, it ignores the value function prediction and instead selects a child node and traverses to it as the regular MCTS algorithm. If the selected child node is newly created, the model’s prediction is used there. The *AZ-Ins-LE* player is parameterized with steps (for controlling the simulation budget) and model accuracy parameters. However, the frequency of estimated certainty parameter is not applicable to it as its way of using the model is fixed.

This approach is similar to ours; however, the key difference is that *AZ-Ins-LE* always uses the model immediately after creating a node. In contrast, our method employs the model conditionally, based on its confidence level. There can even be iterations in which the model is not used at all. Furthermore, while *AZ-Ins-LE* can expand a node where the model was used in previous iterations, our approach treats such a node as a leaf node during the current simulation process.

V. RESULTS FOR FUNDAMENTAL ANALYSIS

For each function and steps budget, each player variant (with a specific parameterization) played 300 games – repeats

of the experiment. In this section, after exploring various presentation styles for results, we concluded that aggregated results offer the most clarity for interpretation. Detailed logs and results, allowing for a per-function analysis, can be found in the Appendix. Generally, the first function (i.e., f_1), was the easiest for the methods to optimize.

The aggregation was performed as follows: first, a mean score $\text{mean}(f_i)$ was calculated for each function f_i over 300 repeats. Then, the mean scores were normalized to the $[0, 1]$ interval in the following way:

$$|\text{mean}(f_i)| = \frac{\text{mean}(f_i) - \min(f_i)}{\max(f_i) - \min(f_i)} \quad (11)$$

The final scores presented in the tables were averages among functions:

$$\text{score} = \sum_{i=1}^3 \frac{|\text{mean}(f_i)|}{3} \quad (12)$$

The results calculated according to the aforementioned methodology for different simulation budgets are presented in Fig 5. To simplify analysis of the results, we introduce specific color coding. Cells with a light turquoise background represent scenarios in which the proposed method achieved a better score than the vanilla variant of the MCTS algorithm but not better compared to the AlphaZero-inspired approach (*AZ-Ins-LE*). This vanilla variant is equivalent to using a model with the frequency of estimated certainty equal to 0 (i.e., not using a model at all) and represented in the first rows in the tables. Cells with a darker blue background highlight cases in which the proposed approach outperformed *AZ-Ins-LE* (the last row) using the same prediction model with the same accuracy but not the baseline MCTS. Finally, cells with a gold background represent scenarios in which the proposed method outperformed both comparative methods.

The key observations and insights gained from analyzing the results are as follows:

- For 100 iteration steps per move, the baseline MCTS player exhibits limited efficacy in solving the function-optimization problems, achieving a score of 0.54. In comparison, the method introduced in this paper outperforms the baseline in 78 out of 100 parameter combinations. However, for budgets of 1000 and 10000 steps, the scores of the baseline improve to 0.81 and 0.85 respectively, and the advantage of the proposed method over the baseline MCTS is exhibited in 28 and 22 instances, respectively. The larger the simulation budget allotted per move, the higher the required accuracy of the model to surpass conventional MCTS. Notably, increasing iteration steps from 100 to 1000 has a more substantial impact on the score than increasing from 1000 to 10000.
- Using a predictive model in the proposed approach enables an MCTS-based player with a smaller computational budget to be as effective (in terms of the achieved results) as one with a substantially larger budget. For example, with 100 iteration steps,

the proposed approach can achieve a score of 0.81 – equivalent to that of an MCTS player performing 1000 steps – provided the model has sufficient frequency of estimated certainty and accuracy. Specifically, with (frequency-of-EC, accuracy) parameters like (0.4, 0.85), (0.5, 0.85), (0.3, 0.85), or ($\geq 0.3, 0.95$), the proposed method can achieve or surpass a 0.81 score with just a 100-step budget. Furthermore, if the model accuracy reaches 0.9 or higher, certain frequency of estimated certainty settings allow the proposed approach to outperform baseline MCTS with even the largest (10000) budget. This highlights the potential leverage that high-accuracy predictive models can offer, even with significantly lower simulation resources. The readers are encouraged to investigate the results to find more such parameter combinations.

- Regardless of the model’s frequency of estimated certainty or simulation budget, employing a model with an accuracy of at least 0.95 consistently provides an advantage over the baseline MCTS (as indicated by the entire column being highlighted in light turquoise or gold). Thus, 0.95 might be considered a recommended safe threshold for model accuracy when adapting this method to a new problem. Notably, for a simulation budget of 100 steps, an accuracy of 0.8 is already sufficient to consistently outperform the baseline MCTS.
- In most cases, it is beneficial to mix tree-search with model predictions. In the experiments, the highest frequency of estimated certainty yields the best score only when the model accuracy is 1.00. For lower accuracies, the score distribution across frequencies is non-monotonic. For example, with an accuracy of 0.95, the optimal frequency is 0.8. As accuracy decreases, the optimal frequency also reduces, allowing the search process to “correct” some of the model’s errors. To ensure the model is not used too frequently, one can set a high confidence threshold.
- Accurately estimating the model’s uncertainty is crucial, as it is preferable to rely on pure search rather than using inaccurate models provided that the pure-search-based performs relatively well (as observed in the case of 1000 and 10000 simulation budgets for the tested problem).
- The proposed method is capable of achieving the optimal score for the problem, but this requires a (perfect) model accuracy of 1.0. Moreover, the model must be used to terminate iterations at least 80%, 60%, and 40% of the time for simulation budgets of 100, 1K, and 10K steps, respectively. This indicates that as the number of performed iterations increases, the required frequency of estimated certainty decreases.
- Compared to the *AZ-Ins-LE* method, the proposed approach generally shows an upper hand. In all three tables shown in Fig. 5, the dark blue and gold areas,

which indicate the advantage of our method, are large. The only consistent exceptions occur when either the prediction model is always used or the model has perfect accuracy. In the former case, both approaches become very similar in their underlying procedures. In the latter case, i.e., when the model has maximal accuracy, the *AZ-Ins-LE* approach becomes more favorable if the model frequency of estimated certainty is less than 1.0. However, when both frequency and accuracy are 1.0, both approaches achieve the optimal score.

- When the model accuracy is ≤ 0.85 , increasing simulation budgets appears to have only a marginal impact on the performance of the model-equipped players and, in some cases, may even lead to slight performance deterioration. To investigate this behavior, we analyzed potential causes. In some instances, the worse results fall within statistical error. However, in other cases, the primary reason for the decline in performance was overestimation (an overly optimistic evaluation) by the model - which the MCTS algorithm is particularly susceptible to. With more iterations, the likelihood of overestimation increases. As model accuracy improves, the less risk of overestimation.
- Finally, it is interesting to note an emerging pattern. The proposed method frequently surpasses both comparative methods (indicated by the large gold area) with the lower computational budget of 100 iteration steps per move. However, with higher budgets, the extent of this gold area becomes small, meaning that the results for which the method is better than the comparative ones are fragmented. With the exception of a model accuracy of 0.95, for which the gold area remains present, the method is typically superior to only one comparative method, depending on the model’s accuracy. Specifically, it tends to outperform *AZ-Ins-LE* at accuracies ≤ 0.85 and baseline MCTS at accuracies ≥ 0.95 .

Please keep in mind that what counts is the conditional accuracy of the model given its confidence is higher than the assumed threshold. Only in such states will the model be used (see Figure 2 for the usage condition). Therefore, the accuracy measured in game states where it is used must be high, rather than the absolute accuracy across all possible game states. Achieving high absolute accuracy is typically very challenging, whereas it is more feasible to train a model that frequently indicates a lack of confidence but delivers highly accurate predictions when it does assert confidence.

VI. EXPERIMENTS WITH CHESS AND OTHELLO

The experiments aimed at analyzing fundamental properties of the proposed approach were supplemented with two experiments using actual games, with two very different ways of introducing prediction models capable of estimating their uncertainty. In this setup, the proposed approach was directly compared against the vanilla MCTS-based player and the AlphaZero-inspired leaf evaluation approach (*AZ-Ins-LE*), as described in Section IV-B.

iterations = 100										
Model	Accuracy									
Frequency of EC	0.4	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	1
0	0.54	0.54	0.54	0.54	0.54	0.54	0.54	0.54	0.54	0.54
0.1	0.51	0.56	0.57	0.63	0.63	0.67	0.70	0.72	0.73	0.74
0.2	0.52	0.53	0.60	0.66	0.67	0.73	0.73	0.78	0.79	0.81
0.3	0.49	0.53	0.59	0.67	0.69	0.75	0.78	0.83	0.85	0.86
0.4	0.46	0.53	0.58	0.66	0.72	0.75	0.81	0.85	0.89	0.93
0.5	0.45	0.54	0.59	0.68	0.68	0.75	0.81	0.86	0.90	0.96
0.6	0.46	0.49	0.57	0.66	0.71	0.76	0.78	0.86	0.91	0.97
0.7	0.41	0.49	0.55	0.64	0.70	0.75	0.81	0.87	0.91	0.99
0.8	0.43	0.50	0.55	0.63	0.68	0.71	0.79	0.85	0.92	1.00
0.9	0.44	0.49	0.55	0.60	0.66	0.69	0.76	0.83	0.88	1.00
1	0.33	0.36	0.40	0.47	0.52	0.57	0.63	0.77	0.86	1.00
AZ-Ins-LE	0.33	0.36	0.4	0.47	0.52	0.58	0.62	0.8	0.85	1.00

iterations = 1000										
Model	Accuracy									
Frequency of EC	0.4	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	1
0	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
0.1	0.54	0.58	0.64	0.69	0.71	0.73	0.76	0.79	0.83	0.85
0.2	0.48	0.50	0.57	0.64	0.72	0.75	0.81	0.85	0.89	0.91
0.3	0.42	0.47	0.58	0.62	0.68	0.76	0.81	0.88	0.93	0.96
0.4	0.39	0.43	0.48	0.61	0.70	0.78	0.82	0.88	0.94	0.98
0.5	0.41	0.44	0.53	0.63	0.69	0.78	0.81	0.88	0.95	0.99
0.6	0.39	0.44	0.49	0.57	0.66	0.71	0.80	0.88	0.93	1.00
0.7	0.40	0.44	0.52	0.62	0.64	0.72	0.81	0.85	0.93	1.00
0.8	0.42	0.46	0.53	0.59	0.66	0.73	0.77	0.86	0.92	1.00
0.9	0.43	0.46	0.54	0.59	0.63	0.68	0.76	0.81	0.91	1.00
1	0.32	0.37	0.41	0.49	0.54	0.59	0.61	0.81	0.86	1.00
AZ-Ins-LE	0.33	0.41	0.42	0.44	0.51	0.57	0.63	0.81	0.89	1.00

iterations = 10 000										
Model	Accuracy									
Frequency of EC	0.4	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	1
0	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85
0.1	0.43	0.50	0.55	0.62	0.65	0.70	0.76	0.83	0.88	0.92
0.2	0.33	0.38	0.45	0.54	0.58	0.70	0.73	0.84	0.92	0.96
0.3	0.31	0.35	0.40	0.51	0.58	0.65	0.74	0.86	0.95	0.99
0.4	0.35	0.36	0.40	0.50	0.57	0.64	0.73	0.83	0.94	1.00
0.5	0.33	0.38	0.45	0.50	0.56	0.69	0.73	0.86	0.95	1.00
0.6	0.37	0.41	0.46	0.52	0.59	0.68	0.74	0.82	0.94	1.00
0.7	0.39	0.41	0.50	0.55	0.60	0.66	0.74	0.83	0.92	1.00
0.8	0.41	0.47	0.52	0.57	0.63	0.66	0.73	0.84	0.92	1.00
0.9	0.42	0.47	0.52	0.58	0.65	0.64	0.72	0.81	0.89	1.00
1	0.34	0.37	0.42	0.49	0.50	0.58	0.62	0.81	0.90	1.00
AZ-Ins-LE	0.34	0.37	0.42	0.46	0.54	0.56	0.62	0.81	0.91	1.00

Fig. 5. The results of the function maximization problem, obtained through the proposed method with given parameters – model’s frequency of estimated certainty (how often it is used to terminate iterations) and model’s prediction accuracy – and the AlphaZero-inspired method (abbreviated as AZ-Ins-LE in the last rows). Each table corresponds to a different simulation budget for MCTS. The scores are aggregated across the three functions and normalized to the range [0, 1]. The colors light turquoise, dark blue, and gold are used to highlight instances where the proposed method surpasses the baseline MCTS, AZ-Ins-LE, or both, respectively.

A. Chess

The first experiments were performed in chess. Here, we used the *Stockfish* [43] engine to serve as the prediction model for MCTS. *Stockfish* is arguably the strongest chess engine created to date. The engine is capable of estimating the advantage of a given player in the current position. In this experiment, we use the value of this advantage as an inverted prediction uncertainty margin, since a higher margin indicates greater confidence by the engine that one player is winning. This setup ignores the uncertainty of draws, as they are treated as just uncertain positions. However, for the purposes of the experiment, this limitation can be accepted – it merely means that the model is certain slightly less frequently than it could be. Nevertheless, we can still compare the approaches using this model. Furthermore, draws between two MCTS-based players in chess are rare unless the players are setup to be very strong (performing millions of iterations per move) or a specific rule for offering and accepting draws is introduced. Without such a rule, games must be played out to endgame draws such as stalemate. We included four values for the advantage (confidence/certainty of the model): [0.5, 1.0, 1.5, 2.0]. A value of 1.0 is equivalent to an advantage of one full pawn.

In addition to the model’s confidence, we parameterized it with the time allowed for calculating chess position evaluations. We set these times relatively low: 40 and 200 milliseconds, respectively, because they affect the total experiment duration, and we prioritized the number of repetitions. Moreover, *Stockfish* is already a strong prediction model even under lower time constraints. The 200ms model produces more accurate estimations than the 40ms model. These two values allow us to measure the effect of stronger/weaker models combined with given confidence estimations. The total number of *Stockfish* engine parameterizations used is $2 \times 4 = 8$.

The results are presented in Table I. Each score represents the average result obtained over a series of 200 games by the proposed method using the model parameterization specified in the first column, compared against two baseline approaches (shown in the remaining columns) for two sets of iteration steps budgets. A player received 1.0 for a win, 0.5 for a draw, and 0.0 for a loss. For each pair of players, roles (white and black pieces) were switched after 100 games. When playing against *AZ-Ins-LE*, both players always used the same prediction model (as specified in the first column), however, the *AdvThreshold* parameter applies only to the proposed method as this is the value that the calculated advantage in the currently visited state is compared against. The *Time* parameter applies to both methods. We computed 95% confidence intervals, all of which had lengths ranging from 0.03 to 0.04.

The results show that the proposed method consistently outperforms classic MCTS by a large margin. The only exceptions occur with an advantage threshold set to 2.0 points, where scores are as low as 0.48, 0.52, 0.52, and 0.53. This suggests that a 2.0-point threshold is too restrictive, leading to the model being used too infrequently. When the conditions for using the model are relaxed, the results against the vanilla

TABLE I

THE RESULTS FOR CHESS. EACH CELL CONTAINS AN AVERAGE SCORE $\in [0, 1]$ (WHERE 0.5 DENOTES EQUAL PERFORMANCE) OF THE PROPOSED METHOD AGAINST FOUR COMPARATIVE APPROACHES. CALCULATIONS FOR 200 REPEATS, STARTING POSITIONS SWITCHED AFTER 100.

Model AdvThreshold / Time	1000 iteration steps scores versus 10 000 iteration steps			
	AZ-Ins-LE	MCTS	AZ-Ins-LE	MCTS
0.5pts / 40ms	0.58	0.78	0.56	0.72
1.0pts / 40ms	0.56	0.82	0.55	0.67
1.5pts / 40ms	0.51	0.61	0.51	0.54
2.0pts / 40ms	0.51	0.52	0.50	0.48
0.5pts / 200ms	0.53	0.89	0.54	0.81
1.0pts / 200ms	0.50	0.85	0.50	0.79
1.5pts / 200ms	0.49	0.72	0.50	0.64
2.0pts / 200ms	0.49	0.52	0.49	0.53

MCTS player improve.

When comparing against *AZ-Ins-LE*, the scores obtained by both players are generally similar, with the exception of the lowest advantage threshold settings, which significantly favor the proposed method. There is no single instance where the proposed method is significantly weaker than a comparative approach. Overall, the strongest variant is the one that utilizes the model most frequently (0.5-point certainty threshold). However, against the base MCTS, it performs best when paired with the stronger prediction model (200ms calculations), whereas against *AZ-Ins-LE*, the best results are achieved with the weaker model (40ms). This is likely because *AZ-Ins-LE* operates using the same model, and the proposed method appears to take better advantage of it.

B. Othello (Reversi)

In contrast to chess, we trained custom machine learning models for Othello. The model type of choice was a Random Forest, trained on a dataset of expert games available on Kaggle [44], played by the top-1000 players on the e-Othello platform. Game transcripts were used to reconstruct states by applying moves from the initial position. From each game, two states between the 5th and 40th moves were randomly selected as training/testing samples. The decision to include only two states was driven by the aim of minimizing correlation between the samples. The total number of states available for the training process was 51298.

To differentiate the models in terms of their efficacy, training was performed on both the complete dataset (51K) and a smaller subset (5K). For both datasets, 5-fold cross-validation (5-CV) was applied. The models achieved accuracies of 0.58 (for the 5K dataset) and 0.68 (for the 51K dataset) in predicting the winners during training with 5-CV. While these results could potentially be further optimized, model performance was not the primary focus of this study.

As with the chess experiments, the second parameter represented the condition for using the model. Here, confidence prediction intervals of the game result (1: win, 0.5: draw, 0: loss) from the perspective of a given player were calculated. For the procedure of calculating them, please refer to [45]. Three threshold values for the lengths of the prediction intervals were included in the experiments: 0.05, 0.1, and 0.2. For

each visited state, the proposed approach tested whether the model’s prediction interval exceeded the given threshold. The total number of model parameterizations used was $2 \times 3 = 6$.

TABLE II

THE RESULTS FOR OTHELLO. EACH CELL CONTAINS AN AVERAGE SCORE $\in [0, 1]$ (WHERE 0.5 DENOTES EQUAL PERFORMANCE) OF THE PROPOSED METHOD AGAINST FOUR COMPARATIVE APPROACHES. CALCULATIONS FOR 200 REPEATS, STARTING POSITIONS SWITCHED AFTER 100.

Model Pred. Interv. / TrainData	1000 iteration steps scores versus 10 000 iteration steps			
	AZ-Ins-LE	MCTS	AZ-Ins-LE	MCTS
0.05 / 5K	0.63	0.63	0.57	0.54
0.05 / 51K	0.60	0.64	0.56	0.59
0.1 / 5K	0.53	0.55	0.55	0.54
0.1 / 51K	0.51	0.61	0.50	0.55
0.2 / 5K	0.53	0.52	0.53	0.48
0.2 / 51K	0.51	0.54	0.49	0.52

The results are presented in Table II. The experimental setup, including the opponents, the number of repeats, starting positions switching, and the procedure for calculating average scores, was analogous to the chess experiments. Each score ≥ 0.54 represents a statistically significant advantage for the proposed method, which occurred in 14 out of 24 matchups. In the remaining 10 instances, 7 showed insignificant advantages. The three lowest scores were 0.50 (with the 0.1/51K model against *AZ-Ins-LE*), 0.49 (0.2/51K against *AZ-Ins-LE*), and 0.48 (0.2/5K against the baseline MCTS), all for 10000 iteration steps.

As observed in the chess experiments, a higher frequency of using the model (resulting from a smaller prediction interval length threshold) proved beneficial for the proposed approach. Additionally, the proposed method performed better with a lower number of MCTS iteration steps, which indicates its potential suitability for time-constrained scenarios. This confirms the findings from the experiments on the function-maximization problem (see Sec. V), where the proposed method gained the most significant advantage over comparative methods when operating under the lowest computational budget.

However, a limitation can be the inference (use) time of the model used with the MCTS algorithm. This does not mean that the proposed approach will be faster than other setups, particularly those that do not use any model during the search. Each case should be evaluated based on the particular time requirements of the problem.

VII. CONCLUSION

In combinatorial games, the most effective playing programs typically combine search techniques with either an explicit evaluation function or a machine learning model. A state-of-the-art example of such a search technique is Monte Carlo Tree Search. We have proposed a method that dynamically combines MCTS with score-predicting models based on the models’ confidence estimation regarding their output. Empirical experiments performed in a single-player environment as well as two multi-player games, have yielded numerous insightful findings. Most notably, in most instances, the models’ predictions need to be very accurate to justify

their inclusion over the standard, purely search-based MCTS algorithm. In our fundamental experiment, scenarios with larger computational budgets required an accuracy of at least 0.95 or 0.9 to outperform baseline approaches. Employing models less frequently but ensuring their use in scenarios where they provide high accuracy is more beneficial than frequent use of less accurate models. Particularly, models that overestimate expected game values negatively impact the player’s performance. This approach strengthens the idea behind the proposed method, which relies on a dynamic test. This is opposed to many existing approaches that consistently use evaluation models in a uniform manner, such as always at a certain depth of the tree. However, if the simulation budget is very limited and/or the baseline MCTS performance is poor, then even a moderately accurate model can perform as well as MCTS with a higher number of simulations. Results in chess and Othello demonstrate that the proposed confidence-aware MCTS method has the potential to become a highly effective approach for game-tree search, provided that a prediction model with uncertainty/confidence estimation is available. We encourage researchers to further investigate and adapt this concept within their game environments.

Future work will focus on conducting additional experiments on complex multiplayer games, such as *Go* and real-time strategy video games, exploring various settings related to model accuracy and computational budgets for the search algorithm. Independently, we aim to further investigate problems, in which the baseline MCTS configuration is very weak. In such problems, it is likely that using prediction models will be more promising. Additionally, another area left for future exploration concerns the methodologies used in training machine learning models for games, that can properly evaluate (with high confidence) whether they will be accurate in a given state.

APPENDIX A

DATA FILES FROM THE EXPERIMENTS

Data from the experiments, along with instructions on how to interpret the files, can be found at the following link: <https://github.com/MaciekArea7/data>

ACKNOWLEDGMENTS

This work has been co-funded by the Smart Growth Operational Programme 2014-2020, financed by European Regional Development Fund under GameINN project POIR.01.02.00-00-0207/20 operated by National Centre for Research and Development in Poland.

REFERENCES

- [1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [2] J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P. Lu, and S. Sutphen, “Checkers is Solved,” *science*, vol. 317, no. 5844, pp. 1518–1522, 2007.
- [3] J. McCarthy, “Chess as the Drosophila of AI,” in *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer, Eds. Springer, 1990, pp. 227–237.

- [4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [5] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. Lucas, “General video game ai: Competition, challenges and opportunities,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.
- [6] N. R. Sabar and G. Kendall, “Population based Monte Carlo Tree Search hyper-heuristic for combinatorial optimization problems,” *Information Sciences*, vol. 314, pp. 225–239, 2015.
- [7] M. H. Segler, M. Preuss, and M. P. Waller, “Planning chemical syntheses with deep neural networks and symbolic AI,” *Nature*, vol. 555, no. 7698, pp. 604–610, 2018.
- [8] S. Asta, D. Karapetyan, A. Kheiri, E. Özcan, and A. J. Parkes, “Combining Monte-Carlo and Hyper-Heuristic Methods for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem,” *Information Sciences*, vol. 373, pp. 476–498, 2016.
- [9] K. Kurzer, C. Zhou, and J. M. Zöllner, “Decentralized cooperative planning for automated vehicles with hierarchical Monte Carlo Tree Search,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 529–536.
- [10] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, “The grand challenge of computer Go: Monte Carlo Tree Search and extensions,” *Communications ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012.
- [11] R. Coulom, “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search,” in *International conference on computers and games*. Springer, 2006, pp. 72–83.
- [12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the Game of Go with Deep Neural Networks and Tree Search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [13] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, “Monte Carlo Tree Search: a review of recent modifications and applications,” *Artificial Intelligence Review*, vol. 56, pp. 2497–2562, 2023. [Online]. Available: <https://doi.org/10.1007/s10462-022-10228-y>
- [14] L. Kocsis and C. Szepesvári, “Bandit Based Monte-Carlo Planning,” in *Proceedings of the 17th European conference on Machine Learning*, ser. ECML’06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 282–293.
- [15] A. L. Samuel, “Some Studies in Machine Learning Using the Game of Checkers,” *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.
- [16] J. Von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, 2nd rev. Princeton University Press, 1947.
- [17] D. E. Knuth and R. W. Moore, “An analysis of alpha-beta pruning,” *Artificial intelligence*, vol. 6, no. 4, pp. 293–326, 1975.
- [18] L. Harris, “The heuristic search and the game of chess. a study of quiescence, sacrifices, and plan oriented play,” in *Computer chess compendium*. Springer, 1988, pp. 136–142.
- [19] M. Campbell, A. J. Hoane, and F.-h. Hsu, “Search Control Methods in Deep Blue,” in *AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, 1999, pp. 19–23.
- [20] R. Lorentz, “Using evaluation functions in Monte Carlo Tree Search,” *Theoretical computer science*, vol. 644, pp. 106–113, 2016.
- [21] R. J. Lorentz, “Amazons Discover Monte-Carlo,” in *International Conference on Computers and Games*. Springer, 2008, pp. 13–24.
- [22] M. Świechowski, D. Lewiński, and R. Tyl, “Combining Utility AI and MCTS Towards Creating Intelligent Agents in Video Games, with the Use Case of Tactical Troops: Anthracite Shift,” in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021, pp. 1–8.
- [23] K. Waleńdzik and J. Mańdziuk, “An Automatically Generated Evaluation Function in General Game Playing,” *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, pp. 258–270, 09 2014.
- [24] J. Goodman, “Re-determinizing MCTS in Hanabi,” in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.
- [25] M. Wunder, M. L. Littman, and M. Babes, “Classes of Multiagent Q-Learning Dynamics with Epsilon-Greedy Exploration,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 1167–1174.
- [26] M. Świechowski and J. Mańdziuk, “Self-Adaptation of Playing Strategies in General Game Playing,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 367–381, 2014.
- [27] J. Schaeffer, “The History Heuristic and Alpha-Beta Search Enhancements in Practice,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 11, no. 11, pp. 1203–1212, 1989.
- [28] D. J. Soemers, É. Piette, and C. Browne, “Biasing MCTS with Features for General Games,” in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 450–457.
- [29] H. Finnsson and Y. Björnsson, “CadiaPlayer: A Simulation-Based General Game Player,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 4–15, 2009.
- [30] N. Sephton, P. I. Cowling, E. Powley, and N. H. Slaven, “Heuristic Move Pruning in Monte Carlo Tree Search for the Strategic Card Game Lords of War,” in *2014 IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–7.
- [31] H. Finnsson and Y. Björnsson, “Cadiaplayer: Search-Control Techniques,” *KI-Künstliche Intelligenz*, vol. 25, no. 1, pp. 9–16, 2011.
- [32] C. D. Rosin, “Multi-Armed Bandits with Episode Context,” *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 3, pp. 203–230, 2011.
- [33] G. M. J. Chaslot, M. H. Winands, H. J. V. D. Herik, J. W. Uiterwijk, and B. Bouzy, “Progressive Strategies for Monte Carlo Tree Search,” *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.
- [34] D. J. Wu, “Designing a Winning Arimaa Program,” *ICGA Journal*, vol. 38, no. 1, pp. 19–40, 2015.
- [35] L. Kocsis, J. Uiterwijk, and J. van den Herik, “Move Ordering Using Neural Networks,” in *Engineering of Intelligent Systems: 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2001 Budapest, Hungary, June 4–7, 2001 Proceedings 14*. Springer, 2001, pp. 45–50.
- [36] P. Auer, “Finite-time Analysis of the Multiarmed Bandit Problem,” 2002.
- [37] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst, “Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron,” in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2012, pp. 242–249.
- [38] R. C. Gray, J. Zhu, and S. Ontaño, “Beyond UCT: MAB Exploration Improvements for Monte Carlo Tree Search,” in *2023 IEEE Conference on Games (CoG)*. IEEE, 2023, pp. 1–8.
- [39] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, “The Second Type of Uncertainty in Monte Carlo Tree Search,” *arXiv e-prints*, pp. arXiv–2005, 2020.
- [40] G. Tesauro, V. Rajan, and R. Segal, “Bayesian Inference in Monte-Carlo Tree Search,” in *Conference on Uncertainty in Artificial Intelligence*, 2010.
- [41] T. Dam, P. Stenger, L. Schneider, J. Pajarinen, C. D’Eramo, and O.-A. Maillard, “Monte-carlo tree search with uncertainty propagation via optimal transport,” *arXiv preprint arXiv:2309.10737*, 2023.
- [42] E. Galvan and F. V. Amenyro, “An Analysis on the Effects of Evolving the Monte Carlo Tree Search Upper Confidence for Trees Selection Policy on Unimodal, Multimodal and Deceptive Landscapes,” *arXiv preprint arXiv:2311.13609*, 2023.
- [43] T. Romstad, M. Costalba, and J. Kiiski, “Stockfish,” 2008, <https://stockfishchess.org/> (Last accessed: 5 May 2024).
- [44] A. Oliveira, 2024, <https://www.kaggle.com/datasets/andrefpoliveira/othello-games>.
- [45] D. N. Haozhe Zhang, Joshua Zimmerman and D. J. Nordman, “Random Forest Prediction Intervals,” *The American Statistician*, vol. 74, no. 4, pp. 392–406, 2020.

Maciej Świechowski was awarded a Ph.D. with distinction in Artificial Intelligence in 2015 from Systems Research Institute of the Polish Academy of Sciences. He received his M.Sc. and B.Sc. in Computer Science from Warsaw University of Technology. In 2013–2014, he was a visiting researcher at the University of New South Wales, Australia. In 2012–2014, Mr. Świechowski participated in the International General Game Playing Competition held by Stanford University, reaching the quarterfinals twice. Currently, his main positions are as the Head of QED Labs and Chief Technology Officer of QED Games. In these roles, he has gained hands-on experience in many R&D projects. In 2022, he co-founded the Information Technologies for Psychiatry Foundation, where he serves as the vice president. The foundation is a non-profit organization with the aim of promoting modern technologies for the monitoring, diagnosis, and treatment of mental disorders. He also teaches at the University of Warsaw.