Monte Carlo Tree Search-Minimax Hybrid in AlphaZero

Bachelor thesis by Thi Thanh Tam Truong
Date of submission: 04.09.2023

Examiner: Prof. Kristian Kersting
Supervisor: M.Sc. Johannes Czech
Darmstadt, Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department

TU Darmstadt

Machine Learning Group

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Thi Thanh Tam Truong, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

[illegible]

Darmstadt, 04.09.2023

Tam Truong

Abstract

AlphaZero, a program developed by DeepMind in 2017, achieved super-human performance in Chess, Shogi, and Go through self-training, showing the potential of deep reinforcement learning and Monte Carlo Tree Search(MCTS) in solving complex problems. Initially introduced for creating computer players in Go, MCTS replaced traditional methods relying on heuristics and proved effective across domains. However, in some adversarial domains like Chess, Minimax with alpha-beta pruning remains superior due to MCTS's selectivity, possibly missing critical moves. Minimax Search is a widely used algorithm for two-player zero-sum games, focusing on finding the best move while considering the opponent's optimal responses. It works well in games with complete information but struggles in complex games with large decision spaces. Previous studies proposed approaches to extract individual features of Minimax, such as Maximum Backpropagation, Implicit Max Backups, and Power-mean Backpropagation, integrate them into MCTS, or nest Minimax Search into MCTS, such as MCTS with Informed Cutoffs (MCTS-IC), and MCTS with Informed Priors(MCTS-IP). This thesis aims to recreate and refine the previous approaches in AlphaZero and then evaluate them in the Crazyhouse and Chess domains. As a result, while Maximum Backpropagation shows no good results, Implicit Minimax Backups and Power-mean Backpropagation outperform standard MCTS when the number of nodes is low. The backpropagation techniques reveal standard MCTS vulnerabilities at lower node counts, and the Power-mean PUCT method outperforms the other proposed approaches in scenarios with a high number of nodes. Combining Minimax Search pruning with MCTS (MCTS-IC) has significant impacts but is expensive.

Zusammenfassung

AlphaZero, ein von DeepMind im Jahr 2017 entwickeltes Programm, erreichte durch Selbsttraining übermenschliche Leistungen in Schach, Shogi und Go, was das Potenzial von tiefem Verstärkungslernen und Monte Carlo Tree Search (MCTS) bei der Lösung komplexer Probleme zeigt. MCTS wurde ursprünglich für die Entwicklung von Computerspielern in Go eingeführt und ersetzte traditionelle Methoden, die sich auf Heuristiken stützten, und erwies sich in allen Bereichen als effektiv. In einigen gegnerischen Domänen wie Schach bleibt Minimax mit Alpha-Beta-Beschneidung jedoch aufgrund der Selektivität von MCTS, das möglicherweise kritische Züge übersieht, überlegen. Minimax Search ist ein weit verbreiteter Algorithmus für Nullsummenspiele mit zwei Spielern, der sich darauf konzentriert, den besten Zug zu finden und dabei die optimalen Antworten des Gegners zu berücksichtigen. Er funktioniert gut bei Spielen mit vollständiger Information, hat aber Probleme bei komplexen Spielen mit großen Entscheidungsräumen. In früheren Studien wurden Ansätze vorgeschlagen, um einzelne Merkmale von Minimax zu extrahieren, wie Maximum Backpropagation, Implicit Minimax Backups und Power-mean Backpropagation, sie in MCTS zu integrieren oder Minimax Search in MCTS zu integrieren, wie MCTS with Informed Cutoffs (MCTS-IC) und MCTS with Informed Priors (MCTS-IP). Diese Arbeit zielt darauf ab, die vorherigen Ansätze in AlphaZero neu zu erstellen und zu verfeinern und sie dann in den Domänen Crazyhouse und Chess zu evaluieren. Das Ergebnis: Während Max Backpropagation keine guten Ergebnisse zeigt, übertreffen Implicit Minimax Backups und Power-mean Backpropagation das Standard-MCTS, wenn die Anzahl der Knoten gering ist. Die Backpropagationstechniken offenbaren die Schwachstellen des Standard-MCTS bei einer geringeren Anzahl von Knoten, und die Power-Mean-UCT-Methode übertrifft die anderen vorgeschlagenen Ansätze in Szenarien mit einer hohen Anzahl von Knoten. Die Kombination von Minimax Search Pruning mit MCTS (MCTS-IC) hat erhebliche Auswirkungen, ist aber teuer.

Contents

1. Introduction **7**
1.1. Motivation 7
1.2. Problem Statement 8
1.3. Contributions of this Thesis 9

2. Background **10**
2.1. Monte Carlo Tree Search (MCTS) 10
2.1.1. Upper Confidence bounds applied to Trees (UCT) 11
2.1.2. Virtual loss 11
2.1.3. Virtual visits 12
2.2. Minimax Tree Search 12
2.3. Power Mean 14

3. Related Works **16**

4. MCTS-Minimax Hybrid **18**
4.1. Maximum Backpropagation 18
4.2. Implicit Minimax Backups 19
4.3. Power-mean PUCT (Power-mean Backpropagation) 20
4.4. MCTS with Informed Cutoffs (MCTS-IC) 23
4.5. MCTS with Informed Priors (MCTS-IP) 25

5. Evaluation and Discussion **27**
5.1. Maximum Backpropagation 28
5.2. Implicit Minimax Backups 30
5.3. Power-mean PUCT (Power-mean Backpropagation) 31
5.4. MCTS with Informed Cutoffs (MCTS-IC) 35
5.5. MCTS with Informed Priors (MCTS-IP) 36

6. **Conclusion and Future Work** **38**

A. **Appendices** **39**

1. Introduction

This chapter begins by outlining the motivation of this thesis. The problem statements will be presented to define the specific issues addressed in the thesis. Lastly, the main contributions of the thesis will be summarized.

1.1. Motivation

Search algorithms are essential strategies applied to different practical areas in computer science, namely database systems, expert systems, robot control systems, and theorem provers. In game development, search algorithms, and machine learning have been studied for decades, thus resulting in highly effective optimization. Consequently, search engines are the core of game-playing systems. Several search algorithms have been proposed to improve search efficiency in many practical applications, such as the Minimax search with alpha-beta($\alpha\beta$) pruning [1], Monte Carlo Tree Search [2], etc.

AlphaZero [3] is a computer program developed by DeepMind [4], a research company owned by Google, in 2017. This single system reaches super-human performance by teaching itself from scratch how to master the games of Chess, Shogi, and Go [5]. Its success has demonstrated the potential of deep reinforcement learning and Monte Carlo Tree Search as a powerful approach to solving complex problems in various domains. MCTS has been successfully applied in multiple domains, from Go, Amazon, and Lines of Actions games to General Game Playing, planning, and optimization [6].

1.2. Problem Statement

Monte Carlo Tree Search (MCTS) is an advanced decision-making algorithm that explores combinatorial spaces represented by search trees. In these trees, nodes correspond to problem states or configurations, and edges represent transitions or actions between states. Initially proposed to create computer players in Go [7], [8], MCTS was a groundbreaking development, enabling a significant leap in performance from an average amateur level (14 kyu) to an advanced level (5 dan) [9]. Before MCTS, bots for combinatorial games relied on modified versions of the minimax alpha-beta pruning algorithm and hand-crafted heuristics [10]. In contrast, MCTS is fundamentally heuristic-based, requiring no additional knowledge beyond the game's rules, though it can incorporate heuristics to improve efficiency and convergence.

MCTS has shown considerable success in various domains [6], [11]. However, several adversarial domains, such as the games of Chess and (International) Checkers, still exist in which the traditional approach to adversarial planning, Minimax search with $\alpha\beta$ pruning [12], remains superior. The selectivity of MCTS, concentrating only on the most promising lines of play, might be a contributing factor. In tactical games such as Chess, a large number of terminal states and shallow traps exist in the search space [13]. These require precise play to avoid immediate loss, and the selective sampling and averaging value backups of MCTS can easily miss or underestimate a critical move. Conversely, MCTS could be more effective in domains such as Go, where terminal states and potential traps do not occur until the latest stage of the game. Here, MCTS can fully play out its strategic and positional understanding resulting from Monte Carlo simulations of entire games.

Minimax Search [12] is a widely used decision-making algorithm in two-player, zero-sum games such as Chess, checkers, and tic-tac-toe. Its main objective is to find the best move for a player, assuming that the opponent will also make optimal moves. The essence of Minimax lies in ensuring that the player maximizes their minimum possible outcome, considering the opponent's best responses at each step. The algorithm is particularly effective for games where all players possess complete information about the current game state, as seen in Chess, Checkers, or Tic-Tac-Toe. By exploring the entire game tree, Minimax guarantees to find the optimal move that leads to the best possible outcome. However, Minimax does have its limitations, especially when dealing with complex games that have an extensive number of possible moves and states. In such cases, the sheer size of the game tree can render an exhaustive search impractical. More advanced search algorithms like Monte Carlo Tree Search (MCTS) or deep learning approaches may address these challenges, enabling efficient decision-making in these intricate game scenarios.

Building on the understanding of MCTS and Minimax Search, numerous research endeavors [14]–[18] have focused on enhancing MCTS performance by integrating or embedding shallow Minimax Searches within its framework. Taking inspiration from these studies, this thesis aims to replicate and refine the approaches used in AlphaZero. The goal is to test and evaluate these advancements in two domains: *Crazyhouse* and *Chess*.

1.3. Contributions of this Thesis

The rest of this section summarizes the material in each of the remaining chapters, which are organized as follows. Chapter 2 serves as an introduction, establishing the foundational terminology for this thesis. It begins with a comprehensive overview of MCTS, followed by the definition of Minimax Search. Additionally, the chapter briefly introduces the Power Mean operator. In Chapter 3, related works relevant to this thesis are reviewed. Chapter 4 delves into the reproduction of MCTS-Minimax Hybrid approaches in the AlphaZero framework. The process of adaptation and implementation is elaborated upon. Chapter 5 presents the experimental results, showcasing the efficiency and quality of the proposed methods through empirical evidence. Lastly, Chapter 6 contains the conclusion of this thesis and potential works for future research.

2. Background

This section explores the foundations of three essential game-playing algorithms: the Monte Carlo Tree Search (MCTS), the Minimax Tree Search, and the innovative combination of MCTS and Minimax Search approaches.

2.1. Monte Carlo Tree Search (MCTS)

MCTS is the best-first approach to tree search. There are many extension versions of MCTS, such as MCTS-Solver [19] or Monte-Carlo Graph Search (MCGS) [20]. Due to space limitations, only the classical version of MCTS is discussed. It repeats the following four-phase loop until computation time runs out [21]. Each iteration of the loop represents one simulated game.

- **Selection phase:** The tree traversal in Monte Carlo Tree Search begins at the root and moves towards one of its leaf nodes. A selection policy determines which move to sample at each state. The selection policy aims to balance exploiting states with high-value estimates and exploring states with uncertain value estimates.
- **Expansion phase:** When the selection phase does not reach a terminal state, the expansion phase involves adding at least one new child node to the memory's tree representation. This newly added node corresponds to a form achieved by executing the last action in the selection phase. However, in the rare event that expansion does reach the terminal state, the current iteration bypasses additional steps and proceeds directly to backpropagation.
- **Simulation phase:** The rollout process involves simulating actions from the current state to the end of the episode, employing either random actions or a heuristic. In some instances, multiple rollouts may be utilized to effectively evaluate a single leaf node.

- **Backpropagation phase:** It propagates the payoffs for all modeled agents in the game back to all nodes along the path from the last visited node in the tree (the leaf one) to the root. The statistics are updated.

2.1.1. Upper Confidence bounds applied to Trees (UCT)

The UCT algorithm [7] builds on Upper Confidence Index 1(UCB1) [22], an algorithm designed to optimally trade-off exploration and exploitation in multi-armed bandit problems. At each time step t and state s_t, a fresh action a_t is chosen using the UCT-formula 2.1 until reaching either a novel unexplored state s^* or a terminal node s_T.

$$\begin{aligned} a_t &= \text{argmax}_a(Q(s_t, a) + U(s_t, a)) \\ \text{where } U(s, a) &= c_{puct} P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}. \end{aligned} \tag{2.1}$$

The new unexplored state s^* is expanded and assessed using the neural network f_θ. Subsequently, the predicted policy $P(s, a_i)$ is applied to all potential actions a_i, and the state evaluation v^* is propagated back through the traversed search trajectory. For a concluding state s_T, a consistent appraisal of either -1, +1, or 0 is employed. The value assessment v^* is periodically negated and integrated into the corresponding Q-values from the leaf node to the root as Equation 2.2 in the backup phase.

$$Q'(s_t, a) = Q(s_t, a) + \frac{1}{n}[v^* - Q(s_t, a)]. \tag{2.2}$$

2.1.2. Virtual loss

Virtual loss is introduced as a technique to address redundancy in tree exploration during parallelized MCTS. When multiple threads start from the root and traverse the tree, the same paths or neighboring leaf nodes may be explored. Since the search tree can have millions of nodes, exploring a small portion of the tree multiple times can be inefficient. *Virtual loss* is assigned to a node when a thread visit is proposed to mitigate this redundancy.

That means that the Q-value of the node is decreased, and the visit counter is increased. When the *virtual loss* equals 1, the Q-value of a specific element undergoes an update as if an additional loss had been encountered within its corresponding subtree. The following thread will only select the same node if its value remains better than the values of its siblings. The *virtual loss* is removed when the line assigned the *virtual loss* starts propagating the result of the finished simulated game. By using *virtual loss*, all threads will still explore nodes that are better than others. However, nodes for uncertain value will not be examined by more than one thread, reducing redundancy and improving the balance between exploration and exploitation in a parallelized MCTS program.

2.1.3. Virtual visits

Based on the concept of *virtual loss*, *virtual visits* offers another efficient approach to reducing redundancy and improving exploration-exploitation balance in parallelized MCTS. In contrast to *virtual loss*, *virtual visits* is applied to the selected node without changing its mean value. This relieves the complexity of calculating the original mean value for updating a new value during the backpropagate phase.

By using *virtual visits*, the overall efficiency and effectiveness of the parallelized MCTS program are substantially enhanced, providing a more refined and robust exploration process.

2.2. Minimax Tree Search

Minimax Search is a widely used algorithm in artificial intelligence and game theory, specifically for two-player, zero-sum games. It determines the best possible move for a player in a given game position, assuming both players are playing optimally. The algorithm aims to maximize the player's potential gain while minimizing the opponent's potential gain. The key idea behind the Minimax Search is to construct a game tree that represents all possible moves and their consequences up to a certain depth or horizon. The algorithm then evaluates the leaf nodes of the tree using a static evaluation function, which estimates the desirability of each game position for the player. The procedure of Minimax Search is summarized in the following Algorithm 1.
Additionally, Minimax Search can be optimized with alpha-beta pruning [12], which significantly reduces the number of nodes explored in the game tree, making it more

efficient and practical for complex games with large branching factors. Negamax is a variation of the Minimax Search algorithm for finding the optimal move in two-player, zero-sum games. It is a more compact and elegant way of writing the Minimax Search algorithm. Negamax takes advantage of the property that in two-player zero-sum games, the value of a player's move is the negation of the opponent's move value. Algorithm 2 summarizes the Negamax with alpha-beta pruning Algorithm.

Algorithm 1 Recursive Minimax

```
procedure MINIMAX($S$,$Maximizing = True$)
    if $S$ is terminal then
        return UTILITY(S)
    end if
    if $Maximizing = True$ then
        $v \leftarrow -\infty$
        for each $child \in S$ do
            $v \leftarrow \max(v, \text{MINIMAX}(child, False))$
        end for
        return $v$
    else
        $v \leftarrow +\infty$
        for each $child \in S$ do
            $v \leftarrow \min(v, \text{MINIMAX}(child, True))$
        end for
        return $v$
    end if
end procedure
```

Algorithm 2 Negamax with alpha-beta pruning

```
1: procedure NEGAMAX($node$,$depth$, $alpha$, $beta$)
2:     if $depth = 0$ or node is a terminal node then
3:         return the heuristic value of node
4:     end if
5:     $childNodes \leftarrow$ GENARATEMOVES($node$)
6:     $childNodes \leftarrow$ ORDERMOVES($childNodes$)
7:     $v \leftarrow -\infty$
8:     for each $child \in childNodes$ do
9:         $v \leftarrow \max(v, -\text{NEGAMAX}(child, depth - 1, -\beta, -\alpha))$
10:        $\alpha \leftarrow \max(\alpha, v)$
11:        if $\alpha \geq \beta$ then
12:            break (*cut-off*)
13:        end if
14:    end for
15:    return $v$
16: end procedure
```

2.3. Power Mean

The *Power Mean* [23] is an operator to combine numbers specially. It includes different types of averages like arithmetic, geometric, and harmonic means, depending on a value called p. Setting $p = 1$ yields the weighted arithmetic mean. When $p \to 0$, the result is the geometric mean; with $p = -1$, it becomes the harmonic mean. Additionally, the minimum value is achieved by $p \to -\infty$ (2.4), while $p \to +\infty$ leads to the maximum value (2.5).

Definition 1. *For a sequence of positive numbers $X = (X_1, \ldots, X_n)$ and positive weights $w = (w_1, \ldots, w_n)$, the power mean of order p (p is an extended real number) is defined as Equation 2.3:*

$$M_n^{[p]}(X, w) = \left(\frac{\sum_{i=1}^{n} w_i X_i^p}{\sum_{i=1}^{n} w_i} \right)^{\frac{1}{p}}. \tag{2.3}$$

$$M_n^{[-\infty]}(X, w) = \lim_{p \to -\infty} M_n^{[p]}(X, w) = \text{Min}(X1, \ldots, X_n). \tag{2.4}$$

$$M_n^{[+\infty]}(X, w) = \lim_{p \to +\infty} M_n^{[p]}(X, w) = \text{Max}(X1, \ldots, X_n). \tag{2.5}$$

It has been proven that $M_n^{[p]}(X, w)$ is an increasing function(2.6), and it can be upper bound by average mean plus with a constant [23].

$$M_n^{[1]}(X, w) \leq M_n^{[q]}(X, w) \leq M_n^{[p]}(X, w), \forall p \geq q \geq 1. \tag{2.6}$$

3. Related Works

Previous work on developing algorithms influenced by MCTS and Minimax Search has taken two principal approaches. On the one hand, one can extract individual features of Minimax, such as Minimax-style backups, and integrate them into MCTS. The first technique of Minimax-influenced backup rules in the simulation-based MCTS framework was Coulom's original Maximum Backpropagation [16]. This method suggests after several simulations of a node have been reached, switching to the propagation of the maximum value instead of the simulated (average) value. The motivation is that after a certain point, the search algorithm should consider a node converged and return an estimate of the best value. This backpropagation has also lately been used in other Monte-carlo search algorithms and demonstrated success in probabilistic planning, as an alternative type of forecaster in BRUE [24], and as Bellman backups for online dynamic programming in Trial-based Heuristic Tree Search [25]. Later, Marc Lanctot et at. [17] proposed using Implicit Minimax Backups to strengthen play performance in Kalah [26], Breakthrough [27], and Lines of Action [28]. This approach uses heuristic evaluations to guide the MCTS search by storing the two sources of information, estimated win rates and heuristic evaluations, separately. Both minimaxing backups of heuristic evaluations and averaging backups of rollout returns are managed simultaneously.
On the other hand, one can nest Minimax Search into MCTS. The study of Hendrik Baier et al. [14] discusses three approaches to integrate Minimax with pruning into the MCTS framework. These are Minimax in the Rollout Phase(MCTS-MR), Minimax in the Selection and Expansion Phases (MCTS-MS), and Minimax in the Backpropagation Phase (MCTS-MB). A novel variant, MCTS-MS, outperforms regular MCTS in games like Catch the Lion, Breakthrough, and Connect-4 [29], while MCTS-MB excels in Catch the Lion and Breakthrough. MCTS-MR is strong in Catch the Lion [30] and Connect-4 but weaker in Breakthrough, likely due to differences in average branching factors. The study suggests that MCTS-minimax hybrids are less effective in games like Othello [31] and Go. The success of MCTS-minimax hybrids is influenced by trap density and difficulty. MCTS-MR is suited for low branching factor domains, while MCTS-MS and MCTS-MB handle higher

branching factors. Baier et al. [14] proposed incorporating domain knowledge into hybrids through evaluation functions or moving to order. Challenges arise in combining heuristic evaluations with rollout results. The study indicates that move ordering could be effective, especially in games like Breakthrough.

Another study by Baier et al. [32] delves into the research on MCTS-minimax hybrids, focusing on cases where heuristic evaluation functions are available as domain knowledge. Three approaches were examined for incorporating this knowledge into MCTS: MCTS-IR enhances the rollout policy, MCTS-IC employs it for early rollout termination, and MCTS-IP utilizes it as prior for tree nodes. The study introduced enhancements such as move ordering and k-best pruning, improving hybrid players like MCTS-IR-M-k, MCTS-IC-M-k, and MCTS-IP-M-k. Experiments revealed that enhanced Minimax (MCTS-IP-M-k) improved MCTS performance across domains, particularly in low-time settings. Additionally, combining MCTS-IP-M-k with minimax rollouts led to further enhancements. Notably, the best-performing hybrid surpassed a simple $\alpha\beta$ implementation in Breakthrough, showcasing the potential strength of integrating MCTS and Minimax. Different directions for research were suggested, including exploring additional $\alpha\beta$ enhancements, optimizing the exploitation of heuristic knowledge, studying domain-specific influences, evaluating the quality of evaluation and move ordering functions, and using artificial game trees to isolate specific effects.

Furthermore, some studies try to improve the UCT algorithm in MCTS. Khandelwal et al. [18] formalized and analyzed different on-policy and off-policy complex backup approaches for MCTS planning based on techniques in the Reinforcement Learning literature. In this study, four novel MCTS variants are proposed: MCTS(λ), MaxMCTS(λ), MCTSγ, and MaxMCTSγ. Through benchmarking with various IPC domains, the authors demonstrate that the choice of backup strategy significantly impacts performance. Among the proposed approaches, MaxMCTSγ is parameter-free and performs as well as or better than Monte Carlo backups in all tested domains. Another approach, MaxMCTS(λ), outperforms all other proposed and existing MCTS variants. It is observed that an intermediate value of λ between 0 and 1 is necessary for optimal performance, unlike existing algorithms that use extreme values. The authors hypothesize that MaxMCTS(λ) with a low λ value excels in domains with infrequent high-reward trajectories and provide empirical evidence to support this hypothesis. However, they do not provide any proof of convergence. Meanwhile, Tuan Dam et al. [15] proposed a novel backup strategy called Power-UCT. This approach uses the power mean operator [33], which computes a value between the average and maximum value.

4. MCTS-Minimax Hybrid

This thesis primarily focuses on reproducing and enhancing the methodologies of Maximum Backpropagation (4.1), Implicit Minimax Backups (4.2), Power-mean PUCT (4.3), MCTS with Informed Cutoffs (4.4) and MCTS with Informed Priors (4.5). The ultimate goal was to optimize the performance of AlphaZero and evaluate its effectiveness in two diverse domains: Chess and Crazyhouse.

Throughout this research, a dedicated effort was made to comprehensively explore and adapt the existing approaches to suit the specific requirements of AlphaZero. This involved integrating novel insights and refining strategies to enhance its capabilities further. The intent was to contribute significantly to the ongoing advancements in AI-powered gaming agents and to optimize their decision-making processes. This section elaborates on the process of recreating and optimizing the previous methodologies in CrazyAra while simultaneously determining the optimal threshold for the two domains: Chess and Crazyhouse. This comprehensive analysis aimed to shed light on the performance and adaptability of AlphaZero in different gaming scenarios.

MCTS-Minimax hybrid approaches are the combination of both MCTS and minimax. These approaches involve extracting particular features of minimax, such as minimax-style backups(Maximum Backpropagation, Implicit Minimax Backups, and Power-mean PUCT), and seamlessly integrating them into MCTS. Another technique explored in this context is the nesting of minimax searches within MCTS searches.

4.1. Maximum Backpropagation

Maximum Backpropagation[16] is a technique employed in the backpropagation phase of the Monte Carlo Tree Search (MCTS) algorithm. It involves updating the value of a node by selecting the maximum value among its child nodes. Equation 4.1 shows the updated form of a node with Maximum Backpropagation. This update is triggered once the node has reached a specific number of visits. If the visit count is below the threshold, the Mean Backpropagation algorithm is used instead. By implementing this approach, the node's value is determined based on the highest value observed among its children, thereby increasing the overall effectiveness of the backpropagation process within MCTS.

$$\widehat{V}(s,a) = max_{a \in A(s)} Q(s,a). \tag{4.1}$$

The primary rationale for switching from the average operator to the maximum operator is considering the number of simulations. The average operator is typically employed when the number of simulations is low. This conservative approach updates the nodes cautiously due to the limited number of samples available. Conversely, the maximum operator is favored when the number of simulations is high [15]. This choice allows for a more decisive update, leveraging the plenty of simulations to make more confident decisions. This incorporation of thresholds provides better control and fine-tuning of the algorithm, resulting in improved overall performance.

4.2. Implicit Minimax Backups

Implicit Minimax Backups [17] is another strategy utilized during the backpropagation phase of the MCTS algorithm. This technique incorporates a supplementary value into each node to enhance the estimator value of a state-action pair (s, a). Specifically, this new value at the node s is a heuristic minimax value derived from evaluating subtrees beneath s. As the backpropagation progresses, the updates of r_s and n_s are performed in the usual way, and the additional value v_s^τ is updated using the minimax backup rule based on children's values. For the selection phase, we use $\widehat{Q}^{IM}$ in Equation 4.2

$$\widehat{Q}^{IM}(s,a) = (1-\alpha)\frac{r_{s,a}^\tau}{n_{s,a}} + \alpha v_{s,a}^\tau. \tag{4.2}$$

where α weights the influence of the heuristic minimax value. The threshold for the algorithm will be determined based on the minimax weight.

4.3. Power-mean PUCT (Power-mean Backpropagation)

Typically, the average backup is favored when there are a low number of simulations, ensuring a conservative update of nodes due to limited samples. Conversely, the maximum operator is preferred with a high number of simulations. Therefore, Tuan Dam et al.[15] introduced a novel backup strategy that utilizes the power mean operator. This strategy calculates a value that falls between the average and maximum values to get both advantages. The Q-Value is calculated as Equation 4.3, where s is the current state, and s' is the next state.

$$\widehat{Q}^{PM}(s,a) = \left(\sum_{a} \frac{n(s',a)}{n(s,a)} \widehat{Q}^{PM}(s',a)^p \right)^{\frac{1}{p}}. \tag{4.3}$$

The original Power Mean PUCT method assumes that all tree values are positive. However, in AlphaZero, the tree values lie within the range of [-1, 1] or [0, 1]. This presents a significant challenge when calculating the value using the exponent because of the pow() function's limitation with negative values. To address this issue, a proposed solution involves shifting the values to the range [0, 2], performing the necessary calculations, and then shifting back the result to the range [-1, 1] before updating the new value. Additionally, Tuan Dam et al. [15] have presented proof of convergence for Power-mean PUCT, which ensures convergence for any finite value of p. Hence, applying value shifting when employing the Power-Mean PUCT method for the tree is reasonable.

$$\widehat{Q}^{PM}(s,a) = \left(\frac{(1+r_0)^p}{n(s,a)} + \sum_{a} \frac{n(s',a)}{n(s,a)} (1 + \widehat{Q}^{PM}(s',a))^p \right)^{\frac{1}{p}} - 1. \tag{4.4}$$

When adapting AlphaZero's requirements and data structure for calculating power mean PUCT, it becomes essential to include a node's initial value r_0 and aggregate all the values from its child nodes. This ensures that Power Mean PUCT will simplify to mean PUCT when p = 1. The Equation 4.4 is the extended version of Equation 4.3 in AlphaZero.
A straightforward and efficient solution to compute the power mean value in AlphaZero is presented without the necessity of traversing all child nodes during backpropagation. The approach involves utilizing an additional value $t(s,a)$, which is calculated as shown in Equation 4.6.

$$\widehat{Q}^{PM}(s,a) = \left(\frac{t(s,a)}{n(s,a)}\right)^{\frac{1}{p}} - 1. \tag{4.5}$$

$$t(s,a) = r_0^p + \sum_a n(s',a)\big(\widehat{Q}^{PM}(s',a)\big)^p. \tag{4.6}$$

Upon expanding a new child node, its value will be directly integrated into the parent node's $t(s,a)$ value. However, suppose the child node has been previously expanded. In that case, the older value and its respective visit count will be replaced by the updated value and the current visit count in the parent node's $t(s,a)$ value. The entire process is summarized in Algorithm 3.

Power-mean PUCT demonstrates promising performance under limited simulation numbers. Nevertheless, as the number of simulations increases, its effectiveness diminishes due to the growing computational complexity. This work introduces two novel approaches that address this limitation by combining Power-mean PUCT with mean PUCT or maximum value updates. Initially, Power-Mean PUCT is employed, transitioning to mean PUCT or maximum value updating once a specific number of visits is reached.

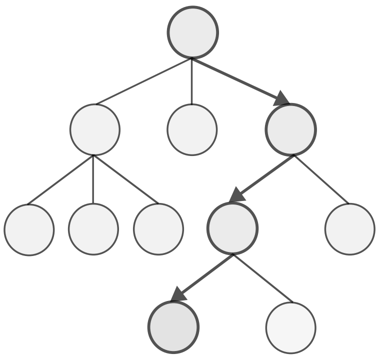
Algorithm 3 Power-PUCT in Alphazero

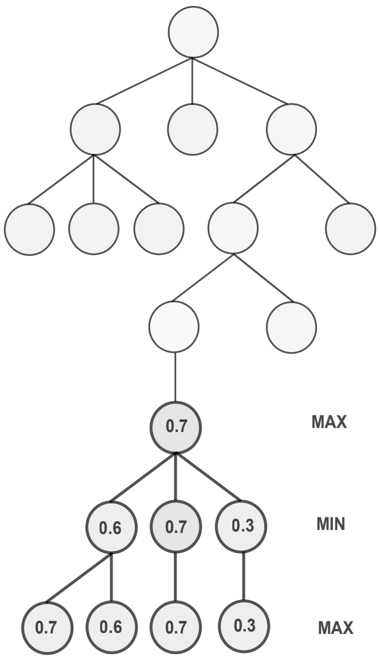
```
1:  procedure SELECT ACTION(s)
2:      return $\text{argmax}_a(\widehat{Q}^{PM}(s,a) + PUCT(s,a))$
3:  end procedure
4:
5:  procedure SIMUALTE($s_{\text{parent}}$,$a_{\text{parent}}$,s)
6:      if $\exists a \in A(s), s' = T(s,a) \notin G$ then
7:          EXPAND$(s)$
8:          $r \leftarrow$ PLAYOUT$(s)$
9:          $t_s \leftarrow r^p$
10:         $t_{s_{\text{parent}}} \leftarrow t_{s_{\text{parent}}} + r^p$
11:         $\widehat{Q}^{PM}(s,a) = r$
12:         $n_s \leftarrow 1$
13:     else
14:         if $s \in Z$ then
15:             return $\text{R}(s', a, s)$
16:         end if
17:         $a \leftarrow$ SELECT ACTION$(s)$
18:         $s' \leftarrow T(s,a)$
19:         $t_{s_{\text{parent}}} \leftarrow t_{s_{\text{parent}}} - n_s\widehat{Q}^{PM}(s,a)$
20:         SIMULATE$(s, a, s')$
21:         $n_s \leftarrow n_s + 1$
22:         $\widehat{Q}^{PM}(s,a) \leftarrow (\frac{t_s}{n_s})^{\frac{1}{p}}$
23:         $t_{s_{\text{parent}}} \leftarrow t_{s_{\text{parent}}} + n_s\widehat{Q}^{PM}(s,a)$
24:     end if
25: end procedure
26:
27: procedure MCTS$(s_0)$
28:     while time left do
29:         SIMULATE$(-, -, s_0)$
30:     end while
31:     return $\text{argmax}_{a \in A(s_0)} n_{s_0,a}$
32: end procedure
```

4.4. MCTS with Informed Cutoffs (MCTS-IC)

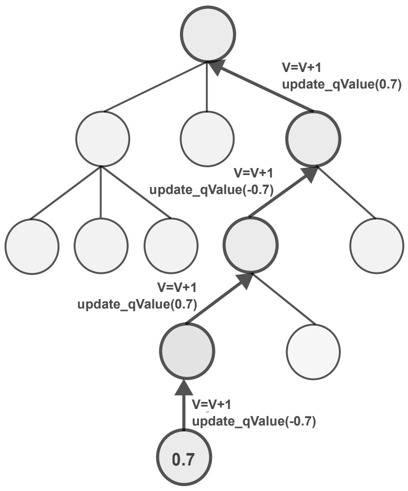
The concept of rollout cutoffs involves stopping a simulation early if it becomes possible to predict the winner or advantaged player using an evaluation function. This helps avoid noise from extended rollouts. Instead of completing the full rollout, the current state is evaluated, and the result is propagated. This can increase sampling speed if the evaluation function is quicker than finishing the rollout. Some rollouts are played before evaluation to introduce diversity. Baier et al. [32] proposed MCTS with the Informed Cuttoffs approach, which avoids determinism by randomly selecting equally valued moves. They also suggested an extension involving a depth-d Minimax Search for backpropagation value determination, which is called MCTS-IC-M.
Since both Minimax Search and MCTS in the original MCTS-IC-M use the same heuristic evaluation and involve making random moves during rollouts, an attempt was made to incorporate deep neural network evaluation, which is commonly used in the Alphazero framework, into Minimax Search, similar to the approach taken in our MCTS strategy. However, the significant computational load and time limitations linked to deep neural network evaluation were acknowledged, resulting in its replacement with the more efficient Stockfish-12[1] classical handcrafted evaluation in the Minimax Search process. This decision was motivated by Stockfish's impressive search speed and efficient pruning mechanisms. Stockfish's prowess can be attributed to its extensive opening book and vast database of human grandmaster games, which augment its already formidable gameplay. The engine's training with traditional chess knowledge and algorithms further bolsters its performance. To speed up the Minimax Search, we use the Negamax algorithm instead.

[1]`https://github.com/official-stockfish/Stockfish`


(a) The selection phase.



(b) The resulting position is evaluated with a d = 2 minimax search. The heuristic evaluation is 0.7 for the maximizing player.



(c) This value is backpropagated with mean backup.

Figure 4.1.: The MCTS-IC hybrid with depth $d = 2$.

4.5. MCTS with Informed Priors (MCTS-IP)

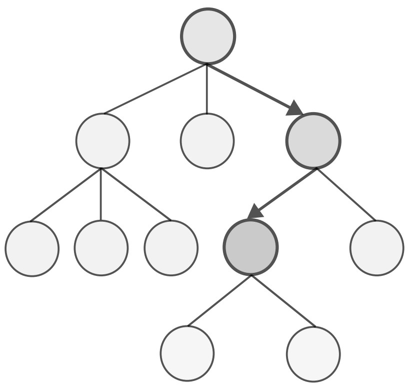
Using node priors is presented as a method to support the selection policy in MCTS (Monte Carlo Tree Search) by incorporating heuristic information. Upon adding a new node to the tree or after a certain number of visits, the corresponding state's heuristic evaluation is stored as virtual wins and losses, weighted by a prior weight parameter w. This technique facilitates tree growth in a promising direction, given that heuristic evaluations are more reliable than MCTS value estimates derived from limited samples. However, as nodes are visited frequently, the influence of the prior diminishes over time, and MCTS rollouts progressively take precedence over the heuristic evaluation. Baier et al. [32] introduced two approaches: MCTS-IP-E, which employs heuristic evaluations, and MCTS-IP-M, which extends the method MCTS-IP-E with a depth-d minimax search to compute more precise prior values, thereby enhancing the selection policy in the MCTS tree.
An advanced version of the MCTS-IP-M algorithm has been developed, and specific modifications have been implemented to enhance its performance significantly. Instead of relying on the evaluation method used in the original MCTS, the robust evaluation of Stockfish is strategically used.

In contrast to the MCTS-IP-M proposal by Baier et al., the Minimax Search is strategically activated only after a node has accumulated a specific number of visits. After a node has been visited n times, the heuristic evaluation h of the corresponding state is stored in this node with a prior weight parameter γ. The following formulas (4.7, 4.8) show how to update the visit (n) counters and the value ($qValue$) of the node at hand.

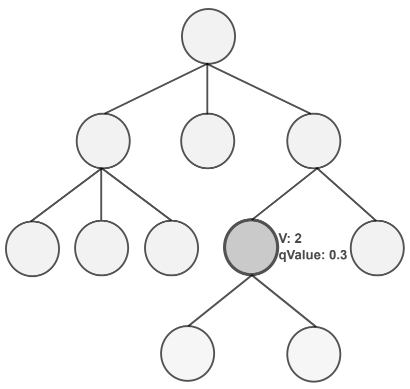
$$n' \leftarrow n + \gamma. \tag{4.7}$$

$$qValue' \leftarrow \frac{qValue \cdot n + \gamma \cdot h}{n'}. \tag{4.8}$$

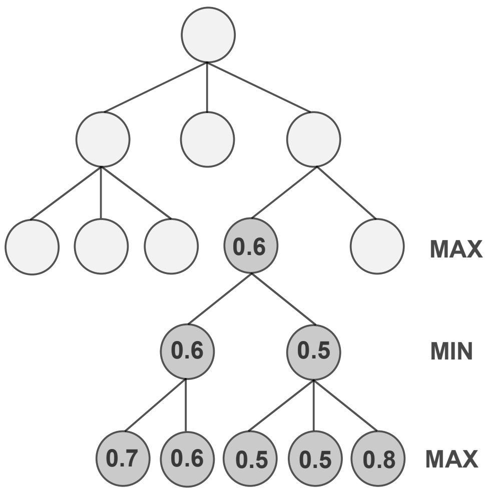
An example is shown in Figure 4.2.



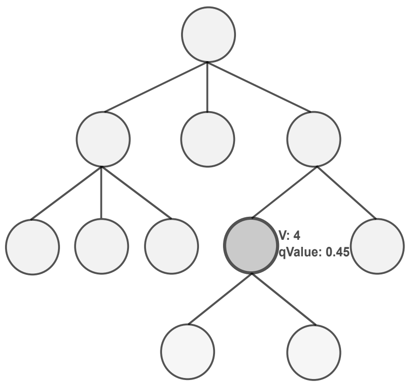
(a) The selection phase.



(b) A tree node with $v = n = 2$ visits is encountered



(c) This triggers a $d = 2$ minimax search. The heuristic evaluation is $v = 0.6$ for the maximizing player.



(d) This value is stored in the node in form of $\gamma = 2$ virtual visits, $n' = n + \gamma = 2 + 2 = 4$ and $qValue = \frac{n \cdot 0.3 + \gamma \cdot 0.6}{n'} = \frac{2 \cdot 0.3 + 2 \cdot 0.6}{4} = 0.45$.

Figure 4.2.: The MCTS-IP-M hybrid. $n = 2$, $d = 2$, and $\gamma = 2$.

5. Evaluation and Discussion

The algorithms were tested in two domains: *Chess* and *Crazyhouse*. These are all deterministic perfect-information turn-taking zero-sum games. To assess the algorithms, tournaments with each proposed approach against the standard MCTS with virtual visits were executed under identical environment settings using Cutechess-cli[1]. The match outcomes were evaluated using the ELO rating system [34], with runtime excluded from this evaluation.

Chess is a two-player strategy board game played on a square board divided into 64 squares of alternating colors. Each player controls an army of 16 pieces consisting of one king, queen, rooks, knights, bishops, and pawns. The game's objective is to checkmate the opponent's king, which means putting the king into a position where it is under attack and cannot escape capture.

Crazyhouse, also known as drop chess, allows players to reintroduce captured pieces, which switch sides and are kept in the capturing player's pocket. The game follows classical chess rules, including castling and en passant capture. Instead of a regular move, players can drop pocket pieces onto empty squares, except for pawns on the first or eighth rank. This feature resembles shogi but with the unique twist that pawns can be dropped for an immediate checkmate. Crazyhouse's tactical depth is higher than conventional Chess due to pieces never leaving the game, leading to more checkmates and shorter game lengths. The ability to drop pieces for checks allows for long forced checkmating sequences. The game is widespread, especially in online communities, and is often played with short-time controls.

This section is organized as follows. Experimental results for Maximum Backprobagation (5.1), Implicit Minimax Backups (5.2), Power-Mean PUCT (5.3), and MCTS-IP (5.5) in 2 domains are presented.

[1]`https://github.com/cutechess/cutechess`

5.1. Maximum Backpropagation

Initially, the focus was on examining the Q-value update, limited to utilizing only the maximum value from child nodes. This examination was conducted without using a transition from mean value to max value after multiple visits. Figure 5.1 shows the performance of this approach in Chess and Crazyhouse. When working with a limited number of nodes, exclusively employing max backpropagation outperforms the standard MCTS approach only in the Chess domain. Nevertheless, as the node count rises, there is a progressive decline in Elo ratings, and the standard MCTS is outperformed.
Subsequently, the Maximum Backpropagation concept was tested by transitioning the parameter n from the set $\{700, 800, 900, 1000, 1100\}$ of switching from mean values to maximum values. It is assumed that the average backup is usually used when the number of simulations is low for a conservative update of the nodes due to the lack of samples. Conversely, the maximum operator is favored when the number of simulations is high [15]. Hence, Maximum Backpropagation was evaluated with an initial parameter value of $n = 700$ and a tournament involving 1500 nodes. The outcomes are depicted in Figures 5.2a and 5.2b. When the switching number is elevated, the corresponding Elo Rating exceeds that of lower switching numbers when considering the same number of nodes. Specifically, as the number of nodes and simulations increases, the basic MCTS approach outperforms the Max Backpropagation strategy. These findings emphasize that Max Backpropagation does not yield favorable results in Chess and Crazyhouse scenarios. Furthermore, a higher switching number leads to a higher Elo Rating due to the limited usage of the maximum value, resulting in reduced adverse impact on the agent.
The Maximum Backpropagation doesn't perform effectively because, during the Backpropagation phase, the maximum value of child nodes replaces the average Q-value. This means that a node will keep getting chosen in the selection phase if its Q-value remains the highest. As a result, the search tree keeps favoring the same exploration path, potentially overlooking critical moves. The scalability analysis of Silver et al. [3] indicates that AlphaZero's MCTS strategy outperforms Stockfish's alpha-beta search in terms of scalability with increasing thinking time, challenging the traditional belief that alpha-beta search is inherently superior in Chess and shogi domains. This means that in the Chess and shogi domains, standard MCTS with the average backup outperforms MCTS with the maximum operator when the number of nodes and simulations is high. This also strengthens our findings.

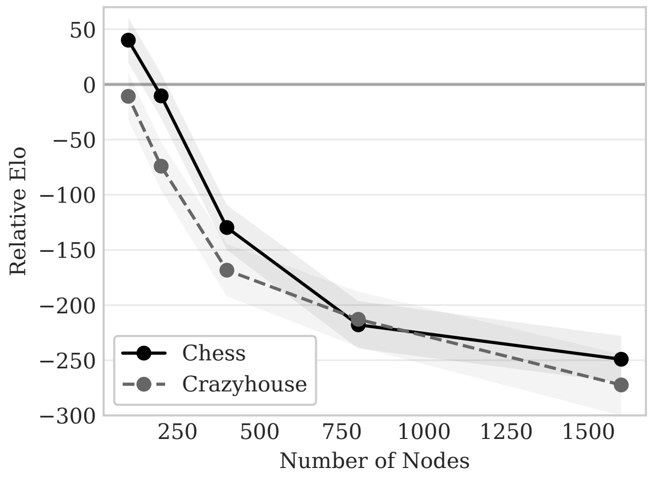
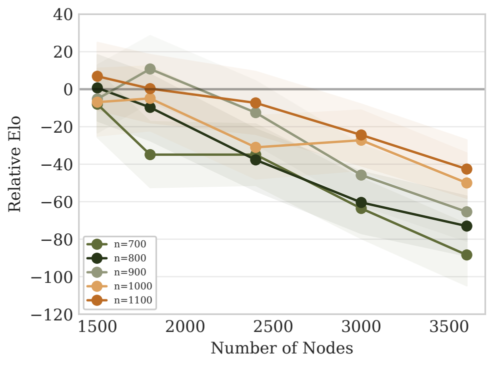
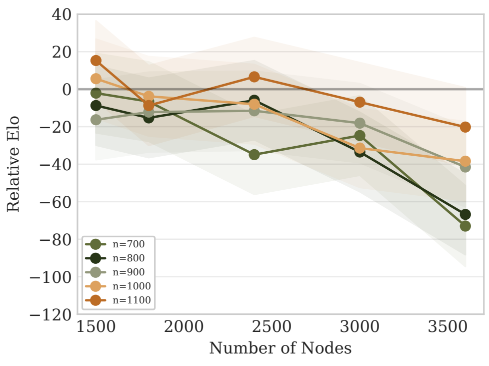


Figure 5.1.: The Comparison of the Elo development concerning the count of MCTS nodes between Chess and Crazyhouse while retaining maximum values instead of transitioning from mean values to maximum values. With a limited node count, this approach outperforms standard MCTS in Chess, but higher node counts decrease Elo ratings.



(a) Max Backpropagation in Chess.

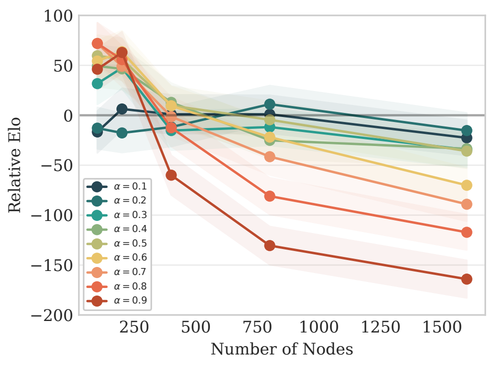


(b) Max Backpropagation in Crazyhouse.

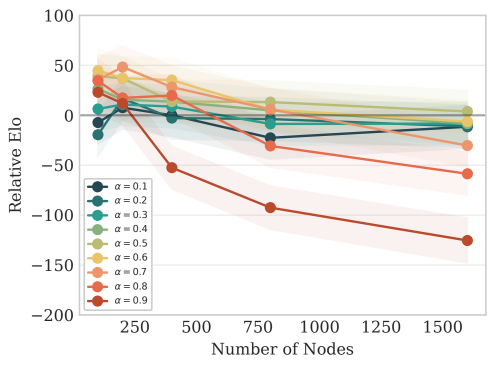
Figure 5.2.: Elo development of Maximum Backpropagation for different numbers of visits to switch from mean value to maximum value in Chess and Crazyhouse. With a high switching number, the Elo Rating is higher than the Elo Rating of lower switching numbers when considering the same number of nodes.

5.2. Implicit Minimax Backups

Implicit Minimax Backups were assessed using various minimax weight (α) parameters within the range of [0.1, ..., 0.9]. The Elo progression for each α is visualized in Figure 5.3a for Chess and Figure 5.3b for Crazyhouse. Notably, the Implicit Max Backups strategy performs better when the number of nodes and simulations remains limited. The Elo rating peak 71.9 is particularly noteworthy, achieved with $\alpha \in \{0.8, 0.9\}$ for 100 nodes in Chess. However, this performance decreases swiftly as the number of Nodes increases. For $\alpha \in \{0.4, 0.5, 0.6\}$, the Elo ratings aren't as great as those within the scope of $\{0.8, 0.9\}$. Yet, they show more stability, with Elo ratings slowly declining as the number of nodes increases. In contrast to the Chess domain, Crazyhouse experiences a less pronounced Elo increase with $\alpha \in \{0.8, 0.9\}$. For $\alpha \in \{0.5, 0.6, 0.7\}$, the Implicit Minimax Backups approach outperforms the competition up to 800 nodes. Implicit Minimax Backups generally performs strongly when the number of nodes is limited in both Chess and Crazyhouse domains.



(a) Implicit Minimax Backups in Chess.



(b) Implicit Minimax Backups in Crazyhouse.

Figure 5.3.: Elo development of Implicit Minimax Backups in Chess and Crazyhouse. The approach shows strong performance when the number of nodes is low in both domains.

The study of Marc Lanctot et at. [17] acknowledges that while it achieved positive results across various domains, the technique's broad applicability is constrained by its dependence on domain-specific tactical and strategic insights acquired through implicit minimax and playouts. This limitation becomes evident in the experiments conducted with Chinese Checkers and Hearts, underscoring the need for continued investigation employing more advanced evaluation functions. As a result, it's reasonable that the technique might be less effective in domains such as Chess and Crazyhouse. Nevertheless, it's important to note that the Implicit Minimax Backups approach performs exceptionally well when node counts are low, and implementing the Implicit Minimax Backups algorithm is straightforward.

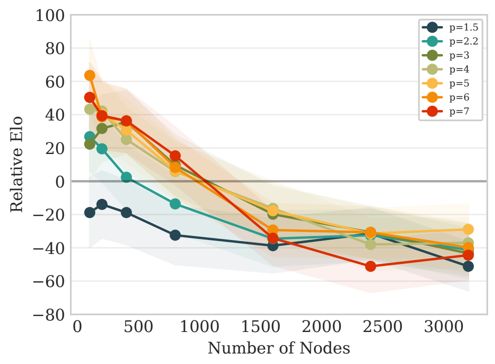
5.3. Power-mean PUCT (Power-mean Backpropagation)

Power-mean Backpropagation was evaluated using power mean (p) values from the set $\{1.5, 2.2, 3, 4, 5, 6, 7\}$. Initially, 1.5 and 2.2 were chosen based on the recommendations of Tuan Dam et al. [15]. Subsequently, higher power mean values ranging from 3 to 7 were explored to determine the optimal setting. The results, including the Chess and Crazyhouse domains, are presented in Figure 5.4.

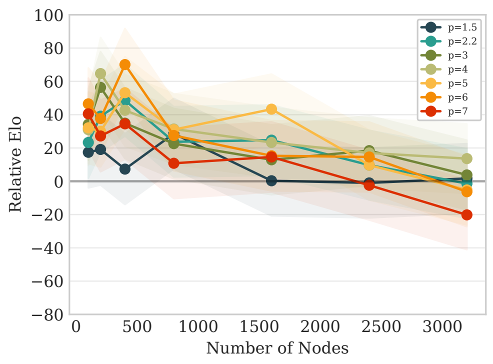
Our findings highlight that Power-mean Backpropagation performs better when p falls within $\{4, 5, 6\}$ in both Chess and Crazyhouse. In the Chess domain, Elo ratings begin to decline beyond 800 nodes. Conversely, within Crazyhouse, Power-mean Backpropagation shows commendable results up to 2400 nodes, with an incremental decline observed from 3200 nodes. The reduction in the Elo Rating of Power-mean PUCT as node count increases can be attributed partly to the complexities involved in its calculations. Moreover, the shifting computations can increase the rounding errors in scenarios with a large number of child nodes, potentially leading to inaccurate Q-value calculations. The proof of exact Q-value calculation following shifting remains unestablished.

A hybrid approach was pursued to enhance performance by combining Power-mean Backpropagation with Mean- or Max Backpropagation. This hybridization involves transitioning from Power Mean PUCT to Mean PUCT or the maximum value of child nodes after a specific number of visits. The goal was to explore whether this novel combination is more stable and effective under high node and simulation counts.

Within this context, $p \in \{4, 5, 6\}$ is selected and incrementally increased the number of visits to ascertain the optimal threshold. Notably, a performance weakening in Chess was recognized at around 2400 nodes, leading to the definition of the threshold for Chess through the 2400 nodes test. In contrast, in Crazyhouse, where Power-mean Backpropagation retained its superiority at 2400 nodes, testing was initiated from 3200 nodes using the same power mean values. Figure 5.5 illustrates the performances of the hybrid approach combining Power-mean PUCT with mean PUCT. Figure 5.6 shows the hybrid approach combining Power-mean PUCT with Max Backpropagation. Both new approaches have shown slight effectiveness in their combined forms. However, the standard MCTS beats the combination involving Mean Backpropagation for tournaments with high node counts. Simultaneously, the Max Backpropagation combination shows a slight advantage over the standard MCTS for specific switching numbers. Nevertheless, it's crucial to emphasize that the extent of these hybrid approaches' effectiveness in stabilizing and improving the performance of Power-mean PUCT under conditions of high node and simulation counts has not been conclusively determined.

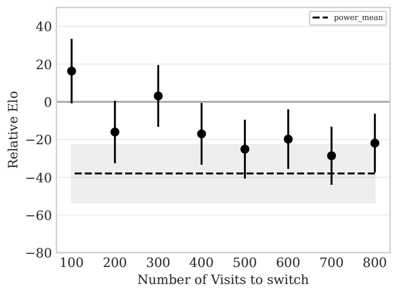


(a) Elo development of Power-mean Backpropagation with different p in Chess.

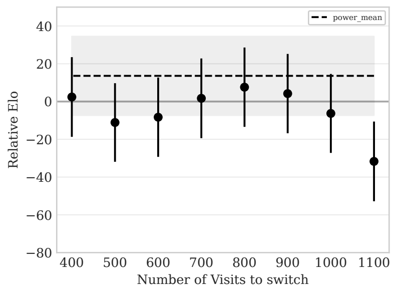


(b) Elo development of Power-mean Backpropagation with different p in Crazyhouse.

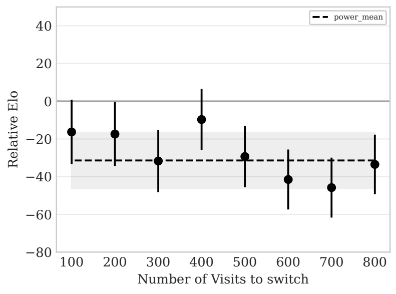
Figure 5.4.: Elo development of Power-mean Backpropagation in Chess and Crazyhouse with $p \in \{1.5, 2.2, 3, 4, 5, 6, 7\}$. Power-mean PUCT is outperformed when the number of nodes is low for $p \in \{4, 5, 6\}$ in both domains. In the Chess domain, Elo ratings decline from 800 nodes, while in Crazyhouse, Power-mean Backpropagation performs well up to 2400 nodes but gradually decreases from 3200.



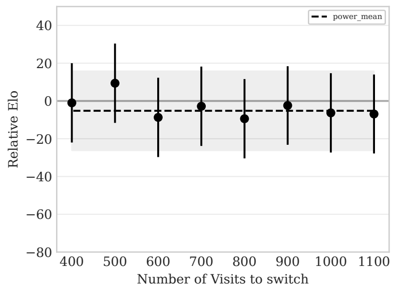
(a) p=4 for tournaments with 2400 nodes in Chess.



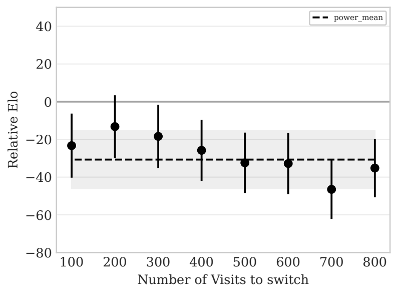
(b) p=4 for tournaments with 3200 nodes in Crazyhouse.



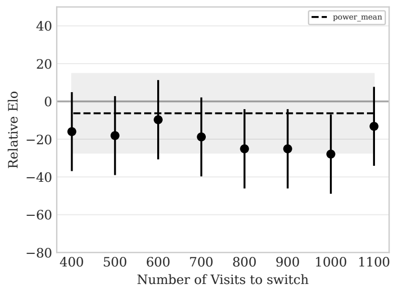
(c) p=5 for tournaments with 2400 nodes in Chess.



(d) p=5 for tournaments with 3200 nodes in Crazyhouse.

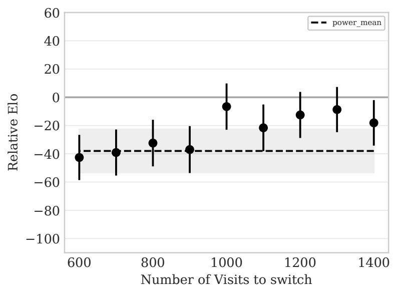


(e) p=6 for tournaments with 2400 nodes in Chess.

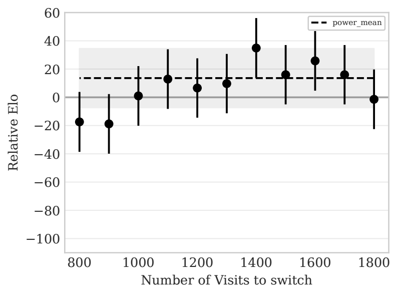


(f) p=6 for tournaments with 3200 nodes in Crazyhouse.

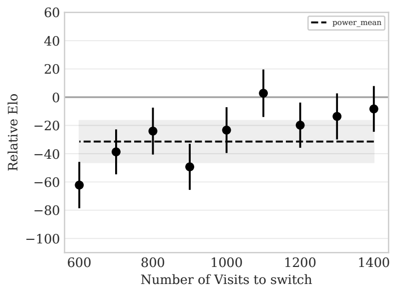
Figure 5.5.: Elo comparison of the hybrid approach combining Power-mean Backpropagation with Mean Backpropagation for different visit numbers to switch in Chess and Crazyhouse. While the new hybrid approach displays slight effectiveness, the standard MCTS still outperforms it.



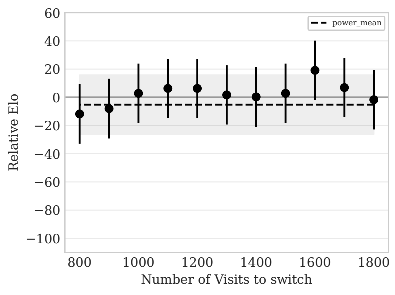
(a) p=4 for tournaments with 2400 nodes in Chess.



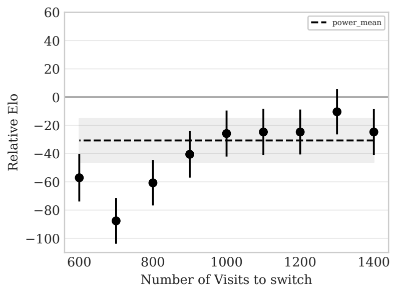
(b) p=4 for tournaments with 3200 nodes in Crazyhouse.



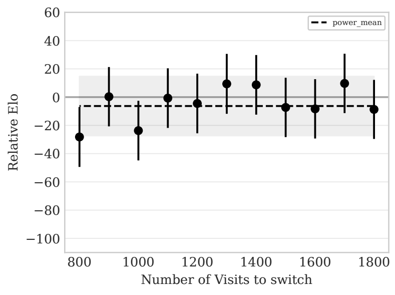
(c) p=5 for tournaments with 2400 nodes in Chess.



(d) p=5 for tournaments with 3200 nodes in Crazyhouse.



(e) p=6 for tournaments with 2400 nodes in Chess.



(f) p=6 for tournaments with 3200 nodes in Crazyhouse.

Figure 5.6.: Elo comparison of the hybrid approach combining Power-mean Backpropagation with Max Backpropagation for each number of visits to switch in Chess and Crazyhouse. The new hybrid approach shows slight effectiveness.

5.4. MCTS with Informed Cutoffs (MCTS-IC)

The initial focus is evaluating the effectiveness of MCTS-IC by incorporating neural network evaluation for Minimax Search. Due to the extensive resources and time required by neural network evaluation, a particular configuration was exclusively considered, encompassing 100 Nodes, Batch Size=1, and Threads=1. Furthermore, Minimax-Search-Depth=1 ($d = 1$) is assessed through a 1000-round tournament, while for Minimax-Search-Depth=2 ($d = 2$), the evaluation is constrained to a 300-round tournament due to time constraints and the related high cost. Table 5.1 provides the Elo Ratings corresponding to each setup. The findings reveal that MCTS-IC demonstrates efficacy in Chess and Crazyhouse when utilizing neural network evaluation, particularly as the minimax search depth is increased, resulting in a more robust performance. However, this approach is hindered by limitations in terms of time constraints and environmental factors.

Nodes	Rounds	d	Chess	Crazyhouse
100	1000	1	21.9 ± 18.6	38.7 ± 21.4
	300	2	83.0 ± 35.4	123.4 ± 41.5

Table 5.1.: The Elo Ratings of MCTS-IC with neural network evaluation, Batch-size=1, Threads=1 in Chess and Crazyhouse. Despite its high cost, MCTS-IC with neural network evaluation shows outstanding performance.

The neural network evaluation was replaced with Stockfish's evaluation to determine outcome differences. Tests were conducted using $d \in \{0, 1, 2\}$, ranging from 100 to 400 nodes and containing 1000 Tournaments, leveraging Stockfish's quick search speed and efficient pruning mechanisms. The results of MCTS-IC with Stockfish Evaluation in Chess and Crazyhouse are presented in Table 5.2. The findings indicate that Stockfish evaluation yields comparatively weaker results than deep neural network evaluation. Nonetheless, it is noticeable that as the Minimax-Search-Depth (d) increases, MCTS-IC's effectiveness becomes more pronounced.

d	Nodes					
	Chess			Crazyhouse		
	100	200	400	100	200	400
0	-490.7 ± 44.8	-600.9 ± 63.7	-816.7 ± 119.3	-629.4 ± 70.2	-816.7 ± 135.5	-1199.8 ± nan
1	-598.0 ± 56.7	-739.1 ± 99.0	-958.5 ± 269.4	-515.2 ± 51.0	-659.2 ± 75.8	-636.4 ± 71.7
2	-365.0 ± 32.7	-469.0 ± 43.3	-659.2 ± 74.7	-495.7 ± 48.3	-539.0 ± 54.5	-616.2 ± 68.2

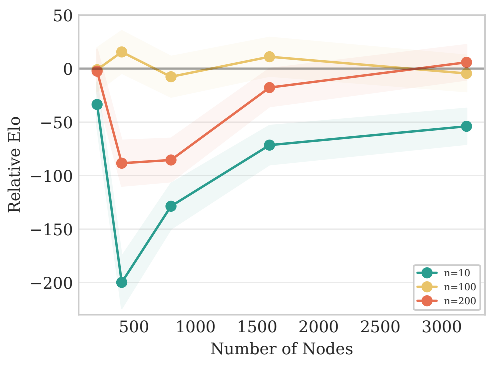
Table 5.2.: The Elo Ratings of MCTS-IC with Stockfish's evaluation in Chess and Crazyhouse. The standard MCTS outperforms MCTS-IC with Stockfish's evaluation.

5.5. MCTS with Informed Priors (MCTS-IP)

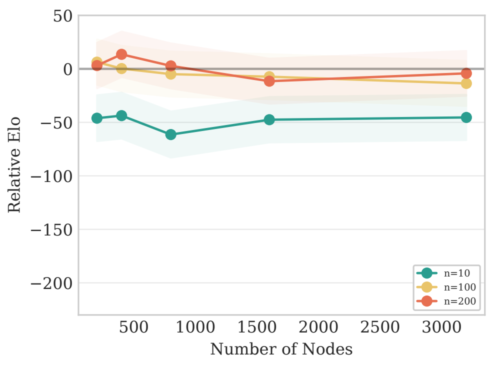
Because of the computational costs linked to deep neural network evaluation and the inclusion of hyperparameters in the MCTS-IP-M approach, the decision was made to skip the testing of MCTS-IP-M with deep neural network evaluation. Instead, the evaluation focused only on MCTS-IP-M with Stockfish's Evaluation. Initially, the MCTS-IP-M assessment used varying numbers of nodes (n) to trigger Minimax-Search during the Selection Phase. Figure 5.7 illustrates the Elo development for $n \in \{10, 100, 200\}$, maintaining a prior weight of 10 $(\gamma = 10)$. The outcomes underscore the more consistent performance of MCTS-IP-M when $n = 100$ in Chess and $n = 200$ in Crazyhouse, compared to other n values.

Subsequently, $n = 100$ was chosen to evaluate MCTS-IP-M, exploring various prior weight (γ) settings within the set 10, 20, 30. Figure 5.8 shows the performances of MCTS-IP-M with different prior weights. It becomes evident that $\gamma = 10$ yields favorable performance in Chess, while in Crazyhouse, $\gamma = 20$ stands out.

As observed from the results of MCTS-IC, Stockfish's evaluation is not as robust as deep neural network evaluation. Consequently, determining the effectiveness of MCTS-IP-M remains challenging due to the limitations of Stockfish's evaluation.

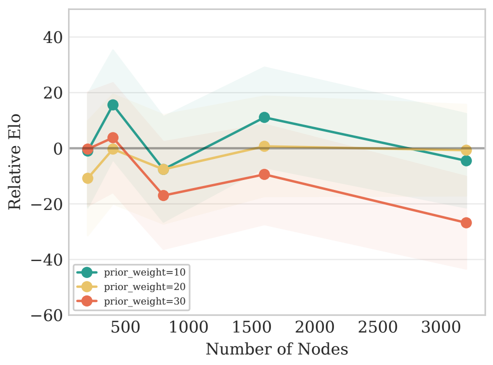


(a) MCTS-IP-M in Chess.

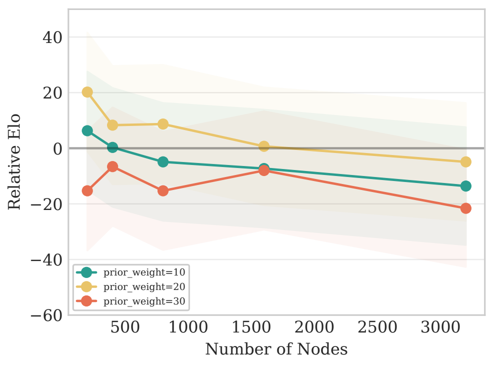


(b) MCTS-IP-M in Crazyhouse.

Figure 5.7.: Elo development of MCTS-IP-M using Stockfish' evaluation with $\gamma = 10$ and $n \in \{10, 100, 200\}$ in Chess and Crazyhouse.



(a) MCTS-IP-M in Chess.



(b) MCTS-IP-M in Crazyhouse.

Figure 5.8.: Elo development of MCTS-IP-M using Stockfish' evaluation with $\gamma \in \{10, 20, 30\}$ and $n = 100$ in Chess and Crazyhouse.

6. Conclusion and Future Work

The five proposed approaches were reconstructed and refined within the Alphazero framework, and an evaluation based on the number of nodes was carried out in the domains of Chess and Crazyhouse. The findings of this thesis provide valuable insights into the strengths and limitations of the adapted techniques and their impact on AlphaZero's overall gameplay. Our analysis still has valuable findings despite the absence of a standout approach that consistently outperforms the standard MCTS across varying node counts and time control settings.

Firstly, the various Backpropagation techniques illuminate vulnerabilities in the standard MCTS at lower node counts while highlighting its strength as node count increases. Firstly, the various Backpropagation techniques illuminate vulnerabilities in the standard MCTS at lower node counts while highlighting its strength as node count increases. The Implicit Minimax Backups technique stands out due to its ease of use and excellent performance in a domain that requires only a few nodes. Similarly, the Power-mean PUCT works well in situations illustrated by low node counts, outperforming Implicit Minimax Backups due to a higher node count.

Finally, combining minimax search pruning within MCTS through the MCTS-IC approach with neural network evaluation emphasizes a substantial impact on MCTS, disregarding the associated cost and runtime considerations. While the MCTS-IC approach with Stockfish's evaluation or the MCTS-IP approach with Stockfish's evaluations exhibits terrible outcomes, they highlight its fast evaluation capabilities and ability to integrate with MCTS.

For future work, there is potential for applying Implicit Minimax Backups and Power-Mean PUCT to domains with limited node requirements, like Pommerman. Moreover, the Power-mean PUCT shows promising outcomes when carefully analyzed to address value calculations in the negative range. Lastly, leveraging Stockfish's rapid evaluation can be employed during the selection phase. After several visits, a limited-depth minimax search will be incorporated to ascertain the optimal move. This approach enables quick identification of blind spots in Monte Carlo Tree Search (MCTS), substantially reducing the cases of missing or undervaluing critical moves.

A. Appendices

Domain	Nodes				
	100	200	400	800	1600
Chess	40.1 ± 20.5	-10.4 ± 20.2	-129.7 ± 20.4	-217.9 ± 21.5	-249.1 ± 21.1
Crazyhouse	-10.8 ± 21.4	-74.1 ± 21.8	-168.4 ± 23.7	-212.9 ± 25.1	-272.4 ± 27.8

Table A.1.: The Comparison of the Elo development concerning the count of MCTS nodes between Chess and Crazyhouse while retaining maximum values instead of transitioning from mean values to maximum values.

n	Nodes				
	1500	1800	2400	3000	3600
700	-8.0 ± 17.6	-34.9 ± 17.5	-34.9 ± 16.3	-63.6 ± 16.0	-88.4 ± 16.4
800	0.7 ± 17.6	-9.7 ± 17.7	-37.7 ± 16.5	-60.4 ± 16.5	-73.0 ± 15.9
900	-5.2 ± 18.0	10.8 ± 17.7	-12.5 ± 16.9	-45.8 ± 16.1	-65.4 ± 15.6
1000	-6.9 ± 18.0	-4.9 ± 17.2	-31.0 ± 16.8	-27.2 ± 16.0	-50.0 ± 15.9
1100	6.9 ± 18.0	0.3 ± 18.1	-7.3 ± 16.7	-24.4 ± 16.4	-42.6 ± 15.5

Table A.2.: Elo development of Max Backpropagation for different numbers of visits(n) to switch from mean value to maximum value in Chess.

n	Nodes				
	1500	1800	2400	3000	3600
700	-2.1 ± 21.2	-6.6 ± 21.2	-34.9 ± 21.2	-24.7 ± 21.2	-73.0 ± 21.6
800	-8.7 ± 21.2	-15.3 ± 21.2	-5.9 ± 21.1	-33.5 ± 21.2	-66.8 ± 21.5
900	-16.3 ± 21.3	-12.2 ± 21.3	-11.5 ± 21.1	-18.1 ± 21.1	-41.5 ± 21.3
1000	5.6 ± 21.1	-3.8 ± 21.2	-8.0 ± 21.2	-31.4 ± 21.0	-38.4 ± 21.2
1100	15.3 ± 21.2	-8.7 ± 21.3	6.6 ± 21.0	-6.9 ± 21.1	-20.2 ± 21.0

Table A.3.: Elo development of Max Backpropagation for different numbers of visits(n) to switch from mean value to maximum value in Crazyhouse.

α	Nodes				
	100	200	400	800	1600
0.1	-16.7 ± 21.0	6.3 ± 20.7	1.0 ± 19.4	1.0 ± 19.0	-22.6 ± 17.6
0.2	-12.9 ± 20.7	-17.7 ± 19.6	-11.8 ± 19.5	11.1 ± 18.6	-15.3 ± 17.6
0.3	31.7 ± 20.8	47.5 ± 20.9	-15.3 ± 19.6	-11.8 ± 18.4	-34.5 ± 17.8
0.5	59.6 ± 20.7	56.1 ± 20.9	9.7 ± 19.4	4.5 ± 18.5	-35.9 ± 17.3
0.6	54.3 ± 20.8	63.6 ± 21.0	9.4 ± 19.4	-21.9 ± 18.4	-70.1 ± 17.5
0.7	71.9 ± 20.9	49.0 ± 20.1	-1.4 ± 19.3	-41.5 ± 18.2	-89.1 ± 17.0
0.8	71.9 ± 20.7	55.7 ± 20.8	-12.5 ± 18.7	-81.0 ± 18.3	-117.2 ± 17.7
0.9	46.1 ± 20.6	62.9 ± 20.7	-60.0 ± 19.8	-130.5 ± 19.1	-164.1 ± 18.8

Table A.4.: Elo development of Implicit-Minimax-Backups for different minimax weight(α) in Chess.

α	Nodes				
	100	200	400	800	1600
0.1	-7.3 $\pm$ 21.4	7.6 $\pm$ 21.4	0.0 $\pm$ 21.3	-22.6 $\pm$ 21.3	-11.5 $\pm$ 21.3
0.2	-19.5 $\pm$ 21.5	16.7 $\pm$ 21.4	-2.8 $\pm$ 21.2	-3.8 $\pm$ 21.3	-11.1 $\pm$ 21.2
0.3	6.3 $\pm$ 21.4	10.8 $\pm$ 21.4	8.7 $\pm$ 21.2	-8.7 $\pm$ 21.3	-8.3 $\pm$ 21.1
0.4	26.5 $\pm$ 21.5	15.6 $\pm$ 21.4	13.6 $\pm$ 21.3	4.9 $\pm$ 21.3	-8.3 $\pm$ 21.1
0.5	39.1 $\pm$ 21.5	37.0 $\pm$ 21.4	13.9 $\pm$ 21.3	13.2 $\pm$ 21.2	3.8 $\pm$ 21.1
0.6	44.7 $\pm$ 21.5	37.3 $\pm$ 21.4	35.2 $\pm$ 21.5	5.6 $\pm$ 21.1	-6.3 $\pm$ 21.0
0.7	35.6 $\pm$ 21.5	48.3 $\pm$ 21.5	28.2 $\pm$ 21.3	5.9 $\pm$ 21.1	-30.3 $\pm$ 21.3
0.8	34.5 $\pm$ 21.5	17.4 $\pm$ 21.4	19.8 $\pm$ 21.1	-30.7 $\pm$ 21.3	-58.6 $\pm$ 21.3
0.9	23.0 $\pm$ 21.4	11.8 $\pm$ 21.4	-52.5 $\pm$ 21.4	-92.5 $\pm$ 21.7	-125.4 $\pm$ 22.5

Table A.5.: Elo development of Implicit-Minimax-Backups for different minimax weight(α) in Crazyhouse.

p	Nodes						
	100	200	400	800	1600	2400	3200
1.5	-18.8 $\pm$ 20.9	-13.9 $\pm$ 20.1	-18.8 $\pm$ 19.2	-32.4 $\pm$ 17.6	-38.7 $\pm$ 16.3	-31.4 $\pm$ 15.3	-51.1 $\pm$ 14.7
2.2	26.8 $\pm$ 20.8	19.5 $\pm$ 20.0	2.4 $\pm$ 18.9	-13.6 $\pm$ 17.1	-34.5 $\pm$ 16.5	-32.4 $\pm$ 15.2	-41.2 $\pm$ 15.3
3	22.3 $\pm$ 20.7	31.7 $\pm$ 20.1	35.6 $\pm$ 18.7	9.7 $\pm$ 18.0	-19.8 $\pm$ 16.6	-30.7 $\pm$ 15.5	-43.3 $\pm$ 14.9
4	43.3 $\pm$ 20.6	42.2 $\pm$ 19.9	25.1 $\pm$ 19.1	5.9 $\pm$ 17.6	-16.3 $\pm$ 16.3	-38.0 $\pm$ 15.4	-37.0 $\pm$ 14.9
5	50.4 $\pm$ 20.7	40.5 $\pm$ 20.1	30.7 $\pm$ 19.0	6.3 $\pm$ 17.5	-17.7 $\pm$ 16.1	-31.4 $\pm$ 14.8	-28.9 $\pm$ 15.2
6	63.6 $\pm$ 20.8	38.7 $\pm$ 19.9	36.3 $\pm$ 18.8	8.3 $\pm$ 17.3	-29.3 $\pm$ 15.9	-30.7 $\pm$ 15.4	-39.8 $\pm$ 14.5
7	50.4 $\pm$ 20.8	39.4 $\pm$ 20.1	36.3 $\pm$ 19.0	15.3 $\pm$ 17.3	-34.2 $\pm$ 16.3	-51.1 $\pm$ 15.7	-44.4 $\pm$ 14.8

Table A.6.: Elo development of Power-Mean-Backpropagation for different power-mean(p) in Chess.

p	Nodes						
	100	200	400	800	1600	2400	3200
1.5	17.4 $\pm$ 21.4	19.1 $\pm$ 21.4	7.3 $\pm$ 21.3	28.6 $\pm$ 21.2	0.3 $\pm$ 21.1	-1.0 $\pm$ 21.0	1.7 $\pm$ 20.9
2.2	23.3 $\pm$ 21.5	39.1 $\pm$ 21.5	48.6 $\pm$ 21.5	23.7 $\pm$ 21.1	24.7 $\pm$ 21.2	9.7 $\pm$ 20.9	-1.4 $\pm$ 20.9
3	34.2 $\pm$ 21.5	56.4 $\pm$ 21.5	34.9 $\pm$ 21.3	22.6 $\pm$ 21.1	12.9 $\pm$ 21.0	18.4 $\pm$ 20.9	3.8 $\pm$ 20.9
4	31.0 $\pm$ 21.5	64.7 $\pm$ 21.7	42.6 $\pm$ 21.5	31.4 $\pm$ 21.2	23.3 $\pm$ 21.0	17.0 $\pm$ 21.0	13.6 $\pm$ 20.9
5	32.1 $\pm$ 21.5	30.0 $\pm$ 21.4	53.2 $\pm$ 21.6	31.0 $\pm$ 21.2	43.3 $\pm$ 21.1	9.7 $\pm$ 21.0	-5.2 $\pm$ 21.0
6	46.5 $\pm$ 21.6	37.7 $\pm$ 21.4	70.1 $\pm$ 21.7	27.5 $\pm$ 21.2	15.3 $\pm$ 21.1	14.6 $\pm$ 21.0	-6.3 $\pm$ 21.0
7	40.5 $\pm$ 21.5	27.2 $\pm$ 21.5	34.5 $\pm$ 21.3	10.8 $\pm$ 21.2	14.6 $\pm$ 21.1	-2.4 $\pm$ 20.9	-20.2 $\pm$ 20.9

Table A.7.: Elo development of Power-Mean-Backpropagation for different power-mean(p) in Crazyhouse.

Switching number	$p = 4$	$p = 5$	$p = 6$
100	16.3 $\pm$ 17.1	-16.3 $\pm$ 17.1	-23.3 $\pm$ 17.0
200	-16.0 $\pm$ 16.6	-17.4 $\pm$ 17.0	-13.2 $\pm$ 16.6
300	3.1 $\pm$ 16.4	-31.7 $\pm$ 16.5	-18.4 $\pm$ 16.8
400	-17.0 $\pm$ 16.4	-9.7 $\pm$ 16.2	-25.8 $\pm$ 16.2
500	-25.1 $\pm$ 15.6	-29.3 $\pm$ 16.3	-32.4 $\pm$ 16.0
600	-19.8 $\pm$ 15.8	-41.5 $\pm$ 15.9	-32.8 $\pm$ 16.2
700	-28.6 $\pm$ 15.4	-45.8 $\pm$ 15.9	-46.5 $\pm$ 15.7
800	-21.9 $\pm$ 15.6	-33.5 $\pm$ 15.8	-35.2 $\pm$ 15.5

Table A.8.: Elo comparison of the hybrid approach combining Power-mean backpropagation with mean backpropagation for each number of visits to switch in Chess with 2400 nodes.

Switching number	$p = 4$	$p = 5$	$p = 6$
400	2.4 $\pm$ 21.1	-1.0 $\pm$ 21.0	-16.0 $\pm$ 20.9
500	-11.1 $\pm$ 20.8	9.4 $\pm$ 21.0	-18.1 $\pm$ 20.9
600	-8.3 $\pm$ 21.0	-8.7 $\pm$ 21.0	-9.7 $\pm$ 21.0
700	1.7 $\pm$ 21.1	-2.8 $\pm$ 21.0	-18.8 $\pm$ 20.9
800	7.6 $\pm$ 21.0	-9.4 $\pm$ 21.0	-25.1 $\pm$ 21.0
900	4.2 $\pm$ 21.0	-2.4 $\pm$ 20.8	-25.1 $\pm$ 21.0
1000	-6.3 $\pm$ 20.9	-6.3 $\pm$ 21.0	-27.9 $\pm$ 21.0
1100	-31.7 $\pm$ 21.1	-6.9 $\pm$ 20.9	-13.2 $\pm$ 20.9

Table A.9.: Elo comparison of the hybrid approach combining Power-mean backpropagation with mean backpropagation for each number of visits to switch in Chess with 3200 nodes.

Switching number	$p = 4$	$p = 5$	$p = 6$
600	-42.6 $\pm$ 16.0	-62.2 $\pm$ 16.4	-57.1 $\pm$ 16.8
700	-39.1 $\pm$ 16.3	-38.7 $\pm$ 15.9	-87.6 $\pm$ 16.2
800	-32.4 $\pm$ 16.5	-24.0 $\pm$ 16.6	-60.7 $\pm$ 16.0
900	-37.0 $\pm$ 16.6	-49.3 $\pm$ 16.3	-40.5 $\pm$ 16.5
1000	-6.6 $\pm$ 16.4	-23.3 $\pm$ 16.2	-25.8 $\pm$ 16.3
1100	-21.6 $\pm$ 16.5	2.8 $\pm$ 16.8	-24.7 $\pm$ 16.4
1200	-12.5 $\pm$ 16.3	-19.8 $\pm$ 16.0	-24.7 $\pm$ 15.9
1300	-8.7 $\pm$ 16.0	-13.6 $\pm$ 16.3	-10.4 $\pm$ 16.0
1400	-18.1 $\pm$ 16.1	-8.3 $\pm$ 16.2	-24.7 $\pm$ 16.2

Table A.10.: Elo comparison of the hybrid approach combining Power-mean backpropagation with max backpropagation for each number of visits to switch in Chess with 2400 nodes.

Switching number	$p = 4$	$p = 5$	$p = 6$
800	-17.4 $\pm$ 21.2	-11.8 $\pm$ 21.1	-28.2 $\pm$ 21.2
900	-18.8 $\pm$ 21.1	-8.0 $\pm$ 21.2	0.3 $\pm$ 21.0
1000	1.0 $\pm$ 21.1	2.8 $\pm$ 21.1	-23.7 $\pm$ 21.1
1100	12.9 $\pm$ 21.1	6.3 $\pm$ 21.0	-0.7 $\pm$ 21.1
1200	6.6 $\pm$ 21.0	6.3 $\pm$ 21.0	-4.5 $\pm$ 21.1
1300	9.7 $\pm$ 21.0	1.7 $\pm$ 21.0	9.4 $\pm$ 21.2
1400	34.9 $\pm$ 21.2	0.3 $\pm$ 21.2	8.7 $\pm$ 21.1
1500	16.0 $\pm$ 21.0	2.8 $\pm$ 21.1	-7.3 $\pm$ 21.0
1600	25.8 $\pm$ 21.1	19.1 $\pm$ 21.1	-8.3 $\pm$ 21.0
1700	16.0 $\pm$ 21.0	6.9 $\pm$ 21.0	9.7 $\pm$ 21.0
1800	-1.4 $\pm$ 21.1	-1.7 $\pm$ 21.1	-8.7 $\pm$ 20.9

Table A.11.: Elo comparison of the hybrid approach combining Power-mean backpropagation with max backpropagation for each number of visits to switch in Crazyhouse with 3200 nodes.

n	Nodes				
	200	400	800	1600	3200
10	-33.5 $\pm$ 20.9	-199.8 $\pm$ 24.2	-128.6 $\pm$ 21.2	-71.5 $\pm$ 18.2	-53.9 $\pm$ 16.7
100	-1.0 $\pm$ 20.4	15.6 $\pm$ 19.7	-7.6 $\pm$ 18.9	11.1 $\pm$ 18.0	-4.5 $\pm$ 16.9
200	-2.4 $\pm$ 20.5	-88.4 $\pm$ 21.3	-85.4 $\pm$ 20.2	-17.7 $\pm$ 17.7	5.9 $\pm$ 16.4

Table A.12.: MCTS-IP-M with a different number of visits(n) in Chess.

n	Nodes				
	200	400	800	1600	3200
10	-46.1 $\pm$ 21.6	-43.7 $\pm$ 21.6	-61.4 $\pm$ 21.7	-47.5 $\pm$ 21.4	-45.4 $\pm$ 21.3
100	6.3 $\pm$ 21.3	0.3 $\pm$ 21.4	-4.9 $\pm$ 21.2	-7.3 $\pm$ 21.2	-13.6 $\pm$ 21.2
200	3.1 $\pm$ 21.4	13.6 $\pm$ 21.4	2.8 $\pm$ 21.2	-11.5 $\pm$ 21.2	-4.2 $\pm$ 21.1

Table A.13.: MCTS-IP-M with a different number of visits(n) in Crazyhouse.

γ	Nodes				
	200	400	800	1600	3200
10	-1.0 $\pm$ 20.4	15.6 $\pm$ 19.7	-7.6 $\pm$ 18.9	11.1 $\pm$ 18.0	-4.5 $\pm$ 16.9
20	-10.8 $\pm$ 20.6	-0.3 $\pm$ 19.7	-7.6 $\pm$ 19.6	0.7 $\pm$ 18.0	-0.7 $\pm$ 16.4
30	-0.3 $\pm$ 20.5	3.8 $\pm$ 19.7	-17.0 $\pm$ 19.3	-9.4 $\pm$ 18.0	-26.8 $\pm$ 16.6

Table A.14.: MCTS-IP-M with different prior weights(γ) in Chess.

γ	Nodes				
	200	400	800	1600	3200
10	6.3 $\pm$ 21.3	0.3 $\pm$ 21.4	-4.9 $\pm$ 21.2	-7.3 $\pm$ 21.2	-13.6 $\pm$ 21.2
20	20.2 $\pm$ 21.4	8.3 $\pm$ 21.3	8.7 $\pm$ 21.3	0.7 $\pm$ 21.2	-4.9 $\pm$ 21.2
30	-15.3 $\pm$ 21.5	-6.6 $\pm$ 21.3	-15.3 $\pm$ 21.3	-8.0 $\pm$ 21.3	-21.6 $\pm$ 21.1

Table A.15.: MCTS-IP-M with different prior weights(γ) in Crazyhouse.

Bibliography

[1] G. M. Baudet, “An analysis of the full alpha-beta pruning algorithm”, in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, ser. STOC ’78, San Diego, California, USA: Association for Computing Machinery, 1978, pp. 296–313, ISBN: 9781450374378. DOI: 10.1145/800133.804359. [Online]. Available: https://doi.org/10.1145/800133.804359.

[2] R. Coulom, “Efficient selectivity and backup operators in monte carlo tree search”, vol. 4630, May 2006, ISBN: 978-3-540-75537-1. DOI: 10.1007/978-3-540-75538-8_7.

[3] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, 2017. arXiv: 1712.01815 [cs.AI].

[4] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, and J. Wang, “Overview on deepmind and its alphago zero ai”, in *Proceedings of the 2018 international conference on big data and education*, 2018, pp. 67–71.

[5] “Alphago”, [Online]. Available: https://www.deepmind.com/research/highlighted-research/alphago.

[6] C. B. Browne, E. Powley, D. Whitehouse, *et al.*, “A survey of monte carlo tree search methods”, *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. DOI: 10.1109/TCIAIG.2012.2186810.

[7] L. Kocsis and C. Szepesvári, “Bandit based monte carlo planning”, in *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, Springer, 2006, pp. 282–293.

[8] R. Coulom, “Efficient selectivity and backup operators in monte carlo tree search”, in *International conference on computers and games*, Springer, 2006, pp. 72–83.

[9] S. Gelly, L. Kocsis, M. Schoenauer, *et al.*, “The grand challenge of computer go: Monte carlo tree search and extensions”, *Communications of the ACM*, vol. 55, no. 3, pp. 106–113, 2012.

[10] A. Junghanns, "Are there practical alternatives to alpha-beta?", *ICGA Journal*, vol. 21, no. 1, pp. 14–32, 1998.

[11] D. Silver, A. Huang, C. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search", *Nature*, vol. 529, pp. 484–489, Jan. 2016. DOI: `10.1038/nature16961`.

[12] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning", *Artificial intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[13] R. Ramanujan, A. Sabharwal, and B. Selman, "On adversarial search spaces and sampling-based planning", in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 20, 2010, pp. 242–245.

[14] H. Baier and M. H. Winands, "Mcts-minimax hybrids", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 2, pp. 167–179, 2014.

[15] T. Dam, P. Klink, C. D'Eramo, J. Peters, and J. Pajarinen, "Generalized mean estimation in monte carlo tree search", *arXiv preprint arXiv:1911.00384*, 2019.

[16] R. Coulom, "Efficient selectivity and backup operators in monte carlo tree search", vol. 4630, May 2006, ISBN: 978-3-540-75537-1. DOI: `10.1007/978-3-540-75538-8_7`.

[17] M. Lanctot, M. H. Winands, T. Pepels, and N. R. Sturtevant, "Monte carlo tree search with heuristic evaluations using implicit minimax backups", in *2014 IEEE Conference on Computational Intelligence and Games*, IEEE, 2014, pp. 1–8.

[18] P. Khandelwal, E. Liebman, S. Niekum, and P. Stone, "On the analysis of complex backup strategies in monte carlo tree search", in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, 2016, pp. 1319–1328.

[19] M. H. Winands, Y. Björnsson, and J.-T. Saito, "Monte carlo tree search solver", in *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, Springer, 2008, pp. 25–36.

[20] J. Czech, P. Korus, and K. Kersting, "Monte carlo graph search for alphazero", *arXiv preprint arXiv:2012.11045*, 2020.

[21] G. Chaslot, M. Winands, H. Herik, J. Uiterwijk, and B. Bouzy, "Progressive strategies for monte carlo tree search", *New Mathematics and Natural Computation*, vol. 04, pp. 343–357, Nov. 2008. DOI: `10.1142/S1793005708001094`.

[22] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem", *Machine learning*, vol. 47, pp. 235–256, 2002.

[23] D. S. Mitrinovic and P. M. Vasic, *Analytic inequalities*. Springer, 1970, vol. 1.

[24] Z. Feldman and C. Domshlak, "Monte carlo planning: Theoretically fast convergence meets practical efficiency", in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'13, Bellevue, WA: AUAI Press, 2013, pp. 212–221.

[25] T. Keller and M. Helmert, "Trial-based heuristic tree search for finite horizon mdps", in *Proceedings of the Twenty-Third International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'13, Rome, Italy: AAAI Press, 2013, pp. 135–143.

[26] Wikipedia contributors, *Kalah — Wikipedia, the free encyclopedia*, [Online; accessed 8-August-2023], 2023. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Kalah&oldid=1165974452`.

[27] R. Lorentz and T. Horey, "Programming breakthrough", in *Computers and Games: 8th International Conference, CG 2013, Yokohama, Japan, August 13-15, 2013, Revised Selected Papers 8*, Springer, 2014, pp. 49–59.

[28] M. H. Winands, Y. Bjornsson, and J.-T. Saito, "Monte carlo tree search in lines of action", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 239–250, 2010.

[29] Wikipedia contributors, *Connect four — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Connect_Four&oldid=1166817902`, [Online; accessed 8-August-2023], 2023.

[30] Wikipedia contributors, *Dōbutsu shōgi — Wikipedia, the free encyclopedia*, [Online; accessed 8-August-2023], 2023. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=D%C5%8Dbutsu_sh%C5%8Dgi&oldid=1160813735`.

[31] Wikipedia contributors, *Reversi — Wikipedia, the free encyclopedia*, [Online; accessed 8-August-2023], 2023. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Reversi&oldid=1166928788`.

[32] H. Baier and M. H. Winands, "Mcts-minimax hybrids with state evaluations", *Journal of Artificial Intelligence Research*, vol. 62, pp. 193–231, 2018.

[33] P. Bullen, "Handbook of means and their inequalities", Jan. 2014. DOI: `10.1007/978-94-017-0399-4`.

[34] M. E. Glickman and A. C. Jones, "Rating the chess rating system", *CHANCE-BERLIN THEN NEW YORK-*, vol. 12, pp. 21–28, 1999.