Utilizing Variance and Uncertainty in Monte-Carlo Tree Search

Bachelor thesis by Martin Andreas Růžička
Date of submission: April 4, 2023

1. Review: Prof. Dr. Kristian Kersting
2. Review: M.Sc. Johannes Czech
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Artificial Intelligence and Machine Learning Lab

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

Hiermit versichere ich, *Martin Andreas Růžička*, die vorliegende Master-Thesis / Bachelor-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt 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 Falle 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.

English translation for information purposes only:

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Martin Andreas Růžička, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

Datum / Date: 04.04.2023

Unterschrift/Signature:

1 Abstract

In this thesis we evaluate two approaches to improving the open-source engine CrazyAra in chess and Crazyhouse. Both of these approaches originate from KataGo, where they have been successfully applied. The first approach uses the variance during Monte-Carlo tree search by giving more playouts to nodes with higher variance. The second approach trains the neural network to predict the difference between its current evaluation and a weighted sum of search-based evaluations in the current and subsequent positions.

We discuss different ways of doing this and test these approaches against an unmodified version of CrazyAra. The evaluation will show that the approach with the variance does not bring an improvement, but rather a deterioration. Unfortunately, we were not able to test the second approach, but the analysis of the data looks promising for future work.

2 Zusammenfassung

In dieser Thesis evaluieren wir zwei Ansätze, die open-source engine CrazyAra in Schach und Crazyhouse zu verbessern. Beide dieser Ansätze stammen von KataGo, wo sie erfolgreich angewandt wurden. Der erste Ansatz nutzt die Varianz während Monte-Carlo Baumsuche, indem er Knoten mit höherer Varianz mehr Playouts gibt. Der zweite Ansatz trainiert das neuronal Netz darauf, die Differenz zwischen seiner aktuellen Evaluation und einer gewichteten Summe von auf Suche basierenden Evaluationen in der aktuellen und den folgenden Positionen, vorherzusagen.
Wir diskutieren verschiedene Möglichkeiten dies umzusetzen und testen diese Ansätze im Vergleich mit einer unmodifizierten Version von CrazyAra. In der Evaluation wird sich zeigen, dass der Ansatz mit der Varianz keine Verbesserung mit sich bringt, sondern eher eine Verschlechterung. Den zweiten Ansatz konnten wir leider nicht mehr testen, die Analyse der Daten sieht aber erstmal vielversprechend aus.

Contents

3 Introduction

3.1 Motivation

When considering which move to evaluate next, the AlphaZero(A0) framework[18] does not use the variance of the moves. But humans certainly do, and it also makes sense intuitively: When basing your decision on an evaluation, you want to be confident that the evaluation is accurate. However, a high variance on an evaluation implies that different branches give drastically different evaluations. This leaves us unsure which evaluation to trust, in an extreme case we would not know whether it is a winning or a losing position. Therefore it is a good idea to take a closer look at this move in order to improve the accuracy of our evaluation. Wu et al.[25] introduce the variance as a factor in the exploration part of the PUCT-formula, such that a higher variance makes the respective move more likely to be considered.

Our second approach to improving the A0 framework is to consider uncertainty. The network should predict it's own short-term error, i.e. the difference between the current evaluation and the evaluation it would get after performing search or in a few moves. Then we scale the playouts so that playouts where we are confident about the evaluation count more than playouts where we are uncertain.

In this thesis, we will evaluate whether these two approaches can improve the performance of the engine CrazyAra[7].

3.2 Outline

In the following background chapter 4 we will introduce the necessary terms for this thesis and introduce Monte-Carlo Tree Search. After that, in chapter 5 we will give a brief overview of related work on MCTS engines and papers on variance and uncertainty. Chapter 6 will present different ways of utilizing variance that we have experimented with in this thesis. Chapter 7 will describe our approach and methodology for utilizing the uncertainty. In chapter 8 we document our evaluation. This will be a primary test of utilizing variance in Multi-Armed-Bandits and then an extensive evaluation of different approaches to use the variance in MCTS. The chapter finishes with some notes on our preparation to use the uncertainty in MCTS. In the following chapter 9 we discuss those results, identify problems during our evaluation and discuss possible reasons for our results. We conclude the thesis in chapter 10 with a summary and an outlook to future work.

4 Background

In this chapter, we will first briefly introduce the problem domain, crazyhouse, give a comprehensive introduction to the empirical variance, then give an overview of the Monte-Carlo Tree-Search (MCTS) algorithm and finish with some notes on uncertainty.

4.1 Crazyhouse

Crazyhouse is a chess variant that is played with the same starting position and pieces as in normal chess. However, when a player captures an opponent's piece, they add that piece to their own pocket. Instead of making a normal, legal, chess move, the player can now also drop a piece from their pocket to a free square on the board. For some more notes on crazyhouse and a description of crazyhouse as a Reinforcement Learning Problem, see[5].

4.2 Variance

In this work we use two mathematical terms, the empirical expected value and the empirical variance. We will provide the definition and the formula by which we calculate them efficiently.

The empirical expected value $\bar{x}$ describes the average value of the samples drawn so far and can be used to approximate the value we would if we drew a new sample. It is defined as

$$\bar{x} := \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \tag{4.1}$$

where n is the sample size and $\{x_1, ..., x_n\}$ are the first n samples that were drawn.
The empirical variance s^2 measures how far the samples deviate from the expected value. It is defined as

$$s^2 := \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2$$

with the same notation as above. The factor $\frac{1}{n-1}$ leads to an unbiased and consistent estimator, which ensures that with growing sample size the empirical variance converges towards the true variance. For these definitions see any standard stochastic textbook, e.g. Eckle-Kohler and Kohler [8].
If we were to compute the empirical expected value and variance using these formulas, we would have to repeat many computations each time we received a new sample (e.g. the first n additions of the sum remain the same for the expected value). Therefore, we use running formulas that compute the new value using the old value that we have already computed. This saves computation time and improves from $O(n)$ to $O(1)$. For the empirical expected value we use the running formula

$$\bar{x}_{n+1} = \frac{\bar{x}_n \cdot n + x_{n+1}}{n+1}. \tag{4.2}$$

One can easily verify that $\bar{x}_n \cdot n = \sum_{i=1}^{n} x_i$ with $\bar{x}$ as defined in 4.1 and therefore our running formula is correct.
For the empirical variance, the running formula is not as straightforward, we do not directly use the previous value, but by rewriting s^2 to

$$s^2 = \frac{n \cdot \frac{\sum_{i=1}^{n} x_i^2}{n} - n \cdot \bar{x}^2}{n-1} \tag{4.3}$$

we can use a running formula to track the components, as introduced in [11].
Tracking the power sum average psa $:= \frac{\sum_{i=1}^{n} x_i^2}{n}$ instead of $\sum_{i=1}^{n} x_i^2$ has the numerical advantage that it doesn't suffer from unbounded growth like the power sum does. The power sum average can be easily tracked with a running formula by adding $\frac{x^2 - \text{psa}}{n+1}$ for each new sample x. For the other summand, we need the empirical expected value, which can be tracked as in 4.2. We will later need the empirical expected value as a component in the MCTS search (introduced in section 4.3), so we will be able to reuse this computation.

We can verify the correctness of formula 4.3 with the following calculation:

$$\begin{aligned}
s^2 &= \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2 \\
&= \frac{1}{n} \left(\sum_{i=1}^{n} \left(x_i^2 - 2 \cdot x_i \cdot \bar{x} + \bar{x}^2 \right) \right) \\
&= \frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - 2\bar{x} \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} \bar{x}^2 \right) \\
&\overset{(4.1)}{=} \frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - 2\bar{x} \cdot \bar{x} \cdot n + n \cdot \bar{x}^2 \right) \\
&= \frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - n \cdot \bar{x}^2 \right)
\end{aligned}$$

4.3 MCTS

Monte Carlo Tree Search(MCTS) is an algorithm that aims to find the best decision among many options in the face of uncertainty. Usually we have an infinite, or finite yet huge, amount of options to choose from, so if we build a tree representing those options and all their descendants, we could not search the entire tree in a reasonable span of time. So have to use heuristics to focus on certain nodes in the tree. MCTS is a popular algorithm for this purpose.

4.3.1 Four Steps Of MCTS

The MCTS-algorithm as shown in Figure 4.1 consists of four steps: selection, expansion, simulation and backpropagation.
First, we use the *tree policy* to select the most urgent node for further evaluation. In the expansion step, we add a node to the tree that represents the new state that will be reached when the selected action is performed in the corresponding state. We then use the *default policy* to simulate the game to the end from the newly created state and save

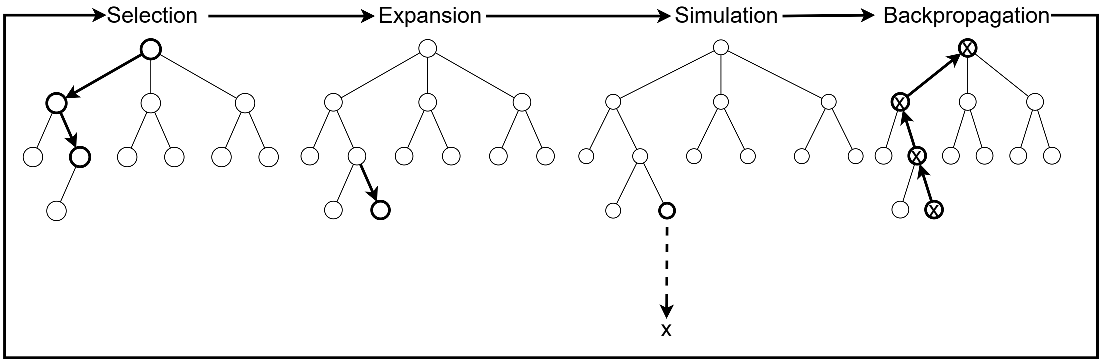

Figure 4.1: Classic MCTS algorithm as shown e.g. in [4]

the result. This result is then backpropagated from the newly added state to its ancestors, so that they can update their respective statistics.

4.3.2 Properties Of MCTS

When choosing a node in the selection phase, we try to balance *exploration* and *exploitation*. Exploration means evaluating nodes that have not been evaluated (much) yet. Exploitation means evaluating nodes that seem promising so far. If we exploit too much, we run the risk of missing a node that didn't show better results in the first few visits, but would give a higher reward if evaluated further. When exploring, we evaluate nodes that seem inferior so far, which we want to do as little as possible.

When evaluating a node, we take the state of the node and keep taking actions according to the default policy, until we reach a terminal state. The simplest way to define a default policy is to make random moves. However, even this policy performs well in some cases[4], as a more promising node should give better results on average. In practice there are many better policies which we will come back to in 4.4.3.

The terminal state we have reached is now evaluated, e.g. assigning 1 for a win, 0 for a draw and -1 for a loss. Next, the corresponding node initialises it's own statistics and backpropagates the result. This evaluation method is a major advantage of MCTS in problems where it is difficult to define good policies for evaluating intermediate states, as in games such as chess or Go[4][21].

Another big advantage of MCTS is it's *anytime* property. Because each result is immediately backpropagated, the algorithm can always return the current best choice. In general the

result will improve with more playouts, and implementations based on UCB1 (Section 4.4.1) have been shown to converge to game-theoretic optimality[13].
In a given state s, the tree policy searches for the argmax of the term Q(s,a)+U(s,a), i.e. the action that has the highest value of this term. The Q-term calculates the expected value of the reward of choosing this action:

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{N(s)} \mathbb{1}_i(s,a) \cdot z_i$$

with $\mathbb{1}_i(s,a) = 1$ if we select action a in time step i with state s and 0 otherwise. $N(s,a)$ counts how many times action a was chosen in state s. The Q-term thus represents exploitation, since an action that seems more promising so far has a higher expected value. The U-term should therefore represent exploration, some possible choices will be introduced in section 4.4.

4.3.3 Multi-Armed Bandit

Multi-armed bandits are a popular use-case for MCTS and a good testing ground for new variations of MCTS. They describe a class of problems that require a balance between exploration and exploitation. The term bandit comes from games of chance, where a slot-machine (or bandit) gives a reward based on an unknown distribution. If you have several arms available, which are assumed to be independent of each other, we want to choose the one that maximizes the reward. To do this, we need the best possible estimate of each bandit's expected reward. We can use various algorithms to determine which arm to play next, some of them are described in the next section 4.4. Often we track the *regret* of an action a instead of the reward. Regret describes the expected difference between playing optimally and playing according to our policy[2]. The regret after n plays is defined as

$$R(a) = \mu^* n - \sum_{j=1}^{K} \mu_j \mathbb{E}[T_j(n)]$$

where μ^* is the maximum expected value of the K arms, i.e. the reward we would receive if we always played the optimal arm. $\mathbb{E}[T_j(n)]$ is the expected number of times arm j will be chosen in the first n rounds. It can be shown that, for many distributions, the optimal achievable regret suffers from logarithmic growth ($O(\ln n)$)[14].

4.4 Exploration

In the selection phase of MCTS, we search for the argmax of the term $Q(s,a) + U(s,a)$ where the Q-term is the empirical expected value. The U-term should manage the amount of exploration, in the following we will look at different approaches to defining U.

4.4.1 Upper Confidence Bounds

In Upper Confidence Bounds (UCB1)[2], we want to use the Upper Confidence Bounds that a particular arm of a multi-armed bandit is optimal. We set

$$U(s,a) = \sqrt{\frac{2\ln n}{n_j}}$$

where n is the total number of plays so far and n_j is the number of times that arm j has been selected. This term comes from the size of the one-sided confidence interval of the empirical expected value, since most likely the true expected reward lies in that interval. More precisely, the probability that the distance between true reward and the empirical expected value is greater than U(s,a) at time t is smaller than $2t^{-4}$ under the assumption of independent, identically distributed samples. UCB1 has logarithmic regret for any reward distribution with support [0,1] and can therefore be considered optimal. It even achieves this regret uniformly over n and not just asymptotically [2].

4.4.2 Upper Confidence Bound For Trees

Upper Confidence Bound for Trees (UCT)[12] is an algorithm that applies UCB1 to general tree search. We want to choose the best action at the root of the tree, so we use UCB1 to select the most promising child node at each node in the tree. In addition we introduce a factor C_p:

$$U(s,a) = C_p \cdot \sqrt{\frac{2\ln n}{n_j}}$$

This factor C_p can be chosen to be 1 as in UCB1, but Kocsis et al.[12] generalised the result and showed that any C_p can be chosen if it satisfies $C_p \cdot \sqrt{\frac{2\ln n}{n}} \geq 2|\delta_{in}|$ where δ_{in} is the difference between the current empirical expected value and the true expected value.

4.4.3 Predictor Plus Upper Confidence Bounds

PUCB(Predictor plus UCB)[16] uses contextual side information to improve UCB1. This contextual side information is a prior probability that the arm is good, which can be given e.g. by domain-specific knowledge or a neural network. This prior is assumed to be imperfect, so search is still needed.
PUCB works well if the prior is good. Unlike UCB1, here we do not necessarily start by playing every arm once, because we don't rely on experimental information to estimate how good an arm is.
In PUCB our U-term is:

$$U(s,a) = c(n, s_i) - m(n,i)$$

where $c(n,s) = \sqrt{\frac{3 \cdot \ln n}{2 n_i}}$ and $m(n,i) = \frac{2}{M_i} \cdot \sqrt{\frac{\ln n}{n}}$ generally and $m(1,i) = \frac{2}{M_i}$ and $c(n,0) = 0$ as initialization. The factor $\frac{3}{2}$ here differs from the 2 in UCB1. M_i is the prior of action i, with the weights summing to 1, $\sum_i M_i = 1$.

4.4.4 Polynomial Upper Confidence Bound for Trees

Polynomial Upper Confidence Bound for Trees(PUCT)[16][20] is a refinement of PUCB which also uses contextual side information. It's U-term is:

$$U(s,a) = c_{puct} \cdot P(s,a) \frac{\sum_b n_b}{1 + n_a} \tag{4.4}$$

where P(s,a) is the prior probability, c_{puct} is a constant that determines the level of exploration and $\sum_b n_b$ and n_a are the total number of visits and the number of visits of action a in state s, respectively.
David Silver et al. revolutionised engine play of board games such as Go, chess and Shogi with AlphaGo[19], AlphaGoZero[20] and AlphaZero[18]. They proved that these games can be learned to superhuman levels without incorporating human knowledge by using a general-purpose reinforcement learning algorithm. They used PUCT with the choice

$$c_puct(s) = \log \frac{\sum_a N(s,a) + c_{puct-base} + 1}{c_{puct-base}} + c_{puct_init}.$$

CrazyAra also uses PUCT with $c_{puct-base} = 19652$ and $c_{puct-init} = 2.5$[6].

4.4.5 MCTS And Multi-Threading

One of the most popular approaches to improving the performance of algorithms is multi-threading. MCTS is no exception, in principle we can run the search in parallel as long as the backpropagation is atomic. However, if we simply use multiple threads to run the MCTS algorithm, we run into a problem: all these threads would choose the same node, since the formula is deterministic. So we introduce *virtual loss*.

When a thread has chosen a node to expand, we pretend to have lost a certain amount (default 1) of games from that position so that it appears less attractive to the other threads. By temporarily adding a number of games with a result of -1, the other nodes are likely to choose another node, since the Q-term has drastically decreased. When the batch has finished and we backpropagate the results, we revert the virtual loss so that at the start of the next batch we have the correct tree again.

4.5 Uncertainty

Applying uncertainty to improve the accuracy of predictions has a long tradition that goes beyond machine learning. The ”wisdom of crowds” describes the phenomenon that aggregating the predictions of a large number of people, who don’t have to be an expert, can lead to surprisingly accurate predictions[9]. To take this a step further, Ugander et al.[23] added uncertainty to improve the accuracy. They asked people to make multiple guesses and score them according to the accuracy of their best guess. This motivates people to make guesses that are close to each other when they are quite certain and spreading out the guesses when they are uncertain. They showed that this approach significantly improved the performance of the crowd. In this work, however, we use a different interpretation of uncertainty.

We use the term uncertainty to describe the confidence that our current value estimation of a position is correct. To achieve this, we train our network to compute an additional output that estimates the difference between our current value estimation and a weighted sum of values over the next moves in the training data.

5 Related Work

In this chapter we will discuss some papers on the use of variance and uncertainty in MCTS, ideas introduced by KataGo and give a brief introduction to CrazyAra.

5.1 Papers

Bauer, Patten and Vincze have successfully combined the empirical variance with the Q-term of the MCTS-formula to improve state-of-the-art object pose estimation in the field of robotic vision systems[3]. They followed Auer et al.[2] and their adaption of UCB1 into the UCB1-tuned-formula and made the equivalent changes to the formula for Upper Confidence bound for rooted Directed acyclic graphs (UCD)

$$\pi_{UCD}(c, d_1, d_2, d_3) = \hat{\mu}_{d_1} + c \cdot \sqrt{\frac{\ln(p_{d_2})}{n_{d_3}}}$$

to arrive at what they call UCD-tuned

$$\pi_{UCD-tuned}(c, d_1, d_2, d_3) = \hat{\mu}_{d_1} + c \cdot \sqrt{\frac{\ln(p_{d_2})}{n_{d_3}} \cdot \min\left(\frac{1}{4}, \hat{\sigma}^2_{d_1} + \sqrt{\frac{2\ln(p_{d_2})}{n_{d_3}}}\right)}. \tag{5.1}$$

Here the index d_1 of the variance $\hat{\sigma}^2$ indicates that we compute the variance up to depth d_1. They went on to show that this (although combined with other improvements) resulted in a performance improvement over state-of-the-art methods. In section 8.2.1 we will replicate their results.

Audibert et al.[1] show that variance-based approaches work well when “the variance of some sub-optimal arm is much lower than b^2” where b is the upper bound of the reward, i.e. 1 in our case. It is not clear whether our domain fulfils this criterion or not, since

much lower is a vague statement.
Guo et al.[10] showed that the softmax prediction of an NN may not match its true confidence, which is the reason why considering the uncertainty is an interesting task in the first place, and thus a foundation of this paper.
Lan et al.[15] evaluated 3 possible ways of predicting the uncertainty. One method is based on the policy of the original network, the other two methods train an auxiliary network. They define a tree status as uncertain if $\exists n' \geq n,$ s.t.$R_{N_{max}}(s, N_{max}) - R_{N_{max}}(s, n') \geq \epsilon$ where ϵ is a small constant, N_{max} is the maximum number of solutions until we assume to have arrived at the ground truth, and $R_{N_{max}}$ the approximate reward of our policy after n simulations. In other words, a status is uncertain if our reward differs from the ground truth by more than a tolerable amount at a later time. Their approaches have been shown to perform at the same level as their base engine while being significantly faster.

5.2 KataGo

Inspired by DeepMind's AlphaZero work, David Wu created the strongest open-source Go engine[1]. To achieve this, they introduced "several new techniques to improve the efficiency of self-play learning" [25]. They also added some domain-specific improvements to achieve results comparable to those of DeepMind and the open-source implementation of AlphaZero, ELF[22], using significantly less computing power. They have documented more recent experimental techniques that proved to be beneficial in a publicly available document [2]. Among other interesting techniques, they described ways to utilize variance and uncertainty in MCTS, which was the inspiration for this thesis.

---

[1] In the most recent computer Go world championship KataGo placed first in the group stage and second in the final.`http://entcog.c.ooco.jp/entcog/new_uec/en/result.html`. The winner used a modified version of KataGo `https://drive.google.com/file/d/1EM1Cy1Zq4EfAqGR1yyu2mLDS3Z05dao8/view`. Also the computing power was not standardised, computer Go tournaments are not very professional yet. Links accessed April 4, 2023

[2] `https://github.com/lightvector/KataGo/blob/master/docs/KataGoMethods.md`, accessed April 4, 2023

5.3 CrazyAra

CrazyAra is an open-source MCTS engine that is capable of playing several chess-variants and excels at crazyhouse, beating the human 2017 crazyhouse world champion [7]. It was originally developed by Johannes Czech, Moritz Willig and Alena Beyer, and was later continued by Johannes Czech. It consists of a Convolutional Neural Network (CNN) which was been trained in supervised manner on a set of human games played on the open-source chess server lichess. Later the network was trained in a reinforcement learning setting[5].

6 Utilizing Variance

In this chapter we will describe our approach to utilize the variance in MCTS.

6.1 Running Formula

There are several running formulas for computing the variance[17]. One can use a naïve approach (as in 4.3), use the fact that the variance is invariant under addition to shift the data, or use more sophisticated algorithms such as Welford's online algorithm[24]. In our evaluation we will mainly use Welford's formula, which we will introduce now. We track the mean value as in 4.2. One way to track the variance would be to track it directly via

$$\sigma_n^2 = \sigma_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1}) \cdot (x_n - \bar{x}_n) - \sigma_{n-1}^2}{n}. \tag{6.1}$$

However, there are two sources of numerical instability, when subtracting the previous mean from our new sample, since these two numbers are likely to be very close to each other, potentially leading to catastrophic cancellation. Also we repeatedly subtract a small number from a large number that scales with n, again leading to loss of precision.
To avoid the latter problem, Welford introduced an algorithm that also tracks the sum of squares of differences from the current mean, M_n, where

$$\begin{aligned} M_n &= M_{n-1} + (x_n - \bar{x}_{n-1}) \cdot (x_n - \bar{x}_n) \\ \sigma_n^2 &= \frac{M_n}{n} \end{aligned}$$

They proved the correctness of this formula in [24].

6.2 Parameter Tuning

To utilize the variance in MCTS, Wu et al.[25] multiplied the standard deviation with the U-term. However, this is not the only option, one could use addition instead of multiplication, one could use the variance instead of the standard deviation, and the choice of the initial value in a concrete implementation is also an interesting question.

6.2.1 Addition Vs. Multiplication

PUCT is already a well-established and widely used algorithm with a careful balance between the U-term and the Q-term, neither of which should dominate the other too much. The variance is a value between 0 and 1, usually closer to 0. So introducing the variance as a factor should generally shift the balance towards the Q-term. One can imagine a scenario where one node has a much higher Q-value than all others, but the U-value of these other nodes is higher. By introducing this new factor we reduce the U-value and therfore the node with high Q-value would be selected more often, resulting in an increase in exploitation.

An alternative would be to leave the U-term as it is and add a new term, the variance. This would maintain the balance between U-term and Q-term, while shifting the attention to nodes with a higher variance. Parameter tuning is required so that the variance-term makes a difference, but does not dominate the other terms. In this scenario we would be searching for the argmax of $Q(s,a) + U(s,a) + \text{var}(s,a)$ with $U(s,a)$ as in 4.4.

6.2.2 Initial Value

Whichever operation we use, the initial value of the variance makes a difference. If we choose to initialize the variance with 0, the variance's term will not matter until the node has been visited twice. If we use multiplication, this even cancels out the entire U-term, leaving the entire early search to the Q-term alone.

With this in mind, we have considered initializing the variance with a value that it is likely to take when first computed (after two visits). Manual analysis of the variance in different board position suggested a value of around 0.05 as a common initial value. The largest observed values were around 0.35, so we also experimented with 0.2 as the mean between these two values and 0.1 as an intermediate step.

6.2.3 Standard Deviation Vs. Variance

KataGo's motivating example[1] does not work when using the variance, only when using the standard deviation. However, our observations implied that the variance also performs well in various configurations, so we included it in our evaluation.

6.2.4 Parameters

In each of the constellations of options described above, additional constant parameters can be introduced. We experimented with multiplying parameters between 0.25 and 2, observing that parameters greater than 1 and smaller than 0.5 did not perform well. Therefore, we used 0.5 and 1 in our final evaluation.

---

[1] `https://github.com/lightvector/KataGo/blob/master/docs/KataGoMethods.md#dynamic-variance-scaled-cpuct`, accessed April 4, 2023

7 Utilizing Uncertainty

We want to use supervised learning to train a new neural network from scratch and then compare the performance of this new network when using uncertainty and when turning it off via an UCI[1]-option.

7.1 Theory

Originally CrazyAra has two output heads, a value output and a policy output. We add a new head, the uncertainty output, which should predict the squared difference between the neural net's value prediction (at time t) and $(1-\lambda)\sum_{t'>=t}\text{MCTS-score}(t')\cdot\lambda^{t'-t}$ where we chose $\lambda=\frac{5}{6}$ as in KataGo[2]. Here t' is every move that comes in the game from the training data set after the current position. The weighted sum's factor $\lambda^{t'-t}$ decreases for bigger t'. Since the training data is not perfect, this makes sense the longer the game goes on, the further it will drift away from optimal play.

7.2 Methodology

For the training data, we gathered every ranked crazyhouse match that was played on a server[3] between January 2016 and March 2018, where both players were in the top 10% of players, corresponding to an Elo rating of 2000. Matches played between April 2018 and August 2018 were used for validation and testing.
We then used CrazyAra to annotate the data with two values, the raw net's evaluation of

---

[1] `https://wbec-ridderkerk.nl/html/UCIProtocol.html`, accessed April 4, 2023

[2] `https://github.com/lightvector/KataGo/blob/master/docs/KataGoMethods.md#uncertainty-weighted-mcts-playouts`, accessed April 4, 2023

[3] `https://lichess.org`, accessed April 4, 2023

the position, i.e. without performing any search and the engine evaluation of the position after 400 nodes. The latter data point is used as ground-truth, i.e. it is assumed to be the correct evaluation of the position and thus our training target. This assumption obviously doesn't hold, but it is a good approximation.

8 Evaluation

8.1 Methodology

In this section, we will describe the methodology we used to fairly compare our modified version of CrazyAra with the baseline.

8.1.1 Cutechess

CuteChess[1] is a set of tools for working with chess engines. We were mainly interested in the command line interface (CLI). It can be used to play single matches or entire tournaments with different participants, such as humans or engines. The CLI allows setting options for all engines or only for specific engines. We used it, for example, to limit the amount of nodes and simulations that can be used, to use MCTS search instead of Monte-Carlo Graph Search (MCGS) and whether or not to use variance. Then we used it to terminate games early when the outcome is sufficiently clear, i.e. an engine is forced to resign when both engines give the other side an advantage of at least 600 centipawns for 5 consecutive moves. A draw was declared when both engines' evaluation was below 20 centipawns for 4 consecutive moves after move 30. The CLI also allowed us to use an opening book. The main reason for using an opening book is to increase the variety of the matches and to get a more accurate estimate of the general strength of the engine. For crazyhouse we used a book[2] that was designed specifically for this purpose. In crazyhouse, another purpose was to *increase* the amount of draws by choosing openings that are more balanced than the usual opening position.
For chess we used another opening book[3], by Stefan Pohl, which was designed to *reduce*

---

[1] `https://github.com/cutechess/cutechess`, accessed April 4, 2023

[2] `https://github.com/ianfab/books/blob/master/crazyhouse_mix_cp_130.epd`, accessed April 4, 2023

[3] `https://www.sp-cc.de/uho_2022.htm`, accessed April 4, 2023

the amount of draws, since top-level chess tends to have a drawish nature.
We decided to limit the amount of nodes and simulations, rather than giving a time-limit, because we wanted to determine if our approach was successful in shifting the engine's attention to more promising nodes in situations of high variance. To do this, we want an equal amount of nodes to be evaluated, the loss in performance is a secondary concern for the moment.

8.2 Results

8.2.1 Multi-Armed Bandit

As a first step, we verified claims from the literature [3] that UCD-tuned (see 5.1) performs better than UCB1 on a bandit problem.
Our experimental setup was to create 50 bandits B, which are randomly initialised, to give normal-distributed reward with $B_i\ N(100 + r, v)$, where r is a random number between -1 and 1, with a step-size of 0.02 and v is a random number between 0.05 and 0.25, with a step-size of 0.005. The idea was to mimic the amount and range of possible moves in crazyhouse. We simulated 10.000 nodes, i.e. 10.000 decisions by each algorithm, summed the reward and compared the two results. This was repeated 100 times to minimize chance. The Python code is documented in a Github-repository[4].
In this experiment, UCD-tuned outperformed UCB1 by a mean margin of 333.55 with a standard deviation of 7.88, the maximum difference was 354.17 and the minimum difference was 307.49. So we can verify the result from [3] and try to port the result to crazyhouse.

8.3 Evaluation Of Variance

In this section we present the results of the evaluation of our modified ClassicAra/CrazyAra engine.

---

[4] `https://github.com/MartinRuz/BA_Multi-armed-bandit`, accessed April 4, 2023

8.3.1 First Tests: Classical Chess With 200 Nodes

First, we will show our evaluation of classical chess, where each engine is assigned a maximum of 200 nodes. We ran two round-robin tournaments, one with six participants and one with seven participants. The first 100 positions of the previously mentioned opening books (8.1.1) were used. Each engine played each other engine twice in every position, once with black and once with white. So in total 42 and 30 matches were played respectively from each position, resulting in a total of 4200 and 3000 matches in the tournament.

In both tournaments we included the baseline engine, an unmodified ClassicAra[5]. For comparison, we chose a few different constellations of parameters that we tuned.

The first distinction is between engines with 'welford' in their name and engines with 'old' in their name. 'Welford' denotes engines that compute the variance using Welford's formula 6.1 and 'old' indicates engines that use the basic running formula 4.3.

The next distinction is between 'var' and 'stddev' where we indicate whether the engine uses the variance or the standard deviation.

'Add' or 'mult' together with a number indicates whether the variance (or standard deviation) was added or multiplied to the U-term, with the number being a further multiplicative factor.

Finally, 'init_x' indicates different initialisations of the variance, using var=x when adding a new node.

In Figure 8.1 we see two engines that performed almost as strong as the reference engine and three engines that performed slightly worse, but still within the margin of error. In this test we didn't include the configuration 'old', adding 0.5 times the variance, so we ran an individual test between the 'old' and the 'welford' configurations The 'welford' configuration won 527-507-194 (W-L-D), corresponding to an Elo difference of +5.7 +/- 17.8. In further tests we will compute the variance using Welford's formula and drop the distinction from the names for brevity.

In Figure 8.2 we tested another set of engine configurations, three of which had a different initial value. They all performed similarly; all of them were significantly worse then the base ClassicAra. ClassicAra_add_05stddev was included again because in a preliminary test it seemed like the most promising engine, unfortunately it performed even worse than in the first test. The other two engines didn't show a notable performance either.

---

[5] `https://github.com/MartinRuz/CrazyAra/commit/c40585ae2553ac042944826fde5ff7a9b7adac18`, accessed April 4, 2023

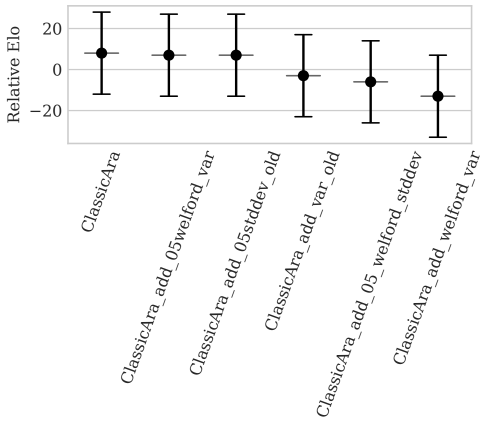

Figure 8.1: Tournament result from Classical Chess with 200 nodes.

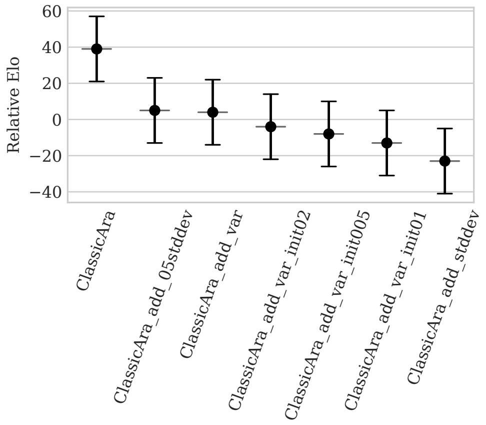

Figure 8.2: Tournament result from Classical Chess with 200 nodes, another batch of engines.

8.3.2 Further Tests In Chess And Crazyhouse

Next, we did evaluations in classical chess, where the engines could use more nodes and tests in crazyhouse with the same number of nodes. In all these tests we used the following 6 engines:

- ClassicAra[6](the baseline),
- ClassicAra_add_05_var[7](one of the two most promising engines from 8.1),
- ClassicAra_add_05stddev[8](the other promising engines from 8.1),
- ClassicAra_add_var_init005[9](one engine with an initial value that is not 0, since

---

[6] `https://github.com/MartinRuz/CrazyAra/commit/c40585ae2553ac042944826fde5ff7a9b7adac18`, accessed April 4, 2023

[7] `https://github.com/QueensGambit/CrazyAra/commit/a01f62182217b2994eb64b50019d8fcc835526dd`, accessed April 4, 2023

[8] `https://github.com/QueensGambit/CrazyAra/commit/c2fcb96f7c16fd847a0a704893a206e09649d3c9`, accessed April 4, 2023

[9] `https://github.com/QueensGambit/CrazyAra/commit/7932786e0770738b3cfad23c4dea94a1ce4d2291`, accessed April 4, 2023

they all performed similarly we chose this one as the conceptually most sensible from 6.2.2),

- ClassicAra_add_var[10](another rather promising engine from 8.1 and used as the 'baseline' of additive engines),
- ClassicAra_welford_mult_stddev[11](the KataGo configuration).

Of course, in the crazyhouse tests we used the corresponding CrazyAra version instead of ClassicAra. Notably we did not include ClassicAra_welford_mult_stddev in the graphs, as it achieved Elo-Ratings in the range from -217 down to -1079. It is included mainly because this is the configuration that KataGo has used successfully [12].

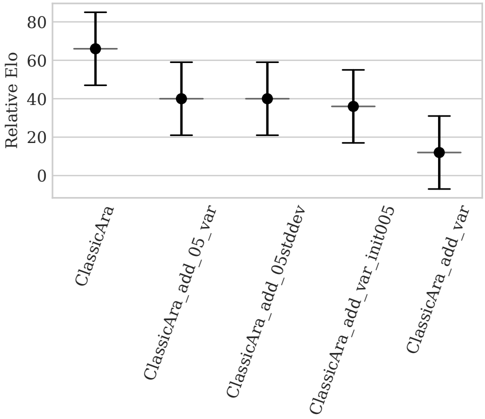

Figure 8.3: Tournament results from chess with 400 nodes (5/6 engines shown).

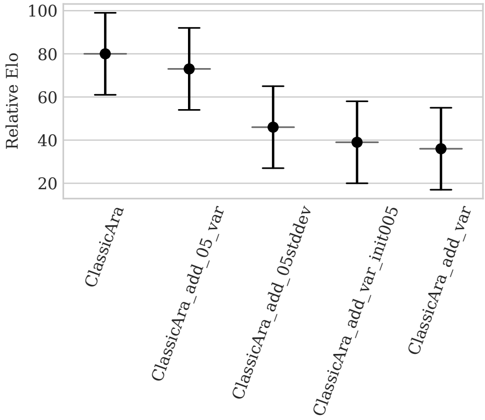

Figure 8.4: Tournament results from chess with 800 nodes (5/6 engines shown).

In every test we ran, the base engine scored the highest. The most promising modified configuration seems to be ClassicAra_add_05_var, coming second in 5 out of 6 tests. In the

---

[10] `https://github.com/QueensGambit/CrazyAra/commit/eedd94e97741fd413ffe9805f13c762d811c171a`, accessed April 4, 2023

[11] `https://github.com/QueensGambit/CrazyAra/commit/7809a2a4b1d1b27bcf818d6cf869c177238abfd5`, accessed April 4, 2023

[12] `https://github.com/lightvector/KataGo/blob/master/docs/KataGoMethods.md#dynamic-variance-scaled-cpuct`, accessed April 4, 2023

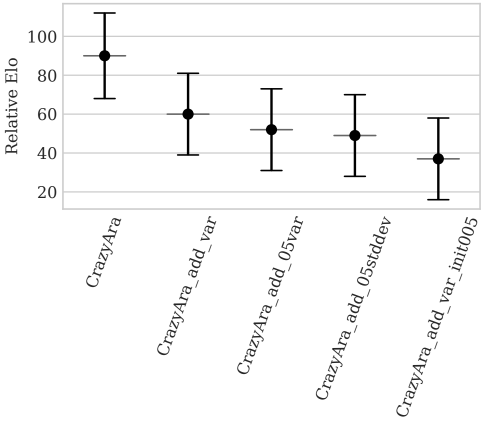

Figure 8.5: Tournament results from crazyhouse with 200 nodes (5/6 engines shown).

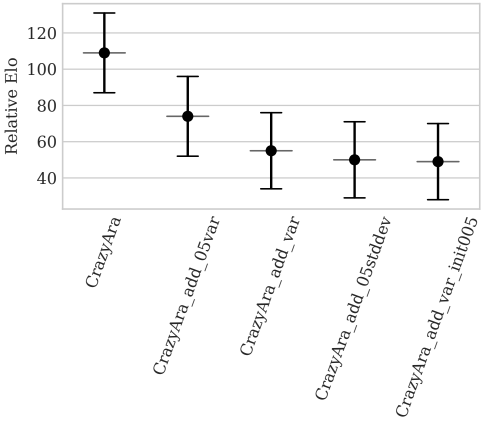

Figure 8.6: Tournament results from crazyhouse with 400 nodes (5/6 engines shown).

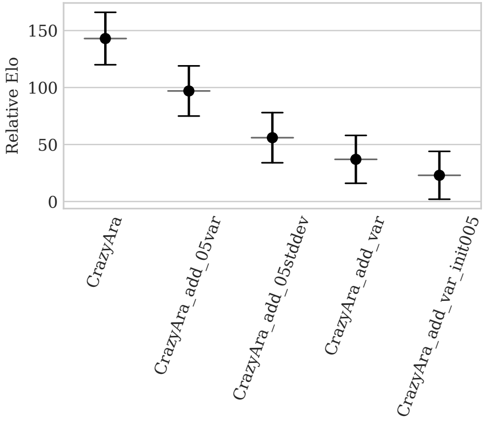

Figure 8.7: Tournament results from crazyhouse with 800 nodes (5/6 engines shown).

test where it came third (8.5), it's rating standard error was well within the standard error of the runner-up, and it had an intersection with CrazyAra's standard error. This overlap was achieved in all but one test (8.7) where it was one point short. We performed another test with ClassicAra and ClassicAra_add_05_var to see their head-to-head performance. For this the complete opening book was used, a total of 614 positions in chess and 691 positions in crazyhouse. Again, each position was played twice, so that both engines had one game with the black pieces and one game with the white pieces. Unfortunately, our modified engine still did not perform well, with the trend showing that increasing the number of nodes seems to increase the gap.

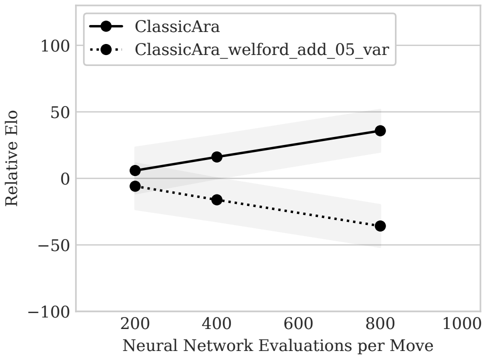

Figure 8.8: Elo evaluation from the match ClassicAra vs ClassicAra_welford_add_05_var with 200, 400, 800 nodes.

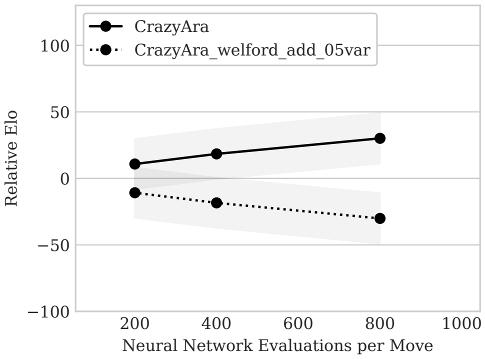

Figure 8.9: Elo evaluation from the match CrazyAra vs CrazyAra_welford_add_05_var with 200, 400, 800 nodes.

8.4 Uncertainty

Due to lack of time, we were not able to train a network with the additional uncertainty head. We did annotate the data and will give some statistics here. From our selection of training data (7.2) we got 550 files of at least 1000 games each. The dataset was kindly provided by Johannes Czech. During our analysis we encountered the problem that CrazyAra sometimes crashed (about once every 5000 games). If a file caused such a crash twice, we decided not to use it, leaving us with a total of 515 files. Three of those were assigned to be used for validation, testing and mate_in_one testing. The datasets were compressed in the Zarr[13] and z5[14] libraries, using the lz4[15] compression format[5]. Zarr has the advantage of being compatible with Python and C++, where Python is used for the data analysis (and historical implementations of CrazyAra), and C++ for the modern implementation of CrazyAra.

We appended two new arrays to each dataset, one consisting of the most recent net's[16] evaluation and the other one being the value evaluation after 400 nodes.

Since we did not train the neural network, we performed some statistical analysis on the data. We were interested in the difference between the network's evaluation and the evaluation after the search, so we took the absolute value of this difference. It showed a mean difference of 0.53 with a standard deviation of 0.34. This suggests that this approach might be promising, as there is not only a large difference between the two values, but it is also quite widely distributed, implying that there are many positions where the net suggests another move than the full search.

---

[13] `https://github.com/zarr-developers/zarr-python`, accessed April 4, 2023

[14] `https://zenodo.org/record/3585752`, accessed April 4, 2023

[15] `https://github.com/lz4/lz4`, accessed April 4, 2023

[16] `https://github.com/QueensGambit/CrazyAra/releases/download/0.9.5/CrazyAra-rl-model-os-96.zip`, accessed April 4, 2023

9 Discussion

9.1 Problems During The Evaluation

A recurring problem during our evaluation was that engines would crash, terminating the game and awarding victory to the opponent. Manual analysis of this issue suggested that this usually happened in positions that were hopelessly lost anyway, with the last evaluation before the crash usually being 'Mate in x moves'. Our first theory was that this pointed to an issue with CrazyAra's Early_Stopping option, which stops the search when only one move is available, since this move has to be played anyway. However, disabling this option did not solve the problem. We even found a position where black is in a good position (depicted in 9.1), but the black engine disconnected. The pattern that the disconnecting engine has only a single move continues, but in this case it is not a bad position. White's attack has failed and they have lost their Queen. Evaluating this position using ClassicAra with 100.000 nodes gives a score of -529cp, i.e. black leads by more than 5 pawns. The overwhelming majority of disconnects still happen in lost positions, but evidently there are exceptions. Therefore we also evaluated the matches of ClassicAra vs ClassicAra_welford_add_05_var without counting abandoned games. In most cases the result remained unchanged, but chess with 200 nodes was notably an exception again. In the match that was documented here (8.8) the score was 504-483-241 (W-L-D) in CrazyAra's favour. Without disconnections (and counting draws as half a point for both players), the result would have been 503.5-517.5, i.e. our modified version would have won. However, as described above, this is probably unfair to CrazyAra, as in most cases the disconnecting engine would have lost and so the score should remain unchanged.

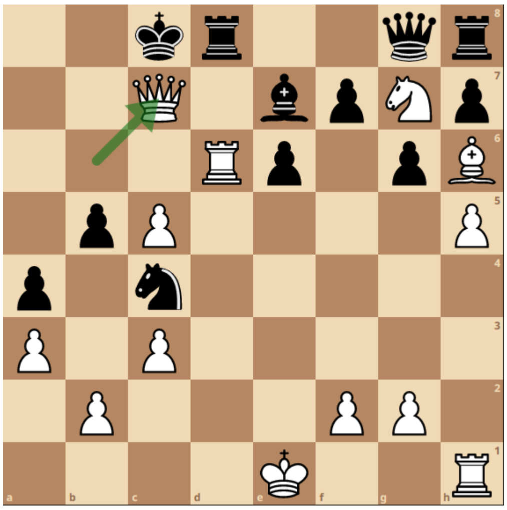

Figure 9.1: Example position where Black disconnected even though Black is winning. The arrow marks the last move.

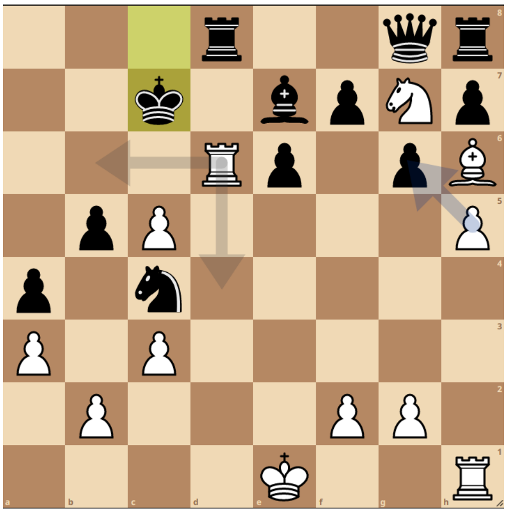

Figure 9.2: Position after Black performs the only possible move Kxc7, with an evaluation of -529cp.

9.2 Discussion Of The Results

In our final systematic tests, our modified engines failed to improve the base engine even once. The difference was usually not big, but it was there in every match. In an earlier test, we once had a match with a result of +18.7 +/- 17.7 for our modified engine, when adding 0.5 times the standard deviation in chess, with 200 nodes. However, in our final tests this intermediate result was not confirmed, and even if it had been, it would have been a single victory among many defeats. In the round-robin tournaments, the base engine scored an Elo advantage ranging from 1 to 46, coming first every time. In the head-to-head encounter against ClassicAra_add_05_var it also won all 6 matches with the Elo advantage ranging from +5.9 +/-17.4 (in chess with 200 nodes) up to +35.8 +/-15.9 (in chess with 800 nodes).

We must conclude that our modifications did not improve the base engine, and probably actually made it worse.

9.3 Possible Reasons For The Results

We will now discuss possible reasons why we did not see the desired improvement.
While the inclusion of variance did improve KataGo and was published in June 2021, to the author's knowledge it has not been replicated since. Leela Chess Zero, an open-source MCTS chess engine, also experimented with various approaches to utilize variance, however all ideas were abandoned[1]. Taken together with our result this may indicate that chess is not a suitable domain for this approach.
Audibert et al.[1] discussed MCTS algorithms with variance and stated that
"Intuitively, algorithms using variance estimates should work better than ones that do not use such estimates (like UCB1) when the variance of some suboptimal arm is much smaller than b^2."
In this context, b is the upper bound of the rewards, i.e. 1 in our case. So perhaps the variance of moves in chess (and especially in crazyhouse) is too large for variance-based approaches to work. Unfortunately no quantization of 'much smaller' has been given, so we cannot verify this theory.
There is also the possibility that more extensive parameter tuning is required, and we have not yet found the best configuration.

---

[1] `https://github.com/LeelaChessZero/lc0/discussions/1593`, accessed April 4, 2023

10 Conclusion

10.1 Summary

In this work, we evaluated different approaches to utilize variance and uncertainty in Monte-Carlo Tree Search. Our evaluation has shown that variance does not provide an overall improvement in chess and crazyhouse. Initial results of our analysis of utilizing uncertainty suggested that there might be a lot of potential, but we did not manage to generate and test a network. In retrospect, perhaps we should have abandoned the variance-approach earlier and focused on the uncertainty instead, but our continued evaluation of variance-based approaches at least resulted in an engine that is only slightly weaker than the unmodified CrazyAra/ClassicAra. This is a massive improvement over the original KataGo[1] approach, as shown in our evaluation.

10.2 Future Work

Our approach to utilize variance in MCTS did not produce the desired results. We have identified some problems that might indicate that this approach is not suitable for chess and crazyhouse. However, we have not exhaustively tested possible parameters, nor have we proven that the approach must fail. A more extensive test might yield positive results. Our first outlook on utilizing uncertainty seemed more promising; training and testing the neural network on the data we gathered could yield interesting results.

Finally, one could question the premise of our work, that in a position with high variance we should explore more. A more precise evaluation of the most promising node, by exploiting it more, might also be a reasonable approach. For this one could consider

---

[1] `https://github.com/lightvector/KataGo/blob/master/docs/KataGoMethods.md#dynamic-variance-scaled-cpuct`, accessed April 4, 2023

multiplying the variance by the Q-term instead of the U-term or finding a completely different way of scaling the U-term with the variance, perhaps antiproportionally.

Bibliography

[1] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. “Exploration–exploitation tradeoff using variance estimates in multi-armed bandits”. In: *Theoretical Computer Science* 410.19 (2009), pp. 1876–1902.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. “Finite-time analysis of the multiarmed bandit problem”. In: *Machine learning* 47 (2002), pp. 235–256.

[3] Dominik Bauer, Timothy Patten, and Markus Vincze. “Monte Carlo tree search on directed acyclic graphs for object pose verification”. In: *Computer Vision Systems: 12th International Conference, ICVS 2019, Thessaloniki, Greece, September 23–25, 2019, Proceedings 12*. Springer. 2019, pp. 386–396.

[4] Cameron B Browne et al. “A survey of monte carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[5] Johannes Czech. “Deep Reinforcement Learning for Crazyhouse”. M.Sc. TU Darmstadt, Dec. 2019, p. 54.

[6] Johannes Czech, Patrick Korus, and Kristian Kersting. “Improving AlphaZero Using Monte-Carlo Graph Search”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 31. 2021, pp. 103–111.

[7] Johannes Czech et al. “Learning to play the chess variant Crazyhouse above world champion level with deep neural networks and human data”. In: *Frontiers in Artificial Intelligence* 3 (2020), p. 24.

[8] Judith Eckle-Kohler and Michael Kohler. *Eine Einführung in die Statistik und ihre Anwendungen*. Springer-Verlag, 2017.

[9] Francis Galton. “Vox populi”. In: (1907).

[10] Chuan Guo et al. “On calibration of modern neural networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 1321–1330.

[11] https://subluminal.wordpress.com/author/jpmccusker/. *Running Standard Deviations*. URL: `https://subluminal.wordpress.com/2008/07/31/running-standard-deviations/` (visited on 04/04/2023).

[12] Levente Kocsis and Csaba 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.

[13] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. "Improved monte-carlo search". In: *Univ. Tartu, Estonia, Tech. Rep* 1 (2006), pp. 1–22.

[14] Tze Leung Lai, Herbert Robbins, et al. "Asymptotically efficient adaptive allocation rules". In: *Advances in applied mathematics* 6.1 (1985), pp. 4–22.

[15] Li-Cheng Lan et al. "Learning to stop: Dynamic simulation monte-carlo tree search". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 1. 2021, pp. 259–267.

[16] Christopher D Rosin. "Multi-armed bandits with episode context". In: *Annals of Mathematics and Artificial Intelligence* 61.3 (2011), pp. 203–230.

[17] Erich Schubert and Michael Gertz. "Numerically stable parallel computation of (co-) variance". In: *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. 2018, pp. 1–12.

[18] David Silver et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[19] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[20] David Silver et al. "Mastering the game of go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[22] Yuandong Tian et al. "Elf opengo: An analysis and open reimplementation of alphazero". In: *International conference on machine learning*. PMLR. 2019, pp. 6244–6253.

[23] Johan Ugander, Ryan Drapeau, and Carlos Guestrin. "The wisdom of multiple guesses". In: *Proceedings of the Sixteenth ACM Conference on Economics and Computation*. 2015, pp. 643–660.

[24] BP Welford. “Note on a method for calculating corrected sums of squares and products”. In: *Technometrics* 4.3 (1962), pp. 419–420.

[25] David J Wu. “Accelerating self-play learning in go”. In: *arXiv preprint arXiv:1902.10565* (2019).