Daan de Jong

Monte Carlo Tree Search with contextual bandits

Bachelor thesis

June 24, 2025

Thesis supervisors: dr. D. van der Hoeven
dr. J.N. van Rijn

Leiden University
Mathematical Institute (MI)
Leiden Institute of Advanced Computer Science (LIACS)

Abstract

Monte Carlo tree search (MCTS) has become the dominant framework in artificial intelligence for game-playing. Popular variants of MCTS, like UCT (which uses the UCB1 bandit), use multi-armed bandits to improve the selection of sub-trees that are explored. However, UCB1 relies on the assumption that the arms are stochastic, an assumption that is not met in bandit-based tree search. In this thesis, we aim to find an algorithm that improves upon UCB1. This new algorithm, contextual bandits applied to tree search (CBT), utilises contextual bandit theory to address these limitations. By treating the strategies of subsequent players as context, CBT uses contextual information to improve move selection. For a maximum tree depth of two, we give a theoretical analysis proving the CBT has a pseudo-regret of $\mathcal{O}\left(\sqrt{T\ln(T)}\right)$ and run experiments to determine real-world practicality. Empirical evaluations on a stylised game and Tic-tac-toe demonstrate that although CBT does seem to confidently identify the best move sooner than UCB1, UCB1 is still superior at finding the minimax value and recommending the best move. These results suggest that incorporating contextual bandit into MCTS may provide a promising direction for future research in tree search algorithms, notably by looking at tree search with more levels than two. But for now, they are not an improvement over UCB1.

Contents

1 Introduction

At first glance, devoting time and energy to making computer programs play board games seems meaningless. One might argue that it is better to use those resources to solve real-world problems that are more relevant. However, the development of computers that can outperform humans at playing board games has been a topic of research since scientists began studying computer science and artificial intelligence (Toosi et al., 2021). For example, the development of a chess-playing computer received decades of attention by some of the biggest names in the history of computer science, like Charles Babbage, Alan Turing, and John von Neumann (Silver et al., 2018). So much so that it has even been called the Drosophila (also known as the fruit fly) of artificial intelligence research (Ensmenger, 2012). This research culminated in the development of IBM's Deep Blue chess computer, which defeated then-reigning world champion Garry Kasparov in 1997 and became the first computer to do so (Campbell et al., 2002). Over the past few years, AI algorithms have become increasingly sophisticated, and AI solutions have become an integral part of daily life for many. It is hard to imagine that this progress would have happened without the research, breakthroughs, and technologies that came from studying the trivial problems like how to play a board game.

To continue in the tradition of scientists teaching computers how to play games, for this thesis, I will propose a new algorithm, CBT, based on (contextual) multi-armed bandit theory and try to improve upon existing game-playing algorithms. But first, we have to talk about the way these algorithms work.

1.1 Playing games

The way computers play games is similar to how people play games. The course of a game can be represented by a tree, where the vertices are possible game states, with the root the beginning state of the game, and where the edges are possible moves. Figure 1 gives an example of a part of the game tree of the game of Tic-tac-toe (this example also takes into account the symmetries of the game). Both humans and computers decide what move to play by searching this tree for end states where they win (i.e., imagining what moves the opponent could pick and so forth).

The problem is that with more complicated games, the game tree becomes too big to be searched completely, even for the most high-performance computers. The development of game-playing algorithms, therefore, focuses on finding efficient ways to search only a subset of the tree to find the best move. One of these algorithms is Monte Carlo tree search (Coulom, 2006), which uses multi-armed bandits to search the tree. In its most popular form, MCTS uses UCB1 bandits. However, there are two problems with this. UCB1 does not make use of all available

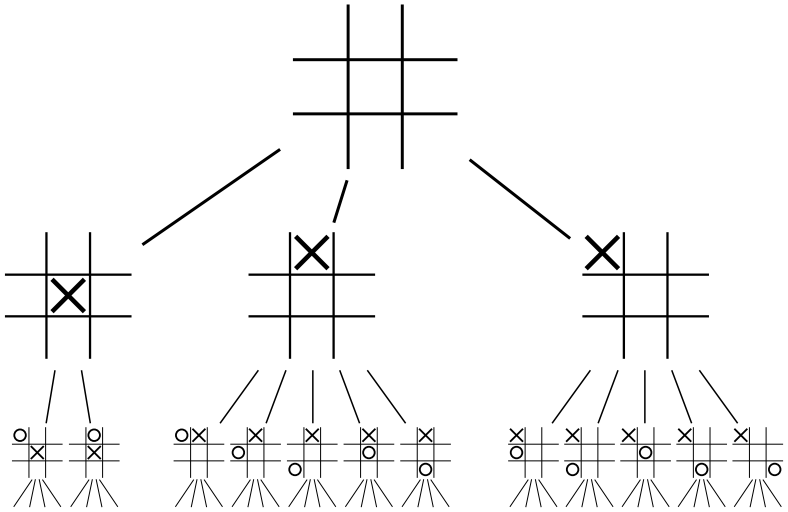

Figure 1: Example of part of the game tree of the game Tic-tac-toe (taking into account the symmetries of the game).[1]

information and has a theoretical drawback when it comes to tree search. We will talk more about these problems in Section 2.1.1 and Section 3.

1.2 Goal

To address the issues mentioned above, in this thesis, we present a new algorithm: contextual bandits applied to tree search (CBT). This algorithm applies squareCB, a contextual bandit algorithm introduced by Foster and Rakhlin (2020), to replace UCB1 in Monte Carlo tree search. The motivation for this is the fact that contextual bandits utilise extra information when choosing a subtree to further explore, and the fact that squareCB does not need to assume that the outcome of each decision is stochastic (which is not the case in MCTS). Both contextual bandits and squareCB will be further elaborated upon in Section 2.3.3. The research question is:

> *"Can we improve upon Monte Carlo tree search with UCB1 and make a more efficient tree search algorithm by applying contextual bandits in a two-player zero-sum game?"*

This can be broken into two further goals. The primary goal is to find an algorithm that is more effective at selecting the optimal move in a game. The second, less important one is finding an algorithm with stronger theoretical guarantees. However, these two goals are not unrelated, as a stronger theoretical foundation for the algorithm should lead to more robust performance in practice.

[1] Figure from https://commons.wikimedia.org/w/index.php?title=File:Tic-tac-toe-game-tree.svg

1.3 Thesis overview

The structure of the thesis is as follows. Section 2 discusses related work and gives an overview of the theory necessary to achieve the research goal; Section 3 defines the mathematical framework for our analysis and presents the algorithm that we will study in this thesis; Section 4 contains the mathematical analysis of the algorithm; Then in Section 5 the experimental analysis and accompanying methods are explained; The results of these experiments are then presented in Section 6; Finally in Section 7.1 presents the findings of the thesis and makes recommendations for future work. At the end of this report is also Appendix A with proofs for various lemmas.

Before we introduce the theory, we first make a general remark. What has become clear when researching literature is that mathematicians and computer scientists (at least in this area of research) engage with the same (or very similar) problems, but they approach these problems in completely different ways that do not align. Mathematicians discuss theoretical or worst-case upper and lower bounds, as well as statistical analysis. Meanwhile, computer scientists try as many things as possible to see what works in practice.

2 Theory

As we explained in the introduction, computers play games by searching the game tree. Several algorithms can be used for this, two of which, the minimax algorithm and Monte Carlo tree search, we will introduce in this section. Then, in the rest of this section, we will introduce the rest of the theory that is necessary before we can introduce and analyse our new algorithm.

2.1 Minimax Algorithm

In a zero-sum two-player game, nodes in the game tree have a 'game theoretical value'. Nodes that have no children are called terminal nodes and represent states in which the game has ended. They have values that show how desirable this end-state is for the player. For example, in a points-based game, this could be the number of points the player has, or with a simple win/lose game, it could be 1 if the player wins and -1 if their opponent wins. The goal of playing is to reach a terminal node with the highest value. The goal of the opponent is to reach a terminal node with the lowest value.

For non-terminal nodes, the value is recursively defined: If some node represents a state where the player has to make a move, the value of that node is equal to the highest value of any of its children, i.e., the value corresponding to the max-

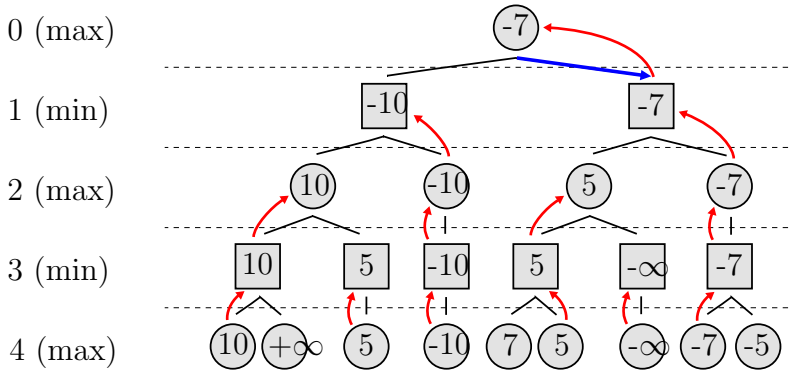

Figure 2: Example of a tree with the minimax values.[2]

imising move. If a node represents an opponent's turn, then their value is that of the child node with the smallest value (Knuth & Moore, 1975).

This gives rise to an algorithm for playing games: the minimax algorithm. The values for the nodes in the tree that correspond with each of the possible actions a player can take are calculated by minimising or maximising the values from the leaves up. The algorithm finally chooses the move with the highest value in the top level of the tree (Knuth & Moore, 1975). An example of such a calculated tree for a random game can be seen in Figure 2. Here right move, with value -7, turned out the be the optimal one for the beginning player.

For the rest of this thesis, we will call the value of a node the 'minimax value'.

2.1.1 Monte Carlo tree search

The minimax algorithms and the improved alpha-beta-pruning have been used since the 1960s (Knuth & Moore, 1975) and work well for games that are small enough so that the entire tree can be calculated, but even with improvements calculating the minimax value becomes impractical for larger game with large branching factors like Chess or Go where calculating the entire tree would simply take too much time (van den Herik et al., 2002). As a solution to this, researchers introduced random play-outs[3] and took the average outcome of those random games to be the 'value' of the move (Munos, 2014). Building on this, instead of uniformly sampling a move to play in the play outs, bandits were used to increasingly exploit the more promising subtrees (Bubeck & Cesa-Bianchi, 2012). This resulted in Monte Carlo tree search (MCTS) algorithms (Coquelin & Munos, 2007; Munos, 2014).

[2]Figure from https://commons.wikimedia.org/w/index.php?title=File:Minimax.svg

[3]A random rollout is defined as simulation of the game from the current state to a terminal state, by continuously choosing uniformly from all available moves.

Combined with an iterative deepening that expands the tree with every iteration, the general MCTS algorithm emerged as it is used today (Chaslot et al., 2008). This MCTS consists of four stages that are repeated a certain number of times, or until time runs out. The four stages are pictured in Figure 3. The first is selection, a multi-armed bandit is used at every node to select the best move, until a leaf of the tree or a terminal state is reached (the game is finished). The second stage has the tree expand with one node. The third is a random rollout up to a terminal state. Finally, in the fourth and last stage, the result of that random rollout is used to update all visited nodes with information used by the multi-armed bandit algorithm.

The most famous and widespread of these MCTS algorithms is Upper Confidence bound applied to Tree search (UCT) by Kocsis and Szepesvári (2006). In this variant, the multi-bandit algorithm upper confidence bound 1 by Auer et al. (2002) (see also Section 2.3.2) is used in the selection stage to select the branch to further explore for every node of the tree. However, the use of UCB1 has a theoretical drawback, i.e. that UCB1 is a stochastic bandit. This means that it is assumed that the outcome of each arm is stochastic with a stationary distribution and mean. In MCTS, the outcome of the random rollouts is stochastic, but the bandits that choose the moves in the next levels of the tree search are not. Their strategy changes with every iteration of the MCTS algorithm. Therefore, the stochastic assumption of UCB1 does not hold for the selection stage in MCTS.

2.2 Alphago

More recent developments are the variants of MCTS combined with neural networks: 'AlphaGo' and its successors (Silver et al., 2016, 2017, 2018). These were the first algorithms ever to beat the world's best players of the game of Go.

2.3 Multi-armed Bandits

The multi-armed bandit problem is a mathematical framework for making sequential decisions under uncertainty (Bubeck & Cesa-Bianchi, 2012), usually for T rounds. In each round $t \in [T]$, the decision maker, or learner, has to choose an action a_t out of K possible actions and based on that choice it receives a reward (or loss) $X_t \in \mathbb{R}$. The learner does not observe what the reward would have been if it had chosen a different action. The available actions are also called 'arms' that can be 'pulled', like the levers on a slot machine, hence the name multi-armed bandits (Lattimore & Szepesvári, 2020, Chapter 1). Although there are many different variations of this concept, an essential part of multi-armed bandit

[4](Świechowski et al., 2023)

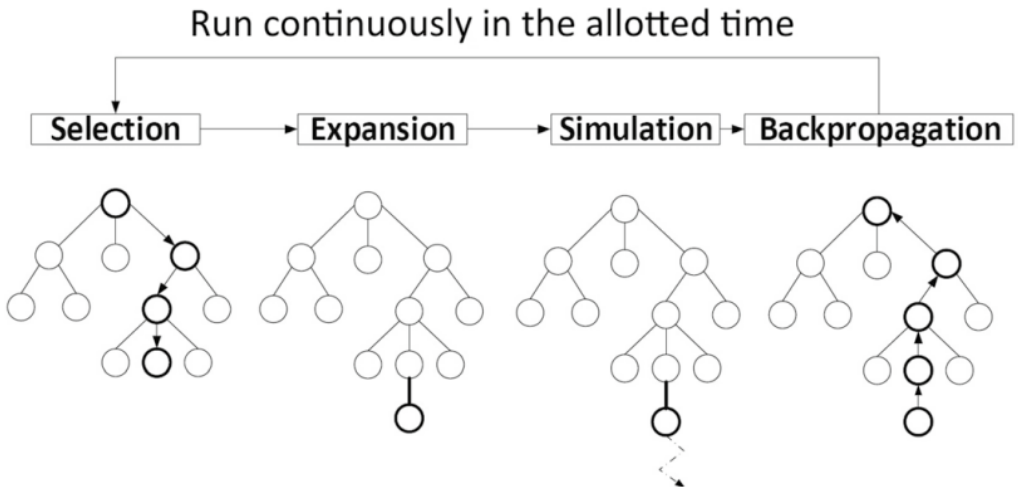

Figure 3: Overview of the Monte Carlo tree search (MCTS) algorithm cycle.[4]

problems is always the trade-off between exploration and exploitation. Here, exploration means trying different arms to find ones that give higher rewards, and exploitation means repeatedly pulling arms that had good results in previous tries to maximise rewards (or minimise losses).

The goal in a multi-armed bandit problem is usually to find a strategy for selecting arms that maximises the (expected) total reward. But it can also be to find the best arm in the least amount of rounds, or for some fixed-confidence, this is called best arm identification (Garivier & Kaufmann, 2016).

A further distinction can be made in the way the rewards are determined. The main variations are stochastic and adversarial multi-armed bandits. Stochastic means that each arm has a stochastic reward with a fixed mean and distribution that are both unknown to the learner. In adversarial bandits, the reward function is not fixed and can be different for each arm at each time step. In the last case, it may even be set at the beginning of each time step based on the previous strategy of the learner by an adversary, hence the name adversarial bandits (Bubeck & Cesa-Bianchi, 2012; Lattimore & Szepesvári, 2020).

2.3.1 Regret

When we look at multi-armed bandits that maximise rewards, we want to compare the performance of different strategies. For this, we use regret, which is the difference between the cumulative rewards of one strategy and the cumulative rewards of always playing an optimal strategy for T rounds. The precise definition depends on the way the multi-armed bandit problem is defined, and different authors also

use different definitions.

2.3.2 UCB1

The UCB1 (upper confidence bound) algorithm for the stochastic multi-armed bandit problem was first introduced by Auer et al. (2002), who proved that the use of UCB1 guarantees a worst case pseudo-regret in the order $\mathcal{O}\left(\sqrt{TK\ln(T)}\right)$, for T repeated plays and K possible arms (Auer et al., 2002; Luo, 2017b). It estimates the mean outcome of each arm, including the confidence bounds, and selects the arm with the highest confidence bound. This means it is as optimistic as possible about the means of each arm, before choosing the arm with the highest estimated mean. Because of this optimism, it is not only good in the worst case, but it is also efficient in many real-world cases.
UCB1 is a maximising algorithm; there is also an opposite variant, sometimes called lower confidence bound (LCB1), that works the same except that it minimises the lower bound instead of maximising the upper bound. Since it is fundamentally the same algorithm, we will call both variants UCB1 in this thesis. The simplicity and efficiency are the reasons it is widely used in practice and used in the UCT algorithm mentioned in Section 2.1.1.

Algorithm 1 Upper confidence bound 1 (Auer et al., 2002)

Require: Observe $K > 1$ the number of arms.
1: Play each arm $i \in [K]$ once and initialise $\bar{x}_i$ as the mean of all observed gains of arm i.
2: Set $n_i = 1$ the number of times arm i has been played.
3: **for** n=1,2,...,T **do**
4: Calculate for each arm $i \in [K]$, $A_i = \bar{x}_i + \sqrt{\frac{2\ln n}{n_i}}$.
5: Play the arm j that maximises this value: $j = \arg\max_i A_i$.
6: Observe the result x_n and update the mean of outcomes $\bar{x}_j$ of playing arm j accordingly.
7: Increment n_j by one.
8: **end for**

2.3.3 SquareCB

A variant of the multi-armed bandits is contextual bandits. The difference being that in a contextual bandit, the outcome depends not only on the choice of arm, but also on a variable, called the 'context', the shape of this variables depends on the exact model, that is observed by the algorithm and known before the algorithm chooses an arm. Furthermore, in the contextual bandits that we will consider, the

losses or rewards are not stochastic any more. That means they are not drawn from fixed distributions, but instead the distributions depend on the context and can vary for different time steps t.

A possible way to approach a contextual bandit problem is by using an oracle. An oracle is a procedure to predict the outcome of a move given a certain context (Agarwal et al., 2014). The outcome is then used to choose the move according to some policy. Such an oracle is essentially a supervised learning problem, which is thoroughly studied.

One recent contribution to the study of contextual bandit is a paper by Foster and Rakhlin (2020), where they introduce the (adversarial) algorithm squareCB. This algorithm is a reduction of a contextual bandit problem to an online squared loss regression problem. Their main result is Theorem 1, presented below:

Theorem 1. *Supposing some assumptions (which can be found in the paper (Foster & Rakhlin, 2020)), for any $\delta > 0$ and with hyperparameters $\mu = K$ and $\gamma = \sqrt{KT/(R_{sq}(T) + \log(2\delta^{-1}))}$, the regret of squareCB is bounded, with probability $1 - \delta$ by*

$$4\sqrt{KT \cdot R_{sq}(T)} + 8\sqrt{KT \log(2\delta^{-1})}$$

Where $R_{sq}(T)$ is a known regret bound of the online square regression algorithm used as an oracle.

This means that we can use an online regression algorithm that is known to be efficient and apply it to get an efficient contextual bandit algorithm with regret bound guarantees.

In this thesis, we will modify squareCB for our scenario, which means that our main analytical result, Theorem 3, is very similar to the theorem of Foster and Rakhlin (2020). To do this, we will also need a regression oracle. Thus, it is necessary to also talk about online regression, or in our case, online linear regression.

2.4 Online linear regression

Online linear regression is an online variant of linear regression. In ordinary regression, we observe all data immediately and can then estimate the regression parameters from the data and use those to make predictions. Online regression observes the data points one by one and updates the coefficients sequentially, using them to predict the new data point. The procedure is as follows, for online linear regression with n independent variables (Schapire, 2018):

- Initialize the regression coefficients $\theta_1 \in \mathbb{R}^n$.
- Repeat for $t = 1, ..., T$:

- Observe $x_t \in \mathbb{R}^n$.
- Predict the outcome $\widehat{y}_t = \langle \theta_t, x_t \rangle$.
- Observe the real outcome $y_t \in \mathbb{R}$.
- Update the coefficients θ_t accordingly.

Here $\langle \theta_t, x_t \rangle = \theta_t^\top x_t$ means the dot product, or inner product of the vectors θ_t and x_t.

The goal is to minimise the total 'loss' according to some loss function $\ell_t(\theta_t)$, for example, a square loss function $\ell_t(\theta_t) = (y_t - \langle \theta_t^\top, x_t \rangle)^2$.

We measure the performance of online regression by looking at the regret:

Definition 1. For an online linear regression with loss function ℓ_t, n the amount of independent variables and the regression coefficients θ_t at time t, the regret is defined as:

$$\text{Regret}(T) = \sum_{t=1}^{T} \ell_t(\theta_t) - \min_{\theta \in \mathbb{R}^n} \sum_{t=1}^{T} \ell_t(\theta)$$

2.4.1 Online ridge regression

One of the algorithms to perform online linear regression is online ridge regression, a generalisation of the ordinary least squares method (Arce & Salinas, 2012). The explicit formula for the regression coefficients θ in ridge regression is shown in Equation 1 as described by Vovk (2001), which is equivalent to Least Squares when $\alpha = 0$.

$$\theta = \left(\sum_{t=1}^{T} y_t x_t \right)^\top \left(\alpha I + \sum_{t=1}^{T} x_t x_t^\top \right)^{-1} \tag{1}$$

This can be transformed into an online algorithm (i.e. an algorithm that updates the regression coordinates after each observation) by defining the following two variables:

$$A \stackrel{\text{def}}{=} \alpha I + \sum_{t=1}^{T} x_t x_t^\top$$

$$b \stackrel{\text{def}}{=} \sum_{t=1}^{T} y_t x_t$$

The weights vector is then equal to $\theta = b^\top A^{-1}$. In the online version of the algorithm, the variables A and b are updated sequentially with each new observation. The complete online ridge regression is then as shown in Algorithm 2:

Algorithm 2 Online ridge regression (Arce & Salinas, 2012; Vovk, 2001)

Require: A hyperparameter α.
1: Set $A \leftarrow \alpha I$ and $b \leftarrow 0$.
2: **for** $t = 1, 2, ...$ **do**
3: Observe x_t.
4: Predict the outcome $\hat{y}_t \leftarrow b^\top A^{-1} x_t$.
5: Observe real outcome y_t.
6: Update $A \leftarrow A + x_t x_t^\top$.
7: Update $b \leftarrow b + y_t x_t$.
8: **end for**

Computationally, the most expensive part of this algorithm is the matrix inversion on line 4. This part can be sped up by using the Sherman-Morrison formula as found in Cesa-Bianchi and Lugosi (2006, Chapter 11), which removes the need for a matrix inversion. The formula is shown in equation 2 below. This equation replaces the update rule on line 4 of Algorithm 2.

$$A_t^{-1} = A_t^{-1} - \frac{\left(A_{t-1}^{-1} x_t\right)\left(A_{t-1}^{-1} x_t\right)^\top}{1 + x_t^\top A_{t-1}^{-1} x_t} \tag{2}$$

When playing this algorithm, Cesa-Bianchi and Lugosi (2006, Chapter 11) show that the regret compared to using any vector of weights u is bounded:

Theorem 2 (Cesa-Bianchi and Lugosi, 2006)**.** *For some sequence of* $(x_t, y_t) \in \mathbb{R}^d \times \mathbb{R}$*, for any* $u \in \mathbb{R}^d$ *and* $T \geq 1$*, and the loss function defined as square loss* $\ell(u) = \frac{1}{2}(ux_t - y_t)^2$*, the regret of playing ridge regression is bounded by:*

$$\sum_{t=1}^{T} \ell_t(\theta_t) - \sum_{t=1}^{T} \ell_t(u) \leq \frac{1}{2}\|u\|^2 + d \ln\left(1 + \frac{T}{d} \cdot \max_{t=1,\ldots,T} \|x_t\|^2\right) \cdot \max_{t=1,\ldots,T} \ell_t(\theta_{t-1})$$

Where $||\cdot||$ *denotes the Euclidean norm.*

In the case that $\ell_t(u)$, and $\|u\|$ are bounded by some constant and $\max_{t\in[T]} \|x_t\|^2 \leq X$ is bounded too, the regret bound for the algorithm is: $\mathcal{O}\left(d \ln\left(1 + \frac{TX^2}{d}\right)\right)$.

3 Definitions

The essence of Monte Carlo tree search is bandits playing a game against each other for a certain number of repetitions. Combined with random rollouts. Later (in Section 3.2), we will show that this leads to the outcome of the repeated games

to converge to the minimax value and the strategy of the bandits to approach an optimal strategy. We achieve this with a proof based on Luo (2017a).

The goal of this thesis is to develop a better algorithm for zero-sum, two-player games. In MCTS, the bandits choose their strategy based on the outcome of previous repetitions. Therefore, when the bandit for the first level in the tree decides the strategy for the first move, it already knows the strategy for the bandits that pick the second move. We can use this information to improve the performance of the bandit by choosing its strategy based on the (known) strategy of the next player. We will use this in the algorithm that is presented later in this section. For simplicity, this algorithm will only look at two-ply games first (so that both players only make one move before the game ends).

Before we show the algorithm, we first need to define some notation.

3.1 Problem setting

Before we can introduce our algorithm, we must first define the problem setting.

We consider a zero-sum game between two players that is repeated for T times, where players take turns to play a move. One repetition consists of the following events:

1. The first player observes what strategy the second player will play. A strategy is a probability vector that assigns some probability to each of the possible moves. The second player has a strategy for each of the possible moves the first player can choose. All these strategies are defined by the symbol p_t^i for some time t and some move by the first player i. What the first player observes is therefore a vector of vectors $p_t = (p_t^i)_{i \in [K]}$.

2. Player 1 chooses a strategy w_t for round t, and a move is sampled from this distribution. This sampled move, denoted by i_t, is then played.

3. A move is sampled for the second player's turn, called h_t. It is sampled from the distribution $p_t^{i_t}$ that was already known in step 1. Note here that the action space (the set of moves available to the second player) depends on the move chosen by player 1. The action space is denoted by $\mathcal{H}^i$ for some first player move i, and therefore it is not generally true that $\mathcal{H}^i = \mathcal{H}^j$ for two different $i, j \in [K]$.

4. Then, both players observe the outcome of the game called $X_t(i_t, h_t) \in [0, 1]$ which is stochastic and sampled from some distribution with mean $\mu(i_t, h_t)$, so that for given i_t, h_t the expected outcome of the game is: $\mathbb{E}\left[X_t(i_t, h_t) \mid i_t, h_t\right] = \mu(i_t, h_t)$.

5. Lastly, if this has been repeated for fewer than T times, the whole process starts over again, beginning with step 1.

The goal of the first player is to choose the strategy w that maximises the expected outcome. Since we are looking at zero-sum games, the first player their reward is the second player their loss, so the second player their objective is to choose a strategy p that minimises the expected outcome.

3.2 Bandits and minimax

Next, we will show, using a modified analysis of Luo (2017a), that when both bandits' regret is bounded and sub-linear over time, the outcome of the scenario introduced in the previous section will converge to the minimax value. For this, we first introduce some notation. When bandits choose their respective strategies w_t and p_t from which their moves i_t and h_t will be sample, the expected outcome of the game is $\mathbb{E}[X_t(i_t, h_t)] = \sum_{i\in[K]} w_t^i \cdot \sum_{h\in\mathcal{H}^i} p_t^{i,h} \cdot \mu(i,h)$ (where the expectation is also the expectation over the randomness of i_t and h_t). Instead of this we will abuse the notation a little, and write $\mu(w_t, p_t) \overset{\text{def}}{=} \sum_{i\in[K]} w_t^i \cdot \sum_{h\in\mathcal{H}^i} p_t^{i,h} \cdot \mu(i,h)$ with w_t being a vector and p_t being a vector of vectors. For readability we also introduce the notation $\mathcal{W} = \Delta^K$ and $\mathcal{P} = \Delta^{|\mathcal{H}^1|} \times ... \times \Delta^{|\mathcal{H}^K|}$ for the sets of all possible values of w_t and p_t respectively.

Definition 2. The minimax value is defined as shown in Equation 3 below. It is the expected outcome of the game when player 1 chooses a strategy to minimise the outcome and player 2 chooses a strategy that maximises the outcome of the game:

$$\min_{w\in\mathcal{W}} \max_{p\in\mathcal{P}} \mu(w,p) \tag{3}$$

Next, we need to define the regret of both bandits. We use the pseudo-regret in our analysis.

Definition 3. The pseudo-regret of player one is the difference between the expected outcomes of the chosen strategies w_t for $t = 1, 2, ..., T$ and the expected outcome of playing the optimal strategy every time. We call the pseudo-regret for player one:

$$\bar{R}_{\text{player 1}}(T) = \mathbb{E}\left[\sum_{t=1}^{T} \mu(w_t, p_t)\right] - \min_{w\in\mathcal{W}} \mathbb{E}\left[\sum_{t=1}^{T} \mu(w, p_t)\right] \tag{4}$$

Inversely, the loss for the first player is the gain for the second player, so the goal of the second player is to maximise the outcome. The pseudo-regret for the second player is therefore:

Definition 4. The pseudo-regret of player one is the difference between the expected outcomes of the chosen strategies p_t for $t = 1, 2, ..., T$ and the expected outcome of playing the optimal strategy every time. We call the pseudo-regret $\bar{R}_{\text{player 2}}(T)$:

$$\bar{R}_{\text{player 2}}(T) = \max_{p \in \mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{T} \mu(w_t, p) - \sum_{t=1}^{T} \mu(w_t, p_t)\right] \tag{5}$$

Suppose both players play with algorithms that have pseudo-regret $\bar{R}_{\text{player 1}}(T)$ and $\bar{R}_{\text{player 2}}(T)$. The difference between the average expected outcome and the minimax value can then be bounded:

$$\begin{aligned}
\min_{w \in \mathcal{W}} \max_{p \in \mathcal{P}} \mu(w, p) &= \min_{w \in \mathcal{W}} \max_{p \in \mathcal{P}} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\mu(w, p)\right] \\
&\leq \max_{p \in \mathcal{P}} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\mu(w_t, p)\right] \\
&= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\mu(w_t, p_t)\right] \\
&\quad + \frac{1}{T} \max_{p \in \mathcal{P}} \sum_{t=1}^{T} \left(\mathbb{E}\left[\mu(w_t, p)\right] - \mathbb{E}\left[\mu(w_t, p_t)\right]\right) \\
&= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\mu(w_t, p_t)\right] + \frac{1}{T} \max_{p \in \mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{T} \left(\mu(w_t, p) - \mu(w_t, p_t)\right)\right] \\
&= \mathbb{E}\left[\frac{1}{T} \sum_{t=1}^{T} \mu(w_t, p_t)\right] + \frac{1}{T} \bar{R}_{\text{player 2}}(T)
\end{aligned}$$

So we get the inequality:

$$\min_{w \in \mathcal{W}} \max_{p \in \mathcal{P}} \mu(w, p) - \mathbb{E}\left[\frac{1}{T} \sum_{t=1}^{T} \mu(w_t, p_t)\right] \leq \frac{\bar{R}_{\text{player 2}}(T)}{T} \tag{6}$$

Similarly, for the first player, we can do:

$$\begin{aligned}
\min_{w \in \mathcal{W}} \max_{p \in \mathcal{P}} \mu(w, p) &= \min_{w \in \mathcal{W}} \max_{p \in \mathcal{P}} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\mu(w, p)\right] \\
&\geq \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\mu(w, p_t)\right]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\mu(w_t, p_t)\right] + \min_{w\in\mathcal{W}}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\mu(w, p_t) - \mu(w_t, p_t)\right] \\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\mu(w_t, p_t)\right] + \frac{1}{T}\min_{w\in\mathcal{W}}\mathbb{E}\left[\sum_{t=1}^{T}\left(\mu(w, p_t) - \mu(w_t, p_t)\right)\right] \\
&= \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\mu(w_t, p_t)\right] - \frac{1}{T}\bar{R}_{\text{player 1}}(T)
\end{aligned}
$$

Which we can rearrange to:

$$
-\frac{\bar{R}_{\text{player 1}}(T)}{T} \leq \min_{w\in\mathcal{W}}\max_{p\in\mathcal{P}}\mu(w, p) - \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\mu(w_t, p_t)\right]
$$

Combining the previous inequality with Equation 6, we get:

$$
-\frac{\bar{R}_{\text{player 1}}(T)}{T} \leq \min_{w\in\mathcal{W}}\max_{p\in\mathcal{P}}\mu(w, p) - \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\mu(w_t, p_t)\right] \leq \frac{\bar{R}_{\text{player 2}}(T)}{T} \tag{7}
$$

From Equation 7 we can see that, if the regret from both players goes down sub-linearly over time, the denominators grow quicker than the numerators and therefore the fractions $\bar{R}_{\text{player 1}}(T)/T$ and $\bar{R}_{\text{player 2}}(T)/T$ get closer to zero and the expected average result $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\mu(w_t, p_t)\right]$ of the strategies that the algorithm plays get closer to the minimax strategy over time as well.

Using bandits with known bounds on the regret means we also have a bound on convergence to the minimax solution. Therefore, finding bandit algorithms with tighter regret bounds will also enable our tree search to converge more quickly.

3.3 The CBT algorithm

At the beginning of this section, we address a setting where the first bandit has access to the strategy that the second bandit will play, and may use this to improve its own strategy. To take advantage of this insight, we can use a contextual bandit instead of a normal multi-armed bandit, and provide the strategy of the next player as context. For this, we introduce a new algorithm: Contextual Bandit applied to Tree search (CBT), which is an adaptation of the squareCB algorithm by Foster and Rakhlin (2020) introduced in Section 2.3.3.

Now that we know the exact setting, it becomes possible to define the algorithm itself, which is done in Algorithm 3.

Algorithm 3 Contextual Bandits applied to Tree search

Require: An exploration hyperparameter $\nu > 0$, a learning rate $\gamma > 0$, and a regression oracle with weights $\hat{\mu}_t = (\hat{\mu}_t^i)_{i\in[K]}$ at time t and an online regression algorithm to update the weights after each round.

1: **for** $t \in [T]$ **do**
2: Observe p_t the strategy of the next player at time t.
3: Predict the outcome $\pi_{t,i} \leftarrow \langle p_t^i, \hat{\mu}_t^i \rangle$ of each arm, using the regression oracle.
4: Find the best arm $j_t \leftarrow \arg\min_{i\in[K]} \pi_{t,i}$.
5: For $i \neq j_t$ let $w_t^i = \frac{1}{\nu+\gamma(\pi_{t,i}-\pi_{t,j_t})}$, for arm j_t let $w_t^{j_t} = 1 - \sum_{i\neq j_t} w_t^i$.
6: Play by strategy w_t and sample a move $i_t \sim w_t$ to play at time t.
7: Observe the move by the next player $h_t \sim p_t^{i_t}$ and observe outcome $X_t(i_t, h_t)$.
8: Update the regression weights for each of the K regressions with the observed outcome, according to the update rule of the online regression algorithm provided in the line at the beginning of this algorithm.
9: **end for**

The algorithm has two hyperparameters that we can tune, the first ν controlling the exploration; setting it to a higher value means the assigned weights are closer together and therefore all arms are played more evenly. Second is the learning rate γ; setting this higher means that the weights will be farther apart, giving higher probability to the most promising arms.

Notice that the algorithm is almost the same as squareCB by Foster and Rakhlin (2020). The difference being that Algorithm 3 does K regressions at the same time, instead of one. Since it has one coefficient for every possible pair of moves, and the independent variable that corresponds with each of those coefficients is the probability of the second player making that move, each of these coefficients $\hat{\mu}_t^{i_t,h_t}$ is essentially the estimated mean of the outcome of playing moves i and h at time t. Therefore, as t increases, for each $i \in [K], h \in \mathcal{H}^i$ the estimation $\hat{\mu}_t^{i,h}$ will approach the real mean $\mu(i, h)$.

4 Analysis

We begin with a theoretical analysis of the performance of the algorithm by stating and proving bounds for the pseudo-regret of both players, as defined in the previous section.

4.1 Analysis of the first player

The first thing we are interested in is how well the first player plays in our scenario outlined in Section 3.1 and playing according to our algorithm, CBT. In this subsection, we state the guarantee on the upper bound of the pseudo-regret for player 1 and prove it using an adaptation of the proof of Foster and Rakhlin (2020). The guarantee can be found in the following Theorem:

Theorem 3 (First player)**.** *Given a standard square loss, real-valued online linear regression model with pseudo-regret bound $\bar{R}_{regression}(T)$, K possible moves and T rounds, set $\nu = K$ and $\gamma = \sqrt{2KT/\bar{R}_{regression}(T)}$. The pseudo-regret for player 1 by playing Algorithm 3 is bounded by:*

$$\bar{R}_{player\ 1}(T) \leq \frac{5}{4}\sqrt{2KT \cdot \bar{R}_{regression}(T)} \tag{8}$$

Notice that this bound is, for a large part, dependent on the regret guarantee for whichever regression is used. However, it does not depend on the strategy of the second player at all. Furthermore, it grows with the square root of both the number of actions $\sqrt{K}$ and the square root of time $\sqrt{T}$.

Before we go on to prove Theorem 3, we will state here some lemmas that are used and for which the proofs can be found in Appendix A. Beginning with a basic inequality found in Lemma 1, which we be used throughout the proof.

Lemma 1. *For any three real numbers $a, b, c \in \mathbb{R}$, we have:*

$$ca - ba^2 \leq \frac{c^2}{4b}$$

We closely follow the proof from Foster and Rakhlin (2020); therefore, we need a version of their Lemma 2, here broken down into two lemmas. The first breaks down the expectations of all random variables and restates them as a function of their probability distribution.

Lemma 2. *With the pseudo-regret defined as in Definition 4 and all variables defined as in Section 3.1, the following equality holds:*

$$\bar{R}_{player\ 1}(T) = \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\left\langle p_t^i, \mu^i\right\rangle - \left\langle p_t^{i^*}, \mu^{i^*}\right\rangle\right)\right]$$

The final lemma restates the pseudo-regret of the online linear regression to a form that we can use:

Lemma 3. *With all variables defined as before, the following equality is true:*

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i\rangle - \langle p_t^i, \mu^i\rangle\right)^2\right]$$
$$= \mathbb{E}\left[\sum_{t=1}^{T}\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \sum_{t=1}^{T}\left(\langle p_t^{i_t}, \mu^{i_t}\rangle - X_t(i_t, h_t)\right)^2\right]$$

We also assume that there exists some upper bound $\bar{R}_{\text{reg}}(T)$ on the regression regret. As said before in Section 2.4, online linear regression is a well-studied problem, and there are many algorithms with known regret guarantees. Here we simply assume that this guarantee exists, without assuming anything about the algorithm that is chosen. This approach allows us to pick any regression algorithm later on and makes the result from Theorem 3 agnostic to the choice of regression algorithm. This regret bound is formalised in the following equation:

$$\mathbb{E}\left[\sum_{t=1}^{T}\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \sum_{t=1}^{T}\left(\langle p_t^{i_t}, \mu^{i_t}\rangle - X_t(i_t, h_t)\right)^2\right] \leq \bar{R}_{\text{reg}}(T) \tag{9}$$

We are now ready to prove Theorem 3.

Proof of Theorem 3. The proof is a close adaptation of Lemma 3 of (Foster & Rakhlin, 2020). We start by combining the regret bound in Equation 9 with Lemma 3. We then get

$$0 \leq \bar{R}_{\text{reg}}(T) - \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i\rangle - \langle p_t^i, \mu^i\rangle\right)^2\right]$$

We can then multiply this by some $\eta \geq 0$ (for which we will choose an appropriate value later), and add to this the equation from Lemma 2. This gives us:

$$\begin{aligned}\bar{R}_{\text{player 1}}(T) &\leq \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\langle p_t^i, \mu^i\rangle - \langle p_t^{i^*}, \mu^{i^*}\rangle\right)\right] \\ &\quad + \eta\bar{R}_{\text{reg}}(T) - \eta\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i\rangle - \langle p_t^i, \mu^i\rangle\right)^2\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\langle p_t^i, \mu^i\rangle - \langle p_t^{i^*}, \mu^{i^*}\rangle - \eta\left(\langle p_t^i, \hat{\mu}_t^i\rangle - \langle p_t^i, \mu^i\rangle\right)^2\right)\right]\end{aligned} \tag{10}$$

$$+ \eta \bar{R}_{\text{reg}}(T)$$

What remains to be bound is the inside of the expectation. We do this by first looking at any single arbitrary value for t. We start by looking at the cases when $i = i^*$ and when $i \neq i^*$:

$$\begin{aligned}
&\sum_{i \in [K]} w_t^i \left((\langle p_t^i, \mu^i \rangle - \langle p_t^{i^*}, \mu^{i_t^*} \rangle) - \eta (\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^i, \mu^i \rangle)^2 \right) \qquad (11) \\
&\quad = w_t^{i^*} \left((\langle p_t^{i^*}, \mu^{i^*} \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle) - \eta (\langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle)^2 \right) \\
&\quad + \sum_{i \in [K], i \neq i^*} w_t^i \left((\langle p_t^i, \mu^i \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle) - \eta (\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^i, \mu^i \rangle)^2 \right) \\
&\quad = -\eta w_t^{i^*} (\langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle)^2 \\
&\quad + \sum_{i \in [K], i \neq i^*} w_t^i \left((\langle p_t^i, \mu^i \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle) - \eta (\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^i, \mu^i \rangle)^2 \right) \\
&\quad = (\star)
\end{aligned}$$

Where we can use Lemma 1 to reduce the inside of the sum:

$$\begin{aligned}
&\langle p_t^i, \mu^i \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle - \eta (\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^i, \mu^i \rangle)^2 \\
&\quad = \langle p_t^i, \hat{\mu}_t^i \rangle + \langle p_t^i, \mu^i \rangle - \langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle - \eta (\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^i, \mu^i \rangle)^2 \\
&\quad \leq \langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle + \frac{1}{4\eta} \qquad (12)
\end{aligned}$$

Because only the first term of line 12 is dependent on i, when we plug this back into the main equation, we get:

$$\begin{aligned}
(\star) &\leq -\eta w_t^{i^*} (\langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle)^2 \\
&\quad + \sum_{i \in [K], i \neq i^*} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle + \frac{1}{4\eta} \right) \\
&= -\eta w_t^{i^*} (\langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle)^2 \\
&\quad + \sum_{i \in [K], i \neq i^*} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle + \langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle - \langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle + \frac{1}{4\eta} \right) \\
&= \left(1 - w_t^{i^*} \right) \left(\langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle \right) - \eta w_t^{i^*} (\langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle - \langle p_t^{i^*}, \mu^{i^*} \rangle)^2 + \frac{1 - w_t^{i^*}}{4\eta} \\
&\quad + \sum_{i \in [K], i \neq i^*} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i \rangle - \langle p_t^{i^*}, \hat{\mu}_t^{i^*} \rangle \right)
\end{aligned}$$

$$\leq \frac{\left(1-w_t^{i^*}\right)^2}{4\eta w_t^{i^*}} + \frac{1-w_t^{i^*}}{4\eta}$$
$$+ \sum_{i\in[K],i\neq i^*} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i\rangle - \langle p_t^{i^*}, \hat{\mu}_t^{i^*}\rangle\right) \tag{13}$$

We have now bounded Equation 11 by an equation that does not use μ any more. Remember that μ is not known to the player of the algorithm. Instead, the remaining equation 13 only uses $\hat{\mu}_t$, a value that is known by – and used in – the algorithm. Furthermore, the part $\langle p_t^i, \hat{\mu}_t^i\rangle$ is exactly the $\pi_{t,i}$ calculated in line 3 from Algorithm 3.

In the next step we want to expand this to also include $\pi_{t,j_t} = \langle p^{j_t}, \hat{\mu}^{j_t}\rangle$, where j_t is the same as in line 4 of the algorithm, so we can use the definition of w_t^i. To that goal, let $j_t \in [K]$ be $\arg\min_{i\in[K]} \pi_{t,i}$ and define $u_{t,i} = \langle p^i, \hat{\mu}^i\rangle - \langle p^{j_t}, \hat{\mu}^{j_t}\rangle \geq 0$. Note that $u_{t,j_t} = 0$; we will use this later on. We now continue with only the sum of Equation 13 to also remove the references to i^*, which is also unknown to the algorithm at the time of playing:

$$\begin{aligned}
\sum_{i\in[K],i\neq i^*} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i\rangle - \langle p_t^{i^*}, \hat{\mu}_t^{i^*}\rangle\right) &= \sum_{i\in[K],i\neq i^*} w_t^i \left(\langle p_t^i, \hat{\mu}_t^i\rangle - \langle p_t^j, \hat{\mu}_t^j\rangle\right. \\
&\quad \left. + \langle p_t^j, \hat{\mu}_t^j\rangle - \langle p_t^{i^*}, \hat{\mu}_t^{i^*}\rangle\right) \\
&= \sum_{i\in[K],i\neq i^*} w_t^i \left(u_{t,i} - u_{t,i^*}\right) \\
&= \sum_{i\in[K],i\neq i^*} w_t^i u_{t,i} - \left(1-w_t^{i^*}\right) u_{t,i^*} \\
&= \sum_{i\in[K],i\neq j} w_t^i u_{t,i} - u_{t,i^*} \\
&= \sum_{i\in[K],i\neq j} \frac{u_{t,i}}{\nu+\gamma u_{t,i}} - u_{t,i^*} \qquad (14)
\end{aligned}$$

In the last line, w_t^i is simply substituted for its definition from Algorithm 3. The fraction from line 14, can be further bounded by $\frac{u_{t,i}}{\nu+\gamma u_{t,i}} \leq \frac{u_{t,i}}{\gamma u_{t,i}} = \frac{1}{\gamma}$, meaning:

$$\sum_{i\in[K],i\neq j_t} \frac{u_{t,i}}{\nu+\gamma u_{t,i}} \leq \sum_{i\in[K],i\neq j_t} \frac{1}{\gamma} = \frac{K-1}{\gamma} \tag{15}$$

Now combining Equations 13, 14 and 15 we then finally get the following regret bound for 11:

$$\frac{K-1}{\gamma} - u_{t,i^*} + \frac{\left(1-w_t^{i^*}\right)^2}{4\eta w_t^{i^*}} + \frac{1-w_t^{i^*}}{4\eta} \leq \frac{K-1}{\gamma} - u_{t,i^*} + \frac{1}{4\eta w_t^{i^*}} + \frac{1}{4\eta} \tag{16}$$

For the last part of the proof, we look at the remaining middle two terms and distinguish between the cases where $j = i^*$ and $j \neq i^*$ and use the definition of w from algorithm 3. In the first case, we get:

$$\begin{aligned}
-u_{t,i^*} + \frac{1}{4\eta w_t^{i^*}} &= -u_{t,j_t} + \frac{1}{4\eta\left(1 - \sum_{i \neq j_t} w_t^i\right)} \\
&= 0 + \frac{1}{4\eta\left(1 - \sum_{i \neq j_t} \frac{1}{\nu + \gamma u_{t,i}}\right)} \\
&\leq \frac{1}{4\eta\left(1 - \sum_{i \neq j_t} \frac{1}{\nu}\right)} = (\star)
\end{aligned}$$

When we choose $\nu = K$ (as we do in Theorem 3) this becomes:

$$\begin{aligned}
(\star) &= \frac{1}{4\eta\left(1 - \sum_{i \neq j_t} \frac{1}{K}\right)} \\
&= \frac{1}{4\eta\left(1 - \frac{K-1}{K}\right)} \\
&= \frac{1}{4\eta\frac{1}{K}} \\
&= \frac{K}{4\eta}
\end{aligned}$$

In the second case, when $j_t \neq i^*$, and also using $\nu = K$ we get:

$$\begin{aligned}
-u_{t,i^*} + \frac{1}{4\eta w_t^{i^*}} &= -u_{t,i^*} + \frac{\nu + \gamma u_{t,i^*}}{4\eta} \\
&= \frac{K}{4\eta} + u_{t,i^*}\left(\frac{\gamma}{4\eta} - 1\right)
\end{aligned}$$

Now choosing $\eta = \frac{\gamma}{4}$ and putting everything together we get the final bound for 11:

$$\frac{K-1}{\gamma} + \frac{K}{4\eta} + \frac{1}{4\eta} = \frac{2K}{\gamma} \tag{17}$$

Plugging this into Equation 10 gives the final regret bound:

$$\bar{R}_{\text{player 1}}(T) \leq \mathbb{E}\left[\sum_{t=1}^{T} \frac{2K}{\gamma}\right] + \frac{\gamma}{4}\bar{R}_{\text{reg}}(T) = \frac{2KT}{\gamma} + \frac{\gamma}{4}\bar{R}_{\text{reg}}(T)$$

Choosing $\gamma = \sqrt{2KT/\bar{R}_{\text{reg}}(T)}$ gives the equation stated in the Theorem. □

4.1.1 The regression oracle

Now that Theorem 3 has been proved, we still need some value for $\bar{R}_{\text{reg}}(T)$ to get a bound for the pseudo-regret of player 1. In order to do this, we need to choose a regression algorithm for the oracle. For this, we choose the ridge regression algorithm from Section 2.4.1. However, getting a value for $\bar{R}_{\text{reg}}(T)$ is not as simple as plugging in the result from Theorem 2. The first player is doing K ridge regressions in parallel, one for each arm that it can play. The regret bound that this gives and the proof are summarised in the theorem below:

Theorem 4. *A player that is playing using the CBT algorithm, with ridge regression to estimate the coefficients $\hat{\mu}$, will have a pseudo-regret of at most:*

$$\bar{R}_{player\ 1}(T) \leq \mathcal{O}\left(\frac{5}{4}K \cdot \sqrt{2TD \cdot \ln\left(1 + \frac{T}{D}\right)}\right)$$

With d being an upper bound on $|\mathcal{H}^i| \leq D$, in other words, an upper bound on the number of moves the second player must choose from.

Proof. The main part of this proof is finding a function $\bar{R}_{\text{reg}}(T)$ that bounds the pseudo-regret of the regression, as seen in this equation:

$$\mathbb{E}\left[\sum_{t=1}^{T}\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \sum_{t=1}^{T}\left(\langle p_t^{i_t}, \mu^{i_t}\rangle - X_t(i_t, h_t)\right)^2\right] \leq \bar{R}_{\text{reg}}(T)$$

Remember, there is one regression per possible move of the first player, which estimates the outcome of playing that move based on the known strategy of the second player.

We begin by separating the regret of the K different regressions:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=1}^{T}\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \sum_{t=1}^{T}\left(\langle p_t^{i_t}, \mu^{i_t}\rangle - X_t(i_t, h_t)\right)^2\right] \\
&= \mathbb{E}\left[\sum_{i\in[K]}\left(\sum_{t:i_t=i}\left(\langle p_t^i, \hat{\mu}_t^i\rangle - X_t(i, h_t)\right)^2 - \sum_{t:i_t=i}\left(\langle p_t^i, \mu^i\rangle - X_t(i, h_t)\right)^2\right)\right]
\end{aligned}$$

When we now look at the regret per arm i we can apply Theorem 2 directly, by substituting $\theta_t = \hat{\mu}_t^i$, $u = \mu^i$ and $x_t = p_t^i$ from the theorem:

$$\sum_{t:i_t=i}\left(\langle p_t^i, \hat{\mu}_t^i\rangle - X_t(i, h_t)\right)^2 - \sum_{t:i_t=i}\left(\langle p_t^i, \mu^i\rangle - X_t(i, h_t)\right)^2$$

$$\leq \left\|\mu^i\right\|^2 + \left(d_i \ln\left(1 + \frac{T}{d_i} \cdot \max_{t:i_t=i} \left\|p_t^i\right\|^2\right)\right) \max_{t:i_t=i} \left(\langle p_t^i, \hat{\mu}_t^i\rangle - X_t(i,h_t)\right)^2$$

With $\|\cdot\|$ denoting the Euclidean norm.

Now μ_i and the loss are bounded, and because p_t is a probability vector, it is also bounded $\|p_t\| \leq 1$. Then we get:

$$\begin{aligned}
&\left\|\mu^i\right\|^2 + \left(d_i \ln\left(1 + \frac{T}{d_i} \cdot \max_{t:i_t=i} \left\|p_t^i\right\|^2\right)\right) \max_{t:i_t=i} \left(\langle p_t^i, \hat{\mu}_t^i\rangle - X_t(i,h_t)\right)^2 \\
&\qquad \leq \mathcal{O}\left(d_i \ln\left(1 + \frac{T}{d}\right)\right)
\end{aligned}$$

The variable d_i in all these equations is the number of regressors, i.e. the number of moves the second player can make: $d_i = |\mathcal{H}|^i$. Since we have a finite amount of $i \in [K]$, there is a D such that $D \geq d_i$ for each arm i. We can use this together with the fact that $d_i \ln(1 + T/d_i)$ is increasing in d_i:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i\in[K]}\left(\sum_{t:i_t=i}\left(\langle p_t^i, \hat{\mu}_t^i\rangle - X_t\left(i,h_t\right)\right)^2 - \sum_{t:i_t=i}\left(\langle p_t^i, \mu^i\rangle - X_t(i,h_t)\right)^2\right)\right] \\
&\leq \mathbb{E}\left[\sum_{i\in[K]} \mathcal{O}\left(D\ln\left(1+\frac{T}{D}\right)\right)\right] \\
&= \mathbb{E}\left[\mathcal{O}\left(KD\ln\left(1+\frac{T}{D}\right)\right)\right] \\
&= \mathcal{O}\left(KD\ln\left(1+\frac{T}{D}\right)\right)
\end{aligned}$$

Now we have proven that $\bar{R}_{\text{reg}}$ is a bound for the regret of the regression when using ridge regression. Plugging this into $\frac{5}{4}\sqrt{2KT \cdot \bar{R}_{\text{regression}}(T)}$ immediately gives the bound $\mathcal{O}\left(\frac{5}{4}K \cdot \sqrt{2TD \cdot \ln\left(1+\frac{T}{D}\right)}\right)$.

□

We now know how well the new CBT algorithm performs on average in theory. From these results, it already becomes clear that the performance has a linear dependence on K, suggesting that performance deteriorates quickly for games with many possible moves. On the other hand, it has only square root dependence on D, meaning bigger action spaces for the second player matter less.

4.2 Analysis of the second player

The analysis would not be complete without also looking at the regret of the second player in our setting. For this, we first need to choose an algorithm for the second player. Note that the choice of algorithm does not have any impact on our regret guarantees for the first player since we made no assumptions in Theorem 3 on how the second player chooses their strategy. We will go for the simple and widely used UCB1 by Auer et al. (2002), also explained in Section 2.3.2. Using UCB1 we can get the regret bound formulated in Theorem 5:

Theorem 5 (Regret p-player)**.** *Let $D \geq |\mathcal{H}^i|$ be an upper bound for the number of actions available to the second player in each subtree. Then the regret for player 2, after playing n rounds, is bounded by:*

$$\bar{R}_{player\ 2}(T) \leq 4K\sqrt{2DT \ln T} + DK\left(1 + \frac{\pi^3}{3}\right)$$

To prove the upper bound for the second player, we will use an intermediate proof from the paper that first introduced the UCB1 algorithm (Auer et al., 2002), which we restate here in Theorem 6.

Theorem 6 (Auer et al., 2002)**.** *For all $K > 1$, if policy UCB1 is run sequentially on K machines having arbitrary loss (Auer et al. (2002) uses gains instead of losses, but the regret is the same either way) distributions with support in $[0, 1]$ and expected outcomes of $\mu_1, ..., \mu_K$, and with $T_j(n)$ being the number of times arm j is played. Then, in any sequence of plays, after n plays, the expectation is bounded for each suboptimal arm:*

$$\mathbb{E}\left[T_j(n)\right] \leq \frac{8 \ln n}{\Delta_i^2} + 1 + \frac{\pi^2}{3}$$

with $\Delta_i \overset{def}{=} \mu^* - \mu_i$ and μ^* equal to the expected/mean loss for the optimal arm.

Remark. *Note that, since the optimal mean μ^* is higher than any μ_i and all $\mu^*, \mu_i \in [0, 1]$, we have $\Delta_i \in [0, 1]$.*

The proof for the regret of the second player works similarly to the original UCB1 proof, by bounding the number of times each suboptimal arm is chosen. Before we can use we must therefore restate the regret as a function of $\mathbb{E}[T_j(n)]$. This is done by Lemma 4, whose proof can be found in the Appendix:

Lemma 4. *The pseudo-regret for the second player is equal to:*

$$\bar{R}_{player\ 2}(T) = \mathbb{E}\left[\sum_{i \in [K]} \sum_{j \in \mathcal{H}^i} \Delta_{i,j} \mathbb{E}[T_j(n_i)|n_i]\right]$$

Where n_i is the number of times that player 1 chooses move i, in other words $n_i = |\{t : i_t = i\}|$ with $\sum_{i\in[K]} n_i = T$, where $T_j(n)$ is equal to the number of times player 2 chooses arm j, given that arm j was available n times and $\Delta_{i,j} = \mu(i, j^{i,*}) - \mu(i, j)$ ($j^{i,*}$ here being the optimal move for the second player after the first player has played i).

We now have all the parts to prove the regret bound for player 2.

Proof of Theorem 5. Because of Lemma 4, the only thing left to proof is:

$$\mathbb{E}\left[\sum_{i\in[K]}\sum_{j\in\mathcal{H}^i}\Delta_j\mathbb{E}[T_j(n_i)|n_i]\right] \leq 4K\sqrt{2DT\ln T} + DK\left(1+\frac{\pi^3}{3}\right) \tag{18}$$

We do this by looking at the inside of the first expectation and assuming nothing about the values of all the n_i, i.e. we don't assume anything about the choices that the first player makes. We do know, however, that the total sum is equal to the total number of games $\sum_{i\in[K]} n_i = T$.

For a certain $i \in [K]$ there is only a finite number of possible values for $n_i \leq T$ so there is some value $\hat{n}_i$ that maximizes $\mathbb{E}[T_j(\hat{n}_i)]$ for some $j \in \mathcal{H}^i$, we can use this to bound the inside of the expectation in (18):

$$\sum_{j\in\mathcal{H}^i}\Delta_{i,j}\mathbb{E}[T_j(n_i)|n_i] \leq \sum_{j\in\mathcal{H}^i}\Delta_{i,j}\max_{\hat{n}_i:\hat{n}_i\leq T}\mathbb{E}[T_j(\hat{n}_i)]$$

This can be further bounded by choosing a certain Δ (which we will later specify) and distinguishing between the cases where $\Delta_{i,j} < \Delta$ and the cases where $\Delta_{i,j} \geq \Delta$.

$$\begin{aligned}
\sum_{j\in\mathcal{H}^i}\Delta_{i,j}\mathbb{E}[T_j(\hat{n}_i)] &= \sum_{j:\Delta_{i,j}<\Delta}\Delta_{i,j}\mathbb{E}[T_j(\hat{n}_i)] + \sum_{j:\Delta_{i,j}\geq\Delta}\Delta_{i,j}\mathbb{E}[T_j(\hat{n}_i)] \\
&\leq \sum_{j:\Delta_{i,j}<\Delta}\Delta\mathbb{E}[T_j(\hat{n}_i)] + \sum_{j:\Delta_{i,j}\geq\Delta}\Delta_{i,j}\left(\frac{8\ln\hat{n}_i}{\Delta_{i,j}^2}+1+\frac{\pi^2}{3}\right) \\
&\leq \hat{n}_i\Delta + \sum_{j:\Delta_{i,j}\geq\Delta}\frac{8\ln\hat{n}_i}{\Delta_{i,j}} + \sum_{j:\Delta_{i,j}\geq\Delta}\Delta_{i,j}\left(1+\frac{\pi^2}{3}\right) \\
&\leq \hat{n}_i\Delta + \sum_{j:\Delta_{i,j}\geq\Delta}\frac{8\ln\hat{n}_i}{\Delta} + \sum_{j:\Delta_{i,j}\geq\Delta}\left(1+\frac{\pi^2}{3}\right) \\
&\leq \hat{n}_i\Delta + \frac{8|\mathcal{H}^i|\ln\hat{n}_i}{\Delta} + |\mathcal{H}^i|\left(1+\frac{\pi^2}{3}\right) \\
&\leq \hat{n}_i\Delta + \frac{8D\ln\hat{n}_i}{\Delta} + D\left(1+\frac{\pi^2}{3}\right)
\end{aligned}$$

Where the first inequality uses Theorem 6, the second uses the fact that $\mathbb{E}[T_j(\hat{n}_i)] \leq \hat{n}_i$ and the third inequality uses the remark made earlier that $\Delta_{i,j} \in [0, 1]$.

Finally, by choosing a Δ strategically to be equal to $\Delta = \sqrt{8D\ln(\hat{n}_i)/\hat{n}_i}$ and using the fact that $\hat{n}_t \leq T$, we get the result:

$$\begin{aligned}
&\hat{n}_i\Delta + \frac{8D\ln\hat{n}_i}{\Delta} + D\left(1+\frac{\pi^2}{3}\right) \\
&\quad = \hat{n}_i\sqrt{8D\ln(\hat{n}_i)/\hat{n}_i} + \frac{8D\ln\hat{n}_i}{\sqrt{8D\ln(\hat{n}_i)/\hat{n}_i}} + D\left(1+\frac{\pi^2}{3}\right) \\
&\quad = 2\sqrt{8D\hat{n}_i\ln(\hat{n}_i)} + D\left(1+\frac{\pi^2}{3}\right) \\
&\quad \leq 4\sqrt{2DT\ln(T)} + D\left(1+\frac{\pi^2}{3}\right)
\end{aligned}$$

In the last step, we make use of the fact that $x\ln x$ is strictly increasing for $x \geq 1$. Finally, we can now put this back in the original equation for the regret:

$$\begin{aligned}
\bar{R}_{\text{player 2}}(T) &= \mathbb{E}\left[\sum_{i\in[K]}\sum_{j\in\mathcal{H}^i}\Delta_{i,j}\mathbb{E}[T_j(n_i)|n_i]\right] \\
&\leq \mathbb{E}\left[\sum_{i\in[K]}\left(4\sqrt{2DT\ln T} + D\left(1+\frac{\pi^2}{3}\right)\right)\right] \\
&= \mathbb{E}\left[4K\sqrt{2DT\ln T} + DK\left(1+\frac{\pi^2}{3}\right)\right] \\
&= 4K\sqrt{2DT\ln T} + DK\left(1+\frac{\pi^2}{3}\right)
\end{aligned}$$

□

5 Experiments

To complement the analysis of our CBT algorithm, we evaluate the new algorithm empirically. In this section, we will describe the experiments used to test the performance of the proposed CBT algorithm. First, we will give a general introduction on how performance is evaluated. After that, we will explain the different experiments. Then we will explain what the 'control algorithm' CBT is tested against. Finally, we will look at hyperparameters throughout all the experiments. We also

introduce a possible generalisation of CBT to more than two levels, which is also tested empirically. The theoretical guarantees of this generalised CBT, however, remain an open problem. The implementation of all the experiments can be found in a public repository.[5]

In the experiments, we measure performance in three possible ways. First, by looking at the estimation of the minimax value by the algorithm. Second, by looking at the estimated chance that the algorithm plays the optimal move (the one with the highest minimax value). Finally, by looking at the move that the algorithm recommends as the best move.

Starting with the estimation of the minimax value. The reason we are letting bandit algorithms play against each other is to find the minimax value of the best strategy. Because, as explained in Section 3.2, the value of Equation 19 (below) goes to zero for higher T. We are interested in how quickly this happens. Therefore, the first measure of success is how small this difference becomes for different values of T.

$$\min_{w \in \mathcal{W}} \max_{p \in \mathcal{P}} \mu(w, p) - \frac{1}{T} \sum_{t=1}^{T} \mu(w_t, p_t) \tag{19}$$

To simplify and speed up the computation, we will use the value of $\frac{1}{T} \sum_{t=1}^{T} X_t(i_t, h_t)$ instead of $\frac{1}{T} \sum_{t=1}^{T} \mu(w_t, p_t)$. This is fine, since the former converges to the latter with increasing T.

The second measure of success is how often the algorithm plays the optimal move. The convergence to the minimax value also depends on how well the second player's bandit performs. Since the point of a tree search algorithm is finding the best move, we are also interested in how quickly it finds i^*, the optimal move. We measure this by looking at the estimated chance of the algorithm choosing the best move, in mathematical terms:

$$\widehat{\mathbb{P}}\left(i_t = i^*\right) = \frac{1}{T} \sum_{t=1}^{T} \mathbf{1}\left(i_t = i^*\right) \tag{20}$$

The final performance measure is to look at what final choice is recommended by the algorithm after T rounds. To do this, we first need a recommendation rule. We look at two possibilities, the first being: Recommend the arm with the highest visit count, i.e. recommend the arm a with:

$$a = \arg\max_{i \in [K]} \sum_{t=1}^{T} \mathbf{1}(i_t = i) \tag{21}$$

[5]Which can be found at https://github.com/daandj/Thesis.

Note the similarity with the second measure of success and the second recommendation rule: Recommend the arm with the lowest loss/highest gain. So in other words, recommend choosing arm a with:

$$a = \arg\min_{i \in [K]} \frac{1}{|\{t : i_t = t\}|} \sum_{t:i_t=i} X_t(i_t, h_t) \tag{22}$$

These are all the performance measures we will use to evaluate CBT performance. In the next section, we will explain all the experiments in which we will test them. All these experiments are run 10 times, and the average outcome is reported to make the results less random and more representative. Furthermore, each experiment is conducted once for each of the following values for T: $T = 1$, $T = 10$, $T = 100$, $T = 10000$ and $T = 100000$.

5.1 Methods

First, we need a game that we can test CBT on. In this study, we use two games. The first is the most minimal example of a game that fits the problem setting in Section 3.1, called the 'minimal game'. Secondly, to test the performance of CBT in a real-life game, we use Tic-tac-toe. Both are explained in more detail below.

5.1.1 Minimal game

Beginning with the minimal game. In this game first the first player picks a number i_t, then the second player picks a number h_t and finally the outcome is sampled from a Bernoulli distribution, where the hyperparameter of the distribution $\mu(i_t, h_t)$ (which is equal to the mean for a Bernoulli) depends on which number both players picked. These hyperparameters are set before the start and are immutable during all repetitions. For simplicity, both players pick a number from the set $[K]$.

With this game, we run three experiments, each with a different value for $[K]$, which is the size of the action space for both players. This way we find if the size of the action space influences the performance of CBT.[6]

In the experiments, the first move i_t is chosen by CBT, exactly as outlined in Algorithm 3 and the second move h_t is chosen by UCB1 as specified in Algorithm 1.

5.1.2 Tic-tac-toe

After looking at the minimal game, we will test the performance of the algorithm in a real-life game: Tic-tac-toe, also known as noughts and crosses. The main

[6]The matrices $(\mu(i,t))_{i,j \in [K]}$ with means for all three experiments can be found in https://github.com/daandj/Thesis.

difference between this game and the minimal game from the previous section is that after two moves, the game is not finished yet. To deal with this, the Bernoulli distribution used to sample the outcome is replaced by a random rollout of the rest of the game. Which means that after both players have picked a move, the next moves for both are uniformly sampled from all available moves until the game is finished. The outcome of the game is then 1 if the first player won, 0 if the second player won, and 0.5 in the case of a draw. Apart from this difference, the algorithm is exactly as with the minimal game in Section 5.1.1.

Exactly as with the minimal game experiment, we run this experiment with boards of different sizes (where a player still needs to score all in a row to win), to see how the performance changes with different action set sizes.

5.1.3 Hyperparameters CBT

Another thing we are interested in is the effects of different values for the hyperparameters: the learning rate and the exploration rate. In all the previously mentioned experiments, we use the values for these hyperparameters that we empirically found to be optimal. But we also want to analyse the effect of different hyperparameters. For this, we perform two experiments, both use the minimal game from Section 5.1.1, with $K = 10$. One of the experiments will be running the algorithm with different values for the learning rate, the other will be the same, but with different values for the exploration rate. All runs are again repeated 10 times to get a more representative result. In the learning rate experiment, the exploration rate will be the same as in the minimal game experiment and vice versa, that is, the empirically found optimal value.

We won't attempt to find the best hyperparameter setting since this will depend on the game, the 'best hyperparameters' that we find here are therefore not generalizable for all games. We will look at the performance of the algorithms under different hyperparameters to gain a better understanding of the way CBT works and to compare the recommendations of the theory with empirical practice.

The values for the hyperparameters with the Tic-tac-toe experiment will also be those empirically found to be optimal.

5.1.4 Algorithms for comparison

In the experiments from the previous subsection, which test the effect of different values for the hyperparameters, only the CBT algorithm is run. The runs with different hyperparameter values are compared with each other. The remaining experiments, however, need a control group to compare against. Regarding the minimal game and the two-level Tic-tac-toe game, the control case is UCB1. The only thing that changes is the bandit that picks the first move. Remember that

the second move was already chosen by UCB1 in the CBT algorithm; this makes for a fairer comparison.

Notice that using UCB1 to pick the first and second move is the same as UCT from Kocsis and Szepesvári (2006) with a maximum search depth of 2. For the experiments with the CBT algorithm that is generalised for dynamic depth search, we therefore use UCT with dynamic depth as a control case as well.

6 Results

Now that we have explained the methodology and setup of the experiments, we will present the results of the experiments. In this section, we will first look at the result of the 'minimal game' experiment, then the experiments that look at different hyperparameter settings for CBT, and finally, we look at the results of the Tic-tac-toe experiment. After presenting them, we will interpret the results and connect them with the theory from the previous sections. In the figure below, with graphs for CBT and UCB1, the crosses show the means of the 10 repetitions of the experiment, and the bars show the 25% and 75% quantiles.

6.1 Minimal game

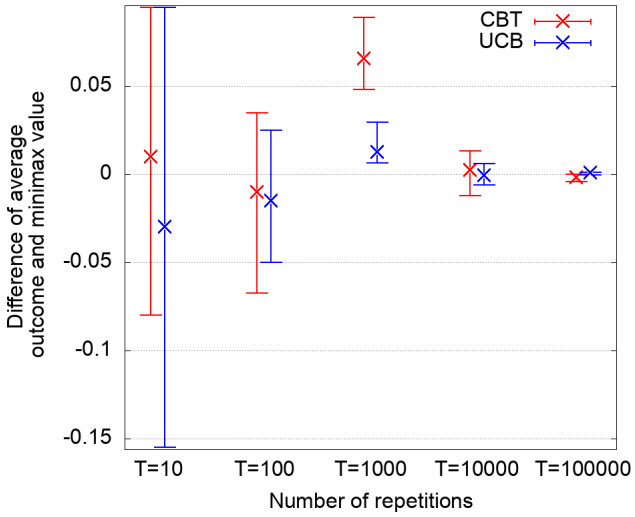

Figure 4: Difference between average outcome and the minimax value in minimal game of CBT and UCB1, with $K = 10$.

The first experiment was a minimal game with $K = 10$. Our first measure of performance was the difference between the average outcome and the minimax value (as seen in Equation 19). This difference is shown in Figure 4 for the first experiment. Here, we see that CBT is not a clear improvement over UCB1. The left-most results don't say anything, because in these runs, each of the 10 arms has been tried an average of 1 time. For higher values of T, however, we can observe three things:

1. Both algorithms get closer to 0, as was predicted in Section 3 (though not monotonically in the case of CBT);

2. The average outcomes of all UCB1 runs are always closer, or as close to 0 as CBT is, meaning that UCB1 performs better at finding the minimax value;

3. The average outcome of CBT consistently has a bigger deviation than UCB1.

The first two observations mean that CBT is a less suitable algorithm than UCB1. The last observation could be explained by the hypothesis that CBT keeps exploring more. Indeed, this is something that we can corroborate by looking at the second measure of performance: how often the algorithms play the optimal move, for this experiment shown in Figure 5. The results in this figure show the opposite of this hypothesis: CBT plays the best move *more* often than UCB1. Which means that CBT performs better than UCB1 for this metric. The difference in divergence in Figure 4 could then be explained by the hypothesis that when CBT explores, it does so more often with the worse moves than when UCB1 explores. But unfortunately, we do not have the data to confirm or deny this.

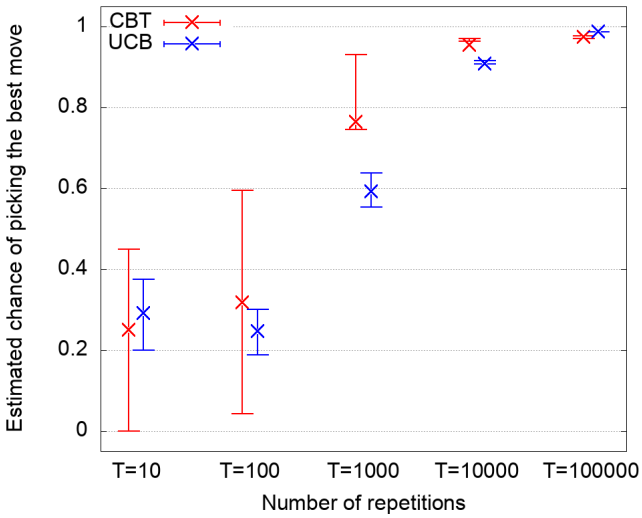

Figure 5: The percentage of times that CBT and UCB1 try the optimal arm in a minimal game with $K = 10$.

The final metric we will look at for this experiment is the moves the algorithms recommend after running for T times. Table 1 shows the percentages of times it recommended the best one. In Figure 1, 'CBT (1)' and 'UCB1 (1)' refer to choosing a move a using equation 21 i.e. the most tried arm, and 'CBT (2)' and 'UCB1 (2)' refer to choosing a move a using Equation 22 i.e. the move with the lowest average loss. In the results from this table, we see two things. First, UCB1 is slightly better than CBT at picking the right move in runs with lower T. Second, picking the best move using the 'lowest average loss' as shown in column 'CBT (2)'

	Times recommended move is optimal move			
T	CBT (1)	CBT (2)	UCB1 (1)	UCB1 (2)
10	0%	10%	30%	50%
100	50%	50%	60%	70%
1 000	100%	10%	100%	100%
10 000	100%	90%	100%	100%
100 000	100%	100%	100%	100%

Table 1: The percentage of times that CBT and UCB1 recommended the best arm with $K = 10$.

does worse than picking the most often tried arm 'CBT (1)', for CBT. Meanwhile, for UCB1, this is the opposite; the lowest loss method in 'UCB1 (2)' performs better than the most tried arm 'UCB1 (1)'.

The reason for this is a fundamental difference in how both UCB1 and CBT work. CBT works by predicting the loss by looking at the strategy of the next bandit. When the next bandit changes their strategy for a certain arm, CBT may already know, without trying that arm, that the loss from that arm is probably higher, and CBT will therefore stop trying that arm. In this case, the expected loss would be higher than the observed average loss. Then, making the final recommendation for the best move based on the lowest observed loss would be disregarding much of the information CBT has on the best move. On the other hand, when choosing an arm, UCB1 looks at the average outcome (which is equivalent to the lowest loss) and adds some optimism (i.e. the confidence bound), and from this it chooses the best value. The arms with low average losses will therefore be the most often chosen. This means that for UCB1, choosing the arms with the lowest observed loss performs well, while for CBT, it performs significantly worse.

6.1.1 Higher K

Next, we look at larger values for K, still with the minimal game, of which the results are shown in Figure 6. Just as with $K = 10$, they both show that the outcome of UCB1 is almost always closer or just as close to the minimax value than the outcome of CBT (except for $K = 30, T = 10\,000$ where CBT is slightly better). Where they both differ from the $K = 10$ case, however, is that the quantiles for CBT are closer together, meaning the results deviate less between reruns of the experiment, with the deviation being smaller for $K = 30$ than for $K = 20$.

When we look at the number of times both algorithms try the optimal move

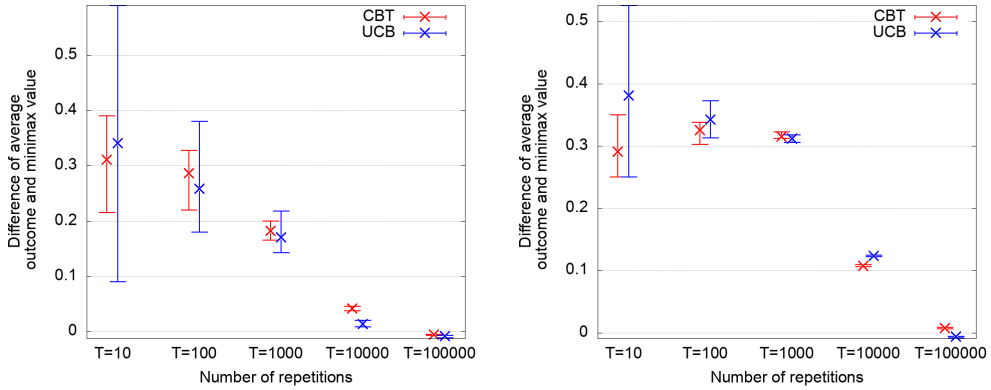

Figure 6: Difference between average outcome and the minimax value in minimal game of CBT and UCB1, with $K = 20$ (left) and $K = 30$ (right).

(in Figure 7), we see that the results with $K = 20$ and $K = 30$ are similar to $K = 10$. First of all, for higher T, CBT consistently tries the best move more often than UCB1 with both $K = 20$ and $K = 30$.

With $K = 20$, just as it was with $K = 10$, there is no discernible difference between the CBT and UCB1 for lower T. But with $K = 30$, UCB1 is better than CBT when $T \leq 1000$.

Finally, we look at the recommended moves for the minimal game with $K = 20$ and $K = 30$. First, for UCB1, there is no difference in performance between choosing based on the most tried arm or the arm with the lowest observed loss. For CBT, there is a difference between the two methods; picking the most tried arm performs better.

When comparing UCB1 and CBT, we see that UCB1 is (with any method) better than CBT at recommending the optimal move. This is again similar to $K = 10$, where UCB1 was also slightly better.

In summary, the main takeaways for the minimal game experiments are that UCB1 consistently performs better than CBT in recommending the right move and converging to the minimax value of the game; the gap is relatively small. Yet, what CBT does better is trying the optimal move more. This indicates that the algorithms make different trade-offs between exploration and exploitation. Based on these results, UCB1 would probably perform better in real-world applications.

6.2 Effect of hyperparameters

After we have seen the results of the minimal game, we look at the effects of different values for the hyperparameters. The hyperparameters used in the previous

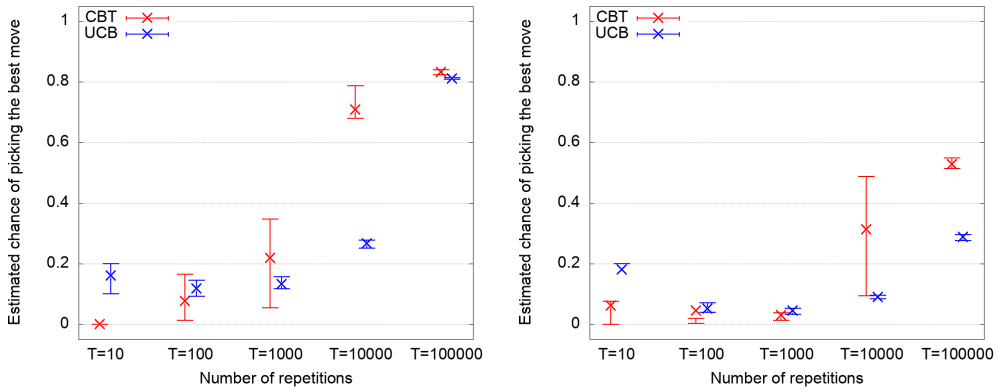

Figure 7: The percentage of times that CBT and UCB1 try the optimal arm in a minimal game with $K = 20$ (left), and $K = 30$ (right).

	Times recommended move is optimal move			
T	CBT (1)	CBT (2)	UCB1 (1)	UCB1 (2)
10	10%	10%	0%	10%
100	20%	10%	40%	40%
1 000	80%	10%	100%	100%
10 000	100%	0%	100%	100%
100 000	100%	100%	100%	100%

Table 2: The percentage of times that CBT and UCB1 recommended the best arm with $K = 20$.

	Times recommended move is optimal move			
T	CBT (1)	CBT (2)	UCB1 (1)	UCB1 (2)
10	0%	0%	0%	0%
100	0%	10%	20%	20%
1 000	0%	10%	40%	30%
10 000	80%	0%	100%	100%
100 000	100%	0%	100%	100%

Table 3: The percentage of times that CBT and UCB1 recommended the best arm with $K = 30$.

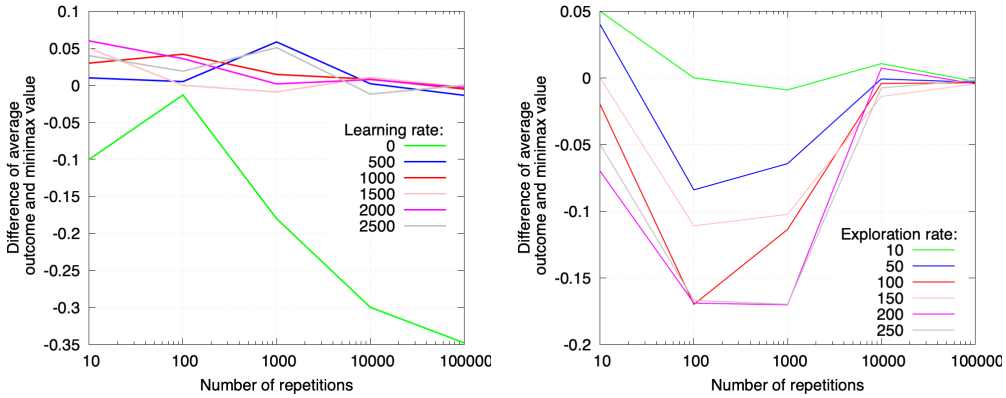

Figure 8: Difference between average outcome and the minimax value in minimal game with $K = 10$ and different learning hyperparameters (left), and exploration hyperparameters (right)

minimal game experiment with $K = 10$ were a learning rate of $\gamma = 1500$ and exploration rate $\nu = 10$, so these are the default settings for this experiment too. Starting with the effect of the learning rate, for which the difference between the minimax value and the estimated minimax value is seen (i.e. Equation 19) in Figure 8 (in the graph on the left side).

The first thing that stands out is the green line corresponding with a learning rate of $\gamma = 0$, for which the results get worse with higher T. This is because of the way CBT chooses a move. Remember the move is sampled for a distribution where move i has a chance of $w_t^i = \frac{1}{\nu+\gamma(\pi_{t,j}-\pi_{t,i})}$. So when $\gamma = 0$, the distribution becomes uniform, $w_t^i = \frac{1}{\nu}$ for all i and t, meaning that the first player doesn't improve over time. However, the second player does improve over time, so the outcomes converge to some value other than the minimax value. Another observation is that the best performing learning rate (the learning rate for which the values are closest to zero) is $\gamma = 1500$, which is why it was used in to other experiments too. Apart from this, there is no noticeable difference between the different values.

The second experiment looks at the exploration rate; the results are shown in Figure 8 (in the graph on the right side). The best performing setting is $\nu = 10(= K)$, shown with a green line here. This is also the setting that is recommended by the theory in Section 4. Lower exploration rates are not possible because the sum of all probabilities $\sum_{i\in[K]} w_t^i$ would then exceed 1, meaning w_t would no longer be a valid probability vector. For higher ν, we observe that the algorithm performs worse for lower T, but recovers with higher T. This can be explained by the fact that with a higher exploration rate, the algorithm keeps exploring more before it commits to playing the best move it has found. But at some point it does still

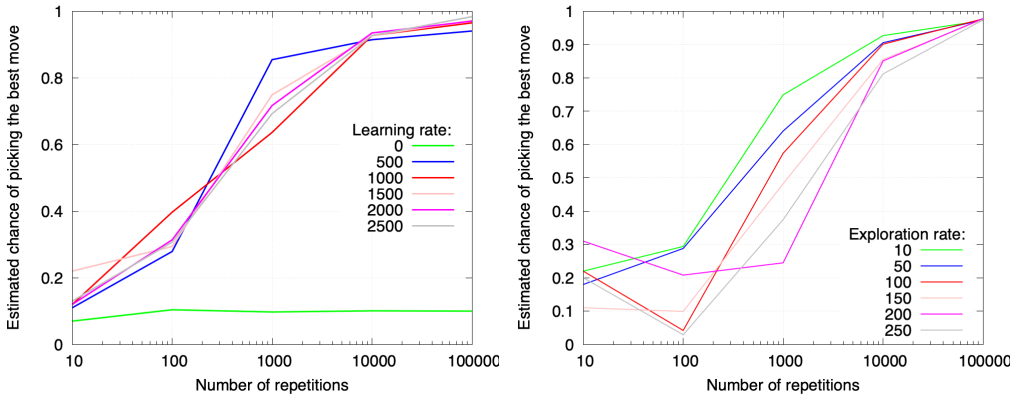

Figure 9: The percentage of times that CBT tries the optimal arm in a minimal game, different learning hyperparameters (left), and exploration hyperparameters (right)

commit, which is what is happening at $T \geq 10\,000$.

When we look at the second measure of performance in Figure 9, how often the algorithm chooses the optimal move, we see similar results. The learning rate graph (on the left of the figure) confirms that for $\gamma = 0$ the algorithm keeps playing the best move about 10% of the time. This is in line with the explanation that keeps exploring uniformly. For the other values, we see no significant differences between them. The right graph showing the performance for different values of the exploration rate ν again shows that the value recommended by the analysis of $\nu = 10$ is the best one. Although the difference with the other values of ν is smaller here.

Lastly, we look at the third measure of performance for the hyperparameter experiments, which is again the final recommendation by the algorithm. In Tables 6 and 7 we see the percentage of times CBT recommends the correct move with the 'most tried arm' and 'lowest average loss' respectively. The one based on the 'most tried arm' method shows what we would expect, the $\gamma = 0$ experiment is bad at recommending the right move – this was expected because it plays every arm approximately the same number of times, therefore the recommendation is a random move – and the other values perform approximately the same (with $\gamma = 500$ being slightly better than the rest). But when we look at the recommendations based on the 'lowest loss', the results are inverted. Here $\gamma = 0$ outperforms the rest. This can also be explained by the fact that with the learning rate equal to 0, CBT only explores. The move with the lowest average loss is then automatically the move with the best average outcome, akin to a Monte Carlo tree search that doesn't use bandits but only random rollouts. We see here that the higher value of γ is, the worse this method works. In other words, exploring is rewarded and exploiting is

	Times recommended move is optimal move					
T	$\gamma = 0$	$\gamma = 500$	$\gamma = 1000$	$\gamma = 1500$	$\gamma = 2000$	$\gamma = 2500$
10	0%	10%	0%	20%	10%	50%
100	10%	60%	40%	50%	60%	30%
1 000	10%	100%	80%	80%	80%	90%
10 000	0%	100%	100%	100%	100%	90%
100 000	20%	100%	100%	100%	100%	100%

Table 4: The percentage of times that CBT recommended the best move using the 'most tried arm' method, for different learning rates and $K = 10$.

	Times recommended move is optimal move					
T	$\gamma = 0$	$\gamma = 500$	$\gamma = 1000$	$\gamma = 1500$	$\gamma = 2000$	$\gamma = 2500$
10	0%	10%	0%	30%	30%	50%
100	50%	80%	40%	50%	20%	50%
1 000	100%	70%	60%	40%	30%	10%
10 000	100%	100%	90%	70%	80%	90%
100 000	100%	100%	100%	100%	100%	100%

Table 5: The percentage of times that CBT recommended the best move using the 'minimum average loss' method, for different learning rates and $K = 10$.

punished with the 'lowest average loss' method.

We also look at the performance of recommending the right move for the experiment with different exploration rates in Tables 6 and 7. These results show nothing new, the recommended $\nu = 10$ works best with the 'most tried arm' method, the other exploration hyperparameters have roughly the same performance, and the 'lowest average loss' method performs worse than the 'most tried arm' method.

The main takeaways from these hyperparameter experiments are that setting $\gamma = 0$ means always playing a uniform random strategy, which performs worse than $\gamma \neq 0$. The effects of different settings for γ had no noticeable effects. Furthermore, we have seen that for the exploration hyperparameter the value recommended by the theory $\nu = K$ works best in practice as well (in our setting). Finally, we must note that our results say nothing about what the correct settings of the hyperparameters would be for any other game. However, they do tell us something about the inner workings of the algorithm and the way this relates to the theory.

	Times recommended move is optimal move					
T	$\nu = 10$	$\nu = 50$	$\nu = 100$	$\nu = 150$	$\nu = 200$	$\nu = 250$
10	20%	40%	10%	10%	10%	0%
100	50%	30%	20%	10%	20%	20%
1 000	80%	70%	50%	50%	50%	50%
10 000	100%	100%	100%	100%	100%	100%
100 000	100%	100%	100%	100%	100%	100%

Table 6: The percentage of times that CBT recommended the best move using the 'most tried arm' method, for different values of the exploration hyperparameter and $K = 10$.

	Times recommended move is optimal move					
T	$\nu = 10$	$\nu = 50$	$\nu = 100$	$\nu = 150$	$\nu = 200$	$\nu = 250$
10	30%	50%	30%	40%	50%	20%
100	50%	20%	30%	40%	60%	60%
1 000	40%	40%	40%	0%	20%	30%
10 000	70%	80%	80%	40%	70%	50%
100 000	100%	100%	100%	100%	100%	100%

Table 7: The percentage of times that CBT recommended the best move using the 'minimum average loss' method, for different values of the exploration hyperparameter and $K = 10$.

6.3 Tic-tac-toe

Our final experiment tests the performance of CBT in the game of Tic-tac-toe. Since we have let the algorithm play this game with board sizes varying up to 10-by-10, it is not possible to calculate the minimax values of best moves for our version of the game, because there are too many possible plays. As a result, Figures 10 and 11 don't show the difference between the minimax value and the average outcome (like in Equation 19), instead they show the average outcomes:

$$\frac{1}{T}\sum_{t=1}^{T} X_t\left(i_t, h_t\right) \tag{23}$$

Similarly, we do not show the number of times the optimal move is recommended or tried, because we do not know the optimal move.

The first experiment with 3 by 3 Tic-tac-toe in Figure 10 shows the average outcomes moving towards some point above 0.64. We can see that UCB1 has

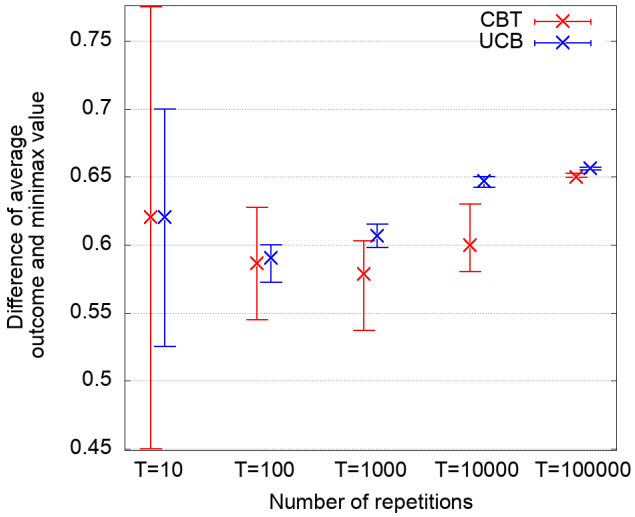

Figure 10: Difference between average outcome and the minimax value in Tic-tac-toe with a 3-by-3 board

higher outcomes, which means it performs better than CBT, which corresponds with the results of the minimal game experiment. In the experiments with bigger boards, 5-by-5 and 10-by-10, on the other hand, both algorithms perform equally well (or possibly, equally badly, since we don't know the minimax value).

The main takeaway for the Tic-tac-toe experiment is that UCB1 either performs better or equal to CBT. These findings confirm what we found in the minimal game experiments before. Further analysing these Tic-tac-toe experiments is difficult because we don't know the minimax values or moves.

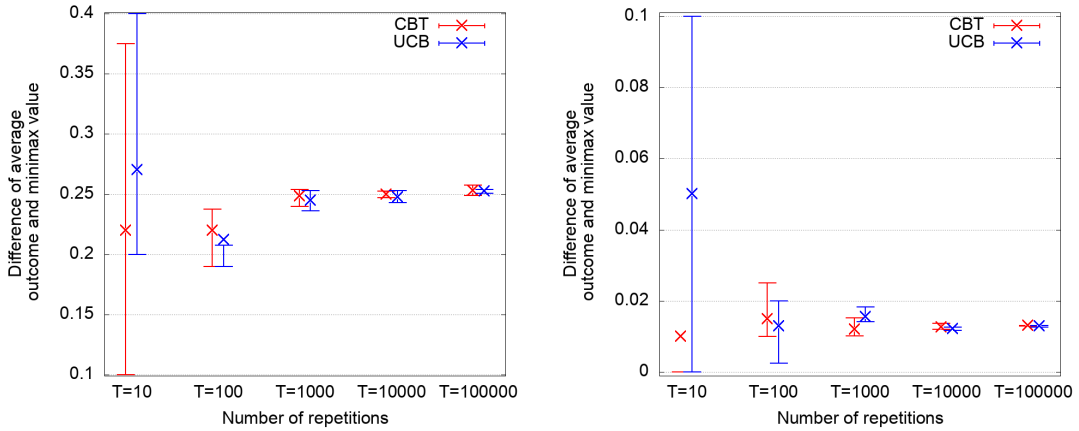

Figure 11: Difference between average outcome and the minimax value in Tic-tac-toe with 5-by-5 (left), and 10-by-10 (right) boards

7 Conclusion

In this research, we have tried to find an answer to the research question:

> *"Can we improve upon Monte Carlo tree search with UCB1 and make a more efficient tree search algorithm by applying contextual bandits in a two-player zero-sum game?"*

We have introduced a new bandit algorithm, CBT, utilising contextual bandits for Monte Carlo tree search of two levels deep. We have shown that the rate at which MCTS finds the minimax value depends on the regret upper bound of the bandit algorithm. We have seen in Section 4 that the regret bound of CBT is $\mathcal{O}\left(K\sqrt{2TD\ln(1+T/D)}\right)$. The proof for this requires no assumptions on a fixed stochastic reward from each arm. Furthermore, it has a worst-case sub-linear dependence on T, and as we have shown, this means that the average result of the strategy that CBT plays approaches the minimax value of the game. CBT does, however, have a linear dependence on the number of arms K, which suggests the performance deteriorates faster for games with a large number of moves that must be considered.

Moreover, we have tested the performance of CBT in a stylised minimal game and the real-world game Tic-tac-toe. In the minimal game experiments, CBT did not perform better than UCB1 in approaching the minimax value of the game. The results did suggest that CBT is slightly better at identifying the best move with lower T, but this did not translate into better recommendations for the best move. The results confirmed the fact that the algorithms perform worse for larger K, and CBT was more sensitive to higher K than UCB1, also confirming our

theoretical analysis. In the Tic-tac-toe experiment, these results were confirmed with a real-world game, UCB1 performing similarly or better than CBT.

We have also tested the effect of different hyperparameters on the performance of CBT. These experiments showed that CBT performs poorly with a learning rate of $\gamma = 0$, which follows from the theory and they showed that the performance remains consistent for learning rates other than 0. The experiment on exploration rates confirmed that the value recommended by the theoretical analysis, namely an exploration rate equal to the number of arms $\nu = K$, is optimal.

To answer the research question, based on the results presented in this thesis, CBT does not suffer from the theoretical drawbacks of UCB1, which means it is an improvement. However, from the experimental results we have seen that CBT is not better at choosing the best move in a game. Therefore, we cannot conclude that CBT is an improvement over UCB1. Still, this may not be the definitive answer. Our approach was to use a contextual bandit to take into account the predicted strategy of the next player for choosing a move. Although this does seem to improve the algorithm's ability to identify the best move, there was no improvement in the recommendations of CBT. We have seen that the performance of the recommendations by CBT is sensitive to the method used; therefore, it might be possible to improve upon UCB1 by using a different method to give a final recommendation. Finally, we only tested CBT in a two-level deep game; extending CBT to search deeper into the game tree might also improve the algorithm. Since by looking at more of the game tree, CBT can use the predicted strategies of the third move, the fourth move, et cetera, as more information to base its decision on.

7.1 Suggestions for future work

We leave these two improvements, extending CBT to more than 2 levels and finding more suitable methods to give the final recommendation, as possible directions for future research. Other possible directions include looking at different regression algorithms for the oracle in CBT and how it affects performance. Or looking at other (not oracle-based) contextual bandits, instead of the framework by Foster and Rakhlin (2020). Furthermore, in our analysis, we have looked at the regret minimisation. It might be interesting to analyse CBT from a best-arm identification perspective, since the overarching goal of the algorithm is to pick the best move in a game.

References

Agarwal, A., Hsu, D. J., Kale, S., Langford, J., Li, L., & Schapire, R. E. (2014). Taming the monster: A fast and simple algorithm for contextual bandits. *Proceedings of the 31th International Conference on Machine Learning, ICML 2014*, *32*, 1638–1646. http://proceedings.mlr.press/v32/agarwalb14.html

Arce, P., & Salinas, L. (2012). Online ridge regression method using sliding windows. *31st International Conference of the Chilean Computer Science Society, SCCC 2012*, 87–90. https://doi.org/10.1109/SCCC.2012.18

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2-3), 235–256. https://doi.org/10.1023/A:1013689704352

Bubeck, S., & Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, *5*(1), 1–122. https://doi.org/10.1561/2200000024

Campbell, M., Jr., A. J. H., & Hsu, F. (2002). Deep blue. *Artificial Intelligence*, *134*(1-2), 57–83. https://doi.org/10.1016/S0004-3702(01)00129-1

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge University Press. https://doi.org/10.1017/CBO9780511546921

Chaslot, G., Winands, M., Uiterwijk, J., Herik, H., & Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, *04*(03), 343–357. https://doi.org/10.1142/S1793005708001094

Coquelin, P., & Munos, R. (2007). Bandit algorithms for tree search. *UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver*, 67–74. https://doi.org/10.5555/3020488.3020497

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In H. J. van den Herik, P. Ciancarini, & H. H. L. M. Donkers (Eds.), *Computers and games, 5th international conference, CG 2006,* (pp. 72–83, Vol. 4630). Springer. https://doi.org/10.1007/978-3-540-75538-8_7

Ensmenger, N. (2012). Is chess the drosophila of artificial intelligence? A social history of an algorithm. *Social studies of science*, *42*(1), 5–30. https://doi.org/10.1177/0306312711424596

Foster, D. J., & Rakhlin, A. (2020). Beyond UCB: Optimal and efficient contextual bandits with regression oracles. *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, *119*, 3199–3210. https://proceedings.mlr.press/v119/foster20a.html

Garivier, A., & Kaufmann, E. (2016). Optimal best arm identification with fixed confidence. *Proceedings of the 29th Conference on Learning Theory, COLT 2016*, *49*, 998–1027. http://proceedings.mlr.press/v49/garivier16a.html

Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial intelligence*, *6*(4), 293–326. https://doi.org/10.1016/0004-3702(75)90019-3

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *Machine Learning: ECML 2006, 17th European Conference on Machine Learning*, *4212*, 282–293. https://doi.org/10.1007/11871842_29

Lattimore, T., & Szepesvári, C. (2020). *Bandit algorithms*. Cambridge University Press. https://doi.org/10.1017/9781108571401

Luo, H. (2017a). Lecture 7. *CSCI 699: Introduction to Online Learning*. Retrieved January 15, 2025, from https://haipeng-luo.net/courses/CSCI699/lecture7.pdf

Luo, H. (2017b). Lecture 14. *CSCI 699: Introduction to Online Learning*. Retrieved January 15, 2025, from https://haipeng-luo.net/courses/CSCI699/lecture14.pdf

Munos, R. (2014). From bandits to Monte-Carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, *7*(1), 1–129. https://doi.org/10.1561/2200000038

Schapire, R. (2018). Lecture 18. *COS 511: Theoretical Machine Learning*. Retrieved January 3, 2025, from https://www.cs.princeton.edu/courses/archive/spring18/cos511/scribe_notes/0411.pdf

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/NATURE16961

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144. https://doi.org/10.1126/science.aar6404

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359. https://doi.org/10.1038/NATURE24270

Świechowski, M., Godlewski, K., Sawicki, B., & Mańdziuk, J. (2023). Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, *56*(3), 2497–2562. https://doi.org/10.1007/s10462-022-10228-y

Toosi, A., Bottino, A. G., Saboury, B., Siegel, E., & Rahmim, A. (2021). A brief history of AI: How to prevent another winter (a critical review). *PET Clinics*, *16*(4), 449–469. https://doi.org/10.1016/j.cpet.2021.07.001

van den Herik, H. J., Uiterwijk, J. W. H. M., & van Rijswijck, J. (2002). Games solved: Now and in the future. *Artificial Intelligence*, *134*(1-2), 277–311. https://doi.org/10.1016/S0004-3702(01)00152-7

Vovk, V. (2001). Competitive on-line statistics. *International Statistical Review*, *69*(2), 213–248.

A Proofs

A.1 Analysis of the first player

Proof of Lemma 1. Given three random real numbers $a, b, c \in \mathbb{R}$, by algebraic manipulations we derive that the following equations are all equivalent:

$$\begin{aligned} ca - ba^2 &\leq \frac{c^2}{4b} \\ 4abc - 4a^2b^2 - c^2 &\leq 0 \\ (2ab)^2 - 4abc + c^2 &\geq 0 \\ (2ab - c)^2 &\geq 0 \end{aligned}$$

Since the square of a real number is always positive, the last line must be true. Therefore, the equivalent first line, which is the original statement, is also true. □

Next, we want to prove Lemma 3. To this end, we first need the following general result:

Lemma 5. *Let $\theta, x \in \mathbb{R}^n$ be two vectors with dimension n and let Y be a random variable. Then we have:*

$$(\langle \theta, x \rangle - \mathbb{E}[Y])^2 = \mathbb{E}\left[(Y - \langle \theta, x \rangle)^2 - (\mathbb{E}[Y] - Y)^2\right]$$

Proof. We start from the right-hand side of the equality:

$$\begin{aligned} &\mathbb{E}\left[(Y - \langle \theta, x \rangle)^2 - (\mathbb{E}[Y] - Y)^2\right] \\ &= \mathbb{E}\left[Y^2 - 2Y\langle \theta, x \rangle + \langle \theta, x \rangle^2 - \mathbb{E}\left[Y^2\right] + 2Y\mathbb{E}[Y] - Y^2\right] \\ &= \mathbb{E}\left[\langle \theta, x \rangle^2 - 2Y\langle \theta, x \rangle + \mathbb{E}\left[Y^2\right]\right] \\ &= \langle \theta, x \rangle^2 - 2\mathbb{E}[Y]\langle \theta, x \rangle + \mathbb{E}\left[Y^2\right] \\ &= (\langle \theta, x \rangle - \mathbb{E}[Y])^2 \\ &= (\langle \theta, x \rangle - \mathbb{E}[Y])^2 \end{aligned}$$

□

Proof of Lemma 2. We start with the definition of the regret of the first player:

$$\bar{R}_{\text{player 1}}(T) = \mathbb{E}\left[\sum_{t=1}^{T} \mu(w_t, p_t)\right] - \min_{w \in \mathcal{W}} \mathbb{E}\left[\sum_{t=1}^{T} \mu(w, p_t)\right]$$

In the proof below, we will sometimes leave out the $\min_{w \in \mathcal{W}}$; instead, we will write w_* for the argument of the $\min_{w \in \mathcal{W}}$ in the regret definition.

$$
\begin{aligned}
\mathbb{E}\left[\sum_{t=1}^{T}\mu(w_t,p_t)\right] &- \min_{w\in\mathcal{W}}\mathbb{E}\left[\sum_{t=1}^{T}\mu(w,p_t)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\mu(w_t,p_t)\right] - \mathbb{E}\left[\sum_{t=1}^{T}\mu(w_*,p_t)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\mu(w_t,p_t) - \sum_{t=1}^{T}\mu(w_*,p_t)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \sum_{h\in\mathcal{H}^i} p_t^{i,h}\mu(i,h) - \sum_{t=1}^{T}\sum_{i\in[K]} w_*^i \sum_{h\in\mathcal{H}^i} p_t^{i,h}\mu(i,h)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left\langle p_t^i,\mu^i\right\rangle - \sum_{t=1}^{T}\sum_{i\in[K]} w_*^i \left\langle p_t^i,\mu^i\right\rangle\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left\langle p_t^i,\mu^i\right\rangle - \sum_{t=1}^{T}\sum_{i\in[K]} 1\{i=i^*\} \left\langle p_t^i,\mu^i\right\rangle\right] \qquad (24) \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left\langle p_t^i,\mu^i\right\rangle - \sum_{t=1}^{T} \left\langle p_t^{i^*},\mu^{i^*}\right\rangle\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left\langle p_t^i,\mu^i\right\rangle - \sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left\langle p_t^{i^*},\mu^{i^*}\right\rangle\right] \qquad (25) \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\left\langle p_t^i,\mu^i\right\rangle - \left\langle p_t^{i^*},\mu^{i^*}\right\rangle\right)\right]
\end{aligned}
$$

In line 24 we used the fact that the optimal distribution for player 1 will always be to play some optimal move i^*. Moreover, in line 25 we used that w_t is a probability vector, and therefore $\sum_{i\in[K]} w_t^i = 1$. □

Proof of Lemma 3. For the proof, we use Lemma 5 and the fact that the move i_t played by player one is distributed according to w_t. Furthermore, we use the tower rule. The proof is then as follows:

$$
\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in[K]} w_t^i \left(\langle p_t^i,\hat{\mu}_t^i\rangle - \langle p_t^i,\mu^i\rangle\right)^2\right]
$$

$$
\begin{aligned}
&= \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{E}\left[\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - \langle p_t^{i_t}, \mu^{i_t}\rangle\right)^2\right]\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{E}\left[\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - \mathbb{E}\left[X_t(i_t, h_t) \mid i_t\right]\right)^2\right]\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \left(\mathbb{E}\left[X_t(i_t, h_t) \mid i_t\right] - X_t(i_t, h_t)\right)^2 \,\middle|\, i_t\right]\right]\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \left(\langle p_t^{i_t}, \mu^{i_t}\rangle - X_t(i_t, h_t)\right)^2 \,\middle|\, i_t\right]\right]\right] \\
&= \sum_{t=1}^{T} \mathbb{E}\left[\left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \left(\langle p_t^{i_t}, \mu^{i_t}\rangle - X_t(i_t, h_t)\right)^2\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T} \left(\langle p_t^{i_t}, \hat{\mu}_t^{i_t}\rangle - X_t(i_t, h_t)\right)^2 - \left(\langle p_t^{i_t}, \mu^{i_t}\rangle - X_t(i_t, h_t)\right)^2\right]
\end{aligned}
$$

□

A.2 Analysis of the second player

Proof of Lemma 4. We will use the fact that there is always an optimal strategy for the second player p that always plays a single optimal move. For each move i by the first player, such a move must exist; we call this optimal move $h^{i,*}$.

$$
\begin{aligned}
\bar{R}_{\text{player 2}}(T) &= \max_{p\in\mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{T} \mu(w_t, p) - \sum_{t=1}^{T} \mu(w_t, p_t)\right] \\
&= \max_{p\in\mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{T} \sum_{i\in[K]} w_t^i \sum_{h\in\mathcal{H}^i} p^{i,h}\mu(i,h) - \sum_{t=1}^{T} \sum_{i\in[K]} w_t^i \sum_{h\in\mathcal{H}^i} p_t^{i,h}\mu(i,h)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T} \sum_{i\in[K]} w_t^i \sum_{h\in\mathcal{H}^i} \mathbb{E}[1\{h = h^{i,*}\}]\mu(i,h) - \sum_{t=1}^{T} \sum_{i\in[K]} w_t^i \sum_{h\in\mathcal{H}^i} \mathbb{E}[1\{h = h_t\}]\mu(i,h)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T} \sum_{i\in[K]} w_t^i \mu(i, h^{i,*}) - \sum_{t=1}^{T} \sum_{i\in[K]} w_t^i \sum_{h\in\mathcal{H}^i} \mathbb{E}[1\{h = h_t\}]\mu(i,h)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T} \sum_{i\in[K]} w_t^i \sum_{h\in\mathcal{H}^i} \Delta_{i,h}\mathbb{E}[1\{h = h_t\}]\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \mathbb{E}\left[\sum_{i\in[K]}\sum_{t=1}^{T}\mathbb{E}[1\{i=i_t\}]\sum_{h\in\mathcal{H}^i}\Delta_{i,h}1\{h=h_t\}\right] \\
&= \mathbb{E}\left[\sum_{i\in[K]}\sum_{t:i_t=i}\sum_{h\in\mathcal{H}^i}\Delta_{i,h}1\{h=h_t\}\right] \\
&= \mathbb{E}\left[\sum_{i\in[K]}\mathbb{E}\left[\sum_{t:i_t=i}\sum_{h\in\mathcal{H}^i}\Delta_{i,h}1\{h_t=h\}\middle|\{t:i_t=i\}\right]\right] \qquad (26) \\
&= \mathbb{E}\left[\sum_{i\in[K]}\sum_{h\in\mathcal{H}^i}\Delta_{i,h}\mathbb{E}\left[T_h(n_i)|n_i\right]\right]
\end{aligned}
$$

Where in line 26 we used to tower rule by letting the outer expectation be on the randomness in i_t and the inner expectation on the randomness in h_t. □