Confidence Bound Algorithms in Game Trees

Effective Monte Carlo Tree Search through Bernstein's Inequality and the Law of the Iterated Logarithm

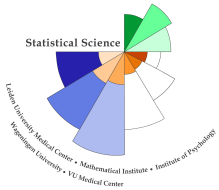
Marc Volkert (s1753908)

Thesis advisor: Dr. W.M. Koolen

Second advisor: Dr. T.A.L. van Erven

MASTER THESIS

Defended on October 19th, 2017



STATISTICAL SCIENCE

FOR THE LIFE AND BEHAVIOURAL SCIENCES

Abstract

Game trees have been utilized as a formal representation of adversarial planning scenarios such as two-player zero-sum games like chess [1, 2]. When using stochastic leaf values based on Bernoulli trials to model noisy game trees, a challenging task is to solve the Monte Carlo Tree Search (MCTS) problem of identifying a best move under uncertainty. Confidence bound algorithms are investigated as one solution, with focus on the FindTopWinner algorithm by Teraoka, Hatano, and Takimoto [3], which uses (a) the minimax rule to evaluate the game tree by alternately minimizing and maximizing over the values associated with each move, (b) Hoeffding's Inequality to estimate sample size requirements by fixing precision and error probability, and (c) an epoch-wise pruning regime to reduce investment on suboptimal nodes. We experimented on this algorithm by equipping it with methods that are based on (i) Bernstein's Inequality to create a tighter confidence bound [4], (ii) the Law of the Iterated Logarithm (LIL) to sample in single-sample steps, allowing for exact pruning and stopping [5, 6], and (iii) a combination of both. An empirically-derived Hoeffding-based Iterated-Logarithm confidence bound will be proposed in a fully refurbished FindTopWinner algorithm, which achieved much better performance in terms of samples required to find a best move, whereas the Bernstein-based approaches did not fare better than the original by Teraoka et al. [3]. Possible reasons such as limited, more asymptotic advantages for Bernstein-based algorithms will be discussed and the recommended parameter space for the empirically-derived Hoeffding-based confidence bound will be provided.

Keywords: game tree, minimax, MCTS, Bernstein's Inequality, Law of the Iterated Logarithm.

Acknowledgments

I would like to thank Wouter Koolen for his extraordinary support during all four months that I spent at the Centrum Wiskunde & Informatica (CWI) in Amsterdam. He established a steep learning curve for me from the start and taught me an abundance of new concepts and methods to reason about stochastic and algorithmic matters. He never avoided any (sometimes hours-long) discussions and always stayed attentive to the important things, especially when giving feedback to my work.

Further I am grateful to the Machine Learning group at the CWI to include me in their activities such as daily lunch and reading groups, which made me enjoy my time there.

Lastly, I wanted to thank Tim van Erven for his thoughtful additional suggestions and good provision of practical information during our meetings as my second supervisor.

Contents

Chapter 1

Introduction

Imagine having the ability to make an optimal choice among any fixed set of alternatives. This would prove to be an invaluable asset in a large number of scenarios: Think about financial planning situations, medical diagnoses, personal career decisions, but also about day-to-day considerations such as which products to buy in a store, or even which diet to follow. Indeed, we are confronted with choices all the time, which justifies the need to develop methods that can aid us with the decision-making processes. The field of statistics provides tools for inference and prediction by means of sampling, which is a general-purpose strategy to form our understanding and greatly improve our ability to make such optimal choices. However, sampling alone often does not suffice, since the adequate framework is missing. The remaining problem is that we often do not know *where* to sample and *when* we actually know enough. This is what we are going to consider in this thesis. To introduce the present work, consider the following decision dilemma:

Example. *You are the CEO of a mobile phone startup company and you are considering bringing out your first smartphone. At this stage you are not sure yet whether you would like to sell a discounted product, a premium product, or something lying between those two, knowing that they come with their own unique production costs and profit margins. You decide to be clever and make it dependent on the revenue estimated from customer polls. That is, you create surveys asking people whether they would buy a certain product for a certain price and, by multiplying the proportion of buys with the profit margin, take the one that maximizes your revenue. However, you realize there is a catch: You are not certain yet what your final costs will be. Depending on factors such as production delays or tax and patent issues, your spendings could be quite more than originally planned. Since you insist on keeping the initial profit margin for your product, the end-user price could be heavily affected. If that happens, your revenue might change depending on the product you chose and depending on whether the customer might still be interested in buying the product at a more expensive price. You face a problem: How can you make a decision when there is a range of possible outcomes that depend both on you and the customer?*

The dilemma derives from the fact that it is unclear how your choice and the choices of the customers might *interact*. If you decide on the premium product and it becomes even more expensive than it already is, customers might not buy it at all. Conversely, you could choose the discounted product and have more sales but a much smaller revenue, again depending on the additional costs that might come up. The example presented belongs to the field of *adversarial planning* scenarios in which two agents (here: you and the customer) seek to optimize their outcome, whereas optimization on one side means loss on the other side. Maximizing revenue on

your side usually means the customer pays more than he or she could have, and money-saving behavior on the side of the customer is damaging to your company profits.

Since real-life scenarios add a lot of unwanted complexity to the problem (for instance, we neglected potential win-win situations in the example above), it makes sense to look at more stylized adversarial planning settings first. Those can be found, for instance, at classical board games such as chess, checkers, or the ancient Chinese game "Go". For all of them, the rules dictate that moves have to be made to win against the other player, while a fixed set of possible moves is allowed during one turn. Note that such games have no noise, that is, in theory it is possible to see whether moves were good or bad by looking at the outcome of the game.

An appropriate framework to investigate such adversarial planning problems is to construct a *game tree*. In a game tree, possible states are modelled as *nodes* and decisions as *edges*, with the original state being called *root* and the possible outcome states (e.g., win or lose) being called *leaves*. See Figure 1.1 for a visualization of how such a game tree could look like.

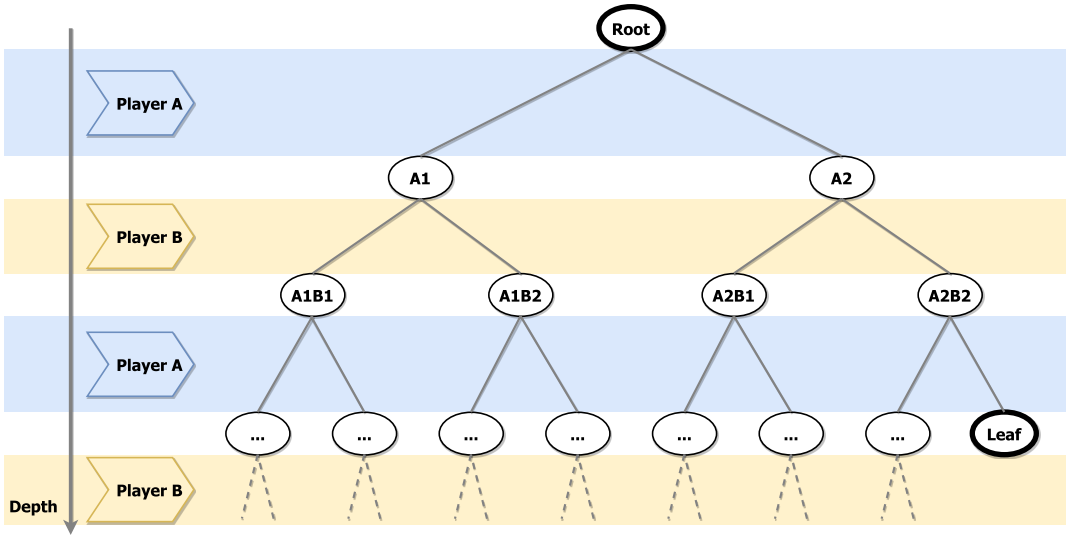


Figure 1.1: Example of a simplified binary game tree, i.e. where both players have two possible moves per turn. Please note that the game tree is incomplete, depicting states only until Player B is about to make his/her second move, while a real game can have many more moves until the leaves are reached and the game concludes. On the far right a leaf is depicted, showing a state where the game is over after Player A made his/her second move. The *depth* of a node describes how far the game has progressed already.

Whereas some games, like the board games above, have no noise, we will need to consider it in the general case (like our Example). That is, some games have noisy payoffs such that we are not always obtaining the same result for the same situation. In others these may arise from summarising subtrees by means of *rollouts* (see Section 1.2), which usually happens when a full game tree is too large, so that the amount of computations on it would become infeasible. We are going to model such noise by considering *stochastic* leaf values that follow Bernoulli distributions with (unknown) probabilities of win or lose.

The primary goal of the present work is to create an efficient algorithm for solving adversarial planning problems, that is, situations in which an opposing agent tries to counteract ones efforts, as modelled by stochastic game trees. More specifically, existing approaches will be outlined that are targeted at searching game trees to propose a best move from a set of alternatives

and subsequently steps will be presented towards upgrading these methods into a more powerful version.

Game tree search approaches have become quite famous recently when AlphaGo, a neural net paired with a game tree search algorithm defeated the Go grandmaster, making it to the front page of Nature [7]. Generally, it proves to be a vibrant field at this time [8, 9, 10], and since there is the potential for future applications in, for instance, financial or medical settings, it provides enough of a reason to dedicate one's efforts into developing this area of research.

In the remainder of this Introduction, I will describe game trees more formally, introducing important elements and operations on game trees. Next, in Chapter 2, I will discuss confidence bound algorithms, the state-of-the-art approach on searching game trees efficiently and will outline opportunities for improvements. In Chapters 3 and 4 I will propose implementation steps towards complete algorithms and validate them through experiments. Finally, I will conclude the thesis with a general discussion in Chapter 5.

1.1 Deterministic Trees

A two-player and competitive game tree, which will from now on be denoted by $\mathcal{T}$, is *deterministic* when everything is known about its structure: We know all nodes u and edges to corresponding children $c \in \text{children}(u)$. Most importantly, it implies that we know all potential outcomes of the game and their corresponding values.

Tree Search Problem

The most important learning problem lies in how to search the game tree in a way that we can predict the best possible move a player can make at any given point during the game - and this as efficiently as possible. In order to learn about whether a move is good or bad, we need to look into the future, i.e. further down the tree until we reach the leaves ℓ. Leaves contain the information on whether Player A or B has won. Assuming that we side with Player A, we can then advise on making a move into a certain direction at the current moment. One valid way of searching a tree for the best move is the *minimax* algorithm which works by backward induction and recursively identifies the best moves in an exhaustive search.

Minimax Action Identification

The special feature about leaves is that we know their value because they model potential endings of the game. In our case, each leaf ℓ contains the score $X_\ell = 1$ if Player A wins at leaf ℓ, and $X_\ell = 0$ otherwise.

Consider a situation in which there are three leaves connecting to a parent node. If the parent node lies at even depth and therefore models the state when Player A makes a move, we know that the player wants to make a winning move at this point. Therefore, each leaf $\ell \in \text{children}(u)$ is checked and, if any of them is a winning move (i.e., $X_\ell = 1$), this move is being made. If all of them are losing moves, the player has no chance of winning at this point and sticks to a losing move. The information will then be pushed to the next-level parent node u (e.g., $X_u \leftarrow 0$). On the next level, which is of uneven depth, the reasoning is reversed because it models the move of Player B, who wants to minimize Player A's chance of winning. Therefore, if $p = \text{parent}(u)$, all $u \in \text{children}(p)$ will be checked, and if there is a losing move for Player A, Player B will take this move and push its assigned value to the next level in the tree, $X_p \leftarrow 0$. The same way, if there is no possible losing move, the assigned value for a winning move for A, $X_p \leftarrow 1$, has to

be passed upwards. This scheme will be repeated until the root is reached. Note that for each internal node to be computed, its complete subtree has to be computed first.

Thus, given that we know the entire game tree $\mathcal{T}$, for any given node u, we can calculate the value of each node as such [3]:

$$X_u = \begin{cases} X_\ell, & \text{if } u = \ell. \\ \max\limits_{c \in \text{children}(u)} X_c, & \text{if depth of } u \text{ is even.} \\ \min\limits_{c \in \text{children}(u)} X_c, & \text{if depth of } u \text{ is odd.} \end{cases} \tag{1.1}$$

As we are interested in finding the best move, we need to consider the children $v \in \text{children(root)}$ and find the child v^* which maximizes X_v:

$$v^* = \arg\max_{v \in \text{children(root)}} X_v. \tag{1.2}$$

Consulting each leaf ℓ and calculating the minimax value will need a complete scan of the tree with computations on each node u; in other words it has complexity $\mathcal{O}(U)$ with $U = |nodes(\mathcal{T})|$.

Pruning

A method that can greatly reduce the effort being exerted when searching a tree, and therefore reduce complexity (i.e. the number of nodes to consult to know v^*), is *pruning*. When pruning a tree, a subtree which is determined to be irrelevant for further analyses is cut off from the tree, so that an algorithm is not required to perform additional computations on its nodes [11]. A node u_2 in a tree including all of its descendants can become a candidate for pruning in the deterministic tree setting when the minimax algorithm has found another neighbouring node u_1 being already assigned the maximal possible value in the even depth case, $u_1 \leftarrow \max\{0, 1\} = 1$, or the minimal possible value in the odd case, $u_1 \leftarrow \min\{0, 1\} = 0$. In other words, we do not need to look any further because an optimal move has been found and therefore the remaining subtrees can be pruned. This is a special case of what is called *alpha-beta pruning* and it is returning the same result as the minimax algorithm, but with reduced complexity averaging at about $\mathcal{O}(b^{3d/4})$, where b is the branching factor and d the combined sum of the moves taken by both players [11]. All of the concepts discussed above are summarised in Figure 1.2.

1.2 Non-Deterministic Trees

The concepts previously discussed assumed that we can model each possible direction a game could unfold within one tree. However, this is usually not the case from a computational perspective. Most game trees contain many moves with a relatively large branching factor, so that it becomes increasingly infeasible or impossible to perform rather expensive node computations on the full tree. Consider a game of chess: After both players have made a move, there are 400 possible chess positions; after both players have made two moves, this number has increased to 72,084 positions. A normal game lasts about 40 moves which, due to the exponential nature, is roughly estimated to allow for as many possible positions as there are electrons in the universe.

A creative solution has been developed to circumvent this issue: *Monte-Carlo tree search (MCTS)* [12]. Instead of walking from the root node to the final outcome of the game, the experimenter decides to ignore all nodes exceeding a fixed depth and only performs minimax and pruning computations on the known "small" tree. The deepest unignored nodes are now considered as leaves and, instead of modeling the true outcome of the game, they are replaced with Bernoulli distributions $\mathcal{B}(\mu)$ which model the potential outcomes from their summarised

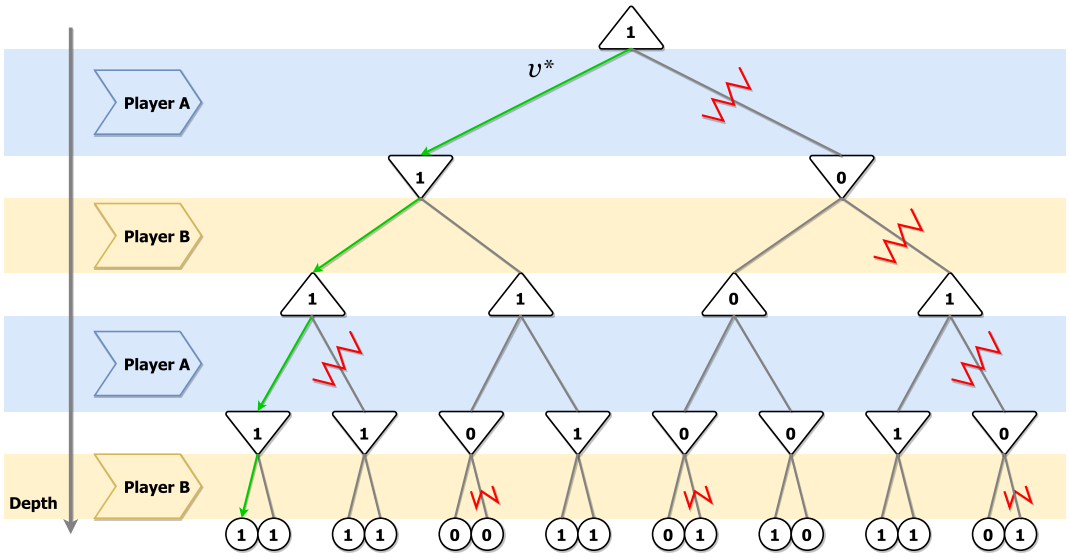


Figure 1.2: Extension of the binary game tree example from Figure 1.1. Here, a game is modeled in which both players make two moves in total before a winner is found. Nodes Δ correspond to *max* and ∇ to *min* computations over their children nodes. The green line shows one of the possible tracks of the minimax algorithm incl. unique best move v^* at the root. Candidate subtrees for pruning are indicated with the red line. Note that when we start the algorithm from the left-most leaf in this example, the pruning process would cut off the complete right side of the tree before even considering its subtrees. Ultimately, v^* would be found after looking at only 12 out of 31 nodes.

subtrees. When taking a sample at a leaf ℓ, computationally rather inexpensive (and biased) heuristics called *rollout* (also *playout*) are executed. For example, a very simple rollout may evaluate the subtree by making random moves for both players until a winner is found and return $X_\ell \leftarrow 1$ if Player A wins and $X_\ell \leftarrow 0$ otherwise. μ_ℓ is then used to represent the average winning probability estimated from this random Bernoulli process. Note that even though rollouts are often biased, here we ignore this bias and make the simplifying assumption that $\mathcal{B}(\mu)$ is the actual winning probability (for some games this is exact, but for the discussed rollouts it is not). The minimax computation for non-deterministic trees as compared to the deterministic version in Equation 1.1 looks as follows [3]:

$$\mu_u = \begin{cases} \mu_\ell, & \text{if } u = \ell. \\ \max\limits_{c \in \text{children}(u)} \mu_c, & \text{if depth of } u \text{ is even.} \\ \min\limits_{c \in \text{children}(u)} \mu_c, & \text{if depth of } u \text{ is odd.} \end{cases} \tag{1.3}$$

Monte Carlo Tree Search Problem

The MCTS problem comes as a logical extension of the tree search problem outlined before and includes information acquisition by sampling. The goal is to find a best move

$$v^* = \underset{v \in \text{children(root)}}{\arg\max} \mu_v, \tag{1.4}$$

which maximizes Player A's chance of winning.

Since we have no prior knowledge about any mean μ_ℓ of a leaf ℓ and only acquire its empirical estimate $\hat{\mu}_\ell = \frac{1}{N_\ell} \sum_{n=1}^{N_\ell} X_{\ell n}$ through sampling, a *strategy* is needed on how to translate sampling steps into knowledge about v^*. Such a strategy consists of three aspects [13]:

- **Sampling rule**: the regime we use to allocate samples at leaves ℓ to refine our knowledge about their respective estimated means $\hat{\mu}$.
- **Stopping rule**: a criterion that tells us when enough samples have been taken at all leaves and a guess about v^* can be made.
- **Recommendation rule**: some rule to decide on a final guess on v^*.

With such a strategy in mind, the MCTS problem is defined as follows:

Definition 1. (Monte Carlo tree search problem [3, 14]). *Given a tree $\mathcal{T}$ with nodes $v \in$ children(root) and corresponding means μ_v, define $\mu^* = \max_{v \in children(root)} \mu_v$. Take precision parameter $\varepsilon > 0$ and confidence parameter $\delta \in (0, 1)$. Then, we call a strategy (ε, δ)-PAC when it outputs a node $v \in$ children(root) such that*

$$\mathbb{P}(\mu_v \geq \mu^* - \varepsilon) \geq 1 - \delta. \tag{1.5}$$

In other words, we consider the largest possible winning probability μ^* after the minimax method has been performed. By allowing for a range of uncertainty ε when making the decision on a move, we can be sure with at least $1 - \delta$ probability that we have identified a move that is close enough (or equal) to having winning probability μ^*. We will make any move v that fulfills the requirement of $\mu_v \geq \mu^* - \varepsilon$. This is referred to as (ε, δ)-PAC learning, that is, being *Probably Approximately Correct* up to precision ε with error probability δ.

When sampling from a leaf ℓ we refine its estimate $\hat{\mu}_\ell$ and, according to the Law of Large Numbers, reduce its deviation from μ_ℓ. Therefore, Inequality 1.5 is related to the number of samples taken at a leaf, N_ℓ, with larger N_ℓ reducing our uncertainty ε. Among all (ε, δ)-PAC algorithms we are interested in algorithms with (near) minimal *sample complexity*, that is, we want the total amount of samples taken across all leaves, $N = \sum_\ell N_\ell$, to be as small as possible.

MCTS: Pruning and Epochs

Pruning mechanisms are also possible for non-deterministic trees being subject to the MCTS problem [3]. However, they must work differently because we are uncertain about the μs. Consider the case where we have spent some samples on the leaves and hence believe, for instance, that it is very likely that a specific leaf ℓ_1 has mean $\mu_{\ell_1} < 0.5$ (i.e., almost all draws from $\mathcal{B}(\mu_{\ell_1})$ turned out 0), while for another leaf ℓ_2 it is the opposite, $\mu_{\ell_2} > 0.5$. If those leaves share the same parent, and this parent lies at even depth, we would want to prune ℓ_1 since it is not a very promising alternative. Similarly, this can happen anywhere in the tree after the minimax rule has been performed as in Equation 1.3. Any time we are very certain that a $\mu_{u_1} > \mu_{u_2}$, we can prune μ_{u_2} and its complete subtree (again in the even depth case). The next chapter will elaborate on how we can be certain that we know enough.

Obviously, the strength of our belief depends completely on the amount of samples spent on a leaf. It implies that we should spend a lot of samples before inquiring on whether to prune any node u with its subtree. On the other hand, we try to minimize sample complexity and want to spend the minimum amount of samples. One solution to this tradeoff is to prune in *epochs* [3]. That is, during each epoch $m \in 1, 2, ..., M$ we take samples at the leaves, perform the minimax

method, and subsequently identify nodes for pruning. The result is a tree that iteratively shrinks and therefore sample complexity is reduced compared to a naive approach where no pruning is being done.

Chapter 2

Confidence Bound Algorithms

Confidence bound algorithms traditionally stem from research on *multi-armed bandits*, which can be considered a special case of non-deterministic trees, i.e. those of height 1. The interest in bandit models originally derived from clinical research [15], but they have been increasingly applied to other areas too [10, 16]. Similar to the MCTS problem, a best arm $a^* \in \{a_1, a_2, ..., a_K\}$ with expected reward $\mu^* = \max_i \mu_i$ needs to be identified, assuming distribution p in which each arm has a mean μ. Through pulling arms, i.e. taking samples, the experimenter learns over time which arm provides the best result.

An important topic in the bandit literature is the *exploration versus exploitation dilemma*, describing two opposed objectives when performing online learning through sequential sampling from alternatives [10, 14, 17]. Exploration refers to the goal of knowing as much as possible about the alternatives, therefore estimating a^* as precisely as possible. Conversely, one talks about exploitation when sampling as often as possible from the alternative that currently holds the strongest estimate. To appreciate the difference, consider a clinical setting in which we want to give the best one from several new drugs to patients. A exploitative researcher might prefer to give the drug that yields the best outcome, based on the drugs he has given so far, as often as possible, hence knowing less about the remaining drugs. An explorative researcher, as is traditionally the case in clinical trials, splits the drug alternatives more or less equally across patients, so that a precise estimate can be made about the drugs' effects. With the first researcher, the patients in the trials benefit, with the latter researcher, future patients benefit.

Algorithms within this dilemma usually seek to minimize *regret*, which is the loss obtained from sampling from alternatives that are rather unlikely to yield success. At the same time they want to obtain enough knowledge by exploration to be sure about which alternative should be exploited. Research has been focused on finding optimality, however it has been proven that one cannot achieve both at the same time [18]. *Upper confidence bound (UCB)* algorithms were first proposed by Auer, Cesa-Bianchi, and Fischer [17]. They take advantage of what is known about the arms a, usually favoring the one that maximizes $\hat{\mu}$, while on the same time giving an "exploration bonus" to the little explored arms.

The MCTS problem in game trees does *not* correspond with this paradigm by proposing a pure exploration setting in which the goal is to identify the best move v^*. However, UCB algorithms were succesfully applied to internal nodes within a tree (*UCT* [19]). In this setting, a node and its children were considered a multi-armed bandit. The advantage of UCT algorithms is that rather extreme winning probabilities are readily identified within a tree, which has led to its implementation in many GO programs [20]. However, it seems rather counter-intuitive to apply regret optimization methods in a pure exploration settings. This claim is further supported by the fact that it is lacking a theoretical analysis for finite-sample results, achieving only asymptotic

validity [21].

2.1 Concentration Inequalities

At the very core of confidence bound algorithms lies some sort of criterion that relates precision ε, error probability δ, and total amount of samples N in such a way that, by fixing two of them, the third parameter can be optimized. Hoeffding's inequality (outlined next in Theorem 1) is a general case of such concentration inequalities and has been successfully applied to UCB/UCT algorithms [13, 10].

Theorem 1. (Hoeffding's inequality: Bernoulli [22]). Given i.i.d. random variables $X_1, X_2, ..., X_N \sim \mathcal{B}(\mu)$ and $\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} X_n$. Then,

$$\mathbb{P}\left(|\hat{\mu} - \mu| \leq \sqrt{\frac{\log(2/\delta)}{2N}}\right) \geq 1 - \delta. \tag{2.1}$$

In other words, the event that the difference between the true mean μ and its estimate $\hat{\mu}$ will be smaller or equal to $\varepsilon = \sqrt{\log(2/\delta)/(2N)}$ will be valid with probability at least $1 - \delta$. By increasing the total amount of samples N, the square root becomes smaller and hence the precision ε gets tighter. Note that δ has only minor impact on the bound (if it is not very small) because of the log(.).

2.2 FindTopWinner

The FindTopWinner algorithm by Teraoka, Hatano, and Takimoto [3] derives from the idea of UCB algorithms and their UCT extension for the MCTS problem (see Algorithm 1 for a summary). It considers a pure exploration scenario in contrast to regret optimization: whereas the original UCT algorithm treats each level of a tree as its own multi-armed bandit from which it seeks to recommend the best arm a^* by minimizing regret [19], Teraoka et al. approached the MCTS problem by sampling from leaves, performing the minimax by applying Equation 1.3 to estimates $\hat{\mu}_u$ instead of the true values μ_u, and pruning weak subtrees. They do this by fixing ε and δ. From there the N necessary to satisfy Hoeffding's inequality is determined. According to Hoeffding's inequality, by making sure that $\varepsilon = \sqrt{\log(2/\delta)/(2N)}$, Inequality 2.1 holds. Solving for the total amount of samples N and rounding to the next integer, we obtain:

$$N_H = \left\lceil \frac{\log(2/\delta)}{2\varepsilon^2} \right\rceil, \tag{2.2}$$

with N_H being the minimum amount of samples when using the Hoeffding confidence bound. Applying this to trees, the FindTopWinner algorithm extends its efficiency by implementing an epoch-wise pruning approach as outlined at the end of Section 1.2. More precisely, for each successive epoch $m = \{1, 2, ..., M\}$, they defined a new δ_m and ε_m (see end of this section for precise allocation) and defined the pruning criterion as follows:

Definition 2. (Pruning criterion [3]). *At the end of epoch m, prune $c \in$ children(u) and its subtree if*

$$|\hat{\mu}_u - \hat{\mu}_c| > 2\varepsilon_m. \tag{2.3}$$

To see why this is a good idea, consider the case where node u is at even depth and we are in epoch m. After the samples have been taken the minimax method pushes the largest updated estimates for the means of its children to u. Then, when we know that $\hat{\mu}_c$ is more than $2\varepsilon_m$ smaller than that, we prune c. Note that a factor of 2 is necessary to incorporate all cases in which both the nodes compared are misestimated into the opposite direction (i.e., $\hat{\mu}_1 = \mu_1 + \varepsilon_m$, $\hat{\mu}_2 = \mu_2 - \varepsilon_m$).

Multiple Testing Problem

According to Theorem 1 after each estimation of μ the error probability δ tells us the likelihood of the estimate $\hat{\mu}$ not satisfying $\varepsilon \geq |\hat{\mu} - \mu|$. Since we are interested in one estimate $\hat{\mu}_\ell$ per leaf ℓ and there are in total L leaves, it comes as no surprise that the side effects of doing multiple tests need to be incorporated into a pruning strategy. Second, there are successive estimation procedures across epochs m, feeding into the same inflation in δ. One way to approach the multiple testing problem is to use a union bound in δ. A general definition of the union bound is as follows:

Proposition 1. (Union bound). *Take event A and B. It holds that*

$$\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B). \tag{2.4}$$

Therefore, by splitting the global δ across leaves and epochs, it is ensured that we will stay below the global δ, that is:

$$\delta = \sum_{\ell=1}^{L} \sum_{m=1}^{M} \delta_{m,\ell}. \tag{2.5}$$

Allocating δ and ε

The FindTopWinner algorithm seeks to prune as quickly as possible, which is achieved by distributing the global error probability δ in a way that the Hoeffding bound is tighter at early stages where more pruning happens, while at the same time satisfying the union bound. They decided to split the input δ equally across leaves and allocate it in a 2^{-m} fashion across epochs m. Similarly, ε is initialized at $\varepsilon_0 = 1$ and treated the same way, so that exponentially more precision (and hence samples) are required. The algorithm continues until epoch $M = \lceil \log_2 2/\varepsilon \rceil$ is reached, which is when the requested input precision ε has been achieved. However, the algorithm may terminate earlier in the case that all but one child $v \in$ children(root) have been pruned away at the end of epoch $m < M$, leaving only one possible recommendation v. A summary of the essential procedure used for updating both δ and ε can be found in Algorithm 1. (Note that details about minimax and pruning are omitted here, for the full implementation including recursive functions consult Teraoka, Hatano, and Takimoto [3].)

While a one-shot algorithm needs $\mathcal{O}(L/\varepsilon^2)$ samples with a tree with L leaves and wanting to know the best move ε-close, they reduced the sampling complexity to

$$\mathcal{O}\left(L \min\left(\frac{1}{\varepsilon^2}, \frac{1}{\Delta^2}\right)\right), \tag{2.6}$$

with

$$\Delta = \min\{\mu^* - \mu_v \mid v \in \text{children(root)}, \mu_v < \mu^*\}, \tag{2.7}$$

Algorithm 1 Updates in FindTopWinner

Input $\varepsilon, \delta, \mathcal{T}$.
Initialization $\varepsilon_0 = 1, \delta_0 = \delta/L$
for $m = 1$ **to** $\lceil \log_2 2/\varepsilon \rceil$ **do**
  if $|\text{children(root)}| = 1$ **then**
    Recommend $I = v \in \text{children(root)}$.
  end if
  $\varepsilon_m = \varepsilon_{m-1}/2$ and $\delta_m = \delta_{m-1}/2$
  $N_m = \lceil \log(2/\delta_m)/(2\varepsilon_m^2) \rceil$
  for $\ell \in \text{leaves}(\mathcal{T})$ **do**
    Add $N_m - N_{m-1}$ samples to leaf ℓ.
    Estimate $\hat{\mu}_\ell$.
  end for
  Perform minimax on $\mathcal{T}$ as in Equation 1.3.
  Prune and update $\mathcal{T}$ as in Inequality 2.3.
end for
Recommend $I = \arg\max_{v \in \text{children(root)}} \hat{\mu}_v$.

which can be considerably smaller. Note that FindTopWinner seeks to know about the possible moves as much as is necessary to rule out alternatives, which makes it an exploration rather than an exploitation algorithm. On the other hand, iterative pruning ensures us of not wasting too many samples on bad alternatives.

In sum, the combination of checks through the Hoeffding bound and epoch-specific pruning of weak subtrees creates an efficient algorithm that further convinces by its applicability to trees of any depth and (a)symmetry.

2.3 Problem Description

While FindTopWinner comes with several advantages, there is also room for improvement. Indeed, it is possible to locate aspects of the algorithm which could be upgraded into a new version which keeps the advantages while potentially improving performance. Three main complication areas of the algorithm can be outlined. Their description plus the approaches to resolve them, making up the following chapters, are as follows:

1. **Concentration inequality**: Hoeffding's inequality is a general case and can be improved upon by inequalities that take more information into account. The most relevant, *Bernstein's inequality*, includes variance parameter σ^2. Since this work considers game trees with binary outcomes, we can take advantage of the fact that for the Bernoulli distribution σ^2 is determined by μ and implement this into the algorithm.

2. **Arbitrary** ε: When sampling according to a regime in which ε is halved in every epoch, we often end up overshooting the anticipated precision and spend more samples as a result. Instead, the FindTopWinner algorithm should be flipped such that N and δ are fixed and ε will be evaluated. This allows us to sample according to our own budget N and to allocate samples at leaves ℓ at which we need to refine our knowledge.

3. δ **management**: The algorithm obtains samples in batches of exponentially increasing size to avoid multiple testing issues. This often lets the algorithm overshoot the amount of

samples needed to be ε-close. It is possible to replace the present paradigm with a smooth transition and an optimal stopping criterion. The *Law of the Iterated Logarithm (LIL)* will be applied to transform the concentration inequality in such a way that samples can be added one by one without any inflation of δ.

In the following chapters, upgrades will be introduced first theoretically by providing the relevant implementation steps. Fully developed algorithms will be proposed, accompanied by simulations on two stages: (1) single-leaf simulations to validate the success of the implementation, (2) large three-level trees identical to how Teraoka, Hatano, Takimoto validated the success of the original FindTopWinner algorithm [3], measuring and comparing performance in a realistic setting that includes minimax and pruning. Their benchmark setup depicts the deviation from the score μ^* of the best move v^* according to the number of samples that has been taken. To achieve this, during each epoch the mean of the best candidate $\mu_I \in \mu_v$ is compared with the minimax rule result in the case that we actually have access to the true μs and therefore know v^*. Their experiment has been reproduced in Figure 2.1.

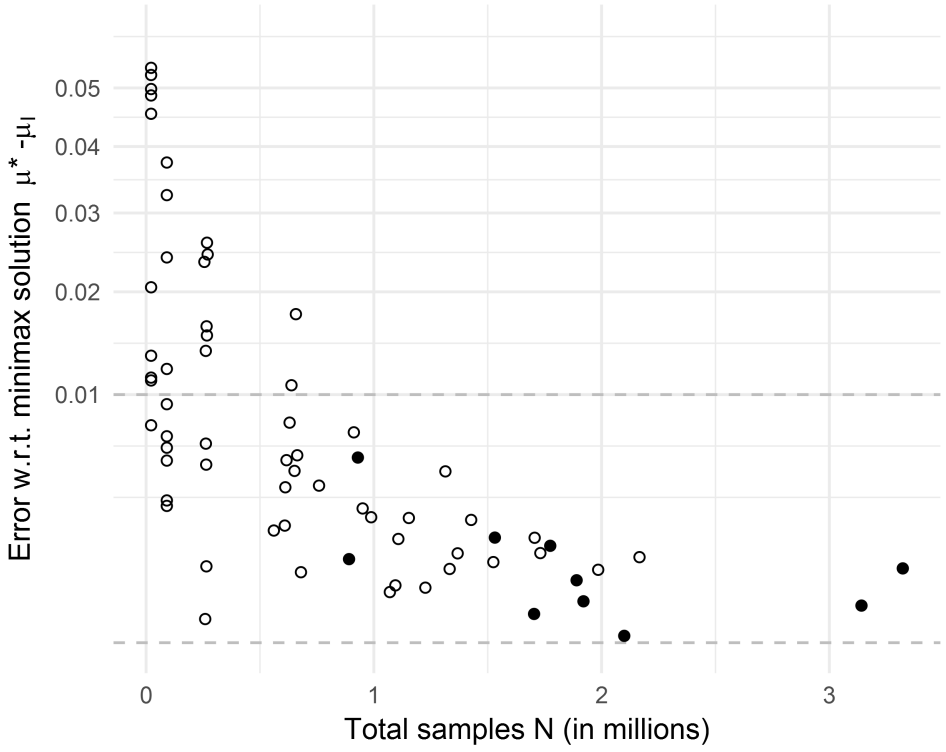


Figure 2.1: Experiments on a full tree of depth 3 with branching factor $b = 10$, resulting in 1000 leaves [3]. Rollouts at leaves were simulated from a uniform distribution with interval [0, 1]. Each observation corresponds to the number of samples taken during an epoch and the deviation $|\mu^* - \mu_I|$. If a point is filled (•) it indicates that the algorithm has converged and μ_I was returned. The y-axis depicts the square root of the deviation to highlight differences when it comes to small errors. Input parameters $\varepsilon = 0.01$ and $\delta = 0.1$ were chosen.

In the last chapter, the results and potential issues will be discussed, followed by a general discussion with future considerations.

Chapter 3

Bernstein Sampling

The Bernstein bound will improve on the simple Hoeffding bound by taking $\sigma^2 = \mathrm{Var}[X]$ into account. For the Bernoulli case, the Hoeffding bound implicitly assumes $\sigma^2 = \max(\mathrm{Var}[X]) = 1/4$, which Bernstein's inequality replaces with the empirical estimate $\hat{\sigma}^2$. Especially when the variance is small this should lead to improvements in the bound and actual performance. This happens for extreme μ (i.e. close to 0 or 1) as is implied by the variance definition of the Bernoulli: $\sigma^2 = \mu(1-\mu)$. Regarding the MCTS, Bernstein's inequality leads to the following statement as compared to Theorem 1:

Theorem 2. (Bernstein's inequality: Bernoulli [4]). Given i.i.d. random variables $X_1, ..., X_N \sim \mathcal{B}(\mu)$, we define $\mu = \mathbb{E}(X_n)$ and $\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} X_n$. Further take $\sigma^2 = \mathrm{Var}[X_n]$ and $\hat{\sigma}^2 = \frac{1}{N-1}\sum_{n=1}^{N}(X_n - \hat{\mu})^2$. Then,

$$\mathbb{P}\left(|\hat{\mu} - \mu| \leq \hat{\sigma}\sqrt{\frac{2\log(3/\delta)}{N}} + \frac{3\log(3/\delta)}{N}\right) \geq 1 - \delta. \tag{3.1}$$

3.1 Implementation

To put this into the FindTopWinner framework, we seek to take enough (but not too many) samples to guarantee $\varepsilon \geq |\hat{\mu} - \mu|$ with error probability δ for fixed ε. Solving for N, we get the following minimum amount of samples that have to be taken to stay inside the Bernstein bound:

$$N_B \geq \frac{1}{\varepsilon^2}\left(\sqrt{\hat{\sigma}^2(\hat{\sigma}^2 + 6\varepsilon)\log^2(3/\delta)} + (\hat{\sigma}^2 + 3\varepsilon)\log(3/\delta)\right). \tag{3.2}$$

Importantly, in order to find N_B, we have to know $\hat{\sigma}^2$, which is dependent on N_B. For that reason, estimation of N_B is not possible at this stage. A simple, yet effective way to approach the problem is to make a guess for N_B, N_B', take N_B' samples, then compute $\hat{\sigma}$, and check whether the resulting amount of samples suffices:

$$\varepsilon \geq \hat{\sigma}\sqrt{\frac{2\log(3/\delta)}{N_B'}} + \frac{3\log(3/\delta)}{N_B'}. \tag{3.3}$$

If it does, it is made sure that the amount of samples is large enough to have estimates inside the Bernstein bound. A naive rule would be to try a small N_B' and test it: If it fails, *double* it, and repeat until it falls within the Bernstein bound. This comes at a cost, however, since involving

additional checks feeds into the multiple testing issue outlined in Section 2.2. The problem is fixed by halving δ as it is done with epochs m in line 4 of Algorithm 1 for any additional check that is being performed, making sure that the union bound holds (see Proposition 1). Thus, by additionally taking into account that there are L leaves in a tree $\mathcal{T}$ (i.e. the computations and checks above need to be executed L instead of 1 times) and there are in total M epochs, global error probability δ is preserved as follows:

$$\delta = \sum_{\ell=1}^{L} \sum_{m=1}^{M} \sum_{t=1}^{T} \delta_{m,\ell,t}, \tag{3.4}$$

where T is the total amount of tests made for leaf ℓ at epoch m. Therefore, during each epoch $m \in \{1, 2, ..., M\}$, we seek to satisfy:

$$\varepsilon_m \geq \hat{\sigma}_{m,\ell,t} \sqrt{\frac{2\log(3/\delta_{m,\ell,t})}{N_{m,\ell,t}}} + \frac{3\log(3/\delta_{m,\ell,t})}{N_{m,\ell,t}}. \tag{3.5}$$

3.2 Algorithm: Naive Bernstein

Reconsidering the simple updating rule to estimate $\hat{\sigma}^2$, we should be able to implement a new updating procedure, which is summarised in Algorithm 2 with simplified notation. (As before, consult Teraoka, Hatano, and Takimoto [3] for details about minimax and pruning.)

Algorithm 2 Updates in FindTopWinner: Naive Bernstein

1: **Input** $\varepsilon, \delta, \mathcal{T}$
2: **Initialization** $\varepsilon_0 = 1$ and $\forall \ell : \delta_\ell = \delta / L$
3: **for** $m = 1$ **to** $\lceil \log_2 2/\varepsilon \rceil$ **do**
4: **if** $|\text{children(root)}| = 1$ **then**
5: Recommend $I = v \in \text{children(root)}$.
6: **end if**
7: $\varepsilon_m = \varepsilon_{m-1}/2$
8: **for all** $\ell \in \text{leaves}(\mathcal{T})$ **do**
9: $\delta_\ell = \delta_\ell / 2$
10: $N_{m,\ell} = \lceil 3\log(3/\delta_\ell)/\varepsilon_m \rceil$
11: Add $N_{m,\ell} - N_{m-1,\ell}$ samples to leaf ℓ .
12: Compute $\hat{\mu}_\ell$ and $\hat{\sigma}_\ell$.
13: **while** $\varepsilon_m < \hat{\sigma}_\ell \sqrt{\frac{2\log(3/\delta_\ell)}{N_{m,\ell}}} + \frac{3\log(3/\delta_\ell)}{N_{m,\ell}}$ **do**
14: $\delta_\ell = \delta_\ell / 2$
15: $N_{m,\ell} = 2N_{m,\ell}$
16: Add $N_{m,\ell} - N_{m-1,\ell}$ samples to leaf ℓ .
17: Compute $\hat{\mu}_\ell$ and $\hat{\sigma}_\ell$.
18: **end while**
19: **end for**
20: Perform minimax on $\mathcal{T}$ as in Equation 1.3.
21: Prune and update $\mathcal{T}$ as in Inequality 2.3.
22: **end for**
23: Recommend $I = \arg\max_{v \in \text{children(root)}} \hat{\mu}_v$.

During each epoch m we make an initial lower-bound guess for the total amount of samples we have to take to satisfy Inequality 3.5 (Line 10 in the algorithm), which we obtained by considering the most friendly case $\hat{\sigma}_\ell^2 \to 0$. In other words, we try the smallest possible number of samples and try whether it achieves the following: the probability of the difference between the estimated mean $\hat{\mu}_\ell$ and the true mean μ_ℓ of a leaf ℓ being larger than ε_m must be smaller than $\delta_{m,\ell}$.

The algorithm will then take samples until $N_{m,\ell}$ is reached, calculate $\hat{\sigma}_\ell^2$, and check whether Bernstein's inequality holds. If this is not the case, $N_{m,\ell}$ will be doubled and the procedure will be repeated. This happens for each leaf ℓ.

The expected gain over the original FindTopWinner Algorithm 1 is that we will be in a tighter bound through Bernstein's inequality. The loss will be that we have to, as with the epochs, account for the increased amount of checks we will use, affecting $\delta_{m,\ell}$. Every time we have to double the amount of samples, we need to consume from, what we could call "resource", δ. Note that by splitting this resource across the leaves and from there handling it individually, doublings for the sampling at a specific leaf l only affects $\delta_{\ell=l}$ instead of all δ_ℓ, which prevents loss from overall δ.

3.3 Improvements

Under the present conditions a naive Bernstein algorithm usually performs worse than the original FindTopWinner Algorithm 1 or gets stuck (!) for several reasons that will be outlined below. This section will propose steps towards a refined algorithm.

Lower Bound Complications

At every epoch we set a new lower bound of samples through updated ε_m and $\delta_{m,\ell}$ in Line 10. Through the doublings happening for each leaf we sometimes end up with an amount of samples that is larger than what would have been sampled as a lower bound during the next epoch (compare Lines 10 with Lines 13). This can be solved by including a $\max(N_{new}, N_{old})$ statement at the beginning of each new epoch: Either we take the minimum sample size given by the lower bound for the new epoch or we take the sample that resulted from the last doubling procedure for that leaf.

Also, the doubling procedure might let a leaf end up being sampled more often than through the traditional Hoeffding bound. To counteract this, we could control the maximum amount of samples being taken during the doubling procedure through a check with the Hoeffding bound. A simple way to do this would be:

$$N_{m,\ell} = \min(N_{m,\ell}, N_{m,H}).$$

$N_{m,H}$ is the amount of samples necessary to satisfy the Hoeffding bound in epoch m. The Hoeffding bound prevents the algorithm from sampling too much: In the worst case scenario it leaves us with the same amount of samples as in the traditional FindTopWinner algorithm. In the best case, the Bernstein check is successful with $N_{m,\ell} < N_{m,H}$. Unfortunately, every time we check whether the Bernstein approach was successful we need to spend a part of δ because of the union bound across tests t, since otherwise the algorithm may no longer be (ε, δ)-PAC. This makes it a costly procedure. It turns out that such an algorithm can get stuck in the following scenario and is therefore not advisable:

1. For leaf ℓ in epoch m, ε_m is not achieved, therefore $N_{m,\ell}$ needs to be doubled.
2. $\delta_{m,\ell}$ is being halved in the process.

3. $N_{m,\ell}$ exceeds the Hoeffding bound, $N_{m,H}$ is used instead.

4. $N_{m,H}$ does not achieve ε_m, because of the new $\delta_{m,\ell}$.

Deterioration of δ

Currently the lower bound is implicitly assuming each epoch anew that $\sigma_\ell^2 = 0$. When σ_ℓ^2 is big it might happen that the algorithm needs to double the lower bound several times until precision ε_m is reached. To work out what is happening, let's assume for now that indeed our estimate for the variance $\hat{\sigma}_\ell^2 = \sigma_\ell^2$ is exactly correct. Then, the factor F^+ that is necessary to get from the lower Bernstein bound as in line 6 of Algorithm 2 to the general Bernstein bound as in Inequality 3.2 is obtained by dividing the first by the second:

$$F^+ = \frac{(3\varepsilon + \sigma_\ell^2)\log(3/\delta_\ell) + \sqrt{\sigma_\ell^2(6\varepsilon + \sigma_\ell^2)}\log(3/\delta_\ell)}{3\varepsilon \log(3/\delta_\ell)}. \tag{3.6}$$

Because all variables are positive, we can simplify this to:

$$F^+ = 1 + \frac{\sigma_\ell\sqrt{6\varepsilon + \sigma_\ell^2} + \sigma_\ell^2}{3\varepsilon}. \tag{3.7}$$

Importantly, δ does not play a role here, whereas, in our algorithm, the current ε_m will have a big impact on how many doublings there might be necessary at a given point to reach N_ℓ. More precisely, the later the epoch, the more doublings will be necessary. This inflates in a multiplicative fashion when σ_ℓ^2 is large and can lead to a strong deterioration of δ through successive halvings. Since we cannot directly alter the effects of epochs without considerably changing the paradigm at this stage, we should therefore consider methods that take our knowledge about σ_ℓ^2 into account.

MaxLow. By using the knowledge we already have about σ_ℓ^2 in the form of its estimate $\hat{\sigma}_\ell^2$ from previous epochs, we can pick a close starting point for the algorithm to double from. Thus, we estimate the sample size $N_{m,\ell}$ of the Bernstein bound as in Inequality 3.2 for leaf ℓ based on $\hat{\sigma}_{m-1,\ell}^2$:

$$N_{m,\ell} = \left\lceil \frac{1}{\varepsilon_m^2}\left(\sqrt{\hat{\sigma}_{m-1,\ell}^2(\hat{\sigma}_{m-1,\ell}^2 + 6\varepsilon_m)\log^2(3/\delta_\ell)} + (\hat{\sigma}_{m-1,\ell}^2 + 3\varepsilon_m)\log(3/\delta_\ell)\right)\right\rceil. \tag{3.8}$$

ε-Low. Another approach would be to make use of the ε-criterion of our algorithm: We know with $1-\delta$ certainty that $\hat{\mu}$ estimates μ within error margin ε. This can be used as separate lower-bound during a new epoch for each leaf ℓ. We might want to make sure that we start low enough to cover the true mean μ_ℓ with high probability. This means that when estimating $N_{m,\ell}$, we replace $\hat{\sigma}_{m-1,\ell}^2$ with

$$\hat{\sigma}_{\text{small}}^2 = \hat{\mu}_{\text{extreme}}(1 - \hat{\mu}_{\text{extreme}}), \tag{3.9}$$

where

$$\hat{\mu}_{\text{extreme}} = \begin{cases} \max(0, \hat{\mu}_{m-1,\ell} - \varepsilon_m), & \text{if } \hat{\mu}_{m-1,\ell} \leq 0.5 \\ \min(1, \hat{\mu}_{m-1,\ell} + \varepsilon_m), & \text{if } \hat{\mu}_{m-1,\ell} > 0.5 \end{cases} \tag{3.10}$$

The result is that either σ_ℓ^2 is small enough to assume it is 0 in our equation or we estimate the lowest possible value it could take up with δ error probability and based on what we know about ε_m. Unfortunately, simulations did not indicate any improvements from this method.

Parameter Tuning

Algorithm 2 allows to introduce new parameters: A best possible "doubling" factor F^* (instead of 2), and a largest allowed σ^{cut} to decide when Bernstein's of Hoeffding's inequality will be used. Here we discuss their tuning.

Factor

So far we were only considering an approach in which we *double* $N_{m,\ell}$ and neglect possible other factors we could use to increase the amount of samples. We could imagine a best doubling factor F^* that leads to optimal performance. However, such a factor depends on μ and δ in a complex function with a typical range between $1.4 < F^* < 2.5$ and can therefore only be estimated through simulations. This was investigated more thoroughly on a single leaf (experiments are not presented here), with the result that the amount of samples necessary were minimized by, instead of doubling, taking a factor of $F^*_{\text{MaxLow}} \approx 1.468$ and $F^*_{\varepsilon\text{-Low}} \approx 1.6$. Note that it is a relatively stable constant for MaxLow, while for ε-Low there is a stronger dependency on μ, which means that we should optimize the approximation by including a function that decides the factor based on our estimate for μ, which makes ε-Low less attractive as a solution to the lower-bound problem. Therefore it will not be included in the final Bernstein Algorithm 3 and the Experiments in Section 3.5 and only the MaxLow approach will be considered.

Largest effective σ

From what we know about Theorems 1 and 2, it is impossible that Bernstein's inequality outperforms Hoeffding's inequality when a leaf ℓ has the maximal estimated variance $\hat{\sigma}^2_\ell = 1/4$. Bernstein's inequality contains additional constants that are necessary for the inclusion of the variance parameter σ^2. Only asymptotically they should converge. On the other hand, extreme μ and therefore small σ^2 should keep the Bernstein below the Hoeffding bound and provide an asymptotical advantage. Additionally, the cost of the doubling procedure leads to a δ-wise restriction for the Bernstein bound to perform better than the Hoeffding bound. Further, the additional constants complicate non-asymptotical results because they affect the behavior for early epochs in the FindTopWinner. This means that even if leaf ℓ has small σ^2_ℓ, constants will make the anticipated advantage through the Bernstein bound vanish. This is also what was found with simulations on a single leaf, see Figure 3.1 for an impression.

This is slightly affected by δ (*not* ε), so that when there are many epochs and leaves, the disadvantage through doublings has less impact as before, suggesting that for large trees the Bernstein FindTopWinner works better.

VarCheck. A crude, but straightforward way to remove disadvantages of this sort would be to only use Bernstein's inequality when $\hat{\sigma}$ drops below a cutoff (as a rule of thumb, take $\sigma^{\text{cut}} = 0.4$). This way, we can use the advantages of the Bernstein bound for leaves with small $\hat{\sigma}$ and avoid its disadvantages when $\hat{\sigma}$ is large.

3.4 Algorithm: Effective Bernstein

The improvements of the previous section bring us to the final Algorithm 3 (using the MaxLow approach). The Bernstein algorithm allows for a tighter confidence bound for extreme μ and adapts its sampling strategy to the original FindTopWinner when μ is less extreme.

Algorithm 3 Updates in FindTopWinner: Bernstein MaxLow VarCheck

Input $\varepsilon, \delta, \mathcal{T}$, cutoff σ^{cut}.
Initialization $\varepsilon_0 = 1$ and $\forall \ell : \delta_\ell = \delta/L, \hat{\sigma}_\ell = 0$
for $m = 1$ **to** $\lceil \log_2 2/\varepsilon \rceil$ **do**
  if $|\text{children(root)}| = 1$ **then**
    Recommend $I = v \in \text{children(root)}$.
  end if
  $\varepsilon_m = \varepsilon_{m-1}/2$
  for all $\ell \in \text{leaves}(\mathcal{T})$ **do**
    $\delta_\ell = \delta_\ell/2$
    if $\hat{\sigma}_\ell < \sigma^{\text{cut}}$ **then**
      Take Bernstein bound:
      $N_{m,\ell} = \left\lceil \frac{1}{\varepsilon_m^2} \left(\sqrt{\hat{\sigma}_\ell^2 (\hat{\sigma}_\ell^2 + 6\varepsilon_m) \log^2(3/\delta_\ell)} + (\hat{\sigma}_\ell^2 + 3\varepsilon_m) \log(3/\delta_\ell) \right) \right\rceil$
      $N_{m,\ell} = \max(N_{m,\ell}, N_{m-1,\ell})$
      Add $N_{m,\ell} - N_{m-1,\ell}$ samples to leaf ℓ .
      Compute $\hat{\mu}_\ell$ and $\hat{\sigma}_\ell$.
      while $\varepsilon_m < \hat{\sigma}_\ell \sqrt{\frac{2\log(3/\delta_\ell)}{N_{m,\ell}}} + \frac{3\log(3/\delta_\ell)}{N_{m,\ell}}$ **do**
        $\delta_\ell = \delta_\ell/2$
        $N_{m,\ell} = 1.468 N_{m,\ell}$
        Add $N_{m,\ell} - N_{m-1,\ell}$ samples to leaf ℓ.
        Compute $\hat{\mu}_\ell$ and $\hat{\sigma}_\ell$.
      end while
    else
      Take Hoeffding bound:
      $N_{m,\ell} = \lceil 1/(2\varepsilon_m^2) \log(2/\delta_\ell) \rceil$
      $N_{m,\ell} = \max(N_{m,\ell}, N_{m-1,\ell})$
      Add $N_{m,\ell} - N_{m-1,\ell}$ samples to leaf ℓ .
      Compute $\hat{\mu}_\ell$ and $\hat{\sigma}_\ell$.
    end if
  end for
  Perform minimax on $\mathcal{T}$ as in Equation 1.3.
  Prune and update $\mathcal{T}$ as in Inequality 2.3.
end for
Recommend $I = \arg\max_{v \in \text{children(root)}} \hat{\mu}_v$.

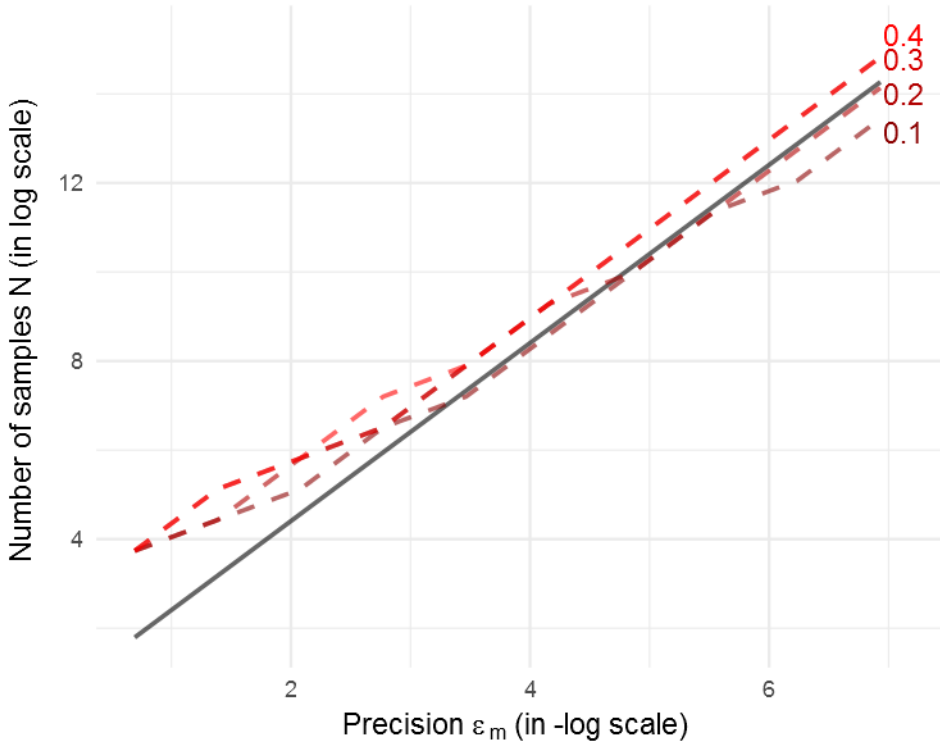


Figure 3.1: Incline in number of samples (in log scale) for logarithmically declining ε_m on a single leaf. The black line depicts the number of samples to satisfy Hoeffding's inequality with error probability $\delta = 0.1$. The dashed lines indicate the Bernstein results from the naive doubling procedure for different known $\mu = \{0.1, 0.2, 0.3, 0.4\}$. Note that (1) for large ε_m (i.e., small $-\log(\varepsilon_m)$ and early epochs m) the naive Bernstein approach always requires more samples to be satisfied and (2) the original FindTopWinner outperforms the naive Bernstein approach for μ's closer to 0.5 even for late epochs.

3.5 Experiments

When just looking at a single leaf, the first thing to test is whether the theoretical bound that has been created is faring better than the original simple Hoeffding bound. For the sake of simplicity, we will first assume that we know σ_ℓ^2 of a leaf ℓ (i.e., drop estimation aspect) and compute the amount of samples that the algorithm would take when using the idea from the Bernstein FindTopWinner versus the original FindTopWinner using the Hoeffding bound. The result is summarised in Figure 3.2.

It is of great interest how the single-leaf advantage will translate into performance on actual trees according to the regime used by Teraoka, Hatano, and Takimoto [3]. This was investigated through comparison with the original FindTopWinner algorithm. See Figure 3.3 for results.

The Bernstein sampling procedure is less likely to ask extremely large N, which comes handy in practice. On average, however, the algorithm did not take less samples before it converged. This can be partially attributed to the loss associated with overshooting N while iteratively checking whether the bound holds. Additionally, as has been shown in Figure 3.1 and Figure 3.2, the advantages of the Bernstein bound unfold during later epochs (i.e. for small ε), which also

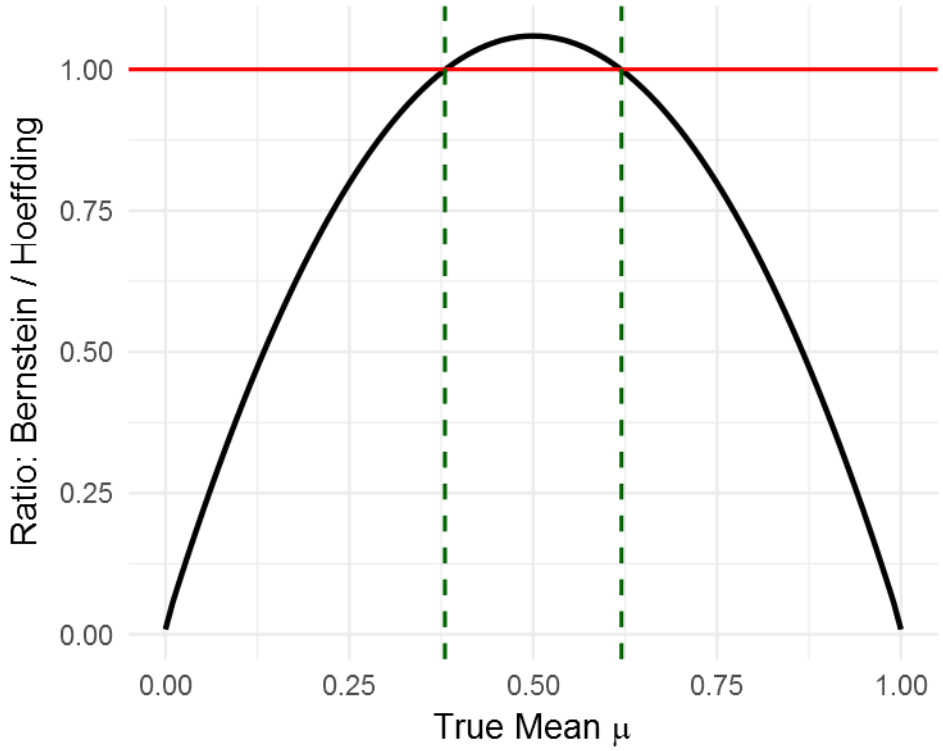


Figure 3.2: Validation of Algorithm 3 on a single leaf ℓ. The y-axis depicts ratio N_B/N_H of total amount of samples taken for all possible known $\mu_\ell = [0, 1]$ until epoch $M = 10$ has been reached (i.e. very small precision $\varepsilon = 2^{-10} \approx \frac{1}{1000}$). Below the red line the refined Bernstein algorithm required less samples. The dashed vertical lines pointing towards a μ_0 and μ_1 indicate the best choice for $\sigma_\ell^{\text{cut}} = \sqrt{\mu_0\mu_1} \approx 0.485$ in this setting. Note that when knowing the true mean μ_ℓ we circumvent mistakes in estimation procedure that would shift the shape upwards, leading to smaller σ_ℓ^{cut}.

explains why there are less "lucky" trials in which an appropriate candidate move for v^* has been identified during an early epoch as it happened with the original Hoeffding-based confidence bound. Indeed, when testing for smaller input ε, the algorithm consistently performs better, which provides further evidence towards an asymptotic advantage. It can be concluded that the Bernstein sampling procedure as it is at this stage does not provide a great advantage for reasonable input ε and only wins asymptotically. The algorithmic setup opened possibilities for many variations of Algorithm 3, however, from which one of them might lead to more satisfactory results. In other words, it is not yet clear at this stage whether a bottleneck was reached or whether algorithmic complications caused underperformance. Therefore the framework should be scrutinized more in the future.

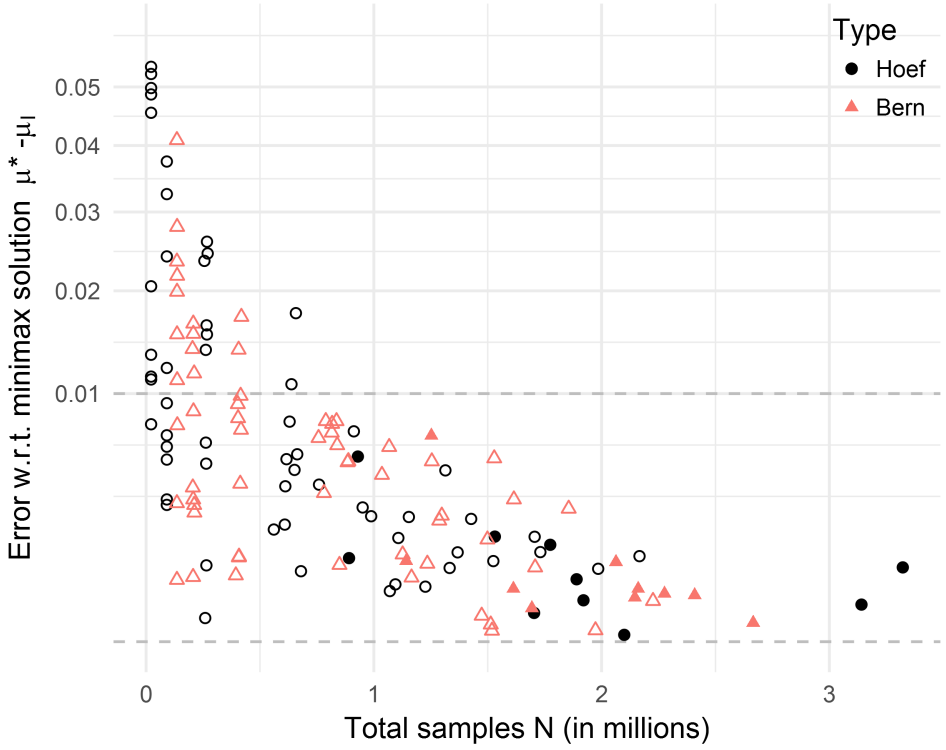


Figure 3.3: Experiments on exactly the same tree as in Figure 2.1 with the same input parameters. Observations correspond to the number of samples taken during an epoch and the deviation $|\mu^* - \mu_I|$. Bernstein observations are depicted as red triangles. Cutoff $\sigma^{\text{cut}} = 0.4$ was chosen for the present experiment. Performance was, on average, not better than for the original FindTopWinner.

Chapter 4

Iterated-Logarithm Confidence Bounds

In the original FindTopWinner Algorithm 1 and its Bernstein-based extension, Algorithm 3, expenses have been made concerning error probability δ, continuously halving it to preserve the overall union bound (see Proposition 1) and counterbalancing for the many tests that are being made. The underlying rationale is that for both Hoeffding's and Bernstein's inequality, δ appears within a $\log(.)$, which alleviates any effects on the computation of N when reducing it by relatively large factors. However, more asymptotically, effects appear and have its price, which was described in more detail in Section 2.2 about the multiple testing problem. To counteract this, ε was halved during each epoch, so that the resulting exponentially spaced intervals for N could be used to estimate means μ and do pruning without splitting δ too often.

4.1 Law of the Iterated Logarithm

A more sophisticated strategy constructs a bound in such a way that the total error probability during all tests is naturally bounded by δ. In other words, we could think about a sequential testing approach which holds true with probability $1-\delta$ at any time when conducting an unbounded amount of tests, whereas at the same time the knowledge about μ increases.

Martingales

It is possible to set up such a bound for a type of sequences of random variables called *martingales*. It is defined as such:

Definition 3. (Martingale [23]). *A sequence of random variables $M_0, M_1, ...$ is called a martingale if, for any time N,*

$$\mathbb{E}[M_{N+1} \mid M_0, M_1, ..., M_N] = M_N. \tag{4.1}$$

Put simply, Definition 3 necessitates that, for a sequence to be a martingale, at any given time within the testing process we cannot draw conclusions from previous observations and need to stick with what we know at the present. A simple example is flipping a fair coin: It does not matter for the next trial how often the coin has been flipped before, the probability of it turning up "heads" will always be $\mu = 0.5$. A special case of martingales is the *Rademacher Random Walk* [24], which describes the successive summation of independent draws from the Rademacher distribution, which has realizations $X = \{-1, 1\}$ and expectation $\mathbb{E}[X] = 0$. The

Law of the Iterated Logarithm (LIL) was first described with the Rademacher Random Walk and states the following:

Theorem 3. (Law of the Iterated Logarithm [24]). *Given a Rademacher Random Walk M_N, we know with probability 1 that*

$$\limsup_{N\to\infty} \frac{|M_N|}{\sqrt{N \log\log N}} = \sqrt{2} \tag{4.2}$$

Finite-Time Bounds

From Equation 4.2 we know that we can construct a zero-error bound for the deviation of a Rademacher Random Walk from its mean μ for $N \to \infty$. This does not have much of a practical value, however, as we deal with finite samples in reality and we obtain samples from the Bernoulli instead of the Rademacher distribution. Balsubramani came up with a way to generalize the LIL to both broader classes of martingales and a limited amount of samples by creating finite-time Iterated-Logarithm concentration inequalities such as Hoeffding's inequality [5].

Theorem 4. (Finite-time iterated logarithm: Uniform Hoeffding bound [5]). *Take a martingale M_N and assume there are constants $\{c_i\}_{i>1}$ such that for all $N \geq 1$, $|M_N - M_{N-1}| \leq c_N$ with probability 1. When fixing a $\delta \in (0,1)$ and defining $N_{min} = \min\{s : \sum_{i=1}^{s} c_i^2 \geq 173 \log(4/\delta)\}$, then, with probability $\geq 1-\delta$, for all $N \geq N_{min}$ simultaneously,*

$$|M_N| \leq \sqrt{3\Big(\sum_{n=1}^{N} c_i^2\Big)\left(2\log\log\left(\frac{3(\sum_{n=1}^{N} c^2)}{2|M_N|}\right) + \log\left(\frac{2}{\delta}\right)\right)} \tag{4.3}$$

Theorem 4 allows us to draw detailed conclusions about the unfolding pattern of the martingale. For every additional sample we can find out how it is bounded with an error probability of at most δ as compared to the traditional Hoeffding bound, where frequent checks require a union bound which inflates δ.

4.2 Implementation

For each leaf, our MCTS paradigm performs successive i.i.d. Bernoulli trials. In order to transform those random variables $X_1, X_2, ... \sim \mathcal{B}(\mu)$ into a martingale, we need to transform the sequence into a centered version by subtracting μ such that

$$M_N = \sum_{n=1}^{N} (X_n - \mu). \tag{4.4}$$

To show that the new sequence is a martingale, consider the following:

$$\begin{aligned} \mathbb{E}[M_{N+1} - M_N | M_0, M_1, ..., M_N] &= \mathbb{E}[X_{N+1} - \mu | M_0, M_1, ..., M_N] \\ &= \mathbb{E}[X_{N+1}] - \mu \\ &= 0, \end{aligned} \tag{4.5}$$

therefore, M is a martingale.

Constants $\{c_i\}_{i>1}$ model the range of possible increments one sample brings to M. From the Bernoulli distribution we know that $\mu \in [0,1]$, which implies that the maximal possible increment $c_N = 1$. However, this only happens in the extreme case when $\mu = \{0,1\}$. In a fair coin example with $\mu = 0.5$, the maximal possible increment would be reduced to $c_N = 0.5$, creating a tighter bound. Since we do not know μ we need to stick with $c_N = 1$ for all $\{c_i\}_{i>1}$. We will elaborate on this decision in Section 4.4. For now, Inequality 4.3 simplifies to

$$|M_N| \leq \sqrt{3N\left(2\log\log\left(\frac{3N}{2|M_N|}\right) + \log\left(\frac{2}{\delta}\right)\right)}. \tag{4.6}$$

To construct a Hoeffding-based Iterated-Logarithm bound for our MCTS paradigm we still have to divide the martingale M_N by the amount of samples N, so that we end up with a bound describing the precision for μ:

$$\frac{|\sum_{n=1}^{N}(X_n - \mu)|}{N} \leq \varepsilon_N = \sqrt{\frac{3}{N}\left(2\log\log\left(\frac{3N}{2|M_N|}\right) + \log\left(\frac{2}{\delta}\right)\right)}. \tag{4.7}$$

An issue that needs to be addressed is that $|M_N|$ appears in the formula for the confidence bound around $|M_N|/N$. To get the tightest confidence interval we are looking for a fixed point where Inequality 4.7 holds with *equality*, i.e. where $\varepsilon_N = |M_N|/N$. Typical methods for finding such a point are the *bisection method* (also: *binary search method*) and the *Newton-Raphson method*. However, in this case a simpler trick to find the optimal value for ε_N is to assume equality and repeatedly plug back in the result into the formula until it converges. See Algorithm 4 for the full procedure.

Algorithm 4 Hoeffding-based Iterated-Logarithm confidence bound.

1: **Input** N, δ, τ : tolerance.
2: **Initialization** $\varepsilon' = 0, \varepsilon_N = 1$
3: **while** $|\varepsilon_N - \varepsilon'| > \tau$ **do**
4: $\quad \varepsilon' = \varepsilon_N$
5: $\quad \varepsilon_N = \sqrt{\frac{3}{N}\left(2\log\log\left(\frac{3}{2\varepsilon'}\right) + \log\left(\frac{2}{\delta}\right)\right)}$
6: **end while**
7: **return** ε_N

Controlled Sampling

In the original FindTopWinner algorithm N was estimated from fixed δ and ε. With the Iterated-Logarithm based estimation, however, we can take advantage of the unlimited amount of estimations we are allowed to make. The most reasonable approach would be to perform all computations of interest (i.e. minimax and pruning) for each added sample. This way, we can make sure that we take exactly the amount of samples we need to identify a candidate move for v^* with the Iterated-Logarithm regime, as compared to a rather crude exponential growth of samples that overshoots the stopping criterion until a candidate move for v^* is identified.

When adding single samples at one leaf at a time, a rule needs to be defined to decide which of the leaves ℓ receives the next sample. A robust strategy would be to investigate the leaf we know the *least* about in terms of ε. This way it is made sure that always the greatest

source of uncertainty is targeted and estimates within the tree are not affected by a large lack of information on a single node.

Node-Dependent Precision

Sampling at the leaf we know least about hints to the fact that ε will vary across different ℓ. In the original FindTopWinner this was not the case, which considerably simplified the updating procedure, as well as the minimax and pruning criterion. Instead, for the present situation we will construct, for each leaf ℓ, lower confidence bound $\mathrm{LCB}_\ell = \max(\hat{\mu}_\ell - \varepsilon_\ell, 0)$ and upper confidence bound $\mathrm{UCB}_\ell = \min(\hat{\mu}_\ell + \varepsilon_\ell, 1)$, where ε_ℓ is the output from Algorithm 4 applied on leaf ℓ.

According to the minimax rule (Equation 1.3), values of children are maximized at even depth and minimized at uneven depth, respectively. Translating this to confidence bounds UCB and LCB, we need to take into account, for each child c, all possible means $\mu_c \in [\mathrm{LCB}_c, \mathrm{UCB}_c]$ and project it to the parent node u in such a way that we obtain the tightest possible bound for u while still being ε, δ-PAC.

For simplicity, consider the even depth scenario (i.e. a *max* node) for now. To define the UCB_u of the parent node u, we need to know how large its value could possibly be. As a first step, we can look at the maximum possible true means μ_c across children, that is, we are interested in the following set of values:

$$\Big\{ \max_{c \in \mathrm{children}(u)} \mu_c \;\Big|\; \forall c \in \mathrm{children}(u) : \mu_c \in \big[\mathrm{LCB}_c, \mathrm{UCB}_c\big] \Big\}. \tag{4.8}$$

Then, to obtain the UCB_u, we simply maximize over the set:

$$\mathrm{UCB}_u = \max \Big\{ \max_{c \in \mathrm{children}(u)} \mu_c \;\Big|\; \forall c \in \mathrm{children}(u) : \mu_c \in \big[\mathrm{LCB}_c, \mathrm{UCB}_c\big] \Big\}. \tag{4.9}$$

Conversely, for the LCB_u we are interested in the smallest possible μ_c of each child. Since we already know from the children intervals $[\mathrm{LCB}_c, \mathrm{UCB}_c]$ that they are ε, δ-PAC, it logically follows that the smallest possible μ_u should not be smaller than the smallest μ_c from any of the intervals, i.e. the minimum can be taken from the set of the maximal values to remain ε, δ-PAC. Thus,

$$\mathrm{LCB}_u = \min \Big\{ \max_{c \in \mathrm{children}(u)} \mu_c \;\Big|\; \forall c \in \mathrm{children}(u) : \mu_c \in \big[\mathrm{LCB}_c, \mathrm{UCB}_c\big] \Big\}. \tag{4.10}$$

The same happens at a *min* node with opposite directions. Hence, simplifying the statements above, we can define the minimax rule for LCBs and UCBs as such:

$$\mathrm{LCB}_u = \begin{cases} \max(\hat{\mu}_\ell - \varepsilon_\ell, 0) & \text{if } u = \ell, \\ \max_c \mathrm{LCB}_c & \text{if } \mathrm{depth}(u) \text{ is even}, \\ \min_c \mathrm{LCB}_c & \text{if } \mathrm{depth}(u) \text{ is odd}, \end{cases} \quad \mathrm{UCB}_u = \begin{cases} \min(\hat{\mu}_\ell + \varepsilon_\ell, 1) & \text{if } u = \ell, \\ \max_c \mathrm{UCB}_c & \text{if } \mathrm{depth}(u) \text{ is even}, \\ \min_c \mathrm{UCB}_c & \text{if } \mathrm{depth}(u) \text{ is odd}. \end{cases} \tag{4.11}$$

Knowing about the confidence bounds of u enables us to define a new pruning criterion. Whenever the minimax has concluded and the interval $[\mathrm{LCB}_c, \mathrm{UCB}_c]$ of child c falls outside of the parent interval $[\mathrm{LCB}_u, \mathrm{UCB}_u]$, we are sure with probability $\geq 1 - \delta$ that $\mu_u > \mu_c$ in the even depth case, and $\mu_u < \mu_c$ in the uneven case. Such an observation qualifies c as a suboptimal node and it should therefore be pruned.

What is still missing at this point is a suitable stopping criterion. We want to find and recommend the best move $v^* \in \text{children(root)}$ by pruning away alternatives for which we believe that their means are smaller than μ_{v^*}. There are two situations in which we can make such a recommendation. Situation 1 occurs when there is only one child v left, so that we can only recommend this child as our candidate for v^*. Situation 2 happens when input precision ε has been reached because enough samples have been added. Since we are interested in knowing μ_v up to input precision ε, rule 2 of an algorithm should be to stop as soon as

$$\text{UCB}_{\text{root}} - \text{LCB}_{\text{root}} < \varepsilon, \tag{4.12}$$

and recommend the best candidate $I = \arg\max_v(\mu_v)$. This implies that $|\mu_I - \mu^*| \leq \varepsilon$ because $LCB_I \geq LCB_{\text{root}}$ and $\mu^* \leq UCB_{\text{root}}$.

4.3 Algorithm: Iterated-Logarithm Sampling

Algorithm 5 combines the implementation steps in the previous section to a fully comprehensive algorithm based on the FindTopWinner rationale [3] and the finite-time version of the Law of the Iterated Logarithm [5]. As compared to the original FindTopWinner, this algorithm adds single samples at a time and evaluates which subtrees to prune away on the fly. Thus, an optimal stopping criterion exists in which the last added sample means that either input precision ε is achieved or that algorithm concludes earlier because all but one candidate move have been pruned away. Earlier pruning further guarantees that no samples are unnecessarily spent on suboptimal leaves.

There are a couple of notes to make about the setup of Algorithm 5. Importantly, N_{min} has been ignored (i.e., set to 0) due to simulations indicating that it does not matter for the per-leaf bound to perform correctly (see the following section for empirical evidence). This can be explained by the fact that, for small N, Inequality 4.7 provides sufficiently large ε (much larger than for the original Hoeffding bound as in Inequality 2.1). As before, the input δ needs to be split across leaves to stay (ε, δ)-PAC. Also, even though adding one sample at a time will provide the best result, in certain situations it may make sense to add more than one sample at a time to considerably speed up computation time, trading off time complexity with sample complexity. The algorithm has been presented in a simplified form to portray the most important aspects. A real implementation should include a recursive function (calling itself during execution) to exploit the nested structure of trees. That way, instead of doing a linear search for the widest leaf each round, information about the widest child can be maintained across the tree, which again speeds up the entire process. The resulting time complexity for locating the destination of the next sample is $\mathcal{O}(bd)$ instead of $\mathcal{O}(L) = \mathcal{O}(b^d)$, where b is the branching factor and d the depth of $\mathcal{T}$.

4.4 Empirical Confidence Bound

Experiments (which are not presented in this work) indicated that Algorithm 5 performs worse when comparing it with the original algorithm from Teraoka, Hatano, and Takimoto [3], even though the algorithmic adaptations would suggest that significant advantages should have been obtained by controlled the sampling and early pruning. When looking more closely into this issue, it appears that throughout experiments the measured error probability of the algorithms discussed so far was much lower than input error probability δ, and *especially* so for the Iterated-Logarithm approach. Generally, when simulating on a full tree, deflation of measured δ is to be expected because of the union bound that is being used over all leaves, which can be viewed

Algorithm 5 Updates in FindTopWinner: Iterated-Logarithm sampling (Hoeffding)

Input $\varepsilon, \delta, \mathcal{T}$.
Initialization $\forall \ell : \delta_\ell = \delta / L$ and $\forall u : \mathrm{UCB}_u = 1, \mathrm{LCB}_u = 0$
while $|\text{children(root)}| > 1$ and $\mathrm{UCB}_{\text{root}} - \mathrm{LCB}_{\text{root}} > \varepsilon$ **do**
    Add 1 sample to widest leaf $\ell = \arg\max_\ell(\mathrm{UCB}_\ell - \mathrm{LCB}_\ell)$.
    $\varepsilon_\ell =$ Algorithm 4
    $\mathrm{UCB}_\ell = \min(\mu_\ell + \varepsilon_\ell, 1)$
    $\mathrm{LCB}_\ell = \max(\mu_\ell - \varepsilon_\ell, 0)$
    for $a \in \text{ancestors}(\ell)$, starting with deepest, **do**
        Perform minimax and pruning:
        if depth(a) is even **then**
            $\mathrm{UCB}_a = \max_{c \in \text{children}(a)} \mathrm{UCB}_c$
            $\mathrm{LCB}_a = \max_{c \in \text{children}(a)} \mathrm{LCB}_c$
            for $c \in \text{children}(a)$ **do**
                if $\mathrm{UCB}_c < \mathrm{LCB}_a$ **then**
                    Prune c with its subtree.
                end if
            end for
        else
            $\mathrm{UCB}_a = \min_{c \in \text{children}(a)} \mathrm{UCB}_c$
            $\mathrm{LCB}_a = \min_{c \in \text{children}(a)} \mathrm{LCB}_c$
            for $c \in \text{children}(a)$ **do**
                if $\mathrm{LCB}_c > \mathrm{UCB}_a$ **then**
                    Prune c with its subtree.
                end if
            end for
        end if
    end for
end while
Recommend $I = \arg\max_{v \in \text{children(root)}} \hat{\mu}_v$.

as a worst-case protection mechanism. Even though improvements might be made here, it is relatively difficult to provide a safe solution that holds for any tree size, therefore it will not be further investigated.

Strikingly, even when only looking at a single leaf, the Iterated-Logarithm sampling approach seems way too conservative. This can happen, since it is based on mathematically derived conclusions which, on the one hand, give solid grounds, but on the other hand might still be improvable. Also, a more obvious source of slack coming from Theorem 4 is the fact that it uses the maximal possible increments $|M_N - M_{N-1}| \geq c_i$ as the leading factor for scaling the confidence bound. For our martingale (see Equation 4.4) such increments can be, depending on how extreme μ is, up to 1, which causes the leading constant to be $3N$. However, this neglects the fact that for more extreme μ such increments are much less likely to happen (e.g., if $\mu = 0.9$ it is expected that 10% of times $|M_N - M_{N-1}| = 0.9$ and 90% of times $|M_N - M_{N-1}| = 0.1$).

Further, when looking more closely at the components of Inequality 4.7, an Iterated-Logarithm component (in the form of $\log\log(.)$) and the original Hoeffding component (in the form of $\log(2/\delta)$) can be distinguished within the square root. Interestingly, this Hoeffding component has been scaled up by a factor of 6, as compared to Hoeffding's Inequality 2.1.

One may wonder how much tighter the bound could be while simultaneously being δ-correct. In the following, an empirically validated bound will be constructed from the Hoeffding-based Iterated-Logarithm bound, minimizing the amount of samples necessary to have the knowledge needed for recommending a candidate for the best move v^*.

Kolmogorov-Smirnov Comparison of Distributions

Since we are interested in being δ-correct, we should look at confidence bounds as a function for obtaining δ. For the original confidence bound from Balsubramani in Theorem 4 that would be

$$\mathbb{P}\left(\forall N : \exp\left(-\frac{M_N^2}{3N} + 2\log\log\frac{3N}{2|M_N|} + \log 2\right) \leq \delta\right) < \delta. \tag{4.13}$$

In order to empirically justify that any newly created confidence bound with such a shape on δ, let's call it B_{emp}, is δ-correct, a comparison bound may be set up that models the case in which the measured error probability is *exactly* δ. In other words, imagine an ideal confidence bound B_{id} where plugging in a δ means that the bound will be incorrect with probability δ instead of probability $< \delta$.

When looking at the cumulative distribution function F_{id} of such a bound that means that we would obtain an *identity line*, i.e. $F_{id}(\delta) = \delta$. Then we can compare F_{id} with the underlying *empirical distribution function* of B_{emp}, F_{emp} respectively. The empirical distribution function belongs to the family of step functions and is defined as follows:

Definition 4. (Empirical distribution function). *Given i.i.d. random variables $X_1, ..., X_N$ the empirical distribution function is*

$$F_N(x) = \frac{1}{N}\sum_{n=1}^{N} I_{[-\infty,x]}(X_n), \tag{4.14}$$

with $I_{[-\infty,x]}(X_n)$ being the indicator function for the condition $X_n \leq x$.

The (non-parametric) Kolmogorov-Smirnov test was developed to find out whether an empirical distribution function corresponds to a given cumulative distribution function by evaluating a test statistic that measures the greatest vertical distance between the two functions [25]. This is not of direct interest in the present setting, however, as we do not want to investigate whether F_{emp} is different from F_{id}. Instead, the goal is to show that, strictly, $F_{emp} < F_{id}$ for any $\delta_0 \in (0, 1)$ we might use. In other words, we need to validate that the steps in F_{emp} do not exceed the worst-case error probability δ.

Simulation Setup

In order to validate a confidence bound, martingale sequences M_N as in Equation 4.4 have to be created and subsequently investigated on whether B_{emp} was performing accurately by being δ-correct, ideally for all δ.

For now assume that S simulations will be made for martingales of some finite length N_{max}. Plugging in a single martingale from the simulation sequence $s \in \{1, ..., S\}$ for an $N \in \{1, ..., N_{max}\}$ provides us with a result $\alpha_{s,N}$ for the term in the left of the probability statement in Inequality 4.13. Since it has to hold for all N, we need to take $\alpha_s^* = \max_N \alpha_{s,N}$ and check whether the inequality holds. Then, the confidence bound B_{emp} is valid if

$$\forall s, \delta : \ \mathbb{P}(\alpha_s^* \leq \delta) < \delta. \tag{4.15}$$

Unfortunately, such an empirical approach is naturally restricted for two reasons:

1. Fixing N_{max} implies that the coverage of ε will be affected, since Algorithm 5 appends new observations to M_N to reduce ε. An upper limit for M_N thus affects the upper limit of a guarantee we can make for ε.
2. A finite number of simulations S implies that there will be a degree of uncertainty. This is complicated by the fact that it is generally hard to observe very small probabilities. For large trees the δ_ℓ that are being used on a single leaf ℓ are indeed typically very small due to the union bound across leaves (for the FindTopWinner benchmark experiment we have: $\delta_\ell = \delta/L = \frac{1}{10{,}000}$). This forces us to increase S to a large number to establish certainty for such very small δ.

To cover most reasonable tree search settings, it was aimed to have certainty up to $\delta_\ell \geq \frac{1}{100{,}000}$ and $\varepsilon \geq \frac{1}{100}$ (i.e. knowing μ up to 0.01 as in the FindTopWinner experiment). It is difficult to estimate how many simulations are needed because the distribution of α^* is unknown (density plots pointed towards a complex distribution with many local maxima). To make a good case a total of 24,000,000 simulations were run. Under the (wrong) assumption of a uniform distribution this would imply that we should be very certain about the behavior for any $\delta \geq \frac{1}{100{,}000}$ since there will be a sufficient amount of simulations to spot enough occurrences of such probabilities. N_{max} was adapted based on the confidence bound's sample requirements to achieve precision $\varepsilon \geq \frac{1}{100}$.

To validate a confidence bound, all corresponding $\alpha_{s,N}$ were explored for all N for all s and subsequently aggregated to one empirical distribution function to compare it with the identity line, that is, the cumulative distribution function where

$$\forall \delta : \ \mathbb{P}(\alpha \leq \delta) = \delta. \tag{4.16}$$

The result with the best confidence bound that was found is depicted in Figure 4.1.

Best Confidence Bound

To be able to apply the resulting confidence bound to any μ later, the case with maximal possible variance was chosen for the simulations of the martingale sequences, i.e. $\mu_s = 0.5$. Starting with Inequality 4.13, outcomes from manipulation of factors were investigated by trial and error to study the change in behavior of the empirical confidence bound as compared to the identity line. Next to trial and error, theoretical guidance could be found from the original Hoeffding's inequality (see Theorem 1). Importantly, the log(.) term in the square root is multiplied with $\frac{1}{2N}$, while the leading multiplier in Balsubramani's confidence bound for the finite-time version of the Law of the Iterated Logarithm in ε-form is $\frac{3}{N}$ (see Inequality 4.7). Since the Iterated-Logarithm aspects are summarised in a $2 \log\log(.)$ term, it seems as if the leading muliplier $\frac{3}{N}$ is overly conservative and it might be possible to reduce its size.

Indeed, when creating a new Iterated-Logarithm confidence bound in which the leading multiplier was replaced with the traditional Hoeffding multiplier of $\frac{1}{2N}$, the result turned out to be δ-correct for the δ that were observed. Thus, in ε form, the following empirical confidence bound is proposed:

$$\varepsilon_N = \sqrt{\frac{1}{2N}\left(2 \log\log\left(\frac{3}{2\varepsilon_N}\right) + \log\left(\frac{2}{\delta}\right)\right)}. \tag{4.17}$$

For the simulations the maximum sequence length N_{max} to obtain $\varepsilon_N = 0.01$ was investigated with Algorithm 4 and found to be $N_{max} = 88,659$.

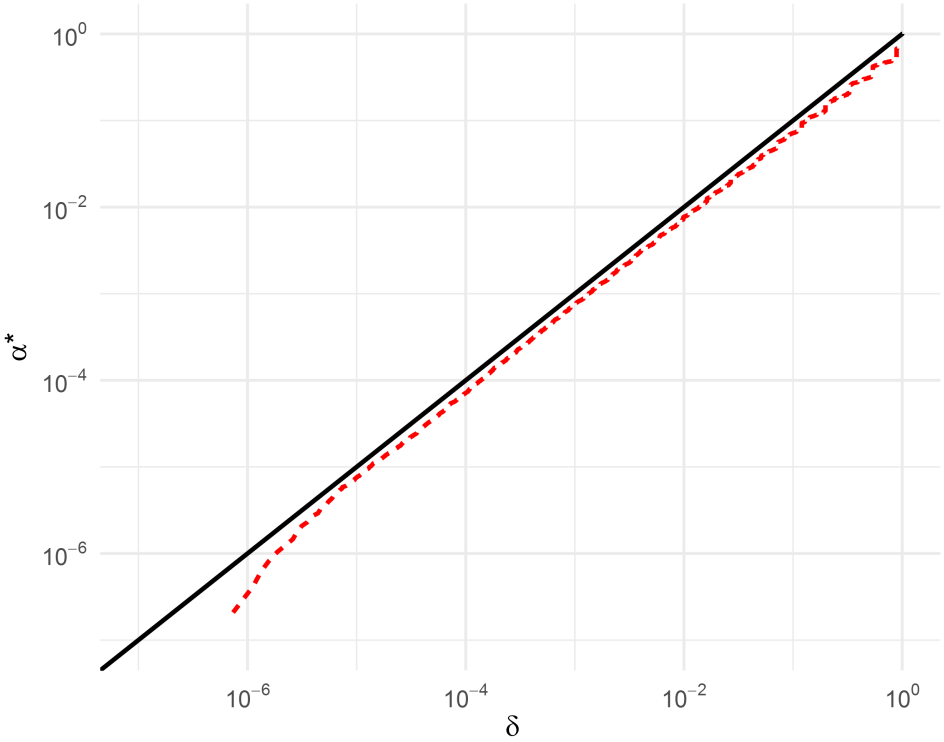


Figure 4.1: Empirical distribution function F_{emp} derived from Inequality 4.17 (dashed red line) as compared to (theoretical) cumulative distribution function F_{id} (solid black). As long as $\alpha^* < \delta$ (i.e. below the black line) the confidence bound makes less errors than what is allowed in order to be δ-correct. The log scale is used on both axes to highlight small δs. Note that empirical guarantee is only suggested up to about $\delta = 10^{-6}$ (e.g., a game tree with input $\delta = 0.1$ and 100,000 leaves), since observations for smaller δ were too rare to draw conclusions from them.

4.5 Experiments

When applying the empirically derived Iterated-Logarithm confidence bound from Inequality 4.17 to Algorithm 5, one may wonder what the general advantage on a single leaf might be (without the benefits from early pruning). As outlined in the previous section we kept the original Hoeffding multiplier (such that the Hoeffding component remains fully intact) in the confidence bound, adding the iterated-logarithm component in the form of $2 \log \log(.)$. The empirically derived confidence bound should therefore always require more samples than the original Hoeffding confidence bound. However, when it comes to which δ will be plugged into the confidence bounds, the iterated-logarithm confidence bound fares better since there is no union bound across successive tests and hence no splitting of δ over epochs. See Figure 4.2 for a comparison.

The single-leaf results look promising, since there is already an advantage from the removal of the union bound. Additionally, the early pruning and the optimal stopping criterion should improve on that result, which was tested in the FindTopWinner benchmark experiment. Indeed, the new confidence bound performed much better than traditional approaches, effectively halving the amount of samples required to be (ε, δ)-PAC for $\varepsilon = 0.01$ and $\delta = 0.1$ (see Figure 4.3).

The behavior of F_{emp} as compared to the F_{id} further suggest good performance for smaller

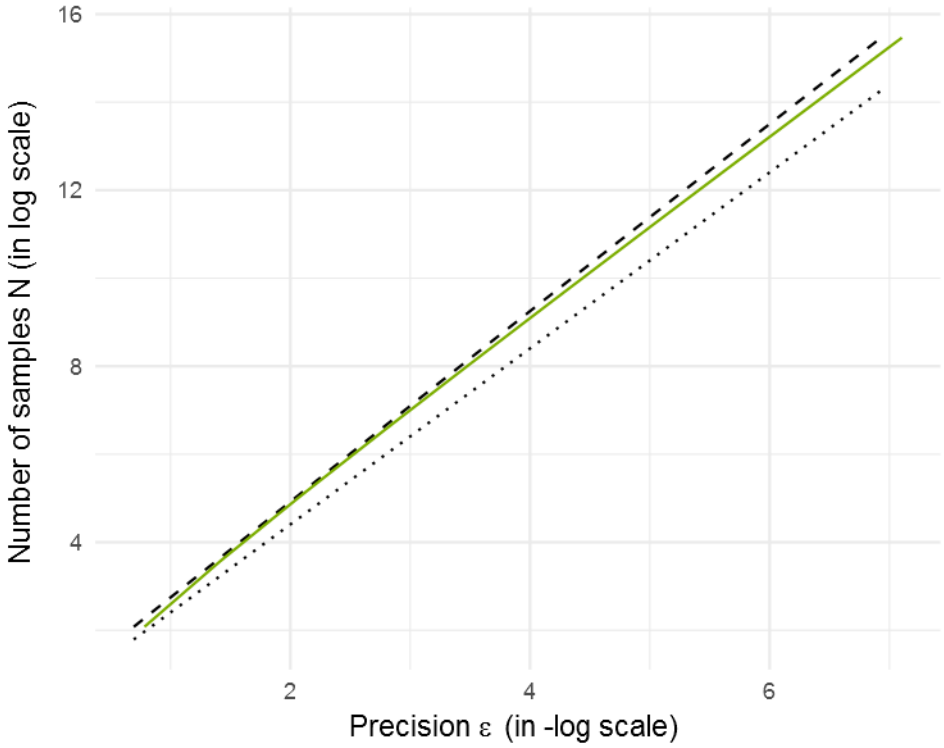


Figure 4.2: Incline in number of samples (in log scale) for logarithmically declining ε on a single leaf. The solid green line depicts the amount of samples the empirical Iterated-Logarithm confidence bound needs to reach a certain ε, while the dashed black line describes the amount of samples the original FindTopWinner requires. Note that even though the original Hoeffding confidence bound would be tighter for a single test, successive tests let δ decline and make the new approach succeed. As an alternative, consider the dotted black line, which models a scenario under which we (wrongly) neglect the union bound across successive tests for the original FindTopWinner. Its distance to the solid green line is the slack induced by the newly added iterated logarithm component to account for the missing union bound.

δ_ℓ and ε. Thus, a good solution would be to do simulations before applying it and to make a decision about its applicability based on the result. From above it can be inferred that the empirical confidence bound qualifies for most realistic tree search tasks. The loss of generality (i.e., the fact that we do not have a theoretical base to prove that it truly works for all parameter values) translated into a huge gain when it comes to the total amount of samples taken for any observed setting.

Considerations about Hoeffding's Inequality

It is not yet obvious whether the improvement in performance stems from resolving conservative aspects from Theorem 4 or whether, simultaneously, the original Hoeffding's Inequality 2.1 has been loosened. There are two reasons why this should not be the case. First, as has been outlined in Section 4.4, the new confidence bound incorporates the original Hoeffding's inequality term.

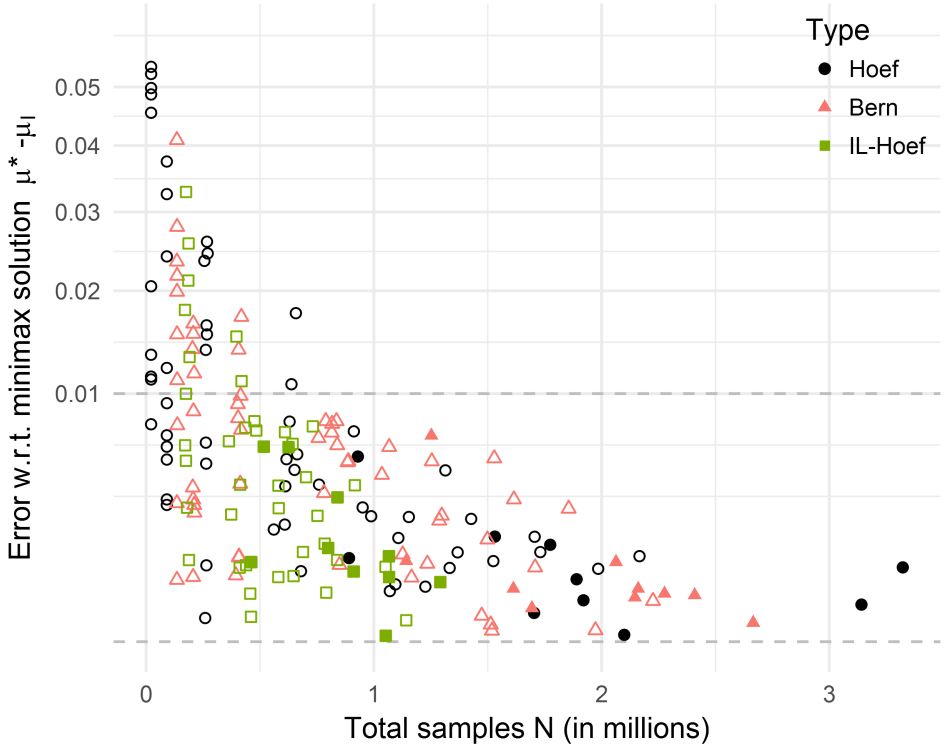


Figure 4.3: Experiments from Algorithm 5 using the empirical Hoeffding-Based Iterated-Logarithm confidence bound (depicted as green boxes) as compared to the original approaches (i.e., black dots and red triangles). Intermediate steps (□) were added whenever the algorithm crossed a ε_m as in the updating regime from the original FindTopWinner (Algorithm 1).

Second, it is only marginally possible to improve on Hoeffding's inequality. Consider a constant k with which we would like to scale

$$|\mu - \hat{\mu}| \leq k\sqrt{\frac{\log(2/\delta)}{2N}}, \tag{4.18}$$

such that the confidence bound holds for all δ.

When approximating many Bernoulli trials with a Standard Normal distribution with distribution function Φ and only considering the left tail for simplicity, then, in order to hold Hoeffding's inequality, we need to satisfy

$$\Phi\left(-k\sqrt{2\log\frac{2}{\delta}}\right) < \delta. \tag{4.19}$$

Using Mill's ratio $(1-\Phi)/\phi$, with ϕ being the Standard Normal density function (i.e. Φ'), it can be shown by continued fraction expansion [26] that

$$\Phi\left(-k\sqrt{2\log\frac{2}{\delta}}\right) \approx \frac{\phi\left(k\sqrt{2\log\frac{2}{\delta}}\right)}{k\sqrt{2\log\frac{2}{\delta}}} = \frac{\exp(-k^2\log\frac{2}{\delta})}{\sqrt{2\pi}k\sqrt{2\log\frac{2}{\delta}}} = \frac{\delta^{k^2}}{2k\sqrt{\pi\log\frac{2}{\delta}}}. \tag{4.20}$$

Combining Inequality 4.19 and 4.20 we would hence need to have

$$\frac{\delta^{k^2}}{2k\sqrt{\pi \log \frac{2}{\delta}}} < \delta. \tag{4.21}$$

From above it follows that for all $k < 1$, Hoeffding's inequality does not hold anymore for all δ. More precisely, for $\delta \to 0$ the numerator grows at a higher order (since powers $k^2 < 1$ make the probability become larger), causing the result to exceed δ on the right-hand side, and hence, violating the inequality. Therefore, it is not recommendable to take a smaller preceding constant than was originally proposed by Hoeffding, and by doing it anyways, δ-incorrect observations should exist.

4.6 Bernstein-Based Confidence Bound

In the previous chapter, Bernstein's inequality was used to seek an advantage over the standard Hoeffding bound by including $\sigma^2 = \mathrm{Var}[X]$. It seems reasonable to assume that similar adaptations could be made for the Iterated-Logarithm bound outlined in this chapter, with some additional advantages through controlled sampling and early pruning. Indeed, Balsubramani has worked out such a bound [5] and recently Balsubramani and Ramdas have submitted a paper in which they established a refined version [6]. The bound is defined as follows:

Theorem 5. (Finite-time Iterated Logarithm: Uniform Bernstein bound [5]). *Take martingale M_N from Equation 4.4 in order to satisfy $Var[M_N] = N\sigma^2$ and fix a $\delta \in (0, 1)$. Then, with probability $\geq 1 - \delta$, for all $N \geq 1$ simultaneously,*

$$|M_N| < C_0(\delta) + \sqrt{2C_1 N\sigma^2 [\log\log]_+(N\sigma^2) + C_1 N\sigma^2 \log\left(\frac{4}{\delta}\right)}, \tag{4.22}$$

with

$$C_0(\delta) = 3(e-2)e^2 + 2(1+\sqrt{1/3})\log(8/\delta) \tag{4.23}$$

and

$$C_1 = 6(e-2). \tag{4.24}$$

(Where $[\log\log]_+ = \log\max[\log\max(.,1),1]$, forcing the outcome to be ≥ 0). There are some differences to the Hoeffding-based adaptation as in Theorem 4. No minimum amount of samples $N \geq N_{min}$ is required and a group of scaling constants are needed to satisfy the inequality.

4.7 Implementation

A striking advantage of the controlled sampling approach outlined in Section 4.2 is that we are not forced anymore to try out whether an N lies within a bound where ε has been fixed (see Section 3.1 for comparison). Instead, we can efficiently add single samples and calculate ε from the formula above. Further, from the parametrization of the Bernoulli distribution we know that σ^2 is related to μ in a direct function ($\sigma^2 = \mu(1-\mu)$) and does not have to be independently estimated.

To adapt Inequality 4.22 to our framework we divide both sides by N and rewrite it so that we obtain ε_N:

$$|\mu - \hat{\mu}| < \varepsilon_N = \frac{C_0(\delta)}{N} + \sqrt{\frac{C_1\sigma^2}{N}\left(2[\log\log]_+(N\sigma^2) + \log\left(\frac{4}{\delta}\right)\right)}. \tag{4.25}$$

Since we only know $\hat{\mu}$ instead of μ (and therefore we do not know σ^2), we need to investigate for which values for μ the bound is as tight as possible while still satisfying Inequality 4.25. By plugging in potential values μ_0 and σ_0^2 there are situations where the bound is true under $\hat{\mu}$ or not true. Thus, similarly to the Hoeffding bound as in Algorithm 4, we need an iterative procedure to calibrate ε. As with the Hoeffding-based bound from Inequality 4.3, when looking at the inequality as a function of μ, it is roughly shaped like a parabola due to the square root. This points to the fact that there should be two crossover points, i.e. two optimal values for μ left and right from $\hat{\mu}$. However, in contrast to the Hoeffding-based bound from Inequality 4.3, the deviation is not symmetric anymore because of the inclusion of $\hat{\sigma}$ which becomes smaller for extreme μ, which means that instead of one ε there will be two distinct deviations ε^- (i.e. $\text{LCB} = \mu - \varepsilon^-$) and ε^+ ($\text{UCB} = \mu + \varepsilon^+$).

This time, we cannot directly compute the deviations by repetitively plugging in the result from the last iteration, but we need to search them in the parameter space. A robust solution here is the bisection method, which first defines the boundaries of the parameter space and then iteratively narrows the possible values a parameter can take. It does this by trying the center (i.e. taking the mean) between the boundaries, calculating whether it was over- or undershooting, and depending on the result continues searching in the relevant subspace (right or left from center) where the true value must lie, consequently halving the parameter space over successive iterations. The precise implementation can be found in Algorithm 6. Consider the case in which deviation ε^- is being searched. Since we know that $\text{LCB} \leq \hat{\mu}$ we can define the interval $I^- = [0, \hat{\mu}]$ and start by plugging in the center of the interval $\mu_0 = \frac{1}{2}(\max I^- + \min I^-)$ into Inequality 4.25. The parameter space is then updated depending on whether the inequality is satisfied or not. If it is, ε^- is too small and the interval of possible μ is updated to only contain μ_0 that lead to larger ε^-, and vice versa respectively for too large ε^-. As soon as the width of I^- is becoming smaller than a tolerance level τ, a good approximation for ε^- is found. This is done in the same way for ε^+ and $I^+ = [\hat{\mu}, 1]$.

With probability $1 - \delta$ Inequality 4.25 holds. On that event, since the UCB is the largest choice for μ such that Inequality 4.25 holds, we find $\mu \leq \text{UCB}$. Analogously, also $\mu \geq \text{LCB}$. Thus, overall,

$$\mathbb{P}\Big(\forall N : \mu \in [\text{LCB}_N, \text{UCB}_N]\Big) \geq 1 - \delta. \tag{4.26}$$

Considerations about Empirical Confidence Bound

In Section 4.4 an empirical confidence bound was proposed for the Hoeffding-based Iterated-Logarithm alternative. It seems reasonable to do the same for the Bernstein-based approach, and even though it *should* be possible to come up with compelling empirical evidence, it turned out to be of a more complex matter. The Bernstein confidence bound incorporates variance σ^2. For that reason, simulations would need to be conducted for martingales of different shape, that is, an approximation through a grid with martingales that are based on different σ^2 (and hence μ) would be required. As pointed out in Section 4.4 the empirical error probabilities α followed a complex distribution, and since the Hoeffding and Bernstein-based approaches are similar in shape, the same problem should appear for the Bernstein-based confidence bound. It is difficult to make any assumptions on its interaction with μ, thus there might be many local maxima across μ too. A grid would have to be defined that is tight enough to cover enough cases to make

Algorithm 6 Bernstein-based Iterated-Logarithm confidence bounds.

1: **Input** $\hat{\mu}, N, \delta, \tau$: tolerance.
2: **Initialization** $I^+ = [\hat{\mu}, 1], I^- = [0, \hat{\mu}]$
3:
4: *Calibrate upper bound:*
5: **while** $\max I^+ - \min I^+ > \tau$ **do**
6: $\mu_0 = \frac{\max I^+ + \min I^+}{2}$
7: $\sigma_0^2 = \mu_0(1-\mu_0)$
8: **if** $\mu_0 - \hat{\mu} < \frac{C_0(\delta)}{N} + \sqrt{\frac{C_1\sigma_0^2}{N}\left(2[\log\log]_+(N\sigma_0^2) + \log\left(\frac{4}{\delta}\right)\right)}$ **then**
9: $\min I^+ = \mu_0$
10: **else**
11: $\max I^+ = \mu_0$
12: **end if**
13: **end while**
14: $\mathrm{UCB} = \max I^+$
15:
16: *Calibrate lower bound:*
17: **while** $\max I^- - \min I^- > \tau$ **do**
18: $\mu_0 = \frac{\max I^- + \min I^-}{2}$
19: $\sigma_0^2 = \mu_0(1-\mu_0)$
20: **if** $\hat{\mu} - \mu_0 < \frac{C_0(\delta)}{N} + \sqrt{\frac{C_1\sigma_0^2}{N}\left(2[\log\log]_+(N\sigma_0^2) + \log\left(\frac{4}{\delta}\right)\right)}$ **then**
21: $\max I^+ = \mu_0$
22: **else**
23: $\min I^+ = \mu_0$
24: **end if**
25: **end while**
26: $\mathrm{LCB} = \min I^-$
27:
28: **return** $[\mathrm{LCB}, \mathrm{UCB}]$

rough assumptions about its behavior and at the same time enough simulations are required to cover a reasonable parameter space for input δ and ε. Due to the associated computational requirements, such a bound will not be further considered in the present work.

4.8 Algorithm: Bernstein

The full implementation, Algorithm 7, is very similar to the implementation for the Hoeffding approach, Algorithm 5 (lines 5-7 replaced). It will be displayed here in full for the sake of completeness.

4.9 Experiments

The Bernstein-based Iterated-Logarithm approach did not turn out to require less samples than the original FindTopWinner. Single-leaf results indicated that, similarly to the original Bernstein

Algorithm 7 Updates in FindTopWinner: Iterated-Logarithm sampling (Bernstein)

Input $\varepsilon, \delta, \mathcal{T}$.
Initialization $\forall \ell : \delta_\ell = \delta/L$ and $\forall u : \mathrm{UCB}_u = 1, \mathrm{LCB}_u = 0$
while $|\text{children(root)}| > 1$ and $\mathrm{UCB}_{\text{root}} - \mathrm{LCB}_{\text{root}} > \varepsilon$ **do**
    Add 1 sample to widest leaf $\ell = \arg\max_\ell(\mathrm{UCB}_\ell - \mathrm{LCB}_\ell)$.
    $[\mathrm{LCB}_\ell, \mathrm{UCB}_\ell]$ = Algorithm 6
    for $a \in$ ancestors(ℓ), starting with deepest, **do**
        Perform minimax and pruning:
        if depth(a) is even **then**
            $\mathrm{UCB}_a = \max_{c \in \text{children}(a)} \mathrm{UCB}_c$
            $\mathrm{LCB}_a = \max_{c \in \text{children}(a)} \mathrm{LCB}_c$
            for $c \in$ children(a) **do**
                if $\mathrm{UCB}_c < \mathrm{LCB}_a$ **then**
                    Prune c with its subtree.
                end if
            end for
        else
            $\mathrm{UCB}_a = \min_{c \in \text{children}(a)} \mathrm{UCB}_c$
            $\mathrm{LCB}_a = \min_{c \in \text{children}(a)} \mathrm{LCB}_c$
            for $c \in$ children(a) **do**
                if $\mathrm{LCB}_c > \mathrm{UCB}_a$ **then**
                    Prune c with its subtree.
                end if
            end for
        end if
    end for
end while
Recommend $I = \arg\max_{v \in \text{children(root)}} \hat{\mu}_v$.

version, they have a large offset and only catch up asymptotically (see Figure 4.4).

Similarly, as expected there were no benefits over the original FindTopWinner on large trees. The results for the benchmark experiment are depicted in Figure 4.5. It seems reasonable to assume that, as with the (non-empirical) Hoeffding-based Iterated-Logarithm version, the confidence bound is too conservative.

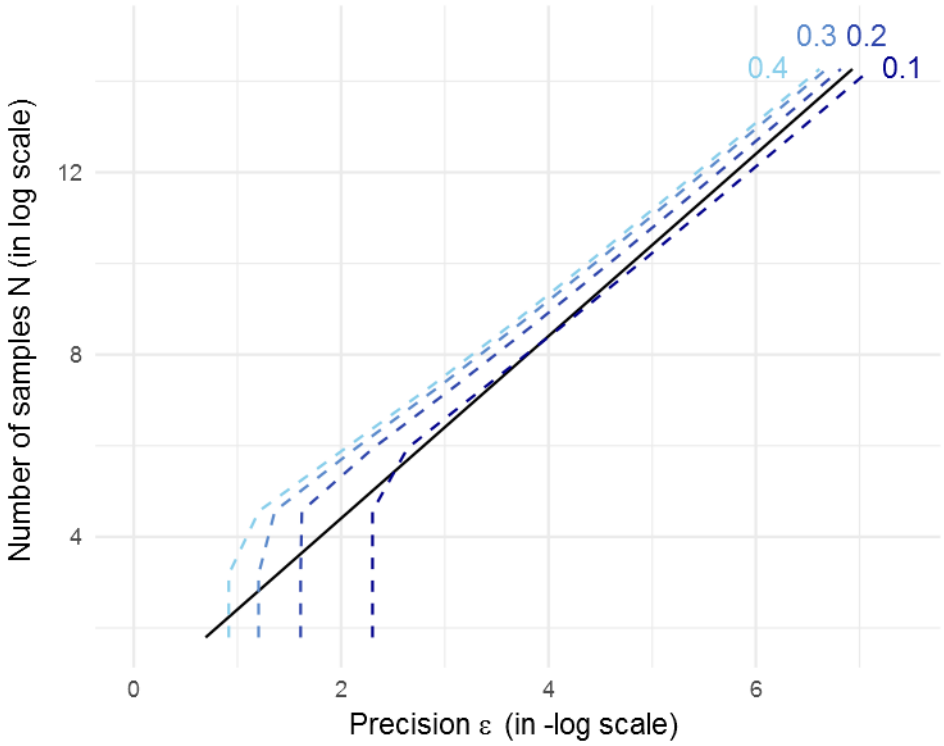


Figure 4.4: Incline in number of samples for lower-bound ε^- on a single leaf. The dashed blue lines depict the amount of samples the Bernstein-based Iterated-Logarithm confidence bounds need to reach a certain lower-bound ε^- for different $\mu = \{0.1, 0.2, 0.3, 0.4\}$. The solid black line shows the original FindTopWinner Algorithm 1. (Note that the dashed blue lines bend sharply for large ε^- (i.e. small $-\log \varepsilon^-$) because ε^- was computed post-hoc from Algorithm 6 (i.e., $\mu - \text{LCB}$). For small N, $\text{LCB} = 0$ and hence the resulting lower-bound precision never exceeds μ.)

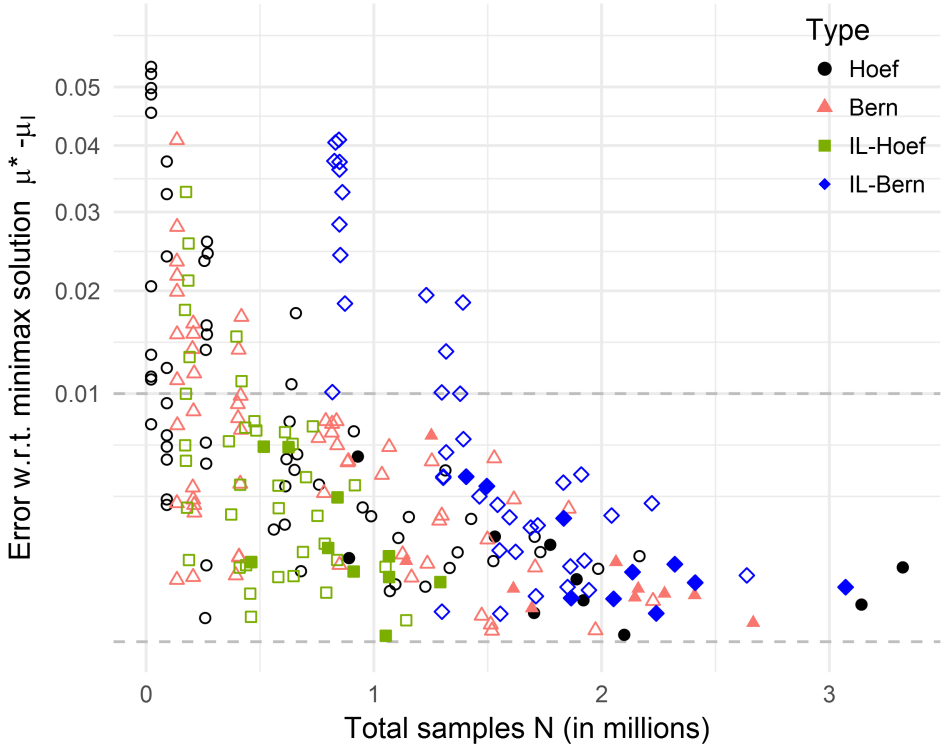


Figure 4.5: Experiments from theoretical Bernstein-Based Iterated-Logarithm bound (depicted as blue diamonds) as compared to the original approaches (black dots and red triangles) and the empirical Hoeffding-based Iterated-Logarithm bound (green boxes). The performance is worse than for the original FindTopWinner Algorithm 1, requiring more samples both during intermediate steps and for convergence.

Chapter 5

Discussion

The previous two chapters were dedicated towards finding a way to reduce sample complexity during MCTS by building on top of the framework proposed by Teraoka, Hatano, and Takimoto in the FindTopWinner algorithm (based on Hoeffding's inequality) [3]. More precisely, three fully functional algorithms, Algorithms 3, 5, and 7, were proposed as potential improvements. Their average performance over many runs on the benchmark experimental setting are summarised in Table 5.1.

Table 5.1: Summary of FindTopWinner benchmark experiments. The empirical Hoeffding-based Iterated-Logarithm bound required least samples. Empirical deviation from μ^* was always below input $\varepsilon = 0.01$.

	N (in millions)	$\lvert\mu^* - \mu_I\rvert$
Hoeffding	1.94	0.0013
Bernstein	1.99	0.0010
IL-Hoeffding (emp.)	0.95	0.0015
IL-Bernstein	2.03	0.0020

Clearly, Algorithm 5, when using the empirically validated Iterated-Logarithm confidence bound (see Inequality 4.17) for Hoeffding's inequality, outperformed all other approaches for the present setting, requiring less than half of the amount of samples than the original (and second-best) Algorithm 1. Single-leaf studies (see Figure 4.2) revealed that this was partially due to savings in error probability δ due to the Law of the Iterated Logarithm [24]. More precisely, input δ was only split across leaves ($\delta_\ell = \delta/L$), whereas in the original FindTopWinner δ was split both across leaves and, in an exponential fashion, across epochs ($\delta_{m,\ell} = \delta/(2^m L)$). Further, early pruning, which happened precisely when the addition of one sample suggested that a node u became suboptimal, reduced the amount of samples spent on u. Finally, the optimal stopping criterion was similarly affected, stopping precisely when the last sample suggested the result to be ε-close (or when early pruning left only one child at the root). For any of these aspects, there is compelling evidence that Algorithm 5 is better in terms of sample complexity.

However, please note that in the present research we optimized for sample complexity. Time complexity was negatively affected by additional computational factors being put on the original method used in the FindTopWinner, especially by introducing an iterative search algorithm to compute precision ε from the confidence bound (see Algorithm 4).

The empirically derived confidence bound points towards some general advantages in sequential testing procedures when using confidence bounds based on the Law of the Iterated Logarithm.

A more powerful future version could be mathematically derived, removing the parameter restrictions that had to be made in the present work. If that would be the case, such confidence bounds would outperform other sequential strategies very quickly. Assume that you want to know the expected probability of a Bernoulli distribution $\mathcal{B}(\mu)$ and you are adding samples (following any policy) until you are sure about the true value of μ up to precision $\varepsilon = 0.1$. Then, the empirically derived confidence bound assures that, as soon as you have 8 or more epochs of adding samples and checking for ε-proximity, it will *always* outperform a confidence bound based on Hoeffding's inequality and using a union bound over epochs, since single-epoch δs will become too small to outperform the empirical Hoeffding-based confidence bound. This scales with ε, i.e. the same holds for $\varepsilon = 0.01$ and 26 or more epochs. Next to MCTS, this could prove to be relevant for any setting in which Upper Confidence Bounds and martingale modelling applies, for instance many multi-armed bandit problems that are based on successive Bernoulli trials.

The two Bernstein-based Algorithms 3 and 7 fared worse than the original, which was unexpected. For the original Bernstein approach in Chapter 3 this was very likely due to the doubling procedure which caused losses both in δ and in overshooting N, yielding advantages only asymptotically. Similarly, the (theoretical) Bernstein-based Iterated-Logarithm approach was largely affected by suboptimal performance for relatively large ε (which could be observed at the steps before Algorithm 7 converges in Figure 3.3). When running the benchmark experiment with input precision $\varepsilon < 0.01$, the Bernstein-based Iterated-Logarithm approach drifted closer towards scores of the Hoeffding-Based Iterated-Logarithm approach with the empirical confidence bound, supporting the idea that asymptotical advantages can be found for very small ε, while the Hoeffding-based approach is better for tasks that require less precision.

In the present work an empirically-validated Bernstein confidence bound was outlined, but not further considered due to the lack of computational feasibility. However, simulations suggested that, similarly to the Hoeffding-based counterpart, great improvements might be made. A speculation about a better confidence bound was to replace $C_1 = 6(e-2)$ (Inequality 4.24) with the constant 2, since this would neither violate the original Bernstein's Inequality 3.3, nor the Law of the Iterated Logarithm (Equation 4.2). Additional research and simulations on such a bound is therefore endorsed.

Confidence bound algorithms are not the only possible way to solve the MCTS problem. Other versions have been proposed, such as using *hierarchical bandits*, where each level in the tree is regarded as its own multi-armed bandit problem [10]. Generally, many methods from the bandit setting could turn out fruitful for MCTS. For instance, a very efficient, yet rather little understood heuristic called *Thompson sampling* could be applied to game trees in a way that much lower sample complexity is achieved by using the probability of an arm to be the best alternative as its likelihood for being pulled next [27, 28].

Confidence bounds for our current framework might be improvable too. For instance, information-theoretic confidence bounds based on the *Kullback-Leibler Divergence* for the Bernoulli distribution have been used successfully in bandit settings [29, 14]. Such a bound would incorporate the variance of a Bernoulli distribution and might thus resolve the deficiencies from Bernstein-based confidence bounds.

Parameter Recommendations

The factors outlined above suggest that the Bernstein-based Algorithms 3 and 7 are recommended when it is very important to be precise (i.e. $\varepsilon < 0.01$). Otherwise, Hoeffding-based algorithms should be used. Also, since the Bernstein-based approach greatly reduces the required amount of samples for extreme μ, the specific properties of a game tree $\mathcal{T}$ may matter a lot. For our experiments, a uniform distribution generated the true means to be estimated, which is clearly

not the case for real settings, where complex functions determine the underlying parameter of the Bernoulli distribution. A Bernstein-based approach might be recommended when those parameters are expected to be extreme.

The confidence bound used in the best-performing Algorithm 5 was derived in Section 4.4 and has been empirically validated roughly for $\delta_\ell \geq 10^{-6}$ and $\varepsilon > 0.01$. Since many martingales had to be simulated to obtain very small error probabilities δ, the simulation results were mostly restricted by hardware. There were no indicators that the δ-correctness would cease for smaller δ, hence, it is strongly recommended to use this algorithm even when the experimental setup would fall outside of the proposed parameter space. For instance, this could be done through ad-hoc simulations on a game tree $\mathcal{T}$ that models the real architecture under investigation.

Conclusion

The present work extended the research on confidence bound algorithms for game trees by introducing a sequential testing method that outperformed traditional approaches such as the FindTopWinner algorithm by Teraoka, Hatano, and Takimoto [3], at least for a relatively large guaranteed parameter subspace.

Coming back to the introductory example dilemma, we have gotten familiar with a framework that could aid with the difficult decision-making process on which type of smartphone to produce. As a CEO, you could create a game tree by constructing a grid of possible end-user prices for each of the potential products. By running one of the algorithms and sequentially sampling customer opinions, you will end up knowing the probabilites of buys ε-close with error probability $< \delta$. When including a minor adaptation on the algorithm such that the minimax method and pruning mechanism can handle expected value computations (product-specific profit range times the probability of buys), the algorithm will make sure to find the product for you that will effectively maximize your minimal revenue.

Bibliography

[1] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Knightcap: a chess program that learns by combining td(λ) with game tree search. *arXiv preprint cs/9901002*, 1999.

[2] Herbert A. Simon and Jonathan Schaeffer. The game of chess. *Handbook of game theory with economic applications*, 1:1–17, 1992.

[3] Kazuki Teraoka, Kohei Hatano, and Eiji Takimoto. Efficient sampling method for Monte Carlo tree search problem. *IEICE TRANSACTIONS on Information and Systems*, 97(3):392–398, 2014.

[4] Volodymyr Mnih, Csaba Szepesvári, and Jean-Yves Audibert. Empirical Bernstein stopping. In *Proceedings of the 25th international conference on Machine learning*, pages 672–679. ACM, 2008.

[5] Akshay Balsubramani. Sharp finite-time Iterated-Logarithm martingale concentration. *arXiv preprint arXiv:1405.2639*, 2014.

[6] Akshay Balsubramani and Aaditya Ramdas. Sequential nonparametric testing with the Law of the Iterated Logarithm. *arXiv preprint arXiv:1506.03486*, 2015.

[7] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with Deep Neural Networks and tree search. *Nature*, 529(7587):484–489, 2016.

[8] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L. Lewis, and Xiaoshi Wang. Deep Learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems*, pages 3338–3346, 2014.

[9] Diego Perez, Edward J. Powley, Daniel Whitehouse, Philipp Rohlfshagen, Spyridon Samothrakis, Peter I. Cowling, and Simon M. Lucas. Solving the physical traveling salesman problem: Tree Search and macro actions. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):31–45, 2014.

[10] Rémi Munos et al. 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, 2014.

[11] George T. Heineman, Gary Pollice, and Stanley Selkow. *Algorithms in a nutshell*. O'Reilly Media, Inc., 2008.

[12] Bruce Abramson. Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):182–193, 1990.

[13] Aurélien Garivier, Emilie Kaufmann, and Wouter M. Koolen. Maximin action identification: A new bandit framework for games. In *29th Annual Conference on Learning Theory*, pages 1028–1050, 2016.

[14] Aurélien Garivier and Emilie Kaufmann. Optimal best arm identification with fixed confidence. In *Proceedings of the 29th Conference On Learning Theory (to appear)*, 2016.

[15] Donald A. Berry and Bert Fristedt. *Bandit problems: Sequential allocation of experiments (Monographs on statistics and applied probability)*. Springer, 1985.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[17] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[18] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[19] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer, 2006.

[20] Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 Computer Go. In *AAAI*, volume 8, pages 1537–1540, 2008.

[21] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michele Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

[22] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[23] Joseph L. Doob. *Stochastic processes*, volume 7. John Wiley & Sons, New York, 1953.

[24] Aleksandr Khintchine. Über einen Satz der Wahrscheinlichkeitsrechnung. *Fundamenta Mathematicae*, 6(1):9–20, 1924.

[25] Frank J. Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[26] Lutz Duembgen. Bounding standard Gaussian tail probabilities. *arXiv preprint arXiv:1012.2063*, 2010.

[27] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[28] Shipra Agrawal and Navin Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *COLT*, pages 39–1, 2012.

[29] Aurélien Garivier and Olivier Cappé. The KL-UCB algorithm for bounded stochastic bandits and beyond. In *COLT*, pages 359–376, 2011.