Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science

# Improving $(\epsilon, \delta)$-PAC Monte Carlo tree search using spherical confidence regions

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

**MASTER OF SCIENCE**
**in**
**APPLIED MATHEMATICS**

by

**Thomas Schiet (4181484)**

**Delft, Nederland**
**December 2020**

CWI
Centrum Wiskunde & Informatica

TU Delft

**MSc THESIS APPLIED MATHEMATICS**

# "Improving $(\epsilon, \delta)$-PAC Monte Carlo tree search using spherical confidence regions"


Thomas Schiet

**Technische Universiteit Delft**


**Supervisors**

Dr. W.M. Koolen

Prof. K.I. Aardal

**Other thesis committee members**

Prof. F.A. Oliehoek

December 2020 Delft

# Abstract

We consider a simplified version of the Monte Carlo tree search (MCTS) problem, a problem where, given a game tree with stochastic reward, one is tasked with finding the best move from the root. This problem is well studied, and recently impressive results have been obtained. For example, in 2016, when the AlphaGo program beat the professional Go player Lee Sedol. Even though these results were spectacular, not much is known about the guaranteed accuracy of these algorithms. In this work, we review the FindTopWinner algorithm for the MCTS problem. The FindTopWinner algorithm is guaranteed to give an accurate result with high probability, as is formalized in the ($\epsilon$,$\delta$)-PAC learning framework. However, its sample complexity is not optimal. We improve FindTopWinner by introducing the Spherical Confidence Region-MCTS (SCR-MCTS) algorithm, which operates by creating a spherical confidence region on the joint values of the leaves in multiple iterations called epochs. We show that SCR-MCTS has the ($\epsilon$,$\delta$)-PAC guarantee, and we show that empirically SCR-MCTS draws fewer samples than FindTopWinner does on various benchmark trees.

# Contents

# Chapter 1

# Introduction

Achieving super-human performance in games has always been a prestigious goal for many computer scientists. When in the '90s the reigning chess world champion was beaten in a game of chess by the Deep Blue computer, developed by IBM Research [2], this shocked the world. More recently, AlphaGo managed to win four out of five games against one of the highest-ranked Go players in the world, Lee Sedol [13]. This feat, which was unimaginable because of its computational complexity, was achieved by the DeepMind team in 2016. Even though these results are impressive, we still lack a lot of understanding in this field. In particular, theoretical guarantees on the accuracy of these algorithms are often absent.

Playing games is in many cases computationally a challenging problem. Algorithms that can play sequential games like chess, checkers or Go work on mathematical structures known as game trees. A game tree contains all states and moves of a game. In theory, it is possible to find the best move in a game tree by applying the minimax algorithm. However, this algorithm requires us to explore the complete tree. For many games, the game tree is so large, that it becomes practically impossible to do so.

One method of finding the best next move is to use a heuristic method. Instead of exploring all sequences of moves, we only try sequences of a fixed short length. After this small number of moves, we cut off the game tree and then stochastically estimate the chance of winning if we were to continue the game, starting from the cut-off point. This estimation is done by playing the game following a specific randomized policy, until it terminates. This is also known as a roll-out. The policy dictates how moves are made until a player wins or loses. By repeatedly performing these roll-outs and averaging the result, an estimate for the probability of winning is produced.

Another way of looking at this, is that instead of a large game tree with deterministic rewards, namely 'win' or 'lose', we have a smaller tree with stochastic rewards that result from performing roll-outs. Repeatedly performing roll-outs allows us to estimate the mean rewards at these terminating nodes in the smaller game tree, also known as leaves. This gives us a general framework, where we can replace roll-outs with any sort of distribution, while being agnostic about the actual policy. For example, we can simply assume that the rewards are normally distributed with some unknown mean.

Finding the next best move in this stochastic reward setting is sometimes also known as the Monte Carlo tree search (MCTS) problem. Various algorithms exist that solve this MCTS problem while having $(\epsilon,\delta)$-PAC guarantees. Roughly speaking, this means that the algorithm outputs a move that is approximately optimal and that the algorithm does this with a high probability. However, for the algorithms that are $(\epsilon,\delta)$-PAC, there is still a lot of ground to win in terms of the sample complexity.

One attempt to solve the MCTS problem under $(\epsilon,\delta)$-PAC constraints is the FindTopWinner

algorithm. This algorithm works in multiple iterations known as epochs. In each iteration, the mean reward of the leaves are estimated, and confidence intervals are constructed on the mean reward of each leaf by taking samples from their reward distribution. Based on these confidence intervals, the FindTopWinner algorithm makes its decisions and ultimately outputs a move that it believes is approximately correct.

As we will see, creating a confidence interval on each leaf is suboptimal. The algorithm works in a way that it always believes that the true mean reward of each leaf is contained in the confidence interval constructed for that leaf. Because these intervals are independent of each other, it also believes that it is plausible that many leaves are over- or underestimated. As a consequence, internal nodes of the game tree also have a wide range of plausible values. Ideally, we want the range of plausible values to be as small as possible, as this would allow us to exclude unfavorable moves early.

The approach with confidence intervals that FindTopWinner uses, can be greatly improved, while keeping its theoretical ($\epsilon$,$\delta$)-PAC bounds. In this thesis, we construct a new algorithm, called Spherical Confidence Region-MCTS (SCR-MCTS). In each epoch, instead of constructing many one-dimensional confidence intervals, SCR-MCTS constructs one multidimensional confidence region that is a hypersphere. This allows us to consider it implausible that many leaves are simultaneously incorrectly estimated, as this configuration is likely outside the radius of the hypersphere.

We will prove that SCR-MCTS is able to solve the MCTS problem while having ($\epsilon$,$\delta$)-PAC guarantees. Furthermore, we will demonstrate empirically that SCR-MCTS takes orders of magnitude fewer samples than FindTopWinner does in various benchmark game trees.

## 1.1 Related work

The Monte Carlo tree search problem that we solve in this thesis has analogies with the stochastic multi-armed bandit (MAB) problem [11]. In the multi-armed bandit problem, the learner is given multiple arms that stochastically give a reward when pulled. The learner is tasked with sampling the arms and finding the arm with the best average reward. In our problem, each 'arm' is a candidate next move, starting from the root of the game tree, with a reward based on the samples taken at the leaves.

The bandit problem is well studied. In the 1930's the method of Thompson sampling was introduced as a strategy of allocating resources in clinical trials [16]. In particular, Thompson sampling is concerned with regret minimization, a version of the MAB problem where the learner should find the best arm, while pulling unfavorable arms as little as possible. It is a classic example of the exploration/exploitation dilemma. Although Thompson sampling was initially introduced for the MAB problem with specifically two arms and Bernoulli distributed rewards, the technique has been generalized to work with any number of arms and without assumptions on the reward distributions [11].

While Thompson sampling and the MAB problem remained mostly a theoretical exercise for a large part of the 20th century, more recently it has found practical usage with the advance of computers. For example, the MAB problem is frequently used in online advertising or A/B-testing [14, 17]. With this renewed interest, a wealth of new algorithms has been developed together with many variants of the MAB problem. One algorithm that has been particularly influential is Upper Confidence Bounds (UCB) [1], that has been published in 2002. The UCB algorithm works in the regret setting and uses upper confidence bounds on the values of the arms to decide which arm to pull. In particular, it uses the principle of *optimism in face of uncertainty*, meaning that, being optimistic about the reward, it pulls the arm with the highest upper bound on its value.

In this work, we do not use the notion of regret. Instead, we do not incur any loss when we take samples in suboptimal arms. This setting is also known as *pure exploration*. In 2012 the LUCB algorithm [8] was published to solve the Explore-$m$ problem, a variant of the MAB problem where the learner is tasked to find the $m$ best arms in a $K$-armed stochastic bandit model. The LUCB algorithm makes decisions based on lower and upper confidence bounds on the values of the arms.

We can view our MCTS problem as an MAB problem with an added tree structure. This allows us to employ more tools to efficiently take samples and thus minimize the exploration. In 2017, Kaufmann and Koolen [9] introduced a class of algorithms, Best Arm Identification-MCTS (BAI-MCTS), that combines existing best arm identification algorithms with Monte Carlo tree search. It is shown that applying the LUCB algorithm to the BAI-MCTS framework, yields the LUCB-MCTS algorithm for which an upper bound on the sample complexity that holds with high probability is known.

Related is also the area of asymptotic best arm identification algorithms. These algorithms are designed to efficiently sample in the regime where the allowed error rate asymptotically goes to 0. In 2016, Garivier et al. [5] have shown that the Track-and-Stop strategy is optimal in an asymptotic sense, if the objective is to find the optimal arm. Note that this is different from the previous setting, where the objective was to find an arm that approximately optimal. Even more recently, in 2019, Degenne et al. [4] have shown that an extension of the Track-and-Stop strategy is optimal in the asymptotic regime when also allowing $\epsilon$-suboptimal arms.

An early attempt of applying bandits on trees is the UCT (UCB applied to Trees) algorithm [10]. It was introduced as an algorithm that applies the UCB algorithm recursively on each node in a tree. Important to note, is that UCT works on Markovian Decision Processes, a different class of problems, whereas our algorithm is designed specifically for game trees.

The FindTopWinner [15] algorithm solves our MCTS problem on trees of any depth. It was published in 2014 by Teraoka et al. Besides solving the MCTS problem, the authors also show an upper bound on the number of samples that FindTopWinner takes, that holds with high probability.

## 1.2 Outline

In this work, we start by reviewing basic theory regarding game tree search in Chapter 2. In this chapter we will also introduce the Monte Carlo tree search problem and formalize the ($\epsilon$,$\delta$)-PAC framework. With this background information, we will review the FindTopWinner algorithm in Chapter 3. This algorithm solves the Monte Carlo tree search problem with ($\epsilon$,$\delta$)-PAC guarantees.

Then, in Chapter 4, we will look at how FindTopWinner uses confidence regions to make its decisions. As we will see, these confidence regions are rectangular. In the same chapter we will argue why we can improve FindTopWinner by using confidence regions that have the shape of a sphere.

In Chapter 5 we will generalize the FindTopWinner algorithm to a meta-algorithm that works with confidence regions of any shape. We will then implement this algorithm for spherical confidence regions, called SCR-MCTS.

Finally, in Chapter 6 we will test our algorithm on various benchmark trees and compare it with the FindTopWinner, UGapE-MCTS and LUCB-MCTS algorithms.

# Chapter 2

# Theory

The current chapter introduces the Monte-Carlo tree search (MCTS) problem. This problem concerns finding the best next move in a game tree with stochastic rewards. We will look at the fundamentals needed to understand this problem fully. This includes basic game tree theory and the ($\epsilon$,$\delta$)-PAC learning framework. Furthermore, we will place this problem in the context of other current research.

## 2.1 Game trees

Let us consider a game played by two players, where the players take consecutive turns. In the literature this is known as a **sequential game**. A game like this has a tree structure, where the game starts in a certain state. Each move changes the state of the game to a new state. In a game like chess, the state is a configuration of the board with pieces on certain positions. We call this tree structure a **game tree**. Each state is called a **node**, and the set of nodes of a game tree $\mathcal{T}$ is denoted nodes($\mathcal{T}$). The moves between states are called **edges**. The set of nodes that can be reached from a node $n$ with one move are the **children of** $n$ and this set is denoted C($n$). In that case, for each child $m$ of $n$, the **parent of** $m$ is $n$, written as parent($m$). The set of nodes that share the same parent as a node $n$ but are not $n$, is called the **siblings of** $n$ and is denoted siblings($n$). The initial state in which the game starts, is called the **root** node. When we only look at a part of the game tree that can be reached by successive moves from a node $n \in$ nodes($\mathcal{T}$), we call that a **subtree** of tree $\mathcal{T}$ that is rooted at $n$. The subtree rooted at $n$ is written as $\mathcal{T}_n$. See Figure 2.1 for an example of a subtree of the game Tic-Tac-Toe.

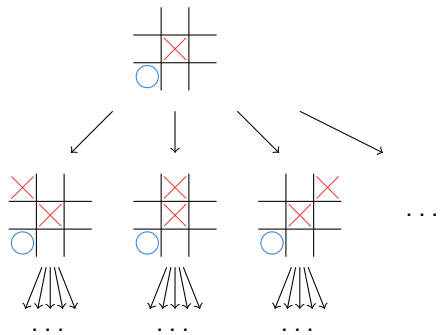


***Figure 2.1:*** *The game of Tic-Tac-Toe can be represented as a game tree. Here a subtree of the entire game tree is visualized. The full game tree has an empty board as its root node. This subtree continues until leaves are reached, which is not illustrated in this figure.*

Especially interesting is what happens once we reach the end of a game tree. In that case, we reach a certain node that has no more outgoing edges. We call a node without outgoing edges a **leaf** node. The set of all leaves of a tree $\mathcal{T}$ is denoted leaves($\mathcal{T}$). When after a move a leaf node is reached, the winner of the game gets a **reward** or **payoff** and the loser has to pay exactly this amount to the winner. This is known as a **zero-sum** game. We say that a leaf has a **value** that is won when player A wins. Since the game is zero-sum, this is the value that player B loses. Therefore player A wants the game to terminate in a high value leaf and player B wants it to terminate in a low value leaf. Consequently we call players A and B the **MAX** and **MIN** players respectively. If at a certain node the MAX player has to make a move, we call that a **max node**. Otherwise we call it a **min node**. See Figure 2.2 for an example of an subtree with leaves and their rewards.

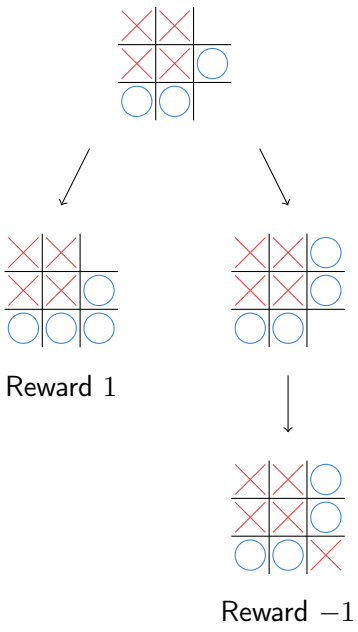



***Figure 2.2:*** *A subtree of the Tic-Tac-Toe game with leaves. The left leaf has value 1 because the MAX player (placing naughts) wins. The right leaf has rewards -1 because the MIN player (placing crosses) wins. Because the root of this subtree is a max node, the root of this subtree has value 1.*

If the game is played by players that are perfectly rational and they know the leaf values, there is a best move from each node in the tree. We say that each node has a **value**. This is the guaranteed reward the players get from that node.

**Definition 1.** *Let $\mathcal{T}$ be a game tree, $n \in \text{nodes}(\mathcal{T})$ and $\boldsymbol{\mu} \in \mathbb{R}^{\text{leaves}(\mathcal{T})}$ the value of the leaves of $\mathcal{T}$. Then the **value of node** $n$, $v_n(\boldsymbol{\mu})$, for leaf values $\boldsymbol{\mu}$, is defined by*

$$v_n(\boldsymbol{\mu}) = \begin{cases} \mu_n & \text{if } n \text{ is a leaf} \\ \max_{m \in \text{C}(n)} v_m(\boldsymbol{\mu}) & \text{if } n \text{ is a max node} \\ \min_{m \in \text{C}(n)} v_m(\boldsymbol{\mu}) & \text{if } n \text{ is a min node.} \end{cases} \tag{2.1}$$

The **minimax** algorithm computes the values of all nodes in a tree. This is done by starting at the root of the game tree and recursively applying Equation (2.1). To execute the minimax algorithm, the entire game tree is visited. In practice however, game trees can become prohibitively large to completely search. Indeed, the number of nodes in a game tree grows exponentially in the number of steps that can be taken from the root node. The number of steps from the root node to a node $n$ is also called the **depth of node** $n$. The smallest number of steps from a node $n$ to a leaf is called the **height of node** $n$. In game trees with a high number of moves per node, also known as the **outdegree** of a node, it can become intractable to search a tree just a few moves into the game. Different methods exists to simplify this problem by reducing the number of nodes that have to be visited.

### 2.1.1 Non-deterministic trees

One method of making game tree search feasible is by the use of heuristics. One heuristic is known as Monte Carlo tree search (MCTS). This method has been successful in playing games

such as Go, for example when in 2016 AlphaGo impressively won from a professional Go player [13]. Instead of traversing the whole tree, we only visit all nodes up to a certain depth. At this cut-off, we introduce **oracles** that estimate the values of the leaves. This can be done by doing **roll-outs**. In a roll-out, the algorithm takes successive random moves until a game tree leaf is reached. The oracle performs a roll-out and returns the reward that is associated with the leaf that was reached at the last step of the roll-out. MCTS repeatedly samples from the oracles to estimate values of the nodes that are associated with that oracle.

We call a game tree that has oracles at the leaves an **oracle tree**. See Figure 2.3 for an example of an oracle tree constructed from a game tree. The leaves in an oracle tree have no deterministic payoff like the leaves in a game tree do. Where before we said that the value of a leaf in a game tree was the reward it gave, we will now say that the value of a leaf in an oracle tree is the mean of the reward that the oracle at that leaf returns. The values of the rest of the nodes are then recursively defined as in Equation 2.1 with $\boldsymbol{\mu}$ being the vector of means of the oracles.

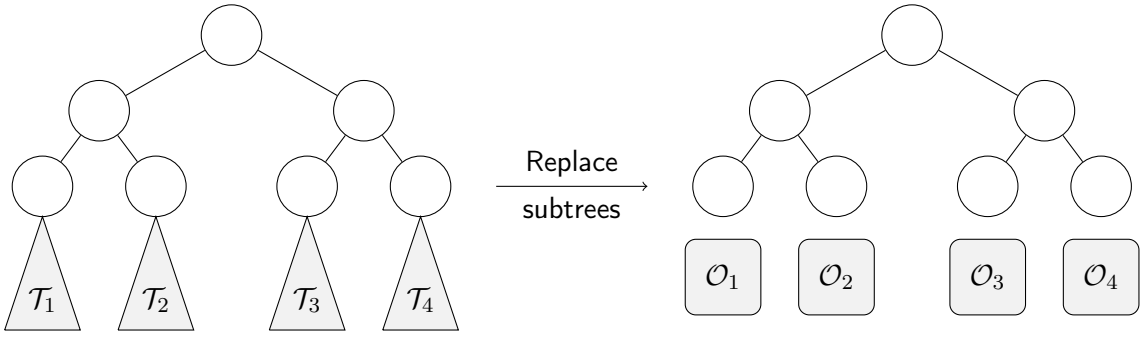



***Figure 2.3:*** *The game tree on the left has large subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_4$ indicated by the gray triangles. Each of these subtrees are replaced by oracles $\mathcal{O}_1, \ldots, \mathcal{O}_4$. This creates the oracle tree on the right. Notice that the leaves of the game tree are within the gray triangles. The leaves of the oracle tree are the nodes at depth 2 in this figure.*

**Remark 1.** *Creating an oracle tree from a game tree may change the values of nodes that remain in the oracle tree. This depends on the roll-out strategy of the oracles. Take for example Figure 2.2. Shown is a subtree of a game tree of Tic-Tac-Toe. If we replace the root of the subtree with a single oracle and use a roll-out strategy that sequentially takes independent uniformly random moves, then the value is $1/2 - 1/2 = 0$, even though clearly in Figure 2.2 the value is 1.*

Oracle trees are interesting in their own right and are not only used as a way of making game tree search tractable. Instead, oracle trees are the natural model if the reward is stochastic and not deterministic like in game trees. Oracle trees of depth one are well studied in the literature on **multi-armed bandits** (MAB), also known as bandits. Bandits form an interesting class of problems that arise in topics such as medical trials and A/B-testing.

**Remark 2** (Etymology of multi-armed bandits)**.** *The name multi-armed bandit comes from the slot machine, also known as a single-armed bandit. Single-armed because it has a lever, or arm, to pull and it's called a bandit because it may 'steal' your money. In the MAB problem there are multiple arms to pull from, and the goal is to find the best arm, typically using as few pulls as possible.*

## 2.2 Monte-Carlo tree search problem

Having introduced the concept of oracles trees, we now formulate the Monte-Carlo tree search problem. Given an oracle tree $\mathcal{T}$, what is the best move from the root? In other words, given a vector with the values of the leaves $\boldsymbol{\mu} \in \mathbb{R}^{\text{leaves}(\mathcal{T})}$, which child of the root $n \in \text{C}(\text{root})$ has the best value $v_n(\boldsymbol{\mu}) = v_{\text{root}}(\boldsymbol{\mu})$? Furthermore, we will permit an answer that is almost correct and finally we also allow that with some probability we get an arbitrarily bad answer.

### 2.2.1 $(\epsilon,\delta)$-PAC learning

This idea of allowing approximately correct answers, and with some probability any answer, is formalized in the PAC learning model. Here PAC stands for Probably Approximately Correct.

**Definition 2** ($(\epsilon,\delta)$-PAC [12]). *An algorithm is $(\epsilon,\delta)$-**PAC** if, with probability at least $1-\delta$ it outputs an answer that is at most $\epsilon$ away from the optimal answer.*

The definition contains two parameters, $\epsilon$ and $\delta$. The accuracy $\epsilon$ indicates how far away from the optimum the learner can return an answer. The confidence level $\delta$ determines with what probability the returned child node is not $\epsilon$-correct, i.e., the recommended node has a value that is more than $\epsilon$ different from the value of the root node.

### 2.2.2 Problem formulation

We will now formally define the problem that is solved in this thesis. In the following problem definition, we will call the entity interacting with the problem, the 'learner'.

***Problem 1: Monte-Carlo tree search problem***

**Input**

- Oracle tree $\mathcal{T}$ with leaf values $\boldsymbol{\mu} \in \mathbb{R}^{\text{leaves}(\mathcal{T})}$. The tree $\mathcal{T}$ is known to the learner, but the vector of values $\boldsymbol{\mu}$ is not known.
- Accuracy $\epsilon \geq 0$
- Confidence $\delta \in (0,1)$

**Protocol**
The learner is permitted the sample the oracles in rounds. The learner can stop at any time and must output a node $c \in \text{C}(\text{root})$ on the condition that the algorithm is $(\epsilon,\delta)$-PAC, i.e.,

$$\mathbb{P}(|v_c(\boldsymbol{\mu}) - v_{\text{root}}(\boldsymbol{\mu})| \leq \epsilon) \geq 1-\delta. \tag{2.2}$$

**Objective**
Output with probability at least $1-\delta$ an at most $\epsilon-$suboptimal node while taking as few samples as possible.

We call the problem the Monte-Carlo tree search problem, following the FindTopWinner paper [15]. However, we should note this is a term frequently used for algorithms that solve a similar problem on game trees. In these algorithms, often the tree is expanded while being searched [3], which is something we do not do in this thesis.

How well an algorithm solves this problem will be measured by the expected total number of samples taken from the oracles. This is also known as the **fixed-confidence** setting [6].

Alternatively, one might work in the **fixed-budget** setting, where the number of samples is fixed and the learner has to optimize the probability of returning an $\epsilon$-correct child node [6].

Having introduced the Monte-Carlo tree search problem, we will now review an existing approach to solving this problem, namely the FindTopWinner algorithm. After introducing FindTopWinner we will motivate a direction of improvement in Chapter 4.

# Chapter 3

# The earlier FindTopWinner algorithm

In this chapter we will review the FindTopWinner algorithm developed by Tereoka, Hatano and Takimoto [15]. This algorithm is designed to efficiently solve Problem 1. We will state the algorithm and sketch a proof of why FindTopWinner is ($\epsilon$,$\delta$)-PAC. Finally, we briefly look at its sample complexity.

## 3.1 The FindTopWinner Algorithm

The FindTopWinner algorithm, which is shown in Algorithm 1, works in multiple iterations called **epochs**. In each epoch $m = 1, \ldots, \lceil \log 2/\epsilon \rceil$, all remaining oracles are sampled a fixed and equal number of times. The number of samples taken per leaf is roughly exponential in the number of epochs. With the resulting samples, the value of each leaf is estimated. Moreover, a confidence interval is constructed on the value of each leaf. The estimates are used to estimate the value of the internal nodes of the oracle tree. At the end of each epoch nodes from the oracle tree are pruned if they are implausible to be considered winners, where a winner is defined as in Definition 3 and we say a node $u$ is implausible to be a winner if there is no plausible model in the confidence region $\boldsymbol{\mu}' \in \mathcal{C}_m$ for which node $u$ is a winner. Finally, a child of the root is returned with the best value estimate.

**Definition 3** (Winner [15])**.** *Let $\mathcal{T}$ be an oracle tree and let $\boldsymbol{\mu} \in \mathbb{R}^{\text{leaves}(\mathcal{T})}$ be the vector of values of the leaves. A node $u \in \text{nodes}(\mathcal{T})$ is a **winner** if*

$$v_u(\boldsymbol{\mu}) = v_{\text{parent}(u)}(\boldsymbol{\mu}). \tag{3.1}$$

Hence the name of the algorithm, FindTopWinner. As an example of winners, see Figure 3.1.

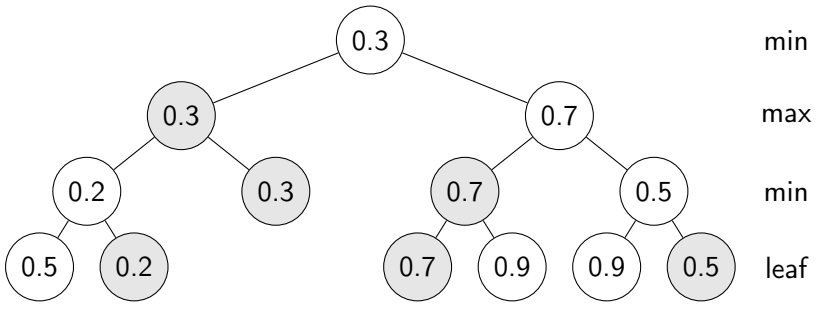



***Figure 3.1:*** *Shown is an oracle tree with the values of the nodes (oracles omitted). The filled nodes are winners as defined in Definition 3.*

Below, the algorithm is explicitly stated.

**Algorithm 1** The FindTopWinner algorithm solves the Monte Carlo tree search Problem 1

**procedure** FINDTOPWINNER($\epsilon, \delta, \mathcal{T}$)
  $\epsilon_0 = 1$
  $\delta_0 = \delta / L$
  **for** $m = 1$ to $\lceil \log_2(2/\epsilon) \rceil$ **do**
    $\epsilon_m = \epsilon_{m-1}/2$ ▷ Halves the interval size
    $\delta_m = \delta_{m-1}/2$ ▷ Doubles confidence of new interval
    **if** $|\,\mathrm{C}(\text{root})| = 1$ **then**
      **break** ▷ Return only child remaining
    **end if**
    ESTIMATEVALUES(root, $\epsilon_m, \delta_m$)
    PRUNE(root, $\epsilon_m$)
  **end for**
  **return** $I = \arg\max_{u \in \mathrm{C}(\text{root})} v_u(\hat{\boldsymbol{\mu}})$
**end procedure**

**Algorithm 2** The EstimateValues subalgorithm samples the oracle leaves.

**procedure** ESTIMATEVALUES($u, \epsilon_m, \delta_m$)
  **if** $u \in \text{leaves}(\mathcal{T})$ **then**
    Sample oracle $O_u$ until the total of calls of $O_u$ reaches $n_m = \left\lceil (1/(2\epsilon_m^2)) \ln(2/\delta_m) \right\rceil$. Let $S_u$ be the sum of output of $O_u$.
    **return** $S_u / n_m$
  **end if**
  **for** $v \in \mathrm{C}(u)$ **do** ▷ Recursively call ESTIMATEVALUES to sample all leaves.
    $\hat{\mu}_v =$ ESTIMATEVALUES($v, \epsilon_m, \delta_m$)
  **end for**
  **if** $u$ is max node **then** ▷ Return the value $v_u(\hat{\boldsymbol{\mu}})$
    **return** $\max_{v \in \mathrm{C}(u)} \hat{\mu}_v$
  **else**
    **return** $\min_{v \in \mathrm{C}(u)} \hat{\mu}_v$
  **end if**
**end procedure**

**Algorithm 3** The PRUNE subalgorithm prunes nodes that are implausible to be winners.

**procedure** PRUNE($u, \epsilon_m$)
  **for** $v \in \mathrm{C}(u)$ **do**
    **if** $|\hat{\mu}_u - \hat{\mu}_v| > 2\epsilon_m$ **then**
      $\mathrm{C}(u) = \mathrm{C}(u) \setminus \{v\}$ ▷ Prune child $v$
    **else if** $v \notin \text{leaves}(\mathcal{T})$ **then**
      PRUNE($v, \epsilon_m$) ▷ Recursively prune the descendants of $v$
    **end if**
  **end for**
**end procedure**

## 3.2 FindTopWinner is $(\epsilon,\delta)$-PAC

In this section, we will give a sketch of why the FindTopWinner algorithm is correct. The full proof is constructed by Terraoka et al. [15].

First, it is shown that with probability at least $1-\delta$, simultaneously in every epoch $m$, all estimates of the values of the non-pruned leaves are at least $\epsilon_m$-correct. This event is called Event $A$. In some sense, this is the 'happy' event, if Event $A$ happens everything goes well.

**Definition 4** (Event $A$ [15])**.** *The event $A$ is such that, simultaneously at each epoch $m$ and for each leaf $\ell \in \text{leaves}(\mathcal{T})$, the estimate $\hat{\mu}_{\ell,m}$ obtained by* EstimateValues$(\ell, \epsilon_m, \delta_m)$ *(see Algorithm 2) satisfies $|\mu_\ell - \hat{\mu}_{\ell,m}| \leq \epsilon_m$.*

It can be proven that indeed Event $A$ happens with probability at least $1-\delta$. We state this in Lemma 1.

**Lemma 1.** [15] *The event $A$ occurs with probability at least $1-\delta$.*

*Proof.* [15] With Hoeffding's inequality we have for all epochs $m$ and for all leaves $\ell \in \text{leaves}_m(\mathcal{T})$ that for the estimate $\hat{\mu}_{\ell,m}$ after $m$ epochs holds

$$\mathbb{P}(|\mu_\ell - \hat{\mu}_{\ell,m}| > \epsilon_m) \leq 2\exp(-2\epsilon^2 n_m) \leq \frac{\delta}{2^m L}. \tag{3.2}$$

Then using the union bound on all epochs and leaves we get

$$\mathbb{P}(\text{Event A does not occur}) = \mathbb{P}(\exists m \geq 1, \exists \ell \in \text{leaves}_m(\mathcal{T}) : |\mu_\ell - \hat{\mu}_{\ell,m}| > \epsilon_m) \tag{3.3}$$

$$\leq L \sum_{m=1}^{\infty} \frac{\delta}{2^m L} = \delta. \tag{3.4}$$

Hence the probability that event $A$ occurs is at least $1-\delta$. □

Next, Teraoka et al. show that when Event $A$ happens, the estimates of the values of the internal nodes are also $\epsilon_m$-correct, i.e., we have for any node $u$ that has not been pruned we have,

$$|v_u(\boldsymbol{\mu}) - v_u(\hat{\boldsymbol{\mu}})| \leq \epsilon_m, \tag{3.5}$$

where $\boldsymbol{\mu}$ is the vector of leaf values and $\hat{\boldsymbol{\mu}}$ is the estimate of $\boldsymbol{\mu}$. In other words, if our estimate on the value of the leaves is good, then the estimate of the values on the internal nodes is also good. Furthermore, it is shown that because each node is approximately correct, that only nodes that are not winners are pruned.

Finally, because no winners are pruned, the values of all remaining nodes are unaffected by pruning. Indeed, the value of a parent node is by definition equal to that of its winning children. Since in the final epoch, $\epsilon_m$ equals $\epsilon$, only $\epsilon$-suboptimal arms remain.

**Theorem 1.** [15] *With probability at least $1-\delta$, the* FindTopWinner *algorithm returns a node $u$ such that $|v_u(\boldsymbol{\mu}) - v_{root}(\boldsymbol{\mu})| \leq \epsilon$. In other words,* FindTopWinner *is $(\epsilon,\delta)$-PAC.*

## 3.3 Sample complexity

The FindTopWinner algorithm has a upper bound on the number of samples taken. We will review this bound and give some intuition behind it. Let us first define the parameter $\Delta_\ell$. As we will see, the sample complexity will depend on this parameter.

**Definition 5.** *For each leaf $\ell$ we define the parameter $\Delta_\ell$ as*

$$\Delta_\ell = \max_{u \in \text{anc}(\ell) \setminus \{\text{root}\}} |v_u(\boldsymbol{\mu}) - v_{\text{parent}(u)}(\boldsymbol{\mu})|. \tag{3.6}$$

This parameter can be thought of as the largest gap between values above some leaf $\ell$. The value of $\Delta_\ell$ is determined by the most suboptimal move that has to be taken to reach the leaf $\ell$ from the root node.

With this parameter, we state the sample complexity upper bound found by Teraoka et al. in the following theorem.

**Theorem 2.** [15] *With probability at least $1-\delta$, the number of oracle calls of the* FINDTOPWINNER *algorithm is at most*

$$\sum_{\ell:\Delta_\ell > 2\epsilon} \left( \frac{32}{\Delta_\ell^2} \ln \frac{16L}{\Delta_l \delta} + 1 \right) + \sum_{\ell:\Delta_\ell \leq 2\epsilon} \left( \frac{8}{\epsilon^2} \ln \frac{8L}{\epsilon\delta} + 1 \right). \tag{3.7}$$

Notice that Equation (3.7) contains two summations, the first sums over all leaves such that $\Delta_\ell > 2\epsilon$ and the second sums over all other leaves. It can be shown that if Event $A$ happens and if a leaf $\ell$ satisfies $\Delta_\ell > 2\epsilon$, that the leaf $\ell$ is pruned at the latest in epoch

$$m_\ell = \lfloor \log_2 4/\Delta_\ell \rfloor + 1. \tag{3.8}$$

Leaves that do not satisfy the condition $\Delta_\ell > 2\epsilon$ are not guaranteed to be pruned in any epoch, even if Event $A$ happens. Consequently, the leaf is only guaranteed to be pruned in the last epoch, $M = \lceil \log_2(2/\epsilon) \rceil$. Since the number of samples per leaf is deterministic in each epoch, it is easily shown that the upper bound on the number of samples for these leaves is given by the second term in Equation (3.7).

## 3.4 Working with normally distributed oracles

The algorithm that is introduced in Chapter 5 works primarily with normally distributed oracles. To compare FINDTOPWINNER to the algorithm introduced in Chapter 5, we can adjust the sampling rule of FINDTOPWINNER to work with any $\sigma$-sub-Gaussian random variables as defined in Definition 6.

**Definition 6** ($\sigma$-sub-Gaussian)**.** *Let $X$ be a real-valued random variable with mean $\mathbb{E}[X] = \mu$. $X$ is $\sigma$-sub-Gaussian if there exists a $\sigma > 0$ such that*

$$\mathbb{E}\left[\exp(\lambda(X-\mu))\right] \leq \exp\left(\sigma^2\lambda^2/2\right) \quad \forall \lambda \in \mathbb{R}. \tag{3.9}$$

Note that a normally distributed random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ is $\sigma$-sub-Gaussian. Then the mean of $n$ i.i.d. copies of $X$ is $(\sigma/\sqrt{n})$-sub-Gaussian.

In the original paper, only in event A the fact that rewards are Bernoulli distributed is used. Therefore, it is relatively straightforward to extend the algorithm to the sub-Gaussian case. If the rewards are $\sigma$-sub-Gaussian distributed instead of Bernoulli, then instead of using the Hoeffding bound in Eq (3.2), we use the Chernoff bound. Then Equation (3.2) becomes

$$\mathbb{P}(|\mu_\ell - \hat{\mu}_\ell| > \epsilon_m) \leq 2\exp(-\epsilon^2 n_m/(2\sigma^2)) \leq \frac{\delta}{2^m L}. \tag{3.10}$$

So to sample $\sigma$-sub-Gaussian random variables, we can set the number of samples taken, as can be see in line 4 of Algorithm 2, to

$$n_m = 4\sigma^2 \lceil \ln(2/\delta_m)/(2\epsilon_m) \rceil. \tag{3.11}$$

By applying the union bound on all epochs and leaves as in the proof of Lemma 1 we obtain that event $A$ occurs with $1 - \delta$ probability.

Now that we have reviewed the FindTopWinner algorithm, we will visit the concept of confidence regions and we will see how FindTopWinner uses confidence regions.

# Chapter 4

# Confidence regions

In the bandit literature, confidence intervals are a regular theme. Usually, a confidence interval is constructed on the values of nodes or leaves. These confidence intervals are then used to make pruning, sampling, or recommendation decisions. In this chapter we will generalize confidence intervals to multidimensional confidence regions. Furthermore, we will look at how FindTopWinner uses confidence regions. Finally, we will compare the performance of rectangular and spherical confidence regions to motivate our new algorithm introduced in Chapter 5.

## 4.1 Confidence regions

A confidence region is the generalization of a confidence interval to multiple dimensions. Consequently it does not only have a size but also a shape. Formally we define it as follows.

**Definition 7.** *Let $X$ be a $\mathbb{R}^K$-valued random variable with mean $\boldsymbol{\mu} \in \mathbb{R}^K$. A **confidence region** $\mathcal{C}(X) \subseteq \mathbb{R}^K$ is a random set such that it contains the mean $\boldsymbol{\mu}$ with confidence $\delta \in (0, 1)$. That is,*

$$\mathbb{P}(\boldsymbol{\mu} \in \mathcal{C}(X)) \geq 1 - \delta. \tag{4.1}$$

*Typically, we write $\mathcal{C}$ instead of $X$.*

A confidence region should be interpreted as a set such that if an experiment or sample is repeated, then in a fraction $1 - \delta$ of all the confidence regions, the true parameter is contained. Note that for a random variable $X$ it is possible to choose from different shapes of confidence regions that have the same confidence $\delta$. In principle, any shape is permissible as long as Equation (4.1) is satisfied.

**Remark 3.** *Confidence regions are frequently misunderstood. A common misinterpretation is that after a confidence region $\mathcal{C}$ is created, it will contain a different estimate $\hat{\boldsymbol{\mu}}$ with probability at least $1 - \delta$, but this is not true.*

We call any $\boldsymbol{\mu}' \in \mathcal{C}$ **plausible** and any $\boldsymbol{\mu}' \notin \mathcal{C}$ is called **implausible**. Furthermore, a plausible vector $\boldsymbol{\mu}'$ not equal to the empirical estimate $\hat{\boldsymbol{\mu}}$ is sometimes called an **alternative model**.

### 4.1.1 Confidence regions in FindTopWinner

In FindTopWinner, in each epoch and for each leaf, a confidence interval is constructed. In each subsequent epoch, the size of the interval is halved. This can be seen in Algorithm 1, where in line 5 the interval is halved. In the EstimateValues algorithm, as described in Algorithm 2, the confidence region is constructed by sampling the oracles in Line 4.

In each epoch $m$, the union bound is applied over all confidence intervals created, and consequently the confidence intervals are simultaneously true with a probability at least $1 - \delta_m$. This can be seen in Lemma 1. This creates a confidence region with the shape of a box as is shown in Figure 4.1.

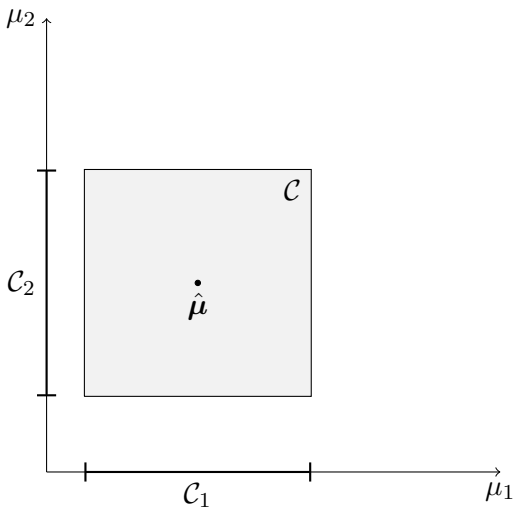



***Figure 4.1:*** FINDTOPWINNER *creates a confidence interval per leaf, $\mathcal{C}_1$ and $\mathcal{C}_2$. Because the union bound is applied over all leaves, these confidence intervals will be simultaneously true with probability at least $1 - \delta_m$. Consequently, $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ is a confidence region with confidence $\delta_m$.*

The confidence regions in FINDTOPWINNER are on the values of the leaves. But what does that say about the values of the internal nodes? The fact that the confidence region is a hypercube with sides of length $2\epsilon$, means that for all leaves $\ell \in \text{leaves}(\mathcal{T})$ we can independently pick any value $\mu_\ell$ in the interval $[\hat{\mu}_\ell - \epsilon, \hat{\mu}_\ell + \epsilon]$. Now, pick any node $u \in \text{nodes}(\mathcal{T})$ of height one, so that all its children are leaves. No matter if $u$ is a min node or a max node, its value can attain $v_u(\hat{\boldsymbol{\mu}}) \pm \epsilon$. Since $u$ does not share any children with its siblings and we can independently pick the value of the leaves, each sibling $s \in \text{siblings}(u)$ can attain a value $v_s(\hat{\boldsymbol{\mu}}) \pm \epsilon$. We conclude that the set of plausible values of height 1 nodes is a hypercube with sides of length $2\epsilon$. Using an induction argument, we see that this happens at any height of the tree.

## 4.2 Motivation

In Definition 7 we defined the term confidence region. As mentioned, a confidence region can have any shape. We have seen in the previous section that FINDTOPWINNER uses a rectangular confidence region, but nothing restricts us from creating a confidence region over the values of the leaves that has a different shape. One problem of a rectangular confidence region, is that for a parent node, the range of its values is still $\pm\epsilon$. However, if the parent node is a max node, this seems a very wide range of values. Indeed, if the parent nodes achieves a low value, then all its children must also attain a low value, which is unlikely.

To motivate why using a different shape of confidence region is advantageous for solving the MCTS problem, Problem 1, we will investigate how well different shapes of confidence regions perform in a similar but simplified problem. In this simplified problem, we have $K$ arms that give normally distributed rewards with mean $\boldsymbol{\mu} \in \mathbb{R}^K$ and unit variance. Furthermore, we fix a threshold $\gamma \in \mathbb{R}$ and assign a number of samples $n$ that we take from each arm. We construct a

confidence region $\mathcal{C}$ of some shape on the mean reward of these arms. We then ask: can we make it plausible that none of the arms have a value that is above the threshold $\gamma$? This is similar to a pruning operation, because when making a pruning decision, we also compare the values of nodes with a threshold, namely the value of another node. We formalize this in Problem 2.

***Problem 2: Pure exploration toy problem***

**Input**

- Random variable $X \sim \mathcal{N}(\boldsymbol{\mu}, I_K)$, where $\boldsymbol{\mu} \in \mathbb{R}^K$ and $I_K$ is the $K \times K$ identity matrix.
- Threshold $\gamma \in \mathbb{R}$ such that $\mu_k < \gamma$ for each $k \in [K]$.
- Number of samples $n \in \mathbb{N}$
- Confidence level $\delta \in (0, 1)$

**Protocol**

- The algorithm must sample $n$ i.i.d. copies of $X$ to create a confidence region $\mathcal{C}$ for the mean $\boldsymbol{\mu}$
- The confidence region $\mathcal{C}$ must have confidence $\delta$
- The algorithm must return true if and only if for all $\boldsymbol{\mu}' \in \mathcal{C}$ there exists an arm $k$ such that $\mu'_k < \gamma$. In other words, it must return true if and only if $\mathcal{C} \cap \mathbb{R}_{\geq \gamma} = \emptyset$.

**Objective**
Minimize the probability $\mathbb{P}(\mathcal{C} \cap \mathbb{R}_{\geq \gamma} \neq \emptyset)$.

In other words, the learner should choose a shape of the confidence region such that with high probability it can show that in each plausible model $\boldsymbol{\mu}' \in \mathcal{C}$, not all arms pass the threshold $\gamma$. This is illustrated in Figure 4.2.

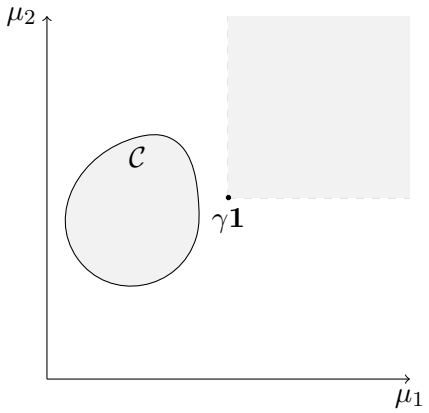



***Figure 4.2:*** *If it is implausible that all arms pass the threshold $\gamma$, i.e., for all $\boldsymbol{\mu}' \in \mathcal{C}$ we see that $\min_k \mu'_k < \gamma$, then the confidence region $\mathcal{C}$ must be outside the shaded area $\mathbb{R}_{\geq \gamma}$.*

We want to compare two strategies that solve Problem 2. Because FindTopWinner uses confidence regions with a box shape, we will construct confidence regions that are $K$-dimensional

cubes. We will compare this with a $K$-dimensional sphere. This is illustrated in Figure 4.3

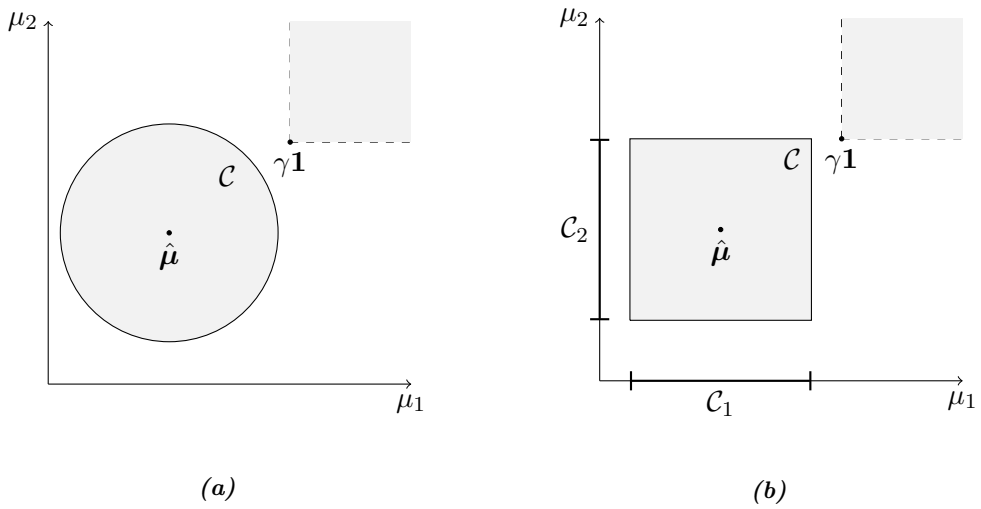



***Figure 4.3:*** *We will compare a spherical confidence region with a rectangular confidence region. In this this example, both strategies would decide that yes, $\gamma > \min_k \mu_l$, because both confidence regions do not intersect with $\mathbb{R}_{\geq\gamma}$.*

We hypothesize that the sphere yields better results if the means $\mu_1, \mu_2, \ldots, \mu_K$ are close to each other, as in that case all arms need to be overestimated for $\min_k \mu_k > \gamma$ to hold. This is less likely plausible in a spherical confidence region than it is in a box shaped confidence region.

### 4.2.1 Simplification

We would like to know the probability that a confidence region $\mathcal{C}$ intersects $\mathbb{R}_{\geq\gamma}$ in the case where $\mathcal{C}$ is a sphere and when it is a box. However, it turns out that this is hard to explicitly compute in the case that $\mathcal{C}$ is a sphere. To illustrate, to calculate this probability we have to integrate the probability density function of the multivariate normal distribution over the domain in which the sphere around $\hat{\boldsymbol{\mu}}$ would intersect the threshold area. This is shown in Figure 4.4.

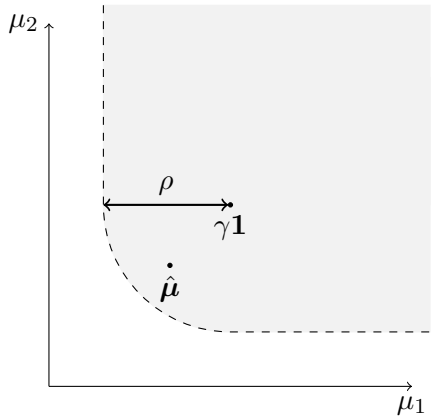



***Figure 4.4:*** *The sphere $B(\hat{\boldsymbol{\mu}}, \rho)$ intersects $\mathbb{R}_{\geq\gamma}$ if and only if $\hat{\boldsymbol{\mu}}$ is within the shaded region.*

The result of this integral does not seem to have an answer that can be expressed in elementary functions. Specifically because of the quarter (or rather $1/2^K$-th) sphere in the south-west

quadrant. Since the center of this sphere is not at the center of the probability density function, this integral becomes intractable. Therefore, we introduce a problem that is an analogous but simplified version of Problem 2. Instead, the learner must try to create a confidence region $\mathcal{C}$ that excludes the point $\boldsymbol{\gamma}$ with high probability.

***Problem 3: Simplified pure exploration toy problem***

**Input**

- Random variable $X \sim \mathcal{N}(\boldsymbol{\mu}, I_K)$, where $\boldsymbol{\mu} \in \mathbb{R}^K$ and $I_K$ is the $K \times K$ identity matrix.
- Threshold $\boldsymbol{\gamma} \in \mathbb{R}^K$ such that $\mu_k \leq \gamma_k$ for each $k \in [K]$.
- Number of samples $n \in \mathbb{N}$
- Confidence level $\delta \in (0, 1)$

**Protocol**

- The algorithm must sample $n$ i.i.d. copies of $X$ to create a confidence region $\mathcal{C}$ for the mean $\boldsymbol{\mu}$
- The confidence region $\mathcal{C}$ must have confidence $\delta$
- The algorithm must return true if and only if for all $\boldsymbol{\mu}' \in \mathcal{C}$ we have for each arm $k$ that $\mu'_k \leq \gamma_k$. In other words, it must return true if and only if $\boldsymbol{\gamma} \notin \mathcal{C}$.

**Objective**
Minimize the probability $\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C})$.

In the rest of this section we will discuss the two strategies we introduced for Problem 2 and how well they solve Problem 3. We argue that the output of the strategies is similar for both problems, since it is unlikely that the means will be so overestimated that the estimate $\boldsymbol{\mu}'$ ends up behind the threshold $\boldsymbol{\gamma}$, as illustrated in Figure 4.5.

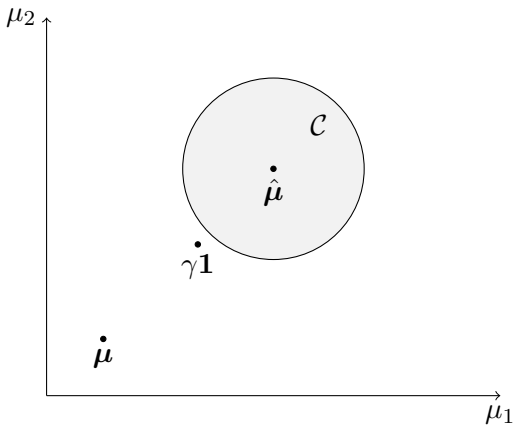



***Figure 4.5:*** *Because the mass of the density function is mostly concentrated near $\boldsymbol{\mu}$, it is unlikely for the estimate $\hat{\boldsymbol{\mu}}$ to attain a value such that $\min_k \mu_k > \gamma$ and the confidence region $\mathcal{C}$ does not contain $\gamma\mathbf{1}$.*

We will now compute the probability that the confidence region $\mathcal{C}$ contains $\boldsymbol{\gamma} \in \mathbb{R}^K$, $\mathbb{P}(\gamma \in \mathcal{C})$.

This is the probability that our algorithms return false in Problem 3 and the objective is to minimize this value. In the following two settings, the number of samples taken is fixed at $n$, the confidence level equals $\delta$, we have threshold $\boldsymbol{\gamma}$ and the means of the arms is $\boldsymbol{\mu} \in \mathbb{R}^K$. The estimate $\hat{\boldsymbol{\mu}}$ is the arithmetic mean of $n$ samples of i.i.d. copies of $X$.

### Sphere

First, let us determine the radius $\rho$ of the spherical confidence region $\mathcal{C} = B(\hat{\boldsymbol{\mu}}, \rho)$ that we will construct. Recall that $X$ is a multivariate normal distribution with mean $\boldsymbol{\mu}$ and unit variance, so we have $\hat{\mu}_k \sim \mathcal{N}(\mu_k, 1/n)$. Let $Z = n||\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}||^2$. Then $Z$ has a $\chi^2_K$-distribution. Indeed,

$$Z = n \sum_{k=1}^{K} (\hat{\mu}_k - \mu_k)^2 \sim \chi^2_K. \tag{4.2}$$

By inverting the following equation we obtain the radius $\rho$ of the confidence region.

$$\mathbb{P}(\boldsymbol{\mu} \in \mathcal{C}) = \mathbb{P}(||\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}||^2 \leq \rho^2) = \mathbb{P}(Z \leq n\rho^2) = 1 - \delta. \tag{4.3}$$

We can numerically solve (4.3) for $\rho$ by using the quantile function for the $\chi^2_K$-distribution. Now that we know the radius, we can compute the probability that the sphere contains $\boldsymbol{\gamma}$. We have,

$$\begin{aligned} Z' &= n||\hat{\boldsymbol{\mu}} - \boldsymbol{\gamma}||^2 && (4.4)\\ &= n \sum_{k=1}^{K} (\hat{\mu}_k - \gamma_k)^2 && (4.5)\\ &\sim \chi'^2_K(\lambda), && (4.6) \end{aligned}$$

where $\chi'^2_K(\lambda)$ is the non-central $\chi^2$-distribution with parameter

$$\lambda = n \sum_{k=1}^{K} (\mu_k - \gamma_k)^2. \tag{4.7}$$

This yields the probability

$$\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C}) = \mathbb{P}(||\hat{\boldsymbol{\mu}} - \boldsymbol{\gamma}||^2 \leq \rho^2) = \mathbb{P}(Z' \leq n\rho^2) = F_{Z'}(n\rho^2). \tag{4.8}$$

Where $F_{Z'}$ is the cumulative density function of the non-central $\chi^2_K$-distribution. We conclude that we can compute the radius $\rho$ of the confidence region $\mathcal{C} = B(\hat{\boldsymbol{\mu}}, \rho)$, and the probability that $\mathcal{C}$ contains is given by Equation 4.8.

Next, we will compute the same probability for $\mathcal{C}$ if it is a box.

### Box

We start with the size of the box. We sample each arm $n$ times and find an estimate $\hat{\boldsymbol{\mu}}$. The confidence region $\mathcal{C}$ is a box with center $\hat{\boldsymbol{\mu}}$ and sides of length $2\epsilon$. The probability that $\boldsymbol{\mu}$ is in the confidence region $\mathcal{C}$ can be expressed in terms of $\epsilon$ and is set equal to $1 - \delta$,

$$\begin{aligned} \mathbb{P}(\boldsymbol{\mu} \in \mathcal{C}) = \mathbb{P}(\forall k : |\hat{\mu}_k - \mu| \leq \epsilon) &= \prod_{k=1}^{K} \mathbb{P}(|\hat{\mu}_k - \mu| \leq \epsilon) && (4.9)\\ &= \prod_{k=1}^{K} \left(F_X(\mu_k + \epsilon) - F_X(\mu_k - \epsilon)\right) && (4.10)\\ &= 1 - \delta. && (4.11) \end{aligned}$$

We numerically solve this equation for $\epsilon$. Then we find that the probability that the threshold $\boldsymbol{\gamma}$ is contained in the confidence region $\mathcal{C}$ equals,

$$\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C}) = \mathbb{P}(\forall k : |\hat{\mu}_k - \gamma_k| \leq \epsilon) = \prod_{k=1}^{K} \mathbb{P}(|\hat{\mu}_k - \gamma_k| \leq \epsilon) = \prod_{k=1}^{K} F_{X'_k}(\epsilon), \tag{4.12}$$

where $X'_k \sim \mathcal{N}(\mu_k - \gamma_k, 1/n)$.

Now that we have the probability of including the threshold $\boldsymbol{\gamma}$ in the confidence region $\mathcal{C}$ for both strategies, we will show numerical results for both and compare them.

### 4.2.2 Strategy comparison results

We start by comparing the output of both strategies on Problem 2 and its simplified version, Problem 3. In this particular case, we set $\boldsymbol{\gamma} = 0.1 \cdot \mathbf{1}, \delta = 0.01, \boldsymbol{\mu} = \mathbf{0}$. It shows empirically that the simplification of Problem 2 into Problem 3 is justified for the sphere and the box.

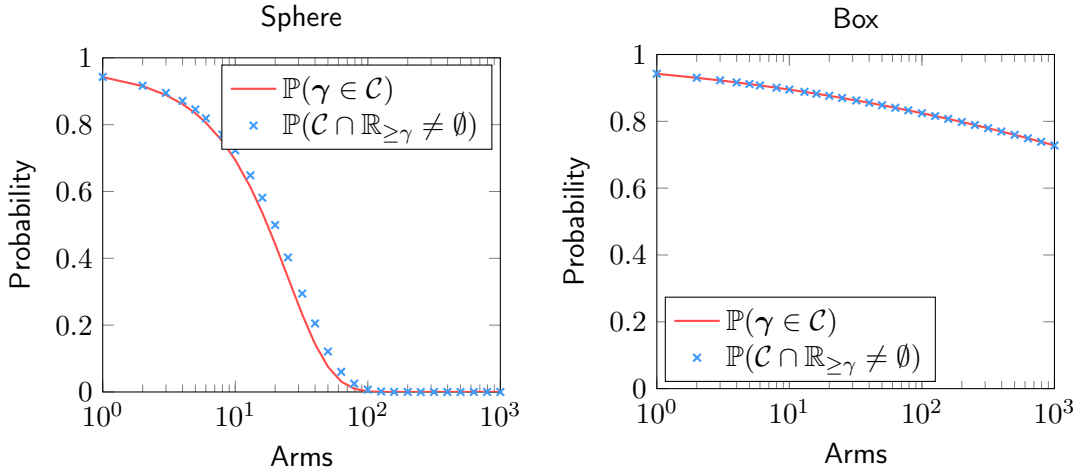



***Figure 4.6:*** *It is shown that the probabilities $\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C})$ and $\mathbb{P}(\mathcal{C} \cap \mathbb{R}_{\geq\gamma} \neq \emptyset)$ are roughly equal for any number of arms if $\mathcal{C}$ is a sphere. If $\mathcal{C}$ is a box, we see that the results are nearly indistinguishable. The results for $\mathbb{P}(\mathcal{C} \cap \mathbb{R}_{\geq\gamma} \neq \emptyset)$ were obtained using numerical integration. In all results we set $n = 100, \boldsymbol{\mu} = \mathbf{0}, \delta = 0.01$ and $\boldsymbol{\gamma} = 0.1 \cdot \mathbf{1}$.*

We will now compare the two strategies on the simplified problem. We look at the case of $K = 10$ arms, a mean of $\boldsymbol{\mu} = \mathbf{0}$ and threshold $\boldsymbol{\gamma} = 0.01 \cdot \mathbf{1}$. As can be seen in Figure 4.7, fewer samples are needed to confidently exclude the threshold from the confidence region if the confidence region is a sphere instead of a box.

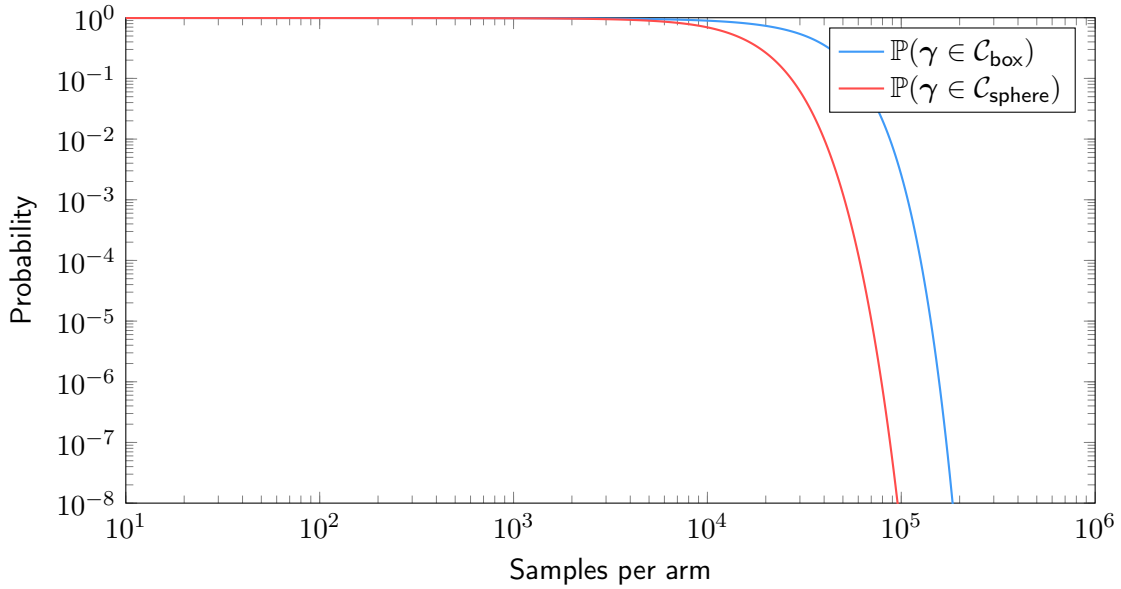



***Figure 4.7:*** *Comparison between $\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C}_{box})$ and $\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C}_{sphere})$. We see that for a small number of samples per arm, it is very unlikely to find that $\boldsymbol{\gamma}$ is outside the confidence region $\mathcal{C}_{box}$ or $\mathcal{C}_{sphere}$. This probability quickly vanishes for both shapes. We see that for the box, this happens at a much later point than for the sphere. In these experiments we fix $K = 10, \delta = 0.01, \boldsymbol{\gamma} = 0.01 \cdot \mathbf{1}$ and $\boldsymbol{\mu} = \mathbf{0}$.*

We also see that the difference becomes greater when the number of arms is increased. Below, we run the same experiment but then for different numbers of arms. Specifically, we look at the case with $10, 10^2, 10^3$ and $10^4$ arms.

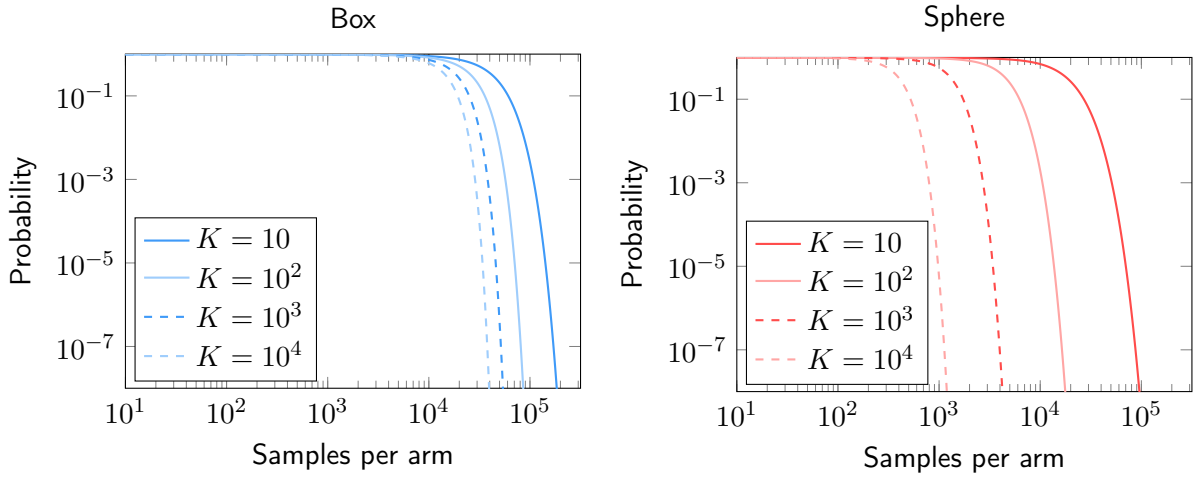



***Figure 4.8:*** *Comparison between $\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C}_{box})$ and $\mathbb{P}(\boldsymbol{\gamma} \in \mathcal{C}_{sphere})$ at different number of arms $K$. We see that the probability vanishes rather quickly. For a high number of arms, this happens with fewer samples per arm. This effect is greater for the sphere than for the box. In these experiments we fix $\delta = 0.01, \boldsymbol{\gamma} = 0.01 \cdot \mathbf{1}$ and $\boldsymbol{\mu} = \mathbf{0}$.*

The results suggest that using a spherical confidence region has benefits over using a box shape, like done in FindTopWinner. In particular, we see that fewer samples per arm are needed to make it implausible that all arms cross a threshold $\gamma$ if the confidence region is a sphere instead of a box. Therefore, we believe that it is easier to make correct pruning decisions with a spherical confidence region.

### 4.2.3 Limitations

In the experiment we compared confidence regions that are either a sphere or a box. Even though the set of plausible values of the leaves is a sphere, the set of plausible values of internal nodes may not be. Let us start with an example of a tree of depth 2 and let the root node be a min node. Because the tree has four leaves, we can construct a spherical confidence region $\mathcal{C}$ that is the 4 dimensional sphere $B(\hat{\boldsymbol{\mu}}, \rho)$. Since $\mathcal{C}$ is the set of all plausible values of the nodes, one might ask what is the set of all plausible values of the internal nodes? Let us assume for simplicity that $\hat{\boldsymbol{\mu}} = \alpha\mathbf{1}$ for some real number $\alpha$. We find that the plausible set of joint values for the two nodes of depth 1 is not a sphere. This set is illustrated in Figure 4.9. We were able to compute this set using tools introduced in Chapter 5. See Appendix B for details.

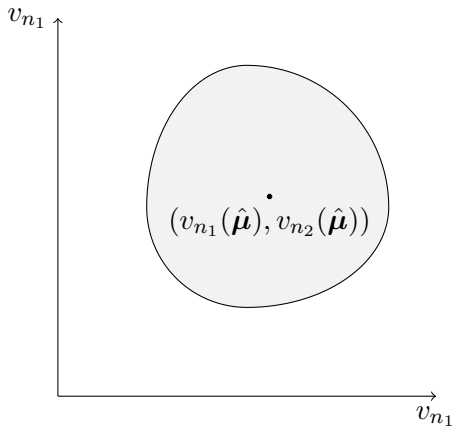
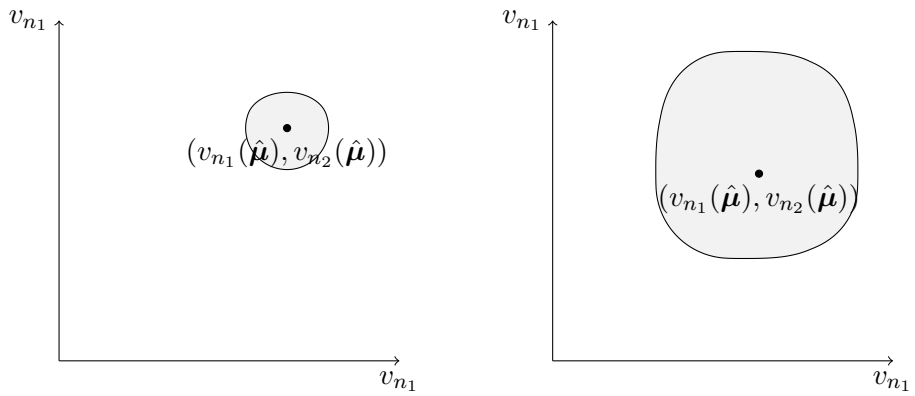



***Figure 4.9:*** *Plausible values for two max nodes $n_1$ and $n_2$, given some $\hat{\boldsymbol{\mu}} = \alpha\mathbf{1}$. These nodes have a height of 1 and they both have 2 leaves. The confidence region of values of the leaves is $\mathcal{C} = B(\hat{\boldsymbol{\mu}}, \rho) \subseteq \mathbb{R}^4$. Notice that the set illustrated here is not a sphere, even though the confidence region $\mathcal{C}$ is.*

Interestingly, this shape is only visible at nodes of odd heights. If one considers a tree of depth 3, and looks at a node of depth 1, i.e. height 2, the set of plausible values for those nodes is again a sphere. However, if $\hat{\boldsymbol{\mu}}$ does not equal $\alpha\mathbf{1}$ for some $\alpha \in \mathbb{R}$, things become interesting.

Consider a tree of depth 3 and outdegree 2. The values of the leaves are uniformly sampled from $[0, 1]$. In Figure 4.10 we plot the plausible joint values for the two children of the root $n_1$ and $n_1$. If all leaf values were equal, then we would expect a plot like in Figure 4.9. However, we see that the set is more shaped like a box with rounded corners. When we increase the number of leaves, the edges of this set become more defined. Let us replace the subtrees rooted at $n_1$ and $n_2$ by trees of depth 2 and outdegree 20. Again, we uniformly sample leaf values from $[0, 1]$ and assume $\hat{\boldsymbol{\mu}} = \boldsymbol{\mu}$.




***(a)*** *Outdegree 2* ***(b)*** *Outdegree 20*

***Figure 4.10:*** *For an oracle tree of depth 3 we plot the plausible values of two nodes of depth 1, $n_1$ and $n_2$ for two different outdegrees. Note that the number of leaves in the subtrees rooted at $n_1$ and $n_2$ is the outdegree squared. Both are based on a spherical confidence region $\mathcal{C}$ with confidence $\delta = 0.01$ and an $n = 100$ samples. This creates a sphere of radius approximately $0.366$ for the outdegree 2 case, whereas the outdegree 20 case has a radius of approximately $2.92$. We see that the set with a high outdegree has straighter sides than the set with the low outdegree.*

These results suggest that the comparison between spherical and rectangular confidence regions might not hold up as well on internal nodes, especially if the outdegree is high. This is an interesting result, as in the previous section we have seen in Figure 4.8 that increasing the number of arms quickly decreases the number of samples per arms and that this effect is more pronounced of there sphere. However, the results from this section suggest that this effect might not as great for internal nodes.

# Chapter 5

# Building the SCR-MCTS algorithm

In this chapter we will introduce the SCR-MCTS algorithm with the goal of improving the FindTopWinner algorithm, which we introduced earlier in Chapter 3. We start by introducing the CR-MCTS meta-algorithm that works with confidence regions of any shape. In Chapter 4 we gave some motivation on why it is interesting to look at spherical confidence regions instead of rectangular ones. With this meta-algorithm we will look at how to precisely implement the sampling, pruning, stopping and selection rules when using spherical confidence regions. Together, these will form the algorithm introduced in this thesis.

## 5.1 Generalization

Algorithms in the bandit literature typically consist of the following tree main rules:

- **Sampling rule** How many samples are taken from each leaf and when do we inspect the samples?
- **Stopping rule** How and when do we return an answer?
- **Recommendation rule** What answer do we return?

With the added tree structure in the MCTS problem that we solve, we add a fourth rule:

- **Pruning rule** How do we decide which nodes to prune from the tree?

Our meta-algorithm, Confidence Region-MCTS, or CR-MCTS, implements these rules as can be seen in Algorithm 4. The algorithm works in multiple iterations, which we will call **epochs**. Furthermore, as can be seen in the Sample subalgorithm, in each epoch $m$, we create a confidence region with a certain confidence $\delta_m$ and a size parameter $\epsilon_m$. For a sphere, this size parameter is the radius of the sphere, whereas for a box it is the length of the sides. In the case of a sphere, we will also denote this parameter as $\rho_m$. The confidence and size parameters must be known beforehand and are thus not adaptive. They are the input sequence $(\delta_m)_{m\geq 1}$ and $(\epsilon_m)_{m\geq 1}$.

**Algorithm 4** The CR-MCTS algorithm is the main algorithm that calls the stopping, sampling, pruning en recommendation rules.

**function** CR-MCTS($\mathcal{T}, (\epsilon_m)_{m\geq 1}, (\delta_m)_{m\geq 1}$)
    $m \leftarrow 1$
    **while** not STOP() **do**
        SAMPLE($\epsilon_m, \delta_m$)
        PRUNE(root)
        $m \leftarrow m + 1$
    **end while**
    **return** RECOMMEND()
**end function**

**Algorithm 5** The SAMPLE subalgorithm samples leaves and constructs a confidence region.

**procedure** SAMPLE($\epsilon_m, \delta_m$)
    Sample each leaf $\ell \in \text{leaves}_m(\mathcal{T})$ such that $\mathcal{C}_m$ is a confidence region with confidence $\delta_m$ and size parameter $\epsilon_m$.
**end procedure**

**Algorithm 6** The PRUNE subalgorithm recursively traverses the tree, pruning nodes that are implausible to be winners as defined in Definition 3.

**procedure** PRUNE($u$)
    **for** $c \in \text{C}(u)$ **do**
        **if** not exists $\boldsymbol{\mu} \in \mathcal{C}_m$ such that $c$ wins **then**
            $\text{C}(u) = \text{C}(u) \setminus \{c\}$
        **else**
            PRUNE($c$)
        **end if**
    **end for**
**end procedure**

**Algorithm 7** The STOP rule returns true if a node can be recommnded.

**procedure** STOP
    **return** True iff RECOMMEND() returns at least one node.
**end procedure**

**Algorithm 8** The RECOMMEND subalgorithm returns nodes that are at most $\epsilon$-suboptimal for every plausible $\boldsymbol{\mu}' \in \mathcal{C}_m$

```
procedure RECOMMEND
    for u ∈ C(root) do
        if for all k ∈ siblings(u) and all μ' ∈ C : v_u(μ') − ε ≤ v_k(μ') then
            return u
        end if
    end for
    return Null
end procedure
```

Now that we have defined the meta-algorithm and its subalgorithms, we implement the meta-algorithm for spherical confidence regions. This algorithm will be called SCR-MCTS. We will start with the sampling rule, where the confidence regions are constructed. Then we will move on to the pruning rule, where the essential problem is to determine whether there exists a plausible model $\boldsymbol{\mu}' \in \mathcal{C}_m$ such that a node $u$ wins or not. Furthermore, we will look at the recommendation rule. As we will see, this problem is quite similar to pruning. Finally, we will look at the stopping rule and prove the correctness of the SCR-MCTS algorithm.

## 5.2 Sampling rule

The main constraint on the sampling rule is that we need to pick the sequence $(\delta_m)_{m\geq 1}$ such that the algorithm has $\delta$ confidence. Since the algorithm will work with multiple epochs in which we sample the leaves and then inspect them, we have to make sure that the confidence regions created in each epoch, i.e. all $\mathcal{C}_m$, will be simultaneously true with probability at least $1-\delta$. To ensure this, we make use of the union bound. Indeed, for each epoch $m$, let $\mathcal{C}_m$ be the confidence region created. Then we have the following constraint

$$\mathbb{P}\left(\exists m : \boldsymbol{\mu} \notin \mathcal{C}_m\right) \leq \sum_{t=1}^{\infty}(1 - \mathbb{P}(\boldsymbol{\mu} \in \mathcal{C}_m)) = \sum_{t=1}^{\infty} \delta_m \leq \delta. \tag{5.1}$$

As we will see in Section 5.6, this is the only constraint on the correctness of the algorithm. However, it is not sufficient for sampling efficiently. The confidence region should likely at least shrink in size each epoch. With a small confidence region, we can prune more nodes, meaning that in future epochs, there are fewer leaves to sample. However, if we construct a small confidence region, this will require more samples per leaf. This shows that there is a delicate balance between the radius $\rho_m$ of $\mathcal{C}_m$, the number of samples taken $n_m$, and the confidence level $\delta_m$. Of course, it is not allowed to take these values any way you like. For example, we cannot create a confidence region using a small number of samples while having high confidence and small size.

However, we can fix two of the three variables. Let each oracle $k$ sample from a normal distribution with mean $\mu_k$ and variance $\sigma^2$. We denote the estimate of each $\mu_k$ by $\hat{\mu}_{k,m}$ and the vector of all estimates by $\hat{\boldsymbol{\mu}}_m$. Let $Z_m = n_m/\sigma^2||\hat{\boldsymbol{\mu}}_m - \boldsymbol{\mu}||^2$. Then $Z_m$ is $\chi^2_K$-distributed where, $K$ is the number of leaves, and we have,

$$\mathbb{P}(||\hat{\boldsymbol{\mu}}_m - \boldsymbol{\mu}||^2 \leq \rho_m^2) = \mathbb{P}(Z_m \leq n_m\rho_m^2/\sigma^2) = F_{Z_m}(n_m\rho_m^2/\sigma^2) = 1 - \delta_m, \tag{5.2}$$

where $F_{Z_m}$ is the cumulative distribution function of the $\chi^2_K$-distribution. This gives us a method to fix two of the variables, and get a valid value for the third variable.

We have seen the constraints on picking the variables $\delta_m, \rho_m$ and $n_m$. Let us now formulate a strategy for selecting these values. The strategy we use is close to the FindTopWinner strategy. Just like FindTopWinner, we will halve the confidence level each epoch $m$. That is, $\delta_m = \delta/2^m$. It is easily checked that this geometric sequence indeed sums up to $\delta$ for all $m \geq 1$. Next, let us pick the radii of all spheres $\mathcal{C}_m$. We start with the unit sphere, so $\rho_1 = 1$. Then in each round we halve the radius, i.e., $\rho_{m+1} = \rho_m/2$. This is closest to the sampling rule of FindTopWinner, since in that algorithm the size of the intervals is halved each epoch. This can be seen in line 5 of Algorithm 1. A small radius for the confidence region means that we can prune more nodes, because fewer plausible models exist. However, this comes at a cost, since more samples per leaf are required. Note that even though we fix the starting radius at 1, any positive radius is allowed. It is possible that an other value is better, but this has not been analyzed. In Chapter 6 we will show empirical results for different starting radii. These results suggest that the starting radius does not gravely affect the total number of samples taken.

In conclusion, we have a confidence $\delta_m = \delta/2^m$ and radius $\rho_m = 1/2^{m-1}$ in each epoch $m$, which we use to solve for the number of samples,

$$n_m = 4^{m-1}\sigma^2 F_{Z_m}^{-1}(1 - \delta/2^m). \tag{5.3}$$

This concludes the sampling rule. Next, we will look at the pruning rule.

## 5.3 Pruning rule

Typically a game tree contains many nodes that are states that would never be chosen by a rational player. If we know that some node is not favorable, then we should not investigate it or the subtree rooted at that node. But how do we know if a node is not favorable?

Recall that in the Sample subalgorithm (Algorithm 5) we construct a spherical confidence region $\mathcal{C}_m$ for each epoch $m$. In this setting we say that a node $u$ is not interesting if there is no plausible $\boldsymbol{\mu}'$ in the confidence region $\mathcal{C}_m$ such that node $u$ has a better value than its siblings. In other words, we consider it to be implausible that $u$ can win. In that case $u$ is pruned by the Prune subalgorithm.

If the parent of $u$ is a max node, this means that we can prune $u$ if there is no $\boldsymbol{\mu}' \in \mathcal{C}$ such that for all $s \in \text{siblings}(n)$ we have $v_u(\boldsymbol{\mu}') > v_s(\boldsymbol{\mu}')$. This happens if and only if there is no $\boldsymbol{\mu}' \in \mathcal{C}$ such that $v_u(\boldsymbol{\mu}') > \max_{s \in \text{siblings}(n)} v_s(\boldsymbol{\mu}')$. With this insight, we can replace the children with a new max node, call it $v$, that contains all the original siblings of $u$ as its children. This will make it easier to decide if a node is interesting or not, since we now only have to compare it with one other node, $v$, instead of with each of the siblings of $u$. See Figure 5.1 for an example of replacing siblings with a new node.

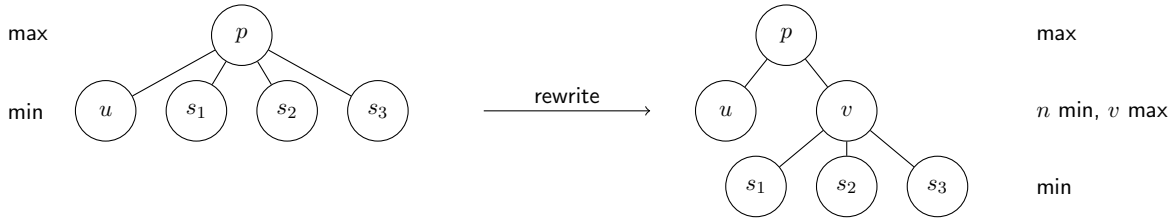



***Figure 5.1:*** *The tree rooted at $p$ can be rewritten be replacing children $s_1, s_2$ and $s_3$ by the max node $v$ that has $s_1, s_2$ and $s_3$ as its children. Note that none of the values in the tree are altered by this operation.*

We now keep $u$ if there exists a plausible model $\boldsymbol{\mu}' \in \mathcal{C}$ such that

$$v_u(\boldsymbol{\mu}') > v_v(\boldsymbol{\mu}'). \tag{5.4}$$

If not, then we prune node $u$. We now observe that $\boldsymbol{\mu}' \in \mathcal{C}$ if and only if

$$\sum_{\ell \in \text{leaves}(\mathcal{T}_u)} ||\mu'_\ell - \hat{\mu}_\ell||^2 + \sum_{\ell \in \text{leaves}(\mathcal{T}_v)} ||\mu'_\ell - \hat{\mu}_\ell||^2 \leq \rho^2, \tag{5.5}$$

where $\mathcal{T}_u$ and $\mathcal{T}_v$ denote the subtrees rooted at $u$ and $v$ respectively. From another point of view, $\boldsymbol{\mu}_u$ and $\boldsymbol{\mu}_v$ can be seen as being lower dimensional vectors that live in spheres with each a fraction of the total radius $\rho$. In other words, there is some $\alpha \in [0, 1]$ such that

$$\sum_{\ell \in \text{leaves}(\mathcal{T}_u)} ||\mu'_\ell - \hat{\mu}_\ell||^2 \leq \alpha\rho^2 \tag{5.6}$$

and

$$\sum_{\ell \in \text{leaves}(\mathcal{T}_v)} ||\mu'_\ell - \hat{\mu}_\ell||^2 \leq (1-\alpha)\rho^2. \tag{5.7}$$

We define the greatest or smallest value that a node $u$ attains in a confidence region, as the upper and lower confidence bounds on the value of $u$. We will use this definition to rewrite (5.4) in terms of confidence bounds.

**Definition 8.** *Let $\mathcal{T}$ be an oracle tree, $n \in \text{nodes}(\mathcal{T})$ and let $\mathcal{C} = B(\hat{\boldsymbol{\mu}}, \rho) \subseteq \mathbb{R}^m$ be a spherical confidence region centered at the empirical estimate of the value of the leaves $\hat{\boldsymbol{\mu}}$ with radius $\rho$. Then the **upper and lower confidence bounds** on the value of $u$ are defined by,*

$$\text{UCB}_u(\rho^2) = \max_{\boldsymbol{\mu} \in B(\hat{\boldsymbol{\mu}}, \rho)} v_u(\boldsymbol{\mu}), \tag{5.8}$$

$$\text{LCB}_u(\rho^2) = \min_{\boldsymbol{\mu} \in B(\hat{\boldsymbol{\mu}}, \rho)} v_u(\boldsymbol{\mu}). \tag{5.9}$$

Consequently, if the confidence region is a sphere of radius $\rho$, to know if there is a plausible $\boldsymbol{\mu}' \in \mathcal{C}$ such that $v_u(\boldsymbol{\mu}') > v_v(\boldsymbol{\mu}')$, we can ask if there exists an $\alpha \in [0, 1]$ such that

$$\text{UCB}_u(\alpha\rho^2) = \max_{\boldsymbol{\mu}' \in B(\hat{\boldsymbol{\mu}}, \sqrt{\alpha}\rho)} v_u(\boldsymbol{\mu}') > \min_{\boldsymbol{\mu}' \in B(\hat{\boldsymbol{\mu}}, \sqrt{1-\alpha}\rho)} v_v(\boldsymbol{\mu}') = \text{LCB}_v((1-\alpha)\rho^2). \tag{5.10}$$

If this $\alpha$ does not exist, we can prune node $u$. We employ a grid search to check if Equation (5.10) holds for all $\alpha$ in $[0, 1]$. Note that this is not strictly safe, because it might be possible that the expression is not true for some value we did not evaluate. We address this by measuring the error rate in Chapter 6 and will see that the error rate indeed stays below the chosen $\delta$.

We have now expressed the pruning decision in terms of confidence bounds. In the next section we will develop a method of computing the confidence bound functions, allowing us to decide which nodes have to be pruned.

## 5.4 Confidence bounds

In the previous section we briefly introduced the concept of confidence bounds on the value of nodes in the oracle tree. Confidence bounds represent the range of values that are plausible for a node in the oracle tree if the spherical confidence region that was constructed has radius $\rho$.

**Remark 4.** *Note that even though for any node $u \in \text{nodes}(\mathcal{T})$ the interval $[\text{LCB}_u(\rho^2), \text{UCB}_u(\rho^2)]$ is a confidence interval with confidence at most $\delta$, for any two nodes $n \neq n' \in \text{nodes}(\mathcal{T})$, $[\text{LCB}_u(\rho^2), \text{UCB}_u(\rho^2)] \times [\text{LCB}_{u'}(\rho^2), \text{UCB}_{u'}(\rho^2)]$ is not the optimal confidence region with confidence $\delta$. For example, take distinct $u, u' \in \text{leaves}(\mathcal{T})$. The ball $B(\hat{\mu}, \rho^2) \subset \mathbb{R}^2$ is a confidence region for these two nodes and is strictly contained in $[\text{LCB}_u(\rho^2), \text{UCB}_u(\rho^2)] \times [\text{LCB}_{u'}(\rho^2), \text{UCB}_{u'}(\rho^2)]$. This is illustrated in Figure 5.2.*

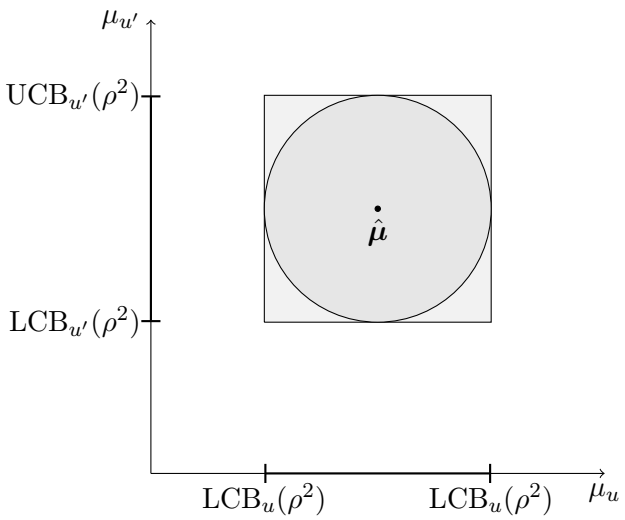



***Figure 5.2:*** *For any two leaves $u, u'$, the set $[\mathrm{LCB}_u(\rho^2), \mathrm{UCB}_u(\rho^2)] \times [\mathrm{LCB}_{u'}(\rho^2), \mathrm{UCB}_{u'}(\rho^2)]$ is not the optimal confidence region. This region is illustrated by the light gray square. Instead, the sphere $B(\hat{\boldsymbol{\mu}}, \rho^2)$ is a better confidence region that is strictly included in the box.*

*In other words, it is plausible that either $u$ or $u'$ deviates a lot from the empirical estimate, but it is implausible that both do.*

We have defined confidence bounds and in the previous section we have seen how they are used in the pruning subalgorithm. But how do we compute the confidence bounds? In the next subsection we will compute bounds of leaf nodes. In later sections we will compute confidence bounds of internal nodes.

### 5.4.1 Confidence bounds on leaves

For a leaf it is possible to explicitly compute the confidence bounds on its value. Let $\ell \in \mathrm{leaves}(\mathcal{T})$ and let $\mathcal{C}$ a spherical confidence region. We have

$$\mathrm{UCB}_\ell(\rho^2) = \max_{\boldsymbol{\mu} \in \mathcal{C}} v_\ell(\boldsymbol{\mu}) \tag{5.11}$$

$$= \max_{\boldsymbol{\mu} \in \mathcal{C}} \mu_\ell \tag{5.12}$$

$$= \hat{\mu}_\ell + \rho. \tag{5.13}$$

Similarly for the lower confidence bound, we get $\mathrm{LCB}_\ell(\rho^2) = \hat{\mu}_\ell - \rho$. Visually, this corresponds to Figure 5.3. We start in $\hat{\boldsymbol{\mu}}$ and move towards the edge of the circle to maximize the value.

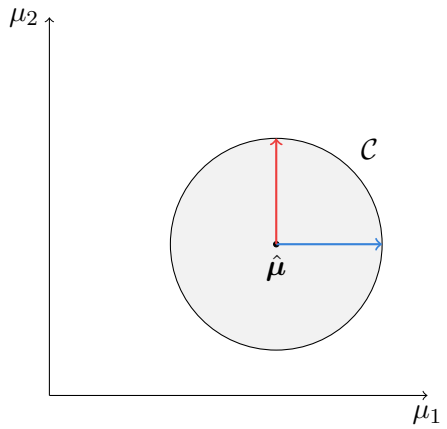



***Figure 5.3:*** *To calculate the maximum value of $v_1(\boldsymbol{\mu})$ for $\boldsymbol{\mu} \in \mathcal{C}$, we follow the blue arrow to the edge of the sphere and find $\hat{\mu}_1 + \rho$. Similarly for $v_2(\boldsymbol{\mu})$ we follow the red arrow and find $\hat{\mu}_2 + \rho$.*

Now that we know how to compute confidence bounds on the values of leaves, let us look at how to compute bounds for nodes in the rest of the tree.

### 5.4.2 Recurrence relation

For an internal node it is not as straightforward to find the confidence bounds on the value. Luckily, it is possible to derive a recurrence relation. The confidence bounds of internal nodes are defined as a function of the confidence bounds of their children. If we the compute the confidence bounds on the leaves, we can compute the confidence bounds on nodes in the rest of the tree.

As we will see, there are two different relationships. First, there is a 'symmetric' case, where the upper bound is computed of a max node or the lower bound of a min node. And second, there is the 'asymmetric' case for the lower bound on a max node or the upper bound on a min node. In Lemma 2 we state the symmetric case and in Lemma 3 we state the asymmetric case.

**Lemma 2.** *Let $\mathcal{T}$ be an oracle tree and let $\mathcal{C} = B(\hat{\boldsymbol{\mu}}, \rho) \subseteq \mathbb{R}^m$ be a confidence region centered at the empirical estimate $\hat{\boldsymbol{\mu}}$ with radius $\rho$. Let* UCB *and* LCB *be defined as in Definition 8. If $u \in \text{nodes}(\mathcal{T})$ is a max node, then*

$$\text{UCB}_u(\rho^2) = \max_{c \in \text{C}(u)} \text{UCB}_c(\rho^2). \tag{5.14}$$

*If $u$ is a min node, then*

$$\text{LCB}_u(\rho^2) = \min_{c \in \text{C}(u)} \text{LCB}_c(\rho^2). \tag{5.15}$$

*Proof.* Assume that $u$ is a max node. For brevity in notation, $c \in \text{C}(u)$ is replaced with $c$ in the subscripts. We can write $\text{UCB}_u$ as

$$\text{UCB}_u(\rho^2) = \max_{||\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}||^2 \leq \rho^2} \max_c v_c(\boldsymbol{\mu}) \tag{5.16}$$

$$= \max_{\sum_c \rho_c^2 \leq \rho^2} \max_{||\boldsymbol{\mu}_{c_1} - \hat{\boldsymbol{\mu}}_{c_1}|| \leq \rho_{c_1}^2} \cdots \max_{||\boldsymbol{\mu}_{c_K} - \hat{\boldsymbol{\mu}}_{c_K}|| \leq \rho_{c_K}^2} \max_c v_c(\boldsymbol{\mu}_c) \tag{5.17}$$

We can change the order of the maxima, which yields

$$= \max_{\sum_c \rho_c^2 \leq \rho^2} \max_c \max_{||\boldsymbol{\mu}_{c_1} - \hat{\boldsymbol{\mu}}_{c_1}|| \leq \rho_{c_1}^2} \dots \max_{||\boldsymbol{\mu}_{c_K} - \hat{\boldsymbol{\mu}}_{c_K}|| \leq \rho_{c_K}^2} v_c(\boldsymbol{\mu}_c) \tag{5.18}$$

$$= \max_{\sum_c \rho_c^2 \leq \rho^2} \max_c \mathrm{UCB}_c(\rho_c^2) \tag{5.19}$$

$$= \max_c \mathrm{UCB}_c(\rho^2). \tag{5.20}$$

For a min node the proof is similar. □

Next, in Lemma 3 we state the asymmetric case.

**Lemma 3.** *Let $\mathcal{T}$ be an oracle tree and let $\mathcal{C} = B(\hat{\boldsymbol{\mu}}, \rho) \subseteq \mathbb{R}^m$ be a confidence region centered at the empirical estimate $\hat{\boldsymbol{\mu}}$ with radius $\rho$. Let* UCB *and* LCB *be defined as in Definition 8. If $u \in \mathrm{nodes}(\mathcal{T})$ is a max node, then*

$$\mathrm{LCB}_u(\rho^2) = \min_{\sum_{c \in \mathrm{C}(u)} \rho_c^2 \leq \rho^2} \max_{c \in \mathrm{C}(u)} \mathrm{LCB}_c(\rho_c^2), \tag{5.21}$$

*where in the first minimum the variables are $\rho_c$ for each $c \in \mathrm{C}(u)$. If $u$ is a min node, then*

$$\mathrm{UCB}_u(\rho^2) = \max_{\sum_{c \in \mathrm{C}(u)} \rho_c^2 \leq \rho^2} \min_{c \in \mathrm{C}(u)} \mathrm{UCB}_c(\rho_c^2). \tag{5.22}$$

*Proof.* Let $u$ be a min node. Then we have for its upper confidence bound

$$\mathrm{UCB}_u(\rho^2) = \max_{||\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}||^2 \leq \rho^2} \min_c v_c(\boldsymbol{\mu}_c) \tag{5.23}$$

$$= \max_{\sum_c \rho_c^2 \leq \rho^2} \max_{\substack{||\boldsymbol{\mu}_c - \hat{\boldsymbol{\mu}}_c|| \leq \rho_c^2 \\ \forall c}} \min_c v_c(\boldsymbol{\mu}_c) \tag{5.24}$$

$$= \max_{\sum_c \rho_c^2 \leq \rho^2} \min_c v_c(\boldsymbol{\mu}^*(c)) \tag{5.25}$$

$$= \max_{\sum_c \rho_c^2 \leq \rho^2} \min_c \mathrm{UCB}_c(\rho_c^2), \tag{5.26}$$

where

$$\boldsymbol{\mu}^*(c) = \arg\max_{||\boldsymbol{\mu}_c - \hat{\boldsymbol{\mu}}_c||^2 \leq \rho_c^2} v_c(\boldsymbol{\mu}_c). \tag{5.27}$$

For a max node, the proof is similar. □

With these lemmas we have some insight into how to compute the confidence bounds on internal nodes.

Furthermore, as we will see next, the function $\rho \mapsto \mathrm{UCB}_u(\rho^2)$ is a piecewise continuous function that consists of pieces with the same parametric form. We will prove this by showing that leaves satisfy this lemma and then by showing that it holds by induction.

**Lemma 4.** *Let $\mathcal{T}$ be an oracle tree and $\mathcal{C} = B(\hat{\boldsymbol{\mu}}, \rho) \subseteq \mathbb{R}^m$ a spherical confidence region centered at the empirical estimate $\hat{\boldsymbol{\mu}}$ with radius $\rho$ and let $u \in \mathrm{nodes}(\mathcal{T})$. Then $\rho \mapsto \mathrm{UCB}_u(\rho^2)$ is a piecewise continuous function where its pieces are given by*

$$\rho \mapsto \frac{M + \sqrt{M^2 - LN + L\rho^2}}{L}, \tag{5.28}$$

*for some* $M, N, L \in \mathbb{R}$. *Analogously, the function* $\rho \mapsto \mathrm{LCB}_u(\rho^2)$ *is parameterized by*

$$\rho \mapsto \frac{M - \sqrt{M^2 - LN + L\rho^2}}{L}, \tag{5.29}$$

*for some* $M, N, L \in \mathbb{R}$. *In both cases we have*

$$M = \sum_{\ell \in \mathcal{L}'} \hat{\mu}_\ell, \quad N = \sum_{\ell \in \mathcal{L}'} \hat{\mu}_\ell^2, \quad L = |\mathcal{L}'| \tag{5.30}$$

*for some subset* $\mathcal{L}' \subseteq \mathrm{leaves}(\mathcal{T})$.

*Proof.* We will present a proof by induction. We start with the base case.

**Base case**

We have seen that for any leaf $\ell \in \mathrm{leaves}(\mathcal{T})$, the confidence bounds are given by $\mathrm{UCB}_\ell(\rho^2) = \hat{\mu}_\ell + \rho$ and $\mathrm{LCB}_\ell(\rho^2) = \hat{\mu}_\ell - \rho$. Then the LCB and UCB satisfy Lemma 4 since substituting $M = \hat{\mu}_u$, $N = \hat{\mu}_u^2$ and $L = 1$ yields the correct confidence bounds. This proves the base case.

**Inductive step**

We continue with the inductive step. We only show the proof for upper confidence bounds, since the proof for lower confidence bounds is analogous. Let $u \in \mathrm{nodes}(\mathcal{T})$ be a node in the oracle tree $\mathcal{T}$. Assume that the lemma is true for all descendants of $u$. We will now prove the inductive step for the symmetric case.

Assume that $u$ is a max node. Then with Lemma 2 we see that $\mathrm{UCB}_u$ is the pointwise maximum of the upper confidence bound of its children. Then this directly implies that $\mathrm{UCB}_u$ has the same parametric pieces as $\mathrm{UCB}_c$ for each child $c$ of $u$. If $u$ is a min node the proof is similar, but then we take the pointwise minimum instead of the pointwise maximum. This shows the inductive step for the symmetric case.

For the asymmetric case, assume that $u$ is a min node. Then with Lemma 3 we see that $\mathrm{UCB}_u$ equals

$$\mathrm{UCB}_u(\rho^2) = \max_{\sum_{c \in \mathrm{C}(u)} \rho_c^2 \leq \rho^2} \min_{c \in \mathrm{C}(u)} \mathrm{UCB}_c(\rho_c^2). \tag{5.31}$$

Now let $\{\rho'_c\}$ be the set of optimizing radii in the first maximum. We then have for this set $\{\rho'_c\}$,

$$\mathrm{UCB}_u(\rho^2) = \min_{c \in \mathrm{C}(u)} \mathrm{UCB}_c((\rho'_c)^2). \tag{5.32}$$

Since the number of children of $u$ is finite, there exists a child node $c' \in \mathrm{C}(u)$ such that

$$\mathrm{UCB}_u(\rho^2) = \mathrm{UCB}_{c'}((\rho'_{c'})^2). \tag{5.33}$$

By the induction hypothesis, $\mathrm{UCB}_{c'}((\rho'_{c'})^2)$ can be expressed in the parametric form (5.28). Consequently, for each $\rho \geq 0$, the confidence bound $\mathrm{UCB}_u(\rho^2)$ can be expressed in the parametric form.

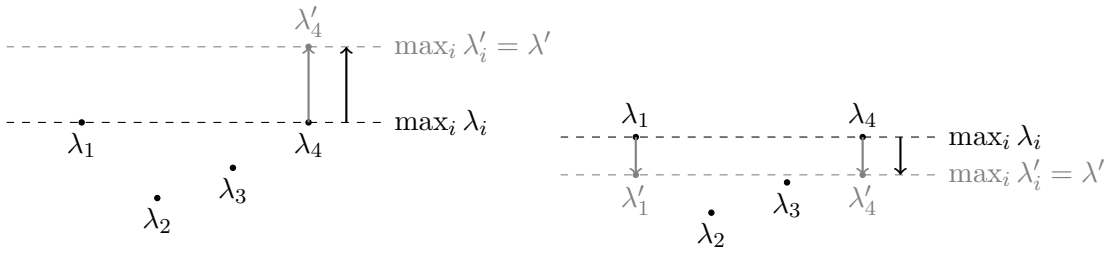
Since the inductive step holds for all internal nodes, and we have shown that the base case holds for all leaf nodes, the result follows. □

### 5.4.3 Confidence bounds on internal nodes

As mentioned before it is less straightforward to calculate the confidence bounds of internal nodes. We have seen in Lemmas 2 and 3 that the bounds depend on all their descendants. We made a distinction between two cases. First the symmetric case for calculating upper confidence

bounds on max nodes and lower confidence bounds on min nodes. Secondly, the asymmetric case for lower bounds on max nodes and upper bounds on min nodes.

An important insight is that for a max node to get a higher value, just one child has to increase in value. On the other hand, for the same max node to get a lower value, all children that attain the highest value, should decrease value. This is illustrated in Figure 5.4.




***(a)*** *To increase* $\max_i \lambda_i$ *to* $\lambda'$*, only one of the lambdas has to increase. For the maximum the other values are not important.*

***(b)*** *To decrease* $\max_i \lambda_i$ *to* $\lambda'$*, both* $\lambda_1$ *and* $\lambda_4$ *must decrease by the same amount.*

***Figure 5.4:*** *It is easier to increase a maximum than it is to decrease a maximum.*

To see one consequence of this behavior, consider the following situation. We are given the oracle tree illustrated in Figure 5.5.

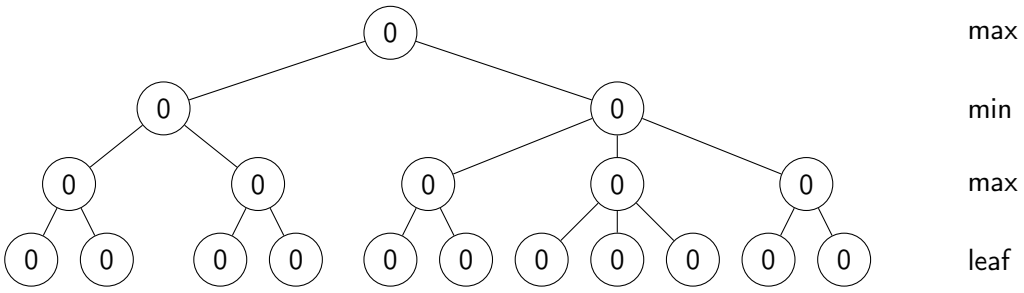



***Figure 5.5:*** *An oracle tree. The root node is a max node and the leaves of the tree have values* $\boldsymbol{\mu} = \mathbf{0}$*.*

We sample the leaves and we find that the estimate of the values of the leaves is the origin $\hat{\boldsymbol{\mu}} = \mathbf{0}$. The number of samples taken allow us to create a spherical confidence region $\mathcal{C}$ with radius $\sqrt{2}$ and center $\mathbf{0}$. Next, we want to know what the upper bound is on the value of the root node in this tree. In other words, we need to find a plausible model $\boldsymbol{\mu}' \in \mathcal{C}$ such that the value of the root $v_{\text{root}}(\boldsymbol{\mu}')$ is maximized. We can do this by adjusting the values of the leaves, subject to the constraint that $||\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}'||^2 \leq \rho^2$. This yields an optimal $\boldsymbol{\mu}'$ that takes the value 1 on two of the leaves and the value of 0 on the rest of the leaves, as illustrated in Figure 5.6. We see the same asymmetry as in Figure 5.4. Indeed, at depth 1 we see that just one node has its value changed since their parent is a max node. Conversely, at depth 2 we see that both node values are changed.

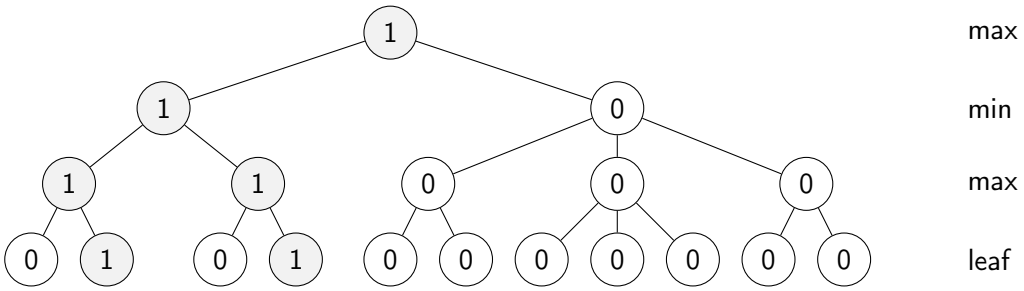



***Figure 5.6:*** *When finding a $\boldsymbol{\mu}' \in \mathcal{C}$ that maximizes the value of the root, the filled nodes must attain the same value. If any of them would be lower than 1, the value of the root would be lower than 1 too. Notice the asymmetry between min and max nodes. Min nodes activate all their children, but max nodes only have to activate one child.*

### Symmetric case

We use these insights to interpret the recursion relations from Lemma 2. We relate Figure 5.4a with the symmetric case in Lemma 2. Lemma 2 states that to find the upper bound on a max node $u$, we should take the maximum value of the upper confidence bounds of its children. If we know the full function $\mathrm{UCB}_c$ for each $c \in \mathrm{C}(u)$, then by taking the pointwise maximum of this set of functions, we obtain $\mathrm{UCB}_u$.

This new function, $\mathrm{UCB}_u$, can be expressed in pieces with the same parametric form, as is stated in Lemma 4. We express each piece as a triple, $(\rho_{\text{start}}, \rho_{\text{end}}, \mathcal{L})$, consisting of the start and end of the interval on which the pieces is defined, and all leaves that determine the $M, N, L$ parameters of the piece in Equation (5.30). For course, the start of the interval $\rho_{\text{start}}$ is less than the end of the interval $\rho_{\text{end}}$, and the end of the interval can be at infinity.

The entire function is then expressed as a vector of pieces. The central idea behind finding this explicit form is that we keep track of which piece is the greatest at some point $\rho$. By increasing $\rho$, either this piece stays the largest or it is overtaken by another piece. This can only happen if there is an intersection. By finding all interesting points such as intersections and endpoints of pieces, we can find the pointwise maximum function. At each interesting point $\rho$ we check which pieces are defined on $\rho$ and then we take the piece with the highest function value on $\rho$. If multiple pieces attain this function value, we keep the piece with the highest slope. Again, if this piece is not unique, we look at the piece with the highest second derivative value at $\rho$.

This is formalized in the MAXIMUM algorithm (Algorithm 9). It has two UCB functions as input. If the maximum of more functions needs to be taken, it is either trivially extendable to more input functions, or one can successively apply the MAXIMUM algorithm over all children.

**Algorithm 9** The MAXIMUM algorithm takes two upper confidence bounds functions as input and returns the pointwise maximum function of the input functions.

**function** MAXIMUM($\mathrm{UCB}_a, \mathrm{UCB}_b$)
  $\boldsymbol{\rho} \leftarrow$ FINDINTERSECTIONS($\mathrm{UCB}_a, \mathrm{UCB}_b$)
  $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} \cup$ GETENDPOINTS($\mathrm{UCB}_a$) $\cup$ GETENDPOINTS($\mathrm{UCB}_b$)
  $pieces \leftarrow \{\}$
  **for** $\rho \in \boldsymbol{\rho}$ ascending **do**
    $\boldsymbol{p} \leftarrow$ All pieces of $\mathrm{UCB}_a$ and $\mathrm{UCB}_b$ that have $\rho$ in their domain
    $\boldsymbol{p} \leftarrow \boldsymbol{p}$ where $p(\rho)$ is maximal ▷ Remove pieces with low function values at $\rho$
    $\boldsymbol{p} \leftarrow \boldsymbol{p}$ where $p'(\rho)$ is maximal ▷ Remove pieces with a low slope
    $\boldsymbol{p} \leftarrow \boldsymbol{p}$ where $p''(\rho)$ is maximal
    $newPiece \leftarrow p_1$
    $newPiece_{\text{start}} \leftarrow \rho$
    **if** $prevPiece$ exists **then** ▷ The previous piece ends where the new piece starts
      $prevPiece_{\text{end}} \leftarrow \rho$
    **end if**
    $pieces \leftarrow pieces \cup \{newPiece\}$
    $prevPiece \leftarrow newPiece$
  **end for**
  **return** *pieces*
**end function**

The MAXIMUM algorithm has two subalgorithms. The GETENDPOINTS subalgorithm only outputs the endpoints of the domains of all the pieces of the confidence bounds functions it takes as input.

**FINDINTERSECTIONS subalgorithm** The FINDINTERSECTIONS subalgorithm that finds intersections boils down to solving for $\rho$ in the equation

$$\frac{M_1 + \sqrt{M_1^2 - L_1 N_1 + L_1 \rho^2}}{L_1} = \frac{M_2 + \sqrt{M_2^2 - L_2 N_2 + L_2 \rho^2}}{L_2}. \tag{5.34}$$

We start with some small substitutions,

$$a = M_1^2 - L_1 N_1 \tag{5.35}$$
$$b = M_2^2 - L_2 N_2, \tag{5.36}$$

and we multiply by $L_1 L_2$. This yields

$$L_2 M_1 + L_2 \sqrt{a + L_1 \rho^2} = L_1 M_2 + L_1 \sqrt{b + L_2 \rho^2} \tag{5.37}$$
$$L_2 M_1 + L_2 \sqrt{a + L_1 \rho^2} - L_1 M_2 = L_1 \sqrt{b + L_2 \rho^2}. \tag{5.38}$$

Now we apply another substitution

$$c = L_2 M_1 - L_1 M_2. \tag{5.39}$$

At this point we make a case distinction between $c \neq 0$ and $c = 0$, because later we have to divide by $c$.

**Assume that** $c \neq 0$ We collect the terms and square the expression to get,

$$c^2 - 2cL_2\sqrt{a + L_1\rho^2} + L_2^2(a + L_1\rho^2) = L_1^2(b + L_2\rho^2) \tag{5.40}$$
$$2cL_2\sqrt{a + L_1\rho^2} = -L_1^2(b + L_2\rho^2) + c^2 + L_2^2(a + L_1\rho^2) \tag{5.41}$$
$$2cL_2\sqrt{a + L_1\rho^2} = c^2 - L_1^2 b + L_2^2 a + (L_2^2 L_1 - L_1^2 L_2)\rho^2. \tag{5.42}$$

We apply the substitution

$$d = c^2 - L_1^2 b + L_2^2 a \tag{5.43}$$
$$e = (L_2^2 L_1 - L_1^2 L_2). \tag{5.44}$$

Again, we collect terms and square the expression to get rid of the last square root,

$$2cL_2\sqrt{a + L_1\rho^2} = d + e\rho^2 \tag{5.45}$$
$$4c^2L_2^2(a + L_1\rho^2) = d^2 + 2de\rho^2 + e^2\rho^4 \tag{5.46}$$
$$0 = d^2 - 4c^2L_2^2 a + (2de - 4c^2L_2^2L_1)\rho^2 + e^2\rho^4. \tag{5.47}$$

To solve this equation for $\rho$, we have to divide by $e$. Therefore we make a case distinction between $e \neq 0$ and $e = 0$.

**Assume that** $e \neq 0$ Collecting terms we get

$$4c^2L_2^2(a + L_1\rho^2) = d^2 + 2de\rho^2 + e^2\rho^4 \tag{5.48}$$
$$\rho^2 = ABC(e^2, 2de - 4c^2L_2^2L_1, d^2 - 4c^2L_2^2 a) \tag{5.49}$$
$$\boxed{\rho = \pm\sqrt{ABC(e^2, 2de - 4c^2L_2^2L_1, d^2 - 4c^2L_2^2 a)}} \tag{5.50}$$

Note the abuse of notation, since the quadratic equation has multiple solutions. This means that in total there are 4 solutions for $\rho$. Since we are only concerned with non-negative values there are at most 2 feasible solutions.

**Assume that** $e = 0$ Continuing from (5.47), we get

$$2cL_2\sqrt{a + L_1\rho^2} = d \tag{5.51}$$
$$\boxed{\rho^2 = \left(\left(\frac{d}{2cL_2}\right)^2 - a\right)/L_1} \tag{5.52}$$

If $d/2CL_2 < 0$ then there are no solutions.

**Assume that** $c = 0$ Continuing from (5.39)

$$L_2^2(a + L_1\rho^2) = L_1^2(b + L_2\rho^2) \tag{5.53}$$
$$\boxed{\rho = \pm\sqrt{\frac{bL_1^2 - aL_2^2}{(L_2 - L_1)L_2L_1}}} \tag{5.54}$$

So we conclude that there are three interesting cases, where there is an intersection. Furthermore is it possible that the two pieces are equal. In that case we ignore that they technically

intersect, since the algorithm would have to choose between two identical pieces, so no matter which piece is chosen, the MAXIMUM algorithm returns the same function.

Summarizing, we created the MAXIMUM subalgorithm that uses FINDINTERSECTIONS as a subalgorithm. MAXIMUM find the pointwise maximum for a pair of piecewise functions as in Lemma 4. This algorithm allows us to obtain the upper confidence bound in the symmetric case. Equivalently, we could create a MINIMUM subalgorithm to find the lower confidence bound in the symmetric case. This shows how to completely solve the symmetric case. In the next subsection we show how confidence bounds can be found in the asymmetric case.

### Asymmetric case

The asymmetric case corresponds to Figure 5.4b and Lemma 3, that is, upper confidence bounds on values of min nodes or lower bounds on values of max nodes. Recall that Lemma 3 states that the lower confidence bound function of a max node $u$ is given by

$$\mathrm{LCB}_u(\rho^2) = \min_{\sum_{c\in \mathrm{C}(u)} \rho_c^2 \leq \rho^2} \max_{c\in \mathrm{C}(u)} \mathrm{LCB}_c(\rho_c^2). \tag{5.55}$$

First, note that since $\mathrm{LCB}_u$ is a decreasing function in $\rho$ for each node $c$. Indeed, the larger the radius of the spherical confidence region, the lower value node $c$ can attain. We therefore conclude that the condition $\sum_c \rho_c^2 \leq \rho^2$ holds with equality. We can view this problem as distributing the total radius $\rho$ over all the children of node $u$. Since we want the lowest value for $\max_c \mathrm{LCB}_c(\rho_c^2)$, we increase $\rho_m$ for some child $m$ until $\mathrm{LCB}_m(\rho_m^2) = \max_c \mathrm{LCB}_c(\rho_c^2)$.

**Two leaves example** As an example, let us have a look at the lower confidence bounds of a max node $u$ with two leaves as children. Suppose we sample the oracles at the leaves and find the estimate $\hat{\boldsymbol{\mu}} = (1, 0)$. We want to know the smallest attainable value of $u$, given a confidence region with radius $\rho$. At $\rho = 0$, the bound clearly is 1 since the whole confidence region is $\{\hat{\boldsymbol{\mu}}\}$, and the only value we attain is $v_u((1, 0)) = \max\{1, 0\} = 1$. For $\rho \in (0, 1]$ the confidence region is $B(\hat{\boldsymbol{\mu}}, \rho)$, and the best plausible model we can find is $\boldsymbol{\mu}' = (1 - \rho, 0)$. With this plausible $\boldsymbol{\mu}'$, the max node $u$ attains at the lowest a value of $1 - \rho$, so in the domain $\rho \in [0, 1]$ we find that $\mathrm{LCB}_u(\rho^2) = 1 - \rho$.

For the whole domain $\rho \geq 0$ the plausible model $\boldsymbol{\mu}' = (1 - \rho, 0)$ is insufficient, because $v_u(\boldsymbol{\mu}') = \max\{1 - \rho, 0\}$ equals 0 for any $\rho \geq 1$. Instead, to find a better $\boldsymbol{\mu}' \in \mathcal{C}$, we first note that both leaf values are necessarily the same. This means we have to solve,

$$\mu_1 = \mu_2 \tag{5.56}$$

subject to

$$||\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}||^2 \leq \rho^2. \tag{5.57}$$

Expanding the last equation, and subtituting $\mu_1 = \mu_2$ yields,

$$2\mu_1^2 - 2\sum_{i=1}^{2} \hat{\mu}_i \mu_1 + \sum_{i=1}^{2} \hat{\mu}_i^2 = \rho^2. \tag{5.58}$$

We solve for $\mu_1$ and find

$$\mu_1 = -\frac{M + \sqrt{M^2 - 2(N - \rho^2)}}{2}, \tag{5.59}$$

with $M = \sum_{i=1}^{2} \hat{\mu}_i = 1$ and $N = \sum_{i=1}^{2} \hat{\mu}_i^2 = 1$. Hence, on the complete domain $\rho \geq 0$ we find that the lower confidence bound for max node $u$ is

$$\mathrm{LCB}_u(\rho^2) = \begin{cases} 1 - \rho & \text{for } \rho < 1 \\ -\frac{1+\sqrt{2\rho^2-1}}{2} & \text{for } \rho \geq 1 \end{cases}. \tag{5.60}$$

This function is illustrated in figure 5.7 together with the optimal $\boldsymbol{\mu}$.

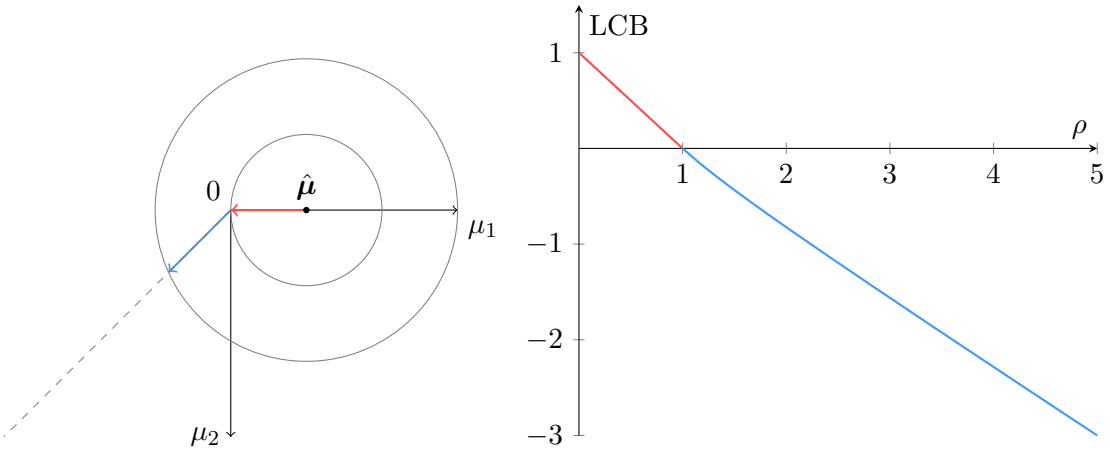



***Figure 5.7:*** *Illustrated is how we can find the lower confidence bound on a node with two children. The empirical estimate is $\hat{\boldsymbol{\mu}} = (1, 0)$. In the range $\rho \in [0, 1]$ the confidence bound scales linearly with slope 1. This corresponds with the red lines in both subfigures. For $\rho > 1$ both leaves are active and the confidence bound grows slower, asymptotically having slope $1/\sqrt{2}$. This corresponds with the blue lines in both subfigures.*

The asymmetric case for a min node $u$, that is, the upper confidence bound of $u$, has a similar derivation. Indeed, we still have a region $\rho \in [0, 1]$ where only one leaf attains the value of $u$, followed by a region where both leaves attain the value. It is easily verified that the upper confidence bound in this case equals

$$\mathrm{UCB}_u(\rho^2) = \begin{cases} \rho & \text{for } \rho < 1 \\ \frac{-1+\sqrt{2\rho^2+1}}{2} & \text{for } \rho \geq 1 \end{cases}. \tag{5.61}$$

**General case** We have seen how to compute the confidence bounds in a simple case with two leaves. Let us now generalize this approach to any internal node $u$. We can determine the value of $\mathrm{LCB}_u(\rho^2)$ for any $\rho \geq 0$, by using a sweeping approach. At $\rho = 0$, clearly $\mathrm{LCB}_u(\rho^2) = v_u(\hat{\boldsymbol{\mu}})$ since then the confidence region is the singleton set $\{\hat{\boldsymbol{\mu}}\}$. When we increase $\rho$, it is possible to lower the confidence bound. By how much depends on the behavior of the lower bound on the value of the child nodes of $u$. As we have seen before, the confidence bounds functions consist of pieces, where each piece has a set of leaves that are active.

From these pieces it is clear at what value of $\mathrm{LCB}_u(\rho)$ which leaves are active. Indeed, if we fix the lower bound at $\alpha$, then all the children should attain a value of at most $\alpha$. By intersecting $\rho \mapsto \mathrm{LCB}_c(\rho^2)$ by $\alpha$ we either find a piece that intersects $\alpha$, so these leaves are active, or we find no intersection, meaning that the child $c$ is not active and so none of its children are.

Using this method we have a function $\tau_c : \mathbb{R} \to 2^{\mathcal{L}}$ that tells us at what value of $u$ which leaves are active. Since the confidence bound function is monotone, we know in which order all pieces are, we only have to figure out the domain of each piece. We can do this with a sweeping strategy. We know that the first piece starts at $\mathrm{LCB}_u(0) = \hat{\mu}_u$. By intersecting the

parameterization of the first piece with the value of the start of the second piece, we can find out the endpoint of the first piece. This equals the starting point of the second piece. We repeat this process until there is only one piece left. This piece starts at the endpoint of the penultimate piece and its endpoint is at infinity.

With this procedure, we conclude the asymmetric case and therefore we are able to calculate the confidence bounds on any node in the tree. In the next section, we will look at the stopping and recommendation rules.

## 5.5 Stopping and recommendation rule

In this section we will develop the stopping rule and recommendation rule simultaneously. Indeed, we will stop once there exists an arm that can be recommended. But when can an arm be recommended? This happens when there exists an arm $u \in \mathrm{C}(\text{root})$ such that for all plausible models $\boldsymbol{\mu} \in \mathcal{C}$, the arm $u$ is the best arm with some slack $\epsilon$. This slack is always in favor of $u$. Assuming the root node is a min node, we can recommend it if the following holds

$$\forall \boldsymbol{\mu}' \in \mathcal{C}\ \forall k \in \text{siblings}(u) : v_u(\boldsymbol{\mu}') - \epsilon \leq v_k(\boldsymbol{\mu}'). \tag{5.62}$$

Using a similar trick as used in the pruning section (Section 5.3), we introduce a virtual node $v$ that has all siblings of $u$ as children. Instead of comparing with all siblings $k \in \text{siblings}(u)$, we now only compare with the virtual node $v$. This yields the following recommendation condition

$$\forall \boldsymbol{\mu}' \in \mathcal{C}' \ : v_u(\boldsymbol{\mu}') - \epsilon \leq v_v(\boldsymbol{\mu}'). \tag{5.63}$$

With a similar analysis as in Section 5.3 we can express this condition using confidence bounds. That is, we recommend $u$ if for every $\alpha \in [0,1]$ the following holds,

$$\mathrm{UCB}_u(\alpha\rho^2) - \epsilon \leq \mathrm{LCB}_v((1-\alpha)\rho^2). \tag{5.64}$$

We can check this using a simple grid search. We let $A \subset [0,1]$ be a finite subset and check if for all $\alpha \in A$ equation (5.64) holds. Note that this is not strictly safe, as there might be an $\alpha \in A \setminus [0,1]$ such that (5.64) does not hold, but then the algorithm erroneously stops.

At this point we have introduced all rules needed to implement the algorithm fully. In the next section, we will prove the correctness of the algorithm.

## 5.6 Correctness

In this section we will prove correctness of the algorithm, that is, we show that it is ($\epsilon$,$\delta$)-PAC and that the algorithm terminates.

**Lemma 5.** *The* SCR-MCTS *terminates at the latest in epoch $M$ such that $2\rho_M < \epsilon$.*

*Proof.* Observe that for each node $u$ and in each epoch $m$ we have that the range of plausible values is bounded as follows,

$$\{v_u(\boldsymbol{\mu}') | \boldsymbol{\mu}' \in \mathcal{C}_m\} \subseteq [\mathrm{LCB}_u(\rho_m^2), \mathrm{UCB}_u(\rho_m^2)] \subseteq [v_u(\hat{\boldsymbol{\mu}}_m) - \rho_m, v_u(\hat{\boldsymbol{\mu}}_m) + \rho_m], \tag{5.65}$$

where the first inclusion follows from the definition of the confidence bound functions and the second inclusion holds because the sphere $\mathcal{C}_M = B(\hat{\boldsymbol{\mu}}, \rho_M)$ is contained in the box $\{\boldsymbol{\mu}' \in \mathbb{R}^K : |\mu'_k - \hat{\mu}_k| \leq \rho_M \forall k \in [K]\}$. Now, assume without loss of generality that the tree $\mathcal{T}$ has a root node that is a min node. Furthermore, assume that the algorithm has not terminated yet, and

is in an epoch $M$ such that $2\rho_M < \epsilon$. Let $u$ be the node such that its estimated value $v_u(\hat{\boldsymbol{\mu}}'_M)$ is not greater than any of its siblings, in other words, it has the best estimated value. Then, by Equation 5.65 we have for any $u' \in \text{siblings}(u)$, that if

$$v_u(\hat{\boldsymbol{\mu}}) - \epsilon + \rho_M \leq v_{u'}(\hat{\boldsymbol{\mu}}) - \rho_M, \tag{5.66}$$

then we have that for all plausible $\boldsymbol{\mu}' \in \mathcal{C}_M$, $v_u(\boldsymbol{\mu}') - \epsilon \leq v_{u'}(\boldsymbol{\mu}')$. By our choice of epoch $M$ and child node $u$, we have for all $u' \in \text{siblings}(u)$,

$$v_u(\hat{\boldsymbol{\mu}}) - \epsilon + 2\rho < v_u(\hat{\boldsymbol{\mu}}) \leq v_{u'}(\hat{\boldsymbol{\mu}}). \tag{5.67}$$

Therefore, we find that in epoch $M$, the node $u$ must be recommended and the algorithm terminates. The result follows. □

Note that a consequence of Lemma 5 is that the total number of samples is bounded. Indeed, each epoch a finite number of samples is taken per leaf, and since the number of epochs is finite, so is the total number of samples. We have seen that the SCR-MCTS algorithm always returns a node. In the following lemma, we will show that this node is at most $\epsilon$-suboptimal with a probability of at least $1 - \delta$.

**Lemma 6.** *Let $\mathcal{T}$ be an oracle tree with $\boldsymbol{\mu} \in \mathbb{R}^{\text{leaves}(\mathcal{T})}$ as values of oracles at the leaves and let $\epsilon, \delta \in (0, 1)$. The algorithm outputs a node $u \in \text{C}(\text{root})$ such that*

$$|v_u(\boldsymbol{\mu}) - v_{\text{root}}(\boldsymbol{\mu})| < \epsilon \tag{5.68}$$

*with probability at least $1 - \delta$.*

*Proof.* Since the algorithm only accepts a sequence $(\delta_m)_{m \geq 1}$ such that

$$\sum_{m \geq 1}^{\infty} \delta_m \leq \delta, \tag{5.69}$$

we have that with probability at least $1 - \delta$ that simultaneously for each epoch $m$ the confidence region $\mathcal{C}_m$ contains $\boldsymbol{\mu}$. For the rest of the proof, we will assume that this happens.

The pruning rule only prunes a node $u$ if no plausible model $\boldsymbol{\mu}' \in \mathcal{C}$ exists such that $u$ wins. Since we assume that in each epoch $m$ the true value $\boldsymbol{\mu}$ is in the confidence region $\mathcal{C}_m$, the node $u$ is never incorrectly pruned.

As a consequence, in a pruned tree the value of any node $n$, $v_u(\boldsymbol{\mu})$ is never altered, as either $u$ is an internal node and its value equals the value of a winning child node, or $u$ is a leaf node and its value is not dependent on other nodes.

Finally, we prove the correctness of the stopping and recommendation rule. Recall that we recommend a node $u$ if for each plausible $\boldsymbol{\mu}' \in \mathcal{C}_m$ and each node $k \in \text{siblings}(u)$ the node $u$ wins with some slack, i.e., $v_u(\boldsymbol{\mu}') - \epsilon \leq v_k(\boldsymbol{\mu}')$. Since we only recommend $u$ if this holds for every $\boldsymbol{\mu}' \in \mathcal{C}_m$, $u$ is only recommended if the condition also holds for the true $\boldsymbol{\mu}$ which we assume to be in $\mathcal{C}_m$. Therefore, only $\epsilon$-suboptimal nodes are returned by the algorithm. Since this happens with probability at least $1 - \delta$, we conclude that the algorithm is ($\epsilon$,$\delta$)-PAC. □

# Chapter 6

# Results

In this chapter we present experimental results of the SCR-MCTS algorithm we introduced in Chapter 5 and compare it with FindTopWinner and two other algorithms from the literature. We are interested in comparing the empirical results on the sample complexity. Furthermore, we examine whether SCR-MCTS is empirically ($\epsilon$,$\delta$)-PAC. Finally, we will look at how choosing a different starting radius affects the number of samples taken.

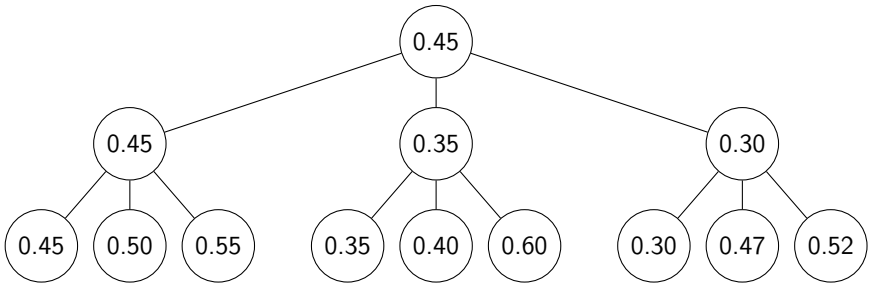
## 6.1 Depth 2 tree benchmark

We start by running a set of algorithms on an oracle tree that was used in experiments in a paper by Koolen and Kaufmann [9]. This paper introduces a class of algorithms BAI-MCTS and two algorithms within this class, namely LUCB-MCTS and UGapE-MCTS. These algorithms are designed to solve Problem 1 and are known to be ($\epsilon$,$\delta$)-PAC. The algorithms in this class use an adaptive approach, where each sample is inspected directly after the sample is taken. This is in contrast to our algorithm and FindTopWinner, where we take a batch of samples in each epoch and only inspect the samples at the end of the epoch.

The experiments we perform use stochastic rewards that are normally distributed with variance $\sigma^2 = 1/4$. We want to benchmark this tree with ($\epsilon$,$\delta$)-PAC with $\epsilon = 0, \delta = 0.1$. However, we follow [9] and set the ($\epsilon$,$\delta$)-PAC parameters to $\epsilon = 0$ and $\delta = 0.1 \cdot 9$ for the FindTopWinner, LUCB-MCTS and UGapE-MCTS algorithms. In [9] the $\delta$ parameter is multiplied by the number of leaves to undo the union bound over all leaves, which is considered to be too conservative. Recall that this union bound occurs when we upper bound the probability that event $A$ occurs, as defined in Definition 4. Since we do not apply a union bound over all leaves in SCR-MCTS, we do not multiply the confidence level with the total number of leaves and we set $\delta = 0.1$ for our algorithm.

Note that setting $\epsilon = 0$ requires a minor adjustment to FindTopWinner, by allowing it to run any number of epochs, whereas before the number of epochs was at most $\lceil \log_2(2/\epsilon) \rceil$, as can be seen on line 4 in Algorithm 1. One consequence of this, is that the reasoning behind FindTopWinner and SCR-MCTS terminating does not hold. However, in practice both algorithms always terminate.

We present the results in Figure 6.1 below. The values in the leaves are the mean rewards. Below each leaf, we state the number of samples taken by each algorithm.




| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| LUCB-MCTS | 849 | 210 | 91 | 688 | 252 | **21** | 207 | 43 | **19** |
| UGapE-MCTS | 785 | 207 | 90 | 723 | 256 | **21** | 211 | **42** | **19** |
| FTW | 2,890 | 2,871 | 2,448 | 2,876 | 2,873 | 423 | 1,158 | 746 | 568 |
| SCR-MCTS | **192** | **124** | **87** | **166** | **144** | 48 | **122** | 79 | 63 |

***Figure 6.1:*** *Empirical results averaged over 10,000 runs. The rewards are normally distributed with variance $\sigma^2 = 1/4$. We set $\epsilon = 0$ for all algorithms and $\delta = 0.1 \cdot 9$ for all algorithms except* SCR-MCTS*, where we set $\delta = 0.1$. Shown is the average total number of samples taken per leaf by* LUCB-MCTS *(2,415 samples and 0.43% error rate),* UGapE-MCTS *(2,395 samples and 0.44% error rate),* FindTopWinner *(16,853 samples, 0% error rate) and* SCR-MCTS *(1,025 samples, 2.73% error rate).*

We observe that our algorithm draws fewer samples in total than all competing algorithms by a significant margin. Furthermore, we see that the error rate of SCR-MCTS is also significantly higher than the other algorithms. Even though our error rate is higher, the error rate of 2.73% is well within the allowed error budget of $\delta = 0.1$. Another clear difference is that FindTopWinner does not seem to be able to choose between the four leaves with value $0.45, 0.50, 0.35$ and 0.40. This is in contrast with out own algorithm, which is mostly sampling from the nodes with values 0.45 and 0.35.

Since the SCR-MCTS error rate is relatively high, we also kept track of the number of times each child is picked. We see that over 10,000 runs the 0.45 arm is picked 9,727 times, the 0.35 arm is picked 197 times and the worst arm with value 0.30 is picked 76 times.

## 6.2 Depth 3 tree benchmark

We further test our algorithm on the depth 3 tree used in [9]. Similar as in the depth 2 tree in the previous section, we set the confidence level $\delta$ to $0.1 \cdot 27$ for FindTopWinner, LUCB-MCTS and UGapE-MCTS to cancel the union bound, and set $\delta = 0.1$ for SCR-MCTS. See Figure 6.2 on page 44 for the results. We again find that on this larger tree, we observe that our algorithm draws a significantly smaller number of samples from the leaves than FindTopWinner does. Furthermore, we observe that FindTopWinner does not make any mistakes, even though we set the confidence level to $\delta = 0.1 \cdot 27$. The SCR-MCTS does make mistakes, but it does so at a significantly lower rate than in the depth 2 tree. In the next section we will see that this is not necessarily specific to these two benchmark trees.

Notice that setting the confidence level $\delta$ to a value greater than 1, entails that there is no $(\epsilon,\delta)$-PAC guarantee on FindTopWinner. Also note that even though this value for $\delta$ is not a valid confidence level, all operations within the FindTopWinner algorithm are still valid.

Interestingly, SCR-MCTS samples the leaf with value 0.76 more than it samples the leaf with value 0.86. It is likely that the 0.86 leaf is pruned a bit more often than the 0.76. However, this does not directly lead to an incorrect answer. Indeed, if the parent of the 0.86 leaf is pruned, then the parent still has a sibling with the value of 0.87. If this sibling has not been pruned,

then the algorithm still outputs the correct value.

Furthermore, we see that the BAI-MCTS algorithms take fewer samples from most leaves than SCR-MCTS. Interestingly, in many runs they even take zero samples from some unpromising leaves. In particular, this happens when a sibling leaf has a relatively high value. This is not possible in SCR-MCTS, since we take a higher number of samples from each leaf in the first epoch before the samples are inspected. Taking a single sample is likely not worthwhile, as this would create such a large confidence region that no pruning decisions can be made. Conversely, the algorithms in the BAI-MCTS class take an adaptive approach, where after each sample the sample is inspected.

Even though SCR-MCTS cannot achieve very low sample counts on these leaves, this is not necessarily a problem, as these leaves together receive only a small fraction of the total number of samples.

***Figure 6.2:*** *10K runs. $\epsilon = 0, \delta = 0.1 \cdot 27$ for* FindTopWinner *and $\delta = 0.1$ for our algorithm. We find FTW (21,259 samples, 0% error rate), SCR-MCTS (4,307 samples, 0.25% error rate), UGapE-MCTS (1,584 samples, 0.75% error rate) and LUCB-MCTS (1,551 samples, 0.72% error rate)*

## 6.3 Randomly generated trees

We also run the algorithms on randomly generated trees. The results of the experiment are shown in Tables 6.1, 6.2 and 6.3. We construct trees with depths 2 to 4 and various outdegrees. The oracles are normally distributed with a variance of $\sigma^2 = 1/4$ and a mean that is uniformly sampled from $[0, 1]$. This resembles the experiment done in the original FindTopWinner paper [15], where 10,000 runs are done on a depth 3 outdegree 10 oracle tree, with leaf values uniformly drawn from $[0, 1]$. We formalize this procedure in Algorithm 10.

**Algorithm 10** The RandomTree procedure creates a random oracle tree $\mathcal{T}$ of depth $d$ and outdegree $w$. The root node is a max node if and only if the boolean value $isMax$ equals true. The oracles at the leaves are normally distributed with a randomly chosen mean and variance $\sigma^2 = 1/4$.

1: **procedure** RandomTree($isMax, d, w$)
2: $\mathcal{T} \leftarrow$ an oracle tree of depth $d$ and outdegree $w$. The root node is a max node if and only if $isMax$ is true.
3: Sample $\mathcal{U}(0,1)^{\text{leaves}(\mathcal{T})}$ and observe $\boldsymbol{\mu} \in [0,1]^{\text{leaves}(\mathcal{T})}$.
4: **return** Oracle tree $\mathcal{T}$ with leaf values $\boldsymbol{\mu}$. For each leaf $\ell$ the associated oracle samples from $\mathcal{N}(\mu_\ell, 1/4)$.
5: **end procedure**

In the original experiment, the samples are taken from a Bernoulli distribution, but since our algorithm works on the normal distribution we use normally distributed payoffs. In the following tables, the average total number of samples taken are denoted by $n_{\text{FTW}}$ and $n_{\text{SCR}}$ for the FindTopWinner algorithm and the SCR-MCTS algorithm respectively. Furthermore $err_{\text{FTW}}$ and $err_{\text{SCR}}$ indicate the average error, $|v_{\text{root}}(\boldsymbol{\mu}) - v_i(\boldsymbol{\mu})|$, where $i$ is the returned arm and $\boldsymbol{\mu}$ is the vector of values of the leaves. Finally, $m_{\text{FTW}}$ and $m_{\text{SCR}}$ indicate the fraction of runs where the error passes the accuracy threshold $\epsilon$. If the error is greater than $\epsilon$ we also call this a mistake.

| **Outdegree** | $n_{\text{FTW}}$ | $n_{\text{SCR}}$ | Improvement | $err_{\text{FTW}}$ | $err_{\text{SCR}}$ | $m_{\text{FTW}}$ | $m_{\text{SCR}}$ |
|---|---|---|---|---|---|---|---|
| 2 | 38027 | 165 | ×229.9 | 0.00009 | 0.00484 | 0.000 | 0.067 |
| 3 | 71230 | 612 | ×116.4 | 0.00011 | 0.00269 | 0.000 | 0.059 |
| 4 | 99960 | 1425 | ×70.1 | 0.00017 | 0.00168 | 0.000 | 0.046 |
| 6 | 189707 | 4937 | ×38.4 | 0.00030 | 0.00081 | 0.000 | 0.031 |
| 8 | 263534 | 12715 | ×20.7 | 0.00033 | 0.00064 | 0.000 | 0.023 |
| 10 | 362861 | 26867 | ×13.5 | 0.00040 | 0.00044 | 0.000 | 0.017 |
| 12 | 475649 | 49726 | ×9.6 | 0.00040 | 0.00026 | 0.000 | 0.007 |
| 14 | 614996 | 84896 | ×7.2 | 0.00048 | 0.00023 | 0.000 | 0.006 |

***Table 6.1:*** *Depth 2 tree with varying outdegree. $\epsilon, \delta = 0.01, 0.1$. Average over 5000 runs.*

| **Outdegree** | $n_{\text{FTW}}$ | $n_{\text{SCR}}$ | Improvement | $err_{\text{FTW}}$ | $err_{\text{SCR}}$ | $m_{\text{FTW}}$ | $m_{\text{SCR}}$ |
|---|---|---|---|---|---|---|---|
| 2 | 42623 | 423 | $\times$100.8 | 0.00007 | 0.00256 | 0.000 | 0.041 |
| 3 | 109309 | 3242 | $\times$33.7 | 0.00013 | 0.00067 | 0.000 | 0.022 |
| 4 | 202309 | 13649 | $\times$14.8 | 0.00020 | 0.00059 | 0.000 | 0.026 |
| 6 | 600572 | 114530 | $\times$5.2 | 0.00044 | 0.00019 | 0.000 | 0.004 |
| 8 | 1242682 | 548808 | $\times$2.3 | 0.00070 | 0.00011 | 0.000 | 0.000 |
| 10 | 2243143 | 1871750 | $\times$1.2 | 0.00082 | 0.00010 | 0.000 | 0.001 |

***Table 6.2:*** *Depth 3 tree with varying outdegree. $\epsilon, \delta = 0.01, 0.1$. Average over 2500 runs.*

| **Outdegree** | $n_{\text{FTW}}$ | $n_{\text{SCR}}$ | Improvement | $err_{\text{FTW}}$ | $err_{\text{SCR}}$ | $m_{\text{FTW}}$ | $m_{\text{SCR}}$ |
|---|---|---|---|---|---|---|---|
| 2 | 74279 | 1352 | $\times$54.9 | 0.00010 | 0.00112 | 0.000 | 0.034 |
| 3 | 242368 | 20703 | $\times$11.7 | 0.00024 | 0.00035 | 0.000 | 0.011 |
| 4 | 562534 | 163797 | $\times$3.4 | 0.00035 | 0.00013 | 0.000 | 0.002 |

***Table 6.3:*** *Depth 4 tree with varying outdegree. $\epsilon, \delta = 0.01, 0.1$. Average over 5000 runs.*

First, we observe that our algorithm improves the sample complexity of FindTopWinner by a large margin. The improvement is especially apparent for trees with low depth and a low outdegree. One possible explanation was touched upon in Section 4.2.3, where the limitations were discussed of the toy problem that was introduced to compare spherical and cubic confidence regions. When the number of leaves is higher, the set of plausible values of internal nodes becomes less sphere-shaped and looks more like a box. This could explain how in trees with more leaves, the sample complexity of FindTopWinner and our algorithm look more alike.

Furthermore, we observe that FindTopWinner never makes an error that is greater than the permitted accuracy $\epsilon = 0.01$. This is in agreement with the results from the experiments by Teraoka et al. [15]. The average error made by FindTopWinner does seem to increase with the outdegree. One possible explanation could be that with a higher outdegree, there are more arms for the algorithm to choose from. In that case, it is more likely that there exists another arm within the permitted accuracy $\epsilon$. This would cause a higher average error, but does not affect the number of mistakes.

On the other hand, we observe that our algorithm does make mistakes and this happens especially on smaller trees, both in depth and in outdegree. Even though there is a clear difference with FindTopWinner in this regard, the observed mistake rate is well below the permitted confidence $\delta = 0.1$. Indeed, for the smallest tree we find that the error rate is $m_{\text{SCR}} = 0.067$. Furthermore, we can show that this is a significant result. Since a mistake either happens or not, the estimate $m_{\text{SCR}}$ is a random variable bounded in $[0, 1]$. Therefore, we can apply Hoeffding's inequality (Corollary 1) on the estimate $m_{\text{SCR}}$. Indeed, for any $[0, 1]$-valued random variables $X_1, X_2, \ldots, X_n$, we have,

$$\mathbb{P}(|\bar{X} - \mathbb{E}[\bar{X}]| \geq \Delta) \leq 2\exp(-2n\Delta^2), \tag{6.1}$$

where $\bar{X}$ is the empirical mean $\bar{X} = 1/n \sum_{i=1}^{n} X_n$. Equating the right hand side to 0.01, we find that for the depth 2 tree with outdegree 2, that the error rate is $m_{\text{SCR}} = 0.067 \pm 0.023$ with 99% confidence.

Finally, we remark that possibly the more conservative results are due to a starting radius that is rather small for a high number of leaves. Currently we fix the starting radius at 1, as FindTopWinner similarly starts with $\epsilon = 1$ to construct a confidence region that is a cube with sides of $2\epsilon$. However, we note that in volume it is not similar. Indeed, in high dimensions

the volume of a sphere is very small compared to the volume of a hypercube [7]. As we will see in the next section for small trees this does not seem to have a major effect.

The general conclusion is that our algorithm empirically samples more efficiently than FindTopWinner in all our test cases. Furthermore, our algorithm seems to make better use of the confidence level $\delta = 0.1$. Indeed, it makes more mistakes but the fraction of mistakes is well within the error budget.

## 6.4 Starting radius

The sampling strategy used by SCR-MCTS in all experiments constructs a spherical confidence region in the first epoch with a radius $\rho_1 = 1$, as this is similar to FindTopWinner. We would like to know what the effect is of taking a different starting radius $\rho_1$. To answer this question, we run our algorithm on randomly generated trees of depth 2 and outdegree 2 to 4 and trees of depth 3 and outdegree 3. These trees are generated using the procedure described in Algorithm 10. We run the SCR-MCTS algorithm 200 times for starting radii between 0.1 and 10 and take the average total number of samples taken. We use an accuracy of $\epsilon = 0.01$ and a confidence of $\delta = 0.1$. The result is illustrated below in Figure 6.3.

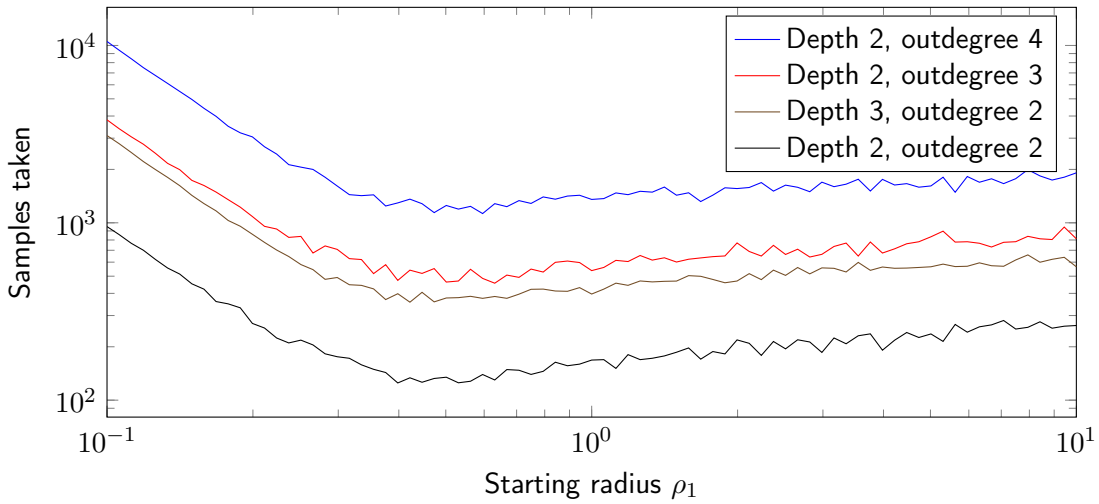



***Figure 6.3:*** *The average number of samples taken by* SCR-MCTS *on various randomly generated trees with $\epsilon = 0.01$ and $\delta = 0.1$. We see that when the starting radius is small, the number of samples taken becomes large. The number of samples taken is averaged over 200 runs for each starting radius $\rho_1$.*

We find that there is a certain sweet spot at which the number of samples is minimized. For the small trees that we tested this optimal starting radius $\rho_1$ seems to be quite close to the starting radius of $\rho_1 = 1$ that was used in previous results. Starting with a too small radius quickly increases the number of samples taken. In this regime the first confidence region estimates the values of the leaves very precisely. However, this requires us to take a lot of samples that are only inspected at the end of the epoch. The extra samples do not add much benefit if the required precision has been met. Therefore we expect this graph to continue rising rapidly as $\rho_1$ goes to 0.

On the right hand side of the graph we see that the average number of samples also increases. For large radii no pruning decision can be made as there is only a very inaccurate estimation of the value of the leaves. Consequently, we spend our $\delta$-budget on confidence regions that do not shrink the oracle tree and confidence regions that prune more leaves will have a greater sampling cost. We therefore expect that the sample complexity will rise.

# Chapter 7

# Conclusion

The introduced SCR-MCTS algorithm shows a significant improvement on small trees, taking an order of magnitude fewer samples than FindTopWinner. On larger trees, our algorithm still outperforms FindTopWinner, but while the number of samples taken is still significantly less, the difference is relatively smaller than for small trees. On a small benchmark tree of depth 2 we have also seen that SCR-MCTS outperforms other algorithms from the literature.

Furthermore, we observe that the number of mistakes made by our algorithm is significantly below the $\delta$-confidence in experiments on larger trees. This suggests there is still room for improvement here. In practice, this might mean that a user of this algorithms could consider using a larger $\delta$ while not losing much precision, but at the expense of losing theoretical ($\epsilon$,$\delta$)-PAC guarantees.

We suggest various possible new extensions of this research. First, the use of different shapes of confidence regions might yield interesting results, improving the current algorithm. Using a different shape could lead to better pruning decisions, although it is currently unclear what the effect would be. Different shapes might also allow for tighter confidence regions for oracles that are not (sub-)Gaussian.

Second, one might try and use multiple confidence regions per epoch for subtrees, instead of creating one large confidence region containing all nodes. A consequence is that we effectively apply SCR-MCTS on smaller subtrees and something more like FindTopWinner on the larger tree. To illustrate, imagine a tree with two childs node below the root, both containing a, possibly large, subtree each. If we construct two separate spherical confidence regions on the value of the leaves in these subtrees, then we have a square of possible values on the children of the root. The decisions that our algorithm would make on this square are equivalent to what FindTopWinner would do. Since SCR-MCTS seems to perform relatively better on small trees, this might improve the sample complexity.

Third, it might be possible to inspect samples directly after taking a single sample, instead of inspecting the samples after a batch of exponentially many samples have been taken. This could lead to earlier pruning decisions. To keep the ($\epsilon$,$\delta$)-PAC property, one would have to work with for example any-time valid confidence bounds. In a practical sense, for the algorithm to run, the pruning and confidence bound computation should be significantly sped up.

Fourth, to improve the run time of the algorithm, it is possible to compute the confidence bounds differently. It is relatively easy to compute the radius needed to attain a certain value for a node. This technique was briefly touched upon in Appendix B. Although this does not change sample complexity, improving runtime complexity could be an advantage for some practical applications.

Finally, it might be possible to construct an upper bound on the sample complexity of our algorithm using a similar approach to the approach taken by Teraoka et al. [15] We believe this

is feasible, at least in a crude way. Since a spherical confidence region of radius $\rho$ is contained in a box with sides $2\rho$, we think that their approach could be applied to the the box containing the confidence region. Because the containing box is much larger than the sphere it contains, the upper bound found using this method would likely be worse than the upper bound for FindTopWinner. Nevertheless, we believe that it is possible to improve this crude upper bound. One of the reasons the proof by Teraoka et al. works well, is because their confidence region projected to the values of internal nodes is again a box. In our case, the confidence region is a sphere $B(\hat{\boldsymbol{\mu}}, \rho) \subseteq \mathbb{R}^{\text{leaves}(\mathcal{T})}$, but when we map it to the values of internal nodes, this property is lost, as we have seen in Figure 4.10. However, it can be shown that this projected set is contained in a sphere of radius $\rho$ in a lower dimensional space. We speculate that given this, it is possible to develop a tighter upper bound on the sample complexity than the crude strategy described above.

# Appendix A

# Probability theory

## A.1 Probability bounds

**Proposition 1.** *Union bound For a countable set of events $\mathcal{A} = \{A_1, A_2, A_3, \ldots\}$ we have*

$$\mathbb{P}\left(\bigcup_{A\in\mathcal{A}} A\right) \leq \sum_{A\in\mathcal{A}} \mathbb{P}(A). \tag{A.1}$$

*Proof.* This directly follows from the $\sigma$-subadditivity of the probability measure. □

**Proposition 2** (Hoeffding's inequality)**.** *Let $X_1 \ldots, X_T$ be $[0,1]$-valued independent random variables such that $\mathbb{E}(X_t) = \mu$ for $t \in [T]$. Then for any $c > 0$*

$$\mathbb{P}\left(\frac{1}{T}\sum_{t=1}^{T} X_t > \mu + c\right) \leq \exp\left(-2c^2T\right) \tag{A.2}$$

$$\mathbb{P}\left(\frac{1}{T}\sum_{t=1}^{T} X_t < \mu - c\right) \leq \exp\left(-2c^2T\right). \tag{A.3}$$

**Corollary 1.** *Let $X_1 \ldots, X_T$ be $[0,1]$-valued independent random variables such that $\mathbb{E}(X_t) = \mu$ for $t \in [T]$. Then for any $c > 0$*

$$\mathbb{P}\left(\left|\frac{1}{T}\sum_{t=1}^{T} X_t - \mu\right| > c\right) \leq \exp\left(-2c^2T\right). \tag{A.4}$$

**Proposition 3** (Chernoff bound)**.** *Let $X$ be a random variable. The for any $\eta > 0$ we have*

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[\exp(\eta X)]}{\exp(\eta a)}. \tag{A.5}$$

*Proof.* By applying Markov's inequality to $\exp(\eta X)$ the result immediately follows. □

# Appendix B

# Plausible values of internal nodes

In Section 4.2.3 we showed the set plausible values of leaves and the set of plausible values of internal nodes of the tree. The first plot is relatively easy to obtain, indeed, this is precisely equal to the confidence region. The second plot however, is less straightforward. In this appendix chapter, we will derive the methods that were used to obtain this plot.

In Section 5.3 we introduced the concept of upper and lower confidence bounds on the value of any node, and we computed these bounds in Section 5.4. In the last section, we also saw a technique of 'distributing' the radius of the spherical confidence region over two nodes.

Assume that we have an oracle tree with leaf values $\hat{\boldsymbol{\mu}} \in \mathbb{R}^{\text{leaves}(\mathcal{T})}$. What is the smallest radius $\rho$ such that a node $n$ attains value at most $\gamma = v_n(\boldsymbol{\mu}')$ for some $\boldsymbol{\mu}' \in B(\hat{\boldsymbol{\mu}}, \rho)$? If $n$ is a leaf node, this is clear. Let $c_{\text{low}}(n, \gamma, \hat{\boldsymbol{\mu}}')$ be the squared radius $\rho^2$. Then if $n$ is a leaf we make a distinction between two cases. If the value is less than $\gamma$, then the leaf already attains a value at most $\gamma$. Otherwise, the squared radius has to be at least $(\gamma - \hat{\mu}_n)^2$. We have,

$$c_{\text{low}}(\ell) = \begin{cases} (\gamma - \hat{\mu}_\ell)^2 & \text{if } \gamma \geq \hat{\mu}_\ell \\ 0 & \text{otherwise.} \end{cases} \tag{B.1}$$

If $n$ is a min node, it takes the value of its child with the smallest value. Therefore, only one child has to attain a value of at most $\gamma$. Therefore, we have,

$$c_{\text{low}}(n, \gamma, \hat{\boldsymbol{\mu}}) = \min_{c \in \text{C}(n)} c_{\text{low}}(c, \gamma, \hat{\boldsymbol{\mu}}). \tag{B.2}$$

If $n$ is a max node, it takes the value of its child with the greatest value. Therefore, all children have to attain a value of at most $\gamma$. The set of leaves below any child node $c$ has no leaves in common with a set of leaves below any other child node $c'$. Consequently, we have the followingen,

$$\begin{aligned} c_{\text{low}}(n, \gamma, \hat{\boldsymbol{\mu}}) &= \sum_{\ell \in \text{leaves}(\mathcal{T})} (\mu'_\ell - \hat{\mu}_\ell)^2 && \text{(B.3)} \\ &= \sum_{c \in \text{C}(n)} \sum_{\ell \in \text{leaves}(\mathcal{T}_c)} (\mu'_\ell - \hat{\mu}_\ell)^2 && \text{(B.4)} \\ &= \sum_{c \in \text{C}(n)} c_{\text{low}}(n, \gamma, \hat{\boldsymbol{\mu}}) && \text{(B.5)} \end{aligned}$$

We can now compute $c_{\text{low}}(n, \gamma, \hat{\boldsymbol{\mu}})$ for any node in an oracle tree. Analogously we can derive $c_{\text{high}}(\gamma, n, \hat{\boldsymbol{\mu}})$, which returns the radius squared needed to increase the value of a node $n$ with estimate $\hat{\boldsymbol{\mu}}$ to $\gamma$. With $c_{\text{low}}$, $c_{\text{high}}$ and the confidence bounds UCB and LCB we can now compute

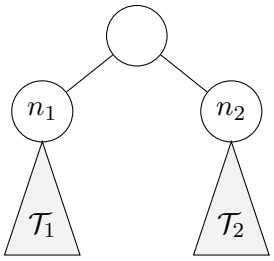



***Figure B.1:*** *The tree $\mathcal{T}$ with two children at the root, both having an unspecified subtree.*

the set of plausible values for internal node. Assume that we have a tree $\mathcal{T}$, with two children at the root $n_1$ and $n_2$. Both children root some subtree $\mathcal{T}_1$ and $\mathcal{T}_2$. This is illustrated in the figure below.

Next, assume that the confidence region $\mathcal{C}$ is a sphere centered at $\hat{\boldsymbol{\mu}} \in \mathbb{R}^{\text{leaves}(\mathcal{T})}$ and has radius $\rho$. Let us assume that $n_1$ has value $\gamma \in [\text{LCB}_{n_1}(\rho^2), \text{UCB}_{n_1}(\rho^2)]$. Then the values that $n_2$ can attain depends on $\gamma$. Indeed, for any $\boldsymbol{\mu}' \in \mathcal{C}$ we have $||\boldsymbol{\mu}' - \hat{\boldsymbol{\mu}}||^2 = ||\boldsymbol{\mu}'_{n_1} - \hat{\boldsymbol{\mu}}_{n_1}||^2 + ||\boldsymbol{\mu}'_{n_2} - \hat{\boldsymbol{\mu}}_{n_2}||^2 \leq \rho^2$, where $\boldsymbol{\mu}'_{n_1}$ and $\boldsymbol{\mu}'_{n_2}$ are the lower dimensional vectors of leaves that exist below $n_1$ and $n_2$ respectively. Since $||\boldsymbol{\mu}'_{n_1} - \boldsymbol{\mu}_{n_1}||^2$ is at least $\max\{c_{\text{low}}(n, \gamma, \hat{\boldsymbol{\mu}}), c_{\text{high}}(n, \gamma, \hat{\boldsymbol{\mu}})\} = \rho_1$, we find that the vector $\boldsymbol{\mu}'_{n_2}$ is in the sphere with center $\hat{\boldsymbol{\mu}}'_{n_2}$ and squared radius $\rho_2^2 = \rho^2 - \rho_1^2$. We then conclude that the plausible range of values of node $n_2$ is $\text{LCB}_{n_2}(\rho_2^2), \text{UCB}_{n_2}(\rho_2^2)]$.

Now that for any value $n_1$ we can compute the range of values of $n_2$, we can construct the set of all plausible pairs of values $(v_{n_1}, v_{n_2})$ by using a scanning approach, scanning over all plausible values of $n_1$.

# Bibliography


[1] P. Auer, Paul Fischer, and N. Cesa-Bianchi. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(3):235–256, 2002.

[2] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57 – 83, 2002.

[3] Guillaume Chaslot, Mark Winands, H. Herik, Jos Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 04:343–357, 11 2008.

[4] Rémy Degenne and Wouter M Koolen. Pure exploration with multiple correct answers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 14591–14600. Curran Associates, Inc., 2019.

[5] Aurélien Garivier and Emilie Kaufmann. Optimal best arm identification with fixed confidence. In *COLT*, 2016.

[6] Aurélien Garivier, Emilie Kaufmann, and Wouter M. Koolen. Maximin action identification: A new bandit framework for games. volume 49 of *Proceedings of Machine Learning Research*, pages 1028–1050, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

[7] C. Giraud. *Introduction to High-Dimensional Statistics*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 2014.

[8] Shivaram Kalyanakrishnan, Ambuj Tewari, Peter Auer, and Peter Stone. Pac subset selection in stochastic multi-armed bandits. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 227–234, Madison, WI, USA, 2012. Omnipress.

[9] Emilie Kaufmann and Wouter M Koolen. Monte-carlo tree search by best arm identification. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4897–4906. Curran Associates, Inc., 2017.

[10] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[11] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[12] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.

[13] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[14] Liang Tang, Romer Rosales, Ajit Singh, and Deepak Agarwal. Automatic ad format selection via contextual bandits. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, page 1587–1594, New York, NY, USA, 2013. Association for Computing Machinery.

[15] 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.

[16] 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.

[17] J.M. White. *Bandit Algorithms for Website Optimization: Developing, Deploying, and Debugging*. O'Reilly Media, 2012.