國立臺灣大學電機資訊學院電機工程學系
碩士論文
Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

非穩態下之極值多臂吃角子老虎機於數值優化領域之探討
Non-stationary Extreme Bandits: An Optimization Case Study

吳柏儒
Po-Ju Wu

指導教授：于天立 博士
Advisor: Tian-Li Yu, Ph.D.

中華民國 112 年 7 月
July, 2023

doi:10.6342/NTU202303980

致謝

在碩士班的日子邁向尾聲，回顧過去在實驗室的生活，我想我最想念的是meeting的時光。每次跟于天立老師開會時，總能激發我的想法，並在我不清楚的地方爲我解惑。但更重要的，是老師給我們做研究的態度，學習用自己的話解釋理論，從數據看出端倪，並勇於對他人的研究提出問題並反思。這樣的研究態度，也讓我能夠應用在生活上其他的事情，到了碩士班，我似乎才眞的了解如何運用手中的知識去創造價值。此外，在人生閱歷上，老師也跟我們分享了許多，讓我知道自己是可以創造許多可能的。

這三年來，我在人工智慧的領域中探索，感謝老師總是給予我很大的自由，讓我能往我有興趣的方向前進，並適時的給予指導。過程中也遇到許多卡關的地方，感謝實驗室的夥伴們，尤其是張麒仙學姐，不但總是關心、了解我的研究外，在實驗室日常生活中也時時協助我，在最後撰寫論文的過程也提供我許多意見，非常感謝。還有其他同學們在平常討論時總能激發我的研究想法。

另外，我想要感謝求學過程中對我幫助最大的人：我的父母，照顧我的日常生活，並鼓勵支持我。感謝求學路上曾相遇的人們，還有我的女朋友，在我爲研究煩惱時，總能給我繼續向前的動力，也是我的情感依靠。

在AI應用越來越廣泛的今日，身爲人類，我會以No Free Lunch Theorem做爲比喻，總會有某些任務，是人類有辦法做得比機器好的。在我看來，喜怒哀樂與愛，就是身而爲人最重要的資產。

最後，謝謝自己願意持續探索新事物的心，從物理、經濟到人工智慧，我發現不同領域的知識多少會互通，偶然相會。期許自己，有一天能把所學爲社會作出貢獻，把理論變爲眞實的應用。離開台大後，我的旅程才正要開始。

doi:10.6342/NTU202303980

中文摘要

在工程、科學和財務領域，我們經常遇到在有限的資源下做出選擇的問題，這種問題可歸結爲多臂吃角子老虎機問題，也是強化學習的基礎。現有的研究主要集中在穩態場景的期望報酬優化，但實際上許多問題並非穩態，或是追求極值報酬而非期望報酬。我們開發了一種以順序統計量爲基礎的演算法，並配合自適應分佈模型，以優化在非穩態場景下追求極值之資源分配。我們將此演算法應用於實數數值優化、蒙地卡羅樹搜索以及深度學習模型超參數優化等三種問題，並與目前經典的多臂吃角子老虎機演算法做比較。在實數數值優化中，我們使用自適應共變異數矩陣演化策略作爲優化器，在CEC2005的多模態基準函數上做測試，在有限的抽樣預算下我們的演算法具有優勢。在蒙地卡羅數搜尋中，我們設計了獎勵函數，透過實驗測試演算法在不同場景下之穩態性，在獎勵函數值域不受限制的場景下我們的演算法具有統計上的優勢。而在超參數優化問題上，我們設計了一個架構結合目前之最新架構，並且在我們測試的電腦視覺訓練問題上能取得較原版較高之測試集準確度。

關鍵詞

多臂吃角子老虎機、順序統計量、非穩態、極值、強化學習、機器學習、實數優化、蒙地卡羅樹、超參數優化、自適應共變異數矩陣演化策略

doi:10.6342/NTU202303980

Abstract

In engineering, scientific, and financial domains, we frequently encounter the challenge of making decisions when faced with limited resources. This issue can be framed as the multi-armed bandit problem, which serves as a fundamental concept in reinforcement learning. Existing research primarily focuses on optimizing the expected reward in stationary scenarios. However, many real-world problems exhibit non-stationarity or prioritize attaining the highest possible reward rather than the expected reward. To address this, we have developed an algorithm that leverages order statistics and adaptive distribution models to optimize resource allocation in pursuit of the highest possible reward in non-stationary environments. We applied our algorithm to three different problems: real-valued optimization, Monte-Carlo tree search, and hyperparameter optimization for deep learning models. Also, we compared it with the classical multi-armed bandit algorithm.

In real-valued optimization, we utilized the self-adaptive covariance matrix evolution strategy as the optimizer. We tested our algorithm on the multimodal benchmark functions from CEC2005. Our algorithm exhibited advantages within limited sampling budgets. For Monte-Carlo tree search problems, we designed a reward function and conducted experiments to assess the robustness of our algorithm in various scenarios. Our algorithm demonstrated statistical advantages in scenarios where the reward function's value range was unconstrained. Regarding hyperparameter optimization, we devised a framework that incorporates state-of-the-art architectures. When evaluated on computer vision training problems, our algorithm achieved higher testing set accuracy compared to the original version.

Keywords

multi-armed bandits, order statistics, non-stationary, extreme value, reinforcement learning, machine learning, real-valued optimization, Monte-Carlo tree search, hyperparameter optimization, Covariance matrix adaptation evolution strategy

doi:10.6342/NTU202303980

Contents

doi:10.6342/NTU202303980

doi:10.6342/NTU202303980

List of Figures

doi:10.6342/NTU202303980

List of Tables

doi:10.6342/NTU202303980

Chapter 1

Introduction

Multi-armed bandit (MAB) problems are sequential decision-making problems that arise in various fields, including statistics [1], economics [2], and machine learning [3]. The term 'bandit' refers to a slot machine commonly used as a metaphor for these problems. In a MAB problem, an agent faces a set of options, often called arms or actions, each associated with an unknown reward distribution. The agent aims to maximize its cumulative reward over a sequence of steps by strategically selecting which arms to pull.

The fundamental challenge in MAB problems is the trade-off between exploration and exploitation. On the one hand, the agent needs to explore different arms to learn about their reward distributions and identify the most rewarding arm(s). On the other hand, the agent also wants to exploit the arms that have shown promising rewards in the past to maximize their immediate compensation. Striking the right balance between exploration and exploitation is crucial to achieving optimal performance in MAB problems.

These problems find applications in a wide range of real-world scenarios. For example, in online advertising [4], a platform may need to select the most effective advertisement to display to users to maximize click-through rates or revenue. In clinical trials [5], researchers may want to identify the most effective treatment strategy from a set of options. In recommendation systems [6], algorithms must choose which items to recommend to users based on their preferences.

Various algorithms and strategies have been developed to tackle MAB problems. These algorithms differ in their exploration and exploitation policies, and their performance is evaluated based on metrics such as cumulative regret [7, 8] or average reward. Classic approaches include epsilon-greedy [9], upper confidence bound (UCB) [9, 10], and Thompson sampling (TS) [11]. These algorithms leverage statistical inference, probability theory, and adaptive learning techniques to balance

doi:10.6342/NTU202303980

exploration and exploitation efficiently.

In general, MAB problems present a framework for sequential decision-making under uncertainty. These problems have wide-ranging applications and have spurred the development of various algorithms and strategies that continue to advance our understanding of decision-making in uncertain environments.

Recent research on variations of MAB problems has been prolific, focusing on addressing new challenges and expanding the applicability of MAB frameworks. Here are some notable areas of research and advancements:

1. Contextual bandits [12, 13]: One significant extension of MAB is contextual bandits, where additional contextual information is provided alongside the arms. Recent research has explored the integration of contextual information into MAB algorithms to improve decision-making. Contextual bandits enable personalization and adaptive decision-making by leveraging the available context to select arms more effectively.

2. Non-stationary environments [14, 15]: Traditional MAB assumes stationary reward distributions, but real-world scenarios often involve dynamic environments where reward distributions change over time. Recent research has focused on developing algorithms that can adapt to non-stationary environments by dynamically updating beliefs or exploring and exploiting arms in a time-varying manner.

3. Bandit algorithms with side information [16, 17]: In some applications, additional side information about the arms or the rewards is available. Researchers have explored incorporating such side information into MAB algorithms to improve decision-making accuracy. This includes leveraging reinforcement learning and deep learning techniques to handle complex side information efficiently.

4. Combinatorial bandits [18]: Combinatorial bandit problem involves selecting a subset of arms from a giant arm set. Recent research has focused on developing algorithms that efficiently explore and exploit combinations of arms to optimize the cumulative reward. These advancements open doors to applications where decisions involve selecting multiple options simultaneously.

5. Online learning [19] and regret minimization [20]: The notion of regret, which quantifies the cumulative opportunity loss incurred by a learning algorithm, remains a fundamental measure in MAB research. Recent studies have proposed new algorithms and techniques to minimize regret in online learning

doi:10.6342/NTU202303980

settings, aiming to achieve near-optimal performance in terms of cumulative reward.

6. Variation of objectives: Traditional MAB problems have focused on maximizing cumulative rewards. In contrast, some other MAB frameworks may concentrate on searching for the arm to generate the best possible reward.

7. Practical applications and real-world deployments: Researchers have been actively exploring the practical deployment of MAB algorithms in various domains, such as healthcare, online advertising, recommendation systems, and autonomous systems. Recent studies have addressed the challenges of deploying MAB algorithms in real-world settings, including scalability, robustness to uncertainties, and interpretability.

Recent research on variations of MAB problems has demonstrated the versatility and effectiveness of MAB frameworks in tackling complex decision-making problems. These advancements continue to enable better decision-making in dynamic environments and extend the applicability of MAB algorithms to more practical scenarios.

1.1 Motivation

We address the challenge of MAB on real-world applications in three parts: non-stationary reward distributions, optimization objectives, and unknown reward range.

1.1.1 Non-stationary Reward Distributions

While traditional MAB algorithms assume stationary reward distributions, real-world environments often exhibit dynamics where reward distributions change over time. The optimization of extreme rewards in non-stationary MAB scenarios finds applications in various domains, including financial investments, portfolio management [21], and resource allocation [22]. In financial markets, the identification of arms associated with extreme rewards could lead to significant gains, while in resource allocation, it could result in more efficient allocation decisions with far-reaching impacts.

1.1.2 Optimization Objectives

In terms of optimization objectives, most MAB works focus on maximizing the expected rewards. In other words, maximizing the average performance of the trained

doi:10.6342/NTU202303980

agent. In many cases, it seems to be a pretty reasonable objective as we may need an advertiser that makes the most average income or a trading strategy with the largest expected growth. While for other scenarios, this may not be the case: Imagine being in a winner-takes-all game competition. The winner should be the player who gets the highest score among limited game trials. The best strategy is to chase for the highest possible best score rather than the highest average score. Also, when it comes to anomaly detection problems where the anomaly is defined as observing a value larger than a certain threshold, we might wish to find the distribution that may most likely create extreme values, *e.g.* heavy-tailed distributions. Furthermore, in real-valued optimization problems, we care about the best-optimized value seen so far rather than the average optimized value.

1.1.3 Unknown Reward Range

In many real-world applications, the reward range associated with each arm may not be limited or known in advance. For instance, the reward could represent user satisfaction in recommendation systems, which may not have a predefined upper bound. Similarly, in portfolio management, the potential returns on investments may not be restricted. In such situations, traditional MAB algorithms that assume a known reward range may fail to provide optimal solutions, as they cannot effectively handle the inherent uncertainty and variability associated with a known reward range.

Most MAB algorithms have assumptions that reward distribution is bounded within [0, 1]. Though sometimes it may be general and scalable, there are cases where it fails to fit. For example, a TS with a beta prior may not scale well with unknown reward distributions.

1.2 Thesis Objective

This work aims to address the challenges posed by non-stationary scenarios in MAB problems, with a specific focus on optimizing extreme rewards; such framework is also called the extreme multi-armed bandit (Extreme-MAB) problem [23], or K-armed bandit problem [24]. Recently, research has been done on this topic, including Maxmedian [25] and Quantile-of-Maxima (QoMax) [26]. This research proposes StatMax, a novel MAB strategy that adapts to non-stationary environments with adaptive surrogate distribution and prioritizes the arms associated with extreme rewards from the order statistics model.

doi:10.6342/NTU202303980

1.3 Roadmap

The organization of the following chapters goes as follows:

Chapter 2 gives an overview of the related works, including the traditional MAB and Extreme-MAB algorithms, to see how these algorithms solve the exploration-exploitation dilemma, also with their restrictions.

Chapter 3 presents the details of the StatMax algorithm, and each core of the algorithm is described in this chapter.

Chapter 4 describes the case studies involved in experiments. There are three domains in our experiments:

1. Real-valued optimization: We introduce IEEE CEC2005 benchmark functions [27] and covariance matrix adaptation evolution strategy (CMA-ES) [28].

2. Monte-Carlo tree search (MCTS) [29] games: We introduce the framework of MCTS and the designed k-ary tree benchmark problems.

3. Hyperparameter optimization of neural networks: We introduce the two novel state-of-the-art hyperparameter optimization algorithms: HyperBand [30] and Bayesian optimization of HyperBand (BOHB) [31] and the StatMax-integrated version of them.

Chapter 5 summarizes the thesis, and our contributions and conclusion are discussed. Also, we mention some possible further improvements and works in it.

doi:10.6342/NTU202303980

Chapter 2

Overview

In this chapter, we provide an overview of the works related to this thesis, including traditional MAB algorithms, non-stationary MAB algorithms, and Extreme-MAB algorithms.

2.1 MAB Algorithms

2.1.1 ϵ-greedy Algorithm

The ϵ-greedy [9] algorithm is a simple and intuitive strategy that enables decision-makers to make informed choices while occasionally exploring new alternatives. The algorithm assigns a fixed exploration rate, ϵ, which determines the probability of selecting a random or exploratory action instead of the current best-known action.

The algorithm operates in a cyclic manner, with each iteration representing a decision point. At each decision point, a random number is generated. If this number is less than ϵ, an exploratory action is chosen uniformly at random from the available options. On the other hand, if the random number is greater than or equal to epsilon, the algorithm exploits the current best-known action based on the available information. By adjusting the value of ϵ, decision-makers can control the balance between exploration and exploitation.

doi:10.6342/NTU202303980

The pseudo-code for ϵ-greedy algorithm can be presented by Algorithm 1:

Algorithm 1: ϵ-greedy

Input: probability threshold ϵ, set of arms A, number of rounds T, average reward for arm $\bar{A}_i$

```
t ← 0;
A_k ← random arm ∈ A;
while t ≤ T do
    r ← random(0, 1);
    if r < ε then
        sample random arm A_i ∈ A;
        update Ā_i;
    end
    else
        sample arm A_k;
        update Ā_k;
    end
    A_k ← arg max_i Ā_i;
    t ← t + 1
end
```


doi:10.6342/NTU202303980

2.1.2 Upper Confidence Bound (UCB)

UCB incorporates an upper confidence bound term that quantifies the uncertainty associated with the estimated parameter to balance exploration and exploitation. By iteratively updating the estimate and the upper confidence bound, UCB-based methods provide estimates that converge to the true parameter values over time.

The pseudo-code of UCB is described in Algorithm 2:

Algorithm 2: UCB algorithm

Input: scaling constant c, set of arms A, number of rounds T, number of arms l

Initialize $N(A_i)$ for all $A_i \in A$ as 0;
Initialize $\bar{A}_i$ for all $A_i \in A$ as 0;
$t \leftarrow 1$;
for $i \leftarrow 1$ **to** l **do**
    sample A_i;
    $\bar{A}_i \leftarrow$ received reward;
    $N(A_i) \leftarrow N(A_i) + 1$;
    $t \leftarrow t + 1$;
end
while $t \leq T$ **do**
    $k \leftarrow \arg\max_i(\bar{A}_i + c\sqrt{\frac{2\log(t)}{N(A_i)}})$; //UCB Calculation
    sample arm k and receive reward r_k;
    $N(k) \leftarrow N(k) + 1$;
    $\bar{A}_k \leftarrow \bar{A}_k + \frac{r_k - \bar{A}_k}{N(k)}$; //Running average
    $t \leftarrow t + 1$;
end

2.1.3 UCB-Normal

UCB-Normal [32] builds upon the UCB principle and tailors it specifically for normal distributions. UCB-Normal leverages the inherent properties of the normal distribution to adapt the UCB approach for this specific estimation task, thus ensuring accurate parameter estimation in scenarios with limited data. Also, using normal distributions, the algorithm is suitable for unknown reward range.

doi:10.6342/NTU202303980

The pseudo-code of UCB-Normal is described in Algorithm 3:

Algorithm 3: UCB-Normal algorithm

Input: scaling constant c, set of arms A, number of rounds T, number of arms l

Initialize $N(A_i)$ for all $A_i \in A$ as 0;
Initialize $\bar{A}_i$ for all $A_i \in A$ as 0;
$t \leftarrow 1$;
for $i \leftarrow 1$ **to** l **do**
    sample A_i; $\bar{A}_i \leftarrow$ received reward; $N(A_i) \leftarrow N(A_i) + 1$; $t \leftarrow t + 1$;
end
while $t \leq T$ **do**
    $k \leftarrow \arg\max_i(\bar{A}_i + c\sqrt{\frac{2\log(t)}{N(A_i)}})$; //UCB-Normal Calculation
    sample arm k and receive reward r_k;
    $N(k) \leftarrow N(k) + 1$;
    $\bar{A}_k \leftarrow \bar{A}_k + \frac{r_k - \bar{A}_k}{N(k)}$; //Running average
    $t \leftarrow t + 1$;
end

Despite UCB methods, Thompson sampling is another famous MAB algorithm that utilizes Bayesian method for arm selection.

2.1.4 Thompson Sampling (TS)

TS works by first drawing a sample from the probability distribution for each arm. The sample is the algorithm's belief about the mean reward for the arm. The algorithm then selects the arm with the highest sampled mean reward.

After the arm is selected, the algorithm observes the reward. The reward is used to update the probability distribution for the arm. TS has been shown to be effective in a variety of MAB problems. Here is a more detailed explanation of how TS works:

1. The algorithm initializes a prior distribution over the possible values of the mean reward for each arm.
2. For each arm, the algorithm draws a sample from the prior distribution. This sample is the algorithm's belief about the mean reward for the arm.
3. The algorithm selects the arm with the highest sampled mean reward.
4. The algorithm plays the selected arm and observes the reward. The reward is used to update the prior distribution for the arm.

doi:10.6342/NTU202303980

5. The algorithm repeats steps 2-5 until the end of the horizon.

The prior distribution can be any distribution that the algorithm chooses. A common choice is a beta distribution, as the beta distribution is also the conjugate prior to the likelihood distribution, which means that it can be easily updated using the observed rewards.

The pseudo-code of TS is described in Algorithm 4:

Algorithm 4: Thompson sampling algorithm

Input: set of arms A, number of rounds T, number of arms l

Initialize α_k and β_k for all k in A;
for t *in* 1 *to* T **do**
  for k *in* 1 *to* l **do**
    sample $\hat{\theta_k} \sim \text{beta}(\alpha_k, \beta_k)$;
  end
  play arm $x_t \leftarrow \arg\max_k \hat{\theta_k}$;
  Apply x_t and observe r_t;
  Update: $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t} + r_t, \beta_{x_t} + 1 - r_t)$;
end

These methods are proved to have a theoretical bound on regrets, assuming that rewards are stationary and bounded. Some variants are proposed for non-stationary scenarios. Take Sliding-window UCB, for example:

2.1.5 Sliding-window UCB (SW-UCB)

SW-UCB [33] is an algorithm for MAB problems. It is a variant of the UCB algorithm that is designed to handle non-stationary environments. In a non-stationary environment, the distribution of rewards for each arm can change over time. SW-UCB addresses this by maintaining a window of the most recent rewards for each arm. The algorithm then selects the arm with the highest upper confidence bound, which is calculated using the rewards in the window.

The window size can be adjusted dynamically, depending on the volatility of the environment. If the environment is very volatile, the window size can be small to ensure that the algorithm is not exploiting outdated information. If the environment is less volatile, the window size can be larger to allow the algorithm to learn more about the current distribution of rewards.

SW-UCB is effective in various non-stationary environments, for example, online advertising and financial trading. Here is a more detailed explanation of how SW-UCB works:

doi:10.6342/NTU202303980

1. The algorithm initializes a window of size W.
2. For each arm, the algorithm calculates the UCB for the arm. The UCB is computed using the rewards in the window for the arm.
3. The algorithm selects the arm with the highest UCB.
4. The algorithm plays the selected arm and observes the reward.
5. The reward is added to the window for the selected arm. If the window size gets over W, then the oldest reward in the window will be dropped.
6. The algorithm repeats steps 2-5 until the horizon's end.

The pseudo-code of SW-UCB is described in Algorithm 5:

Algorithm 5: Sliding window UCB algorithm

Input: scaling constant c, set of arms A, reward record for the arm i R_i, number of rounds T, number of arms l, window size W

Initialize $R(i)$ for all $i \in A$ as empty list;
$t \leftarrow 1$;
for $i \leftarrow 1$ **to** l **do**
    sample A_i and receive reward r_i;
    append r_i to R_i;
    $t \leftarrow t + 1$;
end
while $t \leq T$ **do**
    for $i \leftarrow 1$ **to** l **do**
        $X_t(i) \leftarrow$ average of the last W elements in R_i;
        $c_t(i) \leftarrow c\sqrt{\frac{2\log(t)}{W}}$; //UCB Calculation
    end
    $A_t \leftarrow \arg\max_i(X_t(i) + c_t(i))$;
    sample arm A_t and receive reward r_{A_t};
    append r_{A_t} to R_{A_t};
    $t \leftarrow t + 1$;
end

2.2 Extreme Bandit Algorithms

In the below subsections, we introduce several Extreme-MAB algorithms that focus on the maximum possible reward.

doi:10.6342/NTU202303980

2.2.1 Quantile of Maxima (QoMax)

QoMax solves Extreme-MAB problems by using a quantile estimator to estimate the expected maximum reward of each arm.

The quantile estimator works by first collecting a batch of rewards from each arm. The rewards are then sorted, and the median reward is used as the indicator for choice. There are two versions of QoMax: The explore-then-commit (ETC) version and a subsampling dueling algorithm (SDA) version.

The QoMax-ETC follows a two-phase approach. In the first phase, the learner selects a quantile value q and determines the batch size b_T and sample size n_T based on the given time horizon T. During the exploration phase, each arm is pulled n_T times and distributed into b_T batches of size n_T. After completing this step, the learner computes an estimator for each arm using the collected data from the different batches.

In the second phase, known as the exploitation phase, the algorithm selects and pulls the arm I_T with the highest QoMax value repeatedly until the time horizon T is reached. This phase focuses on exploiting the arm that shows the most promising estimated performance based on the QoMax estimators calculated during the exploration phase.

The QoMax-SDA operates in successive rounds, each consisting of three main steps: leader selection, dueling between the leader and challengers, and data collection. At the beginning of each round, the learner has access to the history of different arms denoted as X_{r_k}. The rewards for each arm are collected and organized into batches of equal size, denoted as $b_k(r)$ and $n_k(r)$, respectively. The leader for the round is selected based on the arm that has been queried the most up to that point.

Once the leader is determined, duels are performed between the leader and the remaining challengers. The set of arms to be pulled at the end of the round, denoted as A_{r+1}, is determined based on the outcomes of the duels. An arm is added to A_{r+1} if it wins the duel or if its number of queries is below a certain threshold determined by the sampling obligation function $f(r)$. If no challenger is added to A_{r+1}, the leader is pulled.

The duel procedure compares the QoMax of the challenger using its entire history with the QoMax of the leader on a subsample of its history. The subsampling mechanism used in QoMax-SDA considers the rewards collected from the last $n_k(r)$ queries of the leader and keeps the first batches for the leader, ensuring that both the leader and the challenger use the same amount of data for QoMax computation. This subsampling approach introduces diversity in the encountered subsamples when

doi:10.6342/NTU202303980

the leader is often pulled while reducing the storage requirements.

The data collection procedure in QoMax-SDA updates existing batches by collecting additional queries and creating new batches if necessary. Existing batches are updated by collecting the $n_k + 1$-st query for all batches, while new batches are created until the number of batches b_k reaches the limit determined by the batch function $B(n_k + 1)$.

The pseudo-code of QoMax-ETC is described in Algorithm 6:

Algorithm 6: Quantile of Maxima (QoMax)

Input: Quantile q, number of batches b, batch size n, data $X = (X_{m,i})_{m \le b, i \le n}$

Output: Quantile of Maxima

```
Divide X into b batches of size n;
for j = 1 to b do
    Compute M_j = max_{i=1}^{n} X_{j,i};
end
Sort M = (M_1, ..., M_b) in increasing order;
Compute k = ⌊qb⌋;
return M_k;
```

2.2.2 MaxMedian

In the MaxMedian algorithm, the learner selects an arm I_t at each time step t based on a time-dependent index. The algorithm requires parameters such as the decreasing step size ϵ_t, the number of arms K, the play horizon T, and the number of pulls $N_k(t)$ for each arm up to time t.

The algorithm begins by initializing each arm, pulling them once. Then, for each time step from $K + 1$ to T, the algorithm computes the index values $w_k(t)$ for each arm based on the previous pulls. The arm I_t is chosen as the one with the highest index value with a probability of $1 - \epsilon_t$, while with probability ϵ_t, a random arm is explored.

The algorithm ensures that the minimum number of pulls for any arm $m(t)$ is lower bounded, given the appropriate choice of the decreasing step size ϵ_t.

doi:10.6342/NTU202303980

The pseudo-code of MaxMedian is described in Algorithm 7:

Algorithm 7: MaxMedian

Input : Number of arms K, time horizon T

Output: None

Initialization: Pull each arm once;
for $t = K+1, \ldots, T$ **do**
  Let $m(t) = \min_{k \in K} N_k(t)$ be the smallest number of pulls among all arms up to time t;
  for $k \in K$ **do**
    Let $O_{k,t}(\zeta)$ be the ζ-th order statistic of the rewards of arm k up to time t;
    Let $w_{k,t} = O_{k,t-1}(\lceil N_k(t-1)/m(t-1) \rceil)$ be the median of the top $\lceil N_k(t-1)/m(t-1) \rceil$ rewards of arm k up to time $t-1$;
  end
  Let $I_t = \arg\max_{k \in K} w_{k,t}$ be the set of arms with the largest median reward up to time t;
  Let J_t be a random variable with $\Pr(J_t = k) = \frac{1-\epsilon_t}{|I_t|}$ for $k \in I_t$ and $\Pr(J_t = k) = \frac{\epsilon_t}{K-|I_t|}$ for $k \in K \setminus I_t$;
  Pull arm J_t and observe reward $r_{J_t,t}$;
  Update the number of pulls of arm J_t: $N_{J_t}(t) = N_{J_t}(t-1) + 1$;
end

doi:10.6342/NTU202303980

Chapter 3

StatMax

In this chapter, we introduce the framework of the StatMax algorithm. The first section introduces a brief overview of the StatMax algorithm, and the remaining sections describe the core of the algorithms.

3.1 Overview

Following our hypothesis, our system should tackle Extreme-MAB problems with the properties below:

- Non-stationarity: Reward distribution changes through time.
- Extreme reward: The algorithm should aim to maximize extreme reward.
- Unknown reward range: Range of reward is unknown.

Intuitively, one would suggest a simple greedy algorithm to pull the arm with the highest observed value seen so far, but for the non-stationary scenario, this would cause the algorithm to converge to a local maximum at an early stage, which is not the desired case, also for stationary case, the convergence rate is not optimal. These properties also eliminate the usage of some explore-then-exploit algorithms since they are also prone to early convergence. We aim for an algorithm that can perform well enough both on stationary and non-stationary scenarios. The proposed StatMax algorithm contains three parts:

- Adaptive surrogate distribution: The algorithm sets up a surrogate distribution for each arm, which represents the fitted distribution based on each arm's history, propagated through time, with this setting, StatMax has the ability of handling non-stationary problems.

doi:10.6342/NTU202303980

- Order statistics: Based on the surrogate models, the algorithm builds up an indicator for each arm that indicates the expected maximum order statistics to indicate the arm that has the highest potential to generate maximum reward.
- Budget aware: The indicator is aware of remaining budgets, this property tunes the exploration-exploitation ratio in an adaptive way.

The data pipeline of StatMax can be shown in Figure 3.1:

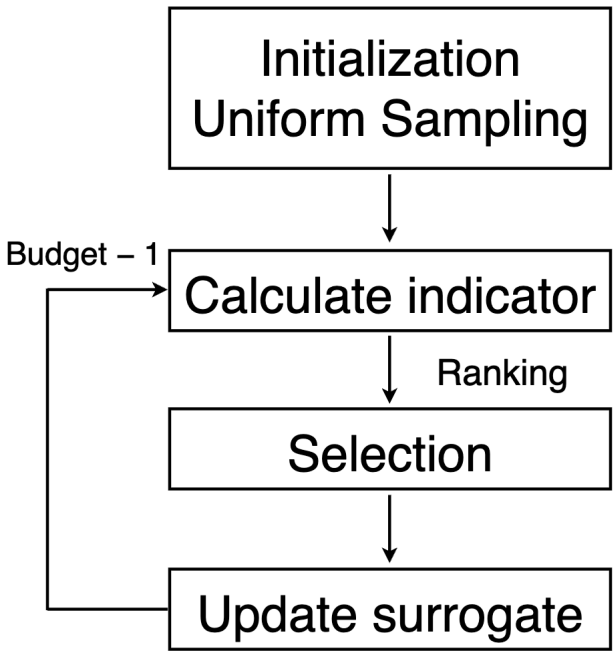


Figure 3.1: Data Pipeline of StatMax

Similar to UCB-type algorithms, StatMax uses indicators at each round to determine which arm to pull and update the parameters related to the chosen arm afterward. Following the concept of optimism in the face of uncertainty, at each round, the agent pulls the arm that has the largest indicator value.

For a better understanding of the methodology, assume there are n arms with b remaining budgets. At each round, the agent calculates the expected maximum value from b i.i.d. samples from each surrogate distribution of every arm and chooses the arm with the highest expected value.

With this methodology, the algorithm picks the arm with the most potential to create extreme value every time. As the remaining budget b decreases, the algorithm adaptively increases the ratio of exploitation to exploration since the expected maximum grows linearly with i.i.d. sample size. The introduction of the three core parts is introduced in the below subsections.

doi:10.6342/NTU202303980

3.2 Adaptive Surrogate Distribution

To tackle the problem of non-stationarity and unknown bound, we designed a discounted method to update the surrogate distribution of each arm when being pulled. Take normal distribution, for example, the distribution is defined by its mean u and standard deviation ρ. Initially, every arm is given a predefined u and ρ. When the arm is pulled, we update the two parameters in a discounted way such that recent observation is given more weight. The update method is stated as Algorithm 8:

Algorithm 8: StatMax: Update

Input : M (model), r (reward)

Output: Updated M

$M_u^* \leftarrow 0.95 \times M_u + 0.05 \times r$

$M_r^* \leftarrow M_r + r$

$M_v^* \leftarrow 0.95 \times M_v + 0.05 \times std(M_r)$

return M^*

3.3 Order Statistics

Different from MAB algorithms focusing on maximizing the expected mean, we adopt order statistics here for maximizing observation value. The maximum order statistics is defined as the distribution of the maximum of i.i.d. samples from a certain distribution.

Given a surrogate distribution, we can calculate the maximum order statistics given n remaining targets. As n increase, the calculation becomes infeasible and time-consuming; we can also take n random sample from i.i.d. surrogate distributions, which would cause the computation complexity grows linearly with the size of n.

Here we adopt the approximation of maximum order statistics[34] to prevent computation issues. Given n i.i.d. normal distributions $N(\mu, \sigma)$, the expect maximum order statistics of $Z = \max(N, N, ...N)$ is upper bounded:

$$\mathbb{E}[Z] \leq \mu + \sigma \log \sqrt{2n}$$

Making use of this bound, we defined our indicator as:

$$ESTIMATE(D) = D_\mu + D_\sigma \log \sqrt{2b}$$

Where D is the surrogate distribution, D_μ is the mean of the surrogate distribution, D_σ is the standard deviation of the surrogate distribution, and b is the remaining targets. With this setting, StatMax would always look for the arm that has potential to generate extreme reward.

doi:10.6342/NTU202303980

3.4 Budget Aware

In real applications, often, there are budget constraints (i.e., the limits of function calls/time). We consider the total budget for our indicator design, given B unlimited budgets, at step t, the remaining target b is given by $B - t$.

The formula shows that as the step increases, the number of i.i.d. sample sizes decreases, which converges a greedy strategy that exploits the arm with the highest mean at the end. This mechanism also balances MAB and Extreme-MAB, making it a general algorithm for MAB problems. Combining the concepts above, the StatMax algorithm is introduced as Algorithm 9:

Algorithm 9: StatMax algorithm

Data: scaling constant c, set of arms A, reward record for the arm i R_i, the surrogate distribution for arm i D_i, number of rounds T, number of arms l, total budget B

```
b ← B;
t ← 1;
k ← random arm ∈ A;
for i ← 1 to l do
    sample A_i;
    append R_i with sampled reward;
end
while b ≥ 0 do
    k ← arg max_i ESTIMATE(i);
    sample arm k;
    append R_k with sampled reward;
    update D_k;
    b ← b − 1;
end
```

We designed the StatMax algorithm as a simple plug-in for decision-making problems; it can be easily implemented for many scenarios. Also, we make the algorithm nearly parameter-less for robustness.

doi:10.6342/NTU202303980

Chapter 4

Case Studies

In this chapter, we introduce the three case studies (real-valued optimization, MCTS, Hyperparameter optimization). We described how StatMax can be integrated into these problems and the experiment results are presented.

4.1 Real-valued Optimization

Real-valued optimization, also known as continuous optimization, is a fundamental and versatile field of mathematical optimization that plays a crucial role in addressing complex problems in various domains. Unlike discrete optimization, which deals with finite sets of variables and feasible solutions, real-valued optimization involves continuous variables and considers an infinite number of potential solutions within a specified domain.

In real-valued optimization, the objective is to find the optimal values of one or more real-valued variables that maximize or minimize a given objective function, subject to a set of constraints. The objective function represents the quantity to be optimized, while the constraints define the allowable range of values for the decision variables, adhering to specific problem requirements.

The applications of real-valued optimization are widespread and have found relevance in diverse fields, including engineering, finance, operations research, machine learning, and many others. It provides an essential framework for tackling problems with continuous variables, such as designing optimal structures, scheduling processes, financial portfolio management, and fitting models to data, among numerous other scenarios.

Solving real-valued optimization problems often requires a combination of mathematical techniques, algorithms, and computational tools. Various optimization methods, such as gradient-based approaches, evolutionary algorithms, and interior-

doi:10.6342/NTU202303980

point methods, are employed to efficiently navigate the vast search space and find near-optimal solutions. The choice of the appropriate optimization technique depends on the problem's characteristics, such as its dimensionality, smoothness, and the presence of convexity or nonlinearity.

Despite the challenges posed by the continuous nature of real-valued optimization problems, modern advancements in computational power and optimization algorithms have significantly enhanced our ability to tackle increasingly complex and high-dimensional optimization tasks. Moreover, the continuous nature of the variables in real-valued optimization renders it well-suited for problems where small changes in the decision variables can lead to significant improvements or consequences.

4.1.1 Benchmark Functions

To evaluate the performance and efficacy of these real-valued optimization methods, benchmark functions play a crucial role. Among the widely recognized and extensively used benchmark sets, the CEC (IEEE Congress on Evolutionary Computation) benchmark functions hold a prominent position.

The CEC benchmark functions were introduced to provide a standardized platform for comparing and evaluating the performance of different optimization algorithms. These functions are specifically designed to pose challenges and assess the capabilities of optimization algorithms in solving a wide range of optimization problems.

In this work, we tested StatMax and other MAB methods on CEC2005 benchmark functions. The CEC2005 benchmark functions consist of 25 single-objective optimization problems. Each problem represents a different function with a specific mathematical formulation. These functions encompass various dimensions and complexities, allowing researchers and practitioners to evaluate the performance of optimization algorithms across different scenarios.

The objective of the CEC2005 benchmark functions is to find the global or near-global optimum within a specified search space. The functions are known for their challenging landscapes, which include multiple local optima, plateaus, and steep valleys. This complexity ensures that optimization algorithms are thoroughly tested and evaluated, providing insights into their ability to explore and exploit the search space effectively.

The CEC2005 benchmark functions have gained significant recognition and have been widely adopted by researchers and practitioners in the field of evolutionary computation and optimization. By providing a standardized benchmarking plat-

doi:10.6342/NTU202303980

form, these functions facilitate fair and objective comparisons between different algorithms, enabling advancements in the field.

4.1.2 Definition of Benchmark Functions

The CEC2005 benchmark functions consist of 25 single-objective optimization problems, each representing a different function with a specific mathematical formulation. Here is a brief introduction to each of the functions:

- F1: Shifted Sphere Function

  This function represents a shifted version of the standard sphere function, which is a simple and convex function.

- F2: Shifted Schwefel's Problem 1.2

  Schwefel's Problem 1.2 is a widely used benchmark function that exhibits strong nonlinearity

- F3: Shifted Rotated High Conditioned Elliptic Function

  This function involves a highly conditioned elliptic function that is subjected to shifting and rotation.

- F4: Shifted Schwefel's Problem 1.2 with Noise in Fitness

  Similar to F2, this function incorporates noise in the fitness evaluation, introducing additional challenges to optimization algorithms.

- F5: Schwefel's Problem 2.6

  Schwefel's Problem 2.6 is a unimodal, scalable function.

- F6: Shifted Rosenbrock's Function

  Rosenbrock's function, also known as the 'banana function', is a popular non-convex and nonsmooth optimization problem.

- F7: Shifted Rotated Griewank's Function without Bounds

  This function involves the Griewank's function, which combines both quadratic and sinusoidal terms and is subjected to shifting and rotation.

- F8: Shifted Rotated Ackley's Function with Global Optimum on Bounds

  Ackley's function is a well-known benchmark function characterized by multiple local optima.

doi:10.6342/NTU202303980

- F9: Shifted Rastrigin's Function

  The Rastrigin's function is a multimodal function with a large number of narrow, regularly distributed local optima.

- F10: Shifted Rotated Rastrigin's Function

  This function extends F9 by introducing shifting and rotation operations, enhancing its complexity.

- F11: Shifted Rotated Weierstrass Function

  The Weierstrass function is a continuous but non-differentiable function that exhibits fractal-like properties.

- F12: Schwefel's Problem 2.13 Schwefel's Problem 2.13 is a complex multimodal function that poses challenges due to its irregular landscape.

- F13: Shifted Expanded Griewank's plus Rosenbrock's Function

  This function combines the characteristics of the Griewank's and Rosenbrock's functions, introducing additional complexities.

- F14: Shifted Rotated Expanded Scaffer's F6 Function

  The Scaffer's F6 function is a unimodal function with plateaus and steep ridges, which are further enhanced through shifting and rotation.

- F15: Hybrid Composition Function 1

  This function combines multiple shifted and rotated basic functions, creating a highly multimodal and challenging optimization problem.

- F16: Hybrid Composition Function 2

  Similar to F15, this function combines multiple basic functions in a shifted and rotated manner, resulting in a complex optimization problem.

- F17: Hybrid Composition Function 3

  The third hybrid composition function combines multiple basic functions to create a highly multimodal optimization landscape.

- F18: Hybrid Composition Function 4

  This function incorporates a combination of basic functions, including shifted and rotated versions, resulting in a challenging optimization problem.

doi:10.6342/NTU202303980

- F19: Hybrid Composition Function 5

  Similar to previous hybrid composition functions, this function combines multiple basic functions to create a complex multimodal landscape.

- F20: Rotated Hybrid Composition Function 1

  This function involves a rotation operation applied to the first hybrid composition function, introducing further complexities.

- F21: Rotated Hybrid Composition Function 2

  Similarly, this function applies rotation to the second hybrid composition function, enhancing its difficulty.

- F22: Rotated Hybrid Composition Function 3

  The third rotated hybrid composition function incorporates rotation to create a diverse and challenging optimization problem.

- F23: Rotated Hybrid Composition Function 4

  This function extends the concept of rotated hybrid composition functions, presenting a complex multimodal optimization landscape.

- F24: Rotated Hybrid Composition Function 5

  Similarly, this function incorporates rotation into the fifth hybrid composition function, creating a challenging optimization problem.

- F25: Non-Continuous Rotated Hybrid Composition Function

  This function introduces non-continuity to the rotated hybrid composition function, adding an additional level of complexity.

The CEC2005 benchmark functions were specifically designed to assess the capabilities of optimization algorithms in solving complex problems. They cover a wide range of characteristics, including multimodality, nonlinearity, and irregular landscapes, making them representative of real-world optimization challenges.

4.1.3 Optimizer

In this work, we use Covariance Matrix Adaptation Evolution Strategy (CMA-ES) as an optimizer instance. CMA-ES is a powerful optimization algorithm that belongs to the family of evolutionary algorithms. It is specifically designed for solving continuous optimization problems, particularly in the field of black-box optimization, where little or no information about the objective function is available.

doi:10.6342/NTU202303980

CMA-ES operates based on the principles of natural evolution, inspired by the concept of Darwinian evolution and survival of the fittest. The algorithm maintains a population of candidate solutions, referred to as individuals, and iteratively updates their values in search of the optimal solution.

The key feature that sets CMA-ES apart from traditional evolutionary algorithms is its ability to dynamically adjust the search distribution, represented by the covariance matrix, to match the landscape of the objective function. This adaptation mechanism allows CMA-ES to efficiently explore the search space and converge towards promising regions.

In each iteration, CMA-ES estimates the distribution parameters, including the mean vector and covariance matrix, based on the fitness values of the candidate solutions. By modeling the distribution of the population, CMA-ES generates new candidate solutions that are biased towards areas of high fitness. This process combines exploration and exploitation, enabling the algorithm to strike a balance between thorough exploration of the search space and exploitation of promising solutions.

The pipeline of the CMA-ES consists of several steps that are executed iteratively until a termination condition is met. Here is a high-level explanation of the CMA-ES pipeline:

1. Initialization:

   Set the initial mean vector as the starting point in the search space. Initialize the covariance matrix to an identity matrix or a diagonal matrix with equal diagonal elements. Set the population size, step size, and other algorithmic parameters.

2. Sample Candidate Solutions:

   Generate a population of candidate solutions (also known as individuals) by sampling from a multivariate Gaussian distribution. The mean vector and covariance matrix determine the distribution parameters. Each candidate solution is represented as a vector of variables in the search space.

3. Evaluate Fitness:

   Evaluate the fitness or objective function value for each candidate solution. The fitness function determines the quality or performance of a solution. The fitness values provide the basis for selection and adaptation in subsequent steps.

doi:10.6342/NTU202303980

4. Update Distribution Parameters:

   Estimate the new mean vector and covariance matrix based on the fitness values of the candidate solutions. The mean vector is updated by considering the weighted average of the solutions. The covariance matrix is updated to capture the dependencies and correlations between variables.

5. Adapt Step Size:

   Adjust the step size or exploration parameter of the distribution to control the search behavior. The step size determines the scale of the distribution and affects the balance between exploration and exploitation. Adaptive mechanisms, such as cumulative path length adaptation, are employed to dynamically adjust the step size.

6. Generate New Candidate Solutions:

   Generate a new population of candidate solutions by sampling from the updated distribution. The new candidates are biased towards regions of higher fitness, promoting the exploitation of promising solutions. The process accounts for the updated mean vector and covariance matrix.

7. Termination:

   Check termination conditions to determine if the algorithm should stop. Termination conditions can be based on the number of iterations, fitness threshold, or other criteria. If the termination condition is not met, return to step 2 and continue the iterations.

By iteratively refining the distribution and exploring the search space, CMA-ES aims to converge toward the optimal solution of the optimization problem.

4.1.4 Integration with StatMax

For real-valued optimization, we initiate multiple optimization instances at different points and treat each instance as an arm. At each stage, StatMax chooses one of the instances and calls the instance to perform a step of optimization. The process is described in the below figures:

doi:10.6342/NTU202303980

First, the algorithm located several arms at the problem landscape, as Figure 4.1:

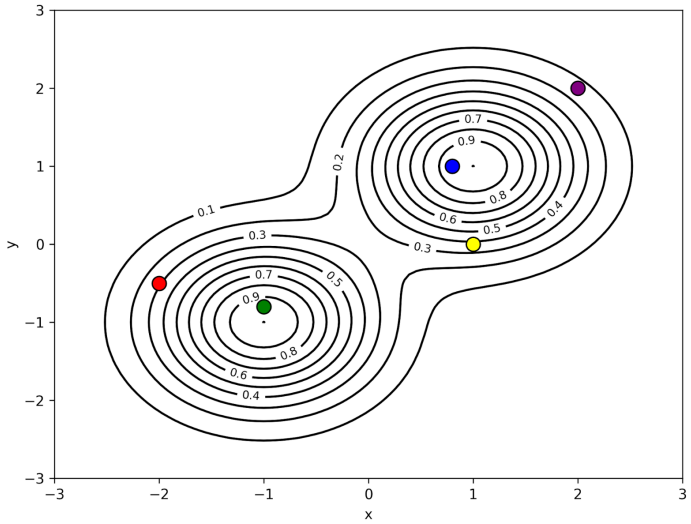


Figure 4.1: Initializing CMA-ES Population, Each Colored Dot Represents An Arm

Every arm would be equally sampled several times, then StatMax would construct the surrogate model for each arm and select the arm with the highest indicator value until the end of optimization. As Figure 4.2:

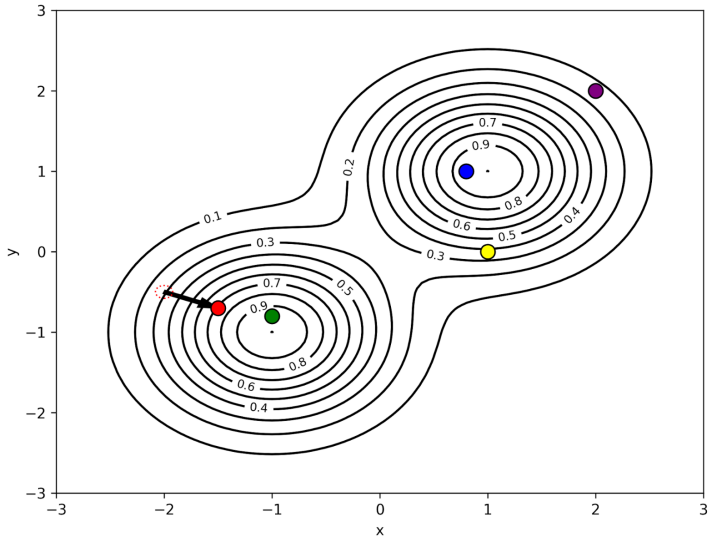


Figure 4.2: Evolution of CMA-ES Population, The Selected Arm Evolves for One Generation.

To see that this scenario fits in our settings, we describe the three properties below:

doi:10.6342/NTU202303980

- Non-stationarity: For each CMA-ES optimizer k, it would have a point $C_k(t)$ which represents the centre of its optimization at time t. When an optimizer is selected, it would perform one step of optimization, sample multiple points from a multi-variate normal distribution with centre $C_k(t)$ and move the centre along the optimization path as Figure 4.2. In this way, the reward distribution of each arm R_k is related with $C_k(t)$, such that $R_k = R_k(C_k(t), t)$, which makes the problem non-stationary.

- Unknown bound: The bound for the optimization is unknown for the algorithm.

- Extreme reward: The goal for the MAB algorithm is to find the highest objetive value.

4.1.5 Experiments

For CEC2005 benchmark problems, we use a 10-dimensional version of each problem for testing. Since CMA-ES is an optimizer for minimizing, we transformed the problem to maximization by applying negation to all benchmark functions. To better represent our result, we first run multiple CMA-ES instances for sufficiently long iterations to make sure at least one of them reaches optimal. Then, the performance of bandit algorithms is measured as the percentage of such optimal value within limited budgets. For the CMA-ES optimizer, each arm is initialized with the parameters below:

1. $\lambda = 20$, $\mu = 10$

2. initial step size $\rho = 1$

3. population size $p = 10$

The definition of the parameters follows from the original work of CMA-ES [28]. The setting of the parameters is based on previous works for better optimization results.

We tested the robustness of the parameters by examining 100 CMA-ES instances on all 25 benchmark problems with a sampling budget of 2000, and the error values are as shown in the following table, which is aligned with previous results [35] except for problem F7. The error table is shown in Table 4.1. The column with a minus sign represents that the optimizer has reached the optimal value.

doi:10.6342/NTU202303980

Table 4.1: The Error Table of Clustered CMA-ES on CEC2005

Problem	F1	F2	F3	F4	F5
Error	-	-	-	-	-
Problem	F6	F7	F8	F9	F10
Error	-	1.228e+03	2.000e+01	1.990e+00	3.980e+00
Problem	F11	F12	F13	F14	F15
Error	3.317e-04	-	4.792e-01	3.153e+00	1.034e+02
Problem	F16	F17	F18	F19	F20
Error	1.027e+02	1.020e+02	3.000e+02	3.000e+02	3.000e+02
Problem	F21	F22	F23	F24	F25
Error	3.000e+02	7.296e+02	5.595e+02	2.000e+02	2.000e+02

doi:10.6342/NTU202303980

Experiment Results

In the experiment, we followed the following bandit algorithm settings:

1. number of arms $N = 100$
2. total budget $B = 2000$

Table 4.2 shows the result of optimization. The value represents the highest sampled value as a proportion of the best-optimized value.

Table 4.2: Comparing Bandit Algorithms in Real-Valued Optimization

Problem	UCB	SW-UCB	UCB-V	UCB-Normal	Greedy	MaxMedian	Qomax	Uniform	StatMax (Ours)
F1	0.995	0.993	0.996	0.996	**1.000**	**1.000**	0.995	0.996	0.999
F2	0.997	0.997	0.997	0.997	**1.000**	**1.000**	0.996	0.997	**1.000**
F3	0.999	0.999	**1.000**	**1.000**	**1.000**	**1.000**	0.999	0.999	0.999
F4	0.994	0.994	0.995	0.995	**1.000**	**1.000**	0.996	0.997	**1.000**
F5	0.991	0.959	0.994	0.994	**0.999**	**0.999**	0.945	0.965	**0.999**
F6	0.999	0.999	**1.000**	**1.000**	0.999	**1.000**	0.999	0.999	0.999
F7	0.989	0.972	0.990	0.990	**1.000**	**1.000**	0.976	0.984	**1.000**
F8	**0.788**	**0.788**	**0.788**	**0.788**	0.714	0.725	0.750	**0.788**	0.763
F9	0.840	0.840	0.840	0.840	**0.969**	0.950	0.836	0.843	0.950
F10	0.912	0.911	0.912	0.912	0.987	0.961	0.917	0.919	**0.991**
F11	0.520	0.520	0.520	0.520	0.812	**0.837**	0.520	0.520	**0.837**
F12	0.990	0.990	0.991	0.991	**1.000**	0.997	0.988	0.986	**1.000**
F13	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**
F14	0.614	0.614	0.614	0.614	0.673	0.813	0.614	0.654	**0.832**
F15	0.810	0.767	0.810	0.810	0.868	0.826	0.801	0.788	**0.947**
F16	0.950	0.950	0.950	0.950	0.969	**0.993**	0.944	0.944	0.991
F17	0.922	0.906	0.922	0.922	0.915	**0.994**	0.911	0.911	0.986
F18	0.793	0.795	0.795	0.795	0.813	0.717	0.803	0.795	**0.978**
F19	0.793	0.787	0.793	0.793	0.886	0.783	0.792	0.808	**1.000**
F20	0.797	0.797	0.797	0.797	0.810	0.836	0.793	0.797	**0.996**
F21	0.646	0.646	0.646	0.646	0.810	0.540	0.647	0.651	**0.835**
F22	0.998	0.998	0.998	0.998	0.998	**0.999**	0.998	0.998	**0.999**
F23	0.750	0.750	0.750	0.750	0.700	0.717	0.739	0.737	**0.780**
F24	0.966	0.688	**1.000**	**1.000**	0.969	**1.000**	0.703	0.769	**1.000**
F25	0.845	0.712	0.845	0.845	0.971	**1.000**	0.738	0.785	**1.000**

From the chart above, StatMax reached optimal in most of the problems and has the highest average performance within such limited budgets. We can see that StatMax performs decently in general, having the highest score of 18 among 25 test cases. We put the learning curves for each problem in Appendix A.

doi:10.6342/NTU202303980

4.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [29] is a powerful algorithmic framework and search strategy that has gained significant attention and success in solving complex decision-making problems in various domains. It has particularly excelled in challenging games and combinatorial optimization tasks. The inherent flexibility and adaptability of MCTS make it an appealing choice for problems with high uncertainty and vast search spaces.

The core idea behind Monte Carlo Tree Search is to build a search tree by repeatedly sampling and evaluating potential moves or actions through randomized simulations, known as Monte Carlo rollouts. By progressively expanding the search tree and dynamically allocating computational resources to promising moves, MCTS focuses on exploring the most promising areas of the search space, thereby improving its decision-making efficiency.

The MCTS algorithm comprises four fundamental steps: selection, expansion, simulation, and backpropagation. During the selection step, the algorithm navigates the search tree based on exploration and exploitation heuristics to identify the most promising node to expand. The expansion step involves adding new child nodes to the selected node, representing potential moves or actions. Subsequently, simulations or rollouts are performed from these newly expanded nodes, randomly sampling actions until a terminal state or a predetermined depth is reached. The results of the simulations are then backpropagated up the tree, updating the statistics and values associated with each node based on the outcome of the rollouts.

One of the key advantages of MCTS is its ability to balance exploration and exploitation, allowing it to efficiently explore the search space while exploiting promising moves. The algorithm dynamically adjusts its exploration-exploitation trade-off based on statistical information collected during the search process, progressively converging to optimal or near-optimal solutions.

MCTS has demonstrated remarkable success in solving complex games, such as Go, chess, and shogi, surpassing human performance in some cases. It has also been applied to various real-world problems, including robotics, scheduling, and resource allocation, showcasing its versatility and effectiveness beyond game domains.

However, it is important to note that MCTS is computationally intensive, and its performance heavily relies on the quality of the domain-specific heuristics, the representation of the problem, and the available computational resources. Consequently, researchers continue to explore and refine different variants and extensions of MCTS to address specific problem characteristics and enhance its performance.

In conclusion, Monte Carlo Tree Search is a powerful algorithmic framework

doi:10.6342/NTU202303980

that excels in solving complex decision-making problems with high uncertainty and large search spaces. By intelligently balancing exploration and exploitation, MCTS has achieved remarkable success in game-playing and optimization domains. With ongoing research and advancements, MCTS holds the potential to revolutionize decision-making processes further and address challenging real-world problems in diverse fields.

4.2.1 Benchmark Functions

We tested StatMax with MCTS on two schemes: one-player game and two-player game. Below is the description of the two schemes:

One-Player Scheme

In the one-player scheme, we utilize a k-ary tree to demonstrate the concept of using StatMax with MCTS. Each node in the k-ary tree represents a game state, and the tree is constructed based on the possible moves available from each state. At the leaf nodes of the tree, we have probability distributions representing the possible rewards or outcomes of sampling from those distributions. When a node is selected during the MCTS exploration phase, the algorithm samples from the corresponding probability distribution to obtain a reward. This reward is then used to update the statistics of the selected node during the backpropagation phase, enabling the algorithm to make more informed decisions over time.

Two-Player Scheme

In the two-player scheme, we select a reversi game as our benchmark. Reversi is played on an 8x8 grid, where two players take turns placing their discs (black and white) on the board. The objective is to have the majority of one's discs of one's color facing up at the end of the game.

4.2.2 Integration with StatMax

For MCTS, we adopt a training procedure similar to upper confidence bounds applied to trees (UCT) methods. While we change the selection algorithm to StatMax, treating each child node as an arm. And update the surrogate distribution through the backpropagate step.

For the one-player scheme, the performance is defined as the maximum reward sampled in the budget limit. For the two-player scheme, the performance is defined

doi:10.6342/NTU202303980

as the win ratio. At each MCTS round, the procedure is composed of four steps: selection, expansion, playout, and backpropagation. A brief introduction is described in the below figures:

Selection: at each round of MCTS, the agent starts playing the game from the root node, and StatMax would choose one of the children nodes if every children node is visited, as Figure 4.3 shows.

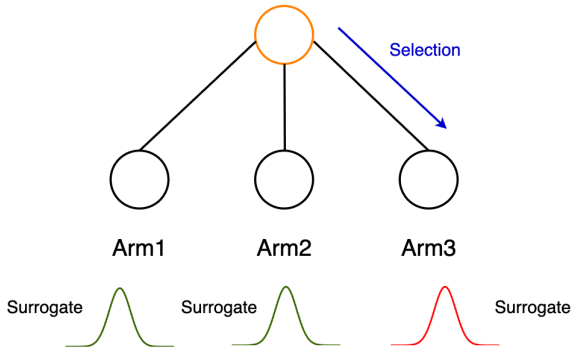


Figure 4.3: Selection in StatMax with MCTS

Expansion: after StatMax selects the child node, the agent moves one layer and repeats the selection until there is at least one child node unvisited, as Figure 4.4:

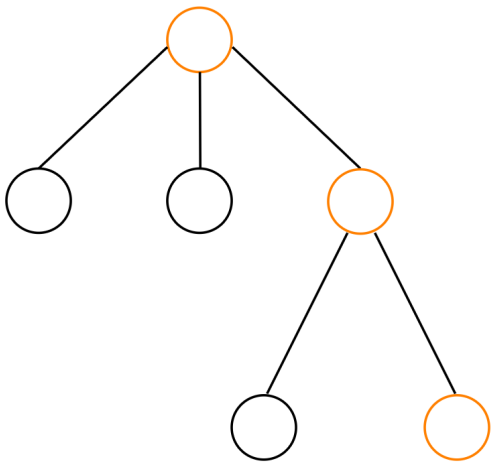


Figure 4.4: Expansion in StatMax with MCTS

doi:10.6342/NTU202303980

Playout: the agent would then randomly explore until it reaches the leaf node.

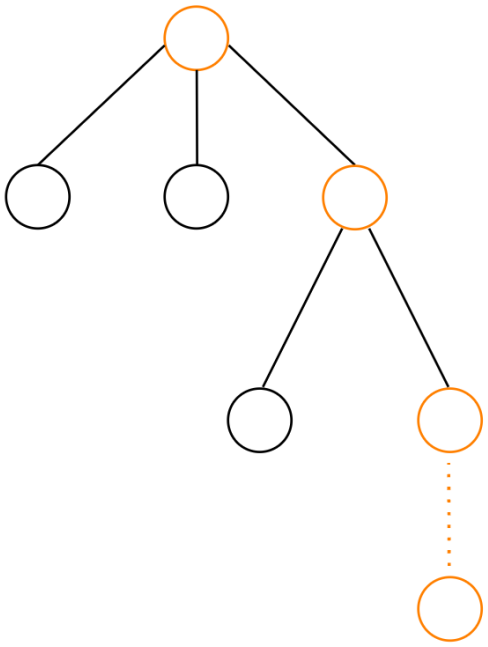


Figure 4.5: Playout in StatMax with MCTS

Backpropagation: when the agent visits a leaf node, a reward is sampled from its reward distribution, and the reward is backpropagated along the agent's path, updating the surrogate distributions of visited nodes. As Figure 4.6:

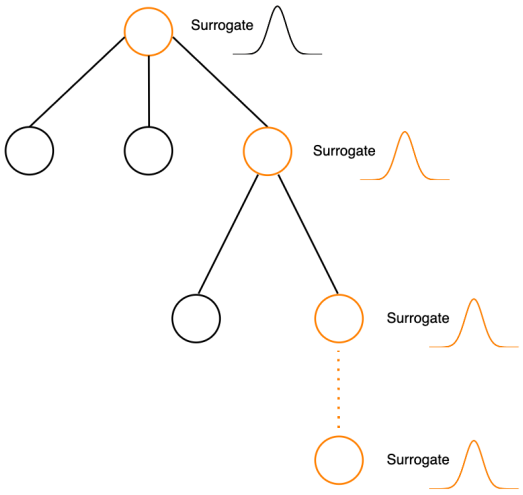


Figure 4.6: Backprop in StatMax with MCTS

doi:10.6342/NTU202303980

The pseudo-code for MCTS can be described as Algorithm 10 :

Algorithm 10: Monte Carlo Tree Search (MCTS)

Input: Current game state s, maximum simulation depth D, exploration constant C, number of simulations N

Output: Recommended action a

```
Initialize the root node V with state s and an empty list of child nodes;
Set n to 0, the number of iterations performed so far;
while n < N do
    v ← Select(V) ;          // Select a node to expand and simulate
    if v is not terminal and not fully expanded then
        Expand(v) ;                  // Add one unexplored child to v
    end
    u ← the last visited child of Expand operation;
    Backpropagate(u) ;  // Backpropagate the simulation result up
     the tree
    n ← n + 1 ;                    // Increment the iteration counter
end
a ← BestAction(V) ;      // Select the best action based on visit
   counts
return a;
```

4.2.3 Experiments

In this section, we tested the performance of StatMax on on single-player games and two-player games. For single-player games, these are three types of games we investigate:

1. Bounded tree: Rewards are bounded within [0, 1], represented by Beta distributions.

2. Non-Bounded tree, with same means, different deviations, represented by normal distributions.

3. Non-Bounded tree, with different means, same deviations, represented by normal distributions.

We designed a scheme for generating k-ary trees for these problems. Take a binary bounded tree for example:

doi:10.6342/NTU202303980

1. A parameter $s_{root} = 1$, $l_{root} = 0.5$ are initialized at root node.
2. The right child of root node is assigned with $s_{rc} = s_{root} + l_{root} \times 0.5 = 1.25$, $l_{rc} = l_{root} \times 0.5 = 0.25$.
3. The left child of root node is assigned with $s_{lc} = s_{root} = 1$, $l_{rc} = l_{root} \times 0.5 = 0.25$.
4. Apply the rule to each layer until the leaf node.
5. For each leaf node with its own s, l, the reward distribution is determined using them.

According to different test cases, the reward distribution would be different. For the settings of s_{root} and l_{root}, we make l_{root} way larger than s_{root} such that the rightmost leaf node have a significantly different distribution compared to the leftmost node, while neighbor nodes are similar. For two-player game, we assume the opponent as a player who utilizes UCT algorithm for decision making.

Beta Tree

The experiment settings:

1. binary tree
2. $s_{root} = 0.5$, $l_{root} = 3$
3. reward distribution beta(l_{leaf}, l_{leaf})
4. budget $B = 15000$

The results are shown below in Table 4.3:

Table 4.3: MCTS, Beta Tree

Layers	UCT	UCT-Normal	UCB-V	MaxMedian	QoMax	Uniform	StatMax (Ours)
2	0.903 ± 0.051	0.892 ± 0.047	0.888 ± 0.041	0.878 ± 0.053	0.867 ± 0.049	0.888 ± 0.057	**0.926 ± 0.032**
3	0.880 ± 0.051	0.870 ± 0.047	0.889 ± 0.056	0.862 ± 0.013	0.847 ± 0.036	0.874 ± 0.033	**0.916 ± 0.038**
4	0.862 ± 0.042	0.879 ± 0.065	0.898 ± 0.044	0.879 ± 0.051	0.849 ± 0.030	0.882 ± 0.028	**0.917 ± 0.037**
5	0.861 ± 0.036	0.841 ± 0.044	0.910 ± 0.059	0.875 ± 0.038	0.875 ± 0.045	0.856 ± 0.043	**0.918 ± 0.027**

Normal Tree, Same Mean, Different Deviations

The experiment settings:

1. binary tree

doi:10.6342/NTU202303980

2. $s_{root} = 0.5$, $l_{root} = 3$

3. reward distribution Normal(0.5, l_{leaf})

4. budget $B = 15000$

The results are shown below in Table 4.4:

Table 4.4: MCTS, Normal Tree with The Same Mean but Different Deviations

Layers	UCT	UCT-Normal	UCB-V	MaxMedian	QoMax	Uniform	StatMax (Ours)
2	3.301 ± 2.082	11.811 ± 5.141	8.319 ± 6.343	11.033 ± 5.132	9.097 ± 3.763	8.004 ± 5.601	**15.184 ± 1.623**
3	4.826 ± 5.352	9.883 ± 5.624	12.485 ± 4.309	10.221 ± 5.333	7.875 ± 4.201	11.636 ± 6.753	**16.725 ± 2.381**
4	5.211 ± 5.634	12.661 ± 5.855	10.035 ± 6.219	11.634 ± 4.712	10.331 ± 3.343	8.833 ± 3.211	**18.051 ± 2.274**
5	4.933 ± 3.531	13.541 ± 5.724	13.038 ± 6.466	9.863 ± 3.732	9.411 ± 4.361	10.642 ± 6.232	**19.081 ± 2.011**

Normal Tree, Different Mean, Same Deviations

1. binary tree

2. $s_{root} = 0.5$, $l_{root} = 3$

3. reward distribution Normal(l_{leaf}, 0.5)

4. budget $B = 15000$

The results are shown below in Table 4.5:

Table 4.5: MCTS, Normal Tree with Different Mean but The Same Deviations

Layers	UCT	UCT-Normal	UCB-V	MaxMedian	QoMax	Uniform	StatMax (Ours)
2	6.471 ± 0.162	6.471 ± 0.162	6.471 ± 0.162	**6.572 ± 0.163**	6.471 ± 0.162	6.471 ± 0.162	6.462 ± 0.163
3	**7.221 ± 0.162**	**7.221 ± 0.162**	7.220 ± 0.160	7.221 ± 0.181	**7.221 ± 0.162**	**7.221 ± 0.162**	**7.221 ± 0.162**
4	7.593 ± 0.162	7.593 ± 0.162	**7.595 ± 0.160**	7.593 ± 0.131	7.593 ± 0.162	7.593 ± 0.162	7.575 ± 0.171
5	**7.654 ± 0.151**	**7.654 ± 0.151**	**7.654 ± 0.151**	7.647 ± 0.166	**7.654 ± 0.151**	7.654 ± 0.163	7.654 ± 0.162

From the experiments above, we can observe that StatMax has advantages in normal distributions of the same means and different variations. Such a problem may represent a 'trap' for traditional MAB problems as they focus on expected rewards. In other problems, StatMax is also competitive with other algorithms, including bounded problems.

Two-Player Game

In the two-player scheme, we tested 5 algorithms including random, Maxmedian, UCB-V, StatMax and UCT + StatMax. As the table 4.6 shown, each entry records

doi:10.6342/NTU202303980

the win ratio against a player trained with UCT with the same budget. For the method UCT + StatMax, we combine the bias term of the two methods together.

Table 4.6: MCTS, Reversi Game

Budget	Random	Maxmedian	UCB-V	StatMax	UCT + StatMax
50	0%	12%	24%	36%	**62**%
100	0%	12%	30%	42%	**48**%
200	0%	0%	42%	44%	**56**%

From the table above, we can observe that UCT dominates at most of the cases, while UCT + StatMax outperforms UCT with 50 and 200 budgets may be a signal that StatMax improves overall performance.

doi:10.6342/NTU202303980

4.3 Hyperparameter Optimization

Hyperparameter optimization is a critical component of training machine learning models, as the choice of hyperparameters significantly impacts the performance and generalization of these models. However, finding the optimal hyperparameter configuration is often a challenging and time-consuming process due to the large search space and computational costs involved.

In recent years, two innovative approaches, Hyperband and BOHB (Bayesian Optimization and Hyperband), have emerged as powerful methods for efficient hyperparameter optimization. These algorithms leverage the concepts of iterative resource allocation and Bayesian optimization to accelerate the search for optimal hyperparameters.

Hyperband is a sequential optimization algorithm that applies the principle of successive halving, a bandit-based method. To efficiently explore the hyperparameter space. The algorithm begins by randomly sampling a set of hyperparameter configurations and allocating limited computational resources to train models with these configurations. After each round of training, a selection criterion is used to retain only the best-performing configurations, discarding the rest. The process is repeated iteratively, gradually increasing the resource allocation for the remaining configurations while continuing to eliminate the poorest performers. By aggressively pruning suboptimal configurations early on, Hyperband effectively focuses computational resources on promising hyperparameter combinations, significantly reducing the overall optimization time.

BOHB combines the strengths of Bayesian optimization and Hyperband to achieve even greater efficiency in hyperparameter optimization. Bayesian optimization employs probabilistic models to model the performance landscape and makes informed decisions on which configurations to explore. BOHB integrates Bayesian optimization with the iterative resource allocation of Hyperband. Initially, a small subset of configurations is explored using Bayesian optimization. Based on the performance observed, Hyperband is applied to the promising configurations, discarding underperforming ones. The combination of Bayesian optimization and Hyperband enables BOHB to efficiently explore and exploit the hyperparameter space, adapting the search to the performance landscape and converging to optimal configurations faster.

Both Hyperband and BOHB have demonstrated significant improvements in the efficiency of hyperparameter optimization across various machine learning tasks. By intelligently allocating computational resources, these algorithms reduce the number of model evaluations required and expedite the search for optimal hyperparameter

doi:10.6342/NTU202303980

configurations. This makes them particularly valuable for applications with limited computational resources or time constraints.

Hyperband and BOHB represent powerful advancements in the field of hyperparameter optimization. These algorithms leverage iterative resource allocation and Bayesian optimization to efficiently explore the hyperparameter space and find optimal configurations in a time and resource-efficient manner. By accelerating the optimization process, Hyperband and BOHB contribute to faster model development, improved performance, and enhanced productivity in the field of machine learning and data science.

4.3.1 Integration with StatMax

We integrate successive halving with StatMax for better parameter selection for the hyperband setting. Each hyperparameter configuration is treated as an arm, while the observation is defined as the validation accuracy. The arms with higher indicator values are kept at each successive halving stage. Figure 4.7 represents a uniform random search strategy where budgets are equally allocated between candidate solutions:

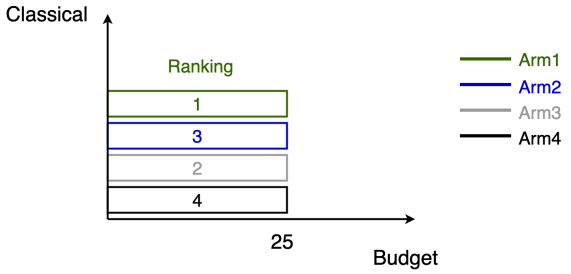


Figure 4.7: Uniform Random Search

Figure 4.8 represents the procedure of successive halving with StatMax.

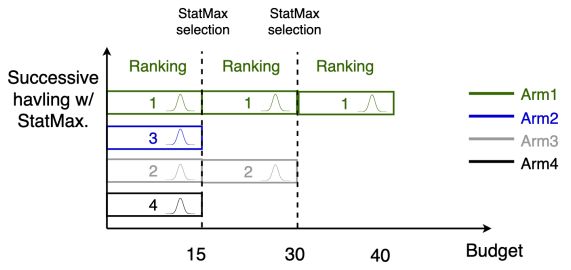


Figure 4.8: Successive Halving with StatMax

doi:10.6342/NTU202303980

4.3.2 Experiments

In this section, we tested StatMax with Hyperband and StatMax with BOHB on benchmark datasets Fashion-MNIST and CIFAR-10. Fashion-MNIST is a gray-scale dataset with ten classes of apparel, with 60000 training images and 10000 testing images. CIFAR-10 is a colored dataset with ten classes of transportation and animals. They're both world-famous benchmark datasets for computer vision. The following subsection describes our experiment setting and results.

Firstly, we choose a simple 3-layer convolutional neural network as a demonstration. We consider the following tunable parameters: learning rate, optimizer, SGD momentum, number of filters for each layer, and dropout rate. And the range for each parameter is as the Table 4.7.

Table 4.7: Experiment Setting of Hyperparameter Optimization

Parameter	Type	Range/Choices	Comment
Learning rate	float	[1e-6, 1e-2]	varied logarithmically
Optimizer	categorical	[Adam, SGD]	discrete choice
SGD momentum	float	[0, 0.99]	only active if optimizer is SGD
# of conv layers	integer	[1,3]	only take integer values
# of filters, 1^{st} conf layer	integer	[4, 64]	logarithmically varied integer values
# of filters, 2^{nd} conf layer	integer	[4, 64]	active if number of layers ≥ 2
# of filters, 3^{rd} conf layer	float	[4, 64]	active if number of layers $= 3$
Dropout rate	integer	[0, 0.9]	standard continuous parameter
# of units, FC layer	integer	[8, 256]	logarithmically varied integer values

Secondly, there are two types of experiment scenarios we consider:

1. Small: Where only 8162 training images are used, this will accelerate fine-tuning speed.

2. Full: Where all training images are used.

To fairly compare these algorithms, we fixed the random seed t make sure the candidates of hyperparameters are the same.

Fashion-MNIST

For Fashion-MNIST, no data augmentation is performed during training and inference. the results are as Table 4.8 shown:

doi:10.6342/NTU202303980

Table 4.8: Result of Hyperparameter Optimization on Fashion-MNIST

Experiment	Type	Hyperband	Hyperband + StatMax	BOHB	BOHB + UCB-V	BOHB + StatMax
Small	Validation	87.2 ± 1.4%	87.0 ± 1.1%	86.8 ± 1.6%	86.8 ± 1.5%	**87.9 ± 1.2%**
Small	Test	87.0 ± 1.0%	86.8 ± 0.9%	86.4 ± 1.4%	86.8 ± 1.0%	**87.4 ± 1.0%**
Full	Validation	90.3 ± 0.5%	90.3 ± 0.6%	90.4 ± 0.6%	**90.9 ± 0.4%**	**90.9 ± 0.1%**
Full	Test	89.5 ± 0.5%	89.5 ± 0.4%	**89.8 ± 0.7%**	89.7 ± 0.4%	**89.8 ± 0.6%**

CIFAR-10

For CIFAR-10, we adopt data augmentation, including cropping and normalization. The results are as Table 4.9 shown:

Table 4.9: Result of Hyperparameter Optimization on CIFAR-10

Experiment	Type	Hyperband	Hyperband + StatMax	BOHB	BOHB + UCB-V	BOHB + StatMax
Small	Validation	66.4 ± 2.4%	66.5 ± 2.3%	67.5 ± 2.1%	66.2 ± 1.5%	**68.9 ± 1.2%**
Small	Test	66.3 ± 2.7%	66.6 ± 2.6%	67.5 ± 2.6%	68.1 ± 1.7%	**68.6 ± 1.6%**
Full	Validation	74.7 ± 1.1%	74.2 ± 1.5%	75.5 ± 1.3%	75.7 ± 1.6%	**75.8 ± 0.2%**
Full	Test	76.5 ± 1.2%	76.8 ± 1.3%	77.7 ± 1.1%	77.6 ± 1.1%	**78.0 ± 0.7%**

The integration of StatMax within each experimental setting resulted in performance improvements in terms of accuracies, albeit of a modest nature. These enhancements were particularly evident when conducting experiments with limited training data. In contrast, the impact of StatMax on Hyperband was observed to be minor. A plausible reason for this could be the resampling of potential candidates from previous sample points employed by BOHB. Through the adoption of StatMax, the quality of candidates within the pool was enhanced, potentially contributing to improved results.

doi:10.6342/NTU202303980

Chapter 5

Conclusion

This thesis focused on developing and analyzing the StatMax algorithm, which leverages the concepts of order statistics and surrogate models. we investigate topics including real-valued optimization, MCTS, and hyperparameter optimization for neural networks. Through comprehensive comparisons with traditional MAB algorithms and extreme bandit algorithms, the effectiveness and robustness of StatMax were demonstrated, particularly under non-stationary circumstances. By incorporating order statistics, StatMax effectively captures the distributional information of rewards and makes informed decisions based on reward samples. Adaptive surrogate models allow for non-stationary environments, improving resource allocation and decision-making. The benchmark experiments showcased the algorithm's ability to handle non-stationary environments and deliver competitive results.

In terms of future work. Firstly, conducting additional theoretical analysis would enhance our understanding of the algorithm's properties and performance guarantees. Secondly, applying the StatMax algorithm to a broader range of practical problems and testing the algorithm on real-world scenarios with complex constraints and dynamics would be beneficial. Additionally, investigating the algorithm's performance in large-scale problems and analyzing its scalability would be necessary for practical implementation. Furthermore, developing schemes for handling more general distributions beyond the assumptions of Gaussian distributions would be valuable. Extending the algorithm to take non-Gaussian distributions or distributions with unknown parameters would allow for more flexible and adaptable resource allocation in diverse problem domains. This could involve incorporating techniques such as non-parametric estimation or Bayesian approaches to effectively model and handle different distributional assumptions.

doi:10.6342/NTU202303980

Bibliography

[1] Joannes Vermorel and Mehryar Mohri. "Multi-armed bandit algorithms and empirical evaluation". In: *European Conference on Machine Learning*. Springer. (2005), pp. 437–448.

[2] Sattar Vakili and Qing Zhao. "Mean-variance and value at risk in multi-armed bandit problems". In: *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE. (2015), pp. 1330–1335.

[3] Eyal Even-Dar, Shie Mannor, Yishay Mansour, and Sridhar Mahadevan. "Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems." In: *Journal of Machine Learning Research* 7.6 (2006).

[4] Eric M. Schwartz, Eric T. Bradlow, and Peter S. Fader. "Customer acquisition via display advertising using multi-armed bandit experiments". In: *Marketing Science* 36.4 (2017), pp. 500–522.

[5] Sofía S. Villar, Jack Bowden, and James Wason. "Multi-armed bandit models for the optimal design of clinical trials: Benefits and challenges". In: *Statistical Science: A Review Journal of the Institute of Mathematical Statistics* 30.2 (2015), p. 199.

[6] Nícollas Silva, Heitor Werneck, Thiago Silva, Adriano CM Pereira, and Leonardo Rocha. "Multi-armed bandits in recommendation systems: A survey of the state-of-the-art and future directions". In: *Expert Systems with Applications* 197 (2022), p. 116669.

[7] Nicolo Cesa-Bianchi and Paul Fischer. "Finite-Time regret bounds for the multiarmed bandit problem". In: *ICML*. Vol. 98. Citeseer. (1998), pp. 100–108.

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

[9] Volodymyr Kuleshov and Doina Precup. "Algorithms for multi-armed bandit problems". In: *arXiv Preprint arXiv:1402.6028* (2014).

[10] Peter Auer. "Using confidence bounds for exploitation-exploration trade-offs". In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.

[11] Olivier Chapelle and Lihong Li. "An empirical evaluation of Thompson sampling". In: *Advances in Neural Information Processing Systems* 24 (2011).

[12] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. "Contextual bandits with linear payoff functions". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. (2011), pp. 208–214.

doi:10.6342/NTU202303980

[13] Aleksandrs Slivkins. "Contextual bandits with similarity information". In: *Proceedings of the 24th annual Conference On Learning Theory*. JMLR Workshop and Conference Proceedings. (2011), pp. 679–702.

[14] Omar Besbes, Yonatan Gur, and Assaf Zeevi. "Stochastic multi-armed-bandit problem with non-stationary rewards". In: *Advances in neural information processing systems* 27 (2014).

[15] Robin Allesiardo, Raphaël Féraud, and Odalric-Ambrym Maillard. "The non-stationary stochastic multi-armed bandit problem". In: *International Journal of Data Science and Analytics* 3 (2017), pp. 267–283.

[16] John Langford and Tong Zhang. "The epoch-greedy algorithm for multi-armed bandits with side information". In: *Advances in neural information processing systems* 20 (2007).

[17] Shie Mannor and Ohad Shamir. "From bandits to experts: On the value of side-observations". In: *Advances in Neural Information Processing Systems* 24 (2011).

[18] Nicolo Cesa-Bianchi and Gábor Lugosi. "Combinatorial bandits". In: *Journal of Computer and System Sciences* 78.5 (2012), pp. 1404–1422.

[19] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. "Learning diverse rankings with multi-armed bandits". In: *Proceedings of the 25th International Conference on Machine Learning*. (2008), pp. 784–791.

[20] Arnab Maiti, Vishakha Patil, and Arindam Khan. "Multi-armed bandits with bounded arm-memory: Near-optimal guarantees for best-arm identification and regret minimization". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19553–19565.

[21] Xiaoguang Huo and Feng Fu. "Risk-aware multi-armed bandit problem with application to portfolio selection". In: *Royal Society Open Science* 4.11 (2017), p. 171377.

[22] Afef Ben Hadj Alaya-Feki, Eric Moulines, and Alain LeCornec. "Dynamic spectrum access with non-stationary multi-armed bandit". In: *2008 IEEE 9th Workshop on Signal Processing Advances in Wireless Communications*. IEEE. (2008), pp. 416–420.

[23] Alexandra Carpentier and Michal Valko. "Extreme bandits". In: *Advances in Neural Information Processing Systems* 27 (2014).

[24] Vincent A. Cicirello and Stephen F. Smith. "The max k-armed bandit: A new model of exploration applied to search heuristic selection". In: *The Proceedings of the Twentieth National Conference on Artificial Intelligence*. Vol. 3. (2005), pp. 1355–1361.

[25] Sujay Bhatt, Ping Li, and Gennady Samorodnitsky. "Extreme bandits using robust statistics". In: *IEEE Transactions on Information Theory* 69.3 (2022), pp. 1761–1776.

[26] Dorian Baudry, Yoan Russac, and Emilie Kaufmann. "Efficient algorithms for extreme bandits". In: *arXiv Preprint arXiv:2203.10883* (2022).

[27] Ponnuthurai N. Suganthan, Nikolaus Hansen, Jing J. Liang, Kalyanmoy Deb, Ying-Ping Chen, Anne Auger, and Santosh Tiwari. "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization". In: *KanGAL Report* 2005005.2005 (2005), p. 2005.

[28] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)". In: *Evolutionary Computation* 11.1 (2003), pp. 1–18.

doi:10.6342/NTU202303980

[29] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[30] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.

[31] Stefan Falkner, Aaron Klein, and Frank Hutter. "BOHB: Robust and efficient hyperparameter optimization at scale". In: *International Conference on Machine Learning*. PMLR. (2018), pp. 1437–1446.

[32] Levente Kocsis and Csaba Szepesvári. "Bandit based monte-carlo planning". In: *European Conference on Machine Learning*. Springer. (2006), pp. 282–293.

[33] Aurélien Garivier and Eric Moulines. "On upper-confidence bound policies for switching bandit problems". In: *International Conference on Algorithmic Learning Theory*. Springer. (2011), pp. 174–188.

[34] J.P. Royston. "Algorithm AS 177: Expected normal order statistics (exact and approximate)". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 31.2 (1982), pp. 161–165.

[35] Christian L. Muller, Benedikt Baumgartner, and Ivo F. Sbalzarini. "Particle swarm CMA evolution strategy for the optimization of multi-funnel landscapes". In: *2009 IEEE Congress on Evolutionary Computation*. (2009), pp. 2685–2692.

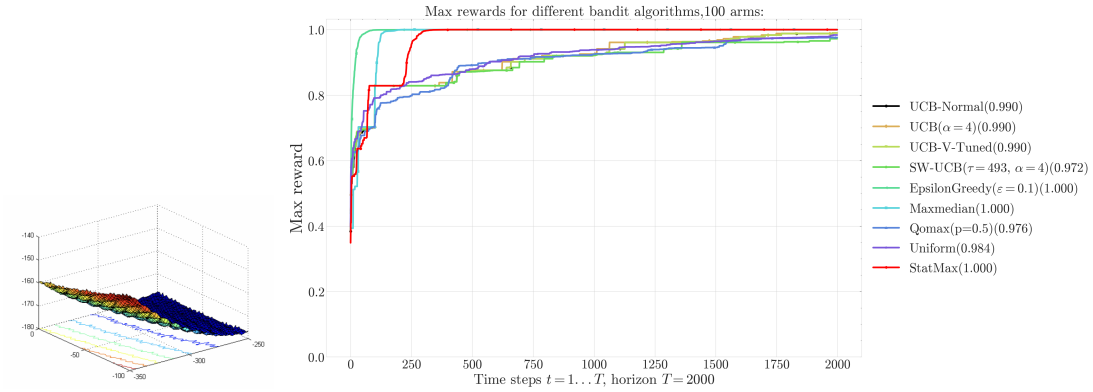
doi:10.6342/NTU202303980

Appendix A

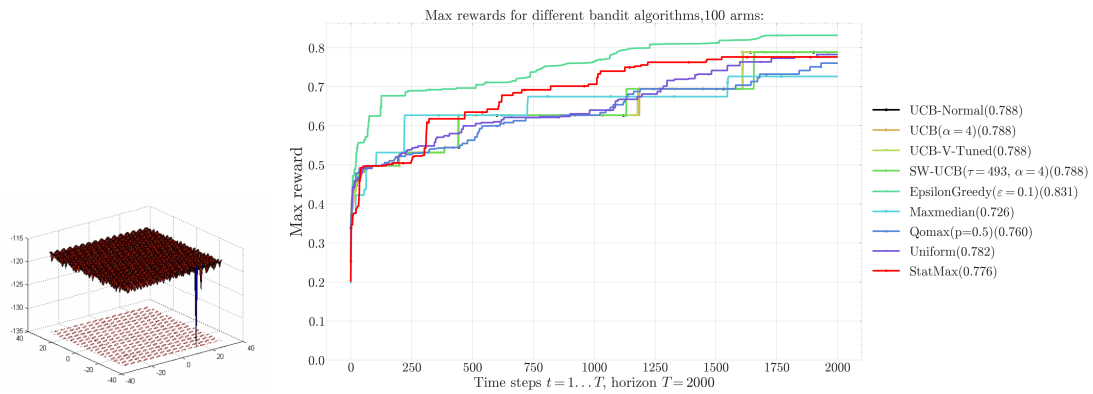
Learning curve of CEC2005 benchmark functions

In this appendix, we show the learning curve of multi-modal CEC2005 benchmark functions (F7 to F25).

- F7:

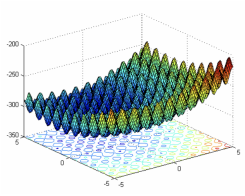


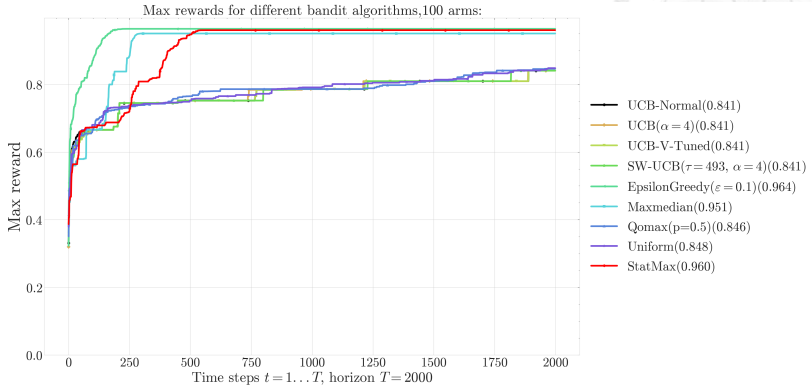
- F8:



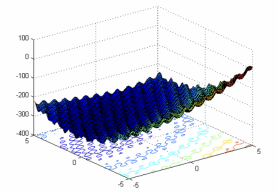
doi:10.6342/NTU202303980

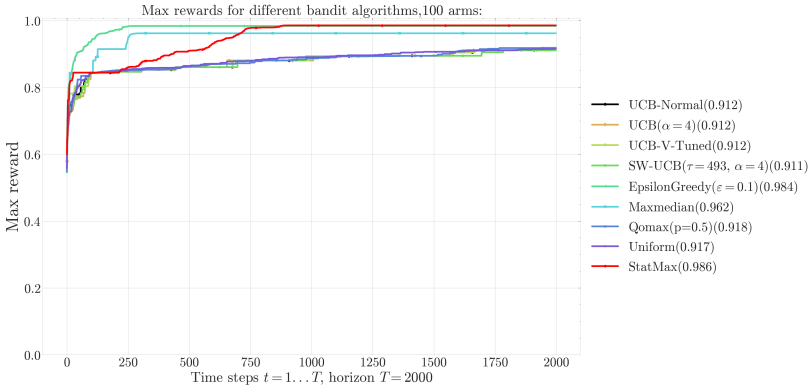
- F9:



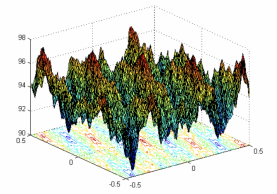


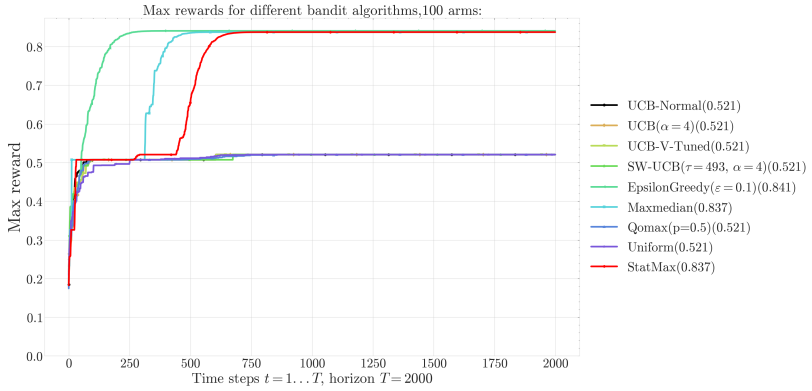
- F10:





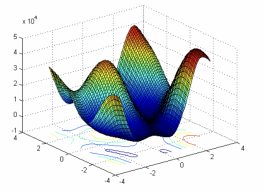
- F11:

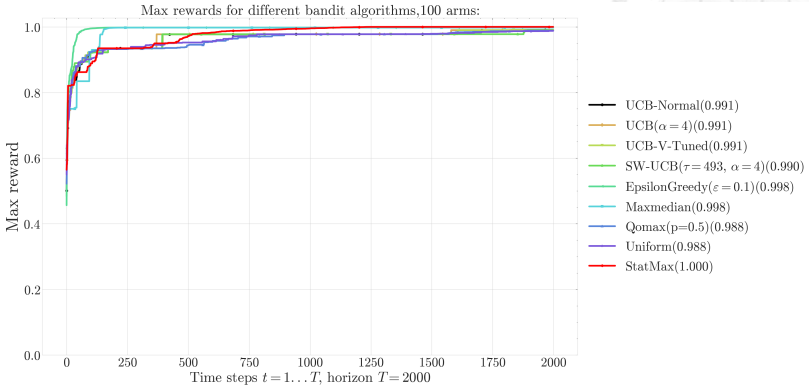




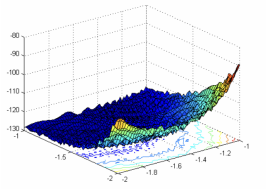
doi:10.6342/NTU202303980

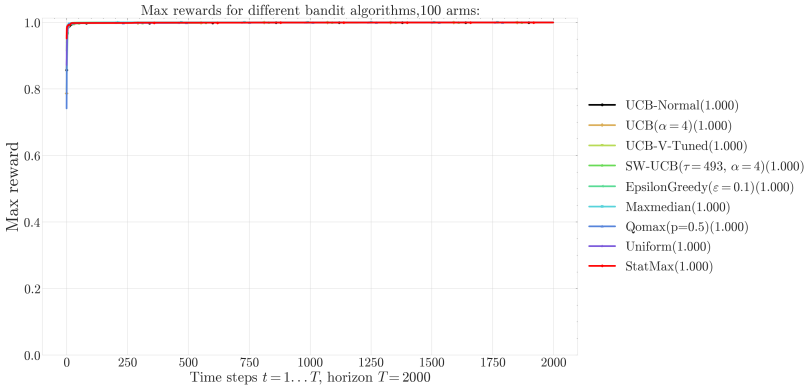
- F12:



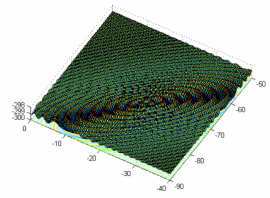


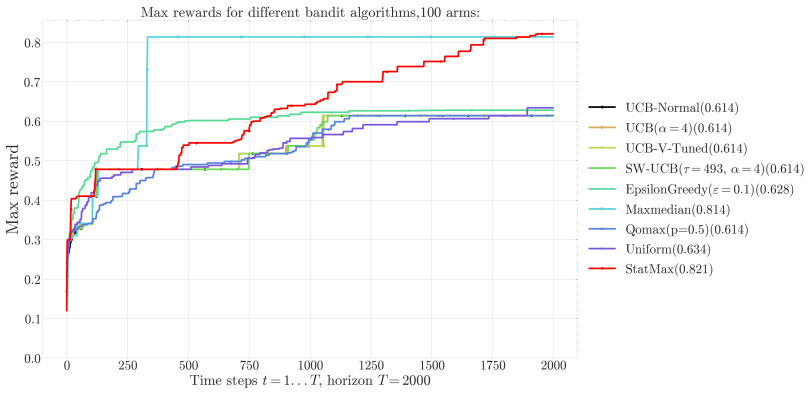
- F13:





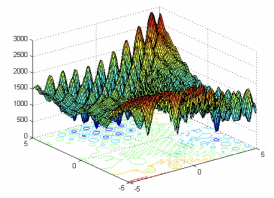
- F14:

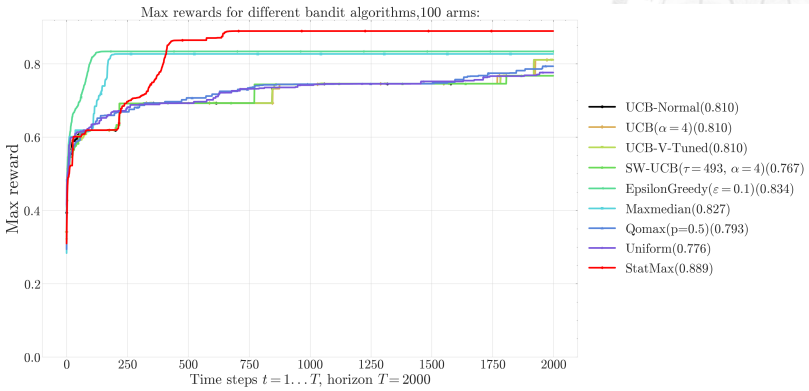




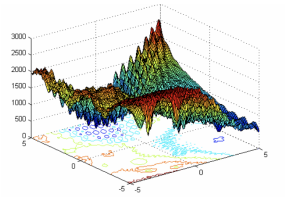
doi:10.6342/NTU202303980

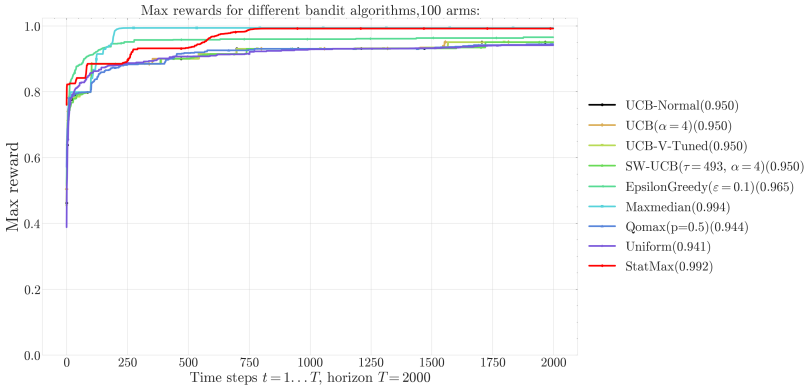
- F15:



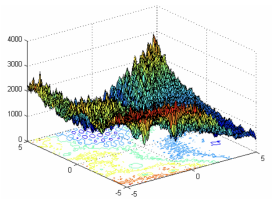


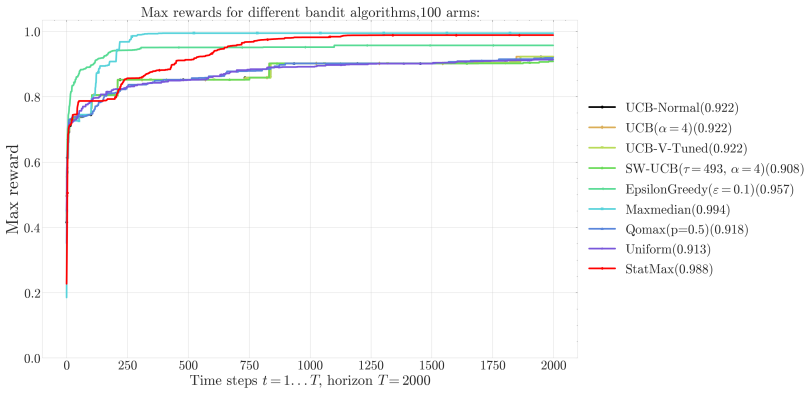
- F16:





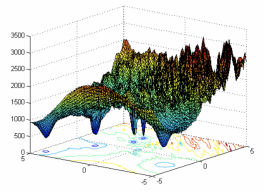
- F17:

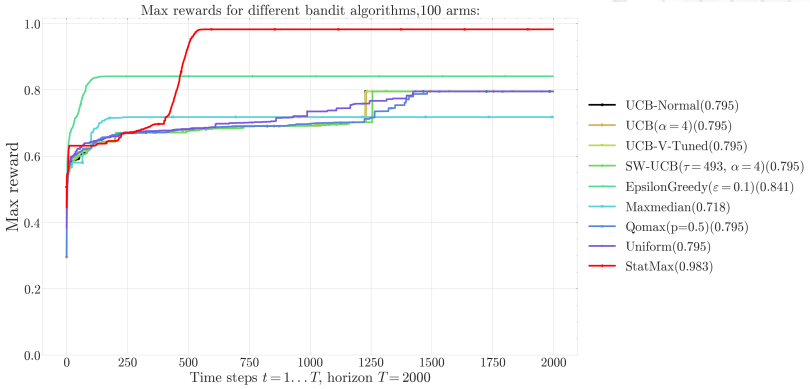




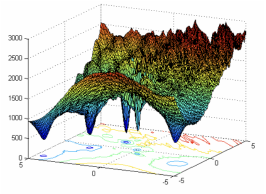
doi:10.6342/NTU202303980

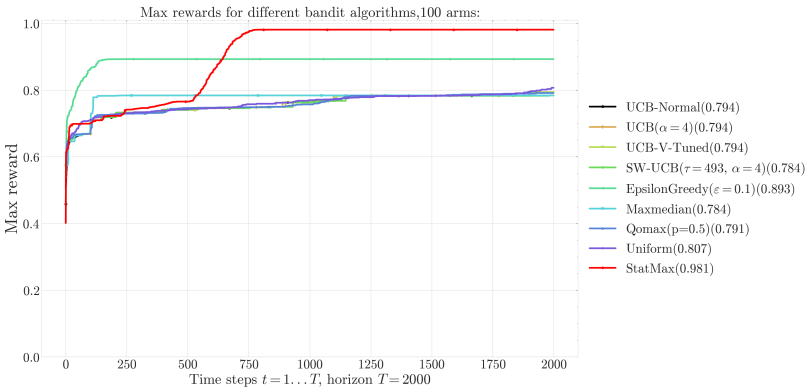
- F18:



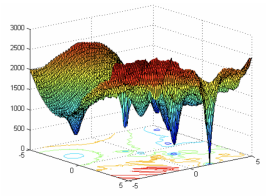


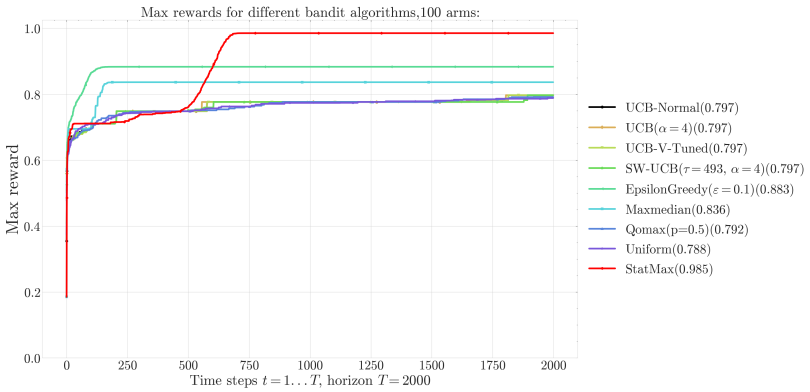
- F19:





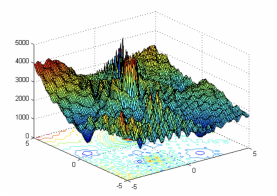
- F20:

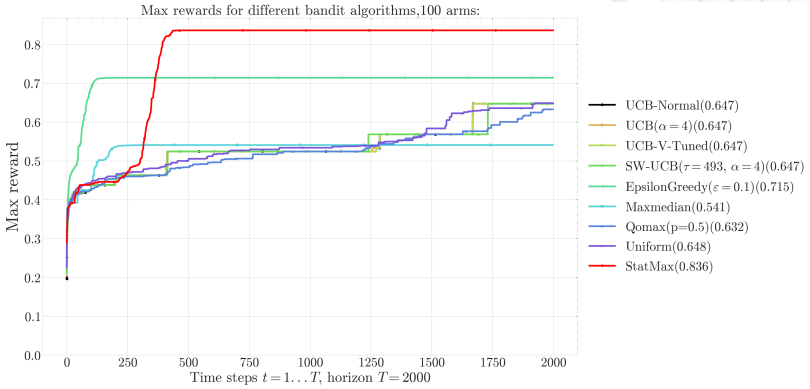




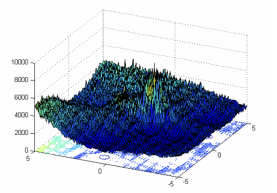
doi:10.6342/NTU202303980

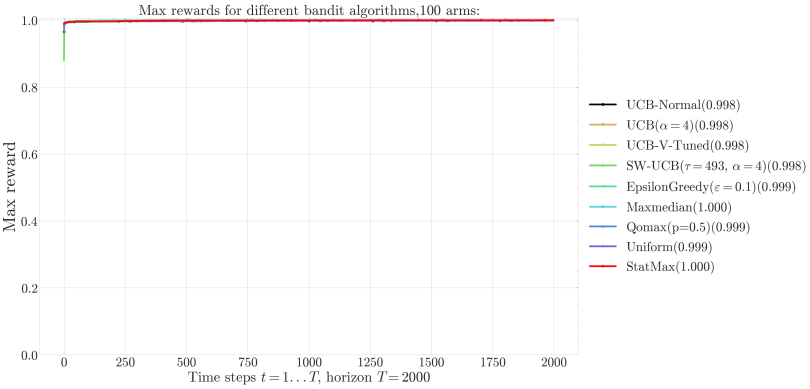
- F21:



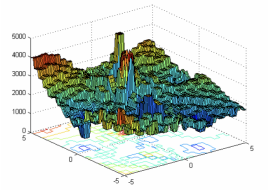


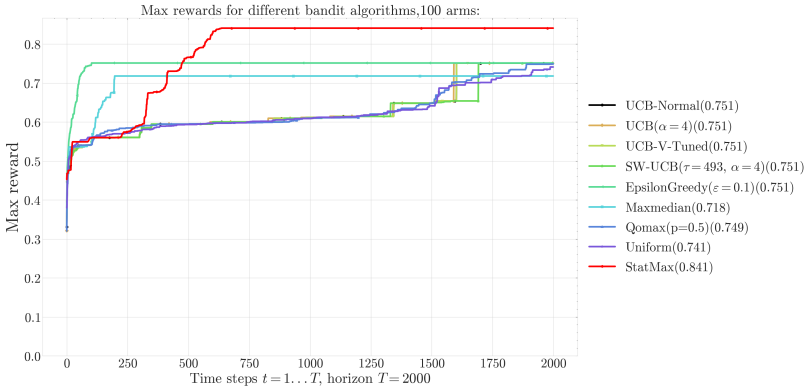
- F22:





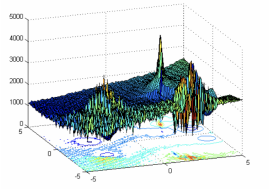
- F23:

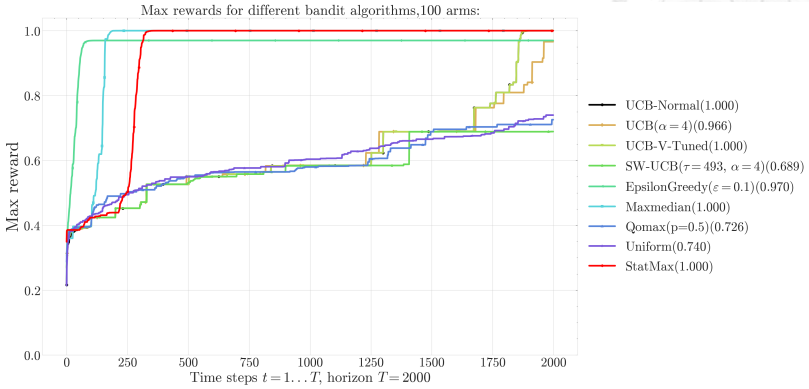




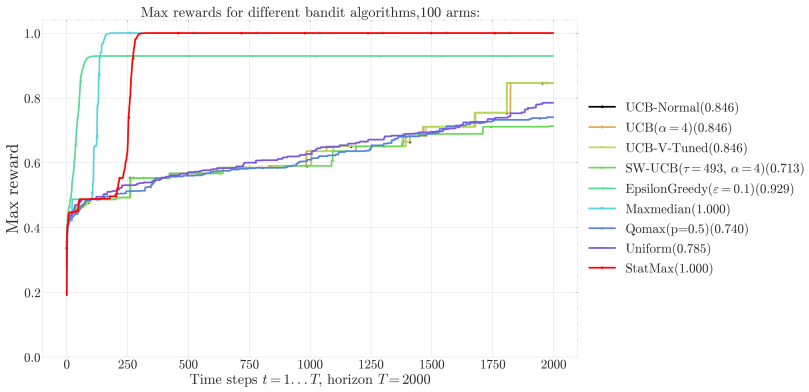
doi:10.6342/NTU202303980

- F24:





- F25:



doi:10.6342/NTU202303980