Thompson Sampling for Monte Carlo Tree Search and Maxi-min Action Identification

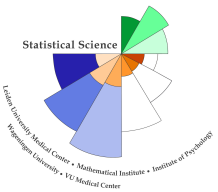
Mingxi Li (s1720554)

First Advisor: Dr. Wouter M.Koolen
Secondary Advisor: Dr. Tim van Erven

MASTER THESIS

Defended on October 12, 2017

Specialization: Statistical Science



STATISTICAL SCIENCE
FOR THE LIFE AND BEHAVIOURAL SCIENCES

mercylee2012@gmail.com

Abstract

The Multi-Armed Bandit(MAB) problem is named after slot machine games. When playing slot machines, one player has to decide which machine to play, in which order to play them and how many times to play each machine. After the choice, that specific machine will offer a random reward from a probability distribution, and the player's target is to maximize the sum of rewards earned through a sequence of lever pulls.

In order to figure out the distribution of each machine as soon as possible and to get as much profit as possible, we will consider the popular Thompson Sampling (TS) method, which is based on Bayesian ideas. TS is a heuristic for choosing actions that maximize the expected reward with respect to a randomly drawn belief.[9] In this thesis, for the first half part, we test the performance of TS and also compare a variation called Top-two Thompson Sampling(TTTS) method to the normal TS, based on uniform sampling. Computationally, TTTS is a slow algorithm, so we also try to improve its performance and create another algorithm: Top-two Gibbs Thompson sampling, which combines TTTS and Gibbs Sampling methods and improves the computation speed of the TTTS method.

In the second half of the thesis, we try to take a step forward in the application of TS, so we combine TS with the Maximin Action Identification(MAI) problem. Maximin is a concept from two-player zero-sum games in game theory. The main idea behind maximin action selection is a constant alternation of minimizing and maximizing the value of moves to account for an adversarial opponent in the game. Existing methods of maximin are related to games such as Checkers, Chess and Go. We try to broaden its application area and apply it to our new algorithms created in the first half part. First of all, the time budget is limited and we use different divisions to test the performance; Afterwards, we create a new algorithm to pick out the right arm with one step for two layers. The results show that there is not any significant difference between the time division method, which consists of even time budgets for he child layer and no budget for parent layer, and the Maximin Thompson Sampling method.

Contents

1 Introduction

Nowadays, people pay more attention to extract information from big data than before, since new technologies allow us to collect data way more conveniently. While new questions we are facing now have become: Which data is effective and which is just noise? How to learn from specific data? And what kind of information can we get from the chosen data set? This paper tries to use the combination of Thompson sampling and Multi-Armed Bandits problem to solve previous questions.

1.1 Example and intuitive theories

The name of "Multi-Armed Bandits"(MAB) comes from slot machine problem. A slot machine is a device in which the player pulls a lever arm and receives a random reward at some expected rate. By analogy, if there are multiple lever arms to pull and each one of them pays out at a different expected rate, we call it "multi-armed bandit".

The "multi-armed bandit" problem refers to the challenge of constructing a strategy for pulling the levers when one has no prior knowledge of the payout rate for any of the levers, so that one must find trade-off between exploring each of the levers in orders to determine the value and exploiting one's current knowledge in order to favor high paying levers. [11, 1] In this problem, a player needs to pick one arm during the given K arms at each play round and every arm is characterized by an unknown reward distribution and indexed by i. Since we use Bernoulli distribution as the inner distribution, so the probability of that player wins is μ_i. Normally, people want to find out the situation from the perspective of this player, then idenfify a maximum action, and this is the best choice for this player:

$$i^* = argmax_i \mu_i \tag{1}$$

Reward realizations are only observed when an arm is selected, and the player has (at least)one of two objectives: One is to maximize his cumulative expected earnings over a period T, another one, which also a method we will apply in this paper, is to minimize the misidentification probability over a period of T.

1.2 Maximin Action Identification (MAI)

According to the background setting of MAB problem, which is the situation of playing games, it is reasonable for us to lay out a scenario which contains two players and they are gambling relations to each other. Which means, if we are one of the players, the problem we are facing now is not to pick out the simple maximum or minimum arm but to choose the best arm which can win our opponent. Thus, a new bandit framework which contains two layers called "Maximin Action Identification" (MAI) will be introduced to solve this problem.

As required, MAI brings in another player, say player B, to compete with the original decision maker, say player A, to make a final decision. Player A has multiple choices to win the competition, while B has authority to choose less winning opportunity of A, in the end the largest winning probability choice of A will be the best decision for A. More precisely, suppose player A and player B join in a two-round zero-sum game together, and this game can be separated into n groups with each group indexed by i. In the very beginning, Player A offers k available actions to each group, each action can be indexed by j. Then player B chooses the minimum action of each group to reduce the winning probability of player A, and the last move is player A selecting the maximum action during these minimum arms. Setting the winning probability

of player A as $\mu_{i,j}$, and identifying a maximin action as below, so this "maximin arm" will be the best choice for player A:

$$i = argmax_i min_j \mu_{i,j} \tag{2}$$

Which $j \in \{1, 2, ..., K\}$, stands for action choices for player B; and $i \in \{1, 2, ..., N\}$, stands for action choices for player A. The parameters are unknown to player A, but A can keep choosing a pair $P = (i, j)$ of actions and observe a sample from a Bernoulli distribution with mean $\mu_{i,j}$.

To be more specific, here we bring up an example in the cyber security area. Suppose recently we built two network systems, say system 1 and 2, and accompanied with two security strategies, say strategy X and Y, to protect our systems. In consideration of the budget limitation, we can only afford one system and one security strategy, so that we need to figure out the best combination. To fulfill this target, we only consider two aspects to help us make the decision: One of them is the "security achieved", which means we need to calculate how many times the attacks had been detected and repelled, but here we fix the total attacking times and observe the repelling probabilities. Another aspect is the "incurred cost", which calculates the total cost of the systems running period.[6] Theoretically, the incurred cost may increase along with the defending time. And besides that, the more sophisticated the security strategy is the more money we need to spend. So we need to find out a trade-off point to keep our system and security strategy working as long as possible, which means we will not be hacked, nor run out of money.

In this game theory structure, we are player A, and our opponent will be the hacker(s), player B. The hacker(s) may attempt to gain unauthorized access to a networked system or render it incapacitated through denial of service[3], and the hacker(s) can adjust his/their behavior according to the defending actions, which means if our security strategy works too good, the opponent part may learn that fast and take action as soon as possible, and in this way, our best strategy could become the worst one.

For different systems, the security achievement is different between two strategies. And for the best performance system and strategy combination, the hacker(s) can easily detect and keep away from it or take action to upgrade his/their strategies, in the end, the best combination turns to be a useless one. So our mission is to figure out the ideal security strategy which is hard to detect and has relative better performance when applied to general systems. For instance, we use successful defense probability to quantify the combination quality, and as showed in figure 2, the four combinations have their own true successful defense probabilities:

$$Prob = \begin{bmatrix} & action1 & action2 \\ player1 & 0.9 & 0.5 \\ player2 & 0.2 & 0.4 \end{bmatrix}$$

Hacker(s) would like to choose the attacking target which has relatively lower success probabilities, which means for security strategy X, system 2(successful defense probability is 0.5) is chosen to attack, while for strategy B, systems 1 (successful defense probability is 0.2) is chose to attack. But our ultimate goal is that the security strategy's performance can be as good as possible, so we take samples of attacking times of each arm to find out the least attacked one, and with that arm, there is also the most efficient security strategy. In this example, we pick out strategy X, which has the maximum success probability between these two minimum values, to apply to our general network systems.

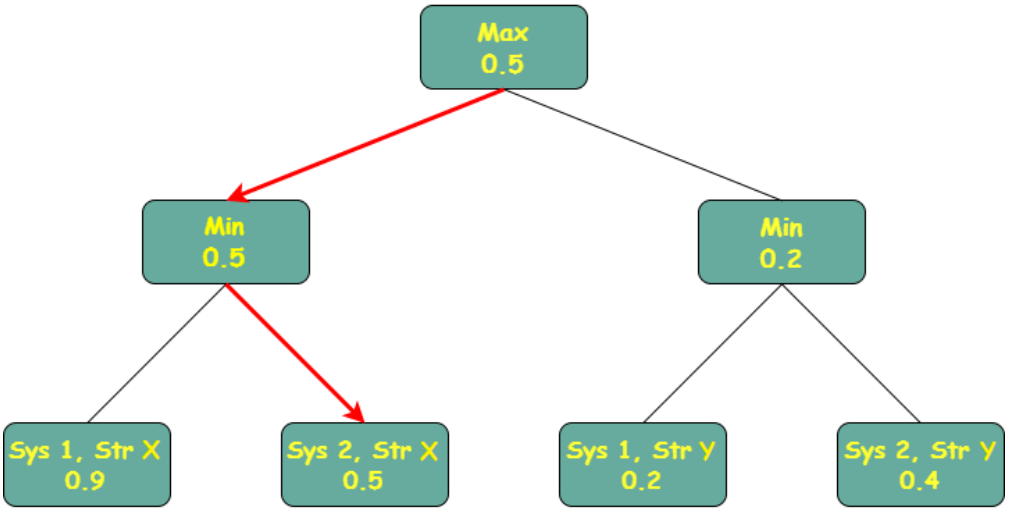


Figure 2: Maximin structure of cybersecurity system

Besides the cyber security area described above, Maximin Action Identification can also be applied to plenty of fields which associate with game theory within multi-parts, for example the medical area which related doctors and patients, financial area which related sellers and buyers, etc.

1.3 Thompson Sampling Apply to Multi-armed Bandit Problem

Thompson sampling is a popular and classical heuristic for choosing actions in the multi-armed bandit problem.[5, 2] It has formed the basis for accelerated learning in decentralized decision making, consisting in choosing the action that maximizes the expected reward with respect to a randomly drawn belief. Rather than consistently predicting the most likely outcome, experimental subjects tend to randomize their predictions based on the probabilities with which the respective events occur. Since each time the algorithm simply samples actions according to the previous optimal probability, it saves plenty of time for us to come into a final decision.[8]

In the previous bandit settings in section 1.1, during each playing time t we have a set of actions I_t. First choosing an action $i_t \in I_t$, then observing the reward $r_t \in [0, 1]$ of that action at time t. The goal is to find a method that can pick out the most reward arm and make the least mistakes. While Thompson sampling is broadly understood as based on a Bayesian philosophy of learning, which only uses Bayes rule but not the real Bayesian. To be specific, when considering the problem of learning a parametric distribution from observations. A frequentist's approach is using the data learn those fixed parameters as accurately as possible, while a Bayesian learner's approach is to maintain the probability distribution, which represents the chance that the parameter is of a certain value, to represent his uncertainty about the parameter.

The set of past observations O are modeled using a parametric likelihood function $P(r_t|i(t), \theta)$ depending on some unknown parameters θ. Given prior distribution $P(\theta)$ on these parameters, the posterior distribution of these parameters is given by the Bayes rule:

$$P(\theta|O) \propto \Pi_t P(r_t|i(t), \theta)P(\theta). \tag{3}$$

In the standard $K - armed$ Bernoulli bandit, every action corresponds to the choice of an arm and the reward of the $i - th$ arm follows a Bernoulli distribution with mean θ_i^*. And Beta

distribution is considered as a standard model of the mean reward of each arm because it is the conjugate distribution of the binomial distribution.

To measure how good TS algorithm works in multi-armed bandit problems, here we introduce a convenient method: measuring the expected total regret. Expected total regret is the amount of loss because of not playing optimal arm in each step. Let μ_i denote the unknown expected reward for arm i, $i(t)$ is the arm played in step t, and let $\mu^* = max_i \mu_i$. According to previous studies, which compared the Thompson sampling for the Bernoulli bandit problem with the Upper Confidence Bound(UCB) algorithm with expected total regret value

$$E(R(T)) = E[\sum_{t=1}^{T}(\mu^* - \mu_{i(t)})], \tag{4}$$

Thompson sampling is asymptotically optimal and achieves a smaller regret than the popular UCB algorithm. Intuitively, we can also adapt Thompson sampling method to the maximin action identification problem to improve its efficiency.

UCB algorithm is the most basic strategy which provides an approximate solution to the multi-armed bandit problem. For each round, we simply select the arm that has the highest empirical reward estimate up to that point plus some term that's inversely proportional to the number of times the arm has been played. To be for formally, define $n_{i,t}$ to be the number of times arm i has been played up to time t. Same as setting in previous 1.3, define $r_t \in [0, 1]$ to be the reward we observe at time t and $I_t \in \{1...N\}$ to be the choice of arm at time t. The empirical reward estimate of arm i at time t is:

$$\hat{\mu_{i,t}} = \frac{\sum_{s=1:I_s=i}^{t} r_s}{n_{i,t}} \tag{5}$$

And UCB assigns the following value to each arm i at each time t:

$$UCB := \mu_{i,t} + \sqrt{\frac{\ln t}{n_{i,t}}} \tag{6}$$

2 Methods

Our starting point is Thompson Sampling (TS). In this section, we review TS and its improvement Top-Two Thompson Sampling (TTTS) by Daniel Russo from *Simple Bayesian Algorithms for Best Arm Identification* .[10] While, after series of tests of TTTS algorithm, we find the calculating speed is getting slower when the play time is getting larger. To make up this defect, we created fake top-two Thompson Sampling (fake TTTS) and top-two Gibbs Thompson Sampling (TTGTS). In the end, we will compare above four algorithms and uniform sampling method, and the R code of five algorithms is open source now (https://github.com/muditalee/Thompson-Sampling-for-Monte-Carlo-Tree-Search-and-MAI.git)

TS samples actions according to the posterior probabilities which are the optimals. In particular, it selects action i with a specific probability. Following the previous multi-armed bandit setting, during each round t, we sample it from the posterior to get win rate θ_i for each action i. Pick out the largest win rate's arm $i = argmax_i(\theta_i)$ and observe the reward r, then updating the parameters $S_i(success)$ and $F_i(failure)$ of Beta distribution to play next round $t+1$. The Vanilla TS pseudo-code is shown in Algorithm 1.

Algorithm 1 Vanilla Thompson Sampling (with unknown true win rate $\mu_1, \mu_2, ..., \mu_k$)

1: Setting initial counts of all arms i as $S_i = 0, F_i = 0$,
2: **for** each play time $t = 1, 2, ...,$ **do**
3: **for** each arm $i = 1, 2, ..., k$ **do**
4: Sample $\theta_i \sim Beta(S_i + 1, F_i + 1)$. ▷ $+1$ comes from uniform prior on $\theta_i \in [0, 1]$
5: **end for**
6: Play arm $i = argmax_i\theta_i$ and observe reward $r \sim Binom(\mu_i)$.
7: **if** $r = 1$ **then**
8: $S_i = S_i + 1$,
9: **else**
10: $F_i = F_i + 1$.
11: **end if**
12: **end for**
13: Recommend arm: $i^* = argmax_i \frac{S_i}{S_i+F_i}$

However, Thompson sampling can have a very poor asymptotic performance for the best arm identification problem, because once it estimates that a particular arm is the best with reasonably high probability, it selects that arm in almost all periods at the expense of refining its knowledge of other arms. For example: If the predicted probability of the best arm $i = argmax_i\theta_i$ is 0.99, then the algorithm will select the arms other than i roughly once per 100 periods, which is greatly extending the sample size.

2.1 Top-two Thompson Sampling

To solve this slow learning problem, Daniel Russo brought up a new method combining Thompson sampling and Top-Two Probability sampling, which he called Top-Two Thompson sampling. In order to prevent the problem of exclusively focusing on one action, TTTS modifies standard TS by adding a resampling step and introduces a tuning parameter $\beta > 0(default\beta = 1/2)$. So that with probability β the first sampling design (step $3 \sim 9$ in Algorithm 2) is measured, with probability $1 - \beta$ the alternative sampling design (step $10 \sim 17$ in Algorithm 2) is measured.

To provide insight into how does Top-Two Thompson sampling work, Algorithm 2 shows how to directly sample an action according to TTTS by sampling from the posterior distribution.

Algorithm 2 Top-Two Thompson Sampling (with unknown true win rate $\mu_1, \mu_2, ..., \mu_k$)

1: Setting initial counts of all arms i as $S_i = 0, F_i = 0$,
2: **for** each play time $t = 1, 2, ...,$ **do**
3: **for** each arm $i = 1, 2, ..., k$ **do**
4: Thompson sampling: sample $\theta_i \sim Beta(S_i + 1, F_i + 1)$
5: **end for**
6: get $i = argmax_i \theta_i$
7: sample $B \sim Bernoulli(\beta = 0.5)$
8: **if** $B = 0$ **then**
9: Play i
10: **else**
11: **repeat**
12: **for** each arm $j = 1, 2, ..., n$ **do**
13: Thompson sampling: sample $\theta_j \sim Beta(S_j + 1, F_j + 1)$
14: **end for**
15: get $j = argmax_j \theta_j$
16: **until** $j \neq i$
17: play j
18: **end if**
19: Observe reward $r \sim Binom(\mu_{i/j})$ ▷ i/j: choose either arm i or arm j, depends on which one was played this round.
20: **if** $r = 1$ **then**
21: $S_i = S_i + 1$,
22: **else**
23: $F_i = F_i + 1$.
24: **end if**
25: **end for**
26: recommend: $i^* = argmax_i \frac{S_i}{S_i + F_i}$

Russo shows that TTTS has the correct behavior in theory, but in practice, there is still a problem. According to the description above, because the resampling process keeps rejecting the best arm, TTTS method can get stuck into repeat sampling loop and becomes extremely slow. Intuitively, we need to change the rejection loop inside the algorithm to some faster algorithms will be the solution.

2.2 Fake Top-Two Thompson Sampling

Firstly, there is an intuitive solution which keeps everything the same as TTTS except for the repeating part. We try to sort estimate win rates from the first TS and directly set the second largest win rate's arm as the sub-optimum arm.

Algorithm 3 Fake Top-Two Thompson Sampling (replace line 11 $\sim$ 17 of Algorithm 2)

1: Sort θ_j and get $j = argmax_j \theta_{j,-i}$
2: play j

Experimentally, learning speed should be tremendously increased, but the problem is also obvious: If the second largest arm is converged on one point of value while some other arms,

which have lower true win rates, are dispersed in a wide, then picking that second biggest arm gives no information so it is a waste for fake TTTS to play the second arm a lot.

For example: according to plot 3, the best arm's true win rate is 0.9 (black), the second largest arm's true win rate is 0.5 (red), the worst arm's true win rate is 0.3 (blue). Obviously that the black line and the blue line are all wide dispersed, while the red line, which is the second best arm , is converged to one point. Fake TTTS algorithm automatically chooses the top two best arms and the red line was picked a lot during the whole calculation, so we wastes lots of budget on ineffective information.

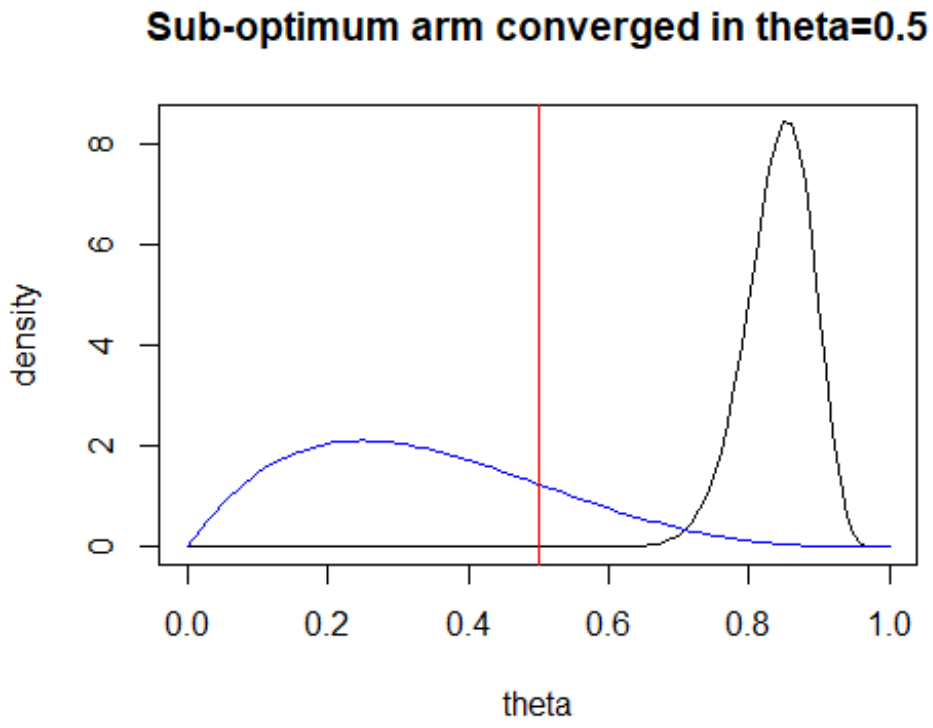


Figure 3: Under fake TTTS method, one extreme possible true win rate combination with $\mu_i = (0.3, 0.5, 0.9)$

In this way, even though fake TTTS is an intuitive method which could be faster than the original TTTS, the ineffective information choosing problem because of convergence nature stops our step to apply it in general calculating.

2.3 Top-two Thompson Sampling Combine Gibbs Sampling

We would rather sample from truncated distribution, which covering area does not surpass the higher probability arm area nor surpass the lower probability arm area, than keep rejecting the samples from "wrong" distribution area, so Gibbs sampling should be introduced here.

2.3.1 What is Gibbs sampling

Gibbs sampling is a Markov Chain Monte Carlo algorithm to obtain a sequence of approximated observations from a specified multivariate probability distribution, especially when direct sampling is difficult. When the joint distribution is not known explicitly or is difficult to sample from directly, but the conditional distribution of each variable is known and easy to sample from, Gibbs sampling can break the problem of sampling from the high-dimensional joint distribution into a series of samples from low-dimensional conditional distributions. [4]

Suppose we have a joint density $f(\theta_1, ..., \theta_n)$ and we are interested in sampling from it, but sampling directly from joint density is hard, so we turn to Gibbs sampling. Assume we can sample the n-many univariate conditional densities:

$$\begin{aligned} &f(\theta_1|\theta_2, ..., \theta_n) \\ &f(\theta_2|\theta_1, \theta_3, ..., \theta_n) \\ &\vdots \\ &f(\theta_n|\theta_1, ..., \theta_{n-1}). \end{aligned} \tag{7}$$

Arbitrarily choose n initial values: $\Theta_1 = \theta_1^0, \Theta_2 = \theta_2^0, ..., \Theta_n = \theta_n^0$, generate:

$$\begin{aligned} &\theta_1^1 \sim f(\Theta_1|\theta_2^0, ..., \theta_n^0); \\ &\theta_2^1 \sim f(\Theta_2|\theta_1^1, \theta_3^0, ..., \theta_n^0); \\ &\vdots \\ &\theta_n^1 \sim f(\Theta_n|\theta_1^1, ..., \theta_{n-1}^1). \end{aligned} \tag{8}$$

This constitutes one Gibbs pass through the n conditional distributions, and iterate the sampling to construct the second round: $(\theta_1^2, \theta_2^2, ..., \theta_n^2)$.

Repeating above process until required number of samples have been generated, then we discard all the precious samples, only the last round is left.

2.3.2 Gibbs Sampling Applied in TTTS

And the property of Gibbs sampling just meets the requirement of second-time sample of top-two Thompson sampling (lines $11 \sim 17$ in Algorithm 2). After the first round of Thompson sampling, the algorithm should sample again without the best arm that it picked in the first round. When it comes to the maximin identification problem with multi-arms and setting the estimated value of each arm as θ_j, we want to know the win rate of each arm which does not belong to any previous sample:

Similarly, according to Russo's rejection theory, suppose we have a joint density $f(\theta_1, ..., \theta_n)$ for second round sampling which excludes the picked arm in the first round:

$$\begin{aligned} &f(\theta_1, ..., \theta_n) \\ &= P(\theta_j|argmax_j\theta_j \neq i) \\ &= \frac{\Pi P(\theta_j; S_j, F_j) * 1(argmax_j\theta_j \neq i)}{\int_{\theta_j} \Pi P(\theta_j; S_j, F_j) * 1(argmax_j\theta_j \neq i) d_{\theta_j}} \end{aligned} \tag{9}$$

$*1(...)$ in equation 9 is an indicator function, if the condition inside function is satisfied, then $1(...) = 1$, otherwise $1(...) = 0$.

But experiment results showed that rejection sampling is not as efficient as Gibbs sampling method, which directly sampling from truncated Beta distribution instead of rejecting "wrong arm", so the precise sampling probability of one Gibbs sampling marginal step will be:

$$\begin{aligned} & P(\theta_j|\theta_{-j}, \theta_i < max(\theta_{-j})) \\ & = \frac{P(\theta_j) * P(\theta_{-j}) * 1(\theta_i < max(\theta_{-j}))}{\int_{\theta_j} P(\theta_j) * P(\theta_{-j}) * 1(\theta_i < max(\theta_{-j}))d_{\theta_j}} \\ & = \frac{P(\theta_j) * 1(\theta_i < max(\theta_{-j}))}{\int_{\theta_j} P(\theta_j) * 1(\theta_i < max(\theta_{-j}))d_{\theta_j}} \end{aligned} \tag{10}$$

* θ_{-j} in equation 10 stands for all the estimated θs except θ_j, which can be expressed as $\theta_{-j} = (\theta_1, ..., \theta_{j-1}, \theta_{j+1}, ..., \theta_k)$

Specifically, when we use Top-two Gibbs Thompson Sampling method, at the re-sampling round, for each iteration (k), calculating conditional win rate of each arm which does not belong to the first round sample set. Since in the resampling process, newly picked arm j could be the same arm as the first sampled arm i or be some other arm $(-i, -j)$:

If $j = i$, then the new sample can be any value between zero and the largest arm beside arm i and j:

$$\begin{aligned} & P(\theta_i|\theta_{-i}, \theta_i < max(\theta_{-i})) \\ & = \frac{P(\theta_i) * P(\theta_{-i}) * 1(\theta_i < max(\theta_{-i}))}{\int_{\theta_i} P(\theta_i) * P(\theta_{-i}) * 1(\theta_i < max(\theta_{-i}))d_{\theta_i}} \\ & = \frac{P(\theta_i) * 1(\theta_i < max(\theta_{-i}))}{\int_{\theta_i} P(\theta_i) * 1(\theta_i < max(\theta_{-i}))d_{\theta_i}} \end{aligned} \tag{11}$$

Thus, θ_i is sampled from truncated Beta distribution between area $[0, max(\theta_{-i})]$, as figure below shows. For the application of truncated sampling, we calculate the boundary value first $U \sim CDF(max(\theta_{-i}))$, then sampling by uniform distribution in range of $[0, U_i]$, the last step is finding out the corresponding θ values of each sample using $CDF^{-1}(U)$.

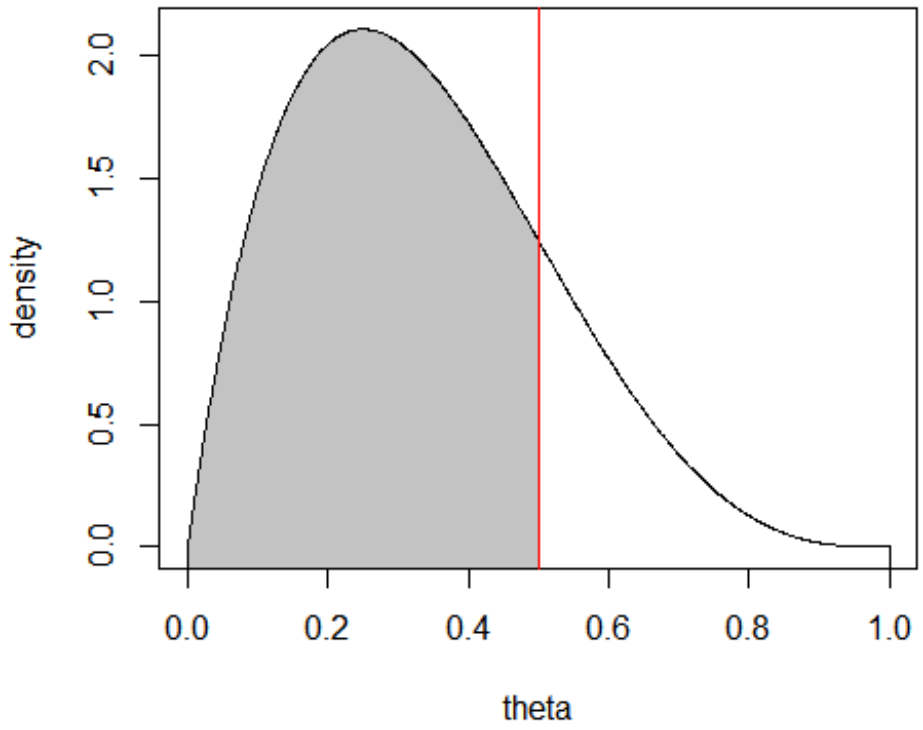


Figure 4: When $j = i$, sampling from Truncated Beta distribution during area $[0, max(\theta_{-i})]$.

If $j \neq i$, arm i either has to be smaller than the new sampled arm j or has to be smaller than rest of the arms $(-i, -j)$, this creates two situations:

Situation 1: If win rate of arm i (θ_i) is larger than the rest of arms $(\theta_{-i,-j})$, then the new sample value θ_j has to be larger than θ_i to make sure arm i is not the best choice in second round sampling process:

$$\begin{aligned} & P(\theta_j|\theta_{-j}, \theta_i < \theta_j) \\ &= \frac{P(\theta_j) * P(\theta_{-j}) * 1(\theta_i < \theta_j)}{\int_{\theta_j} P(\theta_j) * P(\theta_{-j}) * 1(\theta_i < \theta_j) d_{\theta_j}} \\ &= \frac{P(\theta_j) * 1(\theta_i < \theta_j)}{\int_{\theta_j} P(\theta_j) * 1(\theta_i < \theta_j)) d_{\theta_j}} \end{aligned} \tag{12}$$

Under this condition, θ_j is sampled from truncated Beta distribution between area $[\theta_i, 1]$.

Situation 2: If win rate of arm i (θ_i) is smaller than the rest of arms $(\theta_{-i,-j})$, which means the condition of arm i being not the largest one in the second round sampling has already been guaranteed, then the new sample θ_j can be any value on Beta distribution:

$$P(\theta_j) \tag{13}$$

Under this condition, θ_j is sampled from full Beta distribution, which is between area $[0, 1]$ as figure 5 shows.

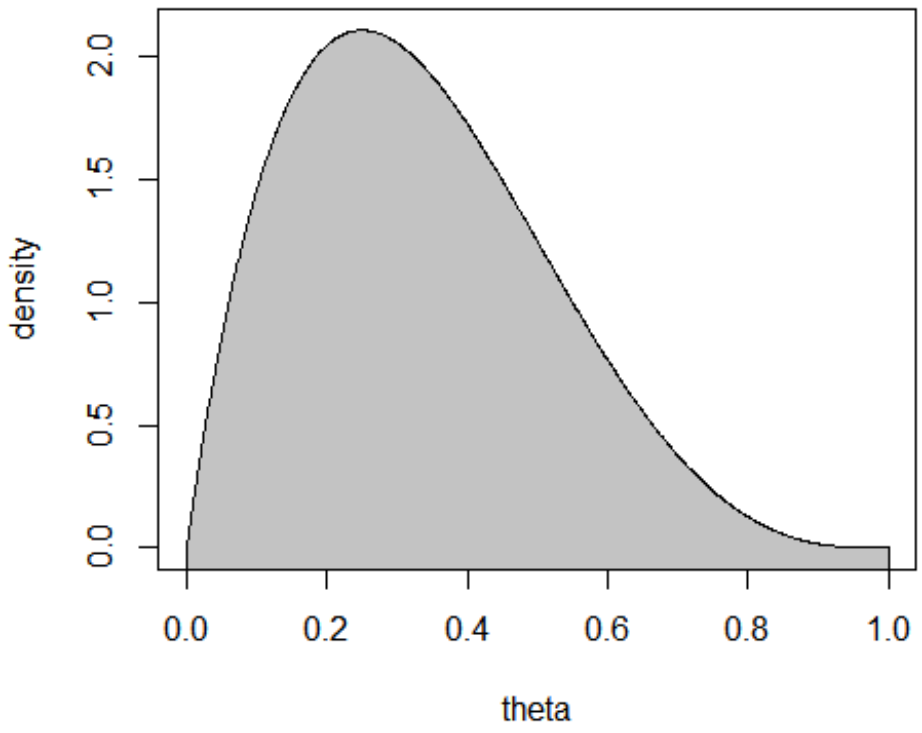


Figure 5: When $j \neq i$, sampling from the whole Beta distribution.

Intuitively showed in figure 6 below, for the truncated Beta distribution of θ_j (e.g. $\theta_j = 0.3$), the sample area is limited under density between $\theta_j \in [\theta_i, 1]$.

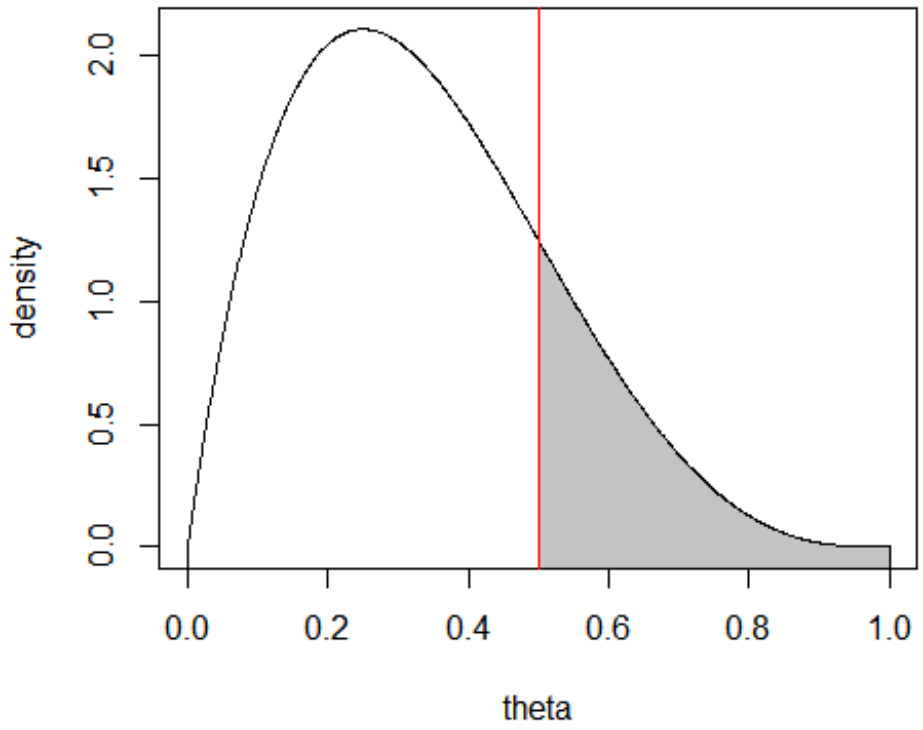


Figure 6: When $j \neq i$,sampling from Truncated Beta distribution during area $[\theta_i, 1]$.

In section2.3, we provided some insights about how to combine Top-two Thompson sampling and Gibbs sampling, and how to implement the new algorithm to important problem classes. At the very beginning, the algorithm is totally the same as Top-two Thompson sampling. The only difference is the second round Thompson sampling was replaced by Gibbs sampling. Before describing the algorithm in details, there should also be some clarifications about the Thompson Sampling algorithm: It initially assumes arm i to have prior Beta(1,1) on real probability μ_i, and at time t, having observed S_i successes (reward = 1) and F_i failures (reward = 0) in $(S_i + F_i)$ plays of arm i, then the algorithm updates the distribution on μ_i as Beta$(S_i + 1, F_i + 1)$. Thus the algorithm samples from these posterior distributions and plays an arm according to the probability of its mean being the largest.

2.4 Algorithms Comparison

According to the description above, we want to compare the performances of above four sampling methods and uniform sampling method (as the control group). Suppose we set r as repeat times of each algorithm. Each algorithm can pick out one arm per round, and if the picked arm is the real best arm, we set the counts of A_{right} plus one. Then the criteria would be the error

Algorithm 4 Top-Two Gibbs Thompson Sampling (Multi-armed Bandits)

Set initial value of all arms as $S_i = 0, F_i = 0$,
for each play time $t = 1, 2, ...,$ **do**
  for each arm $i = 1, 2, ..., k$ **do**
    Thompson Sampling: Sample $\Theta_i \sim Beta(S_i + 1, F_i + 1)$
  end for
  get $i = argmax_i \theta_i$
  Sample $B \sim Bernoulli(\beta = 0.5)$
  if $B = 0$ **then**
    set $\theta_i = (0.5, 0.5, 0.5, 0.5, 0.5)$
    for each iteration $k = 1, 2, ..., 25$ **do**
      for each arm $j = 1, 2, ..., k$ **do**
        if $j \neq i$ **then**
          if $\theta_i > max(\theta_{-i,-j})$ **then**
            $\theta_j \sim CDF^{-1}(U_j; S_j, F_j)$,
            which $U_j \sim unif(CDF(\theta_i; S_j, F_j), 1)$
          else
            $\theta_j \sim Beta(S_j, F_j)$
          end if
        else
          $\theta_i \sim CDF^{-1}(U_i; S_i, F_i)$,
          which $U_i \sim unif(0, CDF(max(\theta_{-i}); S_i, F_i))$
        end if
      end for
    end for
    get $j = argmax_j \theta_j$
  end if
  Play i or j and observe reward $r \sim Binom(\mu_{i/j})$:
  if $r = 1$ **then**
    $S_{i/j} = S_{i/j} + 1$
  else
    $F_{i/j} = F_{i/j} + 1$
  end if
end for
Recommend arm: $i^* = argmax_i \frac{S_i}{S_i + F_i}$

probability of each sampling methods $P(\epsilon)_m$:

$$P(\epsilon)_m = 1 - \frac{A_{right}}{r} \tag{14}$$

which means we calculated the times each algorithm chooses the wrong arm by repeating each algorithm for r times. For example, we simply set repeat time as $r = 100$, m is index of each algorithm, and the truth win rate of each arm is $\mu_i = (0.45, 0.45, 0.45, 0.45, 0.5)$, which means the fifth arm is the best. This probability combination is arbitrary, the purpose is to simulate one extreme situation which the true win rate of five arms are highly close to each other, meaning algorithms are easier to make mistake and going to consume a long time.

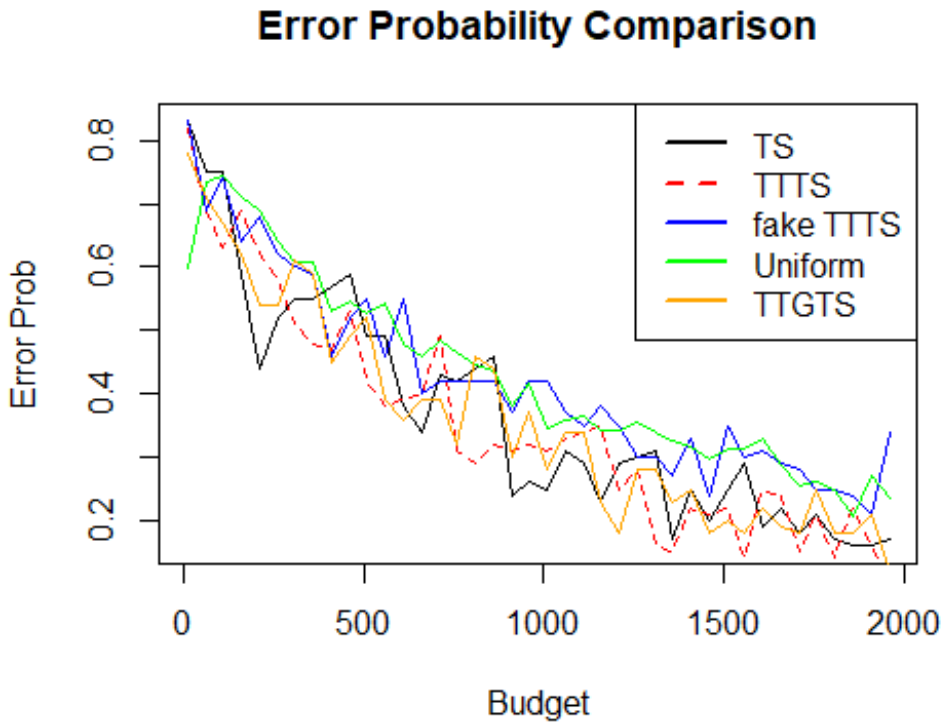


Figure 7: Misidentification probability plots of five sampling methods under condition of $\mu_i = (0.45, 0.45, 0.45, 0.45, 0.5)$.

According to Figure 7, the uniform sampling algorithm is the worst performance algorithm, as the green line shows, and Uniform Sampling(US) group is considered as the control group. Compare to other algorithms which concentrate on the best one or the best two arms, uniform sampling algorithm divides its playing time equally by the arm number, which is the lowest efficient method within these five methods.

Besides its mis-sampling sub-optimum problem, fake TTTS method has similar performance as US method or performs slightly better than US method as the blue line shows. So here, we reject the newly created algorithm: fake top-two Thompson sampling. But still, since it is a very intuitive way to improve the TTTS method's calculation speed and the author of TTTS did not

mention why fake TTTS does not work well in his previous works. We think it is necessary to point out shortcomings of fake TTTS method, for the sake of later researching.

TS, TTTS, and TTGTS work slightly better than the other two, as the black, red dash and orange lines show. And among these three better choices, TS is the most unstable sampling method, TTTS and TTGTS have relative similar better performance. Lines of TTTS and TTGTS are similar because they both have two rounds of sampling to pick out two arms and only the second round sampling methods have some difference, all the other steps are all the same. But we were not satisfied with the noisy result and tried another true win rate combination $\mu_i = (0.1, 0.3, 0.5, 0.7, 0.9)$ to try to find some pattern from a relative smoother result. This probability combination is also arbitrary, the purpose is to simulate an extreme ideal situation: the true win rate of five arms keep the same interval to each other, which is relatively easier to find pick out the right arm and save a lot of time.

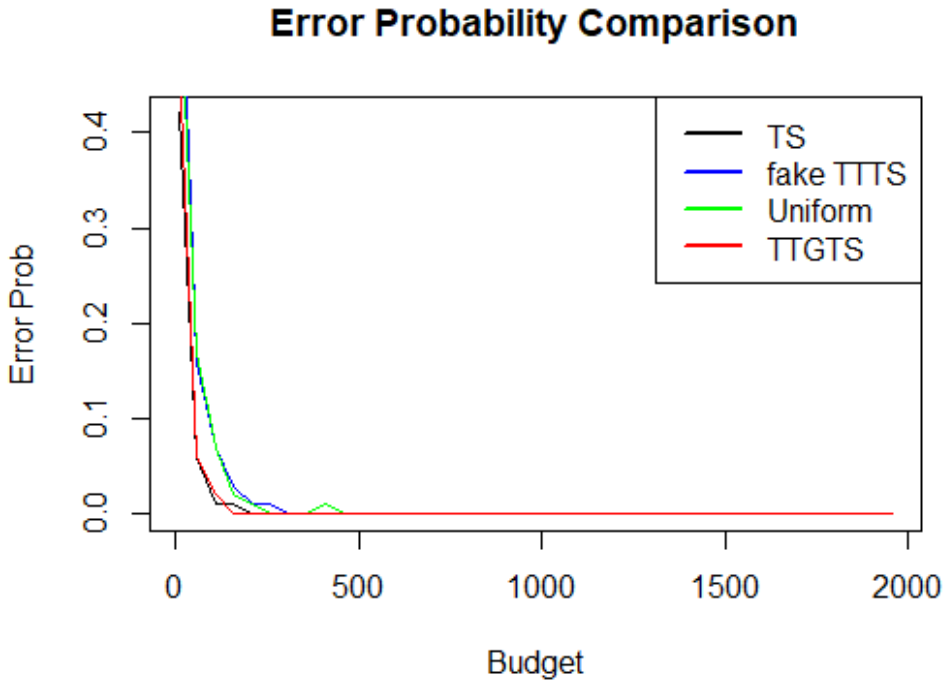


Figure 8: Misidentification probability plots of four sampling methods under condition of $\mu_i = (0.1, 0.3, 0.5, 0.7, 0.9)$.

As figure 8 shows, there are only four misidentification probability lines standing for four methods without TTTS method. Because for the relativly sparse true win rates combination, TTTS method performs extremely slow comparing to other four methods. The black and red lines proved our previous conclusion that TS and TTGTS algorithms are better than fake TTTS and US algorithms, and this kind of advantage is more obvious when the true win rates are more and more sparse to each other.

Literally, Gibbs sampling generates posterior samples by sweeping through each variable (or block of variables) to sample from its conditional distribution with the remaining variables fixed

to their current values, while second round Thompson sampling just repeat the whole sampling process, so Gibbs sampling should be faster than Thompson sampling.

Experimentally, we tried two groups of true win rates combination: one group of relatively dispersed values $\mu_i = (0.1, 0.3, 0.5, 0.7, 0.9)$, another group of very similar values $\mu_i = (0.45, 0.45, 0.45, 0.45, 0.5)$. For both two groups, the larger the playing time is the more advantages Gibbs sampling method shows, especially when the true win rates for all arms are similar. For example: according to one-time experiment, under the condition of dispersed true win rate combination, TTGTS performs better than TTTS when playing time is larger than 70; and under the condition of similar true win rate combination, TTGTS performs better than TTTS when playing time is larger than 4000.

Both two methods aim at picking out suboptimal arm beside the best one, so when the true win rates are similar to each other, the algorithms will take a longer time to find the best two arms out, so these two methods run faster in a dispersed group. And rejection sampling is easier to be stuck into picking the same best arm out because of the character of Thompson sampling method we described above, so it wastes more time than Gibbs sampling, so the larger playing time is, the better the TTGTS algorithm performs. Specifically, since Gibbs sampling estimates true win rates by iteration and here we set the iteration time as 25 (arbitrarily, 4), so ideally the time consuming of Gibbs sampling is 25 times of TS.

3 Thompson Sampling Combine Maximin Bandits Problems

3.1 Time Division Problem

Section2.2 shows that fake TTTS has the ineffective information choosing problem, and uniform sampling has the worst performance during these methods and TTTS consumes plenty of time to get the result, so for the later experiments we only consider TS and TTGTS methods.

Far from now, both of the methods can be used for picking out the best arm when all of the arm's true win rates are unknown, so applying them to two-layer tree structure will be an interesting topic. Back to the introduction part, we have built the four-arm two-branche tree structure, which might be the simplest and the most basic tree structure for multi-armed bandits problems. While this stereotype has some problem for algorithms performance testing, since two-by-two structure can be considered as Bernoulli problem and unrepresentative. Garivier, Kaufmann, and Koolen[7] came up with a benchmark tree search structure, which is more general, so in the later experiments we are going to apply all the methods to this $3*3$ two layers structure:

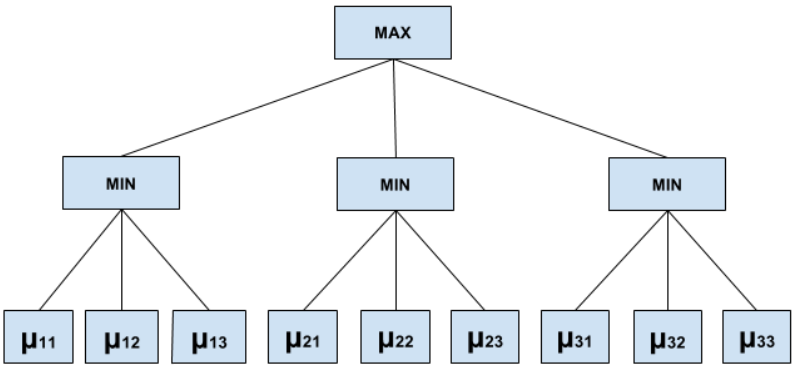


Figure 9: $3 * 3$ two layers maximin-bandit structure

And each of μ_{ij} has its specific true mean value to try to cover the "worst" situation (it is hard to cover, but not impossible), which may contain μ_{ij}'s combination as ambiguous as possible both within a group and between groups. Since j stands for arm number within a group and i stands for the group number, true win rates combination can be shown as below:

μ_{ij}	$j=1$	$j=2$	$j=3$
$i=1$	0.50	0.45	0.55
$i=2$	0.60	0.40	0.35
$i=3$	0.30	0.47	0.52

Table 1: Benchmark $3*3$ true win rates combination

Using above multi-bandit structure with its specific μ_{ij} values, we can compare the performance of TS method and TTGTS method now. But besides comparing these two methods to explore their performance in different structures, the most important task in this part is figuring out an appropriate time division method between two layers under the condition that the total time is limited. The R code of TS applied time division method and TTGTS applied time division method is available (https://github.com/muditalee/Thompson-Sampling-for-Monte-Carlo-Tree-Search-and-MAI.git).

Intuitively, we need two steps to pick out the final max-min arm: Running the sampling algorithm for each group in child layer to pick out the minimum true win rate arms, then running the sampling algorithm again for the new picked arms group to get the final maximum true win rate arm. Also since we calculate the true win rate according to the success and failure times of each arm, the final success and failure values of each minimum arm will be directly applied to sampling algorithm of parent layer to improve the calculation efficiency. In this way, budget allocation in parent layer is not as important as the child layer, because ideally, if the calculation of child layer is correct, posterior S and F can be directly applied to parent layer without one single extra sample (which is also the budget combination of 0 budget for the second layer, and even budget for each group in the first layer). Here use TS sampling method as an example:

Algorithm 5 $3*3$ Two Layers Time Division Algorithm (with unknown true win rates $\mu_{11}, \mu_{12}, \mu_{13}, \mu_{21}, \mu_{22}, \mu_{23}, \mu_{31}, \mu_{32}, \mu_{33}$)

1: Setting initial counts of all arms j in all groups i as $S_{ij}=0, F_{ij}=0$,
2: **for** each group $i=1,2,3$ of child layer **do**
3: Run TS
4: Pick out arm: $j = argmin_j \theta_{ij}$
5: Keep the posterior success and failure values of each selected arm: S_i, F_i
6: **end for**
7: Run TS again on parent layer: $(\mu_{1j}, \mu_{2j}, \mu_{3j})$ with posterior S_i, F_i, get $i = argmax_i \mu_{ij}$.

In real life, we prefer to find out the best solution with the least time-consuming. So in most cases, the calculating ability of computers is limited, and then we have to limit the total budget for the whole $3*3$ two layers structure too.

Here setting the highest total budget as 600 (which also is the total play time), and only considering to separate total budget between these two layers. Which means time budget changes between child layer and parent layer, but all of the groups in child layer have the equal time budget. According to line 5 of Algorithm 5, the success (S_i) and failure (F_i) values of child layer are directly applied to parent layer, so it is reasonable that we assume the ideal time division situation would be all of the child groups take over the 600 budget evenly and there is no budget for parent layer. Thus, we could directly get the target group within one step in two layers.

Before further experiments, we need to discuss some extreme combinations when parent layer takes over majority budget and only few budget left for child layer. One extreme combination of different true win rates could be there is no difference within each group j, but these three

groups are different from each other. In this way, there is no budget shared in child layer, but 600 budget shared in parent layer, this situation's model can be:

μ_{ij}	$j=1$	$j=2$	$j=3$
$i=1$	a	a	a
$i=2$	b	b	b
$i=3$	c	c	c

Table 2: Combination 1: 0 budget in child layer, 600 budget in parent layer.

Another situation might be: One arm in each group is obviously lower than other two arms (for example true win rate of other two arms are all 1) and minimum arms between those groups are less extremely different from each other, which means we only need few budget within groups to pick out the minimum arm and we need majority of the $t00$ budget to apply to the parent layer.

μ_{ij}	$j=1$	$j=2$	$j=3$
$i=1$	a	1	1
$i=2$	b	1	1
$i=3$	c	1	1

Table 3: Combination 2: There is always one arm obviously lower than others within each group, so there needs a few sample in children layer.

Another extreme true win rate combination is the difference within a group is hard to figure out and once we pick out a wrong arm from child layer the final result will be changed. Therefore, we should put majority of the budget to child layer and, ideally, no budget to parent layer. Since this situation is hard to generalize to one model, so we only describe it in words.

Between the range of these two extreme situations, we picked out all combinations of parent-child budget with a length of 10 to apply to the benchmark $3*3$ tree structure(Table 1). According to the experiment results and the properties of time division algorithm, the more budget child layer shared, the lower error probability we can get. So setting X-lab as the budget number for each single child group, which is from 0 to 200 (three leaves in total) and 10 budget difference per step, then comparing error probabilities of TS and TTTS methods under different budget combinations:

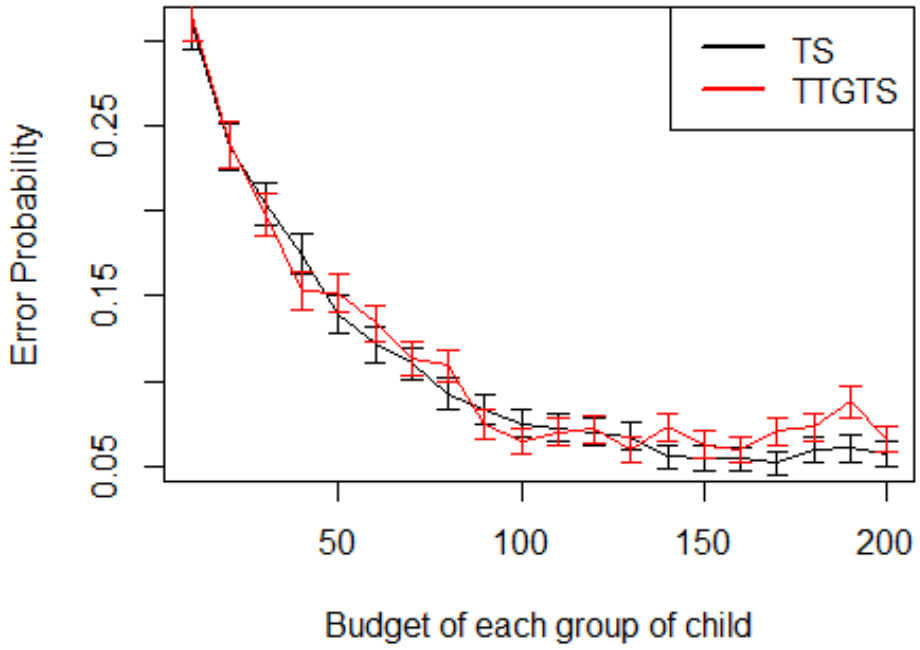


Figure 10: Error probabilities comparison of vanilla Thompson Sampling and Top-two Gibbs Thompson Sampling under different time divisions (600 budget in total, at most 200 budget for each child)

According to the knowledge above and Figure 10, the error probabilities of both algorithms decrease along with the increasing budget for each group in child layer, and stable around 5% $\sim$ 10% after 100 budget for each group. According to previous experience, we play 600 times in total and repeat the experiment 1000 times, so that each group in the first layer can share at most 200 budget in each experiment. TS and TTGTS have similar performance in general, their error probabilities drop down fast when the budget increases from 0 to 100, and after $t = 100$ their values are converged. Considering when $t = 100$, error probabilities are already lower than 10%, if there is not any strict error rate requirement, for standard true win rates combination, 100 budget per group and 300 for the second layer is a good choice. While in this experiment, for TS method, lowest probability 5.21% (one standard deviation 0.75%) appears at combination of 170 budget for each group in child layer and 90 budget for parent layer. For TTGTS method, lowest probability 6% (one standard deviation 0.75%) appears at combination of 130 budget for each group in child layer and 210 budget for parent layer, also combination of 160 budget for each group in child layer and 120 budget for parent layer has the same performance in this case.

For the extreme situation where no budget is available for parent layer and 200 budget for each child group in child layer, the error probability of TS method is 5.71% (one standard deviation 0.73%) and of TTGTS method is 6.6% (one standard deviation 0.79%). Since both extreme error probabilities are not out of range of best situations, so we can also apply the extreme

combination to real life under the specific μ's combinations we discussed before.

On the other hand, also according to figure 10, both of red and black lines have the increasing trend at the budget 180 and 190. Especially for TTGTS method, the error probability at 190 increases 46.67% compare to the best combination. One speculation will be, the budget of 150 ± 20 for each group in child layer and parent layer is the ideal trade-off combination for our classical tree structure. No matter for child layer or for parent layer, budget lowering than 70 can increase the error probability, slightly or dramatically. Comparing the performance of TS and TTGTS algorithm, TS is more stable than TTGTS. And when the parent layer's budget is closing to zero, TTGTS has relatively higher error probabilities.

Here we come up with questions which may deserve further studies of researchers: First of all, according to the plot, TS method has more stable behavior than TTGTS method. Ideally, based on the algorithms of these two methods, TS and TTGTS should have highly similar error probability plots. And even though deviation can exist, it should not be very significant (out of one standard deviation). So obviously, the performance of TTGTS in figure 10 is out of expectation and need deeper exploration. The second problem is the strange behavior of both lines at the budget 180 and 190. Since parent layer directly applies posterior S and F from the child layer, theoretically, no matter how many budget parent layer can get, the error probability should be lower when the child budget goes higher. But in figure 10, both lines go up at point $180 and 190$ and go down again at 200, which may need more general experiments besides 600 budget in total and $3 * 3$ stereotype μ's combinations.

3.2 Proposal for Sampling Rule

In previous sections, we picked out the optimal child group to do further sampling. There are two sampling steps based on a two layers tree structure: Picking out the minimum true win rate arm within each group, comparing those minimum arms and finding out the final arm with maximum true win rate. Once figured out which group the max-min arm belongs, we will focus on this group and keep sampling from it so that the calculation efficiency will be improved in this way.

Appropriate time division combines with sampling method mentioned above is a highly-efficient solution. But still, considering maximin tree structure consists of two layers, while sampling methods are all focus on one step sampling, there should be a new method which can pick out the target group from two layers maximin structure in one step, instead of separately sampling from two layers. According to experiences from above section (section 2.4), we will discard slower sampling method TTGTS and only focus on TS. So we created another new algorithm which can combine maximin tree structure and TS sampling method, and use one step to get the final result: Maximin Thompson Sampling.(6)

Again, we use the "classical" stereotype of win rates combinations, set each true win rate as μ_{ij}, in which $i(i = 1, ..., n)$ stands for the group number and $j(j = 1, ..., k)$ stands for the arm number within the group. Considering, in the end, we want to figure out which group (i) need to be selected to take a sample, so the function can be built as below:

$$i = argmax_i min_j \mu_{i,j} \tag{15}$$

Similar to previous algorithm of vanilla Thompson Sampling advanced Maximin Thompson Sampling algorithm can be created as below:

Algorithm 6 One Step Two Layers Thompson Sampling (with unknown true win rates $\mu_{ij} = \mu_{11}, ..., \mu_{1k}, \mu_{21}, ..., \mu_{2k}, ..., \mu_{n1}, ..., \mu_{nk}$)

Setting initial counts of all arms j in all groups i as $S_{ij} = 0, F_{ij} = 0$,
for each play time $t = 1, 2, ...,$ **do**
  for each group $i = 1, ..., n$ **do**
    for each arm $j = 1, 2, ..., k$ **do**
      Sample $\theta_{ij} \sim Beta(S_{ij} + 1, F_{ij} + 1)$.
    end for
  end for
  Get $i = argmin_j max_i \theta_{ij}$,
  Play arm in group $i = argmax_i \theta_{ij}$ and observe reward $r \sim Binom(\mu_{ij})$.
  if $r = 1$ **then**
    $S_{ij} = S_{ij} + 1$,
  else
    $F_{ij} = F_{ij} + 1$.
  end if
end for
Recommend group: $i^* = argmax_i \frac{S_{ij}}{S_{ij}+F_{ij}}$

There are many methods to test the performance of MTS algorithm, but for the sake of efficiency, here we only compare error probabilities of MTS and time division structure with TS. To make sure that the comparison is impartial and not rigorous, we set the play time t of MTS method as 600, which is equal to the total play time of time division method, and set the repeat experiment time as 1000. In R, we set seed as 1234, the error rate of MTS algorithm is 0.072, for time division method, it is 0.064. The original R code of TS apply to two-layer one-step method is available on line (https://github.com/muditalee/Thompson-Sampling-for-Monte-Carlo-Tree-Search-and-MAI.git).

To figure out if there is a difference between these two methods, we introduce Hoeffding's inequality here. Hoeffding's inequality provides an upper bound on the probability that the sum of independent random variables deviates from its expected value. Generally speaking, it can be applied to analyze the number of required samples (here are play times) needed to obtain a confidence interval by solving the inequality in:

$$P[|\bar{x} - E(\bar{x})| \geq \epsilon] \leq 2exp(-2n\epsilon^2) \tag{16}$$

In which $\bar{x}$ stands for the empirical mean of all probabilities, ϵ stands for deviation values from means and n stands for play times. Since the confidence interval is 95%, the probability upper bound will also be 0.95.

$$0.95 = 2exp(-2 \cdot 1000 \cdot \epsilon^2) \tag{17}$$

According to function above, we can get the standard error $\epsilon = 0.025$ for sample size 1000. Comparing to the difference of error probabilities from MTS and time division methods, which is $0.008 (< \epsilon)$, we can be convinced that the difference between MTS and time division methods are too small to point out a better one. To prove the conclusion, we also try another experiment which setting $t = 6000$ and $rep = 100$, in this way, the total play time (600000) does not change. Considering the property of TS algorithm, increasing play time can decrease the error probability, and for both methods, the error probabilities, in the end, turn to 0. Comparing with the deviation of Hoeffding inequality, which is 0.005, MTS and time division methods do

not have a significant difference from each other. So according to previous experiments, both of Maximin Thompson Sampling and time division with Thompson Sampling methods can be applied to calculate the true win rates of Maximin tree structure.

For further experiments, when the budget is limited and low, one step two layers Thompson Sampling method is a good choice; when the budget is unlimited, time division structure Thompson Sampling method may be better. Also, according to the real experiment result of these two methods in R, time division algorithm performs faster than one step two layers algorithm. Based on all of the advantages and it's stable character of time division Thompson Sampling method, time division structure Thompson Sampling will be more applied to selecting bandit problems, and there should be further development of one step two layers TS algorithm.

4 Conclusion

This thesis mainly described Thompson sampling methods, Top-two Thompson Sampling, and their "advanced" versions: fake Top-two Thompson Sampling and Top-two Gibbs Thompson Sampling. Besides that, we also tried to combine Thompson Sampling series algorithms with maximin action identification problems, which can help to solve two-layer Monte Carlo Tree Search problems. Similarly, we brought up two methods of combination, one is time division method, another is one step two layer maximin Thompson Sampling method.

For the first part, we briefly analyzed advantages and disadvantages of Thompson Sampling and found that Thompson sampling can have very poor asymptotic performance for the best arm identification problem, because once it estimates that a particular arm is the best with reasonably high probability, it selects that arm in almost all periods at the expense of refining its knowledge of other arms. So we tried upgrades version of Thompson Sampling from Russo: Top-two Thompson Sampling. Even though TTTS can overcome the fatal shortcoming of TS, it runs extremely slow as the playing time gest larger and larger. Then we created two new algorithms to overcome the problem of TTTS, one is fake Top-two Thompson Sampling, which tries to sort estimate win rates from the first TS and directly set the second largest win rate's arm as the sub-optimum arm. Experimentally, learning the speed of fake TTTS should be tremendously increased, but once the second largest arm is converged on one point of value, while some other arm is dispersed in a wide range which may surpass that value, the fake TTTS algorithm will automatically choose that dispersed distributed arm instead of the sub-optimum one. Then we introduced Gibbs sampling to combine it with TTTS, and we got Top-two Gibbs Thompson Sampling. Basically, the algorithm is totally the same as TTTS, the only difference is the second round Thompson sampling was replaced by Gibbs sampling.

Then we set two different groups of true win rates to compare above four algorithms, we found that: For very closed combination, TS, TTTS, and TTGTS work slightly better than the other two, and among these three better choices, TS is the most unstable sampling method, TTTS and TTGTS have relative similar better performance; For sparsely separated combination, TS and TTGTS algorithms are better than others. Notably, we did not run TTTS method under this sparsely separated true win rate combination, because it performs extremely slow, which means when the playing time is really large, it can not be applied to real life.

For the second part, we introduced Maximin Action Identification problem and tried to combine TS and its upgrades algorithms with MAI problem. We created two methods, one is fixing the total budget and figuring out an appropriate time division method between two layers. Under specific true win rate combinations and arbitrarily setting total time budget as 600, for TS and TTGTS methods, the higher the child budget the lower the error probabilities. Comparing the performance of TS and TTGTS algorithm, TS is more stable than TTGTS. And when the parent layer's budget is closing to zero, TTGTS has relatively higher error probabilities.

In the last part, we were not satisfied with two layers two steps method, so we created a new method: Maximin Thompson Sampling, which can pick out the target group from two layers maximin structure in one step, instead of separately sampling from two layers. Literally, the core algorithm of MTS is vanilla TS, considering the calculating speed. We compared the performance of time division method and MTS method, figured out that MTS and time division methods do not have a significant difference from each other so that both methods can be applied to calculate the true win rates of Maximin tree structure.

References

[1] Shipra Agrawal and Navin Goyal. "Analysis of Thompson Sampling for the Multi-armed Bandit Problem". In: *Proceedings of the 25th Annual Conference on Learning Theory*. Ed. by Shie Mannor, Nathan Srebro, and Robert C. Williamson. Vol. 23. Proceedings of Machine Learning Research. Edinburgh, Scotland: PMLR, June 2012, pp. 39.1–39.26. URL: http://proceedings.mlr.press/v23/agrawal12.html.

[2] Shipra Agrawal and Navin Goyal. "Further Optimal Regret Bounds for Thompson Sampling". In: *CoRR* abs/1209.3353 (2012). URL: http://arxiv.org/abs/1209.3353.

[3] Tansu Alpcan. "An intrusion detection game with limited observations". In: *Proceedings of the 12th Int. Symp. on Dynamic Games and Applications, Sophia Antipolis*. 2006.

[4] George Casella and Edward I. George. "Explaining the Gibbs Sampler". In: *The American Statistician* 46.3 (1992), pp. 167–174. ISSN: 00031305. URL: http://www.jstor.org/stable/2685208.

[5] Olivier Chapelle and Lihong Li. "An Empirical Evaluation of Thompson Sampling". In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., 2011, pp. 2249–2257. URL: http://papers.nips.cc/paper/4321-an-empirical-evaluation-of-thompson-sampling.pdf.

[6] Cuong T. Do et al. "Game Theory for Cyber Security and Privacy". In: *ACM Comput. Surv.* 50.2 (May 2017), 30:1–30:37. ISSN: 0360-0300. DOI: 10.1145/3057268. URL: http://doi.acm.org/10.1145/3057268.

[7] A. Garivier, E. Kaufmann, and W. Koolen. "Maximin Action Identification: A New Bandit Framework for Games". In: *ArXiv e-prints* (Feb. 2016). arXiv: 1602.04676 [math.ST].

[8] Pedro A. Ortega and Daniel A. Braun. "Generalized Thompson Sampling for Sequential Decision-Making and Causal Inference". In: *CoRR* abs/1303.4431 (2013). URL: http://arxiv.org/abs/1303.4431.

[9] CTI Reviews. *Machine Learning, A Probabilistic Perspective: Computer science, Artificial intelligence*. Cram101, 2016. ISBN: 9781490269269. URL: https://books.google.nl/books?id=fj0hAgAAQBAJ.

[10] Daniel Russo. "Simple Bayesian Algorithms for Best Arm Identification". In: *CoRR* abs/1602.08448 (2016). URL: http://arxiv.org/abs/1602.08448.

[11] Steven L. Scott. "A modern Bayesian look at the multi-armed bandit". In: *Applied Stochastic Models in Business and Industry* 26.6 (2010), pp. 639–658. ISSN: 1526-4025. DOI: 10.1002/asmb.874. URL: http://dx.doi.org/10.1002/asmb.874.