ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ & ΥΠΗΡΕΣΙΕΣ

ΜΕΓΑΛΑ ΔΕΔΟΜΕΝΑ ΚΑΙ ΑΝΑΛΥΤΙΚΗ

RANKING RECOMMENDATIONS VIA ONLINE LEARNING

(ΔΙΑΤΕΤΑΓΜΕΝΕΣ ΣΥΣΤΑΣΕΙΣ ΜΕΣΩ ΑΜΕΣΗΣ ΜΑΘΗΣΗΣ)

ΕΥΣΤΑΘΙΟΣ ΣΟΥΦΛΑΣ (ΜΕ 1944)

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ

ΟΡΕΣΤΗΣ ΤΕΛΕΛΗΣ

ΦΕΒΡΟΥΑΡΙΟΣ 2021

TABLE OF CONTENTS

INDEX OF FIGURES

ACKNOWLEDGEMENT

I would like to warmly thank my supervising professor, Mr Orestis Telelis, for his valuable guidance, patience, and personal support in my academic endeavors.

I would also like to express my heartfelt gratitude to my supportive wife, Maria, for all her encouragement during this challenging period of my life. She stood by me all the time and filled me with confidence to reach my goal. Last but not least, I would like to thank my miracle, my little princess, Katerina, who was born in the middle of this demanding period and filled me with enthusiasm. I would like to dedicate this dissertation to both of them.

Efstathios Souflas

ABSTRACT

These days, that can be characterized by high rates of information transmission and the increasingly usage of world wide web, there are plenty of applications and websites that handle recommendations systems, in order to provide their users with the best possible suggestions. The solution to such systems is given by Online Learning, which demand low computational resources and yield optimum results. In this work, we examine Online Learning and especially the Multi-armed Bandit problem, which is one of its aspects. In later sections, we will present some of the most known online learning algorithms that refer to that problem.

Specifically, we are interested in rankings of recommendations and the usage of the meta-algorithm Ranked Bandits Algorithm (RBA) in our experiments, which manipulates instances of other online algorithms. We study the convergence of RBA through “sponsored recommendations”, in which each document is related to an income, while each user is a subset of the documents he is interested in and derives randomly from a set of different populations.

ΠΕΡΙΛΗΨΗ

Στη σημερινή εποχή, που χαρακτηρίζεται από υψηλούς ρυθμούς μετάδοσης της πληροφορίας και της ολοένα αυξανόμενης χρήσης του παγκόσμιου ιστού, υπάρχουν πολλές εφαρμογές και ιστοσελίδες που χειρίζονται συστήματα συστάσεων, προκειμένου να παρέχουν στους χρήστες τους τις καλύτερες δυνατές προτάσεις. Η λύση για τέτοιου είδους συστήματα δίνεται από την «Άμεση Μάθηση», η οποία απαιτεί χαμηλούς υπολογιστικούς πόρους και προσφέρει βέλτιστα αποτελέσματα. Στην παρούσα διπλωματική εργασία, εξετάζουμε την «Άμεση Μάθηση» και ειδικά το πρόβλημα των «Πολλαπλών Κουλοχέρηδων», που αποτελεί μία από τις διαστάσεις της. Σε επόμενα τμήματα, θα παρουσιάσουμε ορισμένους από τους πιο γνωστούς αλγόριθμους άμεσης μάθησης, που αναφέρονται στο προαναφερθέν πρόβλημα.

Συγκεκριμένα, μας ενδιαφέρουν οι «διατεταγμένες συστάσεις» και η χρήση του μετα-αλγορίθμου «Αλγόριθμος Διατεταγμένων Κουλοχέρηδων» (RBA) στα πειράματά μας, ο οποίος χειρίζεται εκδοχές άλλων αλγορίθμων άμεσης μάθησης. Μελετάμε τη σύγκλιση του RBA μέσω των «επιχορηγούμενων συστάσεων», στις οποίες κάθε άρθρο είναι συσχετισμένο με ένα έσοδο, ενώ κάθε χρήστης είναι ένα υποσύνολο των άρθρων που τον ενδιαφέρουν και προέρχεται τυχαία από ένα σύνολο διαφορετικών πληθυσμών.

1 INTRODUCTION

One of the goals of this work is to introduce even the most inexperienced readers to an area in learning, known as *Online Learning*, which provides crucial solutions to many applications in today’s world. Online Learning deals with two fundamental problem models: the “Prediction with Expert Advice” problem and the “Multi-armed Bandit” problem. We examine each model and analyze some of the most popular algorithms derived from the latter model, which refer to the online contextual learning and the recommendations in rankings. Last but not least, we make several experiments on the ranked recommendations using an algorithm that we found the most interesting, we examine its performance and acquire useful conclusions.

1.1 Online Learning

This chapter provides an introduction in *Online Learning*, a prominent area with rich literature and deep connection to game theory and optimization that is more and more affecting machine learning, statistics, and artificial intelligence in the theoretical and algorithmic advances.

Concretely, online learning takes place in a sequence of consecutive rounds and there is an *agent* (or *learner*, *gambler*, *player*, etc.) who interacts with the *environment* (or *nature*); in each round (time step), the agent selects an action through the available ones and the environment returns a *reward*. The objective is to maximize the received reward and thus to determine the best sequence of actions to achieve that objective. However, the reward might be a negative feedback as the only information he has is just the immediate reward from the last chosen action.

In other words, the agent has to make a sequence of predictions and decisions (called *strategy* or *policy*) and on each round to be in the position of answering a given question. As an example, imagine a learner receiving an encoding of an email message and being questioned whether the message is spam or not. So, the learner comes up with a prediction mechanism in order to answer the question. In the end, as long as he predicts an answer, he gets the correct one for that question.

Another characteristic of online learning is that in its algorithms there is no training and test phase in contrast to the supervised learning; these two phases converge to one. In each round, these algorithms receive an instance, they make a prediction and then they receive the true label and get a loss as feedback for this prediction.

The performance of such algorithms is measured by a *loss function*. So, the ultimate goal of an algorithm is to optimize its loss function and as a result to minimize the notion of *regret*, which compares the cumulative loss of the used strategy and the best strategy. The regret is the expected loss, as a strategy does not play the best machine all the time. This is termed ’regret’ since it considers how ’sorry’ the learner is, in retrospect, not to have followed the predictions of the optimal strategy. In the example above, in order to achieve this goal, the learner might update his strategy after each round, so as to be more precise in the next rounds.

In the next sections and chapters, we are going to provide a brief presentation of two essential online problem models and some popular online learning algorithms, respectively.

1.2 Online Learning Problem Models

There are two fundamental problem models in Online Learning Area; the "*Prediction with Expert Advice*" and "*the Multi-armed Bandit*", which both have been studied extensively in numerous articles and books through the recent years, as they draw much attention in different scientific fields.

As far as the first problem model is concerned, in order to give further information, we should first cite the following scenario: "Suppose that someone wants to know if it is going to rain or not through the day. So, he consults different forecasting websites, so as to make his own prediction. Inevitably, after all at the end of the day, he knows if it rained or not and he is in the position to evaluate each website's prediction and make it clear which websites he should use next time".

This scenario is known as *Prediction with Expert Advice* problem, which is essentially an online classification problem and includes the forecaster who predicts a sequence of events (the rain) and the experts (the forecasting websites) by whom he takes advice about the future outcome of the sequence. Based on them, he makes his prediction and takes back the true label, which is useful for future predictions as he can exclude the experts provided false advice. It is important to underline that each prediction is made in each time step and is irrevocable. In other words, the forecaster cannot predict the sequence of events from the beginning and he cannot change a decision he has already taken in a round. What is more, he is in the position to know every expert's loss in each round.

There are many algorithms in the literature that try to solve this problem; the most well-known are the *Halving algorithm* by Littlestone [26], the *Weighted Majority Average* by Littlestone and Warmuth [28], the *Randomized Weighted Majority algorithm* and the *Exponential Weighted Average algorithm* [27]. We also refer the reader to the textbook of Mohri, Rostamizadeh and Talwalkar [28], for a brief reference in these algorithms. We should underline that Chapter 8 of [28] is the source of the problem described previously.

As for the second problem model, we should examine another scenario: "Suppose that someone wants to play in a casino's slot machines but he does not know which of them is the best to play for receiving the largest payoffs in the long run. He just chooses a slot machine to play and pulls the lever. In the end, he receives a reward, which is the payoff of winning the jackpot".

The goal here is to maximize the sum of the received rewards by finding the best slot machine's levers as early as possible and then keep playing using that best lever.

This scenario is known as the *Multi-armed Bandit (MAB)* problem, in which the gambler chooses an action per round and he gets a reward for that specific action. It is impossible to know the reward of the other arms in each round.

Such problems arise frequently in practice, for example in the context of on-line advertising [37] or clinical trials [38]. The MAB problem offers a simple theoretical formulation for investigating trade-offs between exploration and exploitation, which is going to be analyzed in the next chapter.

There are also many algorithms in the literature that attempt to solve the MAB problem, such as the *UCB1* algorithm proposed by Auer, Cesa-Bianchi and Fischer in [5], the *Exp3* algorithm presented by Auer et al. in [6] and the *e-Greedy* algorithm as shown by Sutton and Barto in [36].

1.3 Overview

Our work is organized as follows:

- *Chapter 2* analyzes the Multi-armed Bandit problem and its popular online algorithms that belong to the stochastic and the adversarial setting.

- *Chapter 3* studies the Ranked Recommendations area and its online algorithms that either provide diverse rankings or are based on multiple clicks.

- *Chapter 4* presents the online algorithms that are based on side information and are known as Contextual Bandits.

- *Chapter 5* provides some experiments in the ranked recommendations setting and compare the performance of an algorithm of it with the greedy algorithm for the max-*k*-cover problem.

2 THE MULTI-ARMED BANDIT PROBLEM

In chapter 1, there was a brief introduction into the *Multi-armed Bandit problem (MAB)*, where the example of a gambler and a casino's slot machines was given. This problem owes its name to the number of levers that are used; so, in the case of *k* slot machines or arms, the problem can also be called as the "*k-armed bandit*" or "*multi-armed bandit*" problem. It is noteworthy to underline that a slot machine was often called a "*one-armed bandit*".

The MAB problem has been originally introduced by Robbins in 1952 [32] and since then, it has been used extensively to model the trade-offs that are confronted by an automated agent, which aims at acquiring new knowledge by exploring the environment and to exploit its current knowledge. In this model, the gambler's purpose is to maximize the reward after a sequence of pulls, while each arm is supposed to deliver rewards from an unknown distribution.

In this regard, the agent does not know whether to explore unknown actions to find out more profitable data about the environment or to exploit the received data to maximize the reward. On the one hand, if the gambler chooses to play on a specific slot machine which he believes is the best (*exploitation*), he may fail to find out the one that actually returns a higher expected payoff. On the other hand, if he plays continuously in all the slot machines and gathers statistics (*exploration*), he may fail to play the best one often enough to get a high payoff. It is about two different situations that cannot happen at the same time in a single action selection; he has to choose exploitation or exploration or even balance between them. The more one explores, the less one exploits and vice versa. This dilemma is called *trade-off between exploration and exploitation*.

There are plenty of applications nowadays that use bandit algorithms in order to achieve better performance. Some popular examples are the below:

- ✓ *Online Advertising*: the goal of an advertising company is to maximize its income by displaying advertisements. The advertiser makes a profit every time a web user clicks on an advertisement. As happens to MAB problems, the objective is to collect information for the performance of each advertisement by using click-through rates and finally choose the one that has performed profitably.

- ✓ *Recommendation System*: a system that tries to catch up with the preferences of a user in order to provide him with the possibly best recommendations. This operation is accomplished by using bandit algorithms in several platforms dealing with news, books, or search queries.

- ✓ *High-Frequency Trading*: it is an algorithmic financial trading that is characterized by high turnover rates and high-speed progress. Its applications must be able to accommodate new input variables that are revealed sequentially. So, they are provided with a set of online algorithms that are responsible for making a decision over each new input, such as to submit a trade or not.

Before diving into the analyzation of some important algorithms of various categories, let us introduce ourselves to some basic definitions and notions based on [5, 6] that refer to all algorithms.

Definition 2.1 [5] A *strategy,* or *policy, A* is an algorithm that selects the next arm to play, based on a sequence of past plays and obtained rewards and its sequence of action choices is noted as:

$$(i_1 , i_2, \dots , i_T)$$

Definition 2.2 [6] At each time step *t* ∈ (1, …, *T*), a player, who follows a strategy *A*, knows only the rewards $x_{i_1}(1), \dots, x_{i_t}(t)$ of the previous actions $(i_1 , i_2, \dots , i_t)$ and he receives the *cumulative reward* equal to:

$$G_A = \sum_{t=1}^{T} x_{i_t}(t)$$

Definition 2.3 [5] Let $T_i(n)$ be the number of times arm i has been played by strategy *A* during the first n plays and $E|\cdot|$ denotes expectation. Then the *regret* of *A* after n plays is defined by:

$$R_T = \mu^* n - \mu_j \sum_{j=1}^{K} E\left|T_j(n)\right| \quad where \quad \mu^* \stackrel{\text{def}}{=} \max_{1 \le i \le K} \mu_i$$

Definition 2.4 [6] Given any time horizon $T > 0$ and a player who follows a strategy *A*, instead of choosing a strategy *B,* a gain or loss is occurred and is measured by the *worst-case regret*, which is the difference:

$$G_A(T) - G_B(T)$$

Definition 2.5 [6] In case of a player, who follows the best strategy *A* between strategies *A* and *B*, a gain or loss is occurred that is measured by the *weak regret,* which is the difference:

$$G_{max}(T) - G_B(T)$$

If we consider a bandit problem, for which we are informed about the best action to select and we keep selecting it continually, we will get the maximum expected reward. However, in a real-life problem, we are obligated to make several trials on the available actions, until we find out the best one that returns the maximum reward. So, the aim in such problems is to maximize the reward or even to minimize the regret (or loss we suffer). Figure 2.1 depicts the average regret of selecting several actions during such a problem.

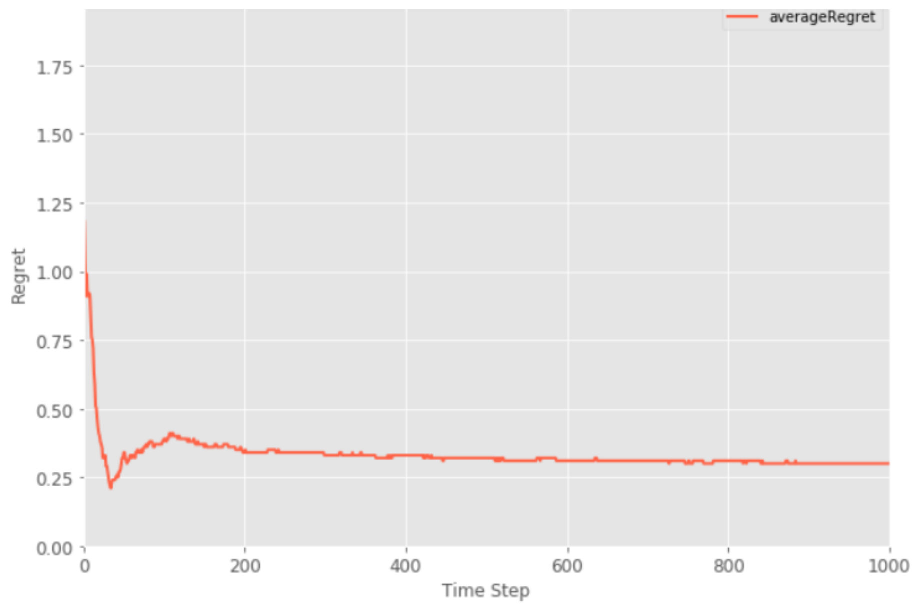


Figure 2.1: Representation of average regret

In the case of a strategy that on purpose stops exploring further and at the same time ends exploiting a suboptimal machine, we observe that, although there is low regret at the beginning, we are not close to the maximum reward for the given problem.

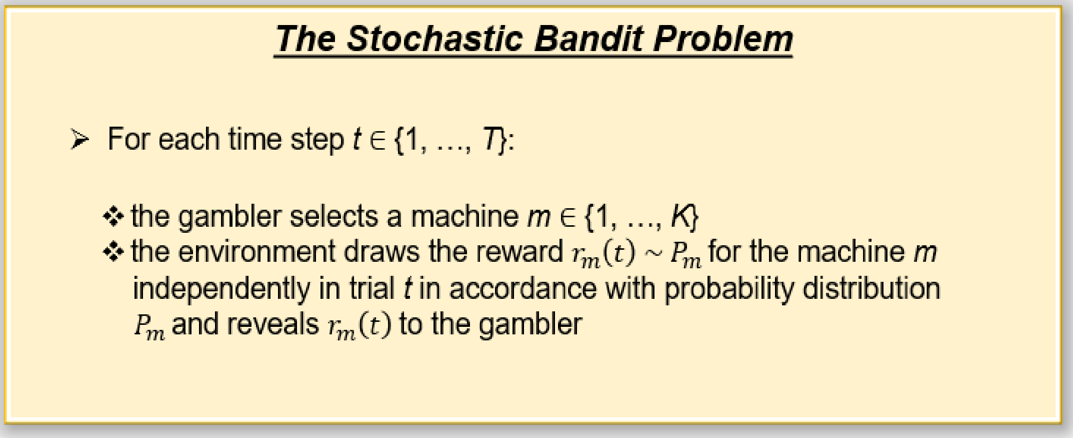
Finally, it should be mentioned that online learning algorithms handle one sample at a time with an update per round. There is no need for past data, but in view of the returned feedback, they appraise the strategy for the next rounds.

2.1 The UCB1 Algorithm

Lai and Robbins were the first to study the upper confidence index in [23], which is a quantity that is related to the sequence of the expected rewards of a bandit. So, they introduced an algorithm that computes upper confidence bounds for all the machines by optimizing the expected reward. That algorithm follows a principle known as “*optimism in the face of uncertainty*”.

They also proved that the expected regret must follow logarithmic behavior in the number of actions and that it is asymptotically minimum possible up to a sublogarithmic factor for a considered set of distributions. However, the computation of index was too slow in contrast to the equivalent index of Agrawal's family of policies he presented in [3], which depend only on the total rewards of each machine.

In their study, Lai and Robbins also worked on multi-armed bandit problems with parametric uncertainties and entered into the *stochastic multi-armed bandit problem*. According to the stochastic setting, they assumed that each of the *K* machines' rewards have initially unknown expectations and originate from unknown probability distributions ($P_1, \ldots, P_K$) with means ($\mu_1, \ldots, \mu_K$) respectively, and they are independent from the others' rewards. So, whenever an agent selects an arm, the environment draws an independent and identically distributed reward, as Garivier and Cappe characteristically mention in [15].



The Stochastic Bandit Problem

- For each time step $t \in \{1, \ldots, T\}$:
  - the gambler selects a machine $m \in \{1, \ldots, K\}$
  - the environment draws the reward $r_m(t) \sim P_m$ for the machine m independently in trial t in accordance with probability distribution P_m and reveals $r_m(t)$ to the gambler

Figure 2.2: The stochastic bandit problem

The stochastic bandit problem was primarily introduced by Robbins [32], who formalized the problem of decision-making under uncertainty, and captured the fundamental tradeoff that occurs between exploration and exploitation.

Later, there was a follow-up study from Auer, Cesa-Bianchi and Fischer [5], who came up with the case where the payoffs come from a bounded support (the payoff distributions are unconstrained otherwise) and introduced some policies that reach logarithmic regret uniformly over time. These policies have many variants, and they are known as *Upper Confidence Bound (UCB)* action selection. The most widely used in the literature is *UCB1* (usually called simply UCB in latter works), which is an algorithm for the stochastic multi-armed bandit problem and its expected cumulative regret is $O(\sqrt{KTlnT})$. In *Theorem 1* of that study, the authors proved that UCB1 achieves logarithmic regret uniformly over time.

As it is already mentioned, UCB1 is based on the principle of "*optimism in the face of uncertainty*". The more uncertain one is about an action, the more exploration it needs to be done on that action. For example, suppose having the Gaussian distribution of the mean reward *R(a)* for three different actions a_1, a_2 and a_3 after a couple of trials, as shown in Figure 2.3.

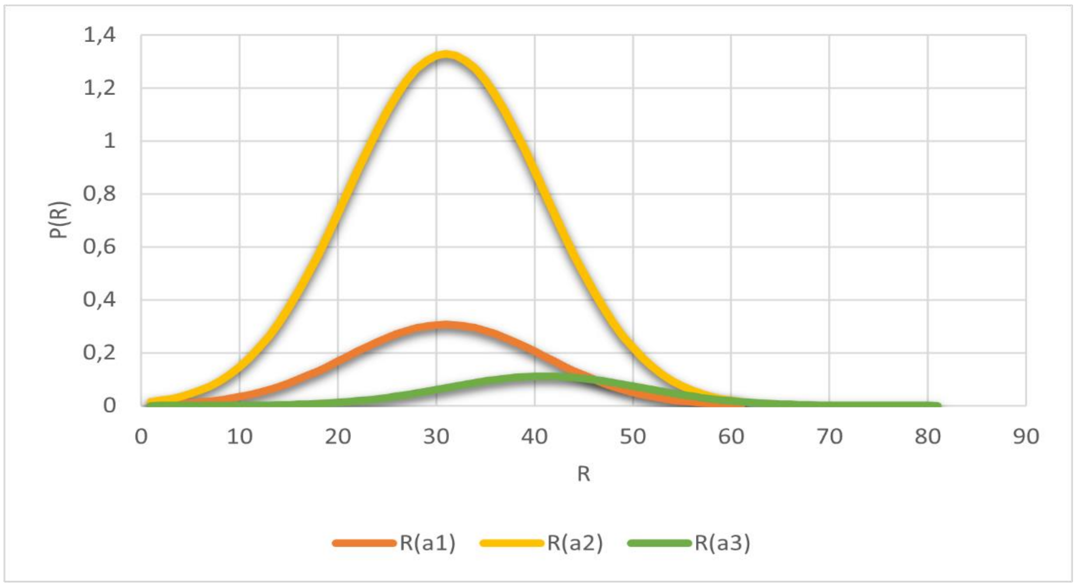


Figure 2.3: Gaussian distribution of mean reward R(a)

As it is obvious, the distribution for a_3 has the highest variance and, hence, the maximum uncertainty. UCB1 chooses the action a_3 and receives a reward. If there is still uncertainty about a_3, the algorithm chooses it again for the next trials, until the uncertainty is decreased under a threshold.

The index of each machine in UCB1 is the sum of two parts and the algorithm tries to maximize it:

- the current average reward *R(a)* and
- the one-sided confidence interval for the average reward $\sqrt{\frac{2\ln t}{N(a)}}$,

where *t* is the current time step in the algorithm, *R(a)* is the mean observed reward of action *a* so far prior to time *t* and *N(a)* is the number of times action *a* was played so far.

So, UCB1 firstly plays each of the available *K* actions once and gives initial values for the mean rewards. Then, for each round *t*, it computes the upper confidence bound for each action *a*:

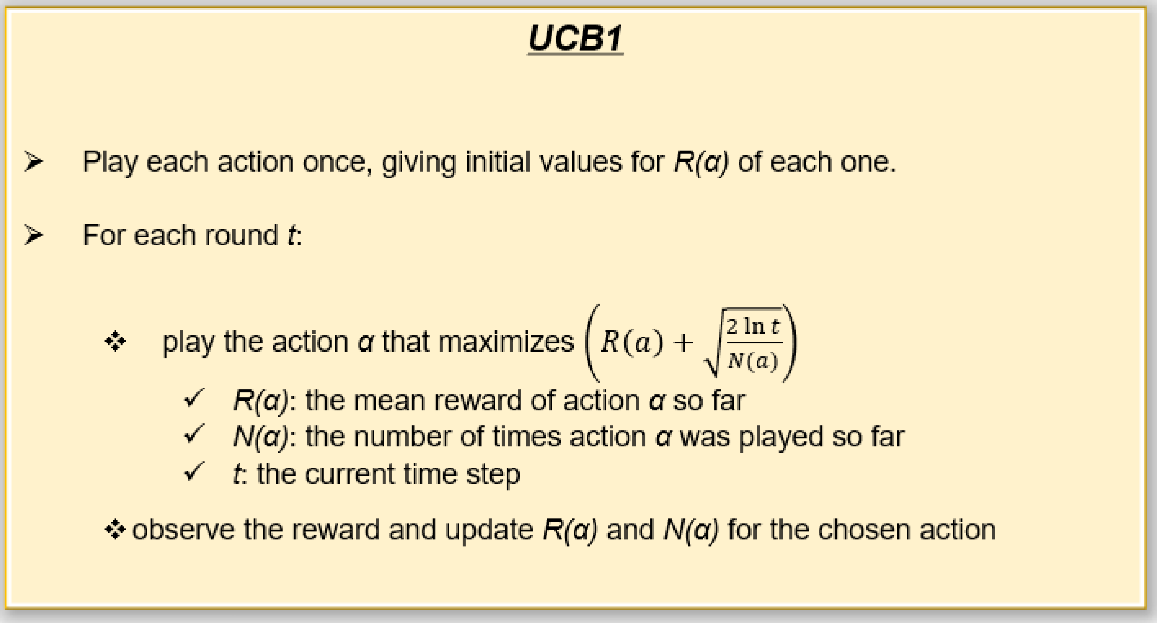
$$\left(R(a) + \sqrt{\frac{2\ lnt}{N(a)}}\right),$$

it chooses to play the action that maximizes the above total sum, it receives a reward, and it updates the mean reward for the chosen action.

In order to understand the above expression that corresponds to the selected arm, it is noteworthy to mention that:

- ✓ In the one hand, the uncertainty might decrease, as the number of times an action was chosen increases (*N(a)* is in the denominator of the square root).
- ✓ On the other hand, the uncertainty might increase, as *t* increases, but *N(a)* remains stable, when the algorithm selects another action apart from *a.*

All the above steps will ultimately lead to the optimal action being selected repeatedly in the end. Furthermore, we should underline that UCB1 uses *K* machines that have arbitrary distributions $P_1, P_2, \ldots, P_K$, with support in [0, 1]. Figure 2.4 concisely illustrates the pseudocode for those steps, based on [5].



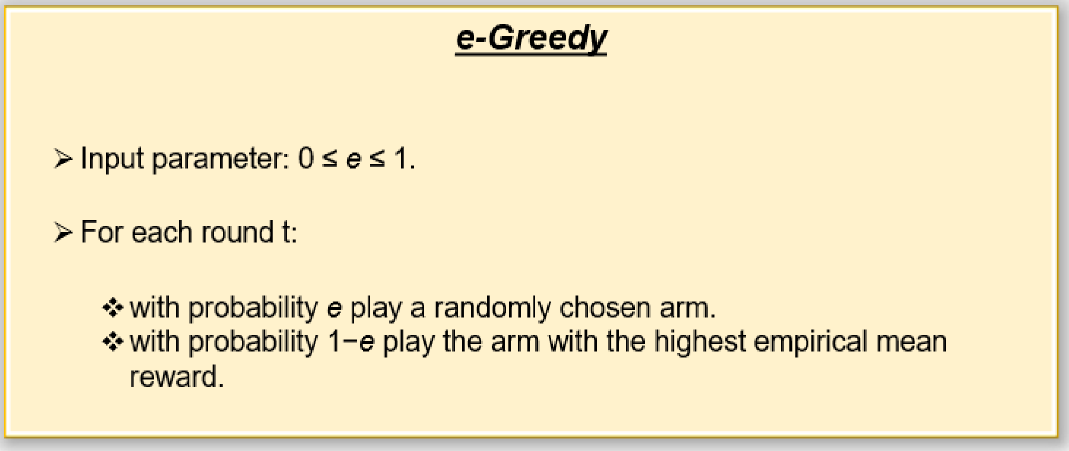
```
UCB1

➢ Play each action once, giving initial values for R(α) of each one.
➢ For each round t:
    ❖ play the action α that maximizes $\left(R(a) + \sqrt{\frac{2 \ln t}{N(a)}}\right)$
        ✓ R(α): the mean reward of action α so far
        ✓ N(α): the number of times action α was played so far
        ✓ t: the current time step
    ❖ observe the reward and update R(α) and N(α) for the chosen action
```

Figure 2.4: Pseudocode for UCB1

2.2 The *e*-Greedy Algorithm

In [36], Sutton and Barto introduced another algorithm for the stochastic multi-armed bandit problem, called *e-Greedy*, which can balance the trade-off between exploration and exploitation by using a fixed ratio of exploration $e \in [0,1]$. This algorithm, at each time step $t \in \{1, \ldots, T\}$, greedily plays with probability $1-e$ the arm *a* ∈ {1, …, *K*} with the highest empirical mean reward r_a (exploitation) and with probability *e* a randomly chosen arm a (exploration).

If *e* is constant throughout the algorithm's execution, just a linear bound on the expected regret can be reached. Cesa-Bianchi and Fisher in [8] showed poly-logarithmic bounds for the algorithm's variants, where *e* decreases over time. The pseudocode of the *e*-Greedy algorithm is shown in Figure 2.5.



e-Greedy

```
➢ Input parameter: $0 \le e \le 1$.

➢ For each round t:

    ❖ with probability $e$ play a randomly chosen arm.
    ❖ with probability $1-e$ play the arm with the highest empirical mean
      reward.
```

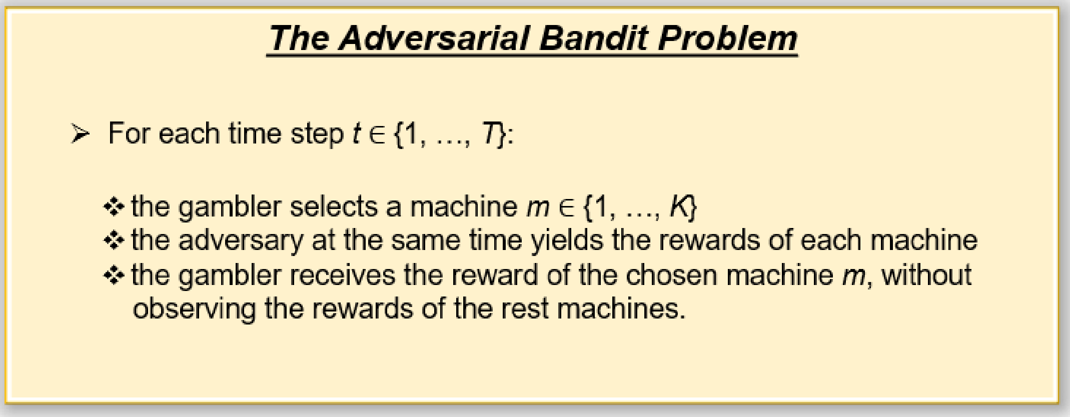
Figure 2.5: Pseudocode for e-Greedy

2.3 The Exp3 Algorithm

The *adversarial* or *non-stochastic* multi-armed bandit problem is a variant of the bandit problem, where there are no statistical assumptions for the generation of the rewards.

According to this problem model, each machine is assigned from the beginning an arbitrary sequence of rewards, from which the gambler receives the corresponding reward, after he pulls one out of the *K* machines. Essentially, he plays against an *adversary* (or *opponent*), who decides the reward for each machine for each time step *t* ∈ {1, …, *T*}. So, when the adversary receives the chosen machine *m* ∈ {1, …, *K*}, he returns to the gambler a reward for that action for round *t*.

Unfortunately, in the real world, the reward structures are more complex than a coin flip and for the gambler’s point of view it is arguably naïve to adopt some kind of “optimism”, as happens in the stochastic problem. The adversary, before deciding the attribution of the rewards per each machine, he invents a strategy depending on the gambler’s past choices and draws the rewards to be unpleasant for the gambler. So, whenever the latter finds out and plays the optimal machine, the adversary could make worse the rewards for that machine and even turn into a non-optimal machine any longer.


The Adversarial Bandit Problem

- For each time step $t \in \{1, \ldots, T\}$:
  - the gambler selects a machine $m \in \{1, \ldots, K\}$
  - the adversary at the same time yields the rewards of each machine
  - the gambler receives the reward of the chosen machine m, without observing the rewards of the rest machines.

Figure 2.6: The adversarial bandit problem

A characteristic example of the adversarial model is the stock trading. An investor buys shares that currently listed at a low price, even if their payoff is not profitable at that time. If he buys a high volume of them, he will cause their price to surge. However, he cannot predict how much their price would become, as at the same time there would be many investors who buy or sell shares.

The pioneers of the adversarial bandit problem were Auer et al. [6], who proved that the nature of the reward generating process of each machine depends on time step *t*. An adversary with unbounded computational power determines the reward at each time step for every machine. Since then, there are several extensions of that type, such as the bandit online linear optimization [2], or the online shortest path problem [19], which belongs to the family of the combinatorial bandits.

There was a different approach for the non-stochastic bandit problem reported by Gittins [17] and Ishikida and Varaiya [20]: according to their approach, they assumed that the gambler computes in advance the exact rewards he is going to receive from each machine and consequently it results to become an optimization problem and not one of exploration and exploitation.

Auer et al. [6] proposed an efficient and randomized algorithm that performs well in the worst case against an adversary; they introduced the *Exponential-weight algorithm for Exploration and Exploitation (Exp3)*, whose expected weak regret is $O(\sqrt{KTlnK})$.

This algorithm is based on the *Hedge algorithm* proposed by Freund and Shapire in [13, 14], which in turn is a variant of the weighted majority algorithm [27] and the aggregating strategies of Vovk [39]. The setting studied by Freund and Shapire applies a full information game: not only the gambler scores in each round *t* the payoff of the chosen machine, but also he gains access to all of the machines' payoff and not just that of the chosen machine. In the meantime, the adversary decides the loss of each machine and the gambler suffers only the loss of the picked machine.

The Exp3 algorithm creates an empirical distribution based on the actions' feedback. Moreover, as shown by Seldin et al. in [34], the algorithm picks an action according to a Gibbs distribution depended on the empirical importance-weighted rewards of the arms. This distribution is a curved combination with a parameter gamma

(γ) in which, for each action, a probability exponential mass is added to the estimated reward for that action. And that happens because it is desirable to try out all actions and get an optimal estimate of the rewards for all of them. Differently, there is a possibility for the algorithm not to take into account a profitable action because in the beginning there were low rewards and as a result, the greater rewards might appear afterwards are not observed, as the action was not selected.

There are many advantages using the importance-weighted sampling in complicated problems, such as in the stochastic multi-armed bandits with side information, as mentioned by Seldin et al. in [33]. It was also used by Bubeck and Slivkins [7] in a strategy that combines stochastic and adversarial settings.

Generally, in an adversarial setting, the algorithm deals with no such fixed reward distributions of the arms, as the next decision one makes might result in the worst possible payoff. Regarding to [6], there are by proof regret bounds that apply to any worst-case sequence of rewards.

The algorithm proceeds in rounds $t \in \{1, 2, \ldots, T\}$. For each t, a decision is made based on a selected probability distribution $p_i(t)$. The result of the decision is a reward that is used to update the labels according to the last step of the pseudocode. In the next round, these labels are used to recompute the distribution according to the first step of the loop.

```
Exp3

➢ Input parameter: $0 \leq \gamma \leq 1$
➢ Initialize the weights for each arm $w_i(1) = 1$, for $i \in \{1, \ldots, K\}$
➢ In each round $t$:
    ❖ Set $p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}$ for each $i$
    ❖ Select the next action $i_t$ at random according to the distribution of $p_i(t)$.
    ❖ Receive reward $r_{i_t}(t) \in [0,1]$
➢ For each arm $j$ in $(1, \ldots, K)$:
    ❖ Set the estimated reward $\hat{r_j}(t) = r_j(t) / p_j(t)$ if $j = i_t$, otherwise 0.
    ❖ Set $w_j(t+1) = w_j(t)e^{\gamma \hat{r_j}(t)/K}$
```

Figure 2.7: Pseudocode for Exp3

The pseudocode for the Exp3 algorithm appears in Figure 2.7, which is based on its presentation in [6].

It is noteworthy to report that the probability *$p_i(t)$* for each arm $i \in \{1, \ldots, K\}$ depends not only on the weights w_i, but also on a uniformly random component determined by the parameter *γ*. This parameter controls the tradeoff between exploration and exploitation. When *γ* tends to be equal to 1, the algorithm only explores in order to find the optimal action and exploit it, whereas when *γ* tends to be equal to 0, the algorithm only exploits the optimal arms found so far.

More specifically:

$$\begin{cases} p_i\,(t) \xrightarrow{\gamma \to 1} \dfrac{1}{K} \\ p_i\,(t) \xrightarrow{\gamma \to 0} \dfrac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} \end{cases} , for\ i\ \in\ \{1, \ldots, K\}$$

An important issue regarding to the γ parameter is its preferred value, as it defines the randomness of the algorithm. As stated in [6], the optimal choice of *γ* is when, for any time horizon *T* > 0 and an upper bound $g \geq G_{max}$, Exp3 runs with input parameter:

$$\gamma = \min\left\{1, \sqrt{\frac{K\ lnK}{(e-1)g}}\right\}.$$

If *T* is known and the payoff of an action is always lower than 1 in total trials number, we assume that *g* = *T*. So, the expected weak regret of Exp3 is bounded as follows:

$$G_{max} - E\left|G_{Exp3}\right|\ \leq 2.63\,\sqrt{g\ K\ lnK},$$

or

$$G_{max} - E\left|G_{Exp3}\right|\ \leq 2.63\,\sqrt{T\ K\ lnK}$$

Moreover, for the selected action, the algorithm sets the estimated reward $\widehat{r_j}(t)$ to:

$$\widehat{r_j}(t) = r_j(t)\ /\ p_j(t),$$

which improves the rewards of actions with low probability, namely those that they are more unlikely to be drawn. So, such actions get an unexpectedly great reward and have now the chance to be chosen in a future round and, hence, it becomes easier for the algorithm to discover the optimal action, as exploration is strengthened. In the final step of Exp3, according to the estimated reward, the exponential weight is computed:

$$w_j(t+1) = w_j(t) e^{\frac{\gamma \hat{r}_j(t)}{K}}$$

3 RANKED RECOMMENDATIONS

The traditional recommendation engines have the ability to work properly when they make computations and evaluations in an offline manner. However, nowadays, where data are produced in rapid rates, services like news aggregators need to repeatedly adjust their recommendations using fast online procedures. For instance, articles about the World Braille Day might be popular on January 4th, but they might not be of interest on the morning of the 5th. As a result, online algorithms and especially multi-armed bandits provide appealing solutions in the area of the ranked recommendations, in order to optimize recommendations and, hence, constantly fulfill users' taste.

The satisfaction of a user can be measured by *abandonment*. This means that the user found no potential interest in any item displayed in the recommended list and so, he did not click on any item. In the terminology of information retrieval abandonment is also known as *%no*. Consequently, ranked recommendation issues aim to minimize abandonment.

In a ranked recommendations problem, there are k machines, where $m \in \{1, ..., k\}$ is one of the k machines to be played, and $r_m \in [0, 1]$ is the reward for machine m. The problem is a game played in rounds: on each round $t \in \{1, \ldots, T\}$, a list of ranked recommendations of dimension D is created, and then the reward r_m for the chosen by the player machine is returned to him. The position of each item in the list is indexed by $d \in \{1, \ldots, D\}$.

If we examine the users' taste further, we could observe a diversified behavior from the side of users. For example, when one uses the word "novel" in his query to a search engine, we do not know whether he refers to the noun, which is a long-written story, normally dealing with imaginary people and events, or the adjective, which means new and different from what has been known before. Thus, in the case of a document addressing a fictional book, it is impossible to be of interest to all the users. As a result, the recommendation systems are obligated to make ranked recommendations in a diversified manner in order to take into consideration the intensions of the users.

There have been developed many approaches that aim at producing diverse rankings. A diverse ranking is a recommendation list of items, such as articles, or advertisements, that, on the one hand, is relevant and connected to the submitted query and, on the other hand, its items are diversified between them. In that way, it is more possible for each user to find what he seeks and so we have a set of satisfied users in the end.

3.1 REC and RBA

Radlinski, Kleinberg and Joachims were the first to examine in [31] the problem of presenting diverse rankings. They proposed an online learning approach that directly learns a diverse ranking of documents from usage data with the tendency of maximizing the total number of closer-to-the-users'-preferences rankings displayed over all time and thus the clicks of the users.

Hence, they presented two algorithms that take advantage of the clicking behavior of users and return a diverse ranking of documents. In other words, they tried to eliminate abandonment, which, as it is already mentioned, is a measure of user satisfaction and means that there is no essential interest in the provided results. Furthermore, they wanted to maximize the probability to find a relevant document in the top positions of the ranking by examining the relevance among different documents.

The first algorithm is *Ranked Explore and Commit (REC)*. It is a simple greedy policy that takes for granted that both user's interest and documents are held stationary over time. At the beginning, for each rank, it explores every document a fixed number of times and records the number of clicks it receives through the exploration. In the end, the algorithm cedes to the current rank the document with the best score. The proved regret bound of REC is in $O(K^3 P/e^2 \ln(K/\delta))$.

Based on [31], the algorithm's pseudocode is given in Figure 3.1.

```
REC

➢ Input: parameters $e$, $\delta$, $k$, documents ($d_1$, …, $d_n$)
➢ Set $x = 2k^2/e^2 \log(2k/\delta)$
➢ Create list with $k$ arbitrary documents
➢ For each rank in (1, …, $k$):
    ❖ Initialize to zero each document's score
    ❖ Loop (1, …, $x$) times:
        ▪ For each document in (1, …, $n$):
            ✓ Display the current document in the current rank
            ✓ Record clicks
            ✓ If the current document clicked, increase by 1 its score
    ❖ Commit to the current rank the document with the best score
```

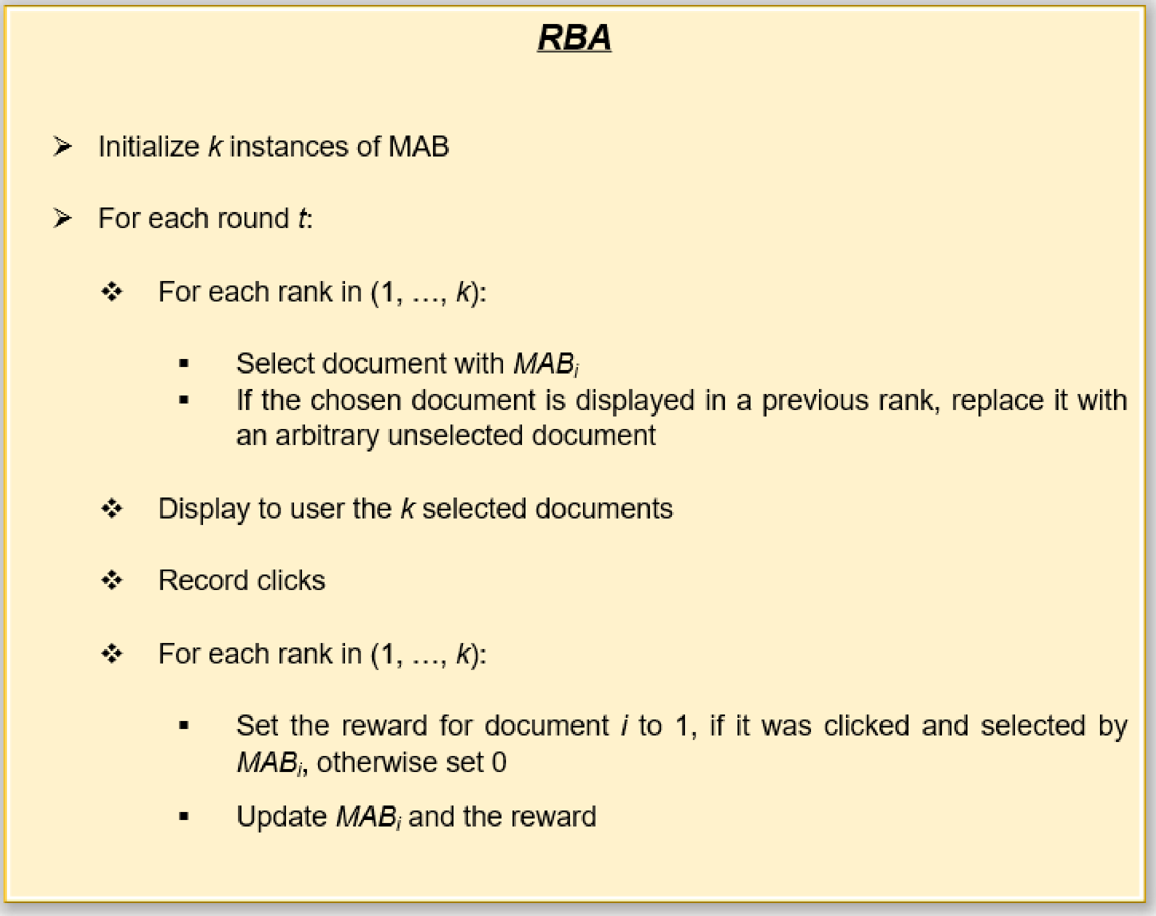
Figure 3.1: Pseudocode for REC

The second algorithm is *Ranked Bandits Algorithm (RBA)*. It differs from REC's assumption that user tastes or documents cannot change over time. Furthermore, RBA takes place in two phases:

- ❖ firstly, the algorithm runs an instance of a multi-armed bandit algorithm for each rank in order to recommend documents, until all ranks have been filled with documents. In the case of a document that has already been selected at the highest rank, the next one is selected arbitrarily instead. As soon as the procedure is done, the selected items are displayed to the user.
- ❖ secondly, whenever a user clicks on a document or not, the instances of the multi-armed bandit algorithm must be updated, as well as the received rewards which take the value of 1 or 0, respectively.

RBA has an upper bound for the expected regret in $O(k\sqrt{KTlogK})$, where k is the number of instances of Exp3. Its computation is based on instances of the Exp3 multi-armed bandit algorithm, as it achieves the optimal bound. Through their experiments, however, the authors used instances of the UCB algorithm and found that the performance of RBA was better, under the condition that the user interests remain static; but that condition violates their initial assumptions. To wit, that variant of RBA has improved performance with the cost of a weaker theoretical guarantee.

Figure 3.2 presents the pseudocode of RBA, based on [31].



```
RBA

➢ Initialize k instances of MAB
➢ For each round t:
    ❖ For each rank in (1, …, k):
        ▪ Select document with MAB_i
        ▪ If the chosen document is displayed in a previous rank, replace it with
          an arbitrary unselected document
    ❖ Display to user the k selected documents
    ❖ Record clicks
    ❖ For each rank in (1, …, k):
        ▪ Set the reward for document i to 1, if it was clicked and selected by
          MAB_i, otherwise set 0
        ▪ Update MAB_i and the reward
```

Figure 3.2: Pseudocode for RBA

Both REC and RBA depend on the parameter *k*, which is stable and defines the number of documents to be displayed in the recommended list provided to a user. It is a parameter that cannot be chosen from the algorithms themselves. They just receive it as input and try to find out which *k* documents are the optimal to display. On the one hand, REC takes that parameter for granted during the phase of initializing and declaring the variables, while, on the other hand, RBA implements in the same phase a multi-armed bandits instance for each rank and each of the *k* instances maintain an index for each document.

The selection of the *k* parameter for the algorithms analyzed above, refers us to the max-*k*-cover problem. In this problem, we consider each website visitor as a subset of documents (those which are of his interest) and we aim to focus on the *k* documents that could impel as many visitors as possible in order to choose at least one of that provided *k* documents. Selecting such number of documents is a *maximum coverage* (or *max cover*) problem. As Chierichetti, Kumar and Tomkins [10] reported in their study, these problems arise whenever we search for the optimal collection of items (documents); however, the items may partially overlap in the provided value, and should not be counted more than once.

The canonical problem of that type, max-*k*-cover, selects *k* sets from a family of subsets of a universe, in order their union to be as large as possible. The problem is *NP-hard*, which means that every known algorithm that solves it optimally demands exponential time.

In polynomial time, this maximization problem can be approximated by a simple greedy algorithm to within a factor of (1−1/*e*) of the best possible, which means that it iteratively selects the document that has relevance to most of the users, for whom no relevant document has been selected so far. That factor is known as "*approximation ratio*" of the greedy algorithm to the optimum.

3.2 IBA

An extension of the ranked recommendations discussed in the previous section is a family of algorithms that support *multiple clicks*. The recommendation systems that use such algorithms take advantage of the behavior of a user, specifically when he clicks on more than one item from a recommended list. In this case, there is more information about the returned rewards, which can allow a faster learning rate and therefore lead to beneficial results in recommendation.

Under such circumstances, Kohli, Salek and Stoddard [22] introduced an online algorithm, called *Independent Bandit Algorithm (IBA)*, that both quickly optimizes recommendation and minimizes abandonment based on users' clicks, called *tastes*. It has an upper bound for the expected regret in $O(\sqrt{nKlogT})$, where *n* is the number of instances of UCB1, when it is used as the bandit algorithm for IBA.

This algorithm depends on the *Probability Ranking Principle (PRP)*; according to PRP, all the articles have to be ranked in descending order of relevance probability. There is also another principle followed by IBA, which is based on *diversity* and handles the user intents. For example, when a user clicks the word "*jaguar*" in a search engine, we do not know if he seeks for the animal, the car, or the American football team. Another algorithm that uses this diversity principle is the *Ranked Bandit Algorithm (RBA)*, which was mentioned in the previous section and introduced in [31].

Similarly to RBA, IBA uses an instance of a multi-armed bandit algorithm for each rank of the recommendation list. These instances act independently to find the most suitable items to recommend. However, they are independent between each other, as IBA uses the *stochastic optimization concept* (or *correlation gap*) [22], in contrast to the diversity principle.

The authors compared IBA with RBA through their experiments using MAB instances of the e-Greedy and the UCB1 algorithms and proved that the first has near-optimal regret and in most cases outperforms RBA. Furthermore, when a user clicks on an article, IBA provides 1 as reward, while RBA provides 1 as reward only to the first clicked article; so, feedback is the main difference between them.

The algorithm's steps are shown in Figure 3.3, as presented in [22].

```
IBA

➢ Initialize $k$ instances of MAB
➢ For each round $t$:
    ❖ For each rank in (1, …, $k$):
        ▪ Select a document with $MAB_i$ that was not selected in highest ranks
    ❖ Display to user the $k$ selected documents
    ❖ Record clicks
    ❖ For each rank in (1, …, $k$):
        ▪ Set the reward for document $i$ to 1, if it was clicked, otherwise set 0
        ▪ Update $MAB_i$ and the reward
```

Figure 3.3: Pseudocode for IBA

3.3 dcmKL-UCB

The click logs of a search engine provide a wide source of information, which is actually biased. A primary source of bias is the order of presentation of documents in a list; the probability of clicking on a document depends on the document's position in the provided list. That is what exactly studied Craswell et al. in [12], who tried to model that probability. They proposed four different models for user clicks, the most important from which is the *cascade model*, which is a popular model of the behavior of a user in web search that is based on two basic hypotheses.

Specifically, a user examines the search result page from top to bottom (*linear traversal hypothesis*) and decides whether click on a document or not, after he examines it (*examination hypothesis*). According to that model, a web user examines the documents one-by-one in order until the first click; after that click, the user leaves the result page and never comes back. In other words, all documents being under the clicked document are skipped.

Furthermore, Guo, Liu and Wang proposed in [18] a generalization of the cascade model to multiple clicks, known as the *dependent click model (DCM)*. Their model includes a couple of position-dependent parameters, so as to model the probabilities that an average web user returns to the search result page, who continues the examination of the page after a click.

A few years later, Katariya et al. examined the *DCM bandits* in [21], which are a variant of the DCM [24] that in its turn is a variant of the cascade model [12] with the difference of not exactly following it to multiple clicks. And that is because they were interested in multiple clicks and not just one in a list of choices; hence, DCM bandits remain a generalization of the cascade model.

What is more, the DCM is parameterized by item-dependent attraction probabilities and position-dependent termination probabilities. DCM bandits describe the user's behavior when he watches a list of web pages, as a result of his search, and clicks on what he feels is near to the item he is looking for (he is attracted by an item). After clicking one item, the user terminates the search and the system considers that he is satisfied or the user continues examining the list's items under the one he selected and the system considers that he is not yet satisfied. A learning agent is responsible for observing this process and receives 1 as a reward when the user clicks on at least one item, otherwise the agent receives 0. The agent uses attraction weights and termination weights for each item and keeps them up to date, based on the user's clicks. So, the goal is to maximize the user's satisfaction through the possible recommendations and minimize the cumulative regret.

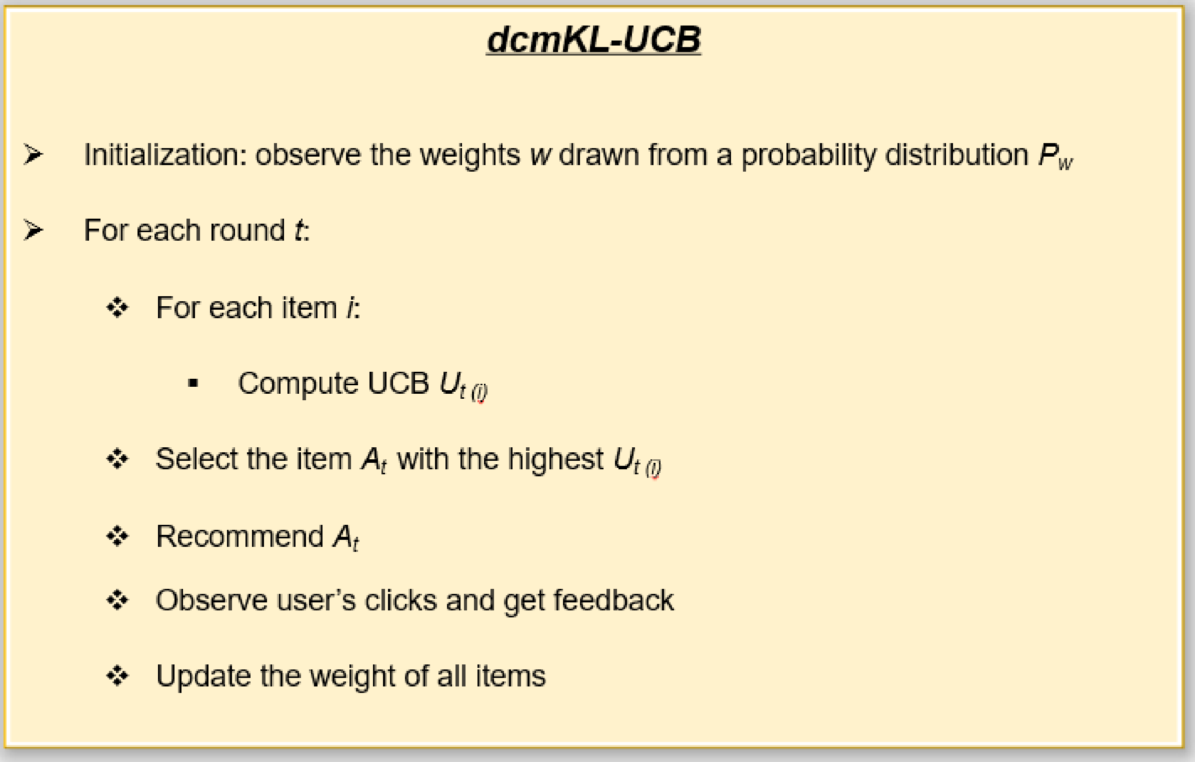
However, the agent in DCM bandits does not know if the user is really satisfied; the only thing he knows is about the clicks. That problem is solved with *dcmKL-UCB algorithm*, which is an extension of Garivier and Cappe's *KL-UCB (Kullback-Leibler UCB)* in [15] and belongs to the family of the UCB algorithms. The regret of dcmKL-UCB is $O(L-K)$, while its regret in ranked bandits is $O(KL)$, where *L* is the number of items and *K* is the number of recommended items.

In that point, it is important to mention a few facts regarding to KL-UCB algorithm. It is an online, horizon-independent index approach for the stochastic MAB problems and owes its name to the Kullback-Leibler divergence, which is used to compute the upper confidence bounds of the arm rewards. In statistics, the Kullback-Leibler divergence or the relative entropy is a measure which calculates the difference between a probability distribution and a second reference probability distribution. According to [15], KL-UCB is efficient and always performs better than UCB and its variants. Specifically, for arbitrary bounded rewards, KL-UCB satisfies the best regret bound and for the case of Bernoulli rewards, the algorithm reaches the Lai and Robbins's lower bound for binary rewards. Hence, KL-UCB is both a general-purpose algorithm for bounded rewards, and an optimal solution for the binary instance.

The dcmKL-UCB algorithm firstly calculates the upper confidence bounds (UCBs) on the attraction probabilities at each round and for all items and then recommends those with the highest UCBs in the form of a list, after selecting them. Finally, depending on what the user is attracted to or not, it provides a feedback and updates the kept weights by the agent.

Even though the algorithm performs well on several problems, some of the assumptions the authors have made are violated. For example, the learning agent does not observe the feedback or generally, the multiple click character of this model is simplified to a single click one. Hence, it is doubtful whether it can be applied in real-life problems under the primary assumptions.

The pseudocode of dcmKL-UCB is displayed in Figure 3.4, as presented in [21].



```
dcmKL-UCB

➢ Initialization: observe the weights $w$ drawn from a probability distribution $P_w$
➢ For each round $t$:
    ❖ For each item $i$:
        ▪ Compute UCB $U_{t(i)}$
    ❖ Select the item $A_t$ with the highest $U_{t(i)}$
    ❖ Recommend $A_t$
    ❖ Observe user's clicks and get feedback
    ❖ Update the weight of all items
```

Figure 3.4: Pseudocode for dcmKL-UCB

4 Contextual Bandits

A recently popular extension of the multi-armed bandits problem is the *contextual bandits problem*. The philosophy of the contextual bandits setting is relevant to that of the standard multi-armed bandits apart from a modification, according to which, the learner also observes context and, hence, it can be used to determine which action to select. In that case, the payoff depends on both the context and the chosen action. As *context* or *side information* we define all the available information of a learner (or a decision maker) in the form of some attributes (or features). For example, one would observe the gender, age, location, and other attributes that refer to the topic of a query, whenever a user issues one.

The problem of bandits with context has been closely studied through the recent decades and we refer the reader to the reports of Pandey et al. [29], Wang et al. [41] and Strehl et al. [35]. The name "*contextual bandit*" is borrowed from Langford and Zhang [24], but this problem is also known by other names such as "*bandits with side information*", "*bandit problems with covariates*", "*associative reinforcement learning*", and "*multi-armed bandits with expert advice*", among others.

In order to understand that setting, let us use a characteristic example of a recommendation system [24], which tries to match advertisements (machines) to webpage contents (context information) by recording clicks. Every user (learner), who visits a page, has a random draw of context from an unknown to the user distribution. In that case, the user's online profile (such as the location, the gender or even the age) can be used as context. Whenever the user clicks on an advertisement, a revenue is generated. The aim of the recommendation system is to maximize the expected revenue by setting the most relevant advertisement on each webpage and, by extension, mapping the context information to the advertisements.

What is more, the *adversarial multi-armed bandits problem with expert advice* was initially analyzed by Auer et al. [6] and remains one of the most fundamental models in online learning. It mixes prediction with expert advice [27, 39] with the framework of multi-armed bandits' limited feedback [6]. In this model, there is a repeated game played in T rounds between a learner and an adversary; the learner is provided by N experts with advice about which documents to choose in each round $t \in \{1, \ldots, T\}$, where the experts essentially represent randomized algorithms for action selection. In other words, the agent does not decide alone which document to select, but instead there is an expert to consult in each round t, so as to select an action, and simultaneously the adversary assigns a loss to each action, which remains unknown to the learner, except for that of the chosen action. As a result, he follows a strategy in order to succeed the mapping of context to the documents.

There are two aspects about the experts; the *randomized experts* and the *deterministic experts*, who form a special case of the randomized ones. Specifically, the first would choose an action at random from a probability distribution made for each action and the latter would recommend just a specific action to select.

In many web-based scenarios, there are several problems to defeat, such as the chance of a significant number of users to be entirely new and the recommendation system not having historical records of their clicking behavior. This situation is called "*cold start*".

Furthermore, in the contextual setting, there are two models that have been deeply analyzed: the *non-stochastic* and the *stochastic* one. In the non-stochastic case, the matching, for example, between advertisements and webpages remains unknown through the procedure. As for the stochastic setting, there is a vector of coefficients *theta* derived from an unknown distribution, which multiplies with the users' features, in order to produce the rewards. So, the expert attempts to learn about an accurate evaluation of the vector of coefficients *theta*. Essentially, it is a linear model, in which each expert is equivalent to *theta*.

Two of the most known works on contextual online learning are described in the following paragraphs.

4.1 LSBGreedy

In order to produce diverse rankings, Yue and Guestin [42] had the lateral thinking of combining the ranked recommendations with the contextual bandits. They examined the problem of the linear submodular bandits, which aims to train and optimize a class of feature-based submodular utility models. So, they presented the *LSBGreedy algorithm*, as a solution to that problem, which basically adds a document to an already existing set of documents as a linear model with regard to the preferences of a user. This algorithm has a regret bound in $O(d\sqrt{LT})$, where *d* is the desired number of topics (or concepts) to cover, and L is the number of articles (ignoring log factors).

Specifically, this algorithm respects the preferences of a user, as it is based on features, and in each round recommends some document rankings picked among thousands of articles per day and results in optimally diversified and personalized recommendation for that round. LSBGreedy is a flexible algorithm which can generalize to new predictions.

The class of *submodular functions* is a natural class of utility functions that has been studied extensively in the literature through the years. A function *f* is submodular if, for any 2 sets of articles *A* ⊆ *B*, the marginal contribution of an article *x*, is bigger when added to *A* than when added to *B* (diminishing returns), i.e.,

$$f(A \cup x) - f(A) \geq f(B \cup x) - f(B)$$

As far as the optimizing class of utility functions in this setting is concerned, utility encompasses the interesting information included in a set of recommendation articles, as we examine the case of personalized news recommendation. In their study, the authors used extensively the key properties of monotony and submodularity of utility functions. Essentially, considering a set of recommended documents, the notion of utility measures the amount of interesting information covered by that set and, hence, the payoffs.

The extension that was given in comparison to the contextual bandits setting reported in [25] is the optimization of the algorithm in the feature-based exploration-exploitation trade-off. LSBGreedy deals with the issue of how to model the interests of a user based on the available features. It also goals to make a recommendation of diversified sets of articles and not just single articles. So, the algorithm succeeded in facing diversification and feature-based exploration, which both belong to the wide range of challenges when designing information retrieval systems, and improve the performance of a system even in a limited number of rounds.

Through their experiments in a live user study, they compare their algorithm to *Multiplicative Weighting (MW)*, *RankLinUCB* (a combination of LinUCB and RBA) and *e-Greedy*. The results show that LSBGreedy and RankLinUCB perform much better than the other two algorithms. Moreover, LSBGreedy outperforms the other algorithms and converges to a near-optimal model.

4.2 LinUCB

Li et al. in [25] proposed *LinUCB*, which is an online algorithm that models personalized web-based services as a contextual bandit problem for linear payoff functions, and they tested it on a real-world application, the news article recommendation. The LinUCB algorithm is motivated by the *UCB algorithm* of Auer et al. [6] and the *KWIK (Knows What It Knows) framework* of Walsh et al. [40].

In the contextual bandit setting with linear payoffs, the learner competes with all the linear predictors on the feature vectors. An example application for this contextual bandit setting is the problem of choosing internet banner advertisements, as reported by Abe et al. [1]. In that type of problem, the advertisements, the webpage features and the users' attributes like age or sex are used to construct a linear function in order to predict the probability of clicking on a given advertisement.

Similar to the LSBGreedy algorithm, LinUCB recommends articles based on context and aims to maximize the number of user clicks. Supposing that the payoff function is linear, the algorithm chooses the bandit with the highest upper confidence bound (like all UCB methods) and so the authors call it *Linear Upper Confidence Bound (LinUCB)*. Its formulation is very close to its ancestor algorithm called *Linear Reinforcement Learning (LinREL)* of Auer [4], which operates in a similar framework.

The main concept of both algorithms is to calculate the expected reward of each machine by constructing a linear combination of the previous rewards of the machine. To accomplish this, LinUCB disintegrates the current round's feature vector into a linear combination of feature vectors observed on the previous rounds and uses the computed coefficients and received rewards of the previous rounds to compute the current round's expected reward. However, LinREL is a more complex algorithm.

There are some practical advantages of LinUCB over LinREL: firstly, it is plain in its implementation and secondly, it is computationally efficient as it applies ridge regression, which is simpler to solve with standard software packages and less susceptible to numerical instability issues compared to the eigenvalue decomposition used by LinREL, as mentioned by Chu et al. [11].

The LinUCB algorithm is presented in two forms: one for the *disjoint linear models* and the other for the *hybrid models*. There is difference between them in the type of features. In particular, there are some features that are not shared among bandits or they are related to a specific bandit and others that are common among the bandits. The former instance regards the disjoint LinUCB algorithm and the latter the hybrid LinUCB algorithm.

Although there is no theoretical analysis, based on the authors' experiments, LinUCB has many advantages when data are not dense, and it outperforms from other equivalent methods related to the contextual bandit problem. It also has been tested by Pavlidis et al. [30], but no theoretical analysis has been carried out, too. Due to that lack of analysis, Chu et al. [11] decomposed LinUCB into two algorithms and offered their aspect of analysis for both: the *BaseLinUCB* and the *SupLinUCB*, which uses BaseLinUCB as a subroutine and has an expected regret of $O(\sqrt{Tdln^3(KTln(T)/d)})$.

5 EXPERIMENTS IN RANKED RECOMMENDATIONS

So far, in the previous chapters we have discussed plenty of algorithms belonging to different categories of online learning algorithms. However, the one that we are interested in is the RBA meta-algorithm, which uses instances of *k* multi-armed bandits algorithms in order to provide a user with ranked recommendations. As RBA recommends more appropriate rankings concerning the preferences of users, then a user would click at least on one document of the list provided and, hence, we could achieve a greater amount of income. Especially, we will make experiments on UCB1, e-Greedy and Exp3 instances into RBA, which are the most studied bandits algorithms in the literature.

According to the scenario of our experiments, we consider a website that provides users with a list of recommended documents and the users in turn can click on one or more documents, or none. Each document has a specific cost for a user, presupposing that it is chosen by that user. We aim to examine the "*sponsored recommendations*" through a diversified set of users, who derive from different populations.

For this scenario we assume that each user arrives in order. That is to say, they visit the website one by one and they never coincide. The more the users visiting the website, the more documents are to be clicked and, as a result, the more earnings the website receives daily. The goal is to select the most appropriate list of documents, which would be the real motor for attracting a lot more users.

In this setting, we are going to calculate the earnings of a website during a whole year and compare the results to the standard greedy algorithm for the maximum coverage problem, which chooses the optimum set of *n* documents. Basically, we will examine the convergence of RBA to that greedy algorithm. There will be three different such comparisons, as the choices of the recommendation list are to be made by RBA-UCB1, RBA-Exp3, and RBA- e-Greedy. In addition, we underline that RBA allows just one click from a user.

The setting described above and the experiments on it can be generalized and adopted by other similar scenarios related to the clicking behavior of a set of users and the acquired profit. For example, an advertising agency that provides customers with a list of recommended advertisements and hopes that it would be profitable.

5.1 Methodology

For the implementation of the greedy algorithm for the max-*k*-cover problem, we firstly suppose that each document of the list is associated with a weight (the income for the website or even the cost for a user). So, we choose that value of each document from the uniform distribution *U* (0, 1). Moreover, each user is formed by a subset of the documents he is interested in. For the given values in each case, we refer the reader to the Appendix.

Under such circumstances, we consider that each user scans a list of *k* recommended documents from slot 1 to slot *k* and he clicks on the first document he is interested in. The generalization of the greedy algorithm selects:

- ✓ for slot 1, the document that maximizes total income from customers who are interested in that document,
- ✓ for slot 2, the document that maximizes total income from customers who are interested in that document, but not in the document of slot 1,
- ✓ for slot 3, the document that maximizes total income from customers who are interested in that document, but not in the documents of slot 1 and slot 2, etc.

So, the outcome is a set of *k* slots that constitutes the final recommended list of a website and optimizes its total income in accordance consistently with the preferences of users.

As far as the implementation of the RBA algorithm is concerned, we used *k* instances of the three mentioned online algorithms. For each one, we uphold an instance per slot. If a user clicks on a slot, the instance of the online algorithm for that slot must update its weights. Essentially, the algorithm converges to the *k* most popular documents, i.e.:

- ✓ to the first popular document for slot 1,
- ✓ to the second popular document among users who are not interested in the document of slot 1,
- ✓ to the third popular document among users who are not interested in the documents of slot 1 and slot 2, and so on and so forth.

As a result, we create a set of *k* slots, which yields a total income, as each slot has its own cost.

In addition, we begin the experiments considering a set of *d* random users, who derive from *p* different populations. Afterwards, we assume that at each time step (each day) there is an addition of a new user to the initial set, so that at time step *d* we have exactly *d* customers. This new user is also chosen randomly from one of the available populations.

The total time steps that both algorithms run are 365 – d + 1, as we want to examine the total annual income of the website through the usage of different online algorithms as instances in RBA and then compare the difference of that income to the outcome of the greedy algorithm using the average regret. Specifically, we want to calculate the expression:

$$\frac{|Greedy_t - RBA_t|}{t} \quad [1]$$

where:

- ➢ $Greedy_t$ is the total annual income of the greedy algorithm,
- ➢ RBA_t is the total annual income of the RBA algorithm using instances of UCB1, Exp3 or e-Greedy,
- ➢ t is the total number of time steps.

Apart from the computation of the average regret of RBA, we are also going to calculate the relative difference of income for RBA, which also measures the convergence of RBA. For this purpose, we will use the expression:

$$\frac{|Greedy_t - RBA_t|}{Greedy_t} \quad [2]$$

5.2 Experiment Analysis

In this section, we present the specifics of the implementation of the experimental phases, for which we used Python, as programming language. Through the experimental analysis we are in the position of observing the way each used policy by RBA performs under specific circumstances, such as the selection of the value of a parameter (*e* for *e*-Greedy and *γ* for Exp3), and manage to find out which one achieves better results as instances for RBA, after the comparison to the greedy algorithm.

Specifically, we ran a first phase of experiments to observe how the UCB1, Exp3, and *e*-Greedy policies perform and find out good values for the parameters of the e-Greedy and Exp3 algorithms in a simulation of 50 arms (documents) during 1,000 plays averaged with a horizon of 500 trials. The second phase of experiments deals with the comparison among RBA and the greedy algorithm in a simulation of *n*=50 documents during a year (*T*=365-20+1=346 time steps), using *k*=5 slots. The initial number of users is *d*=20. (We used the same values for *n*, *k*, and *d* as in [32]).

5.3 First phase of experiments

In this section we are going to decide which is the best option of the parameters used in *e*-Greedy and Exp3. We want to manage the best performance of the algorithms, that are going to be used in the next phase.

To begin with *e*-Greedy and its parameter *e*, we observe that as *e* increases, the exploration is also increased and, hence, the chance of selecting arms randomly, instead of choosing the best one. So, the cumulative reward in Figure 5.1 decreases as the result of not choosing the best option.

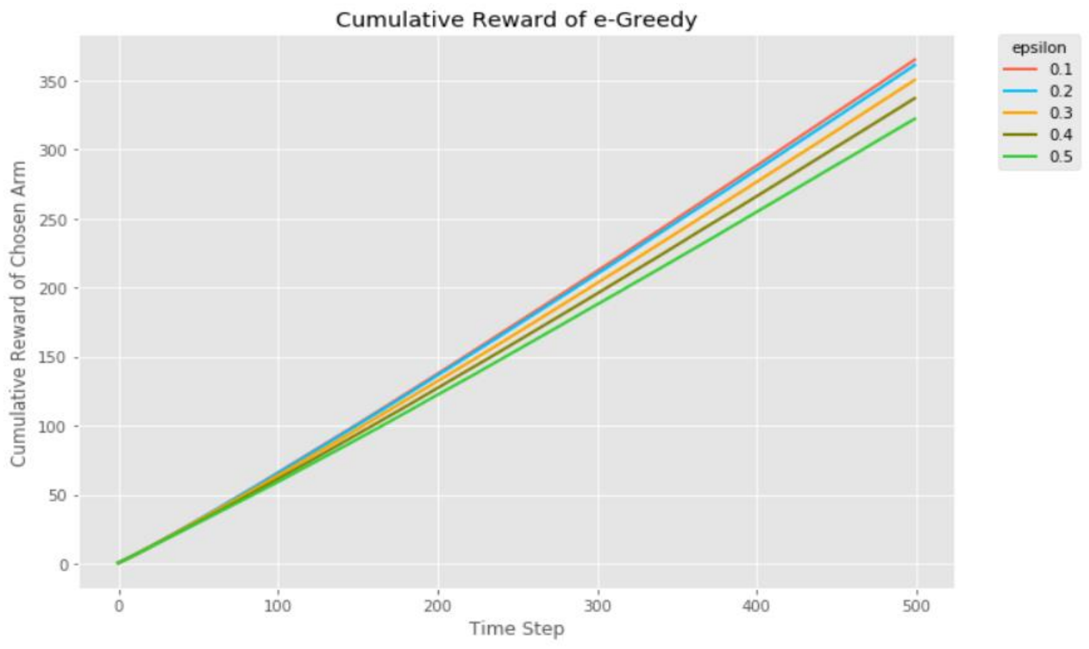


Figure 5.1: Cumulative reward of the e-Greedy algorithm

Moreover, as it is obvious from Figure 5.1, the algorithm with *e*=0.1 outperforms the rest; however, cumulative rewards metric shows that it takes that algorithm quite long time to outperform the algorithm with *e*=0.2.

So, we conclude that we should choose 0.1 as the value of *e*, because in this setting *e*-Greedy should get more aggressive strategy and be more competitive to the other algorithms.

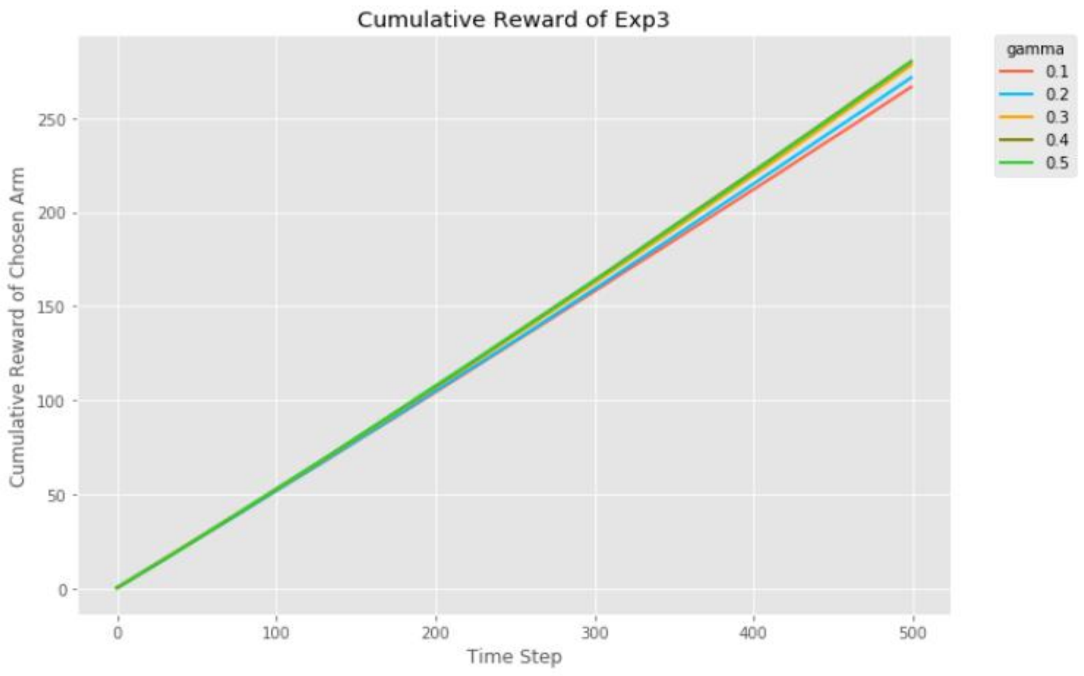


Figure 5.2: Cumulative reward of the Exp3 algorithm

The second algorithm's parameter we want to examine is *γ* of Exp3. This parameter handles the tradeoff between exploration and exploitation. As it increases, the algorithm explores more; otherwise, the algorithm exploits more.

From Figure 5.2 we observe that all versions firstly tend to be equal, but afterwards, 0.5 value performs the best. So, we choose *γ*=0.5 as the best option for the second phase of experiments.

5.4 Second phase of experiments

In this section we are going to deal with the implementation of the variants of RBA (RBA_UCB1, RBA_Exp3 and RBA_*e*-Greedy) and the greedy algorithm. Through the provided plots, we are going to analyze the algorithms and compare their performance.

First of all, we should implement for RBA the 5 instances of UCB1, Exp3 and *e*-Greedy. All the exported plots are included in Appendix and depict the frequency of the chosen documents per instance of each algorithm and as well as the average regret of each instance per algorithm.

However, in this section, we use just the plots of the three first instances of the UCB1 algorithm, in order to mention a basic operation of RBA: in the case of an instance that determines a document to be shown in a slot that already exists in a higher one, the algorithm selects another document for that slot arbitrarily. This is what exactly happened with the third instance, where document No24 was selected, but it was already chosen by the first instance. So, RBA is responsible for its replacement.

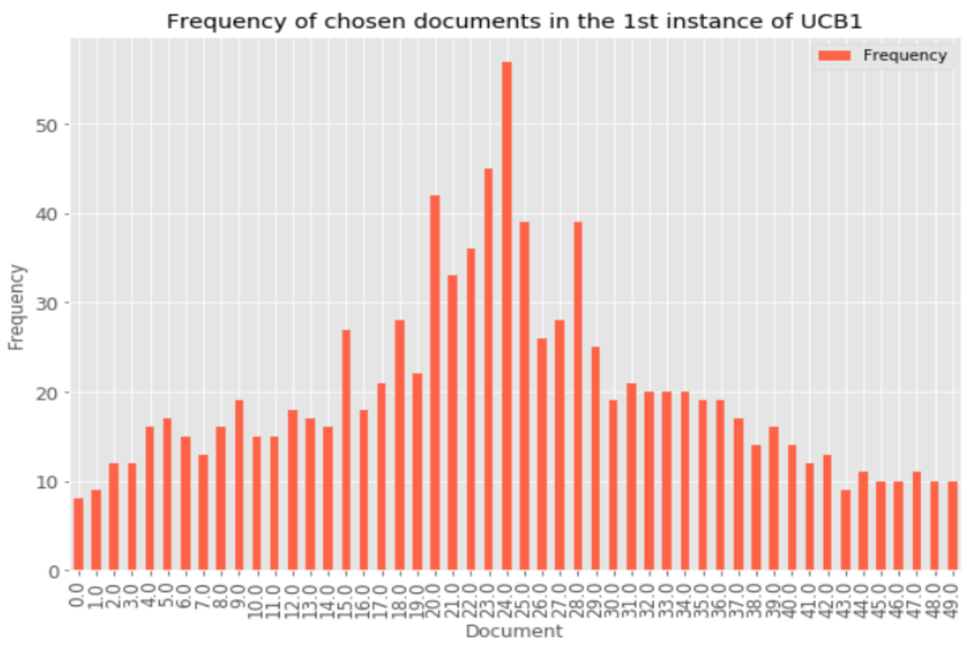


Figure 5.3: Frequency of chosen documents in the first instance of UCB1

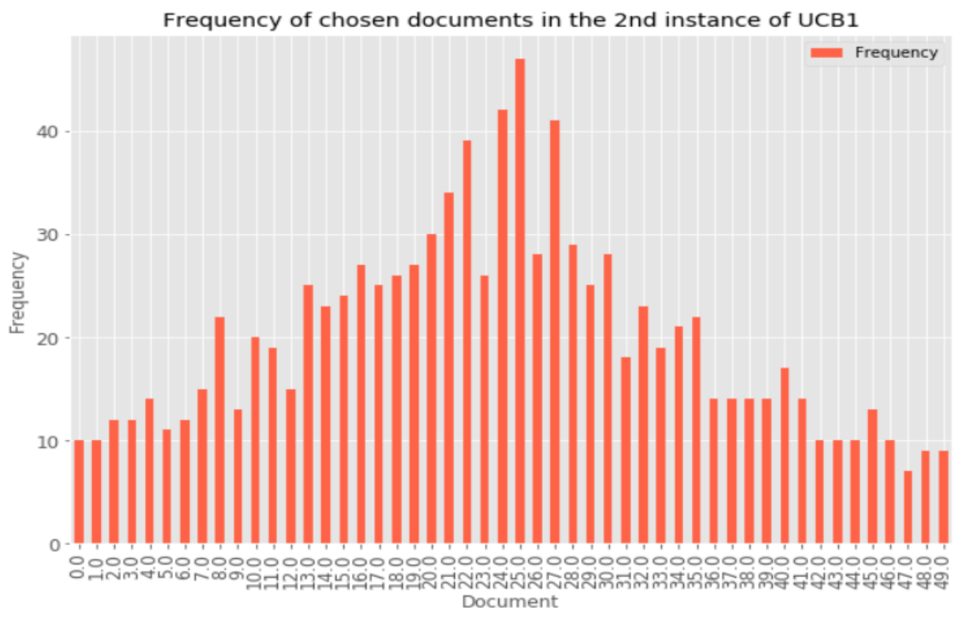


Figure 5.4: Frequency of chosen documents in the second instance of UCB1

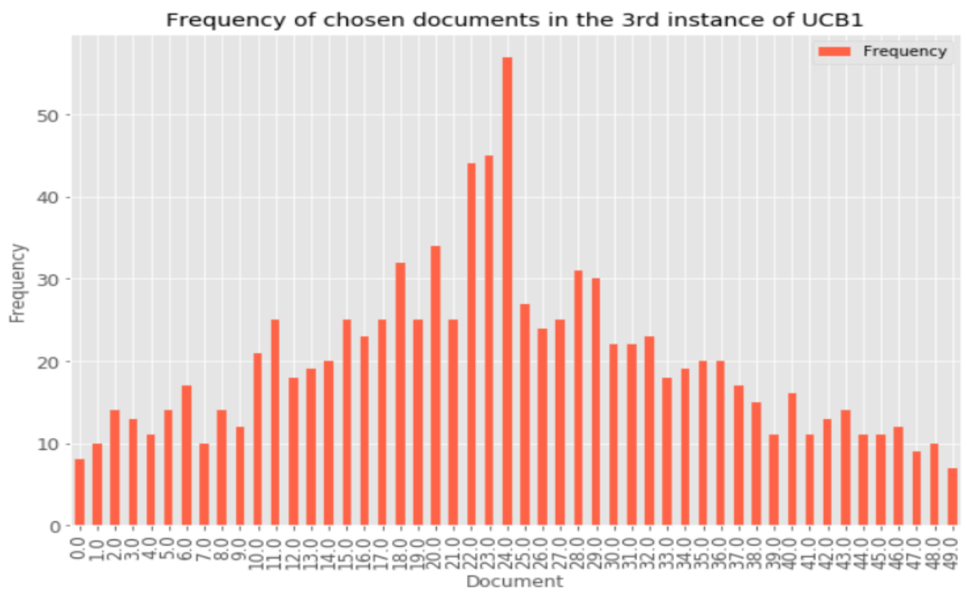


Figure 5.5: Frequency of chosen documents in the third instance of UCB1

Consequently, we implement the next phase of RBA. The algorithm provides the 5 documents as a recommended list to the set of users, who click to one or none of them, and updates the rewards of the clicked documents of the specific multi-armed bandit algorithm instance that determined those documents. At the same time, we implement the greedy algorithm and we receive the results, as shown in Figure 5.6.

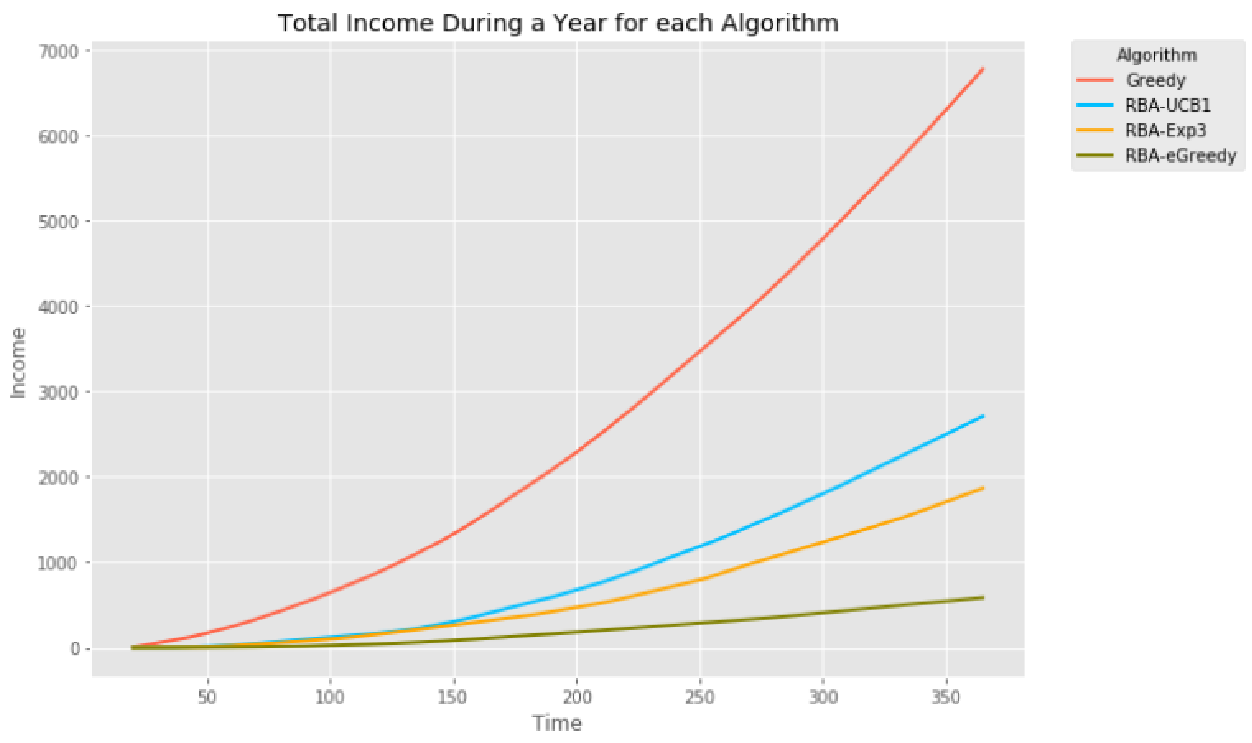


Figure 5.6: Total annual income of different algorithms

Figure 5.6 illustrates the curves for the total income during a year for the greedy algorithm and the meta-algorithm RBA using instances from the UCB1, Exp3 and e-Greedy online algorithms.

As it is expected, the greedy algorithm outperforms RBA in all its permutations, as it is an algorithm that returns the optimal solution. The distance between RBA's curves increases in a stable way as the days go by. Moreover, if we exclusively observe the behavior of RBA, we can distinguish that RBA-UCB1 returns a lot more earnings than RBA- *e*-Greedy and RBA-Exp3. As a result, we expect that the absolute difference between greedy and both RBA-Exp3 and RBA- *e*-Greedy would be greater than that of RBA-UCB1.

In that point, we should mention that the interests of users remain static through the year, as we use the same users per day, with an addition of a new user, one on each day. Considering also that the performance of RBA that used instances of UCB1 was better than that which used Exp3 as instances, we come to an agreement with the authors of [32], who concluded in an equivalent result.

In addition, we unfortunately observe that RBA-UCB1 (which is better than the other versions of RBA) abstains quite a lot from converging to greedy. The optimal scenario would be if those two curves coincide. Although till 50th day all curves of RBA are quite close to greedy's, we can see that after that point their difference more and more increases exponentially.

Figure 5.7 depicts the difference between the optimal cumulative income of the greedy algorithm and the cumulative income of RBA. RBA-UCB1 is unsurprising to have the lower difference, as it has the best performance.

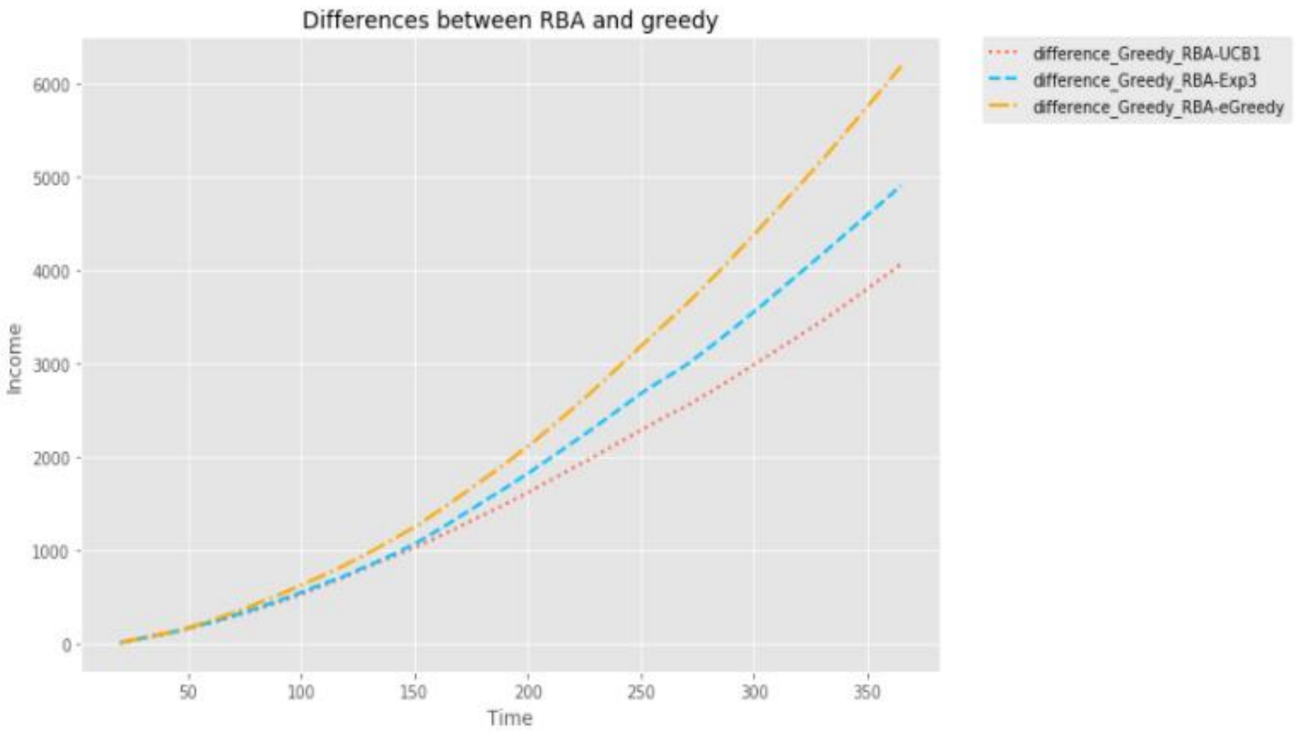


Figure 5.7: Differences between RBA and the greedy algorithm

Finally, we calculated the average regret of RBA using the expression [1], as shown in Figure 5.8. Generally, the greater the tendency, the less efficient RBA is. As mentioned, RBA-UCB1 has the smallest difference from greedy, so we expect that its rate to be below the other rates.

We observe that all these rates are almost always linear through time steps. The rate of RBA- *e*-Greedy remains quite stable through the year, while RBA-Exp3 and RBA-UCB1 follow a similar path until time step 150. As for the rate of RBA-UCB1, if we look through the diagram, we observe that it firstly follows by a small difference the rates of the other two, but, from time step 150 and then, it diverges from them and accomplishes better performance and greater income.

To conclude, RBA-UCB1 is the most suitable version of RBA in our experiments, as it achieves the best convergence compared to the greedy algorithm.

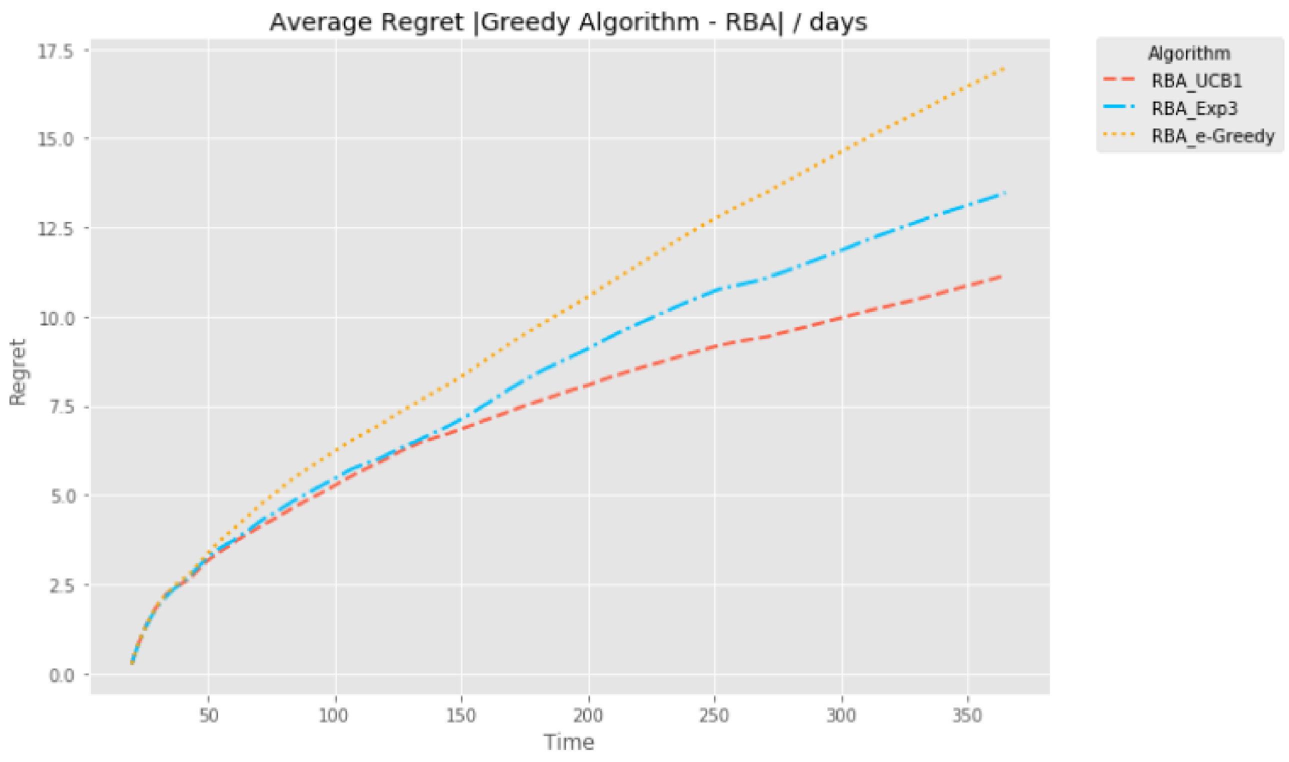


Figure 5.8: Average regret of each version of RBA

However, we expected the average regret of RBA to tend towards elimination at some time. So, we repeated the experiments for much more time steps (specifically for 5 years), in order to observe whether the average regret of RBA tends to become zero or not. The results are shown in Figure 5.9 and Figure 5.10.

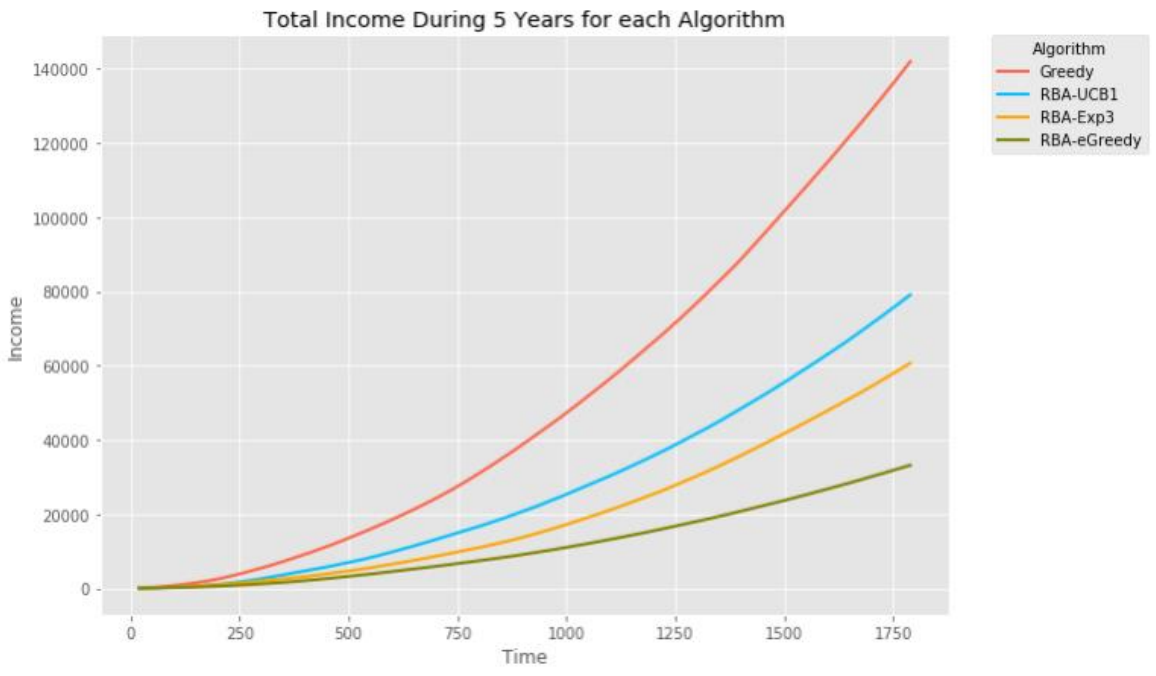


Figure 5.9: Total income of different algorithms for 5 years

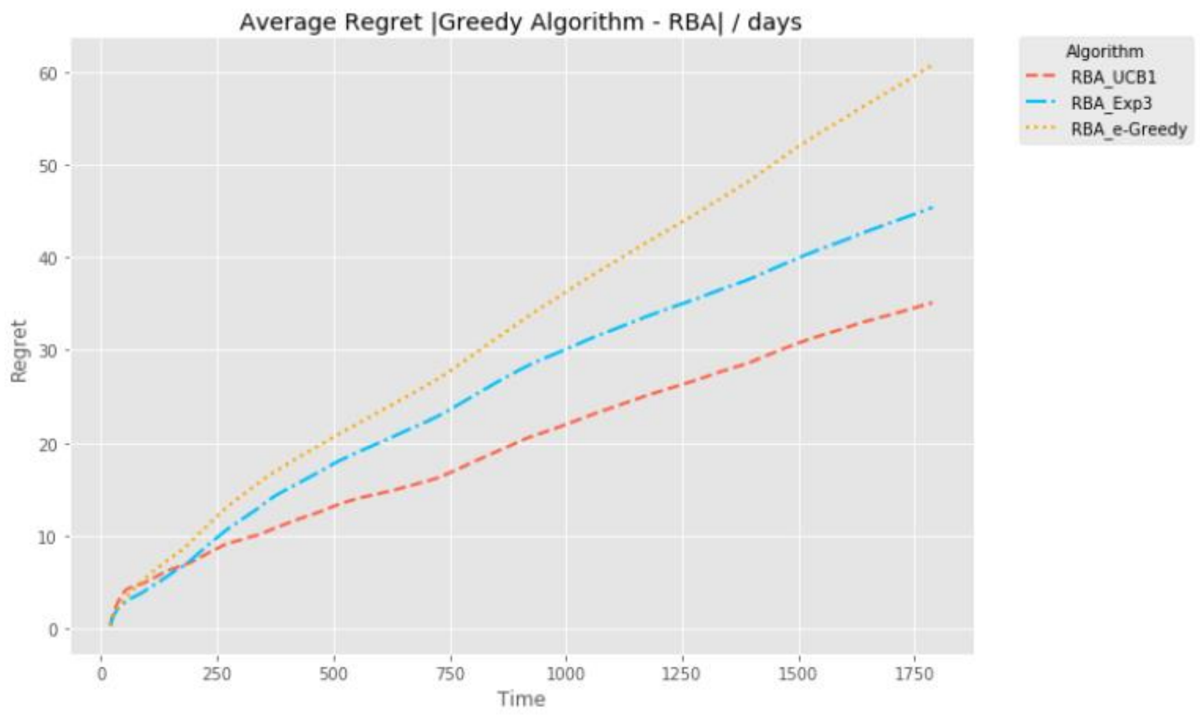


Figure 5.10: Average regret of each version of RBA

As we observe, the results are not what we exactly expected, as the rates are getting more and more greater over time steps. So, we tried to observe the performance of RBA through the relative difference of income, using the expression [2]. Figure 5.11 and Figure 5.12 depict those rates during one and five years, respectively.

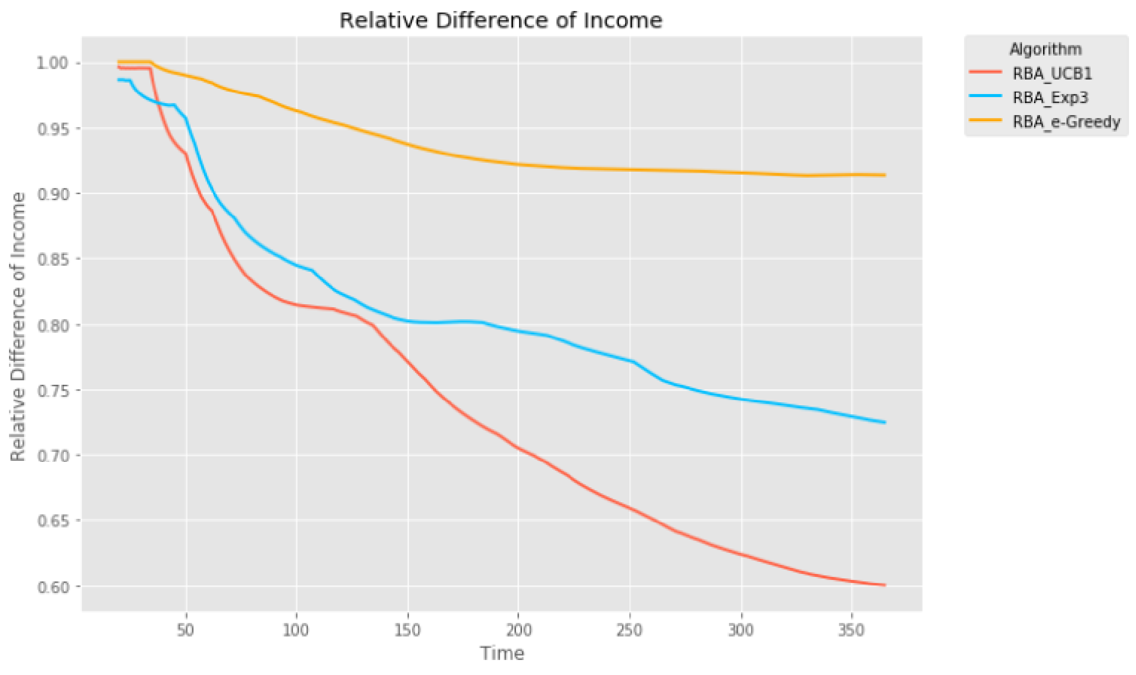


Figure 5.11: Relative difference of income of each version of RBA for 1 year

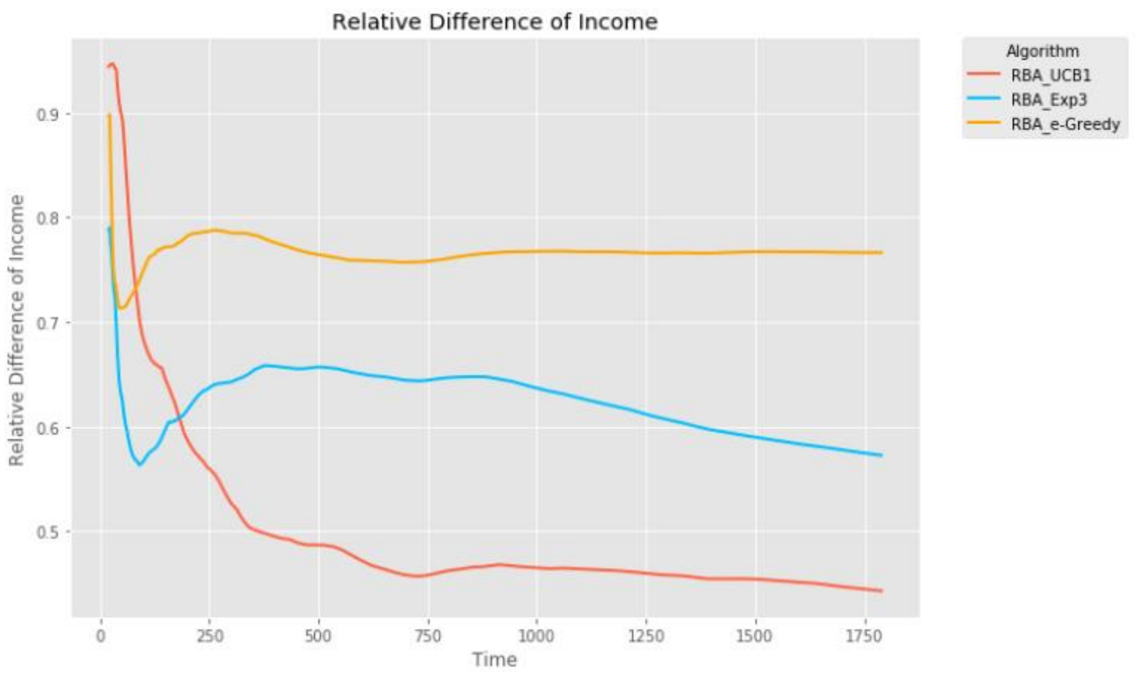


Figure 5.12: Relative difference of income of each version of RBA for 5 years

Through Figure 5.11 and Figure 5.12, it is obvious that RBA-UCB1 tends to zero over time steps, which means that not only has the best performance compared to the other versions of RBA, but also it converges to greedy.

5.5 Conclusion

Throughout this report we tried to present the Online Learning area, providing its theoretical foundations and the needs that provoked its research, and we focused on its applications in the ranked recommendations. We introduced the reader to the two fundamental problem models of online learning, the Prediction with Expert Advice problem, and the Multi-armed Bandit problem. We analyzed the latter one and some of the algorithms that cover the problem. As it is natural, we illustrated only a small number of Multi-armed Bandit policies that we considered to be the most representative of their category. Among others, we studied the problem of rankings of “sponsored recommendations” and we used the Ranked Bandits Algorithm with UCB1, Exp3 and *e*-Greedy instances in several experiments, trying to observe its rate of convergence based on the greedy algorithm for the max-*k*-cover problem.

In addition, in order to achieve diversity in our results, we simulated all the experiments on a diversified set of users, and, hence, we created different populations in which there are a lot more users with different interests and clicking behavior. We compared the performance of the three permutations of RBA and our results showed that RBA-UCB1 outperforms the others. Finally, it was clear that the average regret of

RBA-UCB1 is the lowest, but its convergence is quite far from the optimal strategy. However, through the relative difference, we observed that RBA-UCB1 indeed converges to greedy over time steps.

Since the way that we define our model for computing the average regret of RBA can only be limited in our creativity, we are bound to implement many more scenarios in “sponsored recommendations” in the future. For instance, one could consider more populations of users, or even more documents to begin with the experiments. What is more, one could use the meta-algorithm IBA with UCB1, Exp3 and *e*-Greedy instances and compare its performance to RBA with the respective multi-armed bandit instances.

6 REFERENCES

[1] N. Abe, A. W. Biermann, P. M. Long, Reinforcement Learning with Immediate Rewards and Linear Hypotheses. Algorithmica 37(4): 263-293 (2003).

[2] J. Abernethy, E. Hazan, A. Rakhlin. Competing in the dark: An efficient algorithm for bandit linear optimization. In Proceedings of the 21st Annual Conference on Learning Theory (COLT) 2008.

[3] R. Agrawal. Sample Mean Based Index Policies with O (log n) Regret for the Multi-Armed Bandit Problem. Advances in Applied Probability 27(4): 1054-1078 (1995).

[4] P. Auer. Using Confidence Bounds for Exploitation-Exploration Trade-offs. Journal of Machine Learning Research 3: 397-422 (2002).

[5] P. Auer, N. Cesa-Bianchi, P. Fischer. Finite-time Analysis of the Multi-armed Bandit Problem. Machine Learning Journal 47(2-3): 235-256 (2002).

[6] P. Auer, N. Cesa-Bianchi, Y. Freund, R. E. Schapire. The Nonstochastic Multiarmed Bandit Problem. SIAM J. Comput. 32(1): 48-77 (2002).

[7] S. Bubeck, A. Slivkins. The best of both worlds: stochastic and adversarial bandits. Journal of Machine Learning Research 23: 42.1-42.23 (2012).

[8] N. Cesa-Bianchi, P. Fischer. Finite-time Regret Bounds for the Multiarmed Bandit Problem. In Proceedings of the 15th International Conference on Machine Learning (ICML) 1998: 100–108.

[9] N. Cesa-Bianchi, G. Lugosi. Combinatorial bandits. In Proceedings of the 22nd Annual Conference on Learning Theory (COLT) 2009.

[10] F. Chierichetti, R. Kumar, A. Tomkins. Max-cover in map-reduce. In Proceeding of the 19th International Conference on World Wide Web (WWW) 2010: 231–240.

[11] W. Chu, L. Li, L. Reyzin, R. E. Schapire. Contextual Bandits with Linear Payoff Functions. Journal of Machine Learning Research 15: 208-214 (2011).

[12] N. Craswell, O. Zoeter, M. Taylor, B. Ramsey. An experimental comparison of click position bias models. In Proceedings of the 1st ACM International Conference on Web Search and Data Mining (WSDM) 2008: 87-94.

[13] Y. Freund, R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. System Sci. 55: 119–139 (1997).

[14] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. Games Econom. Behav. 29: 79–103 (1999).

[15] A. Garivier, O. Cappe. The KL-UCB algorithm for bounded stochastic bandits and beyond. In Proceeding of the 24th Annual Conference on Learning Theory (COLT) 2011: 359–376.

[16] J. C. Gittins. Bandit Process and Dynamic. Journal of the Royal Statistical Society 41(2): 148-177 (1979).

[17] J. C. Gittins. Multi-armed Bandit Allocation Indices. John Wiley & Sons, 1989.

[18] F. Guo, C. Liu, Y. M. Wang. Efficient Multiple-Click Models in Web Search. In Proceedings of the 2nd ACM International Conference on Web Search and Data Mining (WSDM) 2009: 124–131.

[19] A. Gyorgy, T. Linder, G. Lugosi, G. Ottucsak. The on-line shortest path problem under partial monitoring. Journal of Machine Learning Research 8: 2369-2403 (2007).

[20] T. Ishikida, P. Varaiya. Multi-armed bandit problem revisited. Journal of Optimization Theory and Applications 83(1): 113-154 (1994).

[21] S. Katariya, B. Kveton, C. Szepesvari, Z. Wen. DCM Bandits: Learning to Rank with Multiple Clicks. In Proceedings of the 33rd International Conference on Machine Learning (ICML) 2016: 1215-1224.

[22] P. Kohli, M. Salek, G. Stoddard, A Fast Bandit Algorithm for Recommendation to Users with Heterogenous Tastes. In Proceedings of the 27th Association for the Advancement of Artificial Intelligence (AAAI) 2013: 1135-1141.

[23] T. Lai, H. Robbins. Asymptotically Efficient Adaptive Allocation Rules. Advances in Applied Mathematics 6: 4-22 (1985).

[24] J. Langford, T. Zhang. The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information. In Proceedings of the Neural Information Processing Systems (NIPS) 2007: 817-824.

[25] L. Li, W. Chu, J. Langford, R. E. Shapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In Proceeding of the International World Wide Web Conference (WWW) 2010: 661-670.

[26] N. Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. Machine Learning 2: 285-318 (1988).

[27] N. Littlestone, M. Warmuth, The Weighted Majority Algorithm. Inform. and Comput. 108: 212-261 (1994).

[28] M. Mohri, A. Rostamizadeh, A. Talwalkar. Foundations of Machine Learning. MIT Press, 2018.

[29] S. Pandey, D. Agarwal, D. Chakrabarti, V. Josifovski. Bandits for Taxonomies: A Model-based Approach. In Proceedings of the 8th SIAM International Conference on Data Mining (SDM) 2007.

[30] N. G. Pavlidis, D. K. Tasoulis, D. J. Hand. Simulation studies of multi-armed bandits with covariates. In Proceeding of the 10th International Conference on Computer Modeling, Simulation and Algorithm (CMSA) 2008: 493-498.

[31] F. Radlinski, R. Kleinberg, T. Joachims, Learning diverse rankings with multi-armed bandits. International Conference on Machine Learning (ICML) 2008: 784-791.

[32] H. Robbins, Some Aspects of the Sequential Design of Experiments. Bulletin of the American Mathematical Society 58(5): 527-535 (1952).

[33] Y. Seldin, N. Cesa-Bianchi, P. Auer, F. Laviolette, J. ShaweTaylor. PAC-Bayes-Bernstein inequality for martingales and its application to multiarmed bandits. Journal of Machine Learning Research 26: 98-111 (2012).

[34] Y. Seldin, C. Szepesvari, P. Auer, Y. Abbasi-Yadkori. Evaluation and Analysis of the Performance of the EXP3 Algorithm in Stochastic Environments. Journal of Machine Learning Research 24: 103-116 (2012).

[35] A. L. Strehl, C. Mesterharm, M. L. Littman, H. Hirsh. Experience-efficient learning in associative bandit problems. In Proceedings of the 23rd International Conference on Machine Learning (ICML) 2006.

[36] R. Sutton, A. Barto. Reinforcement Learning: An Introduction, MIT Press. second edition, 2017.

[37] E. Schwartz, E. Bradlow, P. Fader. Customer Acquisition via Display Advertising Using Multi-Armed Bandit Experiments. Marketing Science 36(4): 500 - 522 (2017).

[38] S. S. Villar, J. Bowden, J. Wason. Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges. Statistical Science 30(2): 199–215 (2015).

[39] V. G. Vovk. Aggregating strategies. In Proceedings of the 3rd Annual Workshop on Computational Learning Theory (COLT) 1990: 371–386.

[40] T. J. Walsh, I. Szita, C. Diuk, M. L. Littman. Exploring compact reinforcement-learning representations with linear regression. In Proceeding of the 25th Conference on Uncertainty in Artificial Intelligence (UAI) 2009: 591-598.

[41] C. C. Wang, S. R. Kulkarni, H. V. Poor, Bandit problems With Side Observations. IEEE Transactions on Automatic Control 50(3): 338-355 (2005).

[42] Y. Yue, C. Guestrin, Linear Submodular Bandits and their Application to Diversified Retrieval. In Proceeding of the Neural Information Processing Systems (NIPS) 2011: 2483-2491.

APPENDIX

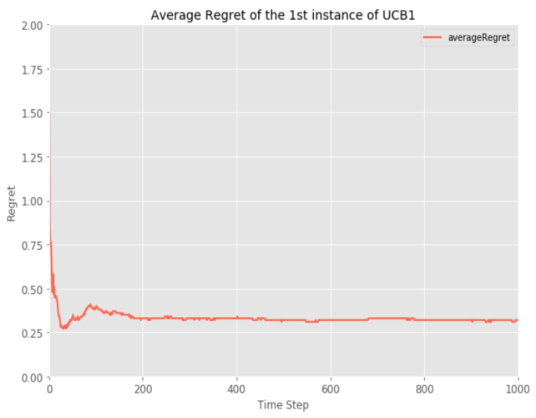
The weights of the documents used in the Greedy algorithm were random samples from a uniform distribution U (0, 1).

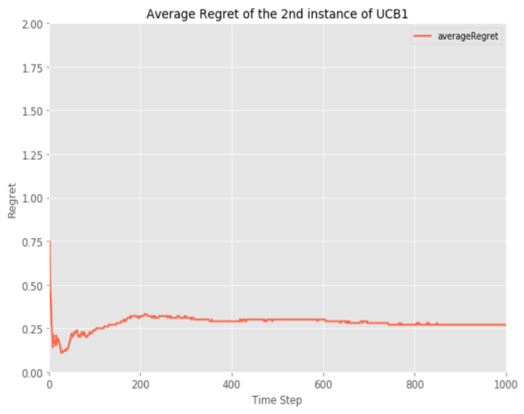
Document	Weight
1	0.85541668
2	0.11724721
3	0.36414238
4	0.29008554
5	0.03431462
6	0.69300833
7	0.29816505
8	0.90947815
9	0.12645037
10	0.17218998
11	0.23170864
12	0.21513629
13	0.91927154
14	0.5082231
15	0.72255419
16	0.0241209
17	0.46877528
18	0.05595119
19	0.88334605
20	0.0381418
21	0.63850356
22	0.19619498
23	0.14251904
24	0.1449802
25	0.62077332
26	0.12801759
27	0.53854302
28	0.03589822
29	0.95463274
30	0.99045034
31	0.33391995
32	0.4121157
33	0.95127818
34	0.80908968
35	0.96567087
36	0.80526034
37	0.85436319
38	0.76925792
39	0.40288344
40	0.8099353
41	0.20299039
42	0.37642978
43	0.48617332
44	0.72305208
45	0.4411356
46	0.64432323
47	0.52519805
48	0.70853792
49	0.5576614
50	0.74360328

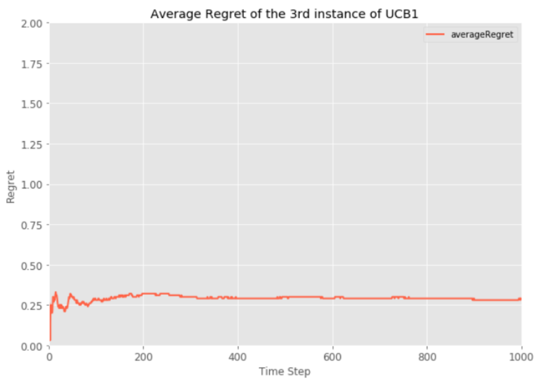
Each user is a subset of his preferences over 50 documents, while the initial 20 users given as input to each algorithm were randomly derived from 3 different populations of 1000 users each.

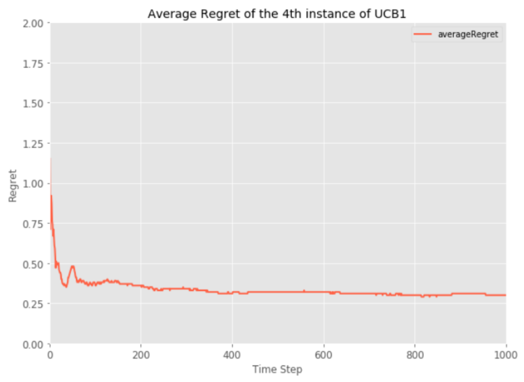
User	Preferences
1	[41, 3, 29, ..., 33, 1, 42]
2	[44, 42, 23, ..., 41, 45, 30]
3	[46, 24, 22, ..., 18, 34, 2]
4	[31, 27, 4, ..., 15, 8, 14]
5	[27, 10, 2, ..., 32, 34, 22]
6	[36, 28, 7, ..., 19, 29, 39]
7	[21, 30, 25, ..., 24, 22, 23]
8	[47, 15, 4, ..., 23, 39, 48]
9	[16, 17, 34, ..., 23, 47, 30]
10	[14, 13, 18, ..., 37, 12, 26]
11	[47, 29, 32, ..., 31, 21, 43]
12	[35, 36, 27, ..., 47, 32, 40]
13	[1, 11, 25, ..., 40, 19, 39]
14	[34, 1, 16, ..., 36, 38, 11]
15	[10, 23, 29, ..., 25, 27, 14]
16	[12, 49, 40, ..., 1, 19, 7]
17	[16, 50, 8, ..., 37, 30, 10]
18	[8, 25, 6, ..., 7, 43, 49]
19	[20, 5, 45, ..., 7, 21, 44]
20	[4, 13, 15, ..., 24, 14, 26]

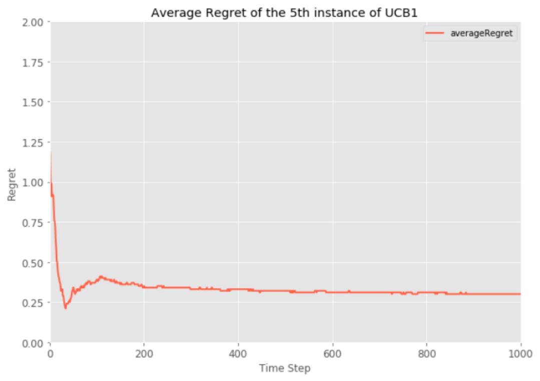
The average regret of each instance of the three different online algorithms used by RBA is illustrated in the next figures.

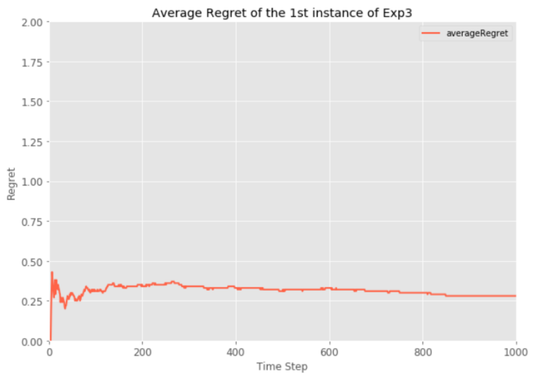


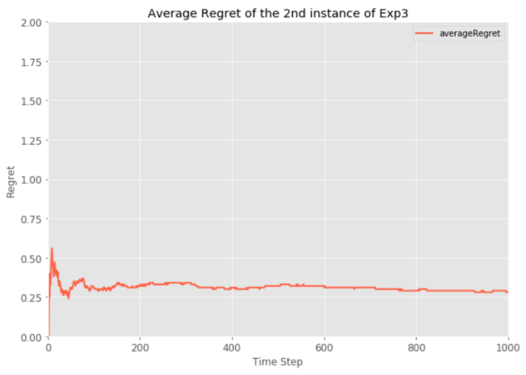


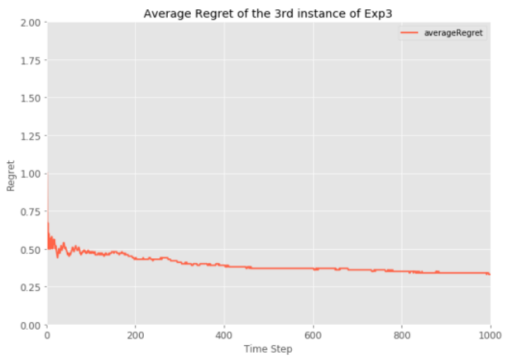


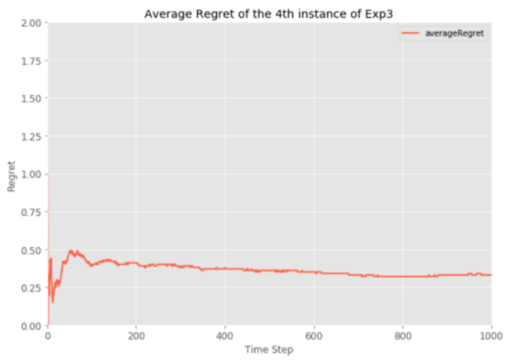


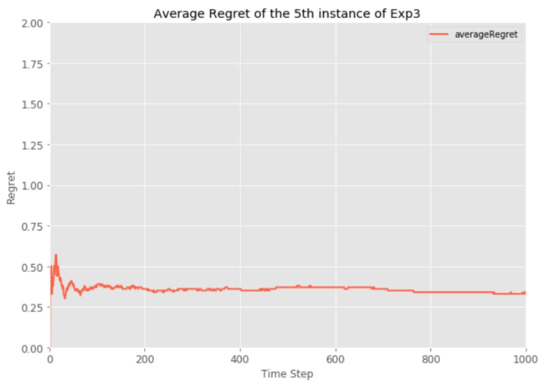


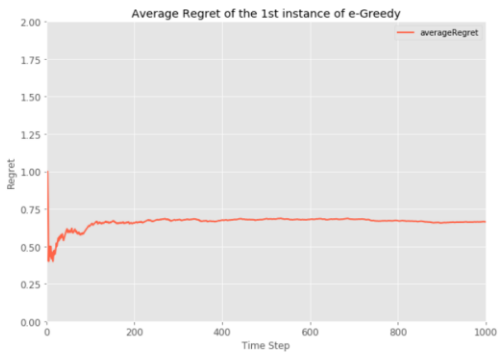


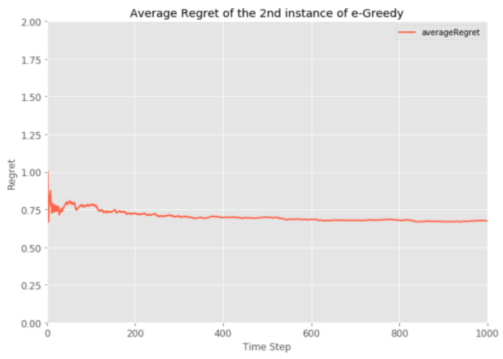


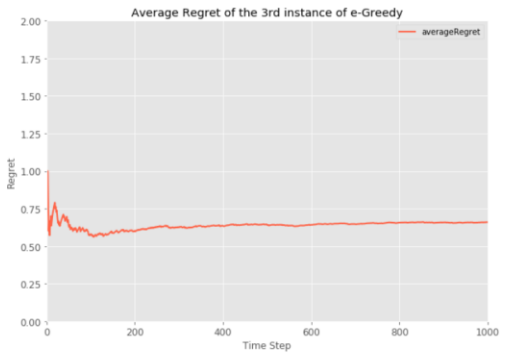

Average Regret of the 3rd instance of Exp3
averageRegret
Regret
Time Step
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
200
400
600
800
1000

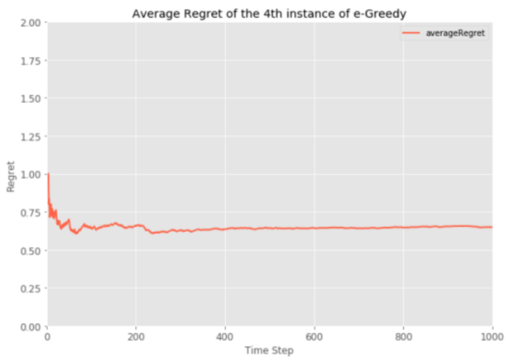

Average Regret of the 4th instance of Exp3
averageRegret
Regret
Time Step

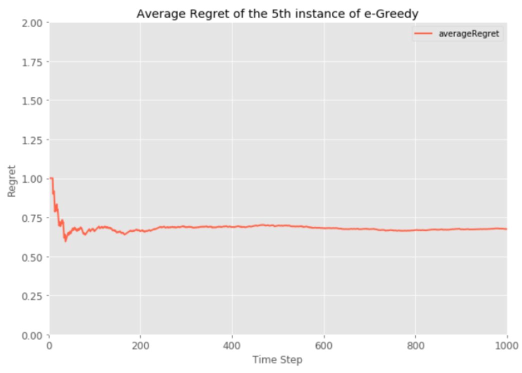

Average Regret of the 5th instance of Exp3
averageRegret
Regret
Time Step


Average Regret of the 1st instance of e-Greedy
averageRegret
Regret
Time Step

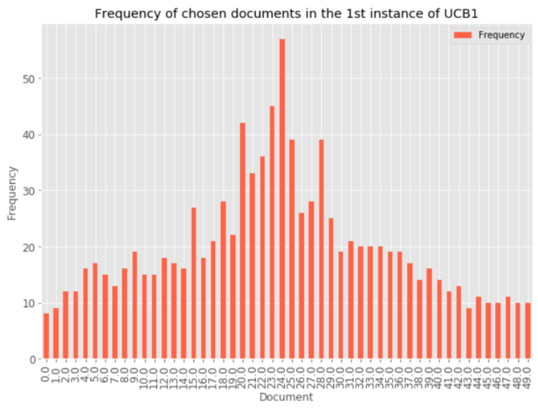

Average Regret of the 2nd instance of e-Greedy
averageRegret
Regret
Time Step

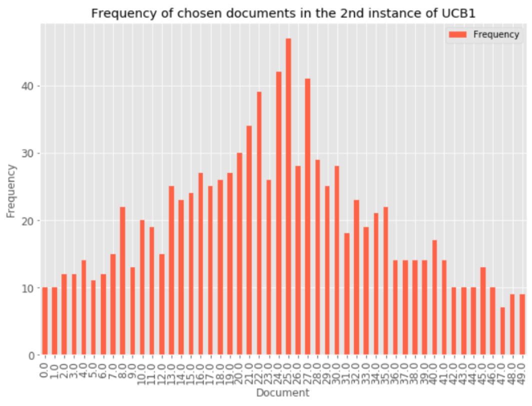

Average Regret of the 3rd instance of e-Greedy
averageRegret
Regret
Time Step

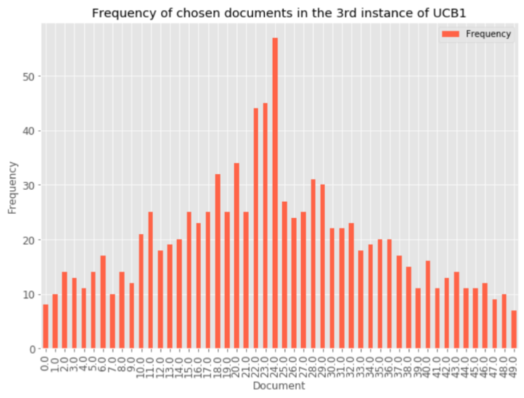

Average Regret of the 4th instance of e-Greedy
averageRegret
Regret
Time Step

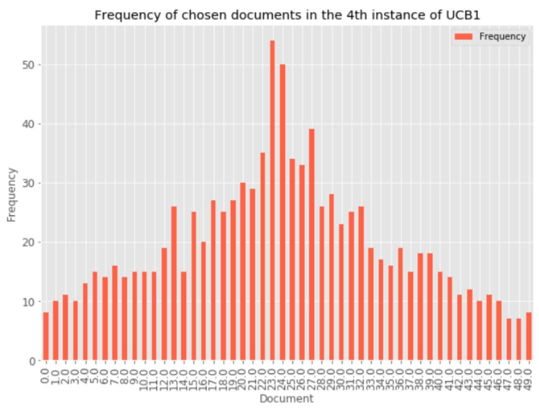


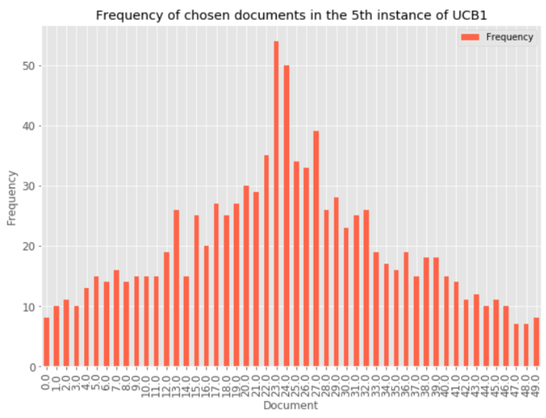
The next figures depict the frequency of the chosen document by the instance of each online algorithm used by RBA.

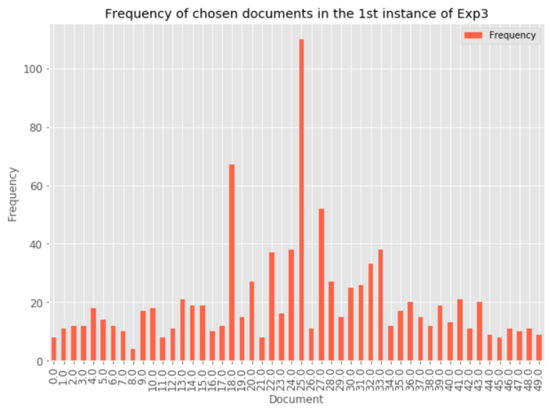
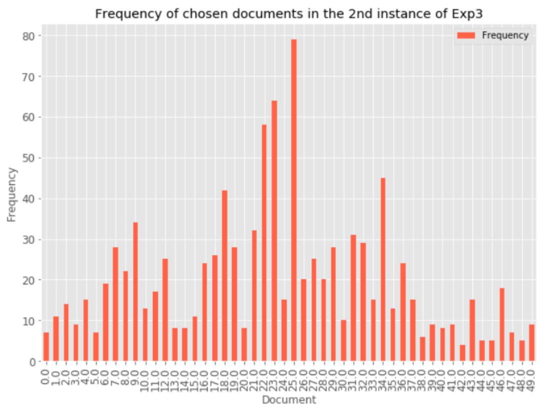
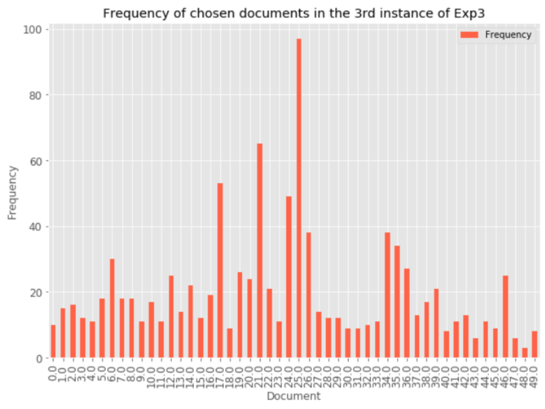
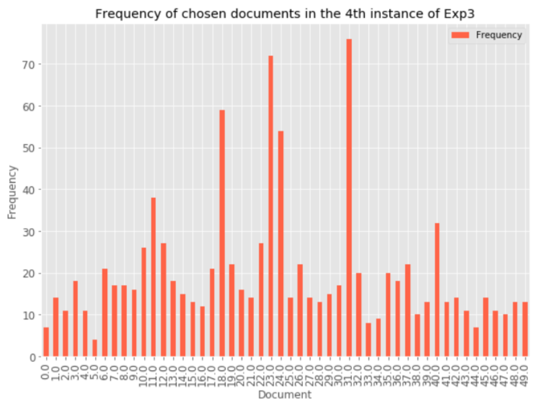
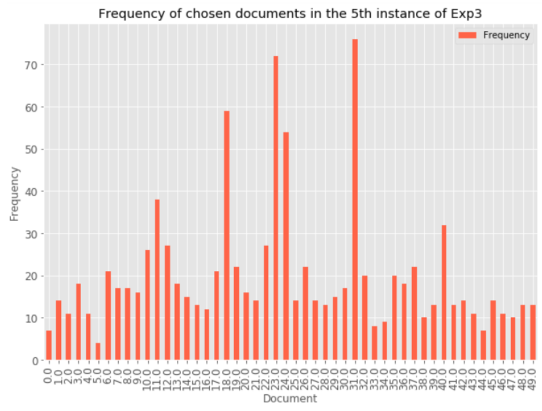
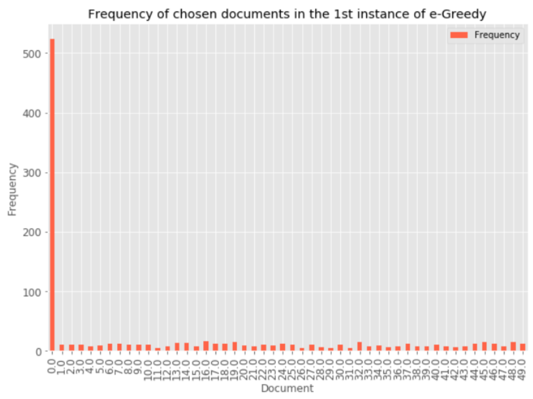
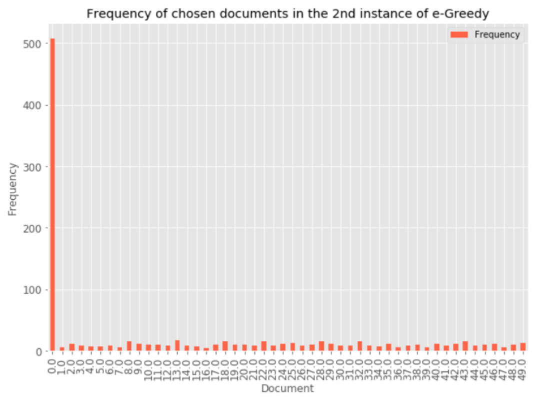


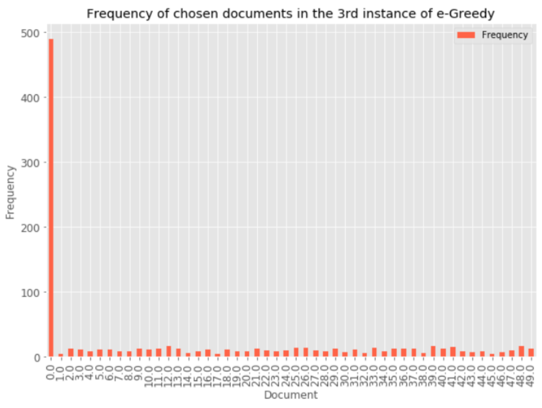


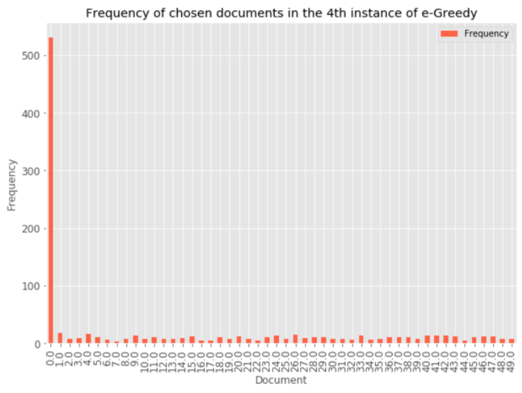


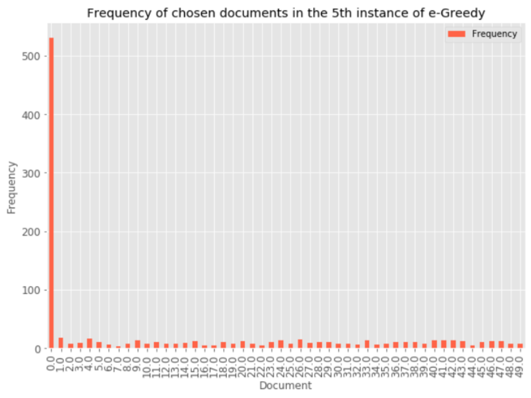





Frequency of chosen documents in the 1st instance of Exp3
Frequency
Document

Frequency of chosen documents in the 2nd instance of Exp3
Frequency
Document

Frequency of chosen documents in the 3rd instance of Exp3
Frequency
Document

Frequency of chosen documents in the 4th instance of Exp3
Frequency
Document

Frequency of chosen documents in the 5th instance of Exp3
Frequency
Document

Frequency of chosen documents in the 1st instance of e-Greedy
Frequency
Document

Frequency of chosen documents in the 2nd instance of e-Greedy
Frequency
Document


Frequency of chosen documents in the 3rd instance of e-Greedy
Frequency
Document
Frequency


Frequency of chosen documents in the 4th instance of e-Greedy
Frequency
Document
Frequency


Frequency of chosen documents in the 5th instance of e-Greedy
Frequency
Document
Frequency