Semi-Bandit feedback: A survey of results

- Karthik Abinav Sankararaman
- University of Maryland College Park

Last Revision: December 5, 2016

Table of Contents

[1]Applications from this section due to a class lecture by Alex Slivkins

Abstract

In this report, we survey the recent results for the Semi-Bandit problems in the stochastic setting. For a motivation of the semi-bandit setting, we describe the first paper which looks at it in the adversarial setting. Other than this, all other results described in this survey look at stochastic setting. A common theme of most algorithms is that almost all of them are based on a UCB (Upper Confidence Bounds) style algorithm. We describe the model, the algorithm and main upper and lower bounds (as appropriate) along with some discussion as appropriate. For full proofs, the reader is encouraged to refer to the original papers.

Acknowledgments

The author would like to thank Alex Slivkins for valuable discussions on the general topic of Bandits. These discussions led to clarity in the understanding of many of these works as well as the connections to other works in Bandit problems. The author would like to thank Aravind Srinivasan for giving feedback on the writing and organization of this survey.

1 From Bandits to Semi-Bandits

In this section, we first describe the Bandit problem. We then motivate the semi-bandit as a natural mid-ground between Bandit and learning with expert advice problems.

1.1 Bandit Problems

Imagine you visit a casino with a number of slot machines. For simplicity, we will call these machines $M_1, M_2, \ldots, M_m$. You have a finite number T tries in total (For example, this could be because of limited budget). In each try, you are allowed to pull one slot machine. In return, the slot machine gives you a reward(say a candy). Your goal would naturally be to pull slot machines in an order, so as to maximize your potential total reward across T tries. This setting is usually captured under the notion of Bandit problems which we formally describe below in 1.1. In general, each of the machine is called an arm and hence, the problem is commonly referred to as multi-arm bandit problem.

Definition 1.1 (Multi-arm Bandit problem)**.** We are given a set of arms $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. The algorithm has T rounds. In each round i, the algorithm is allowed to choose one of the m arms. Associated with the m arms is a reward function $r : \mathcal{M} \to \mathbb{R}$. On choosing one of the arms M_i, the reward corresponding to the machine $r(M_i)$ is revealed. Let i^* denote the best possible arm for round i. Define the regret function to be

$$\mathcal{R}(T) = \sum_{t=1}^{T} r(M_t) - \sum_{t=1}^{T} r(M_{t^*})$$

The goal of Multi-arm Bandit problem is to minimize the regret $\mathcal{R}(\mathcal{T})$.

A closely related and well-studied problem is the notion of Contextual Bandits. In this setting, in each round, the algorithm is presented with a d- dimensional feature vector known as the context. Based on the context vector and the rewards seen in the past, the algorithm needs to choose which arm to play in the current round. Formally it is defined as described in 1.2. To make the problem concrete, we define policy Π to be a function from the set $\mathcal{X}$ to the set $\mathcal{M}$. In other words, given a context, the policy functions maps it to an arm.

Definition 1.2 (Contextual Multi-arm Bandit Problem)**.** We are given a set of arms $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, a context set $\mathcal{X}$ and a set of policies Π. The algorithm has T rounds. In each round i, the algorithm is allowed to choose one of the m arms on seeing a context $x \in \mathcal{X}$. Associated with the m arms is a reward function $r : \mathcal{M} \to \mathbb{R}$. On choosing one of the arms M_i, the reward corresponding to the machine $r(M_i)$ is revealed. For a given policy Π, define the regret function to be

$$\mathcal{R}(T, \pi) = \sum_{t=1}^{T} r(M_t) - \sum_{t=1}^{T} r(\pi(x_t))$$

Hence, the regret of the algorithm would be defined as

$$\mathcal{R}(T) = \sup_{\pi \in \Pi} \left[\sum_{t=1}^{T} r(M_t) - \sum_{t=1}^{T} r(\pi(x_t)) \right]$$

The goal of Contextual Multi-arm Bandit problem is to minimize the regret $\mathcal{R}(\mathcal{T})$.

The literature on Multi-arm Bandits in both the non-contextual and contextual setting has been extensively studied. The reader is encouraged to refer to [19] for an extensive survey on the results.

Hence, on one hand we have the learning with expert advice framework, where after every move of the online algorithm, the losses/rewards for every expert is shown. On the other hand, in the Bandit setting after every move exactly one arm/expert's losses/rewards is shown. However, in many real life scenarios, we have a budget which we can use to query more than one expert, but not all experts. The semi-bandit setting precisely captures this situation. In 1.3, we formally define the problem.

Definition 1.3 (Contextual Semi-Bandit problem)**.** We are given a set of arms $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, a set of costs on the arms $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, a per-round total budget B, a context set $\mathcal{X}$ and a set of policies Π. The algorithm has T rounds. In each round i, the algorithm is allowed to choose any number of the m arms on seeing a context $x \in \mathcal{X}$. The only restriction is that the total cost of the chosen arms should not exceed B. Associated with the m arms is a reward function $r : \mathcal{M} \to \mathbb{R}$. On choosing the arms $\{M_{i_1}, M_{i_2}, \ldots, M_{i_k}\}$, the reward corresponding to the machines $\{r(M_{i_1}), r(M_{i_2}), \ldots, r(M_{i_k})\}$ are revealed. For a given policy Π, define the regret function to be

$$\mathcal{R}(T, \pi) = \sum_{t=1}^{T} r(M_t) - \sum_{t=1}^{T} r(\pi(x_t))$$

Hence, the regret of the algorithm would be defined as

$$\mathcal{R}(T) = \sup_{\pi \in \Pi} \left[\sum_{t=1}^{T} r(M_t) - \sum_{t=1}^{T} r(\pi(x_t)) \right]$$

The goal of Contextual Semi-Bandit problem is to minimize the regret $\mathcal{R}(\mathcal{T})$.

In this survey, we further restrict our attention to stochastic semi-bandits. In stochastic semi-bandits, the reward is chosen i.i.d. from a distribution which may be known or unknown. In the next-section we describe the first work on semi-bandits which is adversarial but a highly motivating example. The sub-sequent sections will focus on stochastic settings. Also, as far as possible, the notations across papers have been kept consistent in this survey. Hence, in most cases, the notation can vary significantly from the ones used in the actual paper.

2 Real World Applications [2]

Bandit problems are motivated from a wide range and variety of real world problems. Even though many of the models are quite simplistic to fully solve the real-world scenario, these models give a very good yardstick to giving efficient solutions to those problems. In this section, we will describe a few applications and see what the parameters of the model are for this setting.

- *Stock Investment:* Assume that each day you have a finite budget of money which you want to use to invest in a few stocks. At the end of that day you see whether each of the stocks went up or down. The reward you obtain is the difference between the stock price at the time of investment and the price at closing. Based on this information, you can update your stocks to invest in the next day. This is a full-information setting, in the sense that feedback for all stocks are available as opposed to the ones invested in. Also, usually stock prices have a particular dynamics in their movement. Hence, here i.i.d. is a simplifying assumption to model this real-world phenomenon.

- *Medical Trials:* Consider the problem of medical trials. We have many drugs to choose from, for a particular health problem. We want to figure out which of those options is best suited for a particular population sample. Here, the set of drugs will form the "arms" of our bandit problem. In each time step, we will choose a particular drug and give this to a random person in the population sample. The "reward" is the outcome of this drug on this person. This problem can be solved using bandit feedback or semi-bandit feedback based on the exact nature of the drugs. Suppose, we are interested in finding the best drug among the options, a bandit feedback is a natural setting. However, in most cases we want to know even if particular combinations among the drugs are the most effective. Hence, this naturally assumes a semi-bandit feedback. In this case, the i.i.d. assumption is a very good approximation of the real-world sample as long as the population sample is random enough.

- *Dynamic Pricing:* Dynamic Pricing problem is defined as follows. Users come to a virtual store to purchase an app. Based on the user meta-data, the system has to suggest a price to this user. If the user finds the price to be reasonable, they buy it. If not, the user leaves the system permanently. For this problem, every price point p forms an arm (Hence, the number of arms can be large

[2]Applications from this section due to a class lecture by Alex Slivkins

or potentially infinite). Every time a user comes to the system, based on the previous history, the system chooses a price dynamically. The reward obtained is 1 if the user chooses to buy and 0 otherwise. Again, bandit feedback for a single chosen arm fits naturally to this application. Assuming that users are coming from a random distribution, the i.i.d. assumption also is not too simplistic in this case.

Besides the three examples above, this problem has a rich landscape of applications. Some of them include in the areas of maximum weighted matching, ad-allocation, news page optimization, wireless networks, game theory, recommendation system, etc.

3 Shortest Path problem under the Semi-Bandit Setting

The problem of Semi-Bandits was first considered by Gyorgy, Linder, Lugosi and Ottucsak [10], in which they considered the Shortest Path problem under the semi-bandit setting. They give regret bounds which are sub-optimal. Later in a parallel work by Uchiya, Nakamura and Kudoi [16] and Kale, Reyzin and Shapire[11] they improve these bounds to give optimal regret bounds.

In this section, we begin with the setting of Gyorgy, *et al.* We are given a *directed acyclic* graph $G = (U, E)$. We are given a source and a destination vertex, s and t respectively. The sequential decision making goal is to find the shortest path between u and v. In each time step $t = 1, 2, \ldots, T$, the algorithm is allowed to choose a path P connecting u to v. The "nature" then reveals the loss corresponding to this path P(denoted by $L_{p,t}$) by giving a loss to each of the edges $e \in P$(denoted by $l_{e,t}$). The loss for the path is defined as

$$L_{p,t} = \sum_{e \in P} l_{e,t}$$

Let $\{I_1, I_2, \ldots, I_T\}$ be the set of paths chosen by the algorithm in rounds $1, 2, \ldots, T$. The *cumulative loss* of the algorithm is then defined as

$$\widehat{L_T} = \sum_{t=1}^{T} L_{I_t}$$

The goal of the algorithm is to minimize the normalized regret over the T rounds defined as follows

$$\mathcal{R}(T) = \frac{1}{T}(\widehat{L_T} - \min_{i \in P} \sum_{t=1}^{T} L_{i,t})$$

Translating this to the language used in the definition of semi-bandits, we have the following. The set of machines $\mathcal{M}$ is the set of paths P.
Additionally, the paper makes the following assumption: Every path between u and v in the graph is of length at most K, for some $K > 0$. This assumption is without loss of generality and the reader is encouraged to refer the original paper for the reduction.

The main result of this paper is stated is as follows. With high probability, the normalized regret $\mathcal{R}(T)$ is upper bounded by $O(K\sqrt{|E| \ln N/T})$, where K is the length of all the paths, $|E|$ is the number of edges in the graph and N is the total number of $u - v$ paths in the graph. The formal description of theorem 3.1.

Theorem 3.1 (Regret Bounds for Shortest Path problem)**.** Consider parameters $\delta \in (0, 1), 0 \leq \gamma < 1/2, 0 < \beta \leq 1$ and $\eta > 0$. Define C as the *covering paths*, which is the set of paths "covering" all the edges. In other words, for every $e \in E$, we have a path $c \in C$ such that $e \in c$. Choose parameters such that, the following additional condition holds: $2\eta K|C| \leq \gamma$. Then, with probability at least $1 - \delta$ we have the following:

$$\mathcal{R}(T) \leq K\gamma + 2\eta K^2|C| + \frac{K}{T\beta} \ln\left(\frac{|E|}{\delta}\right) + \frac{\ln N}{T\eta} + |E|\beta$$

The right hand side is minimized by setting the following values: $\eta = \sqrt{\frac{\ln N}{4TK^2|C|}}$, $\gamma = 2\eta K|C|$, $\beta = \sqrt{\frac{K}{T|E|} \ln\left(\frac{|E|}{\delta}\right)}$ and $|C| = |E|$. The last condition is possible since, there always exists a cycle cover of length at most $|E|$ by a pigeon-hole principle argument. The above result and parameters hold for all $T \geq \max\{\frac{K}{|E|} \ln \frac{|E|}{\delta}, 4|C| \ln N\}$.

We will now briefly describe the algorithm and analysis to prove the above theorem. The algorithm is based on the Multiplicative-Weights update paradigm. Algorithm 1 describes it in detail. To describe the algorithm, we will switch from the notion of losses to gains. In other words, define gain $g_{e,t}$ as

$$g_{e,t} = 1 - l_{e,t}$$

Hence, the definition of gain for a path is similar to that of loss for a path.

Algorithm 1: [Shortest Path algorithm in the Semi-Bandit Setting]

```
1 Choose appropriate parameters $\beta > 0$, $\eta > 0$ and $\gamma < 1$
2 At each time step, for each edge we will maintain a weight denoted by $w_{e,t}$, for
    each path a weight $\overline{w}_{i,t}$ and a sum of total weights over all paths $W_t$. Initially,
    we have $w_{e,0} = 1$, $\overline{w}_{i,0} = 1$ and $W_0 = N$
3 At each time step, define the following probability distribution over the paths
    $$p_{i,t} = \begin{cases} (1-\gamma)\frac{\overline{w}_{i,t-1}}{\overline{W}_{t-1}} + \frac{\gamma}{|C|} & \text{if i} \in \text{C} \\ (1-\gamma)\frac{\overline{w}_{i,t-1}}{\overline{W}_{t-1}} & \text{otherwise} \end{cases}$$
4 Now, sample a random path $i_t$ at time t using the above distribution
5 Based on the above distribution, compute the probability that a single edge $e$
    will be chosen. Denote is as $q_{e,t}$. This can be easily calculated as follows
    $$q_{e,t} = \sum_{i:e \in i} p_{i,t} = (1-\gamma)\frac{\sum_{i:e\in i} \overline{w}_{i,t-1}}{\overline{W}_{t-1}} + \gamma\frac{|\{i \in C : e \in i\}|}{|C|}$$
6 The gains $g_{e,t}$ for each edge $e$ in the chosen path is revealed. Compute an
    estimate of gains for all edges as follows
    $$g'_{e,t} = \begin{cases} \frac{g_{e,t}+\beta}{q_{e,t}} & \text{if e} \in i_t \\ \frac{\beta}{q_{e,t}} & \text{otherwise} \end{cases}$$
    Compute the estimate of gains on all paths $g'_{i,t}$ as $\sum_{e\in i} g'_{e,t}$
7 Compute the multiplicative weights update for all $e \in E$, $i \in P$ as follows
    $$w_{e,t} = w_{e,t-1}\exp[\eta g'_{e,t}]$$
    $$\overline{w}_{i,t} = \prod_{e\in i} w_{e,t} = \overline{w}_{e,t-1}\exp[\eta g'_{i,t}]$$
    $$\overline{W}_t = \sum_{i\in P} \overline{w}_{i,t}$$
```

Note that this is similar in flavor to the multiplicative weights update method (See a survey by Arora et al.[2]). Hence, the analysis proceeds in a similar manner of establishing an upper and lower bound on an appropriate potential function.

Proof. Consider the following potential function; $\phi(t) = \sum_{k=1}^{t} \ln \left[\frac{\overline{W}_t}{\overline{W}_{t-1}}\right]$.

Let T be the total number of time-steps.

$$\phi_T = \ln\left(\frac{\overline{W}_T}{\overline{W}_0}\right)$$

Note that from definition, this is $\ln \sum_{i\in P} \overline{W}_{i,T} - \ln \sum_{i\in P} \overline{W}_{i,0}$. From the algorithm above, $\sum_{i\in P} \overline{W}_{i,0} = N$ and $\sum_{i\in P} \overline{W}_{i,T} = \sum_{i\in P} \eta \exp[\sum_{k=1}^{T} g'_{i,k}]$
Using a simple lower bound that $\sum_{i\in P} Q \geq \max_{i\in P} Q$, we have

$$\phi_T \geq \max_{i\in P} \eta \exp[\sum_{k=1}^{T} g'_{i,k}] - \ln N$$

We will now construct an upper-bound. Consider $\ln\left(\frac{\overline{W}_t}{\overline{W}_{t-1}}\right)$

$$\ln\left(\frac{\overline{W}_t}{\overline{W}_{t-1}}\right) = \ln\left(\frac{\sum_{i\in P} \overline{w}_{i,t-1} \exp[\eta g'_{i,t}]}{\overline{W}_{t-1}}\right) \qquad (3.1)$$

Note that we can write $\overline{w}_{i,t-1} = \frac{1}{1-\gamma}\left(p_{i,t} - \frac{\gamma \mathbb{I}(i\in C)}{|C|}\right)$ And since, $\frac{\gamma \mathbb{I}(i\in C)}{|C|} \geq 0$, we can bound 3.1 as

$$\leq \ln\left(\frac{1}{1-\gamma} * \sum_{i\in P} p_{i,t} \exp[\eta g'_{i,t}]\right)$$

$$\leq \ln\left(\frac{1}{1-\gamma} * \sum_{i\in P} p_{i,t}(1 + \eta g'_{i,t} + \eta^2 g'^2_{i,t})\right) \qquad \text{Using } e^x \leq 1 + x + x^2 \text{, for } 0 \leq x \leq 1$$

$$\leq \ln\left(1 + \frac{1}{1-\gamma} * \sum_{i\in P} p_{i,t}(\eta g'_{i,t} + \eta^2 g'^2_{i,t})\right)$$

$$\leq \left(\frac{\eta}{1-\gamma} * \sum_{i\in P} p_{i,t} g'_{i,t} + \frac{\eta^2}{1-\gamma} \sum_{i\in P} p_{i,t} g'^2_{i,t}\right) \qquad \text{Using } \ln(1+x) \leq x$$

Now we have to upper bound the two sums.

$$\begin{aligned}
\sum_{i\in P} p_{i,t} g'_{i,t} =& \sum_{i\in P} p_{i,t} \sum_{e\in i} g'_{e,t} \\
=& \sum_{e\in E} g'_{e,t} \sum_{i\in P: e\in i} p_{i,t} \\
=& \sum_{e\in E} g'_{e,t} q_{e,t} \\
=& \sum_{E\in I_t} g'_{e,t} q_{e,t} + \sum_{e\in E\setminus I_t} g'_{e,t} q_{e,t} \\
=& |E|\beta + \sum_{e\in I_t} g_{e,t}
\end{aligned}$$

Upper bounding the second sum, we have

$$\begin{aligned}
\sum_{i\in P} p_{i,t} g'^{2}_{i,t} =& \sum_{i\in P} p_{i,t} \left(\sum_{e\in i} g'_{e,t} \right)^2 \\
\le& K * \sum_{i\in P} p_{i,t} \sum_{e\in i} g'^{2}_{e,t} && \text{(Using AM-GM inequality)} \\
=& K * \sum_{e\in E} g'^{2}_{e,t} \sum_{i\in P: e\in i} p_{i,t} \\
=& K * \sum_{e\in E} g'^{2}_{e,t} q_{e,t} \\
=& K * \sum_{e\in E} g'_{e,t} q_{e,t} \left[\frac{\beta + \mathbb{I}(e\in I_t) g_{e,t}}{q_{e,t}} \right] \\
\le& K(1+\beta) \sum_{e\in E} g'_{e,t} && \text{Using the fact that } g_{e,t} \le 1
\end{aligned}$$

□

4 Combinatorial Network Optimization

This work was studied by Gai, Krishnamachari and Jain [7].

4.1 Problem Setting

In this paper, the problem setting they consider is as follows. They consider the linear rewards setting. Recall that m denotes the number of machines and T denotes the total time steps. In addition to the initial setting, here rewards are a function of time. They assume a stochastic assumption on the rewards. For a given machine M_i, the reward $r(M_i, n)$ is an i.i.d.(Independent and identical across time. May be correlated among various machines) random variable (here, n is the time-step of the algorithm). Assuming finite support on these random variables, WLOG they assume that $r(.,.) \in [0,1]$. Denote, $\theta_i = \mathbb{E}[r(M_i, n)]$ and denote $\Theta = \{\theta_i\}$. They do not make any assumptions about $\mathcal{C}$ and the budget B.

Define $\mathbf{a}(n)$ to be the fractional vector of actions chosen at time-step n. In other words, $\mathbf{a}$ is a m-dimensional vector corresponding to the set of machines chosen (a machine can be fractionally chosen). Hence, a machine i is said to be "chosen" at time-step n if $\mathbf{a}_i(n) \neq 0$. There exists an underlying set of policies Π from which this vector $\mathbf{a}$ can be chosen. When a set of machines are chosen, the rewards corresponding to those machines are revealed. The reward is defined as

$$R_{\mathbf{a}(n)}(n) = \sum_{i=1}^{m} a_i(n) r(M_i, n)$$

The notion of regret in this setting is defined as follows. Denote $\theta^* = \max_{\mathbf{a} \in \Pi} \sum_{i=1}^{m} a_i \theta_i$. In other words, it denotes the total reward of the optimal policy.

$$\mathcal{R}_n^{\Pi}(\Theta) = n\theta^* - \mathbb{E}^{\Pi}\left[\sum_{t=1}^{n} R_{\pi(t)}(t)\right]$$

Here, $\mathbb{E}^{\Pi}$ denotes expectation given a fixed policy. Hence, the goal of the problem is to minimize the regret and have uniform convergence i.e. as $t \to \infty$, the regret goes to 0.

4.2 Main Result

The main result of this paper is an algorithm called Learning with Linear Rewards (LLR). They give the following regret bounds for this algorithm.

Theorem 4.1. Denote $a_{\max}$ to be the maximum co-ordinate across all actions and time-steps for the policy Π. Denote $\Delta_{\mathbf{a}}$ to be $\theta^* - \theta_{\mathbf{a}}$ Here, $\theta_{\mathbf{a}} = \sum_{i \in \mathbf{a}} a_i \theta_i$. Let

$\Delta_{\min} = \min_{a \neq a^*} \Delta_a$ and $\Delta_{\max} = \max_{a \neq a^*} \Delta_a$ The expected regret of the LLR algorithm is upper-bounded by

$$O((\frac{a_{\max}^2}{\Delta_{\min}^2} m \ln T + m + \frac{\pi^2}{3} m)\Delta_{\max})$$

4.3 Description of the Algorithm

The algorithm 2 resembles the UCB1 algorithm in many ways. Here, they use the fact that multiple actions can give information about a single arm. The algorithm is divided into two parts. The first part is the initialization phase, where we explore each machine at least once. And the second phase is the "explore-exploit" phase where the crux of the algorithm is defined. To simplify notation, for an arm i, $i \in \mathbf{a}$ is true only if the co-ordinate corresponding to i in $\mathbf{a}$ is non-zero. Number of trials vector keeps a count of number of times a particular arm has been "chosen".

Algorithm 2: [Learning with Linear Rewards (LLR)]

```
1  Initialization
2  Let $L = m$. Initialize the sample mean vector $\hat{\theta} = 0$ and the number of trials
     vector $c = 0$
3  for p = 1 to k do
4      Play any arbitrary action $\mathbf{a}$ such that the arm $p$ is non-zero in that action
5      Update the sample mean ($\hat{\theta}$) and the number of trials vector ($c$)
         appropriately
6
7  Main part of the algorithm
8  for t = k+1 to T do
9      Solve the following optimization problem
10     $\mathbf{a} = \arg\max_{a \in \Pi} \sum_{i \in a} a_i \left( \hat{\theta}_i + \sqrt{\frac{(L+1)\ln t}{m_i}} \right)$
11     Update the sample mean ($\hat{\theta}$) and the number of trials vector ($c$)
         appropriately
```

Note that the confidence radius is similar to the one used in UCB1 algorithm and is $O(\sqrt{\frac{\ln t}{m_i}})$.

5 Combinatorial Multi-armed Bandits: general Framework

This work was studied by Chen, Wang and Yuan [5]
This work first introduces the notion of the combinatorial multi-armed bandits problem(CMAB). CMAB generalizes the classical MAB problem, where one is allowed to choose a set of arms called the super-arms (a.k.a. semi-bandits). The key contribution is that the super-arms can be constrained by arbitrary combinatorial structures such as bipartite matching, vertex cover, etc. The framework assumes that there is an oracle which given the mean rewards and the arms, can compute the optimal super-arm, either exactly or approximately. They provide a UCB-algorithm, called the Combinatorial UCB(CUCB), which gives tight a regret bound. This paper then applies this framework to two applications, (i) Probabilistic Maximum Coverage Problem and (ii) Influence Maximization. In this survey, we will focus mainly on the framework and the CUCB algorithm.

5.1 Problem Setting

As usual, there are m arms and with each arm i and each time step t there is an associated random variable $X_{i,t} \in [0, 1]$ which denotes the reward of pulling arm i in time step t. The $X_{i,t}$ for a particular arm i are i.i.d. drawn from an unknown distribution whose mean rewards is μ_i. Like the previous works, there is no independence assumption on the random variables across arms. Additionally, there is a constraint set $\mathcal{S}$ which is a subset of the power-set of arms. In each round, the algorithm can choose a subset of arms $S \in \mathcal{S}$. For a super-arm S, the function $R_t(S)$ denotes the total reward obtained by playing the super-arm. Here, they allow $R_t(S)$ to be most <u>arbitrary non-linear</u> reward functions and with very few restrictions. In particular, they have the following assumptions.

Define $r_\mu(S) = \mathbb{E}[R_t(S)]$. Here, μ denotes the vector of mean rewards for each arm. They assume that the mean reward is a function of the super-arm S chosen and the vector of mean rewards μ. Additionally, they make the following natural assumptions. Firstly, they assume that the function $r_\mu(S)$ is monotonically non-decreasing. In other words, if there are two sets S and S' such that $S \subseteq S'$, then $r_\mu(S) \leq r_\mu(S')$. Secondly, they assume smoothness of $r_\mu(S)$. Formally, suppose there are two mean reward vectors μ and μ', then there is a strictly increasing function f such that $|r_\mu(S) - r_{\mu'}(S)| \leq f(x)$, where $x = \max_{i \in S} |\mu_i - \mu'_i|$.

(α, β)-approximation oracles: Since the constraint set $\mathcal{S}$ is assumed to have originated from an underlying combinatorial optimization problem, the algorithm is given access to a (α, β)-approximation oracle. This oracle returns a α-approximation to the problem with probability β.

Regret Definition: With all the parts of the framework in place, we now describe the notion of regret used in this paper. Let S_t^A denote the super-arm chosen by the algorithm A at time step t. Then, the total expected reward obtained by the algorithm is $\mathbb{E}[\sum_{t=1}^T R_t[S_t^A]]$. Here the expectation is over the randomness of the algorithm, oracle and the reward distribution. Since the oracle returns an (α, β)-approximation to the OPT, for regret we compare against $T \times \alpha\beta \times OPT$. Hence, regret $R_{\alpha,\beta}(A)$ is defined as

$$R_{\alpha,\beta}(A) = T \times \alpha\beta \times OPT - \mathbb{E}[\sum_{t=1}^{T} R_t[S_t^A]]$$

5.2 Main Result

The main result of this paper is an algorithm called the CUCB, which achieves a regret of $O(\ln T \times m \times h)$, where h is some instance specific parameters. More formally, we make the following definitions.

- Call a super-arm S <u>bad</u>, if $r_\mu(S) < \alpha \times OPT$.
- Define $\mathcal{S}_B = \{S : \text{S is bad}\}$
- For an arm $i \in [m]$, define $\Delta_{\min}^i = \alpha \times OPT - \max\{r_\mu(S) : S \in \mathcal{S}_B, i \in S\}$ and $\Delta_{\min} = \min_{i \in [m]} \Delta_{\min}^i$
- Similarly, for an arm $i \in [m]$, define $\Delta_{\max}^i = \alpha \times OPT - \min\{r_\mu(S) : S \in \mathcal{S}_B, i \in S\}$ and $\Delta_{\max} = \max_{i \in [m]} \Delta_{\max}^i$

Theorem 5.1. There exists an algorithm, called the CUCB algorithm, for the CMAB problem which achieves a regret of at-most $c \times \frac{\ln T}{(f^{-1}(\Delta_{\min}))^2} \times m \times \Delta_{\max}$, where c is some constant and f is the smoothness function which was assumed in the problem setting.

5.3 CUCB Algorithm

In this section we will describe the CUCB algorithm. This algorithm resembles the UCB algorithm in almost all steps, except the call to the approximation oracle.

Algorithm 3: [Combinatorial UCB]

```
1 Initialization
2 Define $T_i$ to be a variable which keeps track of the number of times arm $i$ is
    played
3 Define $\overline{\mu}_i$ to be the empirical mean of observed rewards for arm $i$
4 For every arm $i$, play an arbitrary super-arm $S$ such that $i \in S$. Update $T_i$ and
    $\overline{\mu}_i$ appropriately
5 Main part of the algorithm
6 for t = 1 to T do
7     For each arm $i$, denote $\tilde{\mu}_i = \overline{\mu_i} + \sqrt{\frac{c. \ln t}{T_i}}$
8     S = Oracle($\tilde{\mu_1}, \tilde{\mu_2}, \dots, \tilde{\mu_k}$)
9     Play the super-arm S and update $T_i$ and $\overline{\mu}_i$ for all arms based on the
        observed rewards
```

Note that the UCB radius $\sqrt{\frac{c. \ln t}{T_i}}$ resembles the UCB1 algorithm for the MAB problem. The algorithm essentially maintains UCB over each arm separately and uses the oracle to choose the optimal(or near optimal) set of arms satisfying the constraint $\mathcal{S}$.

5.4 Sketch of the Analysis

In this subsection, we will briefly give a sketch of the proof for theorem 5.1. The high level idea is to bound the number of times the algorithm chooses a "sub-optimal" super-arm. And then relate the regret of the algorithm to the this count. In order to bound the number of "sub-optimal" plays, we will charge a particular element of the super-arm for every sub-optimal move.

We will first make a few definitions.

- Let $T_{i,t}$ denote the number of times arm i has been played in the first t round. We say an arm i is played at time t, if i is in the super-arm chosen at round t.
- Let $\tilde{\mu}_{i,t}$, $\overline{\mu}_{i,t}$ be the values of $\tilde{\mu}_i$ and $\overline{\mu}_i$ after the first t rounds of the algorithm respectively.
- Let F_t denote the event that the (α, β) oracle fails to produce an output

- Let $N_{i,t}$ be the number of times arm i is "charged" for the sub-optimal super arm chosen. For any sub-optimal round t, let $i = \arg\min_{j \in S_t} N_{j,t-1}$. Then in around t, arm i is charged a cost of 1 for this round.
- Finally, let $l_t = \frac{c \ln t}{(f^{-1}(\Delta_{\min}))^2}$

With these definitions in place, we will start our analysis.

$$\sum_{i=1}^{m} N_{i,T}$$

$$= m + \sum_{t=m+1}^{T} \mathbb{I}[S_t \text{ is a bad super-arm}]$$

(Every arm is played once in the initilalization phase)

$$\leq m + m \times l_T + \sum_{t=m+1}^{T} \sum_{i=1}^{m} \mathbb{I}[S_t \text{ is bad and } N_{i,t} = N_{i,t-1} + 1 \text{ and } N_{i,t-1} > l_T]$$

(Distributing the sum across arms and counting after the count for each arm is large enough)

$$= m + m \times l_T + \sum_{t=m+1}^{T} \mathbb{I}[S_t \text{ is bad and } \forall i \in S_t,\, N_{i,t-1} > l_T]$$

(Since for each bad play, the arm with least count thus far is charged 1)

$$\leq m + m \times l_T + \sum_{t=m+1}^{T} \mathbb{I}[F_t] + \mathbb{I}[F_t \text{ and } S_t \text{ is bad and } \forall i \in S_t,\, N_{i,t-1} > l_T]$$

(Splitting across two mutually exclusive events)

$$\leq m + m \times l_T + \sum_{t=m+1}^{T} \mathbb{I}[F_t] + \mathbb{I}[\neg F_t \text{ and } S_t \text{ is bad and } \forall i \in S_t,\, T_{i,t-1} > l_T]$$

We know that the first indicator is upper bounded by $(1 - \beta)$ while we will claim that the second indicator is upper bounded by $2mt^{-2}$. The proof of this similar to UCB1 algorithm, using Hoeffding bounds and an union bound over arms.

Hence, we have

$$\mathbb{E}[\sum_{i=1}^{m} N_{i,T}] \leq m(l_T+1)+(1-\beta)(T-m)+\sum_{t=m+1}^{T} 2mt^{-2} \leq c_1(m\times l_T+(1-\beta)(T-m)+m)$$

Note that for every sub-optimal super arm chosen, the loss is reward is at most $\Delta_{\max}$. Hence, the regret upper bound we have is

$$Regret \leq T \times \alpha \times \beta \times OPT - (T \times \alpha \times OPT - \mathbb{E}[\sum_{i=1}^{m} N_{i,T}]\Delta_{\max})$$

The first term is the total reward of the optimal scheme. The first part second term is the reward of the algorithm when it plays an optimal super-arm while the second part is the regret in the sub-optimal rounds. Substituting the appropriate values gives us the bound in 5.1.
It is important to note that the value of l_T was chosen to be large enough to make the Hoeffding bounds work. In other words, we needed every arm to be played sufficient number of times to claim the upper bound of $2mt^{-2}$.

6 Tight Regret Bounds for Stochastic Combinatorial Semi-Bandits

This work was studied by Kveton, Wen, Ashkan and Szepesvari [15]
In this work, they obtain optimal regret bounds for the Combinatorial MAB problem introduced by Chen et al. In particular, they give UCB like algorithm and improve the analysis. Additionally, they also present lower bounds which match their upper bound up to poly-logarithmic factors.

6.1 Key Contributions

This paper subsumes the work from Chen et al. and Gai et al. The setting they study is the same as those two works and the algorithm they use is the same as the one used by Gai et al.. Their key contribution comes from the analysis. The list of key contributions is as follows

- They give one instance specific upper bounds and one instance independent upper bound

- They give one instances specific lower bound and one instance independent lower bound.
- Finally, they also prove that CUCB for the stochastic combinatorial semi-bandits problem, is both sample efficient and computationally efficient.

  Computationally Efficient: Here computationally efficient is defined as an online algorithm that runs in polynomial time, whenever the offline combinatorial optimization problem can be run in polynomial time.

  Sample Efficient: Sample efficiency means that, up to polylogarithmic factors, the upper and lower bound on the regrets are the same.

In this survey, we will describe the instance-specific lower bound. We encourage the reader to refer the original paper for the instance-independent bounds as well the proof for upper bounds.

6.2 Main results and relation to Linear Bandits

In this sub-section, we will formally describe the tight bounds obtained, both instance-dependent and instance-independent. We will also briefly describe a connection to Linear Bandits and justify the model of semi-bandits.

Theorem 6.1 (Instance-dependent)**.** For the Stochastic combinatorial semi-bandit problem, there is an algorithm, known as CombUCB1, which achieves a regret of at most

$$R(T) \leq O(K \times m \times \frac{1}{\Delta} \times \log T)$$

Here, Δ is the gap of the instance.

Theorem 6.2 (Instance-Independent)**.** For the Stochastic Combinatorial semi-bandit problem, the CombUCB1 algorithm achieves a regret of at most

$$R(T) \leq O(\sqrt{K \times m \times T \times \log T})$$

The next two theorems give the matching lower bounds.

Theorem 6.3 (Instance-dependent)**.** For the Stochastic Combinatorial semi-bandit problem, any algorithm achieves a regret of at least

$$R(T) \geq O(K \times m \times \frac{1}{\Delta})$$

Theorem 6.4 (Instance-independent)**.** For the Stochastic Combinatorial semi-bandit problem, any algorithm achieves a regret of at least

$$R(T) \geq \min\{\sqrt{KmT}, KT\}$$

Hence, this paper completely solves the problem of Semi-Bandit with no generalization across arms.

In this paragraph we will give a brief description of relation to linear bandits. This will further justify the model of semi-bandit feedback. Let us first briefly describe the Linear Bandits problem. In Linear Bandit setting, every action $x_t \in X_t \subseteq \mathbb{R}^d$ is a vector in some dimension d. At each time step, the algorithm choses an action x_t and the reward obtained is $< x_t, \theta^* >$ for some unknown but fixed vector θ^*. Audibert et al. [3], among others, give a $R(T) \geq K\sqrt{dm}$ lower bounds for any linear bandit algorithm. Here, K is $|x_t|_1$ for any time t.

Now, let us see how to solve the stochastic combinatorial semi-bandit problem using linear bandits. Every action of the linear bandit is a binary vector $\{0,1\}^m$. Every vector corresponds to a subset of arms chosen. Hence, the expected reward for a particular subset is basically the inner-product of this action vector with the vector containing all mean weights. However, the above lower bound implies that this approach cannot yield any algorithm with a regret better than $R(T) \geq O(K \times \sqrt{m})$. While the optimal algorithm utilizing the semi-bandit feedback can bring the regret down to $O(\sqrt{Km})$.

6.3 $O(K \times m \times (\frac{1}{\Delta}))$ - lower bound

In this sub-section, we will mention the ideas to show the lower bound. The lower bound is shown on a class of algorithms known as consistent algorithms. Without getting into the details of this class of algorithms, we will describe the lower bound approach.

The key idea is to reduce a well-known problem called the Bernoulli bandits (refer

[4] for a survey) to this problem of Combinatorial semi-bandits. The lower bound for Bernoulli bandits is well-studied and hence, it immediately applies to this problem.

The reduction is as follows. Let the set of items be m edges. Denote them as $\{1, 2, \ldots, m\}$. Let the feasible actions be one of L/K paths defined as follows. Every edge from $(j-1)K+1, \ldots, jK$ forms a path. Denote this set by P. The weight probability distribution over the edges is defined as follows.

- For every edge in the same path, the weight is same. Hence, we will talk about probabilities of a path henceforth.
- For every other path, the probability is chosen from a Bernoulli distribution as follows.
$$\mathbb{E}[w] = \begin{cases} 1/2 & \text{for path 1} \\ 1/2 - \Delta/K & \text{otherwise} \end{cases}$$
Note that Δ is some appropriately chosen constant greater than 0.

Notice that the above is an instance of a Bernoulli bandit problem, with m/K arms. Additionally, the rewards are scaled by a factor K since all the edge weights are same within a given path.

Using the lower bounds for Bernoulli bandits, we immediately get the following proposition.

Theorem 6.5. Any consistent bandit algorithm running for a large enough time steps T, achieves a regret of at least $\tilde{O}(\frac{(m-K)K}{\Delta})$.

6.4 Sketch for the Upper bound

In this subsection, we will give a few words about the key ideas of the upper bound. The key idea is to define an appropriately chosen set of mutually-exclusive events. Then, they argue that every time a sub-optimal action is chosen, each of these events happens only a bounded number of times. Hence, to bound the total regret of this algorithm, it suffices to bound the total occurrences of each of these events. And doing that gives the exact upper bound on the regret.

7 Matroid bandits: Fast Combinatorial Optimization with Learning

This work was studied by Kveton, Wen, Ashkan, Hoda and Eriksson [14]
In this work, the authors introduce a notion called Matroid Bandits. Matroid Bandits are a special case of the above work on Stochastic Combinatorial Bandits. Here, the specific combinatorial optimization problem being studied is the modular function maximization under matroid constraints. This can be solved using a greedy approach. Hence, instead of assuming an oracle, in this work the algorithm solves the optimization problem while performing the UCB-type algorithm. The upper and lower bounds from the previous work, pretty much extends to this setting.

7.1 Matroids

In this sub-section, we will give a brief review of matroids. Matroids is a generalization of vector-space. Formally, it is a tuple $M = (E, I)$, where E is the set of elements and I is the set of subsets of elements which form an independent set. Additionally, the following properties hold.

- $\Phi \in I$
- Let $i \in I$ be an independent set. Then for all j such that $j \subseteq i$, $j \in I$
- Suppose $i, j \in I$ such that $|i| = |j| + 1$, then there exists one element a in $i \setminus j$ such that $j \cup a \in I$

Sometimes, we associate weights with each of the elements in E. Let us denote the weight vector w such that $w(e)$ assigns weight to the element e. Then the problem of finding a maximum weight basis of a matroid is the following optimization problem.

$$A^* = \arg\max_{A \in I} \sum_{e \in A} w(e)$$

This is an instance of maximizing a modular function under matroid constraints. This can be solved using the Greedy approach.

7.2 Matroid Bandits

In this section, we will formally define the notion of matroid bandits. We are given a matroid M and a probability distribution over the weights on the arms. We assume that each element e in M is associated with an arm. The constraint set over the super-arm is defined as follows. The algorithm can pull a super-arm S if and only if the elements in S form an independent set in the matroid M. Other than this, the remaining setting of the problem remains same as in section 6.

Now we will define the regret used for this setting. Define $f(A, w) = \sum_{e \in A} w(e)$. Let A_t be the subset chosen by the algorithm in round t and let A^* be the optimal subset. Then the regret at time t, $R(t, w_t) = f(A^*, w_t) - f(A_t, w_t)$, where w_t is the realized weights at round t. The total regret of the algorithm is then

$$R(T) = \mathbb{E}[\sum_{t=1}^{T} R(t, w_t)]$$

7.3 Main Result

The main result of this paper is an algorithm called the Optimistic Matroid Maximization (OMM) which achieves a regret of $O(\sqrt{m \times K \times T \log T})$ where K is the size of the basis of the matroid M. Note that this bound is same as the one in section 6. Additionally, similar to the previous work they show a matching lower bound under this setting, which is tight up to poly-logarithmic factors. Formally, they prove the following two theorems.

Theorem 7.1 (Upper Bounds)**.** Let K be the size of the basis of the matroid M. The algorithm OMM achieves a regret of at most $c_1 \times \sqrt{m \times K \times T \log T} + c_2 \times K \times m$ for some constants c_1 and c_2.

Theorem 7.2 (Lower Bounds)**.** Let K be the size of the basis of the matroid M. Then no randomized algorithm can achieve a regret better than $\frac{m-K}{4\Delta} \log T$, where Δ is some constant lesser than 1/2.

7.4 Algorithm

In this section we will describe the OMM algorithm. At a high level, the algorithm performs UCB update similar to section 6, but instead of making a call to the oracle, it runs the greedy algorithm for the matroid basis maximization problem. Algorithm

4 describes the algorithm in full detail.

Algorithm 4: [Optimistic Matroid Maximization]

```
1  Initialization
2  Define $T_i$ to be a variable which keeps track of the number of times arm $i$ is played
3  Define $\overline{\mu}_i$ to be the empirical mean of observed rewards for arm $i$
4  For every arm $i$, play an arbitrary super-arm $S$ such that $i \in S$. Update $T_i$ and $\overline{\mu}_i$ appropriately
5  Main part of the algorithm
6  for t = 1 to T do
7      For each arm $i$, denote $\tilde{\mu}_i = \overline{\mu_i} + \sqrt{\frac{c.\ln t}{T_i}}$
8      $A_t = \phi$
9      Sort elements in $E$ to $e_1^t, e_2^t, \ldots, e_k^t$, such that $\tilde{\mu_{e_1^t}} \geq \tilde{\mu_{e_2^t}} \geq \ldots \geq \tilde{\mu_{e_k^t}}$
10     for i=1 to m do
11         if $e_i^t \cup A_t \in I$ then
12             $A_t = A_t \cup e_i^t$
13
14     S = $A_t$
15     Play the super-arm S and update $T_i$ and $\overline{\mu}_i$ for all arms, based on the observed rewards
```

The above algorithm is a special application where the ORACLE is an exact algorithm and no (α, β) approximation is required.

8 Adaptive submodular maximization in bandit setting

This work was studied by Gabillon, Kveton, Wen, Erikkson and Muthukrishnan [6] In this paper, they study the adaptive submodular maximization problem. a well-known problem in combinatorial optimization, in the semi-bandit setting. They present an UCB-style algorithm for this problem and prove upper bounds on regret. In particular, they present an algorithm called the Optimistic adaptive submodular maximization (OASM).

8.1 Adaptive sub-modular maximization

The adaptive submodular maximization problem is defined as follows. Let $f : 2^E \times \{-1, 1\}^k \to \mathbb{R}$. The first set of inputs is a subset of the set of elements while the second set of inputs is the "state" of each of the item. In this setting, the state is assumed to be random and drawn i.i.d. from a Bernoulli distribution. The reward of choosing set of arms A in state ϕ is given by $f(A, \phi)$. Additionally, we denote $\phi(A)$ to be the observed states for the items in set A.

Function f is called adaptive submodular if condition on a state ϕ, the function is sub-modular in the first set of inputs. In other words, let $A \subseteq B \subseteq E$ and $i \in E \setminus B$, then

$$\mathbb{E}_\phi[f(A \cup \{i\}, \phi) - f(A, \phi)|\phi(A)] \geq \mathbb{E}_\phi[f(B \cup \{i\}, \phi) - f(B, \phi)|\phi(B)]$$

Additionally, in this setting we also assume f is adaptive monotonic. This means that for any set $A \subseteq E$ and $i \in E \setminus A$

$$\mathbb{E}_\phi[f(A \cup \{i\}, \phi) - f(A, \phi)|\phi(A)] \geq 0$$

For purposes of measuring regret, we need to define a greedy policy. A function $\pi : \{-1, 0, 1\}^m \to E$ is called a policy function. It maps the observed states(a state of item is 0 if it is not observed) to the set of items. We denote ϕ_0 to be a m dimensional vector, with 0 in position for items whose state is not observed. A greedy policy $\pi^G(\phi_0)$is hence, the one that selects the item with the highest expected gain. In other words,

$$\pi^G(\phi_0) = \arg\max_{i \in E \setminus \phi_0} g_i(\phi_0)$$

where

$$g_i(\phi_0) = \mathbb{E}_\phi[f(A \cup \{i\}, \phi_0) - f(A, \phi_0)|\phi_0]$$

Also, define

$$\overline{g}_i(\phi) = \mathbb{E}_\phi[f(A \cup \{i\}, \phi) - f(A, \phi)|\phi, \phi[i] = 1]$$

8.2 Problem Setting

With all the definitions in place we now formally define the problem setting. We have m arms and an associated state of each arm denoted by a vector ϕ. Whenever a set of arms A is pulled, the reward obtained is $f(A, \phi)$, where f is an adaptive sub-modular and monotone function. The states ϕ are drawn i.i.d. across times and independent across arms, from a Bernoulli distribution whose means are unknown to the algorithm. The total reward obtained by the algorithm is $\sum_{t=1}^{T} f(\pi_t(\phi_t), \phi_t)$. To measure regret, we will compare it against the greedy policy. Golovin and Krause [9] showed that when f is an adaptive submodular and monotone function, one can get a constant factor approximation using the greedy approach. Hence, it is reasonable to measure the regret against this policy. Formally, the regret $R(T)$ is defined as

$$R(T) = \mathbb{E}_{\mathbf{\Phi}}[\sum_{t=1}^{T} f(\pi^{G}(\phi_t), \phi_t) - f(\pi^{t}(\phi_t), \phi_t)]$$

8.3 Main results

The main contribution of this paper is an algorithm, called the Optimistic adaptive submodular maximization (OASM). This is an UCB style algorithm. They show that this algorithm obtains a regret upper bound of $O(\log m)$, where m is the number of arms. More formally, they prove the following theorem. We assume that at each time step, the algorithm chooses the set A_t in H steps. See the algorithm description for more clarity.

Theorem 8.1 (Upper Bound)**.** Let $\alpha_{i,h}$ be a constant that associates fraction of regret of arm i to step h. Hence, $\sum_{h=1}^{H} \alpha_{i,h} = 1$. Let G_h be an upper bound on the expected gain of policy π^g from step h forward. Let $l_{i,h}$ be the number of pulls after which arm i is pulled optimally at step h with high probability. And let $l_i = \max_{h \in [H]} l_{i,h}$. Then the total regret is bounded by

$$R(T) \leq \sum_{i=1}^{m} l_i \sum_{h=1}^{H} G_h \alpha_{i,h} + c_2 \times m(m+1) \sum_{h=1}^{H} G_h$$

for some constant c_2. It can be shown that the first term on the RHS is at most $O(\log m)$ while the second term is at most $O(1)$.

8.4 OASM algorithm

In this section we will describe the OASM algorithm. This algorithm is an UCB style algorithm. It maintains UCB estimates over the mean of probabilities and does a greedy maximization using the UCB estimates. Algorithm 5 describes this algorithm in full detail.

Algorithm 5: [Optimistic Adaptive Submodular Maximization]

```
1  Initialization
2  Define $T_i$ to be a variable which keeps track of the number of times arm $i$ is
     played
3  Define $\overline{p}_i$ to be the empirical mean of observed probabilities for arm $i$
4  For every arm $i$, play an arbitrary super-arm $S$ such that $i \in S$. Update $T_i$ and
     $\overline{p}_i$ appropriately
5  Main part of the algorithm
6  for t = 1 to T do
7      For each arm $i$, denote $\tilde{p}_i = \overline{p_i} + \sqrt{\frac{c.\ln t}{T_i}}$
8      Let $A_t = \phi$ for h = 1 to H do
9          $\Phi_0 = \Phi_t(A_t)$
10         $A_t = A_t \cup \arg\max_{i \in E \setminus A_t} \tilde{p}_i \overline{g}_i(\Phi_0)$
11     S = $A_t$
12     Play the super-arm S and update $T_i$ and $\overline{p}_i$ for all arms, based on the
         observed rewards
```

This is again very similar to the UCB1 algorithm, where the greedy algorithm is "embedded" into the UCB algorithm. In other words, the above is a greedy algorithm on the UCB values of p.

9 Cascading bandits: Learning to rank in the cascade model

This work was studied by Kveton, Szepevari, Wen and Ashkan [13]
In this paper, they introduce the notion of cascading bandits. Cascading model is a popular model, where along with choosing K items, an ordering among the K items is provided. For example, consider the example of an online shopping, where K items are provided in a list. The user goes through the list one-by-one in the order and clicks the first link, he/she likes. Hence, the observation obtained is that, for all the items before the clicked item, the user found it unattractive. However, for the items appearing after the clicked item in the list, no feedback is obtained.

9.1 Problem Setting

In this sub-section, we provide the formal definitions of the problem setting. As usual we have a set of m arms. We have an unknown probability distribution P over the rewards on these arms. We have a number K, where in each round the K recommended items are shown in an ordered form. The weights w_t are drawn i.i.d. from P. For each item e, $w_t(e) \in \{0, 1\}$. $w_t(e) = 1$ implies that the user likes item e. It is 0 otherwise.

At each time step t, the algorithm outputs a list $A_t = (w_1^t, w_2^t, \ldots, w_K^t)$ which is an ordered set of arms. We will call the positions in this order as "slots". The problem instance then draws the weights i.i.d. and reveals only the position of the "first" non-zero slot (if present). The reward obtained at time t is 1 if at least one slot has a non-zero reward. It is 0 otherwise. Note that even though feedback depends on the order, the reward function is independent of the order. This can be written in the following algebraic manner

$$f(A, w) = 1 - \prod_{i=1}^{K}(1 - w_i^t)$$

In this model, an additional assumption about the unknown distribution is made. In particular, we further assume that weights across arms are <u>independent</u>. Additionally, we also assume that for each arm, the weight is drawn from a Bernoulli distribution with mean $\overline{w}_i$.

Definition of Regret: We will now define the notion of regret used in this context. We will first make a few more notations. We will call $\Pi_K(E)$ to be the set of all ordered lists that are admissible. Denote A^* to be the optimal ordered set for a given set of mean weights. In other words

$$A^* = \arg\max_{A \in \Pi_K(E)} f(A, \overline{w})$$

Define instantaneous regret $R(A_t, w_t) = f(A^*, w_t) - f(A_t, w_t)$. Hence the total regret of the algorithm is defined as

$$R(T) = \mathbb{E}[\sum_{t=1}^{T} R(a_t, w_t)]$$

9.2 Main Results

The key contribution of this paper is an UCB-style algorithm for the Cascading Bandits problem. In particular, they define an algorithm called the CascadeUCB1 algorithm. They then prove an instance specific upper bound on this algorithm. Additionally, they also prove a matching lower bound for this version of the problem.

Before formally stating the theorems, let us make the following definitions. Define $\Delta_{a,b} = \overline{w}_a - \overline{w}_b$ for any two arms a, b.

Theorem 9.1 (Upper Bound)**.** Let the optimal ordering given the mean weights be denoted as $a_1, a_2, \ldots, a_K$. Let $\Delta_{e,K}$ denote the Δ function between arm e and arm a_K. The algorithm CascadeUCB1 algorithm achieve a regret bound of at most

$$R(T) \leq \sum_{a=K+1}^{m} \frac{c_1 \log T}{\Delta_{e,K}} + c_2 \times m$$

where c_1 and c_2 are some appropriate constants.

Theorem 9.2 (Lower Bound)**.** Let $p \in (0, 1)$ and $\delta \in (0, p)$ be some appropriately chosen constants. Define KL(p || q) to be the KL divergence between two distributions p and q. Then no algorithm for the cascading bandit problem can achieve a regret lesser than

$$R(T) \geq \log T \times \frac{(m-K) \times \delta \times (1-p)^{K-1}}{KL(p-\delta || p)}$$

Let P be the lower bound instance achieving the lower bound stated in the above theorem. It can be shown that the upper bound obtained on this instance P simplifies to give a multiplicative factor of K greater than the lower bound. However, the paper states another algorithm whose regret matches the lower bound up to poly-logarithmic factors.

9.3 Cascade UCB algorithm

In this section, we will describe the Cascade UCB algorithm described in the paper. The algorithm is similar to UCB1 with similar confidence radius. One small step is incorporated to give an ordering among the elements. Rest of the steps follow through similarly. Algorithm 6 describes this algorithm in detail.

Algorithm 6: [Cascade UCB algorithm]

```
1  Initialization
2  Define $T_i$ to be a variable which keeps track of the number of times arm $i$ is
     played
3  Define $\overline{\mu}_i$ to be the empirical mean of observed rewards for arm $i$
4  For every arm $i$, play an arbitrary super-arm $S$ such that $i \in S$. Update $T_i$ and
     $\overline{\mu}_i$ appropriately
5  Main part of the algorithm
6  for t = 1 to T do
7      For each arm $i$, denote $\tilde{\mu}_i = \overline{\mu_i} + \sqrt{\frac{c. \ln t}{T_i}}$
8      Sort the arms by their UCB values
9      Pick S to be the top $K$ arms with the highest UCB values sorted from
         highest to lowest among them
10     Play the super-arm S and update $T_i$ and $\overline{\mu}_i$ for all arms, based on the
         observed rewards
```

This algorithm again resembles the UCB1 algorithm. The only difference being that after computing the UCB statistics with the same confidence radius, the algorithm sorts the arms by their UCB values and picks the top K arms in the order of their UCB values.

10 Efficient Learning in Large-Scale Combinatorial Semi-Bandits

This work was studied by Wen, Kveton, Ahkan [17].
In this paper, they consider the instance of the semi-bandit problem under combinatorial constraints. They assume a linear generalization over the means of the arms. This assumption helps in getting a regret bound that is almost independent of m, the number of arms.

10.1 Problem Definition

An instance of the combinatorial semi-bandit problem is defined by a tuple $(\mathcal{E}, \mathcal{A}, \mathcal{P})$. Here, $\mathcal{E}$ is the set of arms, $\mathcal{A}$ is the set of allowed actions and $\mathcal{P}$ is a probability distribution over the mean rewards over the arms. For simplicity, in this survey we will consider the simple combinatorial constraint of choosing at most Q arms. The goal of the algorithm is to minimize the total expected regret.

10.2 Linear Generalization across arms

In this section, we will give precise statements about the linear generalization across arms. Let $\overline{\mathbf{w}} \in \mathbb{R}^m$ denote the vector of mean rewards of all arms. Linear generalization means that, there exists a generalization matrix $\Phi \in \mathbb{R}^{m \times d}$, such that $\overline{\mathbf{w}} = \Phi \times \theta$. Here, this generalization matrix is known to the algorithm. It models the fact that there are features, which define each arm. Typically, the number of arms m is very large, while the number of features d is small. Hence, the regret scales as a "nice" function of m (Typically, $\log m$ or independent of m).

10.3 Algorithms

The main contribution of this paper is to extend two well-known algorithms, Thompson Sampling and LinearUCB, to this setting. In this survey we will describe the extension of LinearUCB (which they call CombinatorialLinearUCB) in detail.

10.4 Combinatorial Linear UCB

The algorithm follows the route of estimating the Upper Confidence Bounds on the unknowns and using that to play the arms. Similar to Linear UCB, the following claim on the UCB holds.

Theorem 10.1. *Choose a constant c large enough appropriately (Refer the original paper for the exact constant). With high probability, we have the following inequality to be satisfied*

$$\Phi_e \times \theta_t - c\sqrt{\Phi_e^T \Sigma_t \Phi_e} \leq \Phi_e \times \theta^* \leq \Phi_e \times \theta_t + c\sqrt{\Phi_e^T \Sigma_t \Phi_e} \tag{10.1}$$

This theorem leads to the following natural UCB type algorithm

Algorithm 7: [Combinatorial Linear UCB]

```
1 Initialization
2 $\Sigma_1 = \lambda^2 I \in \mathbb{R}^{d \times d}$
3 $\theta_1 = 0 \in \mathbb{R}^d$
4 Main part of the algorithm
5 for t = 1 to T do
6     Define the following estimate of the mean reward
7         $\overline{w}_t(e) = \Phi_e \times \theta_t + c\sqrt{\Phi_e^T \Sigma_t \Phi_e} \;\; \forall\, e \in \mathcal{E}$
8     Update $\theta_t$ and $\Sigma_t$ using a "Kalman filtering" procedure
9     Update the sample mean ($\hat{\theta}$) and the number of trials vector ($m$)
        appropriately
```

The algorithm above uses a similar approach to the Linear UCB algorithm. The key difference in the above algorithm is the Kalman filter procedure to update the values of θ_t and Σ_t in each time step. At an intuitive level, Kalman filter gives a provable method to estimate mean and variance, when the relationship is governed by a linear dynamics. In particular, it assumes a Markovian relationship in the transition of states; i.e. state at time t depends only on the state of the system at time t-1. Kalman filter is the right notion here, since we have a linear relationship between the mean of the weights and the parameter whose mean and variance we are trying to estimate in the UCB. Since, Kalman filter is a standard procedure, we encourage the reader to refer to the original paper to get the exact update equations.

10.5 Main Result

The main result of this paper bounds the regret of the above Combinatorial Linear UCB algorithm. In particular, we have the following Theorem.

Theorem 10.2. Assume linear generalization across the arms, the constant c to be large enough and the support of the probability distribution $\mathcal{P}$ being a subset of $[0, 1]^m$. Then the regret of the algorithm is upper-bounded by $\tilde{O}(Kd\sqrt{T})$.

The key take-away from this theorem is that, the regret is independent of the number of arms m. It instead depends on d, which is typically much smaller. The other take-away is that one can obtain same UCB analysis by replacing the confidence hyper-cube with confidence ellipsoids introduced in [1].

10.6 Other result in this paper

Without getting into full details, we briefly mention the other result presented in this paper. They extend the Thompson Sampling algorithm to this setting. They call this algorithm the Linear Thompson Sampling. They prove the Bayesian Regret for this algorithm. In this case, the Bayesian regret is bounded by $\tilde{O}(\min\{Kd, K\sqrt{d \ln m}\})$. Unlike the UCB algorithm, the regret of this algorithm depends on the number of arms. However, the dependence is only $\sqrt{\ln m}$ as opposed to the typical $\sqrt{m}$.

11 DCM Bandits: Learning to Rank with Multiple Clicks

This work was studied by Katariya, Kveton, Szepesvari and Wen [12].
The main contribution of this paper is a bandit setting called the Dependent Click Model(DCM)-bandits. This model is a generalization of the cascade model. In particular, similar to the cascade model, this model handles user behavior However, the key difference between cascade model and this is that in the former, the user can click at most one item. On the other hand, this model captures the notion that user can click on multiple items and leaves only when he is satisfied.

11.1 DCM model

In this sub-section, we will define the DCM model more formally. Let
$A = \{a_1, a_2, \ldots, a_m\}$ be a set of m items. The user is provided with a list of K items, denoted as $I = (a_{i1}, a_{i2}, \ldots, a_{iK})$. The user scans the items from a_{i1} to a_{iK} one-by-one in that order. The model is parameterized by two vectors, $\overline{w} \in [0,1]^m$ and $\overline{v} \in [0,1]^m$. Once the user examines item a_{iK} in the list, with probability $\overline{w}(a_{iK})$ he will click the item. Then, with probability $\overline{v}(a_{iK})$ he becomes satisfied and terminates, while with probability $1 - \overline{v}(a_{iK})$ he chooses to continue along the list. The key thing to note is that, both $\overline{w}(a_{iK})$ and $\overline{v}(a_{iK})$ are conditional probabilities. They are conditioned on the event that user examines the event K.

11.2 Bandit Problem in the DCM model

We will now define a natural Bandit problem on this defined DCM model. The problem is defined as follows. Let E denote the set of ground items. At each time step t, the algorithm has to choose a subset k ordered items from a set of policies Π_K. The system then chooses i.i.d. the vector $w_t \in \{0,1\}^m$ and the vector $v_t \in \{0,1\}^m$. The binary vector w_t corresponds to whether the user finds the item attractive, while v_t corresponds to whether the user will terminate or examine other items. The algorithm then receives a reward 1 if and only if user liked at least one item and the stopping value corresponding to it (i.e. v_t) was a 1. In other words, the reward $f(A_t, w_t, v_t)$ can be expressed as

$$f(A_t, w_t, v_t) = 1 - \prod_{k=1}^{K}(1 - v(k)w(a_k))$$

The assumption made is that $w_t(e)$ and $v_t(e)$ are independent across the arms. Hence, for each arm, $w_t(e) = Bernoulli(\overline{w}(e))$ and $v_t(e) = Bernoulli(\overline{v}(e))$.

With the model in place, we will now describe the definition of regret.

Regret: Define $A^* = \arg\max_{A \in \Pi_K} f(A^*, w_t, v_t)$. Then the instantaneous regret $R(t)$ is defined as

$$R(t) = f(A^*, w_t, v_t) - f(A_t, w_t, v_t)$$

And hence, the cumulative regret is defined as

$$\mathbb{E}[R(T)] = \sum_{t=1}^{n} \mathbb{E}[R(t)]$$

11.3 dcmKL-UCB Algorithm for DCM bandits

In this sub-section, we will describe a UCB style algorithm for the DCM bandits problem. This algorithm is motivated by the KL-UCB algorithm [8]. The key idea is as follows. As usual, we will maintain a UCB on the weight vector and update it based on the number of plays and the observed weights from previous time-steps. The key twist is the way in which the UCB update is done. Here, the update is based on a KL divergence bound of the sample mean and log t (as opposed to sqrt(log t) and sample mean). Intuitively, this is strictly better than using UCB since Pinsker inequality gives $KL(\mu||\mu^*) \geq 2(\mu - \mu^*)^2$. Hence, an upper bound on the KL divergence is surely an upper bound on $(\mu - \mu^*)^2$. Another key point of this algorithm is that, only the ordering of the $\overline{v}$ vector values among the arms is required. In particular, without loss of generality we can assume that $\overline{v}(1) > \overline{v}(2) > \ldots > \overline{v}(m)$. Hence, any vector $\tilde{v}$ satisfying this criterion can be used in the algorithm.
Algorithm 8 describes the algorithm in full detail. For ease of description, define

$$W_t(e) = \arg \max_{\tilde{w}(e) \leq w' \leq 1} \{N_{t-1}(e) \times KL(\tilde{w}(e)||q') \leq O(\log t)\} \tag{11.1}$$

here, $N_{t-1}(e)$ is the number of times arm e has been played up to time $t-1$.

Algorithm 8: [KL-UCB style algorithm for DCM bandits]

```
1  **Initialization**
2  Play every arm once and observe weights
3  $N_0(e) = 1$
4  $\tilde{w}(e) = w_0(e)$
5  **Main part of the algorithm**
6  **for** t = 1 to T **do**
7      For all arms $e$, compute UCB $W_t(e)$ using 11.1
8      Choose an action $A_t$ as follows:

                    $\arg \max_{A \in \Pi_K} f(A, W_t, \tilde{v})$

9      Play the action $A_t$ and observe the position of the last click $c$ after which
         the user leaves
10     For all arms from $1, 2, \ldots, c$, update the number of times played and the
         sample mean
```

Notice that, the update is made only for all the arms in the sequence before the last click(i.e. the user leaves). Otherwise, the algorithm follows a skeletal UCB algorithm.

11.4 Main Results

In this sub-section, we will mention the main results of this paper. They provide both upper and lower bounds. On the upper bound side, they prove Theorem 11.1 for equal stopping probabilities and Theorem 11.2 for arbitrary stopping probabilities. While on the lower bound side, they prove Theorem 11.3.

Theorem 11.1. *Let the stopping probabilities for all the arms be the same value, say p. Similar to cascading bandits in section 9. Define $\Delta_{e,K}$ to be the gap of the problem instance. Then the cumulative regret is upper bounded by*

$$R(n) \leq \tilde{O}\left(p \times \sum_{e \in K+1}^{m} \frac{\Delta_{e,K}(1 - \log \Delta_{e,K})}{KL(\overline{w}(e)||\overline{w}(K))} \times \log m\right)$$

Here, $\tilde{O}$ hides the $\log \log m$ terms.

Theorem 11.2. Suppose the stopping probabilities are arbitrary. Additionally, let $\overline{v}(1) > \overline{v}(2) > \ldots \overline{v}(m)$. Then the above theorem can be modified to

$$R(n) \leq \tilde{O}\left(\left(\sum_{i=1}^{m} \overline{v}(i) - \overline{v}(i+1)\right) \times \sum_{e \in K+1}^{m} \frac{\Delta_{e,K}(1 - \log \Delta_{e,K})}{KL(\overline{w}(e)||\overline{w}(K))} \times \log m\right)$$

Here $\overline{v}(m+1) = 0$.

Additionally, they also provide matching lower bounds up to logarithmic factors in the gap of the instance.

Theorem 11.3. Let Δ be an instance dependent constant and p be any constant between 0 and 1. Then, the regret of any DCM bandit algorithm is lower bounded as

$$R(n) \geq \left(\frac{(m-K)\Delta}{KL(p - \Delta||p)} \times \log m\right)$$

Notice that this form is similar to the lower bound we saw in section 6. Unsurprisingly, the reduction is similar.

12 Influence Maximization with Semi-Bandit Feedback

This work was studied by Wen, Kveton and Valko [18].
In this work, they consider the problem of Influence Maximization in the Semi-Bandit feedback setting. Let us first start by defining the Influence Maximization problem as a combinatorial optimization problem.

12.1 Influence Maximization Problem

Consider a graph $G = (V, E)$ and let $w : E \to \{0, 1\}$ be a weight function on the edges. Given two vertices u and v, we say (u, v) is connected if an only if there is path from u to v along edges of weights 1. Given a set $S \subseteq V$ as "source vertices", we say a vertex v is "influenced" by the set S if v is connected to some vertex in S. Finally, define the function $f(S, w)$ to be the total number of influenced vertices by S under the weight function w. With these definition in place, let us now define the combinatorial optimization problem of influence maximization.

Given a graph G and number K which is the size of the source vertices and a set of mean rewards for each edge $\overline{w}$, the goal is to choose a set S of size at most K such that $\mathbb{E}[f(S, w)]$ is maximized. Here, the expectation is over the mean weights and any randomness used by the algorithm. Additionally, the following assumption is usually made. The weight of each edge is drawn independent from other edges.

The general problem of influence maximization is NP-hard. However, there are algorithms to solve this approximately. For the purposes of bandit setting, we will assume that there exists a (α, β) oracle which, given an instance returns a α-approximation to the problem with probability at least β.

12.2 IM in the semi-bandit setting

In this sub-section we formally define the problem in the bandit setting under semi-bandit feedback. For an IM problem $(G, K, \overline{w})$, the means $\overline{w}$ is unknown to the algorithm. At each time-step t, the algorithm chooses a subset $S_t \subseteq V$. Then some weights of edges are revealed based on the following condition. The weight of an edge (u, v) is revealed if and only if one of u or v is influenced by the set S_t under the weights w_t chosen by the adversary. To restrict the power of the adversary, the following additional linear generalization assumption is made. There exists a $\underline{\text{known}}$ generalization matrix $\phi \in \mathbb{R}^{|E| \times d}$ such that $\overline{w} = \Phi.\theta$. As usual, the goal of the algorithm is to minimize the regret. Regret is defined as follows. Let there exist an (α, β)-approximation oracle to the problem. Then

$$R(t) = f(S_{oracle}, w_t) - \frac{1}{\alpha\beta} f(S_t, w_t)$$

and the cumulative regret is

$$R(T) = \mathbb{E}[\sum_{t=1}^{T} R(t)]$$

12.3 Main results of this paper

The main results of this paper is as follows. They first give an UCB based algorithm for this bandit problem. The algorithm, IMLinUCB, is based on the UCB algorithm for the Linear Bandits problem adapted to this setting. They then analyze the regret obtained by this algorithm in the case of forests. The regret bounds depend on a newly defined notion called the maximum observed relevance. To keep the exposition simple, we will give an intuitive description of this measure. Every time a set S is chosen, only certain edge weights are revealed. Fix such an S. For any vertex v only certain edges are "relevant" i.e. the edges that lie on a path that influence v. Let $R(e)$ denote the number of times a particular edge influences some node in the graph. Let R denote the sum of $R(e)$ values across all edges. The maximum observed relevance is related to the maximum value of R among all possible chosen subsets. Additionally, this measure depends on the topology of the graph.

With this definition in place, let us state the main theorem proved in this paper. Since the maximum observed relevance depends on the topology, this regret bound is an instance dependent regret bound.

Theorem 12.1. Let C^* denote the maximum observed relevance. Let E^* denote the maximum cardinality of any observed edge set in the offline graph, for all possible K sized source sets S. Additionally, we make the following assumptions (1) the input graph G is a forest (2) The algorithm has access to an ORACLE that returns a (α, β)-approximation to the offline problem $(G, \hat{w_t})$. Then, IMLinUCB achieves a regret bound upper bounded by

$$R(T) \leq \tilde{O}(d\sqrt{n} \times C^* \times E^* \times \frac{1}{\alpha\beta})$$

12.4 IMLinUCB algorithm

In this section, we will describe the IMLinUCB algorithm. This algorithm resembles the Combinatorial Linear UCB 7 in the UCB update step because of the Linear Generalization assumption. The key idea is to compute UCB values and normalizing them to lie in the interval $[0, 1]$. The use the ORACLE to compute the best approximate subset S of size K and play it. Finally, update the values for the observed edges. Algorithm 9 describes this algorithm in full detail.

Algorithm 9: [IMLinUCB algorithm for Influence Maximization]

```
1  **Initialization**
2  $B_0(e) = 0 \in \mathbb{R}^d$
3  $M_0 = I \in \mathbb{R}^{|E| \times d}$
4  Choose appropriate parameters $\sigma, c > 0$
5  **Main part of the algorithm**
6  **for** t = 1 to T **do**
7      Let $\theta_{t-1} = \frac{1}{\sigma^2} M_{t-1}^{-1} B_{t-1}$
8      For all arms $e$, compute UCB $W_t(e)$ as follows
           $$W_t(e) = Projection(\phi_e \theta_{t-1} + c\sqrt{\phi_e^T M_{t-1}^{-1} \phi_e})$$
9      Choose an subset $S_t$ as ORACLE($G$, $W_t(e)$)
10     For all the edges update the matrices as follows: $B_t = B_{t-1} + \phi_e w_t(e)$ and
         $M_t = M_{t-1} + \frac{1}{\sigma^2} \phi_e \phi_e^T$
```

13 Cascading Bandits for Large-Scale Recommendation Problems

This work was studied by Zong, Ni, Sung, Ke, Wen and Kveton [20]
The goal of this paper is to revisit the Cascading Bandits problem, but the in large-number of arms setting. Hence, similar to section 10, we will assume a linear generalization across arms and re-derive regret bounds.

13.1 Linear Generalization across arms

The setting for this problem remains the same as in section 9 with the following linear generalization assumption. There exists a known generalization matrix ϕ such that $\overline{w} = \Phi\theta$ where $\phi \in \mathbb{R}^{m \times d}$ and $\theta \in \mathbb{R}^d$. Hence, the goal of the algorithms is to obtain regret bounds in terms of d and be independent of m.

13.2 Main Contribution of this paper

The main contribution of this paper is to device an algorithm called CascadeLinUCB, which exploits the linear generalization and achieves regret bounds independent of the number of arms. The algorithm is very similar to 7, 9 and obtains UCB bounds in a similar form. They then prove the regret bounds of this paper. In particular, they prove the following theorem.

Theorem 13.1. Assuming Linear generalization across the arms and for some fixed large constant c, CascadeLinUCB achieves a regret of

$$R(T) \leq \tilde{O}(K \times d \times \sqrt{T})$$

Note that these bounds match the same bounds as that in section 10, inspite of being a strictly harder model.

13.3 Cascade Linear UCB

In this section we will describe the Cascade Linear UCB algorithm. This algorithm is a combination of Algorithm 6 with the UCB updates resembling that of algorithm 7.

Algorithm 10 describes the algorithm in full detail.

Algorithm 10: [IMLinUCB algorithm for Influence Maximization]

```
1  Initialization
2  $B_0(e) = 0 \in \mathbb{R}^d$
3  $M_0 = I \in \mathbb{R}^{|E| \times d}$
4  Choose appropriate parameters $\sigma, c > 0$
5  Main part of the algorithm
6  for t = 1 to T do
7      Let $\theta_{t-1} = \frac{1}{\sigma^2} M_{t-1}^{-1} B_{t-1}$
8      For all arms $e$, compute UCB $W_t(e)$ as follows
           $$W_t(e) = \min(\phi_e \theta_{t-1} + c\sqrt{\phi_e^T M_{t-1}^{-1} \phi_e}, 1)$$
9      Sort the arms by their UCB values
10     Pick S to be the top $K$ arms with the highest UCB values sorted from
         highest to lowest among them
11     Play the super-arm S
12     For all the items $e$ in S up to the last click of the user
           $$B_t = B_{t-1} + \phi_e \mathbb{I}[C_t = e]$$
           $$M_t = M_{t-1} + \frac{1}{\sigma^2} \phi_e \phi_e^T$$
         Here, $\mathbb{I}[C_t = e]$ denotes an indicator for whether user clicked item $e$ or not.
```

14 Conclusion and some remarks

In this survey, we listed all the known theoretical results for the stochastic bandit problems in the semi-bandit feedback setting. All algorithms were UCB based algorithms adapted to the particular problem setting. We would like to summarize a general strategy for considering semi-bandit problems and designing algorithms.

For any version of bandit problem with semi-bandit setting, it is possible to consider two kinds of problems.

- First, consider problems with no assumption on the rewards for individual arms.

In this case, designing the UCB algorithm will use UCB estimates similar to UCB1. In other words, the algorithm maintains the number of times each arm is played and a sample mean. Then uses the UCB estimate of the form $\hat{w}(e) + \sqrt{\frac{\log T}{n_T(a)}}$. Based on the setting, the algorithm assumes these UCB values as the actual weights and plays the best possible set of arms.

- Second, it is possible to consider a linear generalization across arms using a known generalization matrix. The goal then is to remove the dependence on the number of arms. In this case, the design of UCB estimate is slightly modified. A sample estimate of the mean B and co-variance M is maintained. The UCB estimate is then $\phi_e B_e + c\sqrt{\phi_e M \phi_e^T}$ where ϕ_e is the generalization matrix row for arm e. Update of B and M based on the observed rewards can be done using a Kalman filtering procedure.

As a proof strategy, these two variations are very similar. The only difference lies in showing a statement that the UCB estimate is an upper-bound on the mean weight with high probability. In the first case, it is shown by considering a hyper-cube and Chernoff-bounds directly imply the high-probability bound. For the second case, replacing this hyper-cube with a slightly involved Confidence Ellipsoids introduced in [1] and Chernoff-bounds imply the high-probability bounds.

It is indeed an interesting question to ask if a more generalization across arms can give bounds independent of the number of arms. In particular, [17] conjecture that for generalized linear functions, a similar algorithm should give regret bounds independent of the number of arms.

Finally, on the lower bound side, all works rely on the strategy of reducing an instance of the Bernoulli bandit problem to the appropriate setting. Then, the claim is that since Bernoulli bandits have a well-known lower bound, the same lower bound immediately transfers to the setting under consideration. In this survey, we illustrate one such example.

References

[1] Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári, Improved algorithms for linear stochastic bandits, Advances in Neural Information Processing Systems, 2011, pp. 2312–2320.

[2] Sanjeev Arora, Elad Hazan, and Satyen Kale, The multiplicative weights update method: a meta-algorithm and applications, 2012, pp. 121–164.

[3] Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi, Regret in online combinatorial optimization, Mathematics of Operations Research **39** (2013), no. 1, 31–45.

[4] Sebastien Bubeck and Nicolo Cesa-Bianchi, Regret analysis of stochastic and nonstochastic multi-armed bandit problems, Foundations and TrendsÂ® in Machine Learning **5** (2012), no. 1, 1–122.

[5] Wei Chen, Yajun Wang, and Yang Yuan, Combinatorial multi-armed bandit: General framework and applications, Proceedings of the 30th International Conference on Machine Learning (ICML-13) (Sanjoy Dasgupta and David Mcallester, eds.), vol. 28, JMLR Workshop and Conference Proceedings, 2013, pp. 151–159.

[6] Victor Gabillon, Branislav Kveton, Zheng Wen, Brian Eriksson, and S. Muthukrishnan, Adaptive submodular maximization in bandit setting, Advances in Neural Information Processing Systems 26 (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2013, pp. 2697–2705.

[7] Yi Gai, Bhaskar Krishnamachari, and Rahul Jain, Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations, October 2012, pp. 1466–1478.

[8] Aurélien Garivier and Olivier Cappé, The kl-ucb algorithm for bounded stochastic bandits and beyond.

[9] Daniel Golovin and Andreas Krause, Adaptive submodularity: A new approach to active learning and stochastic optimization, CoRR **abs/1003.3967** (2010).

[10] András György, Tamás Linder, and György Ottucsák, The shortest path problem under partial monitoring, Proceedings of the 19th Annual Conference on Learning Theory (Berlin, Heidelberg), COLT'06, Springer-Verlag, 2006, pp. 468–482.

[11] Satyen Kale, Lev Reyzin, and Robert E. Schapire, Non-stochastic bandit slate problems, Advances in Neural Information Processing Systems 23 (NIPS 2010) (J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds.), MIT Press, 2010, pp. 1054–1062.

[12] Sumeet Katariya, Branislav Kveton, Csaba Szepesvári, and Zheng Wen, DCM bandits: Learning to rank with multiple clicks, Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, 2016, pp. 1215–1224.

[13] Branislav Kveton, Csaba Szepesvari, Zheng Wen, and Azin Ashkan, Cascading bandits: Learning to rank in the cascade model, Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (David Blei and Francis Bach, eds.), JMLR Workshop and Conference Proceedings, 2015, pp. 767–776.

[14] Branislav Kveton, Zheng Wen, Azin Ashkan, Hoda Eydgahi, and Brian Eriksson, Matroid bandits: Fast combinatorial optimization with learning., UAI (Nevin L. Zhang and Jin Tian, eds.), AUAI Press, 2014, pp. 420–429.

[15] Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba SzepesvÃ¡ri, Tight regret bounds for stochastic combinatorial semi-bandits., AISTATS (Guy Lebanon and S. V. N. Vishwanathan, eds.), JMLR Workshop and Conference Proceedings, vol. 38, JMLR.org, 2015.

[16] Taishi Uchiya, Atsuyoshi Nakamura, and Mineichi Kudo, Algorithms for adversarial bandit problems with multiple plays., ALT (Marcus Hutter, Frank Stephan, Vladimir Vovk, and Thomas Zeugmann, eds.), Lecture Notes in Computer Science, vol. 6331, Springer, 2010, pp. 375–389.

[17] Zheng Wen, Branislav Kveton, and Azin Ashkan, Efficient learning in large-scale combinatorial semi-bandits., ICML (Francis R. Bach and David M. Blei, eds.), JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, 2015, pp. 1113–1122.

[18] Zheng Wen, Branislav Kveton, and Michal Valko, Influence maximization with semi-bandit feedback, CoRR **abs/1605.06593** (2016).

[19] Li Zhou, A survey on contextual multi-armed bandits, CoRR **abs/1508.03326** (2015).

[20] Shi Zong, Hao Ni, Kenny Sung, Nan Rosemary Ke, Zheng Wen, and Branislav Kveton, Cascading bandits for large-scale recommendation problems, CoRR **abs/1603.05359** (2016).