

BARCELONA GRADUATE SCHOOL  
OF ECONOMICS

MASTERS PROJECT

---

**Combinatorial Bandit  
Algorithms in Practice**

---

*Authors:*

Angus McKAY  
Akhil LOHIA

June 30, 2017

# 1 Abstract

There has recently been a lot of interest in online learning problems where limited feedback is received at each stage of the process, known as bandit problems. The algorithms to solve these problems generally choose a subset of available options to *play* at each round. For the options chosen, some stochastic reward (or cost) is observed. In a semi-bandit situation, the reward is observed for each option which is played. In a full-bandit scenario, only the total reward for all chosen options is observed. Several such practical problems also contain a combinatorial element to them, whereby a combinatorial problem needs to be solved in each round using the partial feedback observed.

Recent literature has proposed many different algorithms and approaches for these bandit problems. The literature is also full of mathematical proofs for bounds on regret risk (i.e., the difference between a chosen solution and the optimal solution). However, there is relatively little information on how these algorithms perform in practice under different scenarios. This project intends to explore the performance of some of the proposed approaches in a more practical setting.

## **2 Acknowledgements**

Thanks

# Contents

<b>1 Abstract</b>	<b>1</b>
<b>2 Acknowledgements</b>	<b>2</b>
<b>3 Introduction</b>	<b>5</b>
3.1 Online learning . . . . .	5
<b>4 Algorithms</b>	<b>8</b>
4.1 CombUCB1 . . . . .	8
4.2 FPL-TRiX . . . . .	10
4.3 CombLinTS . . . . .	11
<b>5 Basic Testing</b>	<b>13</b>
5.1 Basic stochastic setting . . . . .	13
<b>6 Network minimum cost path problems</b>	<b>16</b>
6.1 Stochastic networks . . . . .	16
6.2 Adversarial networks . . . . .	18
6.2.1 Adversarial network 1 - long to short optimal path .	18
6.2.2 Adversarial network 2 - short to long optimal path .	18
6.2.3 Adversarial network 3 - wavelike changes . . . . .	18
<b>7 Parameters</b>	<b>23</b>
7.1 Exploration score . . . . .	23
7.2 Parameters, exploration scores and costs . . . . .	23
7.3 Dynamic $B_t$ parameter . . . . .	24
7.4 Dynamic $\gamma$ parameter . . . . .	25
<b>8 Conclusion</b>	<b>27</b>

# List of Figures

1 CombUCB1 exploration at different stages . . . . .	13
2 Adapted-FPL-TRiX exploration at different stages . . . . .	14
3 CombLinTS exploration at different stages . . . . .	15

4	Optimal paths, CombUCB1, FPL-TRiX and CombLinTS chosen paths . . . . .	17
5	Adversarial network 1 mean path weights at each phases . . . . .	19
6	Adversarial network 2 mean path weights at each phases . . . . .	20
7	Adversarial network 2 - cumulative costs . . . . .	22
8	<b>Exploration score</b> and <b>total cost</b> for different $\eta$ , $\gamma$ and $B_t$ . . . . .	24
9	Cumulative cost for different gamma parameters . . . . .	26

## List of Tables

1	Cost Comparison in Adversarial Networks . . . . .	21
2	Costs for different $\gamma$ . . . . .	25

## 3 Introduction

### 3.1 Online learning

Online learning is a class of machine learning problems where data becomes available sequentially and is used to update the best predictor for future data at each step. This is in contrast to the general approach towards learning problems where the best generator is learned from the entire training dataset at once.

In the online learning environment, a ‘player’ has a selection of options that they can choose from. Once an option is selected the environment provides some cost or reward feedback. The feedback may provide information on all of the available options (i.e., full-information), on only the selected options (semi-bandit) or perhaps only the aggregate of the chosen options (full-bandit). The player then has a decision in the next round, whether to continue with the option chosen in the previous round or to try/explore something different. It is usually a good idea to explore the different options available before exploiting the ones which appear to be the best at the current level of information. There has recently been a lot of interest in developing algorithmic strategies for addressing this exploration vs exploitation trade-off to achieve most efficient results.

In the simplest scenario, the feedback received from all available options is **deterministic**, i.e., the reward from a particular option remains the same in every round. Hence, in this case, as long as the total number of options is not intractable, if one can explore all of the options available then the full environment is known and there is no more learning needed. One can precisely find the optimal choice to play. A more difficult problem is when the feedback is instead **stochastic** in nature. In this scenario, an unknown reward distribution is associated with each option. The observed rewards in each round are realizations of these distributions and this can lead to a significant variation in observations. At each iteration, some feedback on the chosen option provides further insight into the environment, but the actual outcome can never be known with certainty due to the probabilistic nature of the rewards. The player can try to figure out the best option at each iteration in order to reduce/maximize the expected cost/reward. Various algorithms have been proposed for the stochastic situation which allow one to place an upper bound on the *regret* (or sometimes Bayesian regret) of the

system, where regret is defined as the difference in cost/reward between the chosen options and the optimal option.

An even more challenging scenario is when the costs/rewards are **adversarial**. This situation has stochastically drawn feedback from the environment, but the distributions from which the feedback is drawn are not necessarily the same at each iteration. Amazingly, there are algorithms which, given certain constraints on the environment, can still provide upper bounds on the regret risk. These algorithms keep up their exploration aspect in order to adapt to new optimal solutions arising as the situation changes. Hence, the trade-off between exploitation and exploration is pretty evident in order to be flexible enough to adapt to changing scenarios.

To complicate matters, sometimes the problem to be tackled has a combinatorial element to it. This refers to a problem whereby the options to play are chosen subject to satisfying a combinatorial constraint. The algorithms developed to combat such a problem therefore need to be able to include what is commonly referred to as an “optimization oracle”. This can add time and space complexity, but provided the optimization oracle is efficient, the algorithms are in general still efficient.

To provide some motivation for studying such problems, there are various real-world scenarios which fit into the framework outlined above. Perhaps the widest use of online bandit algorithms is in website layout. In particular, the placement of adverts or recommendations on a website. The website provider or advertiser has an array of items from which they can only use a subset at any given time. Once they choose a subset to show to customers they receive feedback on the number of times an advert or recommendation was successful. In the next round, they can then either try some of the other adverts/recommendations or stick with the same ones which were quite successful. This problem can become more combinatorial when there are further restrictions placed on the available options. For example, adverts of different sizes may have to fit onto a website of particular dimensions, or recommendations may be subject to being less than a total stipulated budget.

Another prominent problem that is extensively studied is finding the least cost path in a network, such as a communications network or a road network. The cost of every section of each path may depend on the amount of traf-

fic on that path segment, making the costs non-deterministic. Furthermore, traffic may generally be heavier at certain periods meaning that the expected costs will change over time, giving rise to an adversarial combinatorial bandit problem.

## 4 Algorithms

A stochastic combinatorial semi-bandit is an online learning problem where at each step a learning agent chooses a subset of ground items subject to combinatorial constraints, and then observes stochastic weights of these items and receives their sum as a payoff.

The algorithms shown below are based on the following framework:

There is a set of  $L$  options (known as the *ground set*) that can be selected to form a solution at each stage. There is a set of feasible solutions,  $\Phi$ , which contains combinations of the ground set. For each option there is a distribution  $P$  from which the cost or reward is generated at each iteration.

### 4.1 CombUCB1

One of the most well-known frameworks for multi-armed bandit problems is UCB. This has been adapted to the combinatorial setting by Kveton et al. [2015], who developed CombUCB1. In their paper they developed bounds for the regret risk in a stochastic reward setting. After each iteration the algorithm calculates a confidence bound around the empirical mean reward/cost of each element. These confidence bounds are then used in the combinatorial optimization problem, the solution of which is the chosen play for the next round. The more observations of an element that are available, the tighter the confidence bound around the expected value. Loosely speaking, relatively unobserved items will have larger confidence bounds so they will be explored until there is more confidence in their expected value.

The confidence intervals at time  $t$  are calculated as:

$$\hat{w}_{t-1}(e) \pm c_{t-1,e}$$

where  $\hat{w}_{t-1}(e)$  is the average of the observations of option  $e$  up to time  $t - 1$ .  $c_{t-1,e}$  is calculated as:

$$c_{t-1,e} = \sqrt{\frac{1.5\log(t-1)}{n_{t-1}(e)}}$$

where  $n_{t-1}(e)$  is the number of times that option  $e$  has been observed up to time  $t - 1$ .

The algorithm is set out formally below.

---

### **CombUCB1** algorithm for combinatorial semi-bandit problems

---

*Initialization:*

$$\begin{aligned}\hat{\mathbf{w}}_0 &\leftarrow \text{make one observation of each edge} \\ n_0(e) &\leftarrow 1 \quad \forall e \in E\end{aligned}$$

*For all  $t$  in  $1, \dots, T$ :*

Compute confidence intervals for each edge:

$$U_t(e) \leftarrow \hat{w}_t(e) \pm c_{t-1,e} \quad \forall e \in E$$

Solve the optimization problem:

$$A_t \leftarrow \arg \max_{A \in \Theta} f(A, U_t)$$

( $f$  is the total reward from solution set  $A$  when the current Upper Confidence Bounds are  $U_t$ )

Observe the weights of chosen items:

$$\begin{aligned}w_t(e) &\sim P_e \quad \forall e \in A_t \\ w_t(e) &= 0 \text{ otherwise}\end{aligned}$$

Update counts and expected values for each edge:

$$\begin{aligned}n_t(e) &\leftarrow n_{t-1}(e) + 1 \quad \forall e \in A_t \\ \hat{w}_t(e) &\leftarrow \frac{n_{t-1}(e)\hat{w}_{t-1}(e) + w_t(e)}{n_t(e)} \quad \forall e \in E\end{aligned}$$


---

This assumes that one is able to initialize by choosing to observe each edge at least once before starting the algorithm. In practice if this is not feasible then the algorithm will produce a similar output if the initial weight for each edge is assumed to be the maximum (if a maximization problem) or minimum (if a minimization problem) values of the system (but these initial values are then disregarded in the algorithm once an observation has been made).

Note that the algorithm is computationally efficient, provided that the combinatorial optimization can be solved efficiently. Hence, the part of selecting the optimal  $A_t$  in each iteration, given the current upper confidence

bounds should be a problem which can be solved efficiently. All other operations are polynomial in L and K.

## 4.2 FPL-TRiX

The follow the perturbed leader (FPL) algorithm has been adapted to a combinatorial setting by Neu [2015]. This paper introduces FPL-TRiX, which is a Follow the Perturbed Leader algorithm with TRuncated perturbations and Implicit eXploration. The paper proves regret bounds in the case of an oblivious adversary (i.e. the losses at each iteration are chosen independently of what has happened in the past). In the algorithm, each time an option is observed an amount is added on to a cumulative total for that particular option. This amount is based on the cost of the observed edge at that iteration and the proportion of times that the options has been selected in past iterations. This cumulative total is then used in the combinatorial optimization problem in the next iteration. The fact that the cumulative total increases only for the observed options means that eventually the total of the lower cost edges will always increase above those for the higher cost edges. Thus the higher cost edges will always eventually be explored again. This continued exploration should allow the algorithm to adapt more quickly if the situation changes, i.e., in an adversarial framework.

The algorithm is set out formally below.

---

### FPL-TRiX algorithm for combinatorial semi-bandit problems

---

*Parameters:*

Learning rates  $\eta_t$ , exploration parameters  $\gamma_t$ , truncation parameters  $B_t$

*Initialization:*

$$\begin{aligned}\hat{\mathbf{L}}_0 &= 0 \\ n_0(e) &\leftarrow 1 \quad \forall e \in E\end{aligned}$$

*For all  $t$  in  $1, \dots, T$ :*

Draw perturbation vector  $\mathbf{Z}_t$  with each component drawn from the exponential distributions truncated at  $B_t$

Solve the optimization problem:

$$A_t \leftarrow \arg \min_{A \in \Theta} (\eta_t \hat{\mathbf{L}}_{t-1} - \mathbf{Z}_t)$$

Observe the weights of chosen items:

$$w_t(e) \sim P_e \quad \forall e \in A_t$$

$$w_t(e) = 0 \text{ otherwise}$$

Compute  $\hat{l}_{t,i}$  as:

$$\hat{l}_{t,i} = \frac{w_t(e)}{q_{t,i} + \gamma_t} \text{ otherwise}$$

where  $q_{t,i} = \mathbf{E}[A_{t,i}]$  based on observations up to t-1

Update  $\hat{\mathbf{L}}_t$ :

$$\hat{\mathbf{L}}_t = \hat{\mathbf{L}}_{t-1} + \hat{\mathbf{l}}_{t,i}$$


---

As with CombUCB1, here as well it is assumed that the solution to the optimization/combinatorial part is efficiently available.

### 4.3 CombLinTS

The Combinatorial Linear Thompson Sampling is a modification of the original work of Thompson [1933] by Wen et al. [2015] in a combinatorial setting. This kind of setting is mostly favorable when the total available choices  $L$  is intractably large. However, similar options in such large sets will tend to have similar features which can help make good decisions more efficiently. The *linear generalization* approach to such settings involve modeling the available options and the rewards/costs associated with them as a linear model which outputs the corresponding rewards/costs. The coefficients of such a model are essentially learned by the algorithm by way of exploration and exploitation whereby the estimates for the coefficients improve with the number of iterations. Hence, conditioned on the features of an item, the expected weight of that item can be estimated using a linear model. The problem setup is the same as in the previous 2 algorithms where we must choose at max  $K$  items from a ground set of  $L$  options. We allow the agent to use any approximation/randomized algorithm ORACLE to solve the combinatorial part of the problem.

The algorithm is set out formally below:

---

**Algorithm 1** CombLinTS algorithm

---

**Input:** Combinatorial structure ( $E, \mathcal{A}$ ), generalization matrix  $\Phi \in \mathbf{R}^{L \times d}$ , parameters  $\lambda, \sigma > 0$ , oracle  
 Initialize  $\Sigma_1 \leftarrow \lambda^2 I \in \mathbf{R}^{d \times d}$  and  $\bar{\theta}_1 = 0 \in \mathbf{R}^d$   
**for all**  $t = 1, 2, \dots, n$  **do**  
     Sample  $\theta_t \sim N(\bar{\theta}_t, \Sigma_t)$   
     Compute  $A^t \leftarrow \text{oracle}(E, \mathcal{A}, \Phi\theta_t)$   
     Choose options  $A^t$ , and observe corresponding weights for chosen items  
     Update  $\bar{\theta}_{t+1}$  and  $\Sigma_{t+1}$  based on Kalman Filtering  
**Output:** Chosen items in each iteration

---

Based on this algorithm, updated weights are achieved in every iteration which tend to get closer to the optimal solutions. However, this updatation is highly influenced by the choice of parameters  $\lambda$  and  $\sigma$  which control the exploration or the learning rate. Specifically,  $\lambda$  is an inverse-regularization parameter and smaller  $\lambda$  makes the covariance matrix closer to 0 which leads to slower learning. Thus, a small  $\lambda$  can lead to less exploration and poor performance of the algorithm.  $\sigma$  on the other hand, handles the decrease rate of the covariance matrix during updatation in the Kalman-Filtering step. A large  $\sigma$  ensure more exploration whereas a small value makes the algorithm quickly converge to some sub-optimal coefficient vector.

Hence, CombLinTS makes a fine balance of exploration through a combination of 2 parameters. Depending on the situation, smaller or larger values will provide favorable results. Although, at the same time, this also implies that finding the perfect balance in adversarial scenarios can prove really challenging.

## 5 Basic Testing

### 5.1 Basic stochastic setting

For some initial insight into how the algorithms work in practice we can set up a basic simulation with a trivial combinatorial element. In this scenario we will have a vector of length  $L$ , and at each round  $K$  elements of this vector will be chosen. The weights of each chosen element will then be drawn from a Bernoulli distribution with a different mean for each element. A real-life application of such a system could be arranging adverts on a website where only  $K$  out of  $L$  adverts can be shown at any one time, and the goal is to maximize the number of times a user clicks on a link in an advert.

We start by setting  $L = 20$ ,  $K = 5$  and set the means as incrementally increasing values from  $1/20$  for the first element to 1 for the last element (so they are  $1/20, 2/20, \dots, 18/20, 19/20, 1$ ). The game is to maximize the weights observed at each round. Clearly the optimal elements to play at each round are the last 5 elements as they have the highest probability of being drawn as 1. Let us observe how the algorithms explore the different options at different stages.

#### CombUCB1

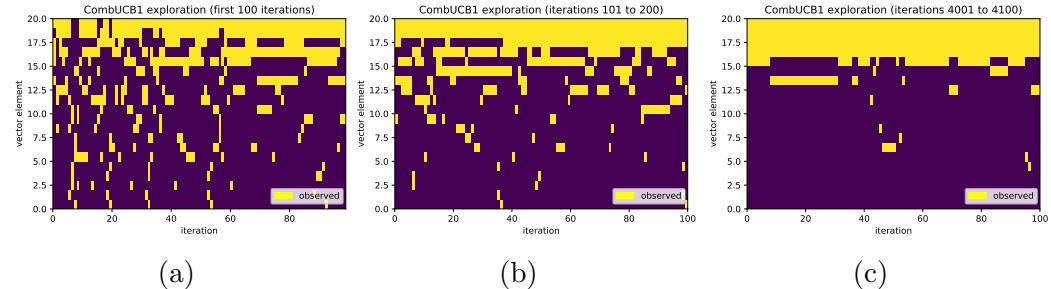


Figure 1: CombUCB1 exploration at different stages

Figure 1(a) shows how at the outset, the algorithm is not confident what the expected value of any of the weights are and so it explores a lot. Figure 1(b) shows iterations 101 to 200, when the algorithm is starting to learn which

elements have the highest expected value. Finally Figure 1(c) shows the algorithm after 4000 iterations at which time it has become quite confident which are the optimal paths, and does relatively little exploration.

The regret risk can be calculated as the sum of observed weights minus the sum of the weights of the top 5 elements of the vector. For 10,000 iterations this gives the following:

$$\begin{aligned}\text{Optimal path reward} &= 42,399 \\ \text{Actual path reward} &= 41,905 \\ \text{Regret} &= 494\end{aligned}$$

The algorithm performs very well at finding a near optimal solution.

### Adapted-FPL-TRiX

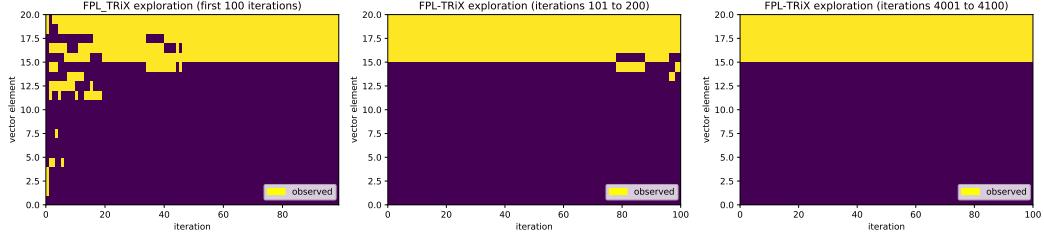


Figure 2: Adapted-FPL-TRiX exploration at different stages

Figure 2 shows the corresponding charts for FPL-TRiX but adapted to find a maximum solution as opposed to a minimum solution. In fact, by making this change to find a maximum, the essence of FPL-TRiX is missed, which is that it adapts well in adversarial cases by keeping more of an open mind about trying the different options even once it has started to recognize the optimal solution. However, even though this is not exactly FPL-TRiX, it is interesting to include here as, with a bit of parameter tuning, it performs incredibly well in this situation. The reward and regret are:

$$\begin{aligned}\text{Optimal path reward} &= 42,399 \\ \text{Actual path reward} &= 42,246\end{aligned}$$

Regret = 153

Thus the algorithm outperforms CombUCB1. If we think of this slightly modified FPL-TRiX as part of the family of FPL-TRiX then it shows that this can be a nice adaptable framework to work in.

## CombLinTS

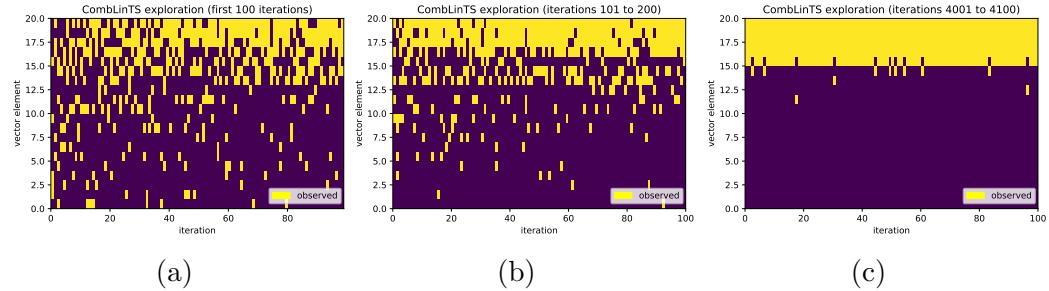


Figure 3: CombLinTS exploration at different stages

Figure 3(a) and 3(b) show how the algorithm is not confident of the best paths and weights and so it explores a lot. Figure 3(c) shows that after a few thousand iterations, the algorithm is confident that the last few arms are best ones which corresponds well to the problem setup and the exploration proportion goes down significantly.

The regret risk can be calculated as the sum of observed weights minus the sum of the weights of the top 5 elements of the vector. For 10,000 iterations this gives the following:

Optimal path reward = 42,399  
 Actual path reward = 41,900  
 Regret = 499

The algorithm performs very well at finding a near optimal solution but after several iterations so it needs time. Although it doesn't get stuck on a similar suboptimal solution so it's rewarding in the long run.

## 6 Network minimum cost path problems

A more sophisticated combinatorial problem to solve is a minimum cost path problem in a network. In this framework there are  $V$  vertices with edges in both directions between each pair of vertices, giving  $V \times (V-1)$  possible edges in total (in practice if an edge does not exist between two vertices then the cost of the edge can be set to infinity). A path is chosen to travel from some starting vertex to an end vertex. The cost of each edge in the path is then stochastically drawn and the sum of these provides the total cost of the chosen path. The goal is to find the path with minimum expected cost. In the following setup we study the minimum cost path from the first vertex to the last vertex of the graph where the weights are generated according to different criteria as described further. Clear application of this type of system in real-life could be any type of mass scale distribution such as electricity or communication networks.

Given the more difficult combinatorial problem, this pushes the algorithms harder. They will need to work out which higher cost edges are beneficial to use in order to take advantage of low cost paths than lead into or out of them. Thus the relative level of regret should be much higher than the previous more basic problem.

### 6.1 Stochastic networks

Firstly we look at how CombUCB1, FPL-TRiX and CombLinTS perform in the stochastic setting, where the weights of the edges are drawn from truncated normal distributions limited to being between 0 and 1. Each edge has a different central point (i.e., different mean), and the variance was kept low at 0.1 so that there is a meaningful difference between the different paths.

With 10 vertices and observing 10,000 iterations the following costs were observed. This was after tuning the FPL-TRiX and CombLinTS parameters, in particular reducing the  $\gamma$  parameter in FPL-TRiX to 0.01.

Optimal path cost = 3,517  
CombUCB1 path cost = 4,989  
FPL-TRiX path cost = 4,217

$$\text{CombLinTS} = 3,553$$

Perhaps surprisingly, FPL-TRiX is able to significantly outperform CombUCB1 in this situation. Figure 4 shows the edges of the optimal path at each iteration, and the edges of the paths chosen by the 3 algorithms. FPL-TRiX explores more at first allowing it to find the optimal path more quickly, whereas CombUCB1 is led towards shorter paths at first. CombLinTS is tuned for less exploration and quickly latches on to the best optimal path and little exploration thereafter. If all edges start with similar wide confidence intervals then the paths with the least number of edges will appear to have the lowest costs. It is only after some time (around 200 iterations in this case) that CombUCB1 finds accurate enough estimates of the edges to work out that a longer path is in fact the optimal.

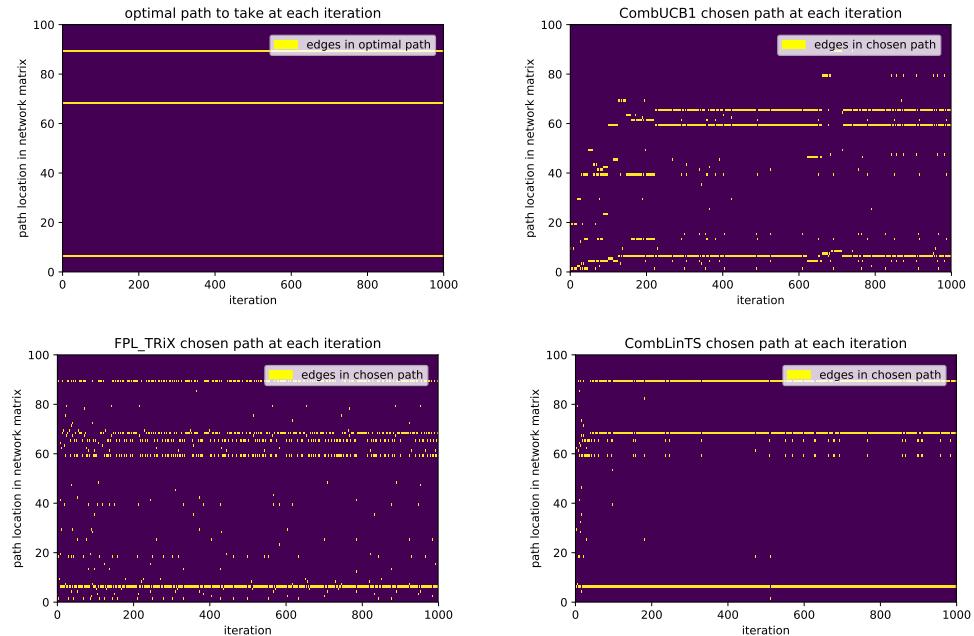


Figure 4: Optimal paths, CombUCB1, FPL-TRiX and CombLinTS chosen paths

## 6.2 Adversarial networks

In this section we will progress to analyzing the performance of the algorithms in an adversarial setting. To do this we continue the minimum cost path setting of the previous section, but allow the weights of the edges to be drawn from different distributions over different periods of time. The following three different settings were investigated:

### 6.2.1 Adversarial network 1 - long to short optimal path

The first adversarial network we analyze starts with the optimal path being to visit each vertex sequentially from the first vertex to the last. Thus with 10 vertices the initial optimal path is nine edges long. This is setup by having the minimum means for these edges when generating the weights. After 5,000 iterations the optimal path changes to a shorter path of only two edges. The mean weights of the edges are shown in the network matrices in Figure 5. Element  $(i,j)$  of each matrix represents the mean of the truncated normal distribution that the weight of the edge from vertex  $i$  to  $j$  is drawn from at each iteration. In Figure 5a the shortest path is from vertex 0 to 3 and then from 3 to 9 in Figure 5b.

### 6.2.2 Adversarial network 2 - short to long optimal path

The second adversarial network is similar to the first but in reverse, whereby at first the optimal path is the shortest path, i.e., one edge straight from the first to the last vertex. After 1,000 iterations the optimal paths changes to one with many edges. The network alternates between these two situations every 1,000 iterations. The mean weights of the edges are shown in Figure 6.

### 6.2.3 Adversarial network 3 - wavelike changes

In this final adversarial case, the mean edge weights increase and decrease gradually over time. Each mean edge weight is essentially a point on a sine wave but shifted up by 1 so that it is always positive and divided by 2 so that the mean edge values fall between 0 and 1. The mean edge weights shift along the sine wave over time so that they gradually increase and decrease.

	0	1	2	3	4	5	6	7	8	9
0	0.008	0.050	0.150	0.250	0.350	0.450	0.550	0.650	0.750	0.850
1	0.050	0.008	0.050	0.150	0.250	0.350	0.450	0.550	0.650	0.750
2	0.150	0.050	0.008	0.050	0.150	0.250	0.350	0.450	0.550	0.650
3	0.250	0.150	0.050	0.008	0.050	0.150	0.250	0.350	0.450	0.550
4	0.350	0.250	0.150	0.050	0.008	0.050	0.150	0.250	0.350	0.450
5	0.450	0.350	0.250	0.150	0.050	0.008	0.050	0.150	0.250	0.350
6	0.550	0.450	0.350	0.250	0.150	0.050	0.008	0.050	0.150	0.250
7	0.650	0.550	0.450	0.350	0.250	0.150	0.050	0.008	0.050	0.150
8	0.750	0.650	0.550	0.450	0.350	0.250	0.150	0.050	0.008	0.050
9	0.850	0.750	0.650	0.550	0.450	0.350	0.250	0.150	0.050	0.008

(a) Adversarial network 1 - initial mean edge weights

	0	1	2	3	4	5	6	7	8	9
0	0.008	0.277	0.425	0.142	0.323	0.988	0.433	0.540	0.323	0.992
1	0.693	0.008	0.714	0.280	0.399	0.796	0.021	0.492	0.865	0.799
2	0.615	0.745	0.008	0.269	0.282	0.593	0.192	0.593	0.020	0.804
3	0.674	0.107	0.848	0.008	0.656	0.895	0.207	0.716	0.210	0.146
4	0.964	0.371	0.868	0.426	0.008	0.185	0.179	0.667	0.888	0.887
5	0.583	0.438	0.749	0.892	0.643	0.008	0.626	0.910	0.635	0.533
6	0.107	0.737	0.053	0.671	0.583	0.349	0.008	0.154	0.191	0.217
7	0.947	0.202	0.611	0.243	0.362	0.287	0.815	0.008	0.204	0.200
8	0.144	0.690	0.283	0.412	0.798	0.604	0.362	0.162	0.008	0.983
9	0.280	0.848	0.660	0.705	0.495	0.614	0.246	0.862	0.533	0.008

(b) Adversarial network 1 - second phase mean edge weights

Figure 5: Adversarial network 1 mean path weights at each phases

The edges are positioned on the sine wave relative to each other dependent on how close the vertices are which they connect. For example, the edges between vertices 1 and 4 and between vertices 5 and 8 will be at the same position on the sine wave (because their vertices are labeled 3 numbers apart), but they will be far away from the edge which connects vertices 2 and 9.

The idea of this adversarial case is that it represents gradual shifts over time. Each cycle of the sine wave could be thought of as day and night periods for example, where the edges represent the cost of a communication path. During the days the costly edges would be those which connect businesses which are active during the day, and at night the costly edges switch to those which connect homes which become busy as people return from work.

The following table shows the total cost of each algorithm for each of the networks. It also shows the cost for the first 1,000 iterations, the middle

	0	1	2	3	4	5	6	7	8	9
0	0.008	0.350	0.450	0.550	0.650	0.750	0.850	0.950	0.998	0.999
1	0.350	0.008	0.350	0.450	0.550	0.650	0.750	0.850	0.950	0.998
2	0.450	0.350	0.008	0.350	0.450	0.550	0.650	0.750	0.850	0.950
3	0.550	0.450	0.350	0.008	0.350	0.450	0.550	0.650	0.750	0.850
4	0.650	0.550	0.450	0.350	0.008	0.350	0.450	0.550	0.650	0.750
5	0.750	0.650	0.550	0.450	0.350	0.008	0.350	0.450	0.550	0.650
6	0.850	0.750	0.650	0.550	0.450	0.350	0.008	0.350	0.450	0.550
7	0.950	0.850	0.750	0.650	0.550	0.450	0.350	0.008	0.350	0.450
8	0.998	0.950	0.850	0.750	0.650	0.550	0.450	0.350	0.008	0.350
9	0.999	0.998	0.950	0.850	0.750	0.650	0.550	0.450	0.350	0.008

(a) Adversarial network 2 - initial mean edge weights

	0	1	2	3	4	5	6	7	8	9
0	0.992	0.050	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992
1	0.992	0.992	0.050	0.992	0.992	0.992	0.992	0.992	0.992	0.992
2	0.992	0.992	0.992	0.050	0.992	0.992	0.992	0.992	0.992	0.992
3	0.992	0.992	0.992	0.992	0.050	0.992	0.992	0.992	0.992	0.992
4	0.992	0.992	0.992	0.992	0.992	0.050	0.992	0.992	0.992	0.992
5	0.992	0.992	0.992	0.992	0.992	0.992	0.050	0.992	0.992	0.992
6	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.050	0.992	0.992
7	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.050	0.992
8	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.050
9	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992

(b) Adversarial network 2 - second phase mean edge weights

Figure 6: Adversarial network 2 mean path weights at each phases

1,000 (when networks 1 and 2 shift) and the last 1,000 to show how each algorithm performs at the different stages.

In the first adversarial network FPL-TRiX is quicker to identify the long optimal path at the beginning (similar to the result in the stochastic setting). CombLinTS performs better than both CombUCB1 and FPL-TRiX in this case due to its greater flexibility in terms of tuning its exploration parameters. All algorithms move towards the optimal path in the second phase, but CombUCB1 explores less in this stage which ultimately leads to it returning a lower total cost.

The second adversarial framework tries to exploit the fact that CombUCB1 can be slower than FPL-TRiX to adapt to new paths. Indeed this seems to be the case as it performs worse. The cost at the end is less with

Costs of each algorithm over each adversarial network				
Network	Optimal Path	CombUCB1	FPL-TRiX	CombLinTS
Adversarial 1				
- total cost	4,711	7,228	9,313	5,907
- first 1000 iters.	450	750	673	482
- middle 1000 iters.	472	984	1,071	1,488
- final 1000 iters.	492	667	1,075	494
Adversarial 2				
- total cost	7,244	12,447	11,890	10,845
- first 1000 iters.	999	1,803	1,321	1,485
- middle 1000 iters.	723	1,087	1,142	1,024
- final 1000 iters.	450	457	1,079	997
Adversarial 3				
- total cost	3,844	5,482	6,262	9,489
- first 1000 iters.	503	809	1,007	1,338
- middle 1000 iters.	178	351	296	406
- final 1000 iters.	781	964	1,245	1,594

Table 1: Cost Comparison in Adversarial Networks

CombUCB1 but this is a result of the end period being the state with the optimal path that CombUCB1 has got stuck in. Looking at the 1,000 iterations prior to this then CombUCB1’s cost is 1,995 and FPL-TRiX’s cost is 1,150. Figure 7 shows the progress of the costs of each algorithm over time. It is clear that CombUCB1 goes through phases of slowly increasing cost for 1,000 iterations and then rapidly increasing cost for 1,000 iterations. Not surprisingly, CombLinTS can be tuned for more exploration in this scenario and performs better than the other 2 algorithms. This provides evidence to say that CombLinTS is the best choice in cases where some prior information about the network structure can be obtained and the feature matrix can be exploited as well.

Finally, when the situation is gradually evolving like in the third case, CombUCB1 never becomes too confident on any paths and so it keeps exploring more and performs well again compared to FPL-TRiX. CombLinTS is not very successful in this scenario due to greater uncertainty. An *exploita-*

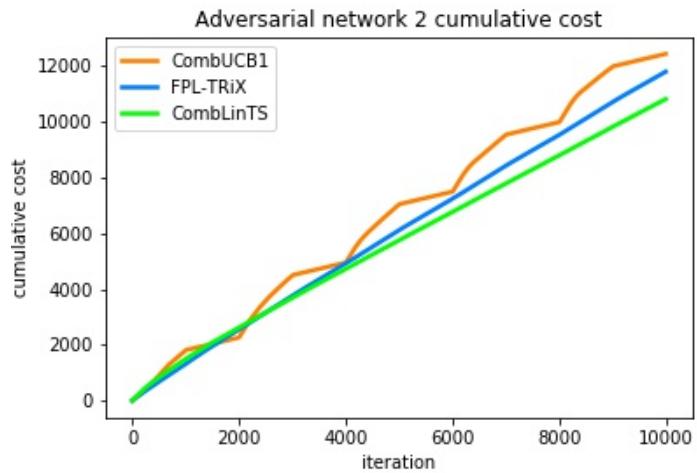


Figure 7: Adversarial network 2 - cumulative costs

*tion* approach can be beneficial in such scenarios depending on how large the gap between the optimal and best suboptimal path is.

## 7 Parameters

From the analysis so far, some benefits of FPL-TRiX over CombUCB1 are that it can identify optimum paths quicker when the minimum cost path is longer, and that it is less prone to becoming stuck in adversarial situations where the the path distributions change relatively quickly. These advantages rely on the ability to tune the parameters in FPL-TRiX to best suit the situation. In the stochastic setting it is beneficial to adjust the parameters to reduce the amount of exploration the algorithm does. In the adversarial case, however, it should help to keep exploration relatively more active in order to identify when the situation changes and there is a new optimal path. In this section we look at the parameters specifically and understand how changing them affects outcomes.

### 7.1 Exploration score

To measure the amount of exploration an algorithm is doing we introduce an “impurity” function. This calculates the proportion of iterations in which each edge is chosen. This value is then multiplied by 1 minus itself (so if the value is  $p$  then the calculation is  $p \times (1-p)$ ) and then multiplied by 4 so that the possible values range between 0 and 1. Finally the average of these values is taken over all the edges and multiplied by 1,000 so that the numbers are not tiny.

### 7.2 Parameters, exploration scores and costs

To see how the parameters affect exploration and cost let us first observe the performance of FPL-TRiX in the stochastic network. Figure 8 shows how the total cost and level of exploration change as the different parameter values change. The main point to note here is that when there is less exploration the algorithm performs better.

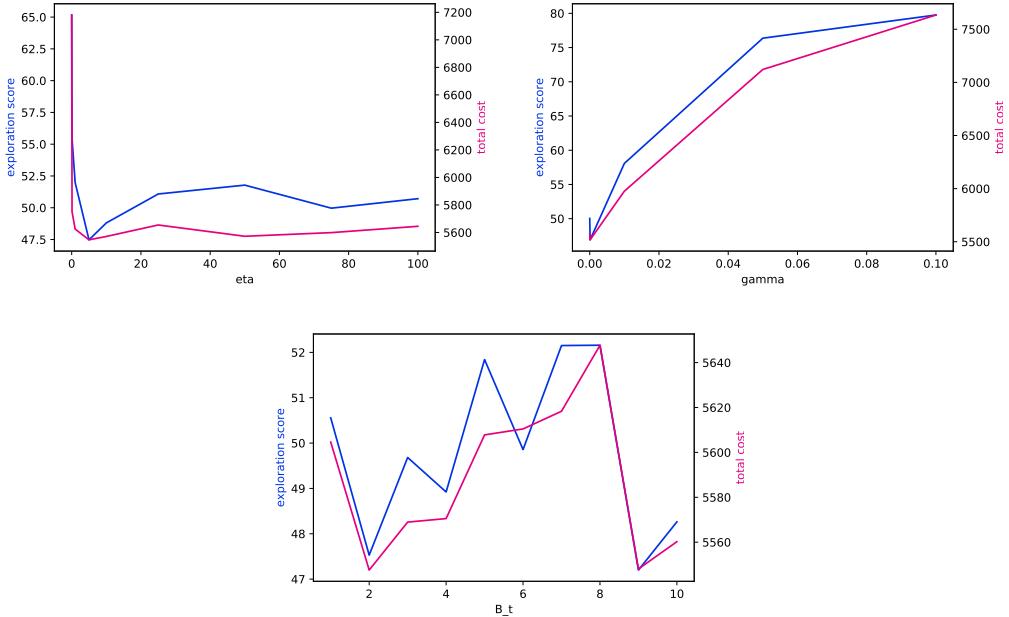


Figure 8: Exploration score and total cost for different  $\eta$ ,  $\gamma$  and  $B_t$

### 7.3 Dynamic $B_t$ parameter

In Neu [2015] it is proposed that the  $B_t$  parameter in FPL-TRiX is non-increasing, but is variable over time.  $B_t$  is a control on the amount that the algorithm explores - the higher  $B_t$  is, the larger the perturbations are able to be and therefore it is more likely for “worse” performing edges to be perturbed to a value where they are chosen in an iteration. Therefore, one idea to improve the performance of FPL-TRiX in adversarial cases is to allow the  $B_t$  parameter to adjust to the situation. In particular, if the situation changes then the parameter could be increased to encourage exploration to find a new optimal path, after which the parameter would be decreased again.

These parameter adjustments could be carried out with some prior knowledge of the system. For example, for communications networks, this could be at the start and at the end of normal working hours, or at the beginning of seasons or holiday periods. Another way to adjust the parameter could be to allow it to somehow recognize itself when the situation changes. For example if the variance of the reward/cost at each iteration changes.

Testing out this idea in the adversarial cases previously discussed, however, bares little fruit. The main reason for this is that  $B_t$  makes little to no difference on the outcome of the algorithm. In fact the main effect it has is to increase the variance of the result. This can be observed in Figure 8(c) whereby changing the  $B_t$  parameter has a seemingly unpredictable effect on the cost.

## 7.4 Dynamic $\gamma$ parameter

Given that, in practice, the  $B_t$  does not have a clearly observable effect on the outcome of FPL-TRiX in an adversarial network, perhaps allowing some of the other parameters to vary could yield better results. Testing the cost at different periods with different  $\gamma$  parameters yields the following:

Costs at different periods for differnt $\gamma$ parameters					
Cost over iterations:	1-1000	4000-5000	5000-6000	9000-10000	total cost
$\gamma = 0.01$	694	626	1,266	821	8,397
$\gamma = 0.1$	725	716	1,037	699	7,735

Table 2: Costs for different  $\gamma$

The lower  $\gamma$  parameter provides the best results in the early stages but then once the system changes, the higher  $\gamma$  parameter yields better results. So can changing the  $\gamma$  parameter over time reduce the total cost returned? It seems like a parameter of 0.01 performs best at first and then changing to 0.1 is better for the second stage. However, in reality simply changing the parameter in this way does not lead to the best performance here. The behavior over time of each scenario can be observed in Figure 9.

The total cost seems to be largely determined by the settings at the initial stages. Although changing the parameter when the adversarial situation changes does appear to be advantageous compared to keeping it at 0.01, it still lags the performance when gamma is set to 0.1 for the entire time. It does, however, suggest that using some sort of changing  $\gamma$  parameter could

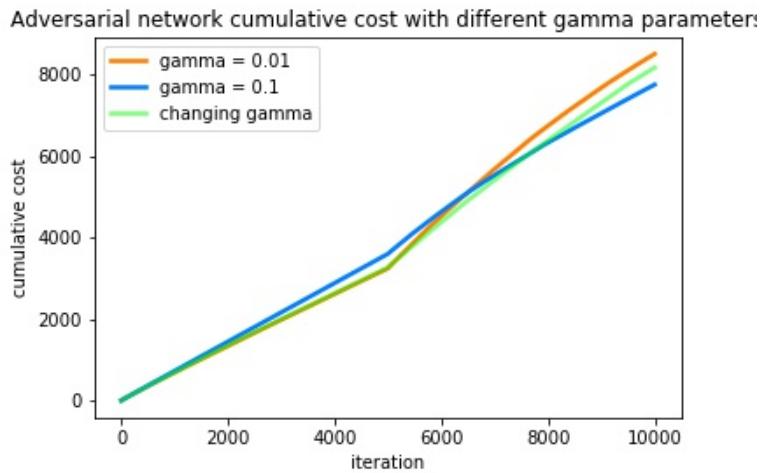


Figure 9: Cumulative cost for different gamma parameters

be beneficial in some cases, and indeed in this case the changing parameter model is lowest cost after around 6,000 iterations.

## 8 Conclusion

The algorithms discussed in this paper have been developed with either the stochastic or adversarial framework in mind. For CombUCB1 and CombLinTS the papers which introduce them analyse regret bounds in the stochastic setting, whereas FPL-TRiX has bounds which hold in an adversarial setting. However, given the evidence in this paper, this does not mean that the algorithms are strictly better in these particular settings. There are advantages and disadvantages to each, in particular CombUCB1 seems to lag behind the others in terms of adapting quickly to a new environment, but its persistence on the optimal path once it finds it can lead it to outperform the other algorithms even in some adversarial cases.

In the stoachastic setting, the ability to tune parameters for FPL-TRiX and CombLinTS allow the exploration element of these algorithms to be minimised, and this leads to good performance. In the adversarial setting the exploration can then be increased to help the algorithms adapt when the situation changes.

## References

- Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba Szepesvari. Tight regret bounds for stochastic combinatorial semi-bandits. In *18th International Conference on Artificial Intelligence and Statistics*, 2015.
- Gergely Neu. First-order regret bounds for combinatorial semi-bandits. In *The 28th Annual Conference on Learning Theory (COLT 2015)*, pages 1360–1375, 2015.
- W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933.
- Zheng Wen, Azin Ashkan, Hoda Eydgahi, and Branislav Kveton. Efficient learning in large-scale combinatorial semi-bandits. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 767–776, 2015.