December 2017

ONLINE LEARNING WITH BANDITS FOR COVERAGE

Mahmuda Rahman
Syracuse University

ABSTRACT

With the rapid growth in velocity and volume, streaming data compels decision support systems to predict a small number of unique data points in due time that can represent a massive amount of correlated data without much loss of precision. In this work, we formulate this problem as the *online set coverage problem* and propose its solution for recommendation systems and the patrol assignment problem.

We propose a novel online reinforcement learning algorithm inspired by the Multi-Armed Bandit problem to solve the online recommendation system problem. We introduce a graph-based mechanism to improve the user coverage by recommended items and show that the mechanism can facilitate the coordination between bandits and therefore, reduce the overall complexity. Our graph-based bandit algorithm can select a much smaller set of items to cover a vast variety of users choices for recommendation systems. We present our experimental results in a partially observable real-world environment.

We also study the patrol assignment as an online set coverage problem, which presents an additional level of difficulty. Along with covering the susceptible routes by learning the diversity of attacks, unlike in recommendation systems, our technique needs to make choices against actively engaging adversarial opponents. We assume that attacks over those routes are posed by intelligent entities, capable of reacting with their best responses. Therefore, to model such attacks, we used the Stackelberg Security Game. We augment our graph-based bandit defenders with adaptive adjustment of reward coming from this game to perplex the attackers and gradually succeed over them by maximizing the confrontation. We found that our graph bandits can outperform other Multi-Arm bandit algorithms when a simulated annealing-based scheduling is incorporated to adjust the balance between exploration and exploitation.

ONLINE LEARNING WITH BANDITS FOR COVERAGE

by

Mahmuda Rahman

Bachelors, North South University, 2004

Masters, University of Dhaka, 2006

Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer & Information Science & Engineering.

Syracuse University

December 2017

To my family and friends

ACKNOWLEDGMENTS

I would like to begin by thanking the Almighty for giving me the strength and knowledge to pursue this degree. Thanks to Allaah for being the best counselor and comforter in all my difficult times. I simply can't count His blessings on me, so I won't even try. Next, I thank my parents, who kept me going, by believing in me. I would like to thank my whole family for being supportive throughout the entire process of finishing this. I can never forget the sacrifice they made to see their girl succeed in her mission.

My advisor, Dr. Jae Oh, who has been my mentor in both research and academic pursuits, has shown extreme patience with me. He taught us the importance of meaningful communication and logical thinking. Research freedom that we enjoy under his supervision is actually a challenge he trains us to handle. Dr. Mohan, Dr, Varshney, Dr. Soundarajan, Dr. Yu and Dr. Bhatia - each was a great mentor to me and blessed me with their affection. I also want to acknowledge Dr. Hasan of IUPUI, Dr. Gene Zhang of Huawei Labs and Nish Parikh of eBay Research for their direction and guidance. I want to thankfully mention the help and cooperation of my manager Gyanit Singh at work.

A very special thanks to my partner in this noble 'crime' - Qinyun Zhu - the candid witness of my struggles and efforts, in and outside the lab. Our outstanding camaraderie gave so many unforgettable memories which I will always hold dear. I thank him for all the tears and laughter, fights and joys... as I hope to continue on. I am grateful to Mina, Joo, Nate, Jeff, Zhi, Zilong and Chris for their friendship. I couldn't have asked for more.

TABLE OF CONTENTS

Page

ABSTRACT i

LIST OF FIGURES ix

1 Introduction 1

1.1 Problem Definition 2

1.2 Related Problems 5

1.3 Overview of this thesis 7

2 Background 9

2.1 Stochastic Multi Armed Bandit 9

2.2 Upper Confidence Bound (UCB) 10

2.3 Related Works 13

2.3.1 Application in Recommendation Systems 13

2.3.2 Application in Dynamic Patrol Allocation 15

3 Online Recommendation System 17

3.1 Reward Bias 19

3.1.1 Initial Results 21

3.2 Graph Based UCB1 24

3.2.1 Motivation for Graph Representation 26

3.2.2 Construction of Relevance Item Graph 29

3.2.3 Discounted Reward Based on Item Correlation 31

3.2.4 Annotation of Arms States 33

3.2.5 State Transitions of Arms 34

3.3 Empirical Evaluation 39

3.3.1 Impact of Sparsity in User Vector 40

3.3.2 Coverage of Different Sets 40

Page

3.3.3 Diversity of Recommended Set 41
3.3.4 Reduction of Arm Space 42
3.4 Parallel & Synchronized UCB2 45
3.4.1 Technical Challenges 46
3.4.2 Solution Sketch 50
3.4.3 Analysis of PAC Bound 52
3.4.4 Proposed Algorithm 55
3.4.5 Empirical Evaluation 58
4 Dynamic Patrol Assignment 63
4.1 Game Theoretic Formulation 64
4.2 Comparison with TSP and the VRP 66
4.3 Patrolling Robots as UCB1 Bandits 67
4.4 Stackelberg Security Game (SSG) for Route Selection 69
4.5 Learning from the Best Response Opponent 72
4.6 Balancing Exploration and Exploitation 75
4.6.1 Active and Dormant Routes 76
4.6.2 Transitions of Routes 77
4.6.3 Randomized Algorithm 79
4.6.4 Simulation Results 80
5 Summary and Future Directions 89
5.1 Capturing Diversity in Online Learning 89
5.1.1 Introducing Graph Based Bandits 90
5.1.2 Parallel and Synchronized Execution of Bandits 91
5.2 Online Learning with Adversarial Game 93
5.3 Future Directions 94
5.3.1 Distributed Graph Bandits 95
5.3.2 Dynamic Addition of Arms 95
5.3.3 Learning against Complex Heterogeneous Attacks 96
5.3.4 Attackers Learning against Defenders 96

Page

LIST OF REFERENCES 97

VITA 103

LIST OF FIGURES

Figure Page

3.1 Performance comparison of recommending 3 jokes out of 10 from Jester dataset, Y-axis showing x1000 times iteration, new and old denoting the performance of with and without Graph based approach 22

3.2 Performance comparison of recommending 5 jokes out of 10 from Jester dataset, Y-axis showing x1000 times iteration, new and old denoting the performance of with and without Graph based approach 23

3.3 Performance comparison of RBA, IBA and Reward bias of recommending 10 movies out of 1700 from MovieLens dataset of around 1000 users. 23

3.4 Limitation of Non Graph Based method where Ti labels the 2 item recommendation set at i^{th} time step coming from 2 bandits b1 and b2 and u_i denotes the user vector it is recommended to 25

3.5 Emergence of item 4 with increasing weight as a potential replacement of item 2 in the graph based method to accommodate user type 3 as in the illustrative example mentioned in Figure 2 30

3.6 Comparison result with different high and low θ values for Movielens data . 39

3.7 Average Coverage of each Item in Recommendation Set after every 10K rounds for dense Movielense dataset with low threshold using RBA, IBA, Graph Bandit compared with with offline greedy approach 40

3.8 Normalized ILD of the Recommendation Set after every 10K rounds for dense Movielense dataset with low threshold using RBA, IBA, Graph Bandit compared with with coverage based greedy approach 41

3.9 Reduction of Active Arm count when state transition mechanism is applied is used with the Graph based bandit 43

3.10 Faster convergence of solution is achieved when state transition mechanism is applied with the graph based bandit 44

3.11 Upper (dotted) line showing max epoch count among all arms where the lower (solid) line expressing that of the corresponding min one for 1000 random sampled users of MovieLens 47

Figure Page

3.12 Upper (dotted) line showing max epoch count among all arms where the lower (solid) line expressing that of the corresponding min one for 1000 random sampled users of Jester 48

3.13 Upper solid line showing ϵ changes over rounds for UCB2. Lower (dotted) line is showing corresponding change in δ 53

3.14 Effect of various tuning function on an initial small α for UCB2 against UCB1 on jester dataset 59

3.15 Effect of various tuning function on an initial small α for UCB2 against UCB1 on movielens dataset 61

3.16 UCB1 based RBA and improved IBA compared against UCB2 with Fixed rate and Logarithmic increase of α, both initialized with 0.1 for movielens dataset after 100K iterations 62

3.17 UCB1 based RBA and improved IBA compared against UCB2 with Fixed rate and Logarithmic increase of α, both initialized with 0.1 for jester dataset after 100K iterations 62

4.1 Zero-sum repeated game between a PR and an attacker, the row player is a defender whereas the column player is an offender. Strategy space consist of two routes 65

4.2 Mixed Strategy Equilibrium for single shot Defender-Attacker Zero-sum Game for one attacker, one defender playing over two routes r1 and r2 73

4.3 Comparison of different learning method of 25 defenders for 100 routes against 25 attackers 81

4.4 Reward statistics of 25 defenders is shown against best response attackers attacking 25 routes at the same time 81

4.5 Total reward of 25 defenders is shown against best response attackers attacking 25 routes at the same time 82

4.6 Total number of successful attacks by 25 attackers over 100 routes against 25 defenders 83

4.7 Total number of successful attacks by 25 attackers compared under different randomized defender deployment scenario 85

4.8 Total number of successful attacks by 25 attackers when various number of transitional defenders are deployed against them 86

4.9 Total number of successful attacks by 25 attackers when various number of routes are considered 87

1. INTRODUCTION

Big-data, with its volume and velocity, challenges modern web-based systems to extract a small and limited amount of critical information effectively and efficiently out of the dynamically changing patterns. As a result, online learning becomes an active area of research with special interest in handling Big Data. In online learning, data become available in a sequential manner and is used for prediction. Unlike batch learning techniques, where the best predictor is generated by learning from the entire training data set at once, online learning is used where it is computationally infeasible to train over the entire dataset. It is especially applicable in the situations where the learning algorithm needs to dynamically adapt to new and diverse data patterns exposed to the system over time.

For example, reinforcement learning based recommendation systems, often need to predict preferences of individuals regarding the special interest group they belong to. The vast diversity of users streaming into the system and their dynamic change of preferences make it tough to find a small number of representative items to cover these groups. In our first work, we developed an online learning algorithm to capture the variety of items these diverse users are interested in, with a limited number of recommended items. In an online environment, with little or no prior knowledge about the distribution of items among the user population, the problem is hard.

In our subsequent work, we leverage this online learning algorithm for the situation, where there is a small team of defenders with the intention to protect a large number of routes from the dynamic attack of the offenders. The idea is to maximize the coverage of available routes based on their sensitivity and vulnerability to attacks. The objective is to increase the team payoff of patrolling defenders and decrease that of the offenders over time. We do this by ensuring the protection of the potentially vulnerable routes by learning the attack patterns. As the number of defenders is limited, their patrolling assignment on the routes needs to be scheduled intelligently. Furthermore, defenders require withstanding various patterns of attacks posed by actively engaging adversarial attackers. In the process of doing that, both attackers and defenders learn from the response they get to their respective actions from the other. Therefore, defending the routes from such opportunistic attackers is an interesting problem which requires continual optimization. We model this as a game between attackers and defenders where the defenders use online reinforcement learning mechanism to predict the next attack pattern. The correlation between different attack patterns observed by the defenders helps their appropriate assignment to routes dynamically.

1.1 Problem Definition

We formulate the problem at hand as a variation of the hitting set problem, where the number of sets and elements within the sets can change dynamically. For the application to Recommendation System, these sets are users' choices exposed to the system. For the Dynamic Patrol Allocation Problem, these are various attacker patterns as attacks happen.

Hitting set is defined as a collection of sets. By definition, a hitting set contains at least one element from every subset in a collection. The Hitting Set Problem (HSP) is to find a set which has non-empty intersection with each of the sets in that collection [49].

Formally, given a finite set of m elements $U = \{e_1, e_2, \ldots e_m\}$, a collection of n finite non-empty subsets, $C = \{S_1, S_2, ..., S_n\}$, s.t. $S_i \subseteq U$ for $i = 1, 2, \ldots n$ and a positive integer k, we say that (U, C) has a hitting Set $S^{'}$ of size at most k, iff

$\exists S^{'} \subseteq U$ s.t. $|S^{'}| \leq k$ AND $\forall S_i \in C$, $\exists e \in S_i$ s.t. $e \in S^{'}$.

When the number of distinct elements is large, an exhaustive search for even a fixed number of elements to form a hitting set is known to be computationally expensive. It becomes more challenging when the sets are streaming online as the system does not have any knowledge about the distribution of elements in these sets.

For our problem, the collection contains all the available items to recommend or the routes to choose from, and every subset of the collection defines a user's preferences or attackers' targets at a particular time. As their choices are diverse, the incoming sets to the learning system are dynamically changing. The set of items to recommend or routes to protect are defined as a hitting set of fixed size k, which can cover maximum user/attack instances. Also, we need to maximize the coverage of different types of attacks or variety of users over time. We define this problem as Online k-hitting Set Problem.

The data stream in our problem is partially observable as we can only examine the feedback from a new incoming set based on the recommendations made from the earlier observations. A negative feedback only notifies the system that the predicted element is not selected by the user (or attacker), but gives no information about what would be a

better serving choice instead. Therefore, we define our Online k-hitting Set Problem under partial observation by the following definition:

Definition 1 *Given a finite set of m elements $U = \{e_1, e_2, ...e_m\}$ and a collection of n finite non empty subsets, $C = \{S_1, S_2, ...\}$ s.t. $S_i \subseteq U$ for $i = 1, 2, \ldots,$ let the set of elements exposed to the system at time t is S_i^t (i.e. $S_i^t \subseteq S_i$).*

For a positive integer $k << m$, The Online k-hitting Set Problem is, to find a set $A^t \subseteq U$ at a time step t before the incoming set for that time is exposed, such that, $|A^t| = k$ and $S_i^t \cap A^t \neq \emptyset$.

If $S_i^t \cap A^t \neq \emptyset$ for a time step t, we set $\delta_t = 1$, otherwise $\delta_t = 0$ (here $\delta_t = 1$ denotes that the set A^t contains at least an element from the set S_i^t arriving at time t).

After T number of time steps, the performance of our algorithm is measured by $\frac{\Sigma_{t=0}^{T} \delta_t}{T}$.

The nature of our problem presents the following unique challenges:

1. No prior knowledge about the distribution of the items makes it an online learning problem under the constraint of a limited number of items to suggest in a dynamic environment where preferences can change over time unintentionally (for Recommendation System) or intentionally (Patrol Allocation).

2. Partial observability of the problem can only allow our algorithm to examine the feedback from an incoming set based on the recommendation or assignment already made. A negative feedback gives no information about what was preferred by the user/attacker instead.

3. The problem requires us to predict a substantially small set of items, solving it is NP-hard even if all the preferences were given offline because the problem is equivalent to the maximum coverage problem which requires the system to try exponentially many combinations of available items.

1.2 Related Problems

A well-studied variant of the hitting set problem is the Minimum Hitting Set (MHS) problem [49] where the target is to find a hitting set with minimum cardinality. The size of the hitting set is given by the number of elements in the hitting set H, i.e., $|H|$. The difference between MHS and HSP is that MHS has the constraint that if H is an MHS, then no proper subset of H can be a hitting set.

More formally, given a finite set of m elements $U = \{e_1, e_2, \ldots e_m\}$ and a collection of n finite non empty subsets, $C = \{S_1, S_2, ..., S_n\}$ s.t. $S_i \subseteq U$ for $i = 1, 2, \ldots n$, the Minimum Hitting Set (MHS) of the collection C is a set $H \subseteq S$ s.t:

$\forall S_i \in C, S_i \cap H \neq \emptyset$ (every subset in C must contain at least one element of H) AND $\neg\exists H^{'} \subset H : S_i \cap H^{'} \neq \emptyset$ for each $i \in 1, 2, \ldots n$ (no proper subset of H is a hitting set)

The difference between the above formulations and our problem is that we want $|S^{'}| = k$ for the above definition where $S_i \in C$ for $i = 1, 2, \ldots$ might be *partially observable* i.e. not all $e \in S_i$ are exposed to our algorithm. Also, we do not know the number of unique sets in C, i.e., n is unknown as sets stream into our system one at a time. We intend to construct an online approximation for a hitting set for the partially observed sets in the collection as observed over a certain amount of time.

The closest problem related to our problem is Implicit Hitting Set Problem (IHS) [16]. In IHS, the following information are given: a finite set of m elements $U = \{e_1, e_2, \dots e_m\}$, a collection of n finite non empty subsets $C = \{S_1, S_2, ..., \}$ s.t. $S_i \subseteq U$ for $i = 1, 2, \dots$ (where C is too large to specify explicitly), a Hitting set H and a polynomial time oracle O that determines whether H is a hitting set or returns a set from C which has not been hit by H. For IHS problem, algorithm needs to find a small hitting set $H^{'}$ by making at most polynomial$(|U|)$ queries to the oracle O.

IHS is different from general hitting set problem but is a generalization of online algorithms for the hitting-set problem–one set (to hit) at a time arrives online from the collection C. Moreover, on obtaining that set, the algorithm needs to decide which new element to include in the hitting set and keep that element.

However, in IHS, hitting set construction is based on an update procedure that can only add element similar to open set problem [35], whereas our algorithm needs to use replacement instead, as the size of our set is limited. i.e k is fixed. Also in IHS, only those sets of C which have not been hit, are revealed online. On the other hand, for our problem, we get feedback from a random sample user (for the case of recommendation system) or the best response from an adversary (for patrol allocation).

We infer the following properties of our solution from MHS:

- If $k =$ size of MHS, then MHS is the exact solution to our problem

  Let our solution be a different set than MHS. If we want to ensure that every subset in the collection contains at least one element from our solution with the constraint

that the size of our set cannot exceed the minimum cardinality k, then our solution becomes exactly same as MHS.

- If $k <$ size of MHS, solution to our problem will not be a hitting set

  If we want to construct a set of size k which is smaller than the size of MHS, there will be one or more sets in the collection which must have an empty intersection with the set we construct. This violates the property of hitting set, so the set we construct will not be a hitting set.

- If $k >$ size of MHS, then MHS $\subseteq$ our solution

  If our solution results in a hitting set of size k which is larger than the size of MHS, then there will be a proper subset of our solution which will be an MHS.

1.3 Overview of this thesis

This dissertation is organized in the following manner:

in Chapter 2, we introduce the baseline stochastic bandit UCB algorithm to solve the coverage problem and discuss how it facilitates the online learning.

In Chapter 3, we describe our contributions to the application area of recommendation systems. First, we introduce a reward bias to select covering items for a large number of sets by a limited number of bandits, We then explain our novel graph-based bandit algorithm and its performance on recommendation systems regarding covering diverse choices of different users.

We also introduce a mechanism to execute several different bandit instances in a parallel and synchronized manner to construct the recommendation set. We show the selection of global parameter for the bandits affects the error bound for such parallel execution.

In Chapter 5, we discuss our work which focuses on leveraging our proposed graph-based bandit algorithm for deploying police patrols to sensitive routes in a neighborhood to withstand the potential attacks. In this Dynamic Patrol Allocation Problem, we formulate a game between offender and defender where a limited number of available defenders as a team of patrolling agents run UCB1 bandit instances to determine which routes to protect from a large number of available routes.

Unlike Recommendation System, in Patrol Allocation problem, feedbacks are coming from adversarial opponents which we model by using Stackelberg Security Game. This imposes additional challenges to our graph-based bandits, and we found that introducing adaptive transition among graph based bandit defenders can facilitate them learning different attack patterns within a limited amount of trials, which is important for such a time-sensitive application where severe loss can incur otherwise.

In Chapter 6, we summarize our contributions and discuss some future directions of our work.

2. BACKGROUND

It is crucial for our problem to find a representative element which is common to a large number of sets in a collection of sets. That will suffice an element to be a candidate for Online k-Hitting Set. As we can pick at the most k number of elements at a particular time, our problem depends on sequential decision making over time with an intention to cover maximum possible incoming sets. Most literature used Multi-Armed Bandits [29] to solve such problem.

2.1 Stochastic Multi Armed Bandit

The baseline algorithm we used for solving the Online k-hitting Set problem is Stochastic Multi-Armed Bandit Algorithms [29]. In Multi-Armed Bandit (MAB) problem, a gambler faces a slot machine with multiple levers (arm). The gambler decides how many times to play each arm in what the order with an objective that the decision will maximize the sum of rewards earned through a sequence of arms played. Playing each arm provides a random reward from a distribution specific to that arm, which is unknown to the gambler.

For example, for an online learning system, at each time t, from n available items/routes (arms), k items needs to be picked by the algorithm. So k instances of multi-armed bandits are instantiated, each having n arms to select from. Once an item (or route) has been selected by the j^{th} bandit, that item will be unavailable for the rest of the

bandits, i.e., $j+1...k^{th}$ bandits. Thus a set of k non-identical elements is constructed. The next user's/attacker's choice is compared with this recommended set and accordingly reward is fed back to the associated bandits.

2.2 Upper Confidence Bound (UCB)

The most popular stochastic bandit algorithm, UCB1 [5] has been used as one of our baselines where each slot of k-element set runs UCB1 to pick an item i from n available options which maximizes $x_i + \sqrt{\frac{2\log(f)}{f_i}}$ where x_i denotes the current average reward of the item i and f_i denotes the number of times item i has been picked so far in total t rounds. Here f denotes the total count of all items picked so far as $f = \Sigma_{i=1}^{t}(f_i)$. It is not difficult to observe that confidence bound grows with the total number of options we have chosen but shrinks with the number of times we have tried a particular option. This ensures each action is tried infinitely often but still balances exploration and exploitation.

After each pulling of arm reward and the corresponding arm, the count gets updated for the next round. Hence UCB1 has specific:

- **update policy** adjusting the the average reward x_i after the current trial
- **selection policy** based on the average reward x_i added with the term $\sqrt{\frac{2\log(f)}{f_i}}$.

This is called Upper Confidence Bound (UCB1) [5] because this value can be interpreted as the upper bound of a confidence interval so that the true average reward of each item i is below this upper confidence bound with high probability. If we have tried an item less often, our estimated reward is less accurate, so the confidence interval is larger. It shrinks as we recommend that item more often.

However, unlike UCB1, in UCB2, choice of picking an option from all available arms are given once in an epoch instead of every time step. Accordingly, reward value for those selected options gets delayed to be adjusted till an epoch is finished. In other words, once the arms get selected by the bandits, those bandits need to keep choosing to those arms for the entire epoch (which may consist of one or more time steps and denoted as *epoch length*). Epoch length for each arm i is denoted as R_i and is controlled by:

- The epoch count, which keeps the record of how many epochs (with variable epoch length) that arm has been played. Longer epoch length or higher epoch count means the bandit was pulling to choosing that arm repeatedly (exploiting).

- an external parameter α which governs the exploration of the available arms and controls the length of epoch for the chosen arm.

The basic UCB2 algorithm [5] is illustrated below:

Parameter: $0 < \alpha < 1$

Initialization: Set the epoch counter vector $r_j = 0$ for all $j = 1, 2, ...n$ where n is the number of available arms

Loop:

- Define $\tau(r) = \lceil (1+\alpha)^r \rceil$ where $r \in \{r_1, r_2, r_3 ... r_n\}$

- $argmax_i \; x_i + a_{f,r_i}$ where x_i is the average reward obtained so far by choosing option i and $a_{f,r_i} = \sqrt{\frac{(1+\alpha) ln(ef/\tau(r_i))}{2\tau(r_i)}}$ where $f = \Sigma_{i=1}^{t}(f_i)$ as f_i denotes the number of times option i has been played so far in total t rounds.

- Play option i exactly $R_i = \tau(r_i + 1) - \tau(r_i)$ times (here R_i is the epoch length for the arm i)

- Set $r_i \leftarrow r_i + 1$

The algorithm keeps a record of the number of times each arm is repeatedly chosen. This counter (epoch count r_i for arm i) is incremented by 1 for all consecutive pulls of the same arm i for the entire epoch length (by the end of the **loop**). However, once an arm i is chosen, UCB2 allows it to get pulled over and over again for a certain period (epoch length R_i) to learn the confidence bound of that arm. Similar to UCB1, the more an arm is chosen, the tighter this bound gets for that arm. Once the assigned period (epoch length R_i) is over, the reward for the arm i gets updated and this adjustment lead the bandit to decide which arm to select next.

The algorithm is designed in a way that only one bandit instance has the full control on choosing an arm i, setting epoch length (R_i) for that arm and update the epoch count for it (r_i). Multiple instances of UCB2 bandits having the same set of arms is bound to have a conflict on selecting an arm exclusively and deciding on a common epoch length of arms chosen from all the instances. Also in the original UCB2 algorithm, α remains fixed between (0,1). As a consequence, epoch length (R_i) for an arm i grows exponentially over time [step 3 in the **loop**], which makes it difficult to its use for practical purposes. In our work, we facilitate the parallel execution of UCB2 bandits by maintaining a synchronized epoch. We also show the PAC [38] bound for the critical parameters which directly impacts the epoch. In our work, we facilitate the parallel execution of UCB2 bandits with

synchronized epoch. We also show how the PAC bound is affected by choice of the critical parameters for the algorithm.

2.3 Related Works

We study two specific application area involving online learning for coverage (1) Recommendation Systems (2) Patrol Assignment for Security. In this section, we discuss the related works done using UCB algorithm in both these areas.

2.3.1 Application in Recommendation Systems

In the existing work, two major approaches have been found to build a recommendation system by using UCB1 bandits. We discuss their advantages and disadvantages in this section. The first approach, Ranked Bandit Algorithm (RBA) [40] used each item in the recommendation set to satisfy a different type of user and hence came up with a consensual set based on diverse users. It used strong similarity measures (dependency) between items and takes into account only the first item selected by a user from the recommendation set to represent the group of users that share at least one common item of interest. It specifically holds i^{th} bandit responsible for i^{th} item in the recommendation set. But performance of i^{th} bandit is actually dependent on picking the appropriate item (in proper order) by on all other bandits preceding i. As a consequence of this cascading effect, the learning for i^{th} item cannot really start before $\Omega(n^{i-1})$ time steps [28]. According to their setting, the probability of user $x \in X$ selecting the i^{th} item from the recommendation set is denoted as p_i which is conditional on the fact that the user did

not select any of the items in that set presented in any earlier positions. Formally, $p_i = Pr(x^i = 1|x^{i-1} = 0)$ for all $i \in k$ where the binary value $\{0, 1\}$ of x^i denotes the probability of selecting the item i by the user x

On its attempt to maximize the marginal gain of i^{th} bandit, where each bandit is a random binary variable, RBA forms a Markov chain where the later bandits have to wait for an earlier one to converge. To speed up the process, Independent Bandit Algorithm (IBA) [28] assumed independence between items and used Probability Ranking Principle (PRP) [44] as a greedy method to select items to recommend. PRP give equal credit to all the item a user selects within the recommendation set, and each bandit responsible for selecting an item of that user's choice gets a reward of 1. The overall payoff for the recommendation set is 1 even more than one item are selected by that user. However, this solution is sub-optimal in minimizing abandonment because diverse users are likely to be a part of the minority, which might not be covered by the top-k items PRP selects. So it often fails to capture diversity.

We study a recommendation system problem, in which the system must be able to cover as many users' preferences as possible while these preferences change over time. This problem can be formulated as a variation of the maximum coverage problem, specifically, we defined a novel problem of Online k-Hitting Set, where the number of sets and elements within the sets can change dynamically. When the number of distinctive elements is large, an exhaustive search for even a fixed number of elements is known to be computationally expensive. The problem is known to be NP-hard, and many known algorithms tend to have exponential growth in complexity. We propose a novel graph-based UCB1 algorithm that effectively minimizes the number of elements to

consider, thereby reducing the search space greatly. The algorithm utilizes a new rewarding scheme to choose items that satisfy more users by balancing coverage and diversity as it constructs a relational graph between items to choose. Experiments show that the new algorithm performs better than existing techniques such as Ranked Bandit[40] and Independent Bandits [28] regarding satisfying diverse types of users while minimizing computational complexity.

2.3.2 Application in Dynamic Patrol Allocation

In our study for online route assignment to patrolling robots for crime-prone areas of a neighborhood. We considered that we could only deploy a fixed number of autonomous robots; therefore, it is necessary to maximize the impact of their routing assignments. We develop an online model to approximate the optimal solution under some attack probability distributions. Our algorithm can direct a group of patrolling robots to a few routes out of many available at a certain time against the best possible adversarial response. Our simulation results show that using UCB1 bandits to determine the strategy for these Patrolling Robots (PR), we can achieve considerable coverage of sensitive routes by playing Stackelberg Game against different types of attacks.

Most works including [7] considered the similar problem as a finite-horizon MDP [10] where the learning process ends after T steps and the objective is to maximize the accumulated total reward. They applied a variant of Monte Carlo Tree Search (MCTS) [27], where UCB1 is used as a tree policy leveraging UCB Tree (UCT) algorithm [26]. They extensively used repeated SSG where an attacker is drawn from a distribution. Their

work is based on [31] in which a defender's mixed strategy is discretized to ensure finite state and action space to formalize a Bayesian Stackelberg Game; otherwise, the branching factor for the tree will be too large to come up with a practical solution.

[52] also investigated repeated security games with unknown game pay off and attacker behavior. In their work, they used "Follow the Perturbed Leader with Uniform Exploration" (FPL-UE) technique assuming no prior knowledge about the attack dynamics. They compared their work with Quantal Response [53] model which requires some prior knowledge about the attackers. They showed that their method works well against typical attacker profiles.

Practical implications of such game using online reinforcement learning methods have been depicted in the work of [39], [54], and [48]. In [55], they used an iterative learning and planning mechanism that keeps updating the adversary model periodically. They used EM [20] algorithm with dynamic planning against adaptive criminals. In that work, instead of using SSG, they used Dynamic Bayesian Network [17] for modeling the problem.

[36] introduced Rolling Down Randomization (RDR), a technique to generate randomized policies where a similar problem has been modeled using POMDP [25]. Above a given expected reward threshold, their defenders need to generate a randomized policy. To make it hard for the opponent to anticipate the defender's action, their method increased the policy entropy. Their work leveraged both linear and nonlinear optimization methods for a single defender.

3. ONLINE RECOMMENDATION SYSTEM

Recommendation systems in Big-data era need to address not only the massive amount of users to satisfy but also the rapidly changing patterns of preferences. With such a rapid and continuous shift of users' preferences, recommendation system needs to learn quickly from the patterns of choices in previous users to suggest items to the new user. As diverse tastes of the incoming users induce a fast turn over time for items, online recommendation systems get little prior knowledge about the distribution of the preference on items among the user population. Moreover, most recommendation systems need to pick a limited number of items to suggest to a user, still, require that at least one of these suggested items can satisfy her taste. This online learning algorithm tries to produce a substantially small but a diverse recommendation set from a large number of items, at the same time satisfying different user types.

Most existing literature including [28] consider a recommendation system where n items $\{i_1, i_2, i_3, ...i_n\}$ are given. When a user arrives, the system needs to show her a set of k items where $k << n$. If she finds any one of them relevant, the system gets a payoff of 1. If none of them is relevant to her interest, then it gets a payoff of 0. [28] defined a user relevance vector as follows:

Definition 2 *Each user j can be represented by a $\{0,1\}^n$ vector X^j where $X^j_i = 1$ indicates that user j found item i relevant where $i \in \{i_1, i_2, i_3, ...i_n\}$. For example,*

$X^j = \{0, 1, 0, 1, 0, 0, 0, 0, 0, 1\}$ *means user* j *finds* 2^{nd}, 4^{th} *and* 10^{th} *items relevant out of* $n = 10$ *items that are given to the recommendation system to recommend from.*

Distribution of these vectors X^j is unknown to the system, but these vectors are assumed to express the "types" of users. We assume that there is a large degree of correlation between these vectors. It indicates that when the system successfully recommends a set of items to a user, that recommendation set may cover other users of the same type.

At time t, after a recommendation set for a random user is already generated, that specific user vector is disclosed in the system and system gets its payoff. Then the system updates the recommendation set for the next possible user so that it maximizes its payoff and thus minimizes the abandonment rate. This is the necessary condition for the online learning.

As the recommendation system presents a set of k items S^t without observing X^t, we used the definition of set relevance function F used by [28] for the fairness of comparison between that work with ours:

Definition 3 $F(X^t, S^t)$ *is a submodular set function [46] stands for the payoff of showing* S^t *to user with vector* X^t*. Characterization of user click event is done by the following conditions: if* $X_i^t = 1$ *for some* $i \in S^t$ *then* $F(X^t, S^t) = 1$ *else* $F(X^t, S^t) = 0$

According to [28], the value of displaying a set S^t, is the expected value $E[F(S^t, X)]$ where the expectation is taken over the realization of the relevance vector X from the distribution D. Once realized, for a fixed set S, it is denoted as $E[F(S)]$ (the fraction of

users satisfied by at least one item in S). Like [28], our target is to minimize abandonment, so we need to maximize click-through rate [24], which is:

$$\begin{aligned} \text{maximize} \quad & E[F(S)] \\ \text{subject to} \quad & |S| = k \end{aligned}$$

This is essentially same as the performance measurement metrics for the Online k-hitting Set Problem that we formulated at the Introduction chapter.

3.1 Reward Bias

We present our first proposed method in Algorithm 1. We initialize k number of bandits $UCB1_1(n), ...UCB1_k(n)$ to construct a set of k items where bandit i gets priority on selecting an item over bandit $i + 1$. Similar to [28], once an item gets selected by a preceding bandit, it becomes unavailable to any later bandits (ref line 6 and 7 of the algorithm). After the recommendation set S^t is created this way, it is compared with a random user vector X^t picked in time t (in line 9-11). The novelty of our approach is in the rewarding scheme for the bandits (shown in line 12,13). If the recommendation set contains more than one items preferred by the user, the first bandit responsible for picking the preferred item get a much higher reward as F_{it} for that item than any other bandits who picked other items of that user's preference. We set that higher reward C to be equal to the accumulated reward of all bandits who picked a preferred item for that user in that recommendation set. In this way, we uplift the average reward for the bandit who picked the first item the user preferred. This creates a bias towards the first item a user prefer and

helps recommending users of same user type with one preferred item which is highly likely to satisfy all of them. For example, if there is a total of 100 users in the system, each

Algorithm 1 Proposed Method

Input: n items
Output: k items
Initialize $UCB1_1(n), ...UCB1_k(n)$
for all $t \in T$ **do**
  $S_0^t \leftarrow \emptyset$
  for all $i = 1$ **to** k **do**
    $select(UCB1_i, N \setminus S_{i-1}^t)$
  end for
  Pick a random user vector from the dataset
  Display S^t **to user for and receive feedback vector** X^t
  $C =$**total number of items clicked by the user from** S^t
  Feedback:
  $$F_{it} = \begin{cases} C, & \textbf{if } X_i^t = 1 \textbf{ for the } i \in S^t \textbf{ which is the first click} \\ 1, & \textbf{if } X_i^t = 1 \textbf{ for any } i \in S^t \textbf{ which is not the first click} \\ 0, & \textbf{otherwise}. \end{cases}$$
  $update(UCB1_i, F_{it})$
end for

of them is represented by a user vector of size 10 (expressing their items of choice out of 10 available items). Lets assume we can only recommend 2 items out of 10. Say, some of these users prefer *item1, item3, item5, item7 and item9* together **(user type-1)** whereas some prefer *item2, item4, item6, item8 and item10* together **(user type-2)** and remaining prefer *item4 and item6* together**(user type-3)**. UCB1 ensures that an item is recommended enough number of times to be reasonably confident about their chance of getting rewarded. Now if we reward a bandit with 5 (accumulated from *items 1,3,5,7 and 9*) for selecting *item1*, the probability of selecting *item1* will be higher for our recommendation set which can eventually cover any user of user type-1. Next item in our recommendation set would be *item2* for adopting the same rewarding strategy with a hope to cover the user

type-2. But unfortunately item2 cannot cover user type-3 who prefer *item4 and item6*, not *item2*. If we could pick *item4* instead of *item2* we could cover both user types which is the optimal recommendation. But as recommending item2 fails to satisfy the users only preferring *item4* and *item6* together, according to UCB1 policy, average reward for a bandit picking *item2* will be decreased if more of the incoming users are of user type-3. Eventually average reward of a bandit selecting item4 will beat that of selecting *item2*. Although the basic idea behind this design follows PRP like IBA, but unlike IBA, unequal rewarding for bandits on selecting an item makes the highly rewarded bandit choose a representative item covering all other items that are correlated to it. Ultimate goal of this mechanism is to reduces the chance of selecting more than one item in our recommendation set preferred by the same user type.

3.1.1 Initial Results

In this section we show the performance boost occurred over the existing methods by adopting our proposed technique. We used real data sets our experiments. Publicly available dataset of MovieLens Dataset [21] has been used for this purpose. MovieLens Dataset consist of more than 900 users rating almost 1700 movies which we picked based on most rated (rating ranges as discrete numbers between 1 to 5) and for that θ value has been conservatively set to 2 to discretize the choices as binary values.

According to the results shown in figure 3.1 and 3.2 our method clearly outperforms RBA in terms of efficiency and accuracy for both the data sets. It is competitive with IBA for Jester dataset [Ref. 3.1] when we tried recommending 5 out of available 10 high rated

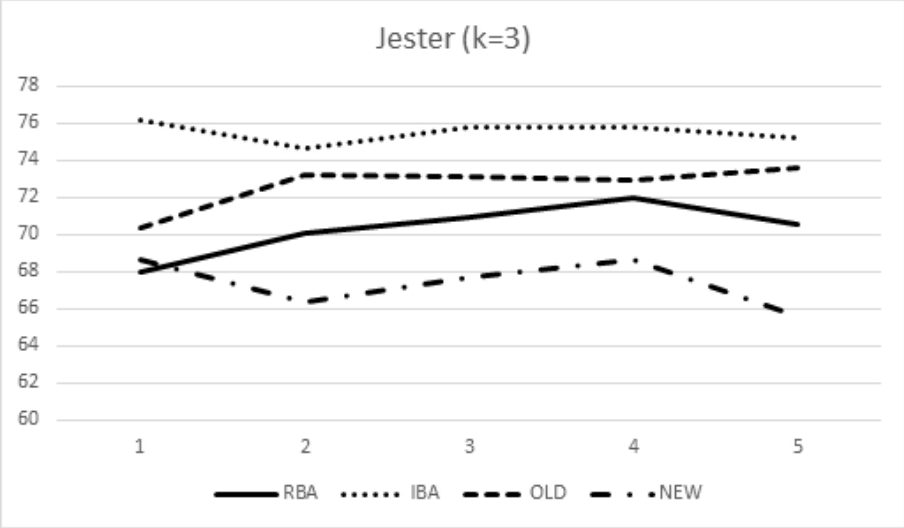


Fig. 3.1.: Performance comparison of recommending 3 jokes out of 10 from Jester dataset, Y-axis showing x1000 times iteration, new and old denoting the performance of with and without Graph based approach

jokes. But our method achieves much better result than IBA when we shrink the recommendation set size to 3 [Ref. 3.1]. Naturally the overall performance got scaled down as we decrease k. The Y-axis of both the plots (drawn in same scale) projects the $\%$ of users satisfied by the recommendation set at the time stamp denoted by X-axis. Each point in both the graph shows an average of 5 runs and went upto 100,000 time steps.

For the other experiment shown in 3.3 we ran with Movielens data set, we decided to keep the recommendation set size (k) fixed as 10, but we had to deal with a much higher volume of available choices S (almost 1700 movies to pick from) for a user. This makes our method run to solve for more than a thousand armed bandit problem - which, to the best of our knowledge has never been tried before. It also shows similar trend of outperforming RBA and IBA on an average, as we simulated till 550,000 time steps. The point in the graph shows the results reached at 10K, 50K, 100K, 200K, 500K, 510K and 550K steps.

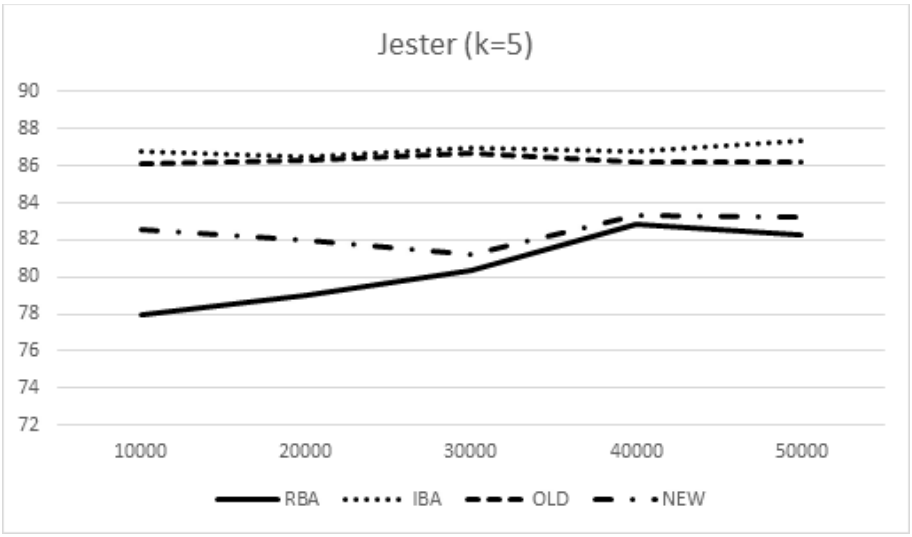


Fig. 3.2.: Performance comparison of recommending 5 jokes out of 10 from Jester dataset, Y-axis showing x1000 times iteration, new and old denoting the performance of with and without Graph based approach

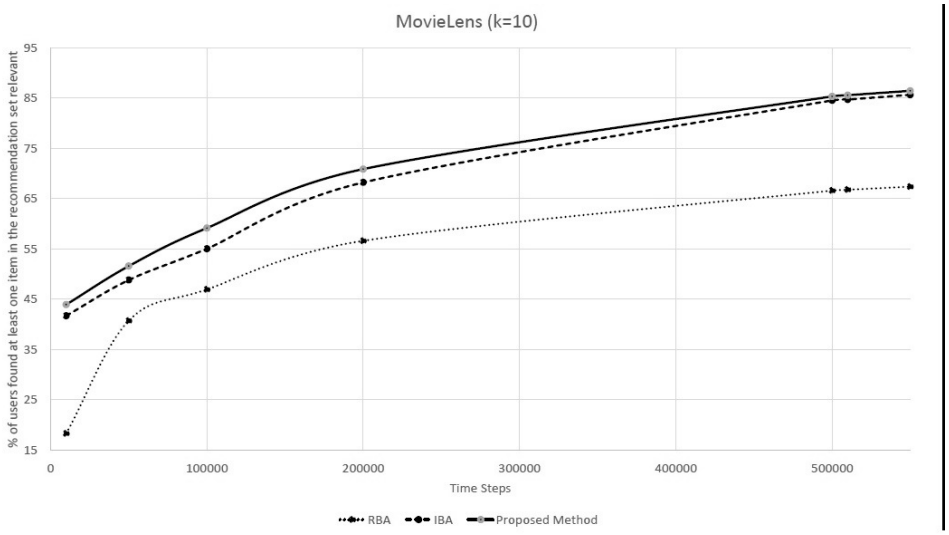


Fig. 3.3.: Performance comparison of RBA, IBA and Reward bias of recommending 10 movies out of 1700 from MovieLens dataset of around 1000 users.

3.2 Graph Based UCB1

We develop a graph based UCB1 bandit model to handle the challenges of online learning by incorporating diversity without any non trivial sacrifice of coverage and applied our model in the area of recommendation systems to compare our technique with existing ones.

Reinforcement learning based recommendation systems often need to predict preferences of individuals in terms of the special interest group they belong to. The vast diversity of users' preference makes it difficult to find a small number of representative items to denote these groups. Our intention is to capture the common interests of these diverse users with limited number of recommended items so that our recommendation set can cover maximum users. We already realize that in an online learning environment, most Recommendation Systems require to learn quickly from choices of the previous users to suggest items to a new user with little or no prior knowledge about the distribution of items among the user population. Hence learning the users' choice online and building a small set to accommodate their diverse choices is a hard problem.

We define this need for "fast coverage of diverse user choices with a limited number of items to suggest" as an Online k-hitting Set Problem and propose a graph-based multi armed bandit algorithm to learn the implicit correlations among the choice of items by different types of users. Our method effectively reduces the number of items to be considered for satisfying the variety of users as they have some common items of preference. As a consequence, our approach reduces computational complexity of the problem and produce a fast solution with a considerably diverse coverage.

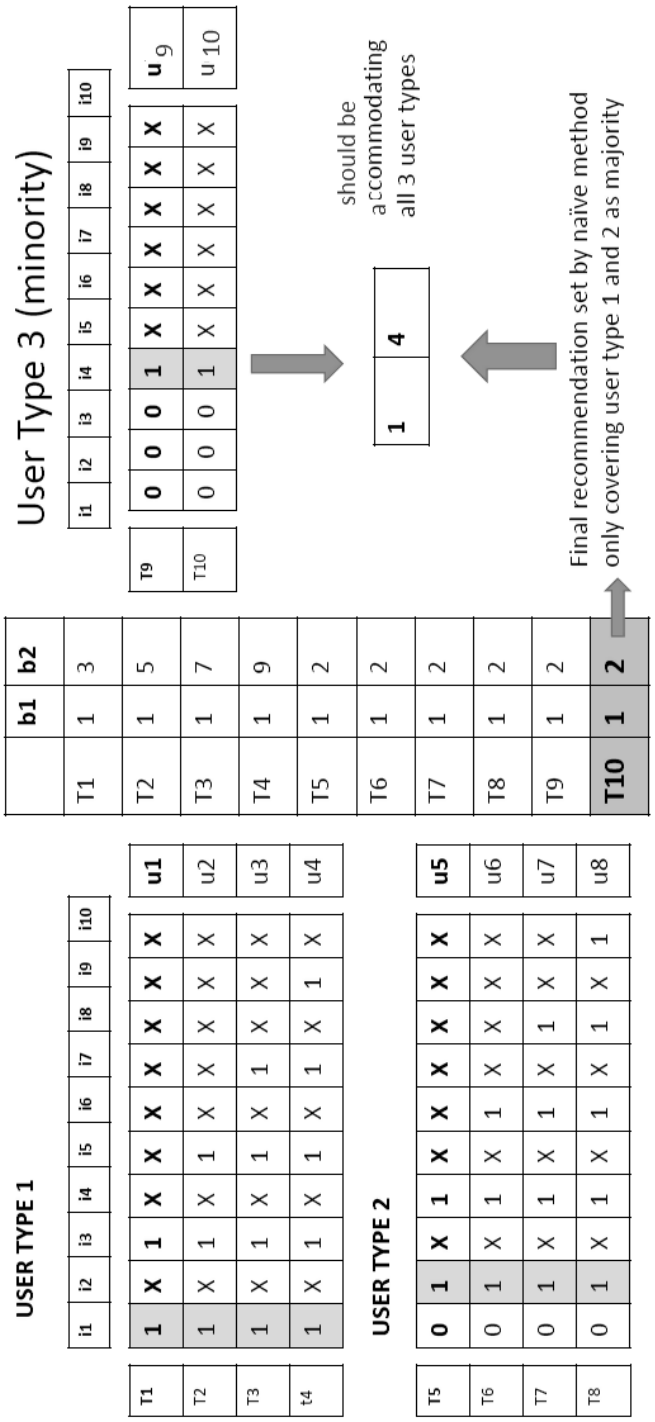


Fig. 3.4.: Limitation of Non Graph Based method where Ti labels the 2 item recommendation set at i^{th} time step coming from 2 bandits b1 and b2 and u_i denotes the user vector it is recommended to

3.2.1 Motivation for Graph Representation

In our naive approach [42], we introduced reward bias to the first item to shrink the search space for possible candidate items to generate a recommendation set. According to that scheme, if the recommendation set contains more than one items preferred by the user, the first bandit responsible for picking the preferred item get a relatively much higher reward than any other bandits who picked other items of that user's preference. This creates a bias towards the first item a user prefers and helps recommending users of same user type with one preferred item.

This naive approach with reward bias introduce unequal rewarding scheme in attempt to find a candidate element with a potential membership to a large number of sets, but it still can exclude many user types with smaller populations (refer them as 'minority user-types' who are more diverse in their choice of items) Unlike [28], unequal rewarding for bandits on selecting an item makes the highly rewarded bandit choose a representative item covering all other items that are correlated to it. Focus of this approach is to reduces the chance of selecting more than one item preferred by the same user type, thereby wasting the precious limit of total number of items to be recommended, k. From that approach, we want to make it more effective in accommodating a diverse user-types by representing those minority users who have at least one of their preferred item overlapped with the majority users; but due to the PRP [44] principle, bandits have less chance to select an overlapping item from a minority user-type which could cover other highly observed users. This problem is illustrated in figure 3.4

Let there be a total of 100 users in the system, each of them is represented by a user vector of size 10 (expressing their items of choice out of 10 available items). Let's also assume that we can only recommend 2 items out of 10. I.e., $k = 2$ and $n = 10$. Say, there are 3 types of users:

- user-type1: prefer *item1, item3, item5, item7 and item9* together
- user-type2: prefer *item2, item4, item6, item8 and item10* together
- user-type3: prefer *item4, item6, item8 and item10* together

UCB1 ensures that an item is recommended enough number of times to be reasonably confident about their chance of getting rewarded. Now according to our scheme:

- For selecting item1, bandit1 will be rewarded with at most a payoff of 2 (accumulated from *items 1,3 or 1,5 or 1,7 or 1,9*) provided that 2nd bandit picked either items 3 or 5 or 7 or 9 and get a reward of 1.
- According to the same mechanism, bandit2 will be rewarded more than others for picking item2.
- This may result in a recommendation set with item1 and item2. This cause dominance of user-types 1 and 2, depriving user-type 3, if the majority of the population are of user-types 1 and 2.
- On the other hand, a closer look into these 3 user-types can reveal that, if we could reward the 2nd bandit for picking item4 instead of item2, it could cover both the user-type2 and user-type3.

According to UCB1 policy, the average reward for a bandit picking *item2* will be decreased if more of the incoming users are of user type-3 and eventually average reward of a bandit selecting item4 will beat that of selecting *item2*. But that will not happen until user-type3 data outnumbers user-type2, which may take a longer time to learn. Recall this is an online learning problem. The limitation of choosing only the first item to be the representative for a user-type reduces the effectiveness of the algorithm, in particular in online situation. We need to be able to change the representative items dynamically as needed.

To solve this problem, discount on the reward of bandits was essential where the item selected by it is not sufficient to address the satisfaction of other user-types. This discount factor is not trivial to compute because this is based on the user relevance vector disclosed to the system since the system started learning. Therefore, we introduce the Relevance relationship between items, which remembers history of relationships among items in a computationally efficient way in terms of a graph structure:

Definition 4 *Relevance between items in the recommendation set is denoted by* $Rel(i, j)$. $Rel(i, j) = 1$ *if items* i *and item* j *are found in the same user vector. Otherwise* $Rel(i, j) = 0$

This Relevance is denoted as **edges** between items in our graph based method and it impacts our rewarding scheme for bandits responsible for picking the corresponding items in the following way:

Definition 5 *Reward for a bandit to select an item* i *where* i *gets the first click from a user:*

$Rwd(i) = 1 + \frac{1}{1+|N|}\Sigma_{j\in\{N\setminus i\}}Rel(i,j)$ *where* N *is the set of all items in the recommendation set*

This term has been used as **node weights** in our proposed graph based algorithm where each node represents an item. This scheme is used to reduce the importance of an item if it appears together with other items from the same user-type historically and hence implicitly facilitate the selection of an item preferred by minority user-types. The reward function of a bandit who picks an item selected by the a user is a logarithmic function of the weight of the node representing that item in the graph.

3.2.2 Construction of Relevance Item Graph

Algorithm 2 Initial_Graph

1: Create graph $G^0(V, E)$ consists of isolated nodes where $v_n \in V$ for each item $n \in N$ and initialize $E = \{\}$
2: **for all** $n = 1$ **to** N **do**
3: Let $w_n \in W = 0$ for $v_n \in V$ where W is the weight vector for nodes
4: **end for**
5: Call $BRecommend\_kItems(G^0, B)$ where B is the set of k number of bandits

To keep track of relations among the items seen and recommended so far in the online environment, we need to build and update a relevance item graph each time the system sees a user and a recommendation is made. We construct the relevance item graph in the following way:

- At the beginning, there are n number of isolated nodes representing the total number of items to choose from.

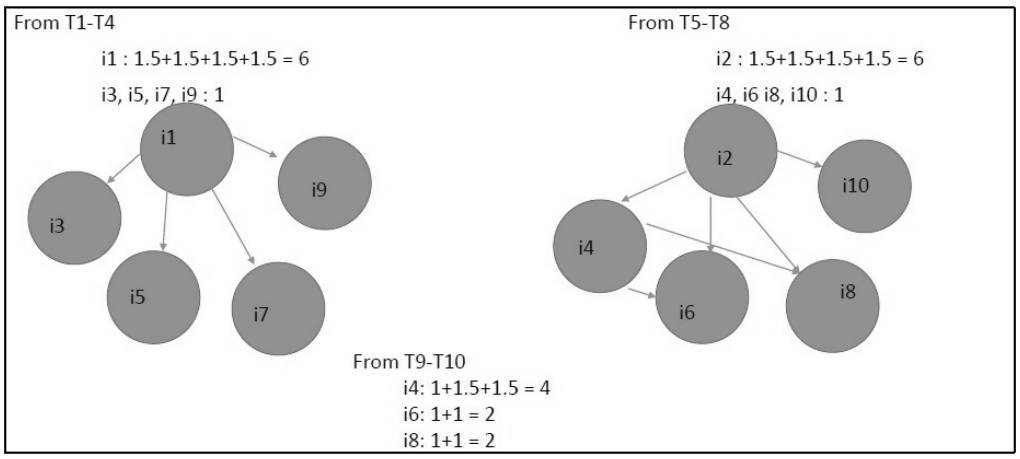


Fig. 3.5.: Emergence of item 4 with increasing weight as a potential replacement of item 2 in the graph based method to accommodate user type 3 as in the illustrative example mentioned in Figure 2

- Each node has a weight associated with it which denotes the relevance of the item in the graph. Initially all node has identical weight of 0
- Each time a random user is shown a set of k items by the recommendation system. This is a simulation of an online environment.
- If the user selects (i.e., likes) more than one of these recommended items then we draw a directed edge from the node, which stands for the first choice of items to the other items (nodes) chosen by the same user. For example, if a user selects *item1, item3,* and *item5*. There will be directed edges from *item1* to *item3* and from *item1* to *item5.*

Fig. 3.5 shows how this weighting scheme alleviate the issue related to our past approach and better handles the overlapping items chosen by different user types.

Algorithm 2 shows the initialization of the graph and Algorithm and 5 shows how the graph is evolving dynamically.

3.2.3 Discounted Reward Based on Item Correlation

As we construct the graph dynamically as each user data is encountered in an online fashion, weights of the nodes get adjusted in the following manner: if the user chooses one or more items from the recommendation set: the node representing the first choice of the user gets an increment of weight by $1 + \frac{1}{C}$ where C is the number of total items in the recommendation set picked by that user according to Definition 4. This technique assigns

Algorithm 3 BRecommend_kItems

Input: The graph at the end of $(t-1)^{th}$ **round:** G^{t-1}**,**
the set of k **bandits each having** N **arms:** $B = UCB1_1(N), ...UCB1_k(N)$
Output: Recommendation set of k **items for user arrived at time** t**:** R_k^t
$R_0^t \leftarrow \{\}$
for all $i = 1$ **to** k **do**
    $select(UCB1_i(N), N \setminus R_{i-1}^t)$
end for
call $Update\_Graph(G^{t-1}, R^t, B)$

Algorithm 4 Adjust_Reward

Input: B **set of all UCB1 bandits,**
k **items recommended,**
R^t **the recommendation set,**
X^t **the user vector**
Output: Reward updated for all Bandits
for all $j = 1$ **to** k **do**
    Feedback the j^{th} **Bandits with the following reward:**
    $F_{jt} = \begin{cases} \log(w_j), & \textbf{if } X_j^t = 1 \textbf{ for the } j \in R^t \\ 0, & \textbf{otherwise}. \end{cases}$
    $update(UCB1_j, F_{jt})$
end for

less weight to each node as more items are selected by the user at a time. Idea behind this is, if more items in our recommendation system is chosen together with an item a user first picks, then that item is not a good representation of the specific user-type as opposed to an

item which is uniquely selected by the user. Other items selected by the same user instance (but are not her first pick), will have an increment by 1 in their respective weights.

This weighting scheme also ensures that, if only one item from the recommendation set is selected by that user, it gets the maximum weight of $1 + \frac{1}{1} = 2$ as that single item is potentially single-handily covering that user-type. If the user selects none of the items recommended, then each node in the recommendation set will have an increment of 0 in its weight. Fig. 3.5 shows how this weighting scheme alleviate the issue related to our past approach and better handles the overlapping items chosen by different user types. Algorithm 4 is developed on this idea.

Algorithm 5 Update_Graph

Input: Graph at the end of $(t-1)^{th}$ **round:** G^{t-1}**,**
Recommendation set: R^t
Set of all bandits: B
Output: Updated graph after t **round:** $G^t(V, E)$
Pick a random user vector u^t
Generate feedback vector X^t **from the indices of** u^t **which contains** 1
Create a set of directed edge E^t **such that for each** $e(i,j) \in E^t$**:**
v_i **represents the item clicked which has the lowest index in** R^t **and** v_j **represents any other items clicked in** R^t **(according to the feedback vector** X^t
$E = E \cup E^t$
$C =$**total number of items in** R^t **which matches with those of** X^t
if $(t\%\tau == 0)$ **then** $C = C \times f(t)$ **where** $f(t)$ **is the discount factor on** C **at time step** t
Update W **by following:**
for all $n \in N$ **do**

$$w_n = \begin{cases} w_n + [1 + \frac{1}{C}], & \textbf{if } v_n \textbf{ is the only item clicked or has one or more outgoing edge(s) in } E^t \\ w_n + 1, & \textbf{if } v_n \textbf{ has an incoming edge in } E^t \\ w_n, & \textbf{otherwise}. \end{cases}$$

end for
call $Adjust\_Reward(B, k, R^t, X^t)$

3.2.4 Annotation of Arms States

In previous section we mainly showed how we modify the reward update policy for graph based bandits. This section we discuss how we further modify the update policy by annotating the arms and incorporate new selection policy of UCB1 bandit leveraging that annotation to find the solution to our k-hitting set problem which can result in a better coverage when used in a Recommendation System. We introduce annotation of arms in the update policy and compliment this upgrade with a novel selection policy for graph based UCB1 bandits.

As described in our Online k-Hitting Set Problem, each element in U is considered as an arm in our bandit algorithm and hence a node in our graph. We further anootate these Arms (nodes) to be in either of the following two states:

Definition 6 *An arm is **active** for a time step if it is currently selected by our bandit algorithm. Such an arm (node) have most outgoing edges in our graph.*

At the beginning, all arms are active because each of them are equally likely to be selected.

Definition 7 *An arm is **dormant** if it is selected by the bandit but found to be a member of a number of sets for which an active arm is already selected by another bandit at the same time. They are the low rewarding neighbor of an active node in the graph.*

These arms are dormant in a sense that they equally qualify to represent the sets they are member of.

Arms selected by bandits with no match in the incoming set receive a reward of 0 but stays active. Because those arms clearly indicates its failure to cover that a specific

incoming set at certain iteration, but our algorithm does not rule out its potential for covering other incoming sets from the collection.

3.2.5 State Transitions of Arms

Algorithm 6 shows a complete Graph Based UCB1 update policy for bandits with this annotation of Arms.

With the arms annotated, we define transition of an arm from one state to the other occurs in the following situations:

- An arm gradually becomes dormant from active with its repeated failure to be the first element match with the incoming set
- An arm gradually becomes active from dormant if it frequently matches with the first element from the incoming sets

Transition of an arm from the dormant state to an active state potentially results in covering more incoming sets. A dormant arm can become active with a the update policy as well as the our adaptive selection policy.

We adopted an Adaptive Simulated Annealing [6] technique for the transition of an arm from one state to the other because of the following reason where the change of state of an arm is denoted by α,

- A positive value for α indicates an improvised solution where a previously dormant arm is likely to achieve more reward than the current active arm. This value is also positive for an arm which is selected by the bandits for our hitting set but does not find a match in the incoming set.

Algorithm 6 Annotation of Arms for Updated Selection Policy

Inputs:
Graph at the end of $(t-1)^{th}$ round: G^{t-1},
Our constructed set: R^t
Set of all k bandits: B
A fixed integer: k
Output: Updated graph after t round: $G^t(V, E)$
Pick a random set X^t from C
Create a set of directed edge E^t such that for each $e(i, j) \in E^t$:
v_i represents the element which is the first element matched with an element in R^t and
v_j represents any matched elements clicked in R^t
$E = E \cup E^t$
$C =$total number of items in R^t which matches with those of X^t
Update $w_n \in W$ by following:
for all $n \in N$ **do**

$$w_n = \begin{cases} w_n + [1 + \frac{1}{C}], \\ \text{(if } v_n\text{is the only item clicked} \\ \text{or has one or more outgoing edge(s) in } E^t) \\ \text{And mark } n \text{ as } \textbf{Active} \\ \\ w_n + 1, \\ \text{(if } v_n \text{ has an incoming edge in } E^t) \\ \text{And mark } n \text{ as } \textbf{Dormant} \\ \\ w_n \\ \text{otherwise.mark the arm as } \textbf{Active} \end{cases}$$

end for
for all $j = 1$ **to** k **do**
Feedback the j^{th} Bandits with the following reward:

$$F_{jt} = \begin{cases} \log(w_j), \\ \text{if } X_j^t = 1 \text{ for the } j \in R^t \\ 0, \\ \text{otherwise.} \end{cases}$$

$update(UCB1_j, F_{jt})$
end for

- A negative value for α shows a downward in solution quality where previously active arm becomes dormant in the current iteration.

- A value of 0 for α denotes the similar quality for solution, for which a state transition does not promise any improvement.

We use the direction of the change in α to modify our arm selection policy of UCB1. We assign an annealing parameter r initialized with 0 for each arm of every bandit. We update the counter in the following manner:

- If a dormant arm from the previous iteration becomes active in current iteration, then r is set to 0.

- If (1) an active arm from the previous iteration becomes dormant in current iteration, or (2) an dormant arm selected by the bandit fails to get reward in current iteration and goes back to active state then r is incremented by one.

- If an arm stays in the same state then the value of r will be unchanged.

Our modified arms selection policy is shown in Algorithm 7. The relationship between α and the counter r is the following:

$$r_n = \begin{cases} 0, & \text{if } \alpha > 0 \\ r_{n-1} + 1, & \text{if } \alpha < 0 \\ r_{n-1}, & \text{if } \alpha = 0 \end{cases} \tag{3.1}$$

where r_n denotes the r value of a specific arm of a bandit at current iteration n.

Algorithm 7 Selection Policy for Bandits

Input: $UCB1_i(N)$ the i^{th} UCB1 bandit with N arms,
$N \setminus R_{i-1}^t$ the number of available arms
Output: The arm with highest UCB value for the i^{th} UCB1 bandit
if none of the available arms have been tried at least once, randomly select an arm
otherwise select the arm with the highest UCB value by the following policy:
for all $i = 1$ **to** N **do**
    if n is an *active* arm, UCB value for the arms is
    $x_i + \sqrt{\frac{2\log(f)}{f_i}}$
    if n is a *dormant* arm, UCB value for the arms is
    $x_i + (1 - \exp^{-1/\theta})\sqrt{\frac{2\log(f)}{f_n}}$
    where $\theta = 1 + \log(1 + r_n)$
    and r_n is the value of the counter r for arm i at iteration n
end for
Return the arm with highest UCB value

While selecting an arm for each iteration, every bandit needs to update the value of r. If an arm is found to be in dormant state, UCB value for that arm is smoothed by the parameter θ in following manner:

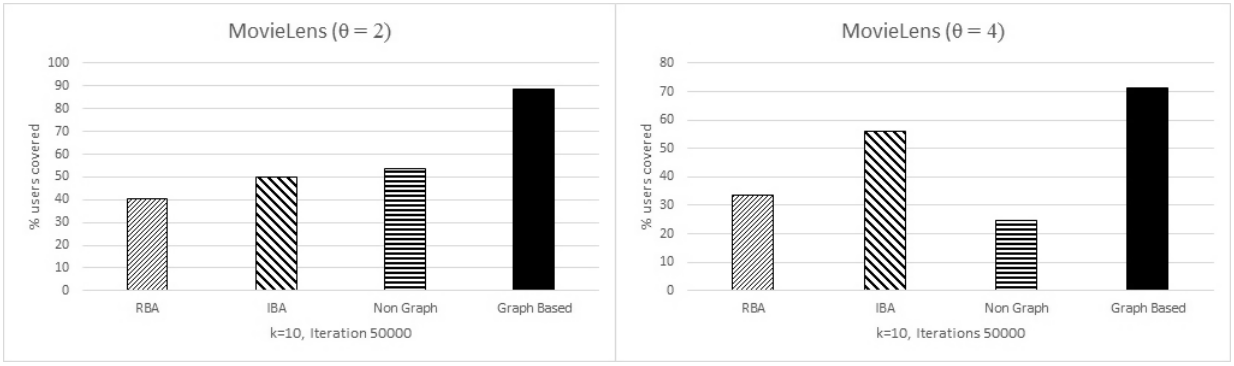
$$\theta = 1 + \log(1 + r_n) \tag{3.2}$$

Therefore, UCB value for every dormant arm i for a bandit is:

$$x_i + (1 - \exp^{-1/\theta})\sqrt{\frac{2\log(f)}{f_i}} \tag{3.3}$$

By setting $r = 0$ for an arm which becomes active from a dormant state, we keep the scaling factor in our algorithm same as UCB1 algorithm. This gives our algorithm enough opportunity to balance between exploration and exploitation. On the other hand, by increasing the value of r for an previously active arm which transformed as dormant in current iteration, we scale down the UCB value of the arms selected by the subsequent bandits by $(1 - \exp^{-1/\theta})$. But these arms can be reset to active in case it was not found in certain incoming sets. This way, we give the dormant arm ample scope to be active.

As a dormant arm with a reward update of 0 gets back to active state, our algorithm needs to distinguish this transition from the transition where a dormant arm becomes active by getting selected by the bandits for its high UCB value. In former, we increment the counter r associated with that arm, in the later, we reset the counter to 0 which helps us leverage the value of r to adjust the exploration-exploitation trade-off appropriately.



(a) For dense user relevance vector (b) For sparse user relevance vector

Fig. 3.6.: Comparison result with different high and low θ values for Movielens data

3.3 Empirical Evaluation

In this section first we show the performance improvement by our proposed graph based reward update policy over the existing non graph based methods. Then we show how incorporating our proposed selection policy facilitates the coverage and diversity of recommendation set. We used publicly available Movielens1000 [21] data set for our experiments. Movielens data set consists of 943 users rating on 1682 movies. The real valued ratings has been converted to binary (relevant or not) by using a threshold θ as rating ranges between 1 to 5. We show our result for $\theta = 2$ and $\theta = 4$ respectively in figure 3.6 (a) and (b) . Our recommendation set selects $k = 10$ movies each time for a random user out of available $n = 1682$ movies. This makes our method run to solve for more than a thousand armed bandit problem - which, to the best of our knowledge has never been tried before.

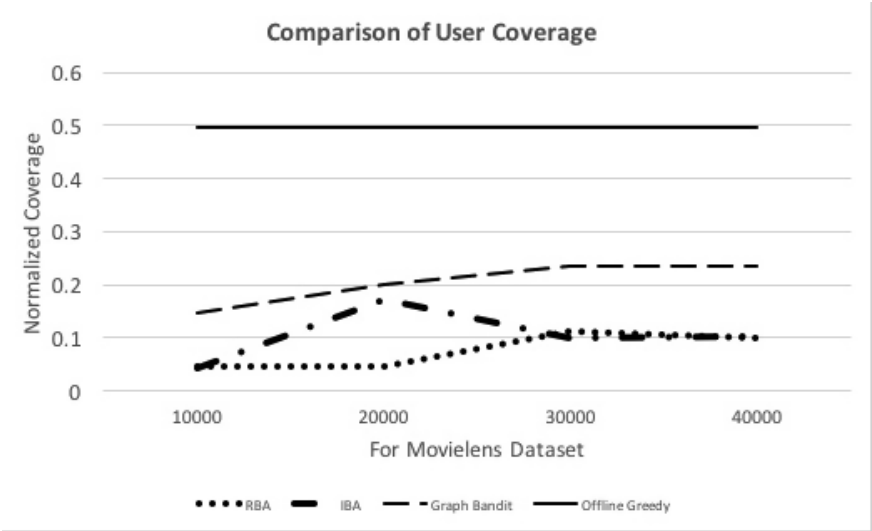


Fig. 3.7.: Average Coverage of each Item in Recommendation Set after every 10K rounds for dense Movielense dataset with low threshold using RBA, IBA, Graph Bandit compared with with offline greedy approach

3.3.1 Impact of Sparsity in User Vector

From the experiments with Movielens dataset, it can be observed that, on an average, our graph based bandit outperforms all other methods. For a high value of θ, performance of graph based bandit is much higher than our previous non graph based approach. In fact, non graph based approach is the worst one in terms of performance as our cutoff value θ rises. This is because our binary value for users' choice is based on a high cutoff value for θ and accordingly more diverse users' choices are identified which is better handled by the graph based approach. Figure 3.6 shows this effect.

3.3.2 Coverage of Different Sets

After incorporating the proposed selection policy for graph bandits, we further compare our method with other online methods as well as greedy offilne method in terms

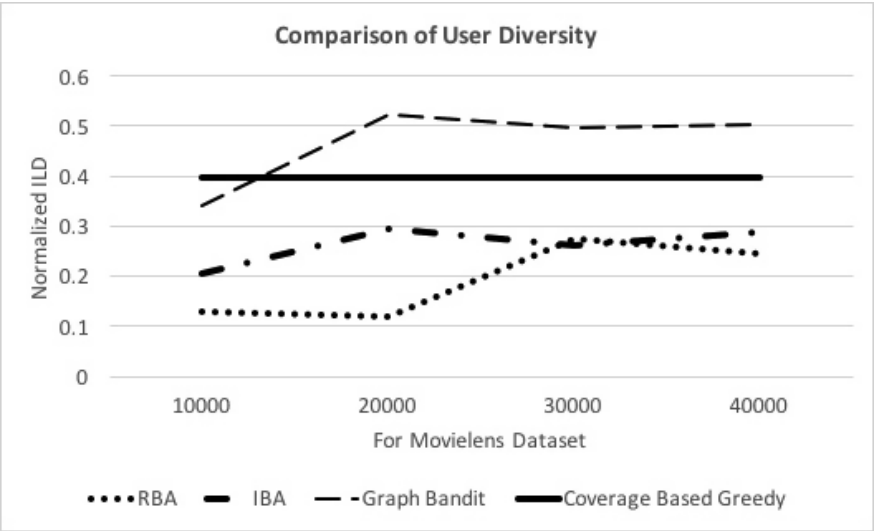


Fig. 3.8.: Normalized ILD of the Recommendation Set after every 10K rounds for dense Movielense dataset with low threshold using RBA, IBA, Graph Bandit compared with with coverage based greedy approach

of its user coverage. In this set of experiments, we show the result of Movielens dataset with $\theta = 2$ but other threshold value for θ also gives similar results. We evaluate the average performance of each item in the recommendation set after 10K, 20K, 30K, 40K and 50K iteration averaged over almost 1K static user vectors. This gives the online algorithms ample observation of total population to be benchmarked with offline greedy coverage shown as the upper bound in figure 3.8 normalized by the total population.

3.3.3 Diversity of Recommended Set

We compared the effectiveness of recommendation set after each 10K iteration upto 50K to observe its performance on capturing diversity. We used the popular Intra-List Distance (ILD) [14] metric where we pairwise distance between items in our recommendation set is defined by $dist(i, j) =$ number of user vectors (sets) containing

item i but not j. The result is shown in figure 3.7. As observed by this experiment, diverse user types are covered more by our method than others. Our method performs better than offline coverage based greedy method which maximizes the number of users covered but not essentially captures diversity.

3.3.4 Reduction of Arm Space

We found that the similar coverage of different types of users found within a shorter amount of iteration if we incorporate our selection strategy with the updated rewarding mechanism for the graph based bandit.

As for the complexity of the algorithm, [40] shows that lower bound of the performance of RBA can be measured by: $(1 - 1/e)OPT - O(k(R(T))$ where OPT denotes the optimal performance scaled from the offline greedy approach [22], k is the number of bandits, $R(T)$ is the regret for that specific type of Bandits. Later for IBA [28], it is proved that for UCB1, this bound is $(1 - 1/e)OPT - O(kNlog(T))$ where N is the number of arms. For our case as the number of effective arms are subset of nodes in the graph of N nodes, the regret can be reduced more than $O(kNlog(T))$, hence the lower bound of performance gets higher.

The reduction of average active arms count is depicted in figure 3.9. Figure 3.10 shows the average coverage over 50K iterations. It is found that, with our novel selection policy, the coverage of same fraction of overall user is achievable within 10K iteration, whereas without this modification the graph based bandit algorithm takes around 20K to

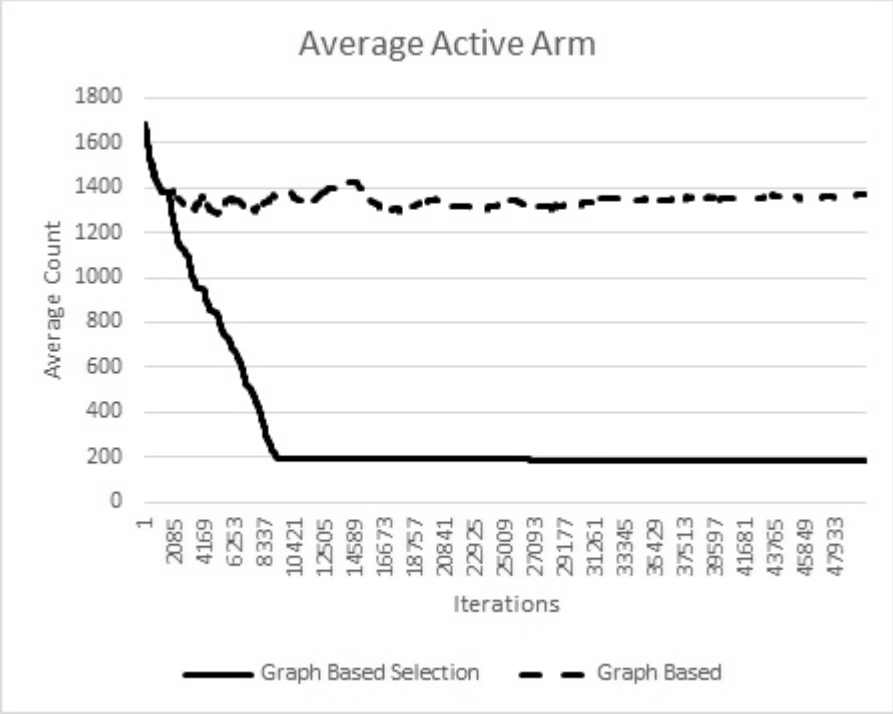


Fig. 3.9.: Reduction of Active Arm count when state transition mechanism is applied is used with the Graph based bandit

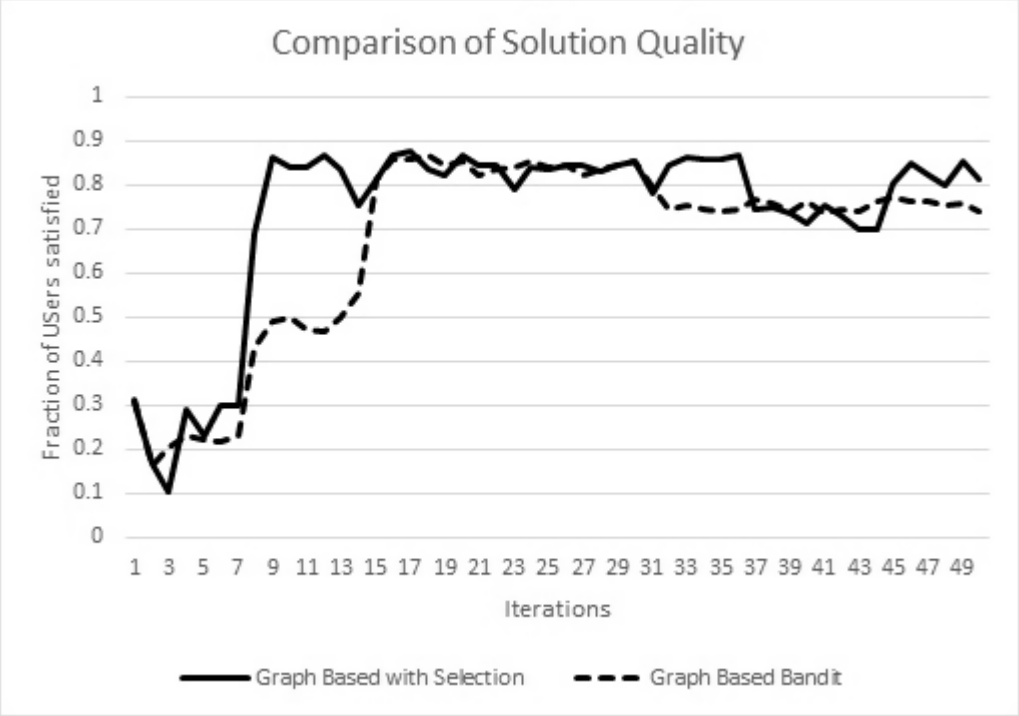


Fig. 3.10.: Faster convergence of solution is achieved when state transition mechanism is applied with the graph based bandit

attain the same outcome. As active arms are identified faster, our algorithm produce more efficient and stable set.

3.4 Parallel & Synchronized UCB2

Our earlier work [42] shows that IBA can be improved to capture diversity with a simple but effective rewarding technique that inherently takes account of correlation gap [4] between the distribution of items so that items picked by same type of users can be grouped together and one representative item from them can be selected by the recommendation system to cover all users of that type. To ensure coverage, we introduced graph based mechanism. However, all these methods discussed in the previous sections used UCB1 bandits for building the recommendation systems as no substantial modification of UCB1 is necessary to run multiple bandit instances in parallel. Attracted by the better theoretical regret bound, we, for the first time, attempt to run multiple parallel instances of UCB2 and face the technical challenges mentioned in the following Section.

For a recommendation system, at each time t, from n available items (arms), k items needs to be picked by the algorithm. Like IBA, [28] we instantiated k instances of multi-armed bandits, each having n arms to select from. But unlike IBA, once an item has been selected to place at any of these k slots by the k^{th} bandit, choosing that option by other bandits can get penalized so that remaining spaces can be filled in by rest of the available options. Thus a set of k non-identical elements is constructed. A random user's choice is compared with this recommended set and accordingly reward is fed back to the associated bandits.

Apart from ensuring unique selections by each bandit, in this section we also introduce the synchronization technique for epoch for arms selected by multiple instances of UCB2,

so that parallel execution of a number of bandits is possible. We also update α parameter for UCB2 as a function of time for online simulation with real datasets.

3.4.1 Technical Challenges

Some of the major technical challenges of running multiple instances of UCB2 bandit on real datasets are understanding the influence of the parameter α which is responsible for the tradeoff between exploration and exploitation of the arms and impact epoch. We also target to construct a recommendation set which fulfills the need of variety of users within its limited size. We came up with an algorithm for multiple instances of UCB2 which is robust enough to follow the basic assumptions of the basic UCB2 algorithm but at the same time relaxed enough to face the adversaries caused by the skewness of real data. In this section we discuss the implementation challenges we faced to realize this idea.

Influence of α

In UCB2, α has the following effects:

- For a fixed epoch count (r_i in UCB2) for any arm i, smaller α results in small epoch length (R_i) and vise versa. Because R_i is the difference of exponential function (τ) of α for two consecutive epoch counts r_i and $r_i + 1$ for arm i.

- Impact of increasing α makes a greater difference in epoch length for an arm i with higher epoch count r_i than that with a smaller one. Higher epoch count for an arm i already manifests that that arm has passed the exploration phase and have enough reward potential to be exploited.

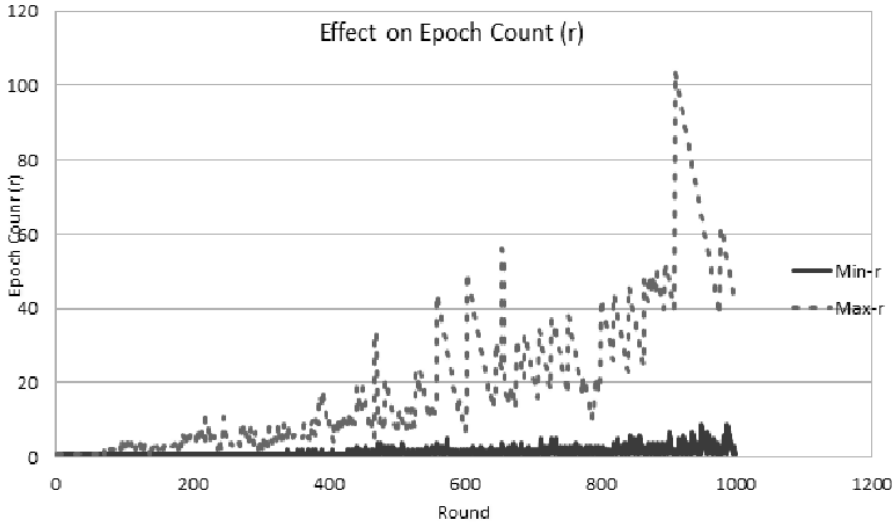


Fig. 3.11.: Upper (dotted) line showing max epoch count among all arms where the lower (solid) line expressing that of the corresponding min one for 1000 random sampled users of MovieLens

- For a fixed α, at any time step, for an arm i, bigger r_i results in larger epoch length (R_i) for the same first reason mentioned above.

As in the initial phase, our intention is to try out every possible arm with fairness (exploration), we decided to keep α small [based on first effect]. To select the rewarding arm more, every time we select an arm i, if it has already been chosen for a higher epoch count r_i, we choose it for a larger R_i for UCB2 (geared towards exploitation mode). We achieved this by updating α with an increasing function of time [according to 2nd and 3rd].

Adjustment of Epoch

As each individual bandit instance has its own epoch counter r_i for each arm i and r_i impacts the epoch length for i, selecting a common value for epoch count to synchronize

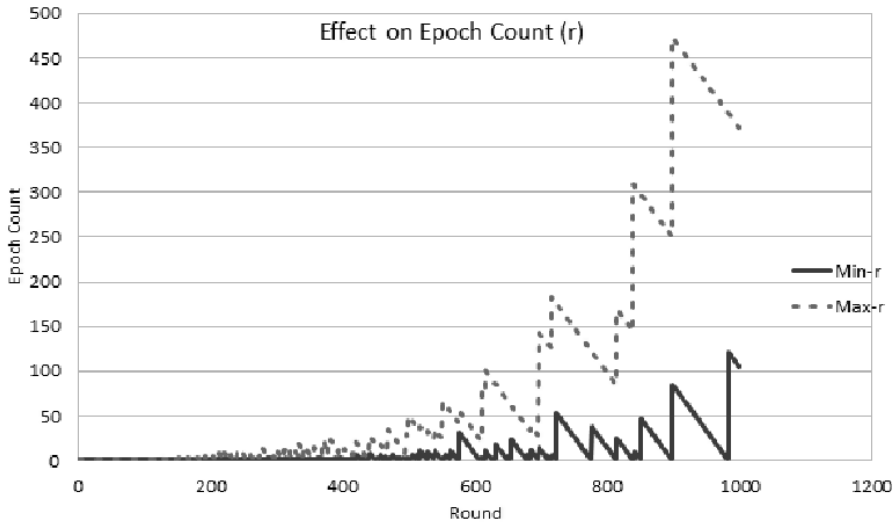


Fig. 3.12.: Upper (dotted) line showing max epoch count among all arms where the lower (solid) line expressing that of the corresponding min one for 1000 random sampled users of Jester

over all the arms that are selected by different bandits is necessary to generate a fixed recommendation set to show different users over a specific period of time (common epoch length R_i). We set the epoch count (r_i) for any arm i chosen by different bandits to the minimum of all the epoch counts of the all the arms selected by the bandits. Our motivation behind this simple strategy is that although in theory minimum epoch count and maximum epoch count can differ by any amount, in practice, there is a positive correlation between them. This phenomena is shown in figure 3.11 and 3.12 for two real datasets. It shows that if different bandit instances selects arms without the synchronization of the epoch, the maximum epoch length produced by an arm chosen by a bandit is proportional to that of the minimum one. We choose the min of them to set for all arms selected by different bandits because we wanted to be conservative on exploiting the selected arms too much and have room for exploration as users choices can shift to less exploited arms anytime.

Penalty for Repetition

Our online learning algorithm strives to produce a substantially small but a diverse recommendation set from a large number of items, satisfying different user types. To maximize the coverage of different users, it is essential to accommodate unique items in the recommendation set. Repetition of an item in the recommended set adds no value to the set for the users. Moreover, it wastes space in the limited size recommendation set, which could be efficiently assigned to a new item which has more potential to cover a different type of user. When several instances of UCB2 bandits run in parallel, it is

difficult to ensure mutual exclusiveness of the arms they pick. We take care of this issue by discouraging a bandit to select an item already in the recommended set under construction. We did it by penalizing the bandit for such a repetitive pick so each bandit instance is forced to select a unique item.

3.4.2 Solution Sketch

In this section we illustrate our approaches to overcome the challenges of parallel execution of a number of UCB2 bandit instances.

Updating of α

The built-in annealing mechanism in basic UCB2 bandits ensures that exploration will be less once the confidence bound becomes tighter. However, there is no mechanism to choose the value of α across the bandits and this can lead to increase in epoch length exponentially for an arm selected by any bandit. So initially we set $\alpha = \alpha_0$ to be small and slowly increase it with the number of t rounds. We used several techniques to gradually increase α which is opposite to starting with a higher temperature in a Simulated Annealing and cooling it down with time. It ensures that, at the beginning more preference are given to exploring the available options and diversifying the recommendation set with an intention to find suitable items to recommend and then eventually settling on exploiting the best performing items found so far according to users feedback. Two different approaches adopted to tune the initial parameter α_0 over time:

- Fixed-Increase Alpha (FinAlpha): In this method α_0 gets increased in a fixed rate η. Hence

$$\alpha_t = \alpha_0 + \eta t$$

For our experiment we set $\eta = (1 - \alpha)/t$ so that α stays within the bound (0,1) as required by UCB2.

- Logarithmic-Increase Alpha (LinAlpha): According to this technique α_0 experiences a logarithmic increase over time. Hence

$$\alpha_t = \alpha_0[log(t + 1)]$$

Synchronization of Epoch

To the best of our knowledge, to develop a recommendation set of unique items, UCB2 instances has never been tried. One of the challenges to deal with multiple instances of UCB2 is synchronization of epoch length across different bandits so that they expire at the same time to initiate a new set of recommendation from the users' feedbacks. We resolve this by simply setting minimum epoch count among all selected arms to their minimum. The logic behind doing so is very straight forward. As we observed that both min and max epoch count among all the chosen arms are positively correlated (in Figure 3.11 and 3.12) setting min epoch for all selected arms is a safe choice. Because, for an arm whose epoch count is higher than the minimum of all epoch counts, it can experience the amount of reward gained within the minimum epoch when it is realized at the end of that epoch length. And according to UCB2, if that arm is rewarding enough, it will be kept

on the recommendation for successive epochs as well without any loss. On the other hand, if it turns out to be non rewarding one, this policy assures that this 'bad' arm is fairly tried (atleast as much as the one with minimum epoch length).

Ensuring Unique Selections

In [42] Mahmuda et al. showed that if a recommendation system (with a fixed small number of items to recommend) could select only one from a group of related items, it would be able to satisfy all users of that user type. To accommodate maximum variety of users in the limited size recommendation set, we avoid repeating an item in the set. As k bandits instances of UCB2 is working together to construct the recommendation set, we made sure that an item picked by a bandit is not selected by another one. This has been done by penalizing the bandit responsible for picking the repetitive item by updating it with a 0 reward; which, in effect, increment the frequency for that arm so the average reward for that arm goes down.

3.4.3 Analysis of PAC Bound

In this section we justified our decision of choosing a small α value initially. We know from [1] that if reward of picking an item (choosing an arm j) is a binary valued random variable Y_j with an unknown distribution, then expected value of Y_j can be denoted by $E[Y_j]$ which is also unknown. For any $t \in T$, reward of pulling an arm j denoted by $Y_{j,t}$ is independent as both j and t varies and as j (arm) changes, the distribution changes.

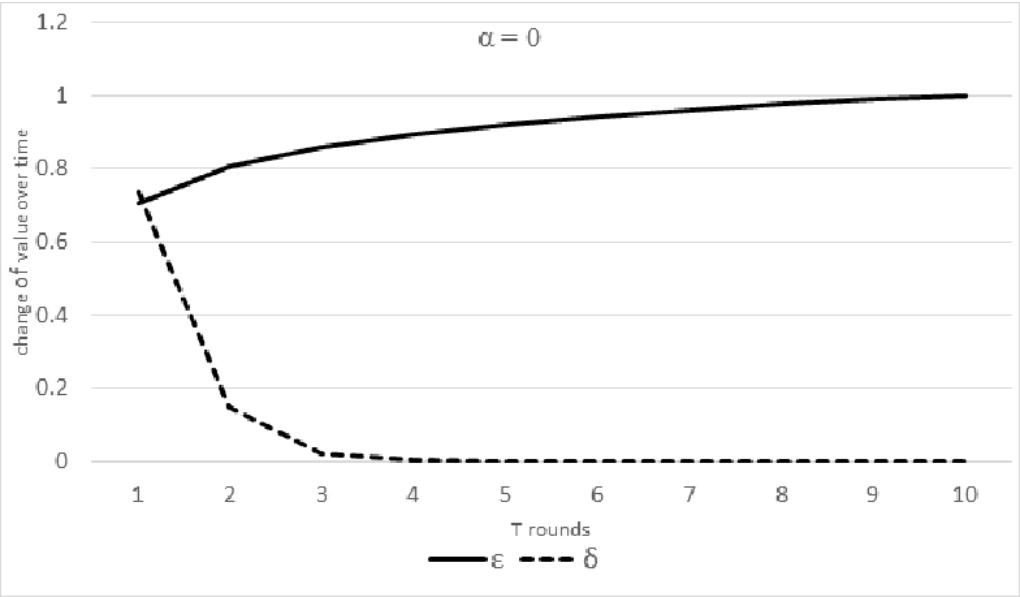


Fig. 3.13.: Upper solid line showing ϵ changes over rounds for UCB2. Lower (dotted) line is showing corresponding change in δ

According to them, let j be a fixed arm to be pulled for all T rounds. Then the average payoff of T rounds will be:

$$G_j(T) = \frac{1}{T}\Sigma_{t=1}^{T} Y_{j,t}$$

A classical approach to estimate the difference between this average and the long term expected reward is determine the Probably Approximately Correct (PAC) bound by using Chernoff-Hoeffding inequality [38] which is:

$$P(|G_j(T) - E[Y_j]| > \epsilon) < \delta \text{ where } \delta = 2e^{-2T\epsilon^2}$$

It shows that probability of the sampled average and true expectation deviates by more than ϵ is bounded by confidence bound δ. For UCB1, [1] showed that, assuming that exploration term causes the deviation in UCB1, and setting

$$\epsilon = \sqrt{\frac{2\log(f)}{f_j}}$$

in the right side of the inequality, the bound for δ becomes T^{-4} if a fixed arm is j always pulled.

As we know that the $\alpha \in (0, 1)$ for UCB2, we want to observe the effect of setting the exploration term of UCB2 to ϵ under the same condition with an α value as close as 0.

If we set $\alpha = 0$ in the UCB2 algorithm where $\tau(r_i) = \lceil (1+\alpha)_i^r \rceil$ for arm i we get,

$$\begin{aligned}\epsilon^2 &= \frac{ln(e) + ln(f) - ln(1)}{2} \\ &= \frac{1 + ln(f)}{2} \\ &= \frac{1 + ln(T)}{2}\end{aligned}$$

Here, $r_i = f = T$ as we pulled only arm i for all T rounds. Now if we set ϵ to this value in PAC analysis we observe,

$$\begin{aligned}\delta &= 2e^{-2T\frac{1+ln(T)}{2}} \\ &= 2e^{-T[1+ln(T)]}\end{aligned}$$

A smaller ϵ corresponds to smaller error tolerance level in approximating the expected value by the average. The target is to keep δ (the probability of keeping an error higher than ϵ) small. But if ϵ is too small, then δ will be large. Relationship between ϵ and δ for this analysis is depicted in Figure 3.13. We decided to keep the initial α value considerably small which results in a relatively small ϵ value to begin with. Even if α grows upto maximum 1, the increase it causes to ϵ over time produces a rapid decrease in delta which serves our purpose with keeping a respectable error bound.

3.4.4 Proposed Algorithm

Apart from tuning α over time and setting minimum epoch count for all bandit instances, according to our method, if a bandit picks an arm which is already picked by

another, it should get a reward of 0 and an increment of pull count of that arm effecting a decrease of average reward for that arm picked by the corresponding bandit. That bandit is given a chance to play again, till it finds a new element to add to the recommendation set. The target is to maximizes the pay off of the recommendation system while constructing the recommendation set for the next epoch. We record the epoch counts of all the selected arms and set the minimal one for all bandits. We keep showing this fixed set of recommendation to a number of the randomly selected users till the epoch lengths expires. Then all these users' relevance vectors are disclose it to the system to compare with our recommendation. If any of the items recommended matches with that of the user vector, the corresponding UCB2 bandit gets a reward of 1 for picking that item.

In this Section, we present our parallel and synchronized UCB2 algorithm for the recommendation system to pick k out of n items for users which is based on k-instances of UCB2 bandits. Initially we create k instances of UCB2 bandits with a fixed α value. We also declare a vector of length k which stores the epoch count of bandits on a specific time step. For each time step $t \in T$ we have an empty set S. Then (in line number 11) we use individual bandits for picking its best possible arm according to the UCB2 objective function and add it to the set to construct a k items recommendation. If a bandit picks an arm which is already picked by another, it should get a reward of 0 and try again till it gets a new item to add to the set (line number 12-15). We also record the epoch counts of all the selected arms and set the minimum for all bandits (as seen in line number 16, 19 and 21). We keep showing the user the same set of recommendation till the epoch length expires. During this time we pick random user relevance vectors in each time instance and disclose it to the system to compare with our recommendation. If any of the items

Algorithm 8 Proposed Algorithm

Input: n items
Output: k items
Initialize $UCB2_1(n), ...UCB2_k(n)$ **with** $\alpha = \alpha_0$
Initialize $epoch(1), ...epoch(k)$ **with 0**
for all $t \in T$ **do**
  $S^t \leftarrow \emptyset$
  for all $i = 1$ **to** k **do**
    $setAlpha(UCB2_i, \alpha)$
  end for
  for all $i = 1$ **to** k **do**
    $s_i^t \leftarrow selectArm(UCB2_i)$
    while $|S^t| < k$ **do**
      if $s_i^t \in S^t$ **then**
        $update(UCB2_i, 0)$
      else
        $S^t \leftarrow S^t \cup s_i^t$
        $epoch(i) \leftarrow getEpochCount(UCB2_i)$
      end if
    end while
  end for
  $e_{min} \leftarrow min(epoch)$
  for $i = 1$ **to** k **do**
    $setEpochCount(e_{min}, UCB2_i)$
  end for
  $L_{min} \leftarrow getEpochLength(e_{min})$
  for all $i = 1$ **to** L_{min} **do**
    Pick a random user vector from the dataset
    Display S^t **to user for and receive feedback vector** X^t
    Feedback:
    $$F_{it} = \begin{cases} 1, & \textbf{if } X_i^t = 1 \textbf{ for some } i \in S^t \\ 0, & \textbf{otherwise}. \end{cases}$$
    $update(UCB2_i, F_{it})$
  end for
  $t \leftarrow t + L_{min}$
  $increaseAlpha(\alpha, t)$
end for

recommended matches with that of the user vector, the corresponding UCB2 bandit gets a reward of 1 for that item. We increase the value of α at the end of each such round to propagate the effect of its update to the next epoch.

3.4.5 Empirical Evaluation

In this Section we show the impact of using our method for online recommendation to users. Publicly available dataset of Jester Project [19] and MovieLens dataset [32] has been used for this purpose.

The impact of both the increasing function on α against a fixed value for UCB2 is shown in the chart of figure 3.14 and 3.15 where we compared the performance of our synchronized and parallel UCB2 with that of UCB1. Two different fixed values of α we tried throughout the runs were 0.5 and 0.1 respectively. For Fixed rate increasing (FinAlpha) α we initialized α with these values for multiple instances of UCB2 and increase it with a fixed rate $\eta = (1 - \alpha)/T$. To simulate for logarithmic increase of α, it is initialized with 0.1 and gradually increased as a logarithmic function of time. For the sake of fairness of comparison, we imposed the same penalty structure for the UCB1 bandits for picking an item repeatedly which is already selected by another bandit.

This simulation has been done on the same jester dataset which [28] used for measuring the performance of IBA and RBA using UCB1 where the size of the recommendation set is 5. Their reported accuracy was at best 65% for IBA up to the same rounds we ran, where our strategy can gain upto 82.5%. For movieLens dataset, on an average, we could reach more than 92% coverage of users while we run upto 3000 rounds.

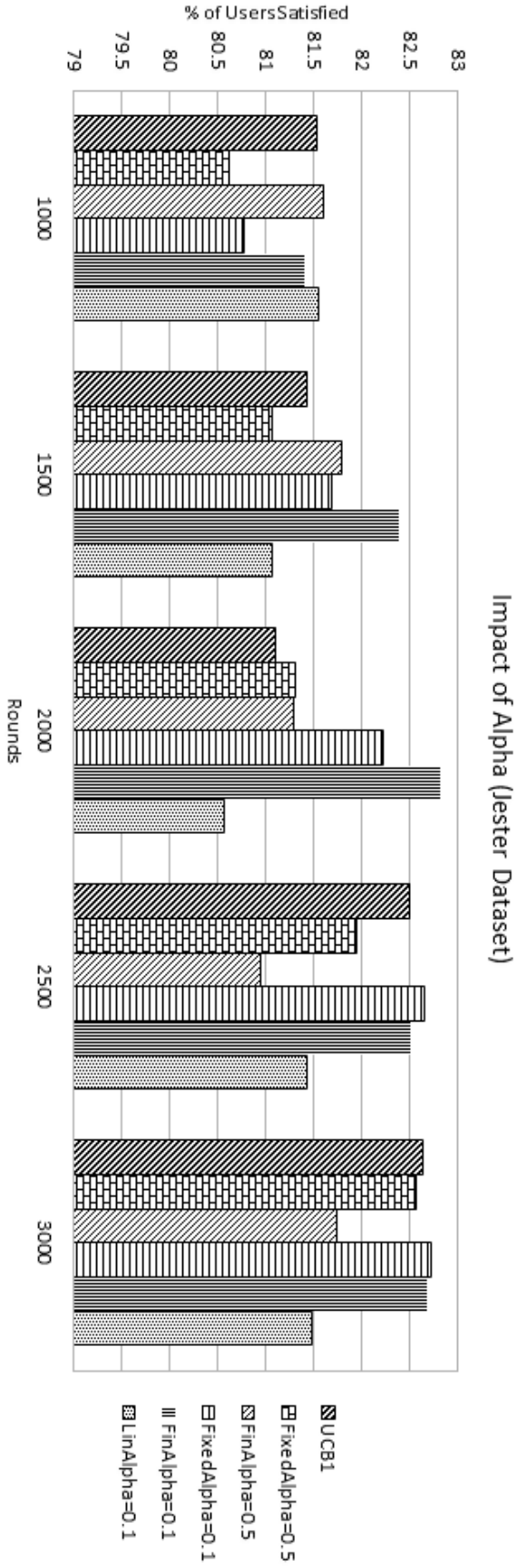


Fig. 3.14.: Effect of various tuning function on an initial small α for UCB2 against UCB1 on jester dataset

3.16 and 3.17. In addition to this comparative study, we also ran UCB1 based RBA and improved IBA (IBA-impr) proposed by Mahmuda et al. [42] which has a higher performance over the original IBA on the same datasets due to an improved reward structure for user coverage. We compared the performance our proposed synchronized UCB2 applying fixed rate increase (FinUCB2) and logarithmic increase (LinUCB2) in α over time where initial value of α is set to 0.1. We run the simulation for this experiment upto 100K rounds and took average of 5 runs for each simulation. For Jester dataset we allowed the recommendation set size was 3 and for movielens it is 10. The results of these experiments are shown in the figure

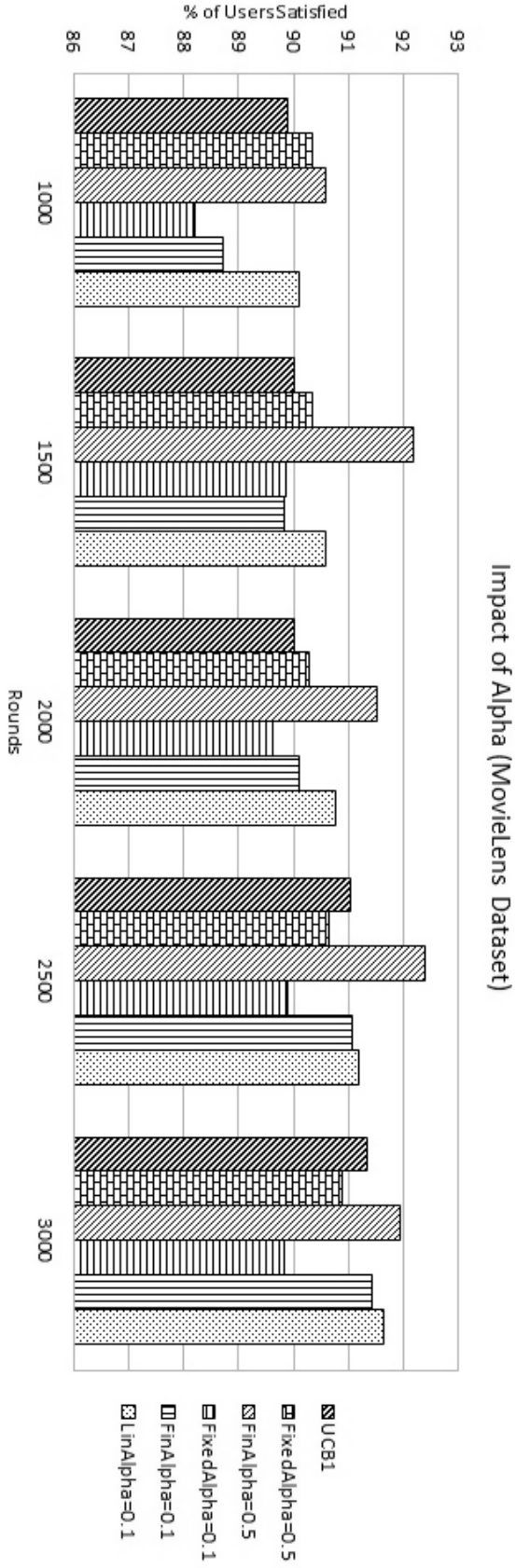


Fig. 3.15.: Effect of various tuning function on an initial small α for UCB2 against UCB1 on movielens dataset

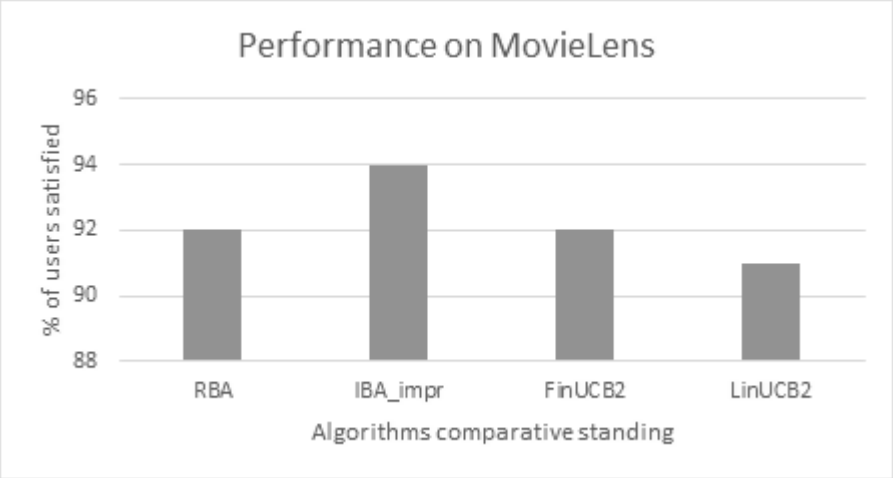


Fig. 3.16.: UCB1 based RBA and improved IBA compared against UCB2 with Fixed rate and Logarithmic increase of α, both initialized with 0.1 for movielens dataset after 100K iterations

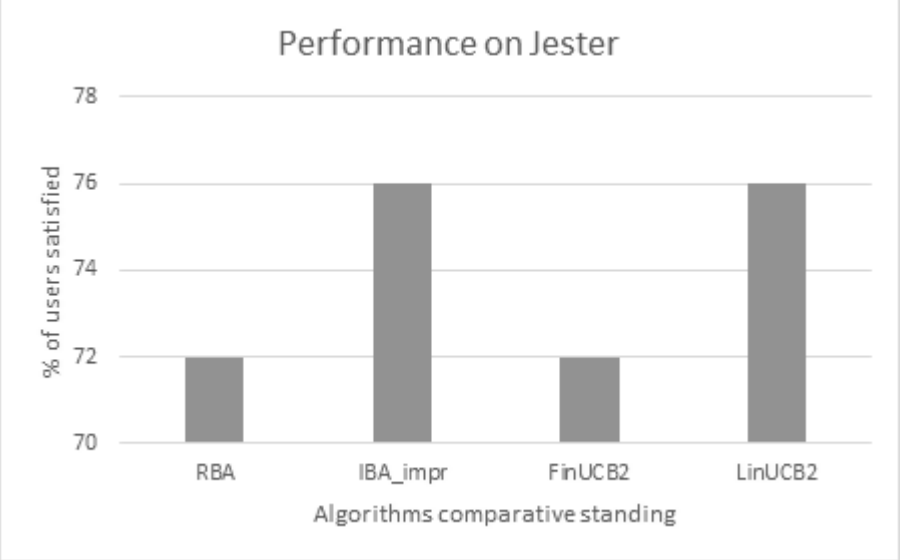


Fig. 3.17.: UCB1 based RBA and improved IBA compared against UCB2 with Fixed rate and Logarithmic increase of α, both initialized with 0.1 for jester dataset after 100K iterations

4. DYNAMIC PATROL ASSIGNMENT

The motivation for developing a model for Dynamic Patrol Assignment Problem (DPAP) comes from the increasing need for protecting major public facilities in response to global threats. Robots can be equipped with necessary defensive tools for this purpose, but they are expensive to build. Therefore we need to consider the constraint that only a limited number of such robots can be deployed for patrolling a comparatively large number of routes. The purpose of this deployment is to defend the routes from potential attacks.

However, unlike most other routing problem, which follows prescribed patterns, patrol activities should ideally exhibit randomness against adversarial observers who might plan an attack. We realize that DPAP needs to be considered as an online learning problem as opposed to an offline planning problem. However, as it mentioned above, it is constrained by the limited number of cooperative robots aim to achieve effective patrolling over a large number of routes. Hence, to attain high group performance, these learning robots need to coordinate. We develop a centralized online scheduling algorithm for this purpose. Our algorithm can serve as a high-level strategic decision maker to assign the patrolling robots to the routes to withstand attacks posed by different attackers.

4.1 Game Theoretic Formulation

At each round of patrolling, a Patrolling Robot (PR) is assigned to a route out of a predefined number of routes. For the sake of simplicity, we assume each such routes take equal amount of time for the assigned robot to patrol. Observing the history of the assignments of PRs, the attacking team selects an attack strategy as its best response. Each attack strategy is capable of posing attacks to a certain number of routes at the same time. We can consider that selecting an attack strategy is a plan of action attacking team executes with the different attacker instances over some routes at the same time. If an attacker attacks a route which is being patrolled by a PR, the attack is defended. Our objective is to develop a centralized online algorithm to assign PRs to different routes where the best response of the attacking team changes over time.

The reward of a PR comes from defending a route against potential damage posed by an attacker when they encounter each other. However, failure to withstand an attack penalize the team of PRs. Under this circumstance, a reward of defending a route is stochastic as the vulnerability of a route can rise and fall over time by the opportunistic attackers. This situation can be modeled as a zero-sum repeated game between a PR and an attacker. Figure 1 shows the dynamics of such a game between two opponent where each can choose one route out of two available routes: r_1 and r_2. Later, we describe how we extend this as a Stackelberg Game between two competing groups over multiple routes.

The game matrix at the center of Figure 4.1 denotes a PR to be the row player and an attacker as a column player. r_i is the route id. Above the horizontal line, we show the game states where PR as a defender who is constantly selecting r_1 and below the line we

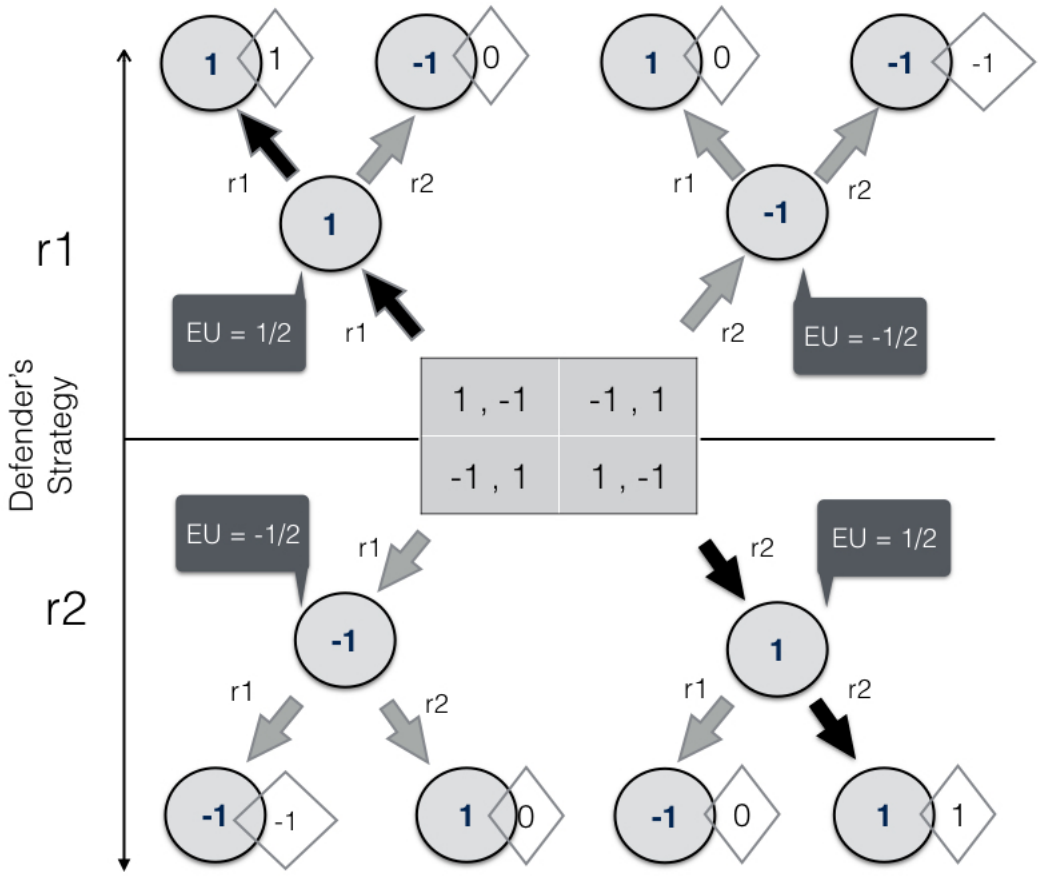


Fig. 4.1.: Zero-sum repeated game between a PR and an attacker, the row player is a defender whereas the column player is an offender. Strategy space consist of two routes

show it choosing r_2. The state nodes are labeled with the utility value gained by the PR whereas each edge between the nodes is labeled by the corresponding attacker's choice. Two rounds have been depicted in the figure to portray that, highest expected utility (marked as EU) for a PR after around can be achieved using this game matrix where both the PR and attacker are making same choices of routes. The average utility of the PR after this two rounds are denoted inside the diamond. With the dark arrow edge, it is highlighted that higher reward is received when the defender is patrolling the route an attacker selects (i.e., a PR and an attacker instance confront each other).

However, observing the earlier deployments of the PRs, attackers can change its route in an adversarial manner. Therefore, a PR cannot keep choosing same routes in too many consecutive rounds. The patrolling schedule must balance exploitation (minimizing violations of offenders) with exploration (maximizing the omnipresence of PRs) to make the next assignments of PRs hard to be predicted by the best response attacker. We can model this trade-off by solving a bi-objective optimization problem using bandits algorithm [29]. Like [31], we also consider this problem based on the dynamic interaction through a game between the PRs and attackers. However, instead of modeling it as a planning problem and solved it using Monte Carlo Tree Search [27] which can result in high branching factor, we formulate it as an online reinforcement learning problem.

4.2 Comparison with TSP and the VRP

Two problems of the same nature as DPAP are the Traveling Salesman Problem (TSP) [43] and the Vehicle Routing Problem (VRP) [9]. The classical TSP seeks the shortest

distance to visit all the vertices in a graph. The coverage of vertices needs to be maximized while minimizing the cost of traveling. On the other hand, in the VRP, vehicles must successfully purchase items from some sites and deliver them to other sites to complete their tasks and earn the corresponding rewards. Unlike DPAP, none of these problems involves any adversary, and there is no risk involved in these tasks.

A closely related problem addressed by [23] is Path Planning Problem involving known risk. Later [51] extended that problem with a scenario where the risk is fixed but unknown and needs to be learned. For our problem, the attackers' selection of routes changes dynamically, and hence the risks evolve in an adversarial way. Therefore, the risks need to be approximated by the PAs, using online learning.

4.3 Patrolling Robots as UCB1 Bandits

We consider each PR in DPAP running a multi-armed bandit algorithm and all the available routes to be its arms. We assume the number of routes is fixed for this problem. Therefore, selecting a route by a PR is same as choosing an arm to pull from the available arms in a Multi-Armed Bandit problem. In our context, the gambler is the PR trying to learn the reward distributions of the routes online with an intention to select the most promising one which has higher expected utility for its team than the attackers.

The most popular stochastic bandit algorithm, UCB1 [5] has been used for that purpose where each PR runs a UCB1 bandit instance to select a route i from n available routes to maximizes the UCB value: $x_i + \sqrt{\frac{2\log(f)}{f_i}}$ where x_i denotes the current average

reward of the route i and f_i denotes the number of times route i has been picked so far in total t rounds. Here f denotes the total count of all routes selected so far as $f = \Sigma_{i=1}^{t}(f_i)$.

The UCB values corresponding to each route can be normalized to the probability for that route to be selected by a PR. This probability changes over time according to strategy selected by the PR and the attackers. $selectPRStrategy$ select a route for a PR by a bandit out of all available choices. In each round, the best response attack strategy is chosen by the attacking team to attack routes which are least defended by the PRs. As the adversarial attackers' selection of routes changes in a stochastic manner, PR bandits try to learn the best assignment of routes for the next round by adjusting their probabilities based on the updated UCB value of the routes (arms of the corresponding bandits). Reward feedback to the corresponding bandit is received by the $update$ function in the We extended our graph-based bandit algorithm according to this. Algorithm 9 shows this extension.

Algorithm 9 Patrolling Algorithm

Input: n routes
Output: k routes
Initialize k **PRs with** $UCB1_1(n), ...UCB1_k(n)$
for all $t \in T$ **do**
  $S_0^t \leftarrow \emptyset$
  for all $i = 1$ **to** k **do**
    $selectPRStrategy(UCB1_i, N \setminus S_{i-1}^t)$
  end for
  select the best response attack strategy
  Match S^t **to the attack vector and receive feedback vector** X^t
  Feedback:
  $F_{it} = \begin{cases} 1, & \textbf{if } X_i^t = 1 \textbf{ for any } i \in S^t \textbf{ which matches with an attacker's choice} \\ -1, & \textbf{otherwise}. \end{cases}$
  $update(UCB1_i, F_{it})$
end for

4.4 Stackelberg Security Game (SSG) for Route Selection

The problem of optimal online assignment for PRs is hard as the corresponding bandits are required to coordinate their decision-making with the ultimate goal of achieving maximum team performance [33]. This is aimed at foiling attacks to increase the reward, which involves frequent visits to every route based on the crime intensity and frequency. To this extent, this online learning problem requires optimization of team reward which can be formulated as a Stackelberg Security Game.

For a generic Stackelberg game, there are two players, a leader, and a follower. This non-cooperative game is originally defined as a duopoly in a homogeneous product market to between two competitor firms for the decision over the optimal quantity to produce that product [50]. Leader firm moves first (gets first movers advantage of making a decision earlier than its competitor) then the follower follows sequentially. Leader must know the ex-ante (before the event) that the follower observes its action. Also, the leader has the power of commitment (i.e to move observably first and once the move is made, it cannot be undone).

However, the players in a Stackelberg game need not represent two individuals. They could also be two rival teams. Members of each team cooperate within themselves to execute a joint strategy to undermine the other. For our scenario, these two teams are PRs and the attackers. According to our definition, we consider the defending team as the leader in the game who belong to the law enforcement group and running multiple bandit instances of PR for online learning. We assume attackers are the follower who is the

members of a terrorist organization responding with its best actions observing the moves taken by the PRs so far.

In Stackelberg Security Game (SSG) [33], the attacker has a perfect model of how defender (i.e PR) moves. [12] developed an offline model using Integer Linear Programming using backward induction where leader considers what the best response of the follower would be before committing. The leader then picks its move to maximizes its payoff anticipating the predicted response from the follower. For the online problem, we consider defending team of PRs to be the leader who is continuously learning whereas attacking team changes its strategy in an adversarial manner.

In SSG, each player has a set of pure strategies: for our problem the strategy space is the set of all routes. A mixed strategy of the defender is a probability distribution over pure strategies and is represented by a probability vector. As mentioned earlier, we compute the strategy of a PR by normalizing the UCB values of the corresponding routes for the associated bandit. Therefore, each normalized value indicates the probability of a route to be selected by that specific PR.

We formalize a zero sum repeated SSG between attacker and defenders similar to [7]. Our SSG model has the following components:

- Time Horizon T: the number of rounds.
- Set S of total n number of routes: $S = \{S_1, S_2, S_3, ...S_n\}$
- A set D of k: Defenders (i.e PR): $D = \{d_1, d_2, d_3, ...d_k\}$ where $k << n$. Each of them running a UCB bandit to compute its strategy against the attacking team and selects non identical routes to patrol.

- A set A of m independent attack strategies: $A = \{a_1, a_2, a_3, ...a_m\}$. Each of these attack strategies can have multiple attacker posing attacks at the same time to different routes. In each round, an attack strategy is selected by the attacking team to pose attacks on the routes that specific attack strategy covers.

- A set of Strategy:

  - Attacking team knows the history of deployments of PRs and selects an attack strategy capable of attacking multiple routes (one by each attacker instance) at the same time. So each attack strategy is defined by a zero-one vector of $n \times 1$ where an entry with value 1 marks the route an attack strategy will attack.

  - Every PR bandit runs a defender bandit d to play a strategy based on the probability derived from the UCB values. It is an $n \times 1$ probability vector P_d proportional to the UCB1 values such that with $p_d^i \in P_d$ denotes the probabilities of a route i among n available routes to be selected for patrolling by the PR d.

- Utilities: Payoff for both the PRs and attackers are defined in the following manner when an attack is posed:

  - For a PR d, let $u_d^c(i)$ and $u_d^{\bar{c}}(i)$ be its payoff when the route i is attacked and covered and not covered by d respectively. Therefore under the probability p_d^i, the expected utility of the PR when route i is attacked is given by

$$U_d(i, p_d^i) = u_d^c(i)p_d^i + u_d^{\bar{c}}(i)(1 - p_d^i) \tag{4.1}$$

– For an Attacker a, let $u_a^c(i)$ and $u_a^{\bar{c}}(i)$ be its payoff by attacking the route i which is covered and not covered by the a PR respectively. Therefore under the probability p_d^i, the expected utility of the Attacker when route i is attacked is given by

$$U_a(i, p_d^i) = u_a^c(i)p_d^i + u_a^{\bar{c}}(i)(1 - p_d^i) \tag{4.2}$$

4.5 Learning from the Best Response Opponent

In Figure 4.2, we show the dynamics of game we defined in figure 4.1. The associated graph illustrates the optimal choices of the players against each other in this zero-sum game where bottom right sub figure shows the equilibrium point for a game if PR would respond back with best strategy instead of using any learning algorithm. As there is 50-50 chances for each player in such a game setting, we want to equip the PRs with bandit based learning.

The challenge in this game theoretic formulation is to maximize the expected payoff of the PRs as a team of bandits. As mentioned before, we want to leverage UCB1 algorithm for this purpose where the utility of every arm of each PR is proportional to its associated UCB value. At time $t = 0$, k uniform random, non identical routes are chosen by k PRs by Algorithm 1. If $P_d(t)$ is this probability vector at time t for all the routes by a PR d, then in pure strategy game, the next PR d' will ignore the highest UCB valued route of d to let d select that and adjust its own probability vector $P_{d'}$ for the remaining routes. In this way, the team of k PRs choices of routes are set against the best attack strategy. For mixed strategy game, we further impose a probability distribution over all the routes

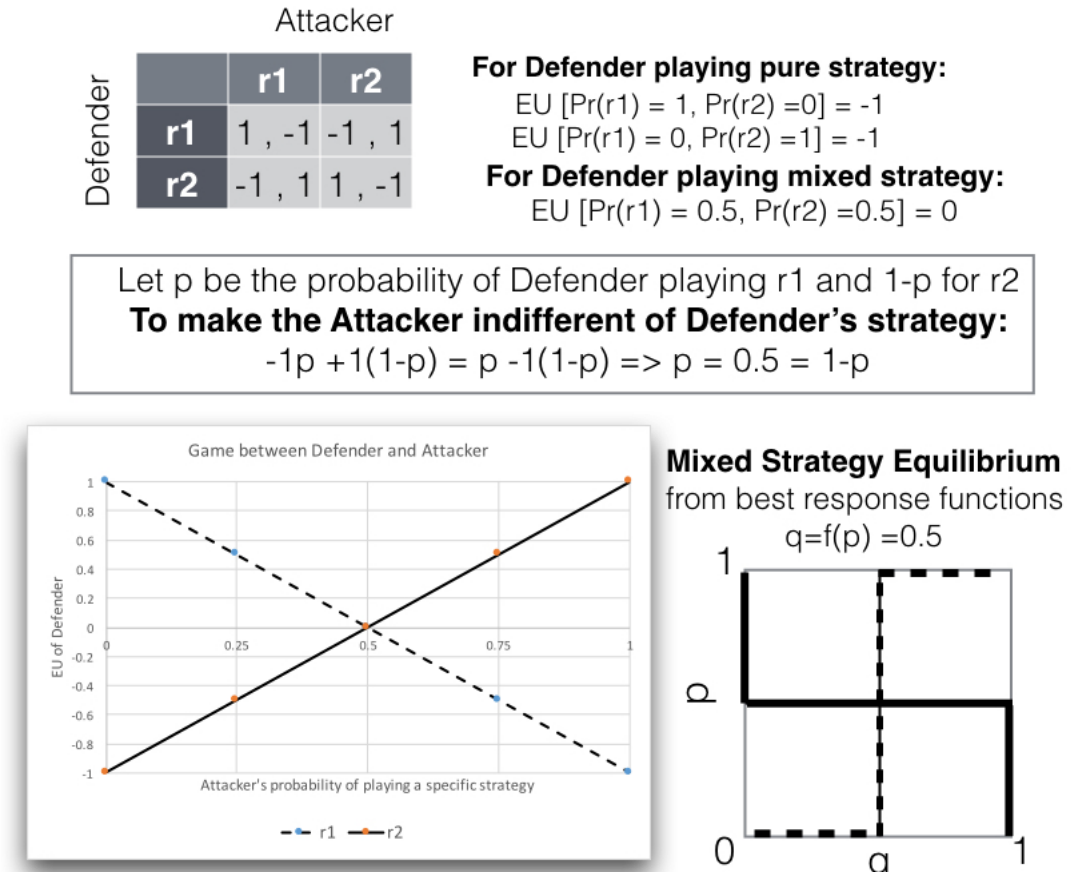


Fig. 4.2.: Mixed Strategy Equilibrium for single shot Defender-Attacker Zero-sum Game for one attacker, one defender playing over two routes r1 and r2

available to a PR. The vector $P(t)$ is the probability computed from normalizing the UCB values of all candidate routes to be selected by the group of PRs according to the above process for a certain round t. Then a single attack strategy a is selected which can attack the route given all such vectors $P'(t)$ till the round t. We define a to be the best response from the attackers as $b_a(t)$:

$$b_a(t) = argmax_{i \in n} U_a(i, P'(t)) \tag{4.3}$$

Upon receiving the feedback from the choices made by the attackers at round t the PRs subsequently adjusts their strategies by updating the UCB values as bandits to maximize their expected payoff as they confront the attackers and gets rewarded according to the game we defined. We intend to maximize expected payoff of the PR d which is:

$$E\left[\ \Sigma_{t=1}^{T} U_d(b_a(t), P'(t))\right] \tag{4.4}$$

The overall payoff of both the teams are measured by summing up their individual utilities as the members of the corresponding team. When normalized, for a bandit $d \in D$, its UCB values corresponds to the probability $p_d^i \in P_d$. Without the loss of generality, P_d preserves the relative ranking of all the routes to be chosen by bandit d. As the team of PRs play more rounds against different attack strategies, it learns to minimize its regret [7] and maximize the expected team utility.

4.6 Balancing Exploration and Exploitation

In the Chapter 2 we mentioned how we used Upper Confidence Bound (UCB) bandit instance for PR and sketched the centralized patrolling algorithm for PRs based on the corresponding bandit instances in Algorithm 9. A UCB value can be interpreted as the upper bound of a confidence interval, therefore, the true average reward of each route i is below this upper confidence bound with high probability. If a PR have tried a route less often, the estimated reward is less accurate so the confidence interval is large opposed to a route which has been routed more.

UCB1 gives a reasonable balance for each PR between exploiting a rewarding route vs exploring a less patrolled one. Suppose, in case of exploration, for a PR there is only one route to select from two: i, j and from both of these routes, the PR achieved same average reward (x_i and x_j are the same in the calculation of UCB value) after a number of rounds are played. Now, if a route i has been tried more often than j, then $f_i > f_j$ with same f as a numerator. Then $\sqrt{\frac{2\log(f)}{f_i}} < \sqrt{\frac{2\log(f)}{f_j}}$. Therefore according to UCB (Upper Confidence Bound) the confidence bound shrinks for i more than j. Again, this bound grows if the f gets higher for the exploration.

As for the exploitation, the UCB value is updated with adjustment of average reward x_i for the route i as that route gets selected. Normalized UCB value, which is used as coverage probability for SSG in our approach reflects this exploitation-exploration trade-off for a PR bandit. However, as we are using Graph Based UCB1 instances, routes are represented as nodes. Routes found to be attacked and get defended by PRs at a round of patrolling are connected with edges according to our graph based approach. However,

unlike recommendation systems, in SSG game the feedback is coming from opponents who want to foil the strategy taken by the bandits. Therefore we enhanced our algorithm to handle the presence of adversary. We relaxed the node-weight sharing mechanism between neighboring nodes and gave each node equal reward like IBA. This is because we assume that the attackers observe all the deployments history of defender PR bandits before selecting an attack strategy. Node-weight imposes essential bias towards some routes over the others which facilitates the selection of a representative node earlier. However, in the presence of adversaries, this method suffers from the weakness of compromising the next defense strategy from the part of the PRs, exposing the most promising route to be selected next by a PR to a best response opponent. To avoid the danger of getting the next action from the PR predicted by the opponents, bandits needs to randomize its strategy and fool the attackers and entice them to attack a seemingly less defended route to capture them.

4.6.1 Active and Dormant Routes

In Chapter 3, we introduced adaptive annotation of graph bandit nodes as active and dormant. There we mainly showed how we modify the selection policy for graph based bandits and reduce the search space by finding active arms faster. In this section, we discuss how we use it to annotating the routes to be active and dormant and incorporate new selection policy of graph bandits proposed in Chapter 3 to find the solution to DPAP which can result in a better coverage under adversarial setting.

As described in DPAP, each route is considered as an arm in our bandit algorithm and hence a node in our graph. On a certain round, a route can be in one of the following two states:

Definition 8 *A route is **active** for a time step if it is currently selected by our bandit algorithm. Such an arm as route (node) have most outgoing edges in our graph.*

At the beginning, all routes are active because each of them are equally likely to be selected.

Definition 9 *A route is **dormant** if it is selected by the bandit but found one of k routes which for which an active route is identified by another bandit at the same time.*

These arms are dormant in a sense that they equally qualify to represent a specific attack pattern.

Routes selected by bandits with no match in the attack pattern receive a reward of 0 but stays active. Because even though those arms clearly indicates its failure to cover that specific attack at certain iteration, our algorithm does not rule out its potential for covering potential incoming attacks from the attackers.

4.6.2 Transitions of Routes

Algorithm 6 in Chapter 3 shows a complete Graph Based UCB1 update policy for bandits with this annotation of Arms. With that arms annotation, we define transition of a route as arm from one state to the other occurs in the following situations:

- A route gradually becomes dormant from active with its repeated failure to be the first route to match with the attack pattern

- A route gradually becomes active from dormant if it frequently matches with the first route from the incoming attack patterns

Transition of an arm from the dormant state to an active state potentially results in covering more attack patterns through randomization as dormant arm can become active with our adaptive selection policy discussed in Chapter 3. The relationship between α and the counter r is the same as described in Chapter 3. While selecting an arm for each iteration, every bandit needs to update the value of r. If an arm is found to be in dormant state, UCB value for that arm is smoothed by the parameter θ in following manner:

$$\theta = 1 + \log(1 + r_n) \tag{4.5}$$

Therefore, UCB value for every dormant arm i for a bandit is:

$$x_i + (1 - \exp^{-1/\theta})\sqrt{\frac{2\log(f)}{f_i}} \tag{4.6}$$

This gives our algorithm enough opportunity to balance between exploration and exploitation in a adaptive manner instead of using a parameter set to a fixed value to scale it. Instead of trying with different parameters, by increasing the value of r for a previously active arm which transformed as dormant in current iteration, we scale down the UCB value of the arms selected by the subsequent bandits by $(1 - \exp^{-1/\theta})$. So that for such a route, less exploration will take place and it gets biased towards exploring that route to give it a fair chance of getting selected without being exposed to the attacker. This gives an implicit preference of a defender to certain routes to withstand attackers but with the

action history of that corresponding bandit, it is hard for the attacker to follow this inherent preference as this arm can get reset to active in an adaptive manner in case it was not found in certain incoming attack pattern.

4.6.3 Randomized Algorithm

The randomized algorithm we propose as a solution to DPAP is based on the updated selection policy we presented in the previous section on graph nodes. As we reconsider the environment with SSG unlike our solution for the recommendation system, each node has its independent reward for each episode without sharing with the correlated arms and average reward is adjusted accordingly keeping the edge relationship among them consistent along with the state annotation. And according to our proposed adaptive simulated annealing policy, for node selection, our graph based algorithm controls the transition of the arms' states and helps PRs select the ones which are active and rewarding. This internally helps the PR bandits to keep track of the rewarding routes while making the selection apparently randomized to the attackers.

We compare this mechanism with several other randomized bandit approaches; namely, uniform-random, ϵ-greedy and α-exploitation.

- With **uniform random** approach, we select a random route for a PR to patrol
- In ϵ**-greedy** approach, with a very small fixed $epsilon$ probability our algorithm assigns a random route to a PR, otherwise it selects the best UCB valued route

- According to α**-exploitation** method, weight the average reward of a route with a small fixed value of α for exploitation; whereas, the exploration is adjusted against this correction with a scaling factor of $(1 - \alpha)$

4.6.4 Simulation Results

In this section, we simulate the algorithm we proposed in Algorithm 9. We conducted several sets of experiments on the synthetic data to show the result of the game we defined. In all our experiments, we considered certain amount of overlap between the strategies of different attackers so that the problem is suitable to model using graph. In reality, most attackers attacking different routes may show this partial overlap in their strategies.

We allowed a 25% overlap between different attack-strategies. Our generated dataset has 7 different attack strategies, each consisting of 25 different attack routes targeted simultaneously by attackers on total 100 routes. A successful attack or defense gives a unit reward according to the game matrix shown in the figure **??** and negative reward otherwise. To view it as a zero sum game we keep 25 defenders playing as a bandits against the attackers.

It is observed that average reward of defenders outperforms that of the attackers over the rounds. After 1000 rounds almost all 25 defenders sent out to defend 25 routes could patrol 25 best response routes selected by the attackers from 7 different overlapping strategies. The average reward of the players are plotted in a graph shown in figure 4.1 which shows the increasing trend in the accumulated reward of defenders beating the attackers.

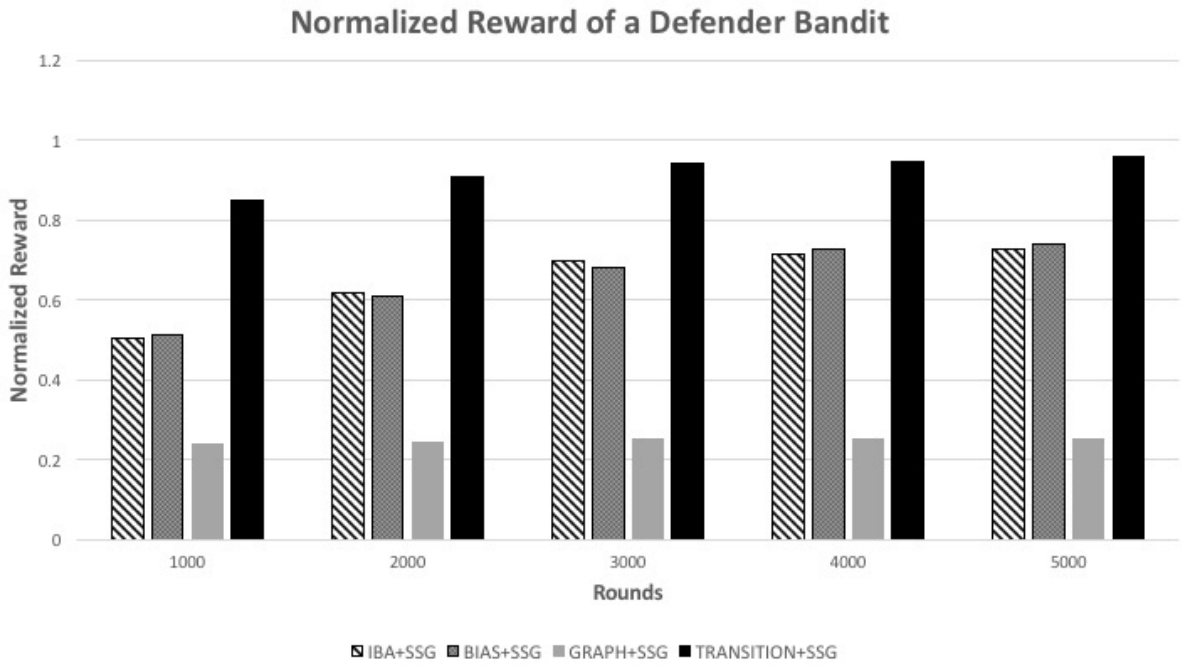


Fig. 4.3.: Comparison of different learning method of 25 defenders for 100 routes against 25 attackers

10 RUNS AVERAGE REWARD

PLAYERS	25 Defender Bandits				25 attackers with Best Response			
ROUNDS	**Average**	**StDev**	**Max**	**Min**	**Average**	**StDev**	**Max**	**Min**
100	6.52	0.18	6.75	6.13	18.39	0.22	**18.86**	18.12
200	13.91	0.89	14.54	12.20	12.96	1.08	14.27	10.56
300	16.35	1.74	17.62	12.12	8.88	1.54	11.28	7.54
400	16.45	1.64	18.94	14.59	7.68	1.67	10.64	5.84
500	18.57	1.66	20.07	14.59	7.33	2.17	11.07	4.93
600	19.36	1.75	20.67	16.16	5.32	1.68	8.80	4.13
700	19.07	2.35	21.37	16.10	5.91	1.89	4.59	3.58
800	19.63	2.27	21.78	15.98	3.82	0.43	4.59	3.40
900	19.89	2.31	21.84	15.94	5.48	2.05	8.48	3.05
1000	**20.33**	2.03	**22.33**	16.99	**3.68**	0.55	4.93	2.74

Fig. 4.4.: Reward statistics of 25 defenders is shown against best response attackers attacking 25 routes at the same time

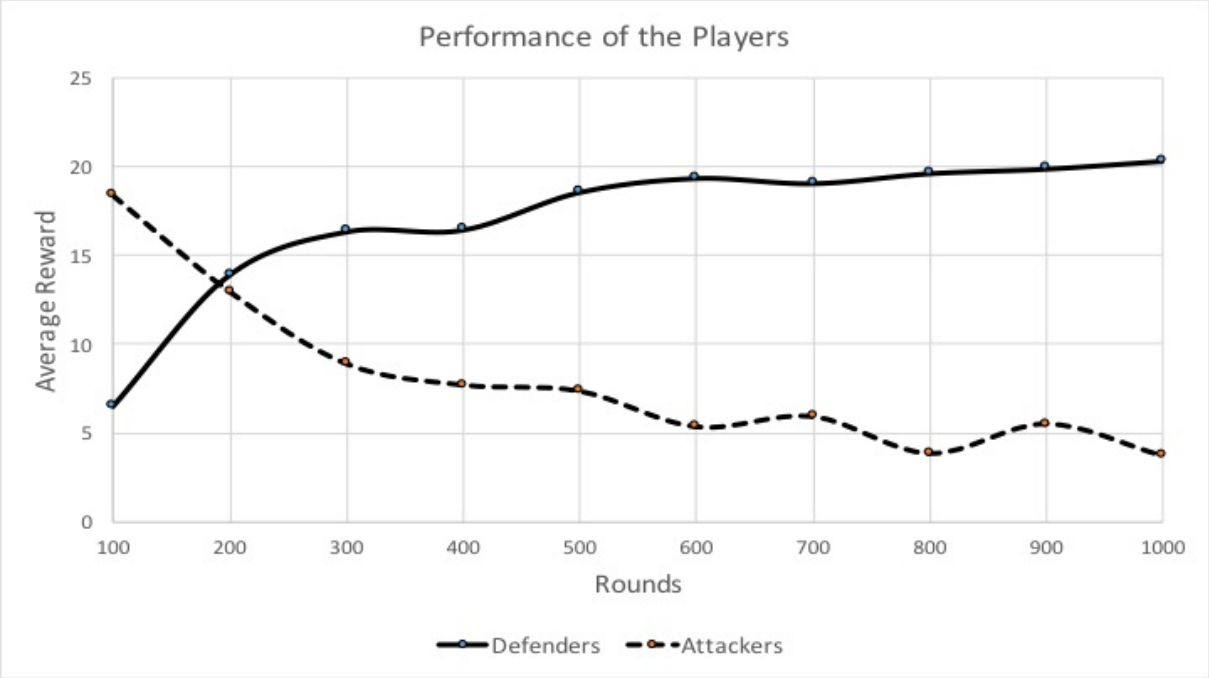


Fig. 4.5.: Total reward of 25 defenders is shown against best response attackers attacking 25 routes at the same time

In the later experiment, figure 4.3 shows the normalized reward each PR is getting being a member of the defending team. The histogram plotted that value for averaged over every 1000 iteration up to 5000 rounds. We show how different strategy taken by the defending team changes the reward dynamics. We run our experiment with IBA based SSG, Reward Bias based SSG, Node weight based graph bandits with and without state transition. We observed that Graph based PR bandit instances playing SSG as a defender with state transition against attackers are receiving the highest amount of reward. Graph with no state transition is worst performing one as it exposes the rewarding pattern to the attackers through node weight adjustment. This gives ample scope to the best response attackers to change their attack-strategy successfully against the defenders.

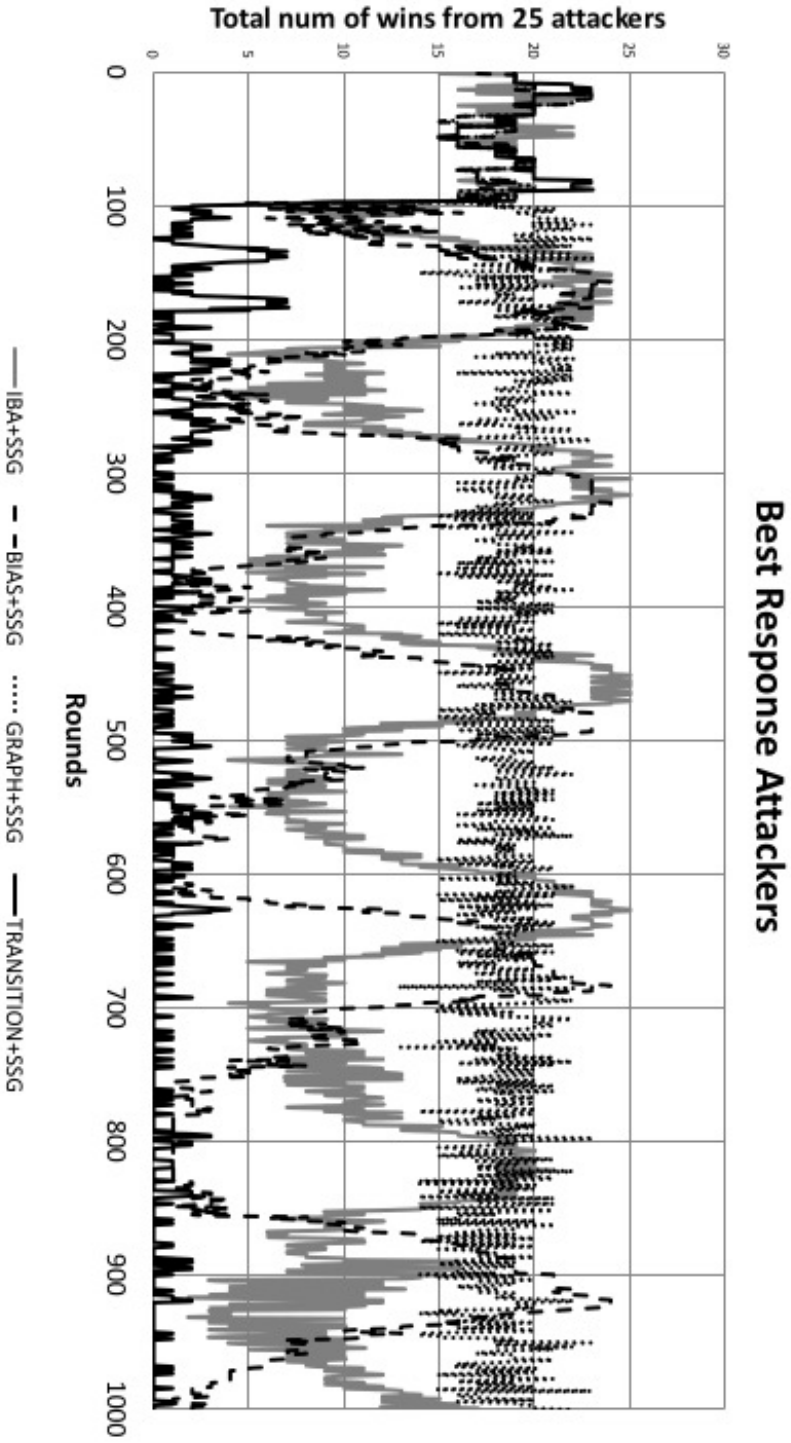


Fig. 4.6.: Total number of successful attacks by 25 attackers over 100 routes against 25 defenders

In figure 4.6, we visualize the point of view of the best response attackers for the same experiment setting. Total reward earned from 25 routes attacked in each episode of SSG game has been plotted up to 1000 episodes where they are playing against the same 4 types of defenders' strategies mentioned in the previous experiment. As it can be observed from this figure that playing against the graph based bandits which has state transition over rewarding routes is hardest to succeed against from the part of the attacking team. It is found that after 100 episodes the team reward of attackers falls sharply down and stays almost zero till our simulation ended till 1000 rounds. Here team reward of attackers can range from 25 to 0; as there are 25 attacking routes in a round and each attacker can contribute a unit reward to the team with a successful attack.

We further compare the performance of our state transition based graph bandit as the members of the defending team against other randomized strategies. Figure 4.7 shows the outcome of the game played under same enviroment using bandits who randomize their decision of selecting routes using graph based bandits but used uniform-random, ϵ-greedy and α-exploitation instead of our proposed state transition mechanism.

In the subsequent experiments shown in 4.8, the number of defending bandits are increased and reduced to double and half respectively to observe its impact on team rewards obtained by the attackers keeping the number of routes to be defended same. When only 12 defender PRs are deployed to protect 25 routes to be attacked over 100 routes, the best defenders could do by using our approach is to protect 12 routes assigned to them and hence reduce the total team reward of the attackers from 25 to 13 (saving 12 units by capturing attackers on these 12 routes). When the defenders are more than the

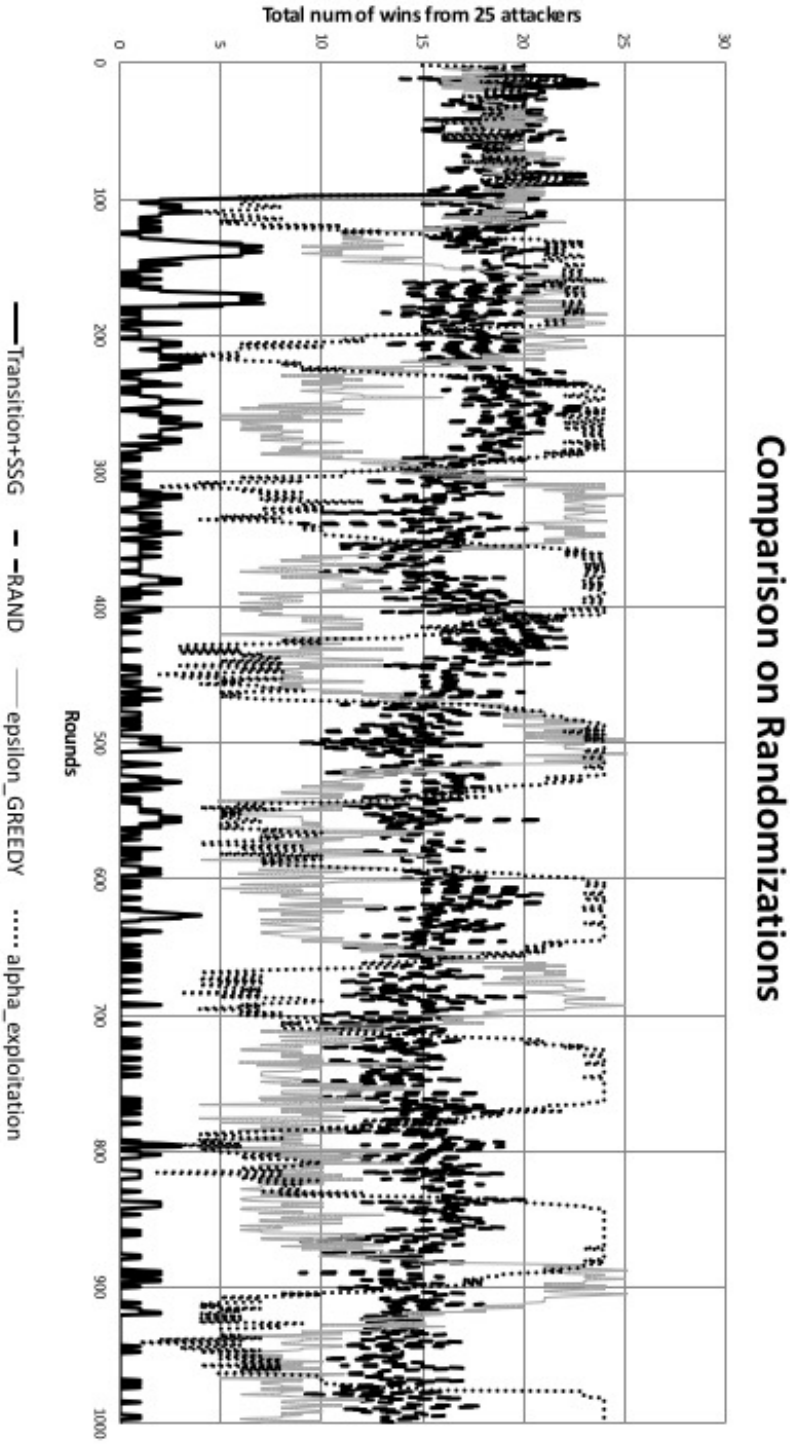


Fig. 4.7.: Total number of successful attacks by 25 attackers compared under different randomized defender deployment scenario

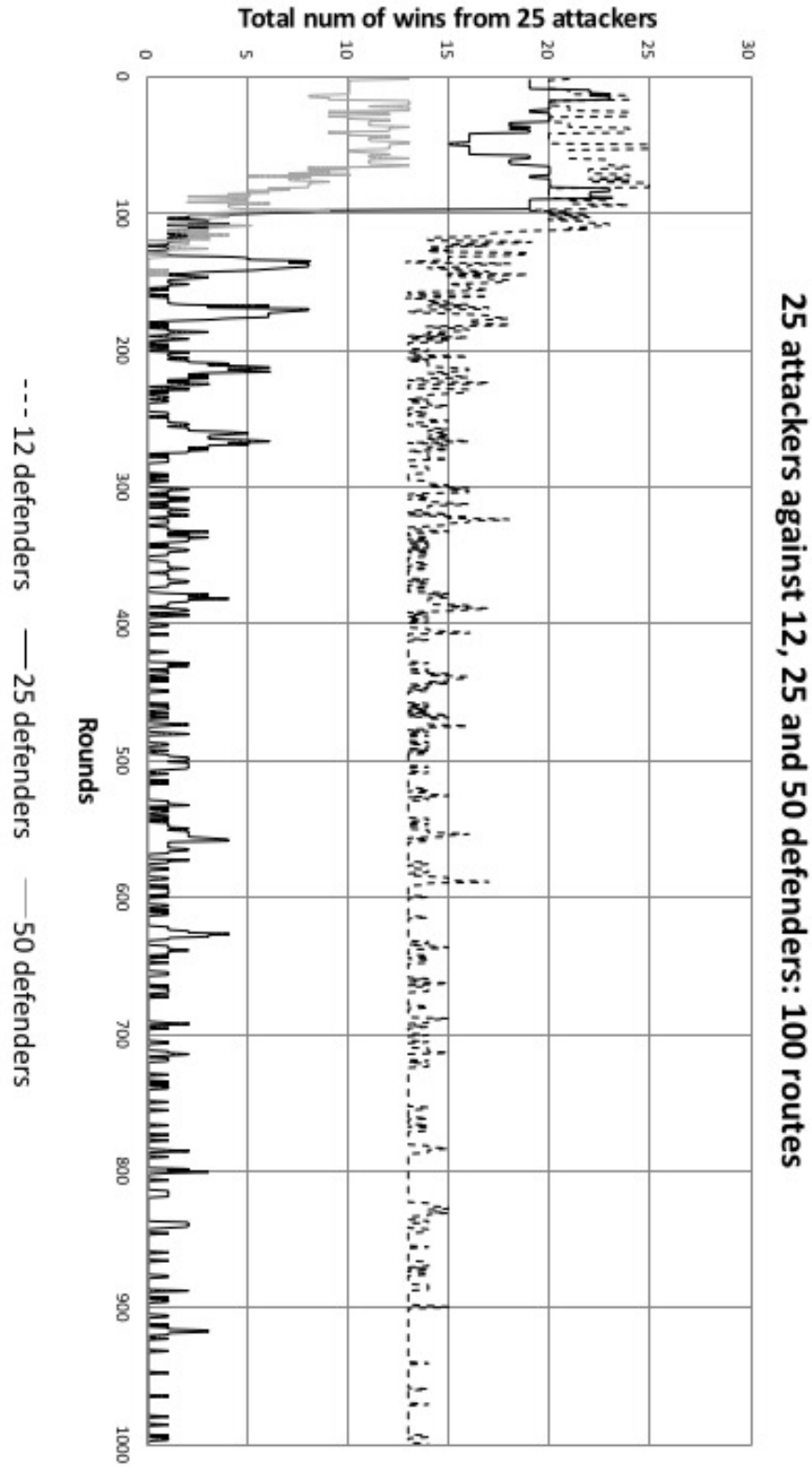


Fig. 4.8.: Total number of successful attacks by 25 attackers when various number of transitional defenders are deployed against them

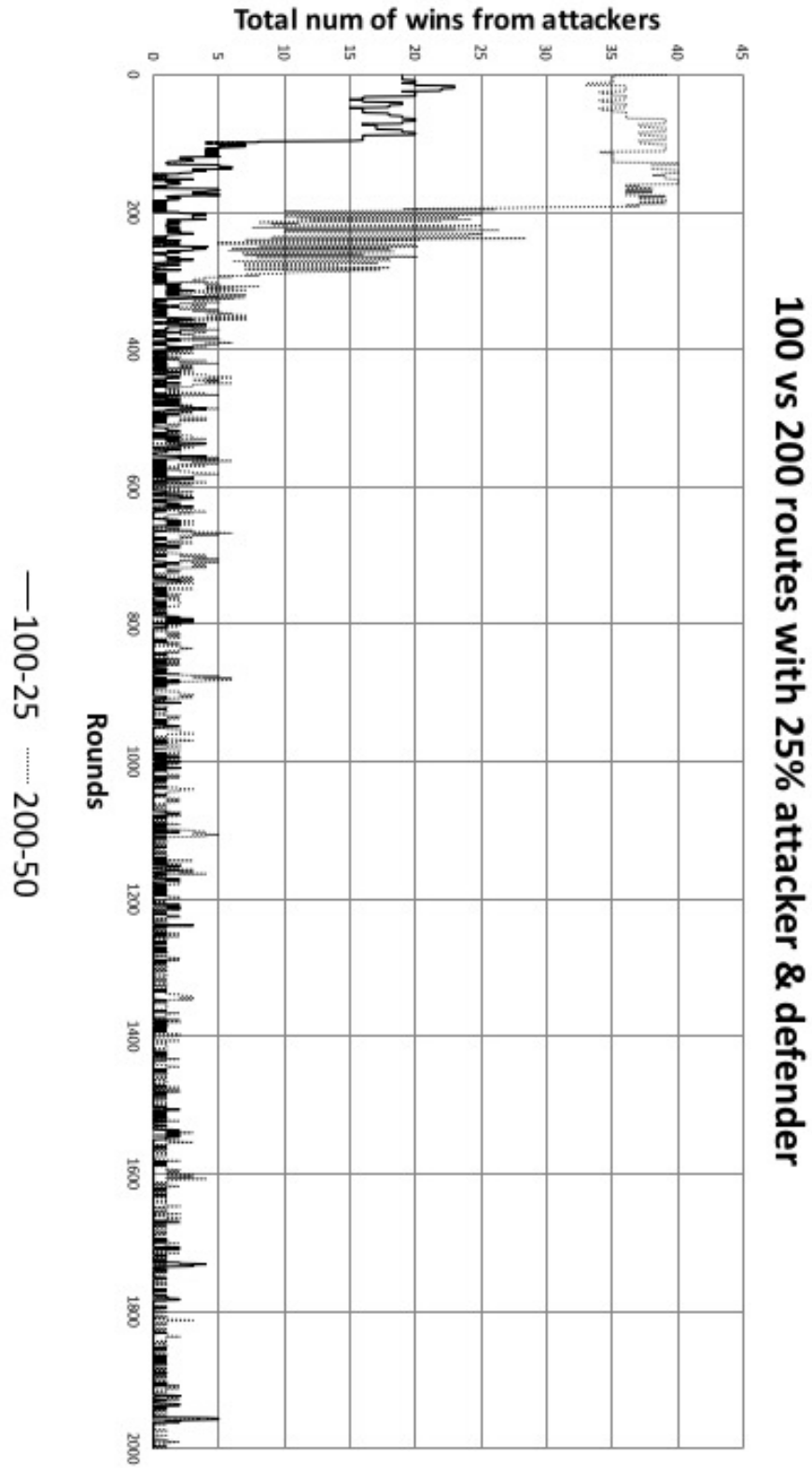


Fig. 4.9.: Total number of successful attacks by 25 attackers when various number of routes are considered

attack instances, it is obvious that attackers team reward diminishes very fast and converges to zero as seen in this figure.

we also change the configuration of the problem environment, where the parameter for the number of routes is changed to 200 from 100 and attack instances and PR bandits as defenders are doubled as well. It is found that the problem gets harder even with scaling up all the parameters with same ratio as we can see in figure 4.9 that convergence took almost 2000 rounds of play instead of 1000.

In this chapter we introduced an online learning algorithm for patrolling robots against adversarial attackers. Our centralized algorithm schedules the route to patrol for each robot in every round by treating the PRs as graph based UCB1 bandits with state transition. We show the effectiveness of our Graph based state transition oriented algorithm to deal with different types of attackers attacking several routes at the same round.

5. SUMMARY AND FUTURE DIRECTIONS

Online Learning problem has also been studied as a Multi-Armed Bandit (MAB) Problem in [47] as Restless Bandit, which is also based on Markov chain like RBA. Mortal Bandit problem [15] is a close variant of MAB where each arm is assigned a lifetime, the expiration of it calls in for a new arm to take over. Thus the total number of arms stays the same all the time. However, the solution is based on some predefined parameters, which need to be set by empirical evaluation. Volatile bandit [11] extends the mortal bandit with an update to exploration and exploitation trade-off. In this Chapter, we summarize our contribution to coverage problem and discuss possible extensions to our work based on some of the above-mentioned works.

In the following sections, we discuss some of our contributions in this dissertation:

5.1 Capturing Diversity in Online Learning

We study a recommendation system problem, in which the system must be able to cover as many users' preferences as possible while these preferences change over time. This problem can be formulated as a variation of the maximum coverage problem, specifically, we defined a novel problem of Online k-Hitting Set, where the number of sets and elements within the sets can change dynamically. When the number of distinctive elements is large, an exhaustive search for even a fixed number of elements is known to be

computationally expensive. The problem is known to be NP-hard, and many known algorithms tend to have exponential growth in complexity.

5.1.1 Introducing Graph Based Bandits

Unlike personalized recommendation systems [45] which often use collaborative filtering [18, 19], content-based filtering [37] or a hybrid of these two [13], we develop a recommendation system that satisfies a diverse number of user types. Some existing works argued that dependency among items can be ruled out by ignoring the correlation gap [4, 28] . However, we show that the correlation between items is an important criterion to identify diversity regarding user types. We proposed an effective graph based bandit rewarding mechanism, which aimed to incorporate this diversity and our empirical evaluation show that it outperformed existing techniques for real data sets regarding covering a large number of user types introducing no additional complexity.

Our novel graph-based UCB1 algorithm can effectively minimize the number of elements to consider, thereby reducing the search space greatly. The algorithm utilizes a new rewarding scheme to choose items that satisfy more users by balancing coverage and diversity as it constructs a relational graph between items to choose. Experiments show that the new algorithm performs better than existing techniques such as Ranked Bandit[40] and Independent Bandits [28] regarding satisfying diverse types of users while minimizing computational complexity.

The contributions of this work minimizing the user abandonment and maximizing the coverage of diversity are following:

- formally defining the novel problem of Online k-Hitting Set and give an anytime approximation algorithm for solving it for the application of recommendation system.

- developing a graph based "update policy" and "selection policy" for UCB1 bandits for recommendation systems where the system learns the dependency structure among items though a novel rewarding scheme and select the most promising items.

- verifying the efficacy of our method for recommending a small number of items from real data sets where there are hundreds of users, each having thousands of choices to pick from.

5.1.2 Parallel and Synchronized Execution of Bandits

We adopted and modified UCB2 [5], a bandit algorithm which shows the same set of recommendation to different users for a certain amount of iterations before it takes account of the users' feedbacks. Even though UCB2 is computationally expensive for online learning, but we applied multiple parallel instances of it and gained competitive result compared to IBA and RBA. However, it is difficult to apply multiple instances of UCB2 for a practical purpose like constructing an online recommending a set of items, as each UCB2 instance assigned to select an item for the recommendation set has its local parameters, and there is no existing coordinating method to adjust them effectively across different bandit instances for parallel execution.

Although UCB2 is theoretically proved to have a better regret bound than UCB1, unlike UCB1, for the reasons mentioned above, it has not been used for parallel execution.

We designed an efficient algorithm which runs multiple instances of UCB2 in parallel. Our algorithm suitably handles parameter synchronization, reward update and exploration decisions across multiple instances of UCB2 and ensures that they are capable of covering different types of users. In our work [41] we introduce the synchronization technique for epoch for arms selected by multiple instances of UCB2, so that parallel execution of UCB2 bandits is possible. We also update the synchronization parameter as a function of time to handle real datasets.

We present an effective method to use the multi-armed bandit algorithm UCB2 as an online approximation learning algorithm for recommendation systems. The work of online content optimization found in [2] and decision on exploration and exploitation trade-off [3] are some other different approaches suggested in the work of others. However, both these approaches are somewhat dependent on associated offline models. As our target to cover a variety of users with our recommendation set rather than developing a personalized recommendation system, our problem is similar to maximum coverage problem as mentioned in [42]. [30] also proposed an epoch-greedy algorithm using contextual bandits based on sound theoretical guarantee but that requires observable side information. The main contribution of our work is manipulating the parameters of UCB2 and allow it for parallel and synchronized execution to build an online recommendation system. Our findings show tuning this parameters has a critical role in determining the nature of recommendation to users as the search gets directed by the users relevance feedback over time. The proposed mechanism has been tested on some real datasets, and we believe that the results are promising enough to extend this work to

capture more dynamics of user preferences. Our contribution in leveraging UCB2 with in this regard is following:

- We systematically dissect UCB2 algorithm and study its parameters to understand their impacts. As α (fixed external parameter in UCB2) is responsible for the trade-off between exploration and exploitation in UCB2, we applied scheduling mechanism similar to simulated annealing [34] to update α. We experimented different datasets with some scheduling functions to show the impact of perturbation of α is on the performance of our method.

- We devised a technique to set a suitable length of the observation period (termed as "epoch") before updating the UCB2 bandits and update this epoch synchronously across all bandits. It is crucial for the parallel execution of UCB2 as for any learning algorithm, this "exploitation period" has to be updated according to the learning rate (i.e., α for UCB2).

- We developed a coordination algorithm with a novel rewarding scheme which facilitates user coverage by prioritizing one bandit over the others and force them to avoid selecting the same element more than once for a single recommendation set to ensure more coverage of users.

5.2 Online Learning with Adversarial Game

With an aim to facilitate our proposed mechanism to be used to solve a critical decision-making problem online, we further study route assignment to patrolling robots for crime-prone areas of a neighborhood where we can only deploy a fixed number of

autonomous robots; with this resource constraint, it is necessary to maximize the impact of their routing assignments. We develop an online model to approximate the optimal solution under some attack probability distributions. Our algorithm can direct a small number of patrolling robots to a few routes out of many available routes at a certain time under a given adversarial environment. Our simulation results show that using UCB1 bandits to determine the strategy for these Patrolling Robots (PR), we can achieve considerable coverage of routes by playing Stackelberg Game against different types of attacks. In this area our contributions are following:

- we formulate a zero-sum repeated Stackelberg Security Game between PRs and the attacking team with a utility function that facilitates the online learning of PRs from the attack dynamics
- we develop a centralized algorithm to compute the online route selection strategy for PRs based on Multi-Armed Bandit algorithm where exploration and exploitation are balanced
- we simulate the performance of our algorithm under different synthetic attack environments to show its efficacy

On average, our proposed mechanism outperforms existing recommendation techniques regarding covering sensitive routes while keeping the computational complexity low.

5.3 Future Directions

In the following sections, we discuss some potential extensions of our work.

5.3.1 Distributed Graph Bandits

With a large number of arms, any bandit algorithm would take a long time to find the best arm to play. This is also true for our graph-based bandits where the UCB1 value needs to be computed for every node in the graph to select the best rewarding ones. Moreover, higher the number of arms, longer it takes to update this value for every arm for the next round. Therefore we want to distribute the arms across bandits so that each bandit needs to compute the UCB1 values for a nonoverlapping subset of arms specifically assigned to it. With k graph based bandit to select k unique nodes from a total of N nodes, we create k graphs where each bandit operates on a graph of N/k nodes. These k graph are mutually disconnected, and each bandit operates on a specific graph. In future, we want to extend this solution to a distributed recommendation system to facilitate a scalable and decentralized decision making for Big Data.

5.3.2 Dynamic Addition of Arms

Another issue of bandit algorithms is that the number of arms is fixed for a bandit. However, in the real-world problem such as building a diverse recommendation set from user data, more items can be added to the existing set of items as new users can stream in with preference to new items (i.e., newly released movie). This requires a dynamic addition of nodes to the graphs for our bandits. As each bandit node represents an item, newly appeared nodes need to be assigned to new bandit nodes. This assignment should be done in such a way that these new nodes get a fair amount of exploration-exploitation trade off on their bootstrap.

5.3.3 Learning against Complex Heterogeneous Attacks

Currently, our algorithm only handles one strategy of attack as the best response at each round, but in most practical situations, different types of attack strategies can be mixed, and attackers can attack a variety of routes to incur damages according to their capacities. In such cases, a route can face damages posed by the varying intensity of the attackers. Therefore, the team of PRs needs to learn the complex nature of heterogeneous attacks within each round of play.

5.3.4 Attackers Learning against Defenders

An attacker can only maximize its utility by performing a successful attack (avoiding a PR). We are interested to see how the learning mechanism of bandits gets affected by the attacking team if attackers also employ learning against the PRs. For future work, we intend to extend the SSG using MAB settings for both the teams where each of them can select its routes by online implicit opponent modeling. [8].

LIST OF REFERENCES

LIST OF REFERENCES

[1] Optimism in the face of uncertainty: the ucb1 algorithm. http://jeremykun.com/2013/10/28/optimism-in-the-face-of-uncertainty-the-ucb1-algorithm/. Accessed: 2013-10-28.

[2] D. Agarwal, B. C. Chen, P. Elango, N. Motgi, S. T. Park, R. Ramakrishnan, S. Roy, and J. Zachariah. Online Models for Content Optimization. In D. Koller, D. Schuurmans, Y. Bengio, L. Bottou, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *NIPS*, pages 17–24. MIT Press, 2008. URL http://dblp.uni-trier.de/rec/bibtex/conf/nips/AgarwalCEMPRRZ08.

[3] D. Agarwal, B.-C. Chen, and P. Elango. Explore/exploit schemes for web content optimization. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3895-2. doi: 10.1109/ICDM.2009.52. URL http://dx.doi.org/10.1109/ICDM.2009.52.

[4] S. Agrawal. *Optimization under uncertainty: Bounding the correlation gap*. PhD thesis, March 2011. URL http://research.microsoft.com/apps/pubs/default.aspx?id=200425.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352. URL http://dx.doi.org/10.1023/A:1013689704352.

[6] N. Azizi and S. Zolfaghari. Adaptive temperature control for simulated annealing: a comparative study. *Computers Operations Research*, 31(14):2439 – 2451, 2004. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/S0305-0548(03)00197-7. URL http://www.sciencedirect.com/science/article/pii/S0305054803001977.

[7] M.-F. Balcan, A. Blum, N. Haghtalab, and A. D. Procaccia. Commitment without regrets: Online learning in stackelberg security games. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15, pages 61–78, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3410-5. doi: 10.1145/2764468.2764478. URL http://doi.acm.org/10.1145/2764468.2764478.

[8] N. Bard, M. Johanson, N. Burch, and M. Bowling. Online implicit agent modelling. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 255–262, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-1993-5. URL http://dl.acm.org/citation.cfm?id=2484920.2484963.

[9] J. C. Beck, P. Prosser, and E. Selensky. Vehicle routing and job shop scheduling: What's the difference? In *Proceedings of the Thirteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'03, pages 267–276. AAAI Press, 2003. ISBN 1-57735-187-8. URL http://dl.acm.org/citation.cfm?id=3036969.3037004.

[10] R. Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957. ISSN 0022-2518.

[11] Z. Bnaya, R. Puzis, R. Stern, and A. Felner. Volatile multi-armed bandits for guaranteed targeted social crawling. In *Late-Breaking Developments in the Field of Artificial Intelligence, Bellevue, Washington, USA, July 14-18, 2013*, 2013. URL http://www.aaai.org/ocs/index.php/WS/AAAIW13/paper/view/7030.

[12] G. Brown, M. Carlyle, J. Salmerón, and K. Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, Nov. 2006. ISSN 0092-2102. doi: 10.1287/inte.1060.0252. URL http://dx.doi.org/10.1287/inte.1060.0252.

[13] R. Burke. The adaptive web. chapter Hybrid Web Recommender Systems, pages 377–408. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72078-2. URL http://dl.acm.org/citation.cfm?id=1768197.1768211.

[14] P. Castells, S. Vargas, and J. Wang. Novelty and Diversity Metrics for Recommender Systems: Choice, Discovery and Relevance. In *International Workshop on Diversity in Document Retrieval (DDR 2011) at the 33rd European Conference on Information Retrieval (ECIR 2011)*, Apr. 2011.

[15] D. Chakrabarti, R. Kumar, F. Radlinski, and E. Upfal. Mortal multi-armed bandits. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 273–280, 2008. URL http://papers.nips.cc/paper/3580-mortal-multi-armed-bandits.

[16] K. Chandrasekaran, R. Karp, E. Moreno-Centeno, and S. Vempala. Algorithms for implicit hitting set problems. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 614–629, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics. URL http://dl.acm.org/citation.cfm?id=2133036.2133084.

[17] P. Dagum, A. Galper, and E. Horvitz. Dynamic network models for forecasting. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, UAI '92, pages 41–48, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. ISBN 1-55860-258-5. URL http://dl.acm.org/citation.cfm?id=143802.143815.

[18] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan. Collaborative filtering recommender systems. *Found. Trends Hum.-Comput. Interact.*, 4(2):81–173, Feb. 2011. ISSN 1551-3955. doi: 10.1561/1100000009. URL http://dx.doi.org/10.1561/1100000009.

[19] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, July 2001. ISSN 1386-4564. doi: 10.1023/A:1011419012209. URL http://dx.doi.org/10.1023/A:1011419012209.

[20] M. R. Gupta and Y. Chen. Theory and use of the em algorithm. *Found. Trends Signal Process.*, 4(3):223–296, Mar. 2011. ISSN 1932-8346. doi: 10.1561/2000000034. URL http://dx.doi.org/10.1561/2000000034.

[21] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. on Interactactive Intelligence System*, 5(4):19:1–19:19, Dec. 2015. ISSN 2160-6455.

[22] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL)*, 45(6):615–627, 1998. ISSN 1520-6750.

[23] J. Hudack and J. C. Oh. Multi-agent sensor data collection with attrition risk. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pages 166–174, 2016. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/12994.

[24] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: 10.1145/775047.775067. URL http://doi.acm.org/10.1145/775047.775067.

[25] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, May 1998. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X. URL http://dx.doi.org/10.1016/S0004-3702(98)00023-X.

[26] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-45375-X, 978-3-540-45375-8. doi: 10.1007/11871842_29. URL http://dx.doi.org/10.1007/11871842_29.

[27] L. Kocsis and C. Szepesvri. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[28] P. Kohli, M. Salek, and G. Stoddard. A fast bandit algorithm for recommendation to users with heterogenous tastes. In M. desJardins and M. L. Littman, editors, *AAAI*. AAAI Press, 2013. ISBN 978-1-57735-615-8. URL http://dblp.uni-trier.de/db/conf/aaai/aaai2013.html#KohliSS13.

[29] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[30] J. Langford and T. Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 817–824, 2007. URL http://papers.nips.cc/paper/3178-the-epoch-greedy-algorithm-for-multi-armed-bandits-with-side-

[31] J. Marecki, G. Tesauro, and R. Segal. Playing repeated stackelberg games with unknown opponents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages

821–828, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-9817381-2-5, 978-0-9817381-2-3. URL http://dl.acm.org/citation.cfm?id=2343776.2343814.

[32] MOVIELENS-DATA. *MovieLens dataset, http://www.grouplens.org/data/*, as of 2003. URL http://www.grouplens.org/data/.

[33] G. D. Nittis and F. Trovò. Machine learning techniques for stackelberg security games: a survey. *CoRR*, abs/1609.09341, 2016. URL http://arxiv.org/abs/1609.09341.

[34] Y. Nourani and B. Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998. URL http://stacks.iop.org/0305-4470/31/i=41/a=011.

[35] W. Park, J. C. Oh, M. K. Blowers, and M. B. Wolf. An open-set speaker identification system using genetic learning classifier system. In M. Cattolico, editor, *GECCO*, pages 1597–1598. ACM, 2006. ISBN 1-59593-186-4. URL http://dblp.uni-trier.de/db/conf/gecco/gecco2006.html#ParkOBW06.

[36] P. Paruchuri, M. Tambe, F. Ordóñez, and S. Kraus. Security in multiagent systems by policy randomization. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 273–280, New York, NY, USA, 2006. ACM. ISBN 1-59593-303-4. doi: 10.1145/1160633.1160681. URL http://doi.acm.org/10.1145/1160633.1160681.

[37] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *THE ADAPTIVE WEB: METHODS AND STRATEGIES OF WEB PERSONALIZATION. VOLUME 4321 OF LECTURE NOTES IN COMPUTER SCIENCE*, pages 325–341. Springer-Verlag, 2007.

[38] J. M. Phillips. Chernoff-hoeffding inequality and applications. *CoRR*, abs/1209.6396, 2012. URL http://dblp.uni-trier.de/db/journals/corr/corr1209.html#abs-1209-6396.

[39] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 125–132. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[40] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 784–791, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390255. URL http://doi.acm.org/10.1145/1390156.1390255.

[41] M. Rahman and J. C. Oh. Parallel and synchronized UCB2 for online recommendation systems. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2015, Singapore, December 6-9, 2015 - Volume I*, pages 413–416, 2015. doi: 10.1109/WI-IAT.2015.164. URL http://dx.doi.org/10.1109/WI-IAT.2015.164.

[42] M. Rahman and J. C. Oh. Fast online learning to recommend a diverse set from big data. In *The 28th International Conference on Industrial, Engineering and Other Application of Applied Intelligent Systems*, IEA/AIE'15, pages 361–370, Switzerland, 2015. Springer.

[43] C. Rego, D. Gamboa, F. Glover, and C. Osterman. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427 – 441, 2011. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2010.09.010. URL http://www.sciencedirect.com/science/article/pii/S0377221710006065.

[44] S. E. Robertson. Readings in information retrieval. chapter The Probability Ranking Principle in IR, pages 281–286. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN 1-55860-454-5. URL http://dl.acm.org/citation.cfm?id=275537.275701.

[45] G. Shani and A. Gunawardana. Evaluating recommendation systems. *Recommender Systems Handbook*, pages 257–297, 2011. URL http://scholar.google.de/scholar.bib?q=info:AW2lmZl44hMJ:scholar.google.com/&output=citation&hl=de&as_sdt=0,5&ct=citation&cd=0.

[46] M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41 – 43, 2004. ISSN 0167-6377. doi: http://dx.doi.org/10.1016/S0167-6377(03)00062-2. URL http://www.sciencedirect.com/science/article/pii/S0167637703000622.

[47] C. Tekin and M. Liu. Online learning of rested and restless bandits. *IEEE Trans. Information Theory*, 58(8):5588–5611, 2012. doi: 10.1109/TIT.2012.2198613. URL http://dx.doi.org/10.1109/TIT.2012.2198613.

[48] J. Tsai, C. Kiekintveld, F. Ordonez, M. Tambe, and S. Rathi. Iris-a tool for strategic security allocation in transportation networks. 2009.

[49] S. A. Vinterbo and A. hrn. Minimal approximate hitting sets and rule templates. *International Journal of Approximate Reasoning*, 25:123–143, 2000. doi: 10.1016/S0888-613X(00)00051-7.

[50] H. von Stackelberg. *Market Structure and Equilibrium*. Springer, 1st edition. edition, Dec. 2010. ISBN 3642125859. URL http://www.worldcat.org/isbn/3642125859.

[51] Z. Xing and J. C. Oh. *Heuristics on the Data-Collecting Robot Problem with Immediate Rewards*, pages 131–148. Springer International Publishing, Cham, 2016. ISBN 978-3-319-44832-9. doi: 10.1007/978-3-319-44832-9_8. URL http://dx.doi.org/10.1007/978-3-319-44832-9_8.

[52] H. Xu, L. Tran-Thanh, and N. R. Jennings. Playing repeated security games with no prior knowledge. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, pages 104–112, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-4239-1. URL http://dl.acm.org/citation.cfm?id=2936924.2936944.

[53] R. Yang, F. Ordonez, and M. Tambe. Computing optimal strategy against quantal response in security games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 847–854, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-9817381-2-5, 978-0-9817381-2-3. URL http://dl.acm.org/citation.cfm?id=2343776.2343818.

[54] Z. Yin, A. X. Jiang, M. P. Johnson, C. Kiekintveld, K. Leyton-Brown, T. Sandholm, M. Tambe, and J. P. Sullivan. Trusts: Scheduling randomized patrols for fare inspection in transit systems. In *IAAI*, 2012.

[55] C. Zhang, A. Sinha, and M. Tambe. Keeping pace with criminals: Designing patrol allocation against adaptive opportunistic criminals. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 1351–1359, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-3413-6. URL http://dl.acm.org/citation.cfm?id=2772879.2773326.

VITA

VITA

Mahmuda Rahman was born in Bangladesh. She received her Bachelor of Science degree in Computer Science at North South University (Dhaka, Bangladesh). She received her Masters in Information Technology form the University of Dhaka (Dhaka, Bangladesh). She received her PhD in Computer and Information Science and Engineering from Syracuse University (Syracuse, New York, USA) in December 2017.