Appl Intell (2018) 48:1979–1995
DOI 10.1007/s10489-017-0977-1

Graph bandit for diverse user coverage in online recommendation

Mahmuda Rahman[1] · **Jae C. Oh**[1]

Published online: 16 June 2017

Abstract 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 introduced 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. Even the static problem is known to be NP-hard (Hochba, ACM SIGACT News 28(2):40–52, 1997) 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 construct a relational graph between items to recommend. Experiments show that the new graph based algorithm performs better than existing techniques such as Ranked Bandit (Radlinski et al. 2008) and Independent Bandits (Kohli et al. 2013) in terms of satisfying diverse types of users while minimizing computational complexity.

✉ Mahmuda Rahman
mrahma01@syr.edu

Jae C. Oh
jcoh@syr.edu

[1] DMA Lab, EECS, Syracuse University, Syracuse, NY, USA

Keywords Recommendation system · Online learning · Diversity · Multi armed bandit · Upper confidence bound · Directed graph

1 Introduction

Recommendation systems in the recent era not only need to address the huge amount of users to satisfy but also their rapidly changing patterns of preferences. With such a fast and continuous shift of users' preferences, a 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 induces 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 recommend to a user, yet they are still required that at least one of these recommended items satisfies her taste. Our graph-based 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 many different user types. In this paper, we define a user by his or her choice of items, therefore, users having the same preference over items are considered to be the same user type.

The graph-based algorithm can discover correlations among the items so that related items can be represented by one of the items. In other words, if a recommendation system (with a fixed small number of items to recommend) could pick only one from a group of related items, it would be able to satisfy all users of that user type. This method effectively reduces the number of items to be considered

thereby reducing computational complexity of the problem. Consequently, the new algorithm can satisfy more diverse user types.

As we need to address dynamically changing user preferences over time, the correlation graph also changes as data stream in. Therefore, an effective and efficient way to update the correlation graph must be designed. Also, the environment is partially observable, as feedback of a user on a given recommended items does not expose other item choices of that users. We formally defined this as Online k-Hitting Set Problem and present an algorithm to address this problem.

We list some of the challenges which made our problem interesting. Then we state our contributions to overcome those challenges.

- The problem of choosing the optimal set of recommended items for a given user population is an NP hard problem [13] even if all the user vectors are given offline. This problem is equivalent to the maximum coverage problem [8] .
- Partial observablity of the problem can only allow our algorithm to examine the feedback from a user after the recommendation is made. A negative feedback gives no information about what would the user prefer instead.
- When a new user data streams in, the system must make decision whether to categorize the new user to an existing user type or create a new user type, if the input data is significantly different from any existing user types. This problem is common to all machine learning classification and clustering problem. It is also known as the *open-set* classification/identification problem [3, 19].

User abandonments [9] in recommendation system is that the system gives up satisfying certain user types. For example, if a user type is quite unique and the population within the type is small, it may be better to ignore the user types to accommodate user type with larger population. However, a good recommendation system must minimize user abandonments. Our method minimizes the user abandonment while maximizing the payoff of the system.

The contributions of this work minimizing the abandonment are following:

- formally defining the novel problem of Online k-Hitting Set and present a graph based anytime approximation algorithm for applying it to online recommendation system. Our robust problem formulation facilitates the design of this graph based bandit algorithm.
- developing a graph based "update policy" and "selection policy" for UCB1 bandits for recommendation systems to address the Online k-Hitting Set Problem, unlike our previous work [22]. The main difference between the previous work and the proposed approach in this paper is that in this new approach, the recommendation system (1) learns the dependency structure among items though a novel Graph Based Update Policy and (2) selects the most promising items by calculating the potential of an item faster by a Graph Based Selection Policy
- 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.

On an average, our proposed mechanism outperforms existing recommendation techniques in terms of covering user types while keeping the computational complexity low.

This paper is organized in the following way: in Section 2 we discuss the online learning problem for recommendation system formulated as an Online k-Hitting Set Problem under partial observation. In Section 3 we discuss the bandit setting for the problem. In Section 4 we cite some of the related works. Section 5 describes our past approach to overcome the issues regarding bandit algorithm for online recommendation system and illustrate the limitations of the past approach. Section 6 details our proposed graph based model to overcome the shortcomings of the past approach as we detail our novel update policy and selection policy. In Section 7 we analyze our approach and in Section 8 we give empirical evaluation. In Section 9 we conclude the paper.

2 Coverage of diversity as online k-hitting set problem

Our problem can be formulated as a variation of the hitting [29] problem, where the number of sets and elements within the sets can change dynamically. 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. Because a hitting set contains at least one element from every subset in a collection. It becomes more challenging when the sets are streaming online an the system does not have any knowledge about the distribution of elements over these sets.

For our problem, the collection contains all the available items to recommend from and every subset of the collection defines a user's preferences. As users' preferences are diverse, these incoming sets to the learning system can change over time, the recommendation set as a hitting set of fixed size k needs to be adjusted with this change dynamically. Also, we need to maximize the coverage of different

types of users over time. We call it as an Online k-Hitting Set Problem.

The data stream in our problem is partially observable as we can only examine the feedback from a new user after the recommendation is made based on the earlier observations. A negative feedback only notifies the system that none of the element of our recommendation set is selected by the user, but gives no information about what would the user prefer 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, $C = \{S_1, S_2, ..., S_n\}$ s.t. $S_i \subseteq U$ for $i = 1, 2, \ldots n$ where n is not known, 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 $S_i^t \cap A^t \neq \emptyset$.

This definition helps us address the online recommendation problem with an intention to cover diverse users where S_i^t is a user's choice vector and A^t is the recommendation set for that user at time t. 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 $\frac{\Sigma_{t=0}^{T}\delta_t}{T}$. Hence this novel problem formulation helps us design an anytime approximation algorithm as a solution to the online recommendation problem.

The difference between classical Hitting Set Problem and our problem is that we want $|S^{'}| = k$ for the above definition where $S_i \in C$ for $i = 1, 2, \ldots n$ 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. Our algorithm constructs an online approximation for a hitting set for the partially observed sets in the collection as seen over a certain amount of time.

We apply our algorithm to the real data set for movie recommendation where each incoming user can be treated as a set with the index of the movie of her choice. Most existing literature including [16] 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. Kohli et al. [16] defined a user relevance vector as following:

Definition 2 Each user j can be represented by a $\{0, 1\}^n$ vector X^j where $X_i^j = 1$ 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 the user vectors is unknown to the system but these vectors are assumed to express the "types" of a users. We assume that there is a large degree of correlation between these vectors as we treat them to be different sets S coming from a single collection of sets C. It is important to note that when the system successfully recommends a set of items to a user, that recommendation set covers other users of having at least those item of choice as in the recommendation set.

At time t, after a recommendation set is generated, a choices of a random user vector is disclosed to 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. Given a static dataset, to simulate an online learning problem environment of the arrival of a random user at time t, the algorithm chooses a vector X^t (as S_i in above definition) independent and identically distributed from an unknown distribution D from all user vectors.

Then the recommendation system presents a set of k items S^t (as A^t in above definition) without observing X^t. We also adopted the definition of set relevance function F used by the above mentioned paper to follow the convention:

Definition 3 $F(X^t, S^t)$ is a submodular set function [26] 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 [16], the value of displaying a set S^t, is the expected value $E[F(S^t, X)]$ where the expectation is taken over 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 [16], our target is to minimize abandonment, so we need to maximize click through rate [15], which is:

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

This is essentially same as our measurement metrics for the Online k-Hitting Set Problem we formulated at the beginning of this section (Fig. 1).

3 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 our online hitting set. As we can pick at most k number of elements at certain time, our problem is an Online k-Hitting Set Problem with an intention to cover maximum possible incoming sets.

We leverage Multi Armed Bandit [28] algorithm, specifically UCB1 [2] to solve this problem. The Multi-Armed Bandit (MAB) problem is the problem a gambler faces given a slot machines with multiple levers (arm), deciding which arm to play, how many times to play a specific arm and the order to play them 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.

The most popular stochastic bandit algorithm, UCB1 [2] has been used as our base, where each slot of k-element set runs a UCB1 bandit to pick an element i from n available options. Here n is considered as the size of $|U|$ defined in our problem. UCB1 selects an element i for which the following **UCB value** is maximum.

$$x_i + \sqrt{\frac{2\log(f)}{f_i}}$$

Here x_i denotes the current average reward of the element i in terms of being a candidate for our hitting set. f_i denotes the number of times element i has been picked so far in total t rounds and f denotes the total count of all elements picked so far as $f = \Sigma_{i=1}^{t}(f_i)$.

After each pulling of arm, and reward, the corresponding arm 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 $\sqrt{\frac{2\log(f)}{f_i}}$.

For an online recommendation system, at each time t, from n available items (arms), k items need 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 has been selected to place at j slots by the j^{th} bandit, that item will be unavailable from the rest of the bandits $j, j+1 ... k^{th}$ bandits. 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.

This is called Upper Confidence Bound (UCB1) [2] 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.

Provided that a UCB1 algorithm has tried enough of each items to be reasonably confident, it rules out the chance that a selected item would be sub-optimal or inferior in terms of achieving reward. While we would like to include this apparently superior arm (item), we have to make sure that the other arms (items) are sampled enough to be reasonably confident that they are indeed inferior. UCB1 does that for us, but unfortunately UCB1 assumes all n items are independent. However, in many applications items are not independent. For example, if a customer likes a certain grocery item, she may also like other related items. Our

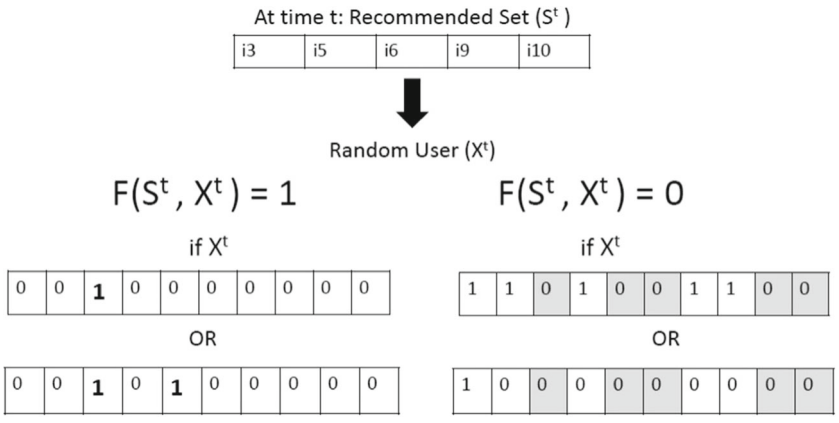

Fig. 1 Payoff for a recommendation set according to user's click response

algorithm also addresses the dependencies among items. In fact, our algorithm leverages the dependencies to maximize the coverage of user preferences.

4 Related work

In the existing work, two different approaches have been found to build a recommendation system by using UCB1 bandits. We discuss their advantages and disadvantages in this section. The Ranked Bandit Algorithm (RBA) [21] 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) 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 [16]. 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) [16] assumed independence between items and used Probability Ranking Principle (PRP) [24] as a greedy method to select items to recommend. PRP give equal credit to 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 if more than one item is selected by that user. But 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 used both RBA and IBA as our baselines in the experiment to compare with our past approach which we call non graph based method and compared our graph based approach with our past approach. Throughout this paper we use the term 'our past approach' and 'Non Graph Based method' interchangeably. Online Learning problem has been also studied as a Multi Armed Bandit (MAB) Problem in [27] as Restless Bandit based on Markov chain. Mortal Bandit problem [7] 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. But their solution is based on a number of predefined parameters, which need to be set by empirical evaluation. Volatile bandit [5] extends the mortal bandit with an update to exploration and exploitation trade off.

Rahman and Oh [23] devised a synchronized and parallel UCB2 algorithm for covering maximum variety of users in an online setting and constructed a recommendation set of items satisfying maximum heterogeneous users. Later they used UCB1 for the same purpose [22]. They empirically showed that their approach works better and faster than existing approaches like RBA [21] or IBA [16] to cover heterogeneous types of users with fixed size of recommended items. Their problem is similar to ours as we are trying to construct a limited size set incorporating elements which belongs to maximum number of different sets in a collection of sets as our work treats each incoming user as a set and generate an online hitting set of size k as a recommendation set. However, they introduce reward bias to the first item chosen by a user which can result in failure of covering the minority user-types. This is discussed with illustrative examples in the next section.

Apart from these, most recent graph based recommendation system has been proposed by [17] where authors used the concept of entropy and the linked items in the graph on their attempt to find recommendations that are both novel and relevant. Nevertheless, they mention that their proposed system does come with its weaknesses; the variance in the relevance of recommendations is high due to the use of items with high entropy as novel items. Unlike a defined trade off between exploration and exploitation of items, there exists a degree of unexpectedness with irrelevant recommendations in their approach that rise from the randomness and risk-taking by their entropy based method. Also, it is difficult to explain specifically why the recommendations were given aside from providing related items from the user profile. Their approach is offline approach as they need the entire user vector to search for the items to recommend. Also, they were not concerned with the issue of user coverage.

5 Preliminary approach: reward bias for bandits

Algorithm 1 shows our past approach we used in [22] for introducing reward bias to shrink the search space for possible candidate items to generate a recommendation set. According to this method, 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 [16], 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 item preferred by the user, the first bandit responsible for picking the preferred item gets a relatively 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 strengthen the average reward for the bandit who picked the first item the user preferred. This creates a bias towards the first item a user prefers and helps recommending users of same user type with one preferred item; by finding one representative item for each user-type, we can maximally utilize the recommendation of vector size k. However, this approach only allows the first item to be chosen in the user preference as the representative item for a user type. We discuss this shortcoming in the next section.

Algorithm 1 Non graph-based approach

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

5.1 Shortcomings of the reward bias

One problem with our past approach is that the unequal rewarding scheme still can exclude many user types with smaller populations (we refer them as 'minority user-types.') The basic idea behind our past approach [22] follows Probability Ranking Principle (PRP) [24] which allows to rank items in decreasing order of relevance probability without considering the correlations between them as in [16]. But unlike [16], 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. The idea is that, 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 principle, no bandit could ever select that overlapping item which could cover both the user-types. This problem is illustrated with an example. 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 causes 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 out numbers user-type2, which may take a long trial. 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. Figure 2 illustrates this situation.

6 Proposed method: graph based UCB1 bandits

In this section we discuss the mechanism we devise to modify the **update policy** and **selection policy** of UCB1 bandit to find the solution to our problem using graph based bandits. To handle the partial observability of streaming sets, our work leverages the relationship between elements where each element is considered as a predefined **node** of a dynamic graph.

Before any call for Update Policy, our algorithm will select an unselected arm from all the available arms uniformly random. Algorithm 2 shows the overview of both the selection and update of arms. According to this algorithm, k number of bandits are responsible for selecting k non identical elements for our hitting set as illustrated in Fig. 1.

Algorithm 2 Modified UCB1 policy

1: Given k as a fixed integer
2: Create graph $G^0(V, E)$ consists of isolated nodes where $v_n \in V$ for each item $n \in N$ and initialize $E = \{\}$
3: **for all** $n = 1$ **to** N **do**
4: Let $w_n \in W = 0$ for $v_n \in V$ where W is the weight vector for nodes
5: **end for**
6: $R_0^t \leftarrow \{\}$
7: **for all** $i = 1$ **to** k **do**
8: $SelectionPolicy(UCB1_i(N), N \setminus R_{i-1}^t)$
9: **end for**
10: $UpdatePolicy(G^{t-1}, R^t, B, k)$

6.1 Update policy for graph based bandits

As we encounter the issues mentioned in the previous section with our past approach [22], we realized that when we increase the reward of a bandit for picking the first item preferred by a user-type, we also need to make sure there is a discount on that reward if that item is not sufficient to address the satisfaction of other user-types. This inter-user-type discount factor is not trivial to compute because this requires remembering history. In our present work, we introduced the Relevance relationship between items, which remembers history of relationships among items in a computationally efficient way:

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 user is a logarithmic function of the weight of the node representing that item in the graph.

6.1.1 Construction of relevance item graph

To keep track of relations among the items seen and recommended so far in the online environment, we need to build and update a relevant 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.
- Each node has a weight associated with it which denotes the relevance of the item in the graph. Initially all nodes have an 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*.

Algorithm 3 shows how the graph is evolving dynamically.

Algorithm 3 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**

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

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

6.1.2 Assignment of node weight

As we construct the graph dynamically and 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 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. Figure 3 shows how this weighting scheme alleviates the issue related to our past approach and better handles the overlapping items chosen by different user types.

6.1.3 Discounting factor on correlation

As we select random sample of users over a period of time for real simulation, we scale down the importance of older correlations as opposed to the newer ones after every τ time steps. We keep τ to be a fixed period of time which we call an "epoch." Within an epoch all items in the recommendation set that appears together and has a match with the sampled user instances, is given same discount. Node weights increase between epochs as we tend to give more importance to relations between items coming from the recent samples. As we are sampling a sufficient amount of user instances in uniform random manner, it ensures our algorithm to retain the popular items in the recommendation set while facilitating diverse user instances according to our novel rewarding mechanism. Algorithm 4 is developed on this discounted reward for bandits.

6.1.4 Rewarding the corresponding bandit

We update the rewards of the corresponding bandits who picked the items in the recommendation set after each iteration, so that bandits gets incentives to pick the appropriate items to cover different user-types. The bandits who are responsible for picking the corresponding items in the recommendation system gets rewarded proportional to the weight of the node representing that item in the graph. This way, over the iterations, edges connect the nodes and their associated weights are accumulated. Bigger the log difference among the weights of different items, better the bandit selects the more rewarding item among them.

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:

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

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

6.2 Selection policy for graph based bandits

In previous sections we mainly showed how we modify the reward update policy for graph based bandits. This section discusses 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.

Algorithm 5 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:

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

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

6.2.1 Annotation of arms states

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 annotate 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 the 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 stay active. Because those arms clearly indicate its failure to cover that specific incoming set at a certain iteration, but our algorithm does not rule out its potential for covering other incoming sets from the collection.

Algorithm 5 shows a complete Graph Based UCB1 update policy for bandits with this annotation of Arms.

6.2.2 State transitions of arms

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 matched 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 the update policy as well as the our adaptive selection policy.

We adopted an Adaptive Simulated Annealing [4] 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

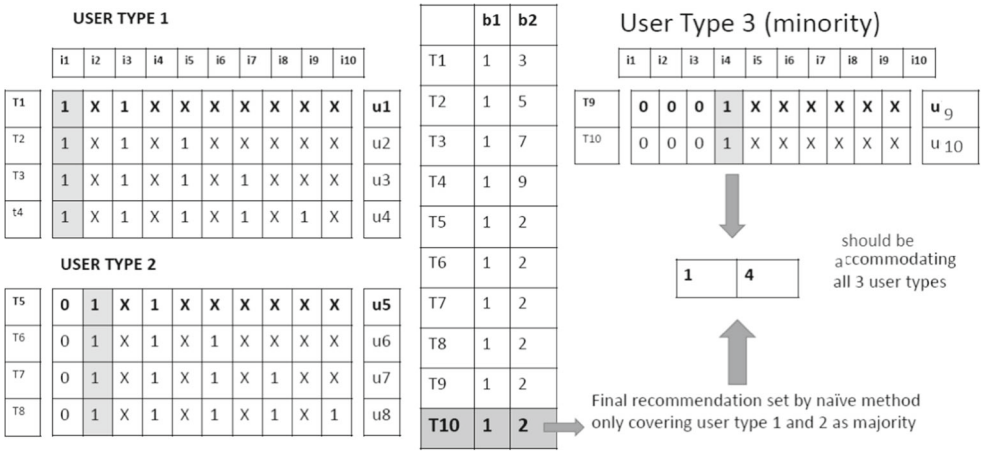

Fig. 2 Limitation of Non Graph Based method where Ti labels the 2 item recommendation set at i^{th} time step coming from 2 bandits b1 and b2 and u_i denotes the user vector it is recommended to

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.
- 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.

The relationship between α and the counter r is the following:

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

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

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:

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

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

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

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

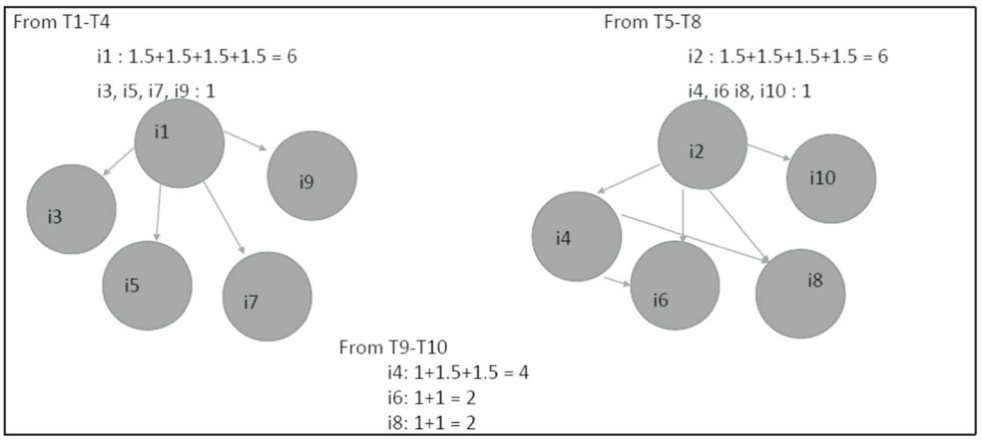

Fig. 3 Emergence of item 4 with increasing weight as a potential replacement of item 2 in the graph based method to accommodate user type 3 as in the illustrative example mentioned in Fig. 2

opportunity to balance between exploration and exploitation. On the other hand, 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 $(1 - \exp^{-1/\theta})$. 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.

Our modified arms selection policy is shown in Algorithm 6.

Algorithm 6 Selection Policy for Bandits

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

7 Analysis of our approach

Unlike the classic UCB1, which assumes all n items are independent, for our case, there are dependency between elements as they appear together in the same set and equally qualify to represent that set. Therefore, we propose the above modification in the selection policy for UCB1. This way our algorithm achieves the coverage of as many unique set element possible in our hitting set. As we introduce the bias in the rewarding scheme introduced by graph based bandit to resolve the dependency between elements, our update policy always emphasizes more on selecting an element for the hitting set which matches with the first element of the incoming set. Our selection policy balances this bias by giving dormant arms (which is not the first match) a fair chance to be active and represent more sets to which it belongs.

For example, let there be only one element to select from two elements: $i, j \in U$ to cover maximum number of the sets in a collection and both of them have achieved same average reward (x_i and x_j are the same) after some random number of trials. Now, if element i has been tried more often than element j, then $f_i > f_j$ with same f as a numerator. Then $\sqrt{\frac{2\log(f)}{f_i}} < \sqrt{\frac{2\log(f)}{f_j}}$. So confidence bound shrinks for i more than j. Again, this bound grows if f gets higher. This confidence on element i does not take into account any correlation between i and j. But once i and j are found together in certain number of sets, then one of them needs to be selected for our hitting set and that element becomes a candidate arm and hence become "active" to exploit.

Let i be the active and j be the dormant arm found in certain time step. It is possible that arm j has not been tried enough because i was always the first match and j was a subsequent match in the incoming sets. So update policy has always been biased towards i to represent the sets both i and j belong. But selecting j could cover an incoming set for which i is not a member. So selecting i more times may result in poor online performance. Our modified selection policy introduces a scaling factor to this confidence term in UCB1 so that all candidate arms for an online hitting set get equal opportunity to actively represent the sets they belong and this way a set of k elements is constructed.

Provided that our algorithm has tried enough of each element to be reasonably confident, it rules out the chance that a selected element would be sub-optimal or inferior in terms of achieving reward. While we would like to include this apparently superior arm (element), we have to make sure that the other arms (elements) are sampled enough to be reasonably confident that they are indeed inferior by testing their potential under a scaled down UCB value.

As for the complexity of the algorithm, [21] 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 [14], k is the number of bandits, $R(T)$ is the regret for that specific type of Bandits. Later for IBA [16], 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.

Pandey et al. [18] addressed this dependency between arms by using MDP to set policy to generate cluster of

dependent arms and then choose an arm from that cluster. But their technique is computationally expensive. Graph based bandit attempts to resolve the dependencies among items by selecting a representative arm to represent such a cluster by introducing a bias in the reward of that arm based on its frequency of dependency on other arms. But the modification of update policy was not sufficient for better coverage, therefore, in this work we augment a novel selection policy with it.

8 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 sets.

8.1 Size of recommendation set

We used publicly available Jester dataset [11] and Movielens1000 [12] data set for our experiments. Jester Project has a small dataset is a collection of user ratings on jokes where the range of rating runs from -1.0 (not funny at all) to 10.0 (very funny). It consist of more than 17000 users rating on 10 jokes (which most users have rated). The real valued ratings has been converted to binary (relevant or not) by using a threshold rule of exceeding θ which is set to 3.5 in our experiment. In Figs. 4 and 5, we show our results for $k = 3$ and $k = 5$ as we recommend 3 or 5 items from available 10.

From Figs. 4 and 5 it is found that for $k = 3$ all the methods performs lower than while $k = 5$ on an average. Which is expected, as recommending more items have the potential to cover more users. In fact, for $k = 5$ our graph bandits outperforms RBA and in long run performs competitive with other methods. But 5 out of 10 was too optimistic a range for choices so we experimented next with Movielens dataset.

8.2 Variation of user choices

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 Figs. 6 and 7. 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.

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.

8.3 Benchmarking with offline greedy coverage

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 of its user coverage. In this set of experiments, we show the result of Movielens dataset with $\theta = 2$ but other threshold value for

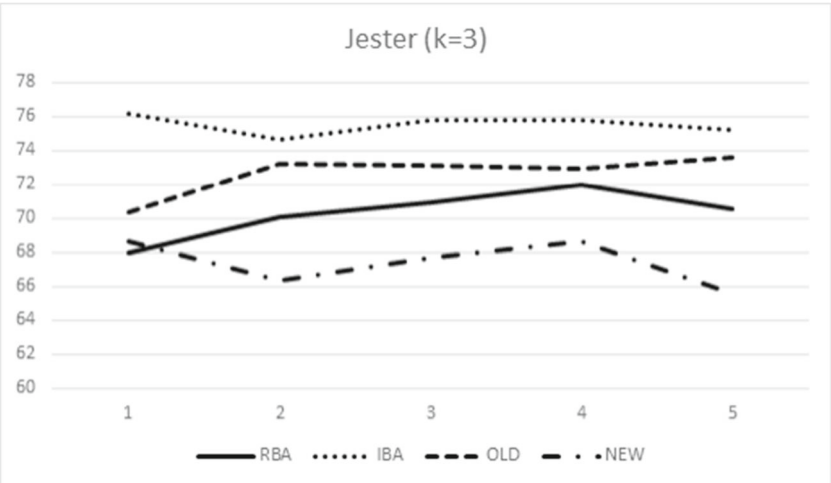

Fig. 4 Performance comparison of recommending 3 jokes out of 10 from Jester dataset, Y-axis showing x1000 times iteration

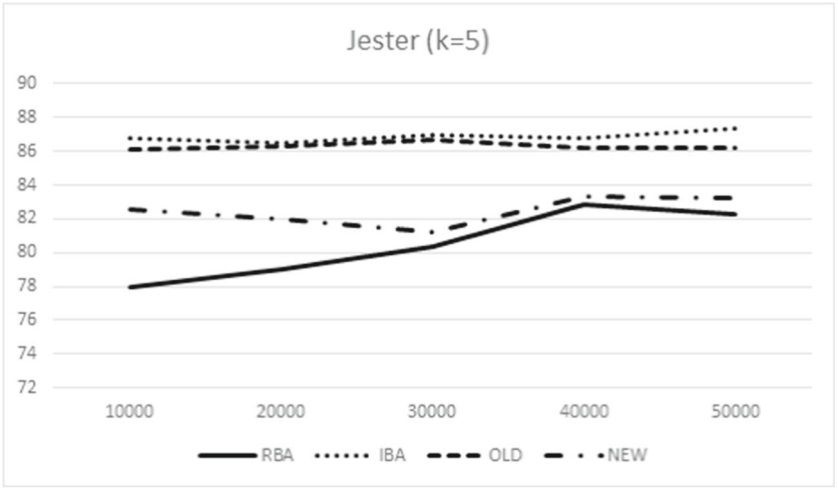

Fig. 5 Performance comparison of recommending 5 jokes out of 10 from Jester dataset

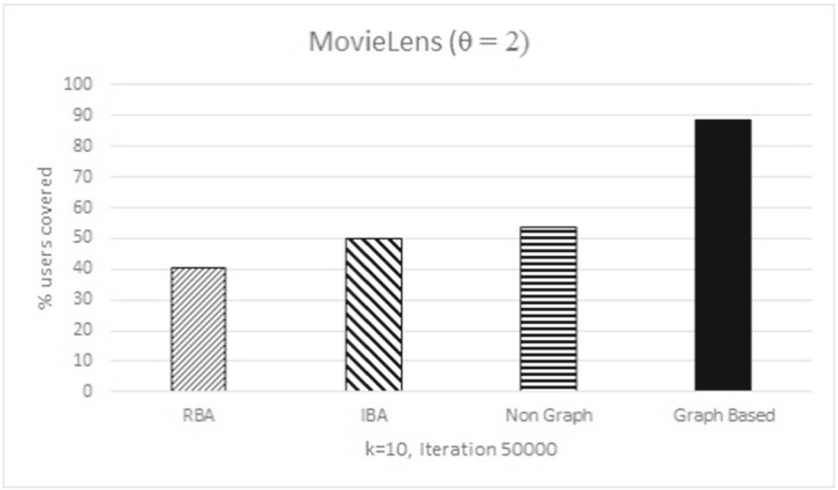

Fig. 6 Performance comparison of recommending 10 movies out of 1682 in Movielens dataset for low cutoff value θ

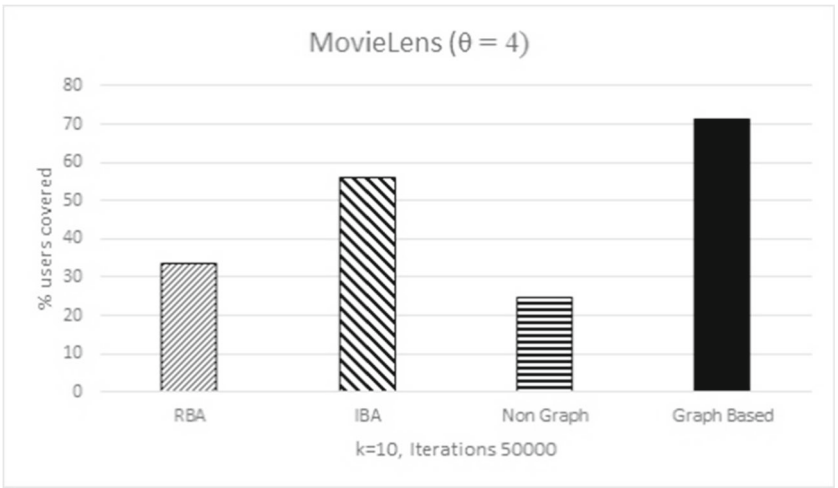

Fig. 7 Performance comparison of recommending 10 movies out of 1682 in Movielens dataset for high cutoff value θ

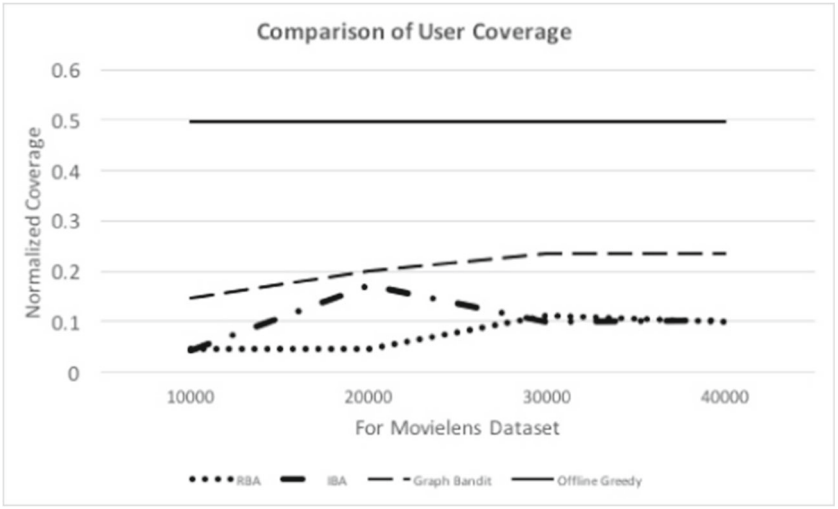

Fig. 8 Normalized user coverage of online algorithms compared to offline baseline for Movielens data set where $\theta = 2$

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 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 Fig. 8 normalized by the total population. According to this result, it is found that our method outperforms other online algorithms in terms of user coverage.

8.4 Diversity of recommendation set

We compared the effectiveness of recommendation set after each 10 K iteration upto 50 K to observe its performance on capturing diversity. We used the popular Intra-List Distance (ILD) [30] metric where 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 Fig. 9. 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.

8.5 Reduction of active arms

We incorporated our novel Selection Policy with the Update Policy and experiment further with Movielens data set for $\theta = 2$ and 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. Figure 10 shows the average coverage over 50 K iterations. It is found that, with our novel selection policy, the coverage of same fraction

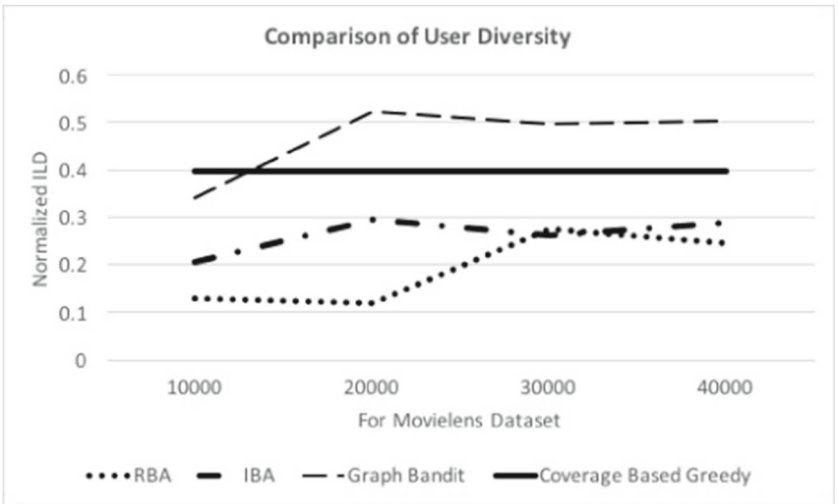

Fig. 9 Normalized user diversity captured by online algorithms compared to greedy coverage for Movielens data set where $\theta = 2$

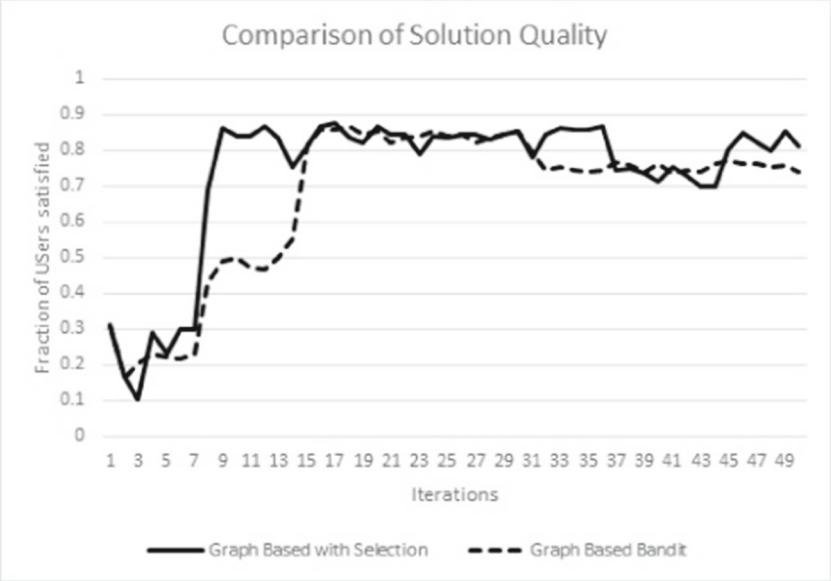

Fig. 10 User coverage over the x1000 iterations: with only our Graph Based Update Policy (labeled as Graph Based) and our Graph Based Update Policy augmented with our Graph Based Selection Policy (labeled as Graph Based with Selection) with Movielens data where $\theta = 2$

of overall user is achievable within 10K iteration, whereas without this modification the graph based bandit algorithm takes around 20K to attain the same outcome. As active arms are identified faster, our algorithm produce more efficient and stable set. Even after introducing update policy for the graph based bandit, it gives a low quality solution for a longer time, unless we incorporate the novel selection policy which leads to a better solution quality faster. The average active arm counts over the iteration drops faster when selection policy compliment our update policy for graph bandits as shown in Fig. 11.

9 Conclusion

Modern web-based applications require the ability to extract important information from the vast amount of streaming data. Big-data with its volume and velocity, challenges us to design systems that can extract a small and limited amount of critical information effectively and efficiently out of its dynamically changing patterns. Online active learning is an emerging area of research with special interest in Big Data for its applicability to real world challenges such as, developing a recommendation system.

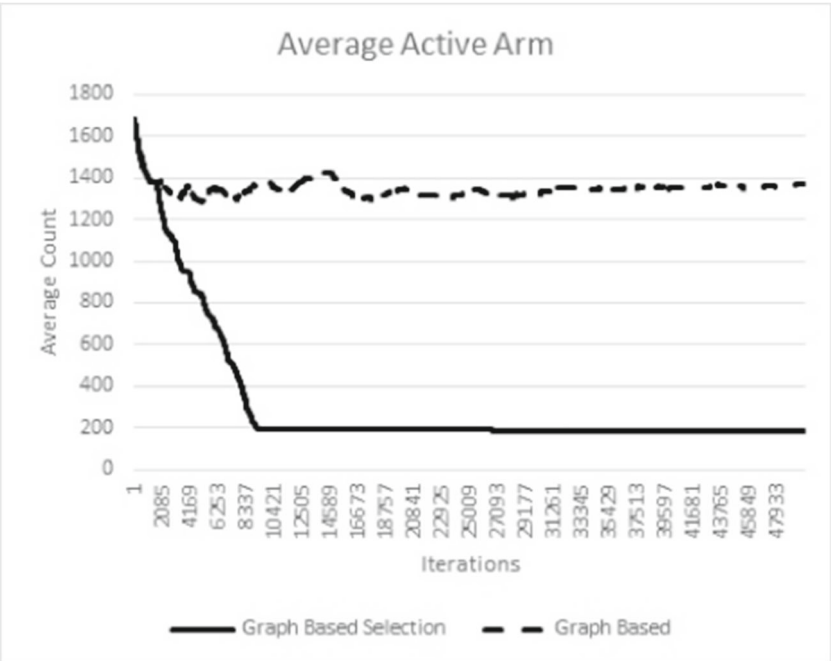

Fig. 11 Average count of active arms drops faster after incorporating our novel selection policy (labeled as Graph Based with Selection) for arms with Graph based reward update (labeled as Graph Based) for Movielens data where $\theta = 2$

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 understand 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.

Unlike personalized recommendation systems [25] which often use collaborative filtering [10, 11], content based filtering [20] or a hybrid of these two [6], we develop a recommendation system that satisfies a diverse number of user types. An existing paper argued that dependency among items can be ruled out by ignoring the correlation gap [1, 16] . But we show that, the correlation between items is an important criteria to identify diversity in terms of user types. We proposed a effective graph based bandit rewarding mechanism, which aimed to incorporate this diversity and our empirical evaluation showed that it outperformed existing techniques for real data sets in terms of covering a large number of user types introducing no additional complexity. In future, we want to extend this solution to a distributed recommendation system to facilitate a scalable and decentralized decision making for Big Data.

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 correlations among the items. 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.

References

1. Agrawal S (2011) Optimization under uncertainty: bounding the correlation gap. Ph D Thesis. Stanford University
2. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. Mach Learn 47(2-3):235–256
3. Ausiello G, Boria N, Giannakos A, Lucarelli G, Paschos VT (2011) Online maximum k-coverage. In: International symposium on fundamentals of computation theory. Springer, pp 181–192
4. Azizi N, Zolfaghari S (2004) Adaptive temperature control for simulated annealing: a comparative study. Comput Oper Res 31(14):2439–2451
5. Bnaya Z, Puzis R, Stern R, Felner A (2013) Volatile multi-armed bandits for guaranteed targeted social crawling. In: Late-breaking developments in the field of artificial intelligence. Bellevue, p 2013
6. Burke R (2007) The adaptive web. In: Brusilovsky P, Kobsa A, Nejdl W (eds) Hybrid web recommender systems. Springer, Berlin, pp 377–408
7. Chakrabarti D, Kumar R, Radlinski F, Upfal E (2008) Mortal multi-armed bandits. In: Advances in neural information processing systems 21, proceedings of the 22nd annual conference on neural information processing systems. Vancouver, pp 273–280
8. Cohen R, Katzir L (2008) The generalized maximum coverage problem. Inf Process Lett 108(1):15–22
9. Diriye A, White R, Buscher G, Dumais S (2012) Leaving so soon?: understanding and predicting web search abandonment rationales. In: Proceedings of the 21st ACM international conference on information and knowledge management CIKM '12. ACM, New York, pp 1025–1034
10. Ekstrand MD, Riedl JT, Konstan JA (2011) Collaborative filtering recommender systems. Foundation and Trends in Human Computer Interaction 4(2):81–173
11. Goldberg K, Roeder T, Gupta D, Perkins C (2001) Eigentaste: a constant time collaborative filtering algorithm. Inf Retr 4(2):133–151
12. Harper FM, Konstan JA (2015) The movielens datasets: history and context. ACM Trans on Interactactive Intell Syst 5(4):19:1–19:19
13. Hochba DS (1997) Approximation algorithms for np-hard problems. ACM SIGACT News 28(2):40–52
14. Hochbaum DS, Pathria A (1998) Analysis of the greedy approach in problems of maximum k-coverage. Nav Res Logista (NRL) 45(6):615–627
15. Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining KDD '02. ACM, New York, pp 133–142
16. Kohli P, Salek M, Stoddard G (2013) A fast bandit algorithm for recommendations to users with heterogeneous tastes. In: Proceedings of the 27th AAAI conference on artificial intelligence AAAI '13. AAAI Press, pp 1135–1141
17. Lee K, Lee K (2015) Escaping your comfort zone: a graph-based recommender system for finding novel recommendations among relevant items. Expert Syst Appl 42(10):4851–4858
18. Pandey S, Chakrabarti D, Agarwal D (2007) Multi-armed bandit problems with dependent arms. In: Proceedings of the 24th international conference on machine learning ICML '07. ACM, New York, pp 721–728
19. Park W, Oh JC, Blowers MK, Wolf MB (2006) An open-set speaker identification system using genetic learning classifier system. In: Proceedings of the 8th annual conference on genetic and evolutionary computation GECCO '06. ACM, New York, pp 1597–1598
20. Pazzani MJ, Billsus D (2007) Content-based recommendation systems. In: The adaptive web: method and strategies of web personalization. Vol 4321 of lecture notes in computer science. Springer, pp 325–341
21. Radlinski F, Kleinberg R, Joachims T (2008) Learning diverse rankings with multi-armed bandits. In: Proceedings of the 25th international conference on machine learning ICML '08. ACM, New York, pp 784–791
22. Rahman M, Oh JC (2015) 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. Springer, Switzerland, pp 361–370

23. Rahman M, Oh JC (2015) Parallel and synchronized UCB2 for online recommendation systems. In: IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology, WI-IAT 2015, vol I. Singapore, pp 413–416
24. Robertson SE (1997) In: Sparck Jones K, Willett P (eds) Readings in information retrieval. Morgan Kaufmann Publishers Inc., San Francisco, pp 281–286
25. Shani G, Gunawardana A (2011) Evaluating recommendation systems. In: Recommender systems handbook. Springer, pp 257–297
26. Sviridenko M (2004) A note on maximizing a submodular set function subject to a knapsack constraint. Oper Res Lett 32(1):41–43
27. Tekin C, Liu M (2012) Online learning of rested and restless bandits. IEEE Trans Inf Theory 58(8):5588–5611
28. Vermorel J, Mohri M (2005) Multi-armed bandit algorithms and empirical evaluation. In: Proceedings of the 16th european conference on machine learning ECML '05. Springer, Berlin, pp 437–448
29. Vinterbo SA, Øhrn A (2000) Minimal approximate hitting sets and rule templates. Int J Approx Reason 25:123–143
30. Zhang M, Hurley N (2008) Avoiding monotony: improving the diversity of recommendation lists. In: Proceedings of the 2008 ACM conference on recommender systems RecSys '08. ACM, New York, pp 123–130

Mahmuda Rahman is a PhD Candidate in Computer Science Department of Syracuse University. She is currently working in the Distributed Multi Agent Lab. Her research interest includes Machine Learning and Game theory. She is also interested in Big Data, Social Networks, Data Mining and Recommendation Systems. Currently she is an Assistant Professor in the University of Dhaka and on study leave. She did research internships with eBay, Huawei and IUPUI. She is an external reviewer/sub reviewer of IEA/AIE since 2014.

Jae C. Oh Ph.D. is a faculty member in the Computer Science program in the Department of Electrical Engineering and Computer Science at Syracuse University. Oh's research interests include understanding interaction dynamics among autonomous agents and studying problems that arise among distributed entities such as robots and software processes. Oh's research also involves in Big-Data analysis and visualizations, social networks, recommendation systems, multi-robot coordination, and text and data mining using various machine-learning algorithms. Jae Oh's projects have been funded by various funding agencies including the US National Science Foundation, US Air Force Office of Scientific Research, the State of New York, IBM, and several private companies. Jae Oh chaired or co-chaired several conferences and he is a member of technical program committee for numerous conferences. He is an Associate Editor for International Journal of Computer Information Systems and Industrial Management Applications since 2013. Jae Oh has earned a Ph.D. degree in Computer Science from the University of Pittsburgh, Pittsburgh, PA.