Spring 5-5-2016

A Graph-based Bandit Algorithm for Maximum User Coverage in Online Recommendation Systems

Mahmuda Rahman
Syracuse University, mrahma01@syr.edu

Jae C. Oh
Syracuse University, jcoh@ecs.syr.edu

my journal manuscript No.
(will be inserted by the editor)

A Graph-based Bandit Algorithm for Maximum User Coverage in Online Recommendation Systems

Mahmuda Rahman · Jae C. Oh

Received: date / Accepted: date

Abstract We study a type of recommendation systems problem, in which the system must be able to cover as many users' tastes as possible while users' tastes change over time. This problem can be viewed as a variation of the maximum coverage problem, where the number of sets and elements within any 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. 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 user types as it construct a relational graph between items to choose. Experiments show that the new algorithm performs better than existing techniques such as Ranked Bandits [17] and Independent Bandits [12] in terms of satisfying diverse types of users while minimizing computational complexity.

Keywords Recommendation System, Online Learning, Diversity, Multi Armed Bandit, Upper Confidence Bound, Directed Graph

Mahmuda Rahman
CST4-288, DMA Lab, EECS, Syracuse University
E-mail: mrahma01@syr.edu

Jae C. Oh
Director, DMA Lab, EECS, Syracuse University
E-mail: jcoh@syr.edu

1 Introduction

Recommendation systems in 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 the desirability of 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 to satisfy the user with at least one of these recommended items. 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 among 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. Because 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. We present an algorithm to address this problem as well.

Now we list some of the challenges which made our problem interesting. Then we state our contributions to overcome those challenges. The challenges are:

- The problem of choosing the optimal set of recommended items for a given user population is an NP hard problem [9] even if all the user vectors are given offline. This problem is equivalent to the maximum coverage problem [5].
- There is a greedy optimal solution to this problem [10] that can be known given the entire user vector offline. But to identify that optimal set, each k-subset of all n items needs to be tried as a potential candidate, so there are exponentially many options for the trials [17].
- When a new user data streams in, the system must make a decision on 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. In fact, this problem is common to all machine learning classification and clustering problem. It is also known as the *open-set* classification/identification problem [3, 15].

User abandonments [6] in recommendation system is that the system gives up in 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 type to accommodate user types with a larger population. However, a good recommendation system must minimize user abandonments. Our method minimizes the user abandonment while maximizing the payoff of the system. Some contributions and results are:

- developing a graph based mechanism for recommendation systems where the system learns the dependency structure among the items though a novel rewarding scheme.
- verifiing the efficacy of our method for choosing a very small number of items to recommend from real data sets where there are hundreds of users, each having thousands of choices to pick from.

On average, our proposed mechanism outperforms existing recommendation techniques in terms of covering different 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 formally. In Section 3 we cite some of the related works. Section 4 discusses the bandit setting for the problem. Section 5 describes our past approach to overcome the issues regarding bandit algorithm for online recommendation system and Section 6 illustrates the limitations of past approach. Section 7 details our proposed graph based model to overcome the shortcomings of the past approach. We used the works mentioned in Section 3 as baselines to compare the performance of our past approach and graph based method in Section 9. In Section 9, we conclude.

2 Problem Description

Most existing literature including [12] 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. [12] defined a user relevance vector as following:

Definition 1 *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.*

At time t, after a recommendation set for a random user is already generated, that specific 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.

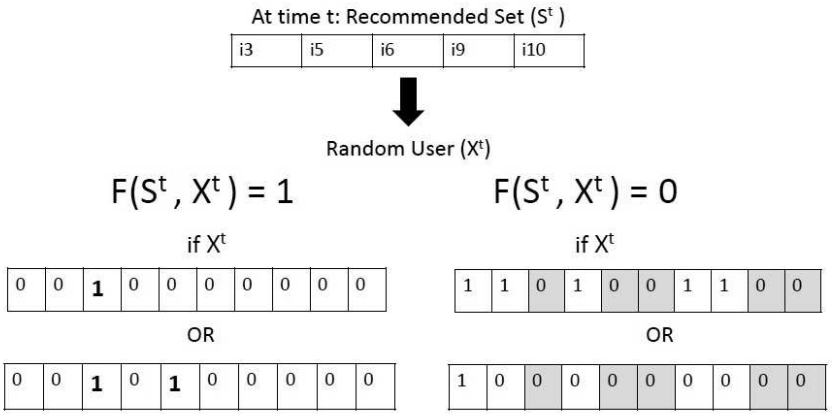

Fig. 1 Payoff for a recommendation set according to users click response

Because plugging-in an algorithm to test its performance to a real-time data stream would take quite a long time, all of the online recommendation experiments are done with a huge dataset that has been collected. We choose the method described in [12] as follows. With given such 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 independent and identically distributed from an unknown distribution D from all user relevance vectors. Then the recommendation system presents a set of k items S^t without observing X^t. We also adopted the definition of set relevance function F used by the paper to follow the convention:

Definition 2 *$F(X^t, S^t)$ is a submodular set function [21] stands for the payoff of showing S^t to user with relevance 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 [12], 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 [12], our target is to minimize abandonment, so we need to maximize click through rate [11], which is:

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

3 Related Work

In the existing work, two different approaches has 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) [17] 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 an user from the recommendation set to represent the group of similar type of user. But as their algorithm strives to produce an ordered set, the learning is slow. 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 [12]. 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) [12] assumed independence between items and used Probability Ranking Principle (PRP) [19] as a greedy method to select items to recommend. PRP give equal credit to all the item a user select 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 items are 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.

Apart form these, most recent graph based recommendation system has been proposed by [13] where they 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 admit 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. So 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.

4 The Baseline Algorithms Used

The underlining approach we are taking is Multi-Armed Bandit Algorithms [22]. 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.

For an online recommendation system, at each time t, from n available items (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 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.

The most popular stochastic bandit algorithm, UCB1 [2] has been used as our baseline 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.

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.

Suppose, there is only one item to select from two: $i, j \in A$ 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 arm i has been tried more often than arm 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.

Provided that a UCB1 algorithm have tried enough of each items to be reasonably confident, it rules out the chance that a selected item would be suboptimal 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 algorithm also addresses the dependencies among items. In fact, our algorithm leverages the dependencies to maximize the coverage of user preferences.

5 Our Past Approach

Algorithm 1 shows our past approach we used [18]. 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 [12], 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 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 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.

6 Shortcomings of the Past Approach

One problem with our past approach is that the unequal rewarding scheme still can exclude many user types with smaller populations (refer them as 'minority user-types.') The basic idea behind our past approach [18] follows Probability Ranking Principle (PRP) [19] which allows to rank items in decreasing order of relevance probability without considering the correlations between them as in [12]. But unlike [12], unequal rewarding for bandits on selecting an item

Algorithm 1 Non graph-based Approach

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

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 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.

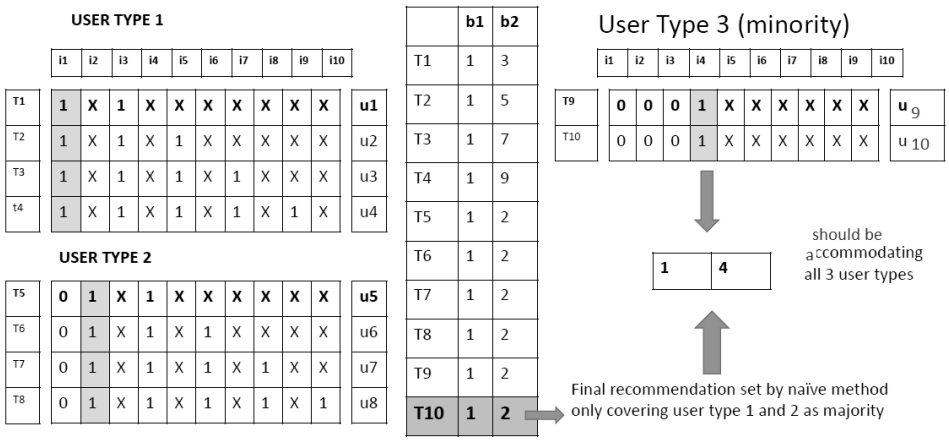

Fig. 2 Limitation of Non Graph Based method where Ti labels the 2 item recommendation set at i^{th} time step coming from 2 bandits b1 and b2 and u_i denotes the user vector it is recommended to

– 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 long trials. 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.

7 Our New Method: Graph Based UCB1 Bandits

As we encounter the above mentioned issue with our past approach [18], 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 easy 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 3 *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 4 *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.

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

7.1 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 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.
- 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.*

Algorithm 2 shows the initialization of the graph and Algorithm 3 shows how the graph is evolving dynamically.

Algorithm 3 BRecommend_kItems

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

7.2 Assignment of Node Weight

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

7.3 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

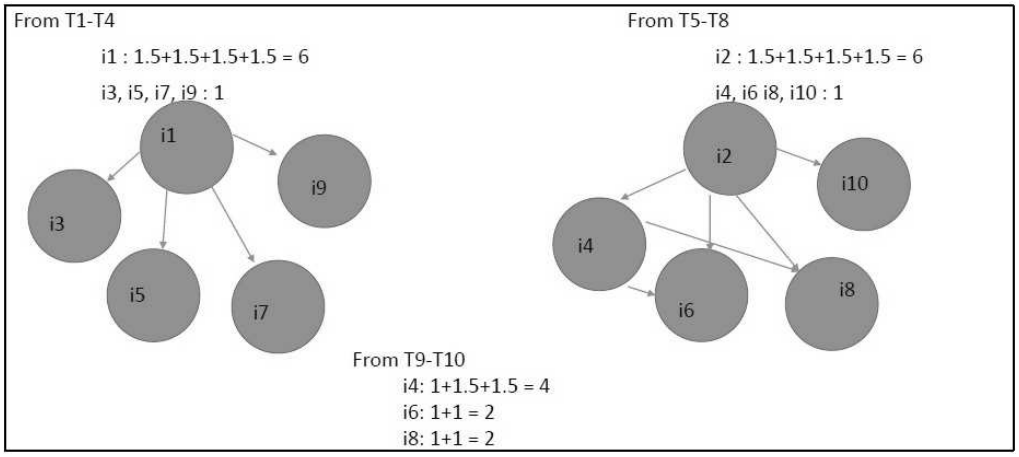

Fig. 3 Emergence of item 4 with increasing weight as a potential replacement of item 2 in the graph based method to accommodate user type 3 as in the illustrative example mentioned in Figure 2

set while facilitating diverse user instances according to our novel rewarding mechanism. This procedure is illustrated in Algorithm 4.

Algorithm 4 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 $Adjust\_Reward(B, k, R^t, X^t)$

7.4 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 5 is developed on this idea.

Algorithm 5 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

8 Empirical Evaluation

In this section we show the performance boost of the graph-based method over the existing ones.

8.1 Performance of Initial Approach

We used Movielens1000 [14] data set for our experiments. The data set consists of 943 users rating on 1682 movies (where users rated at least 20 of them). The real valued ratings has been converted to binary (relevant or not) by using a threshold of exceeding θ which is set to 2 for our experiment as rating ranges between 1 to 5. 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. We compared our past approach with RBA and IBA and average simulation result is presented in Fig 4. The points along the lines in the graph show the results reached at 10K, 50K, 100K, 200K, 500K, 510K and 550K time steps.

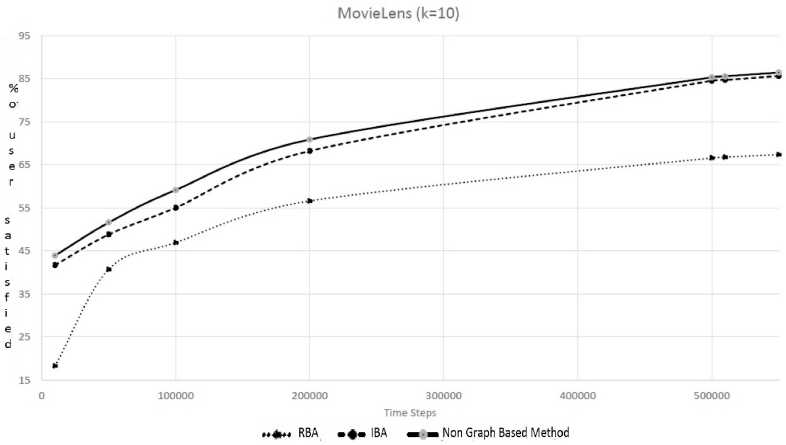

Fig. 4 Performance comparison of recommending 10 movies out of 1682 from MovieLens dataset of all unfiltered 943 users using RBA, IBA and our past (Non Graph Based) approach

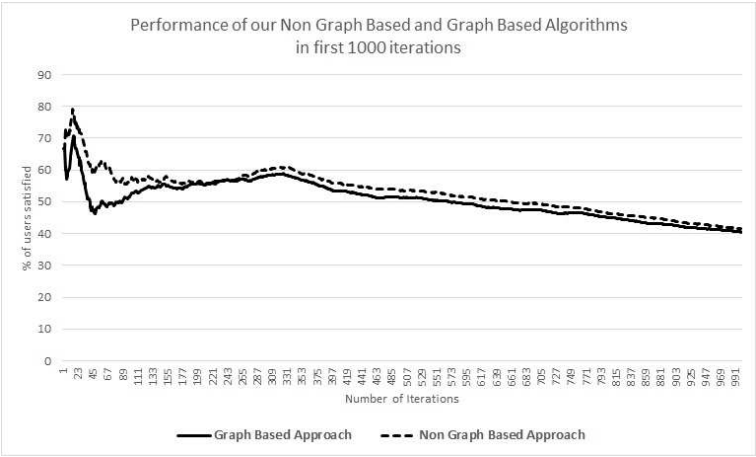

Fig. 5 Average Performance comparison of first 1000 iteration of our Non Graph Based and Graph Based Approach on unfiltered users w/o discount on weight

8.2 Performance of Graph Based Approach

In this section we describe 3 different experiments we conducted with the new graph based bandit algorithm. First, we ran our Graph Based Bandit algorithm on our data set without any discount on weight and found it outperforms our past approach after a considerable amount of user vector is presented to the system. Fig 5 and 6 shows the beginning and end trends of our simulation.

By analyzing the user vectors, we found that the data set has a few user instances who selected unusually many more movies than others. Fig 7 exhibits the user click behavior from the MovieLens1000 data set. To normalize

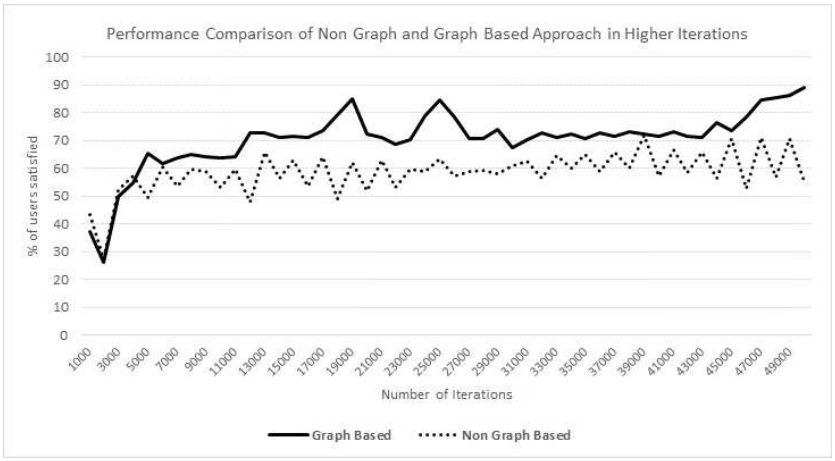

Fig. 6 Average performance comparison of 50000 iteration of our Initial and Graph Based Approach on unfiltered users w/o discount factor

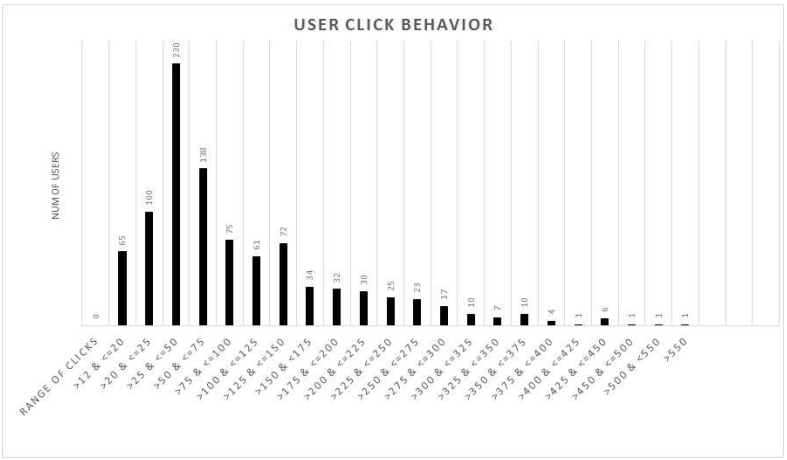

Fig. 7 Click Behavior of User Instances in Movielens Dataset

this skewness, we filtered out the user vectors to take account of those user instances who has selected at least as many movies between 13 to 150. Number of these users makes 80% of the total user instances which results in 741 user instances selecting from 1456 movies. Without the loss of generality, this filtered dataset still captures a diverse user types, yet rules out users with too many movies associated. So in this experiment, we used our non discounted weighting mechanism on our filtered data set. As expected, this shows even faster user coverage. The result of this experiment is shown in Fig 8. Finally, we introduce 2 types of discount function at the end of each epoch where epoch length is fixed as 10K time steps.

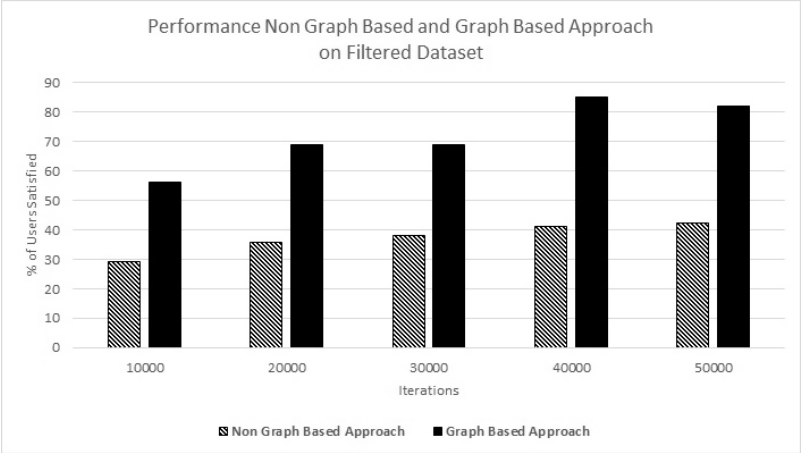

Fig. 8 Average performance comparison of up to 50000 iteration of our Initial and Graph Based Approach on filtered users w/o discount factor

– Linear Discount: In this scheme we scale the weight after each epoch by following function:

$$f(t) = f(t-1) + \epsilon$$

where t denotes the time step. In our experiment, $f(0)$ is set to 0.2 and ϵ is set to 0.2. We run our algorithm upto 50K iteration with $\frac{50K}{10K} = 5$ epochs.

– Logarithmic Discount: According to this technique, we impose a logarithmic smoothing of

$$\frac{1}{(1 - \log(t+\tau)/T))))}$$

where τ is the epoch length and T is the total time steps of the simulation.

We compare the impact of the linear discount with the logarithmic one for the filtered data set in Fig 9. It can be observed that in long run both the linear and logarithmic discounting scheme attains almost same performance as non discounted method because we are randomly sampling from a static pool of data. As more emphasis is given towards recent samples of user instances this discount can capture the taste of the diverse users appeared in later time.

9 Conclusion

Unlike personalized recommendation systems [20] which often use collaborative filtering [7, 8], content based filtering [16] or a hybrid of these two [4], we came up with 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, 12] . But we show that, the correlation between items is an important criteria to identify diversity in terms of

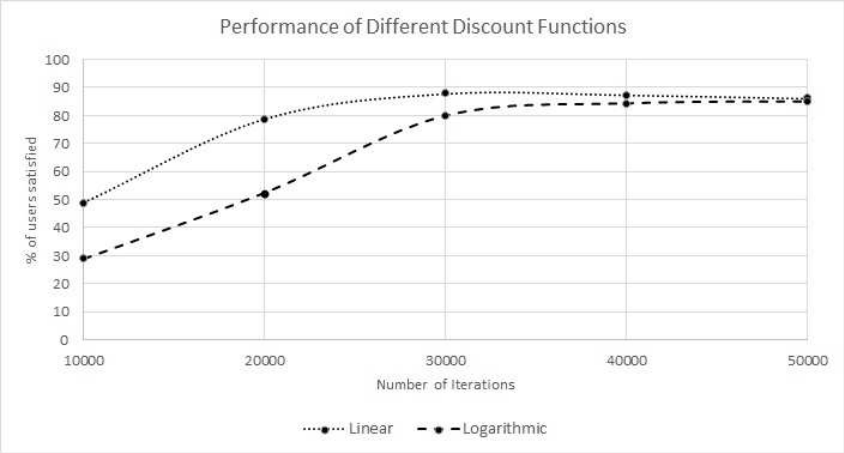

Fig. 9 Average performance comparison of up to 50000 iteration of Linear and logarithmic discount on our Graph Based Approach on filtered users

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.

References

1. Agrawal S (2011) Optimization under uncertainty: Bounding the correlation gap. PhD thesis, URL `http://research.microsoft.com/apps/pubs/default.aspx?id=200425`
2. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. Mach Learn 47(2-3):235–256, DOI 10.1023/A:1013689704352, URL `http://dx.doi.org/10.1023/A:1013689704352`
3. Ausiello G, Boria N, Giannakos A, Lucarelli G, Paschos VT (2010) Online maximum k-coverage
4. Burke R (2007) The adaptive web. Springer-Verlag, Berlin, Heidelberg, chap Hybrid Web Recommender Systems, pp 377–408, URL `http://dl.acm.org/citation.cfm?id=1768197.1768211`
5. Cohen R, Katzir L (2008) The generalized maximum coverage problem. Inf Process Lett 108(1):15–22, DOI 10.1016/j.ipl.2008.03.017, URL `http://dx.doi.org/10.1016/j.ipl.2008.03.017`
6. 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, ACM, New York, NY, USA, CIKM '12, pp

1025–1034, DOI 10.1145/2396761.2398399, URL http://doi.acm.org/10.1145/2396761.2398399
7. Ekstrand MD, Riedl JT, Konstan JA (2011) Collaborative filtering recommender systems. Found Trends Hum-Comput Interact 4(2):81–173, DOI 10.1561/1100000009, URL http://dx.doi.org/10.1561/1100000009
8. Goldberg K, Roeder T, Gupta D, Perkins C (2001) Eigentaste: A constant time collaborative filtering algorithm. Inf Retr 4(2):133–151, DOI 10.1023/A:1011419012209, URL http://dx.doi.org/10.1023/A:1011419012209
9. Hochbaum DS (1997) Approximation algorithms for np-hard problems. PWS Publishing Co., Boston, MA, USA, chap Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems, pp 94–143, URL http://dl.acm.org/citation.cfm?id=241938.241941
10. Hochbaum DS, Pathria A (1998) Analysis of the greedy approach in problems of maximum k-coverage. Naval Research Logistics (NRL) 45(6):615–627, DOI 10.1002/(SICI)1520-6750(199809)45:6⟨615::AID-NAV5⟩3.0.CO;2-5, URL http://dx.doi.org/10.1002/(SICI)1520-6750(199809)45:6<615::AID-NAV5>3.0.CO;2-5
11. Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, KDD '02, pp 133–142, DOI 10.1145/775047.775067, URL http://doi.acm.org/10.1145/775047.775067
12. Kohli P, Salek M, Stoddard G (2013) A fast bandit algorithm for recommendation to users with heterogenous tastes. In: desJardins M, Littman ML (eds) AAAI, AAAI Press, URL http://dblp.uni-trier.de/db/conf/aaai/aaai2013.html#KohliSS13
13. 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, DOI 10.1016/j.eswa.2014.07.024, URL http://dx.doi.org/10.1016/j.eswa.2014.07.024
14. MOVIELENS-DATA (as of 2003) MovieLens dataset, http://www.grouplens.org/data/, URL http://www.grouplens.org/data/
15. Park W, Oh JC, Blowers MK, Wolf MB (2006) An open-set speaker identification system using genetic learning classifier system. In: Cattolico M (ed) GECCO, ACM, pp 1597–1598, URL http://dblp.uni-trier.de/db/conf/gecco/gecco2006.html#ParkOBW06
16. Pazzani MJ, Billsus D (2007) Content-based recommendation systems. In: THE ADAPTIVE WEB: METHODS AND STRATEGIES OF WEB PERSONALIZATION. VOLUME 4321 OF LECTURE NOTES IN COMPUTER SCIENCE, Springer-Verlag, pp 325–341
17. Radlinski F, Kleinberg R, Joachims T (2008) Learning diverse rankings with multi-armed bandits. In: Proceedings of the 25th International Conference on Machine Learning, ACM, New York, NY, USA, ICML '08, pp 784–791, DOI 10.1145/1390156.1390255, URL http://doi.acm.org/10.

1145/1390156.1390255
18. 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, Springer, Switzerland, IEA/AIE'15, pp 361–370
19. Robertson SE (1997) Readings in information retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, chap The Probability Ranking Principle in IR, pp 281–286, URL http://dl.acm.org/citation.cfm?id=275537.275701
20. Shani G, Gunawardana A (2011) Evaluating recommendation systems. Recommender Systems Handbook pp 257–297, 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
21. Sviridenko M (2004) A note on maximizing a submodular set function subject to a knapsack constraint. Operations Research Letters 32(1):41 – 43, DOI http://dx.doi.org/10.1016/S0167-6377(03)00062-2, URL http://www.sciencedirect.com/science/article/pii/S0167637703000622
22. Vermorel J, Mohri M (2005) Multi-armed bandit algorithms and empirical evaluation. In: In European Conference on Machine Learning, Springer, pp 437–448