2014 IEEE International Conference on Data Mining Workshop

Community Detection on Large Graph Datasets for Recommender Systems

Rohit Parimi
CIS Department, Kansas State University
Email: rohitp@ksu.edu

Doina Caragea
CIS Department, Kansas State University
Email: dcaragea@ksu.edu

***Abstract*—The explosion of content on World Wide Web (WWW) means that consumers are presented with a wide variety of items to choose from (items that concur with their taste and requirements). The generation of personalized consumer recommendations has become a crucial functionality for many web applications, yet a challenging task, given the scale and nature of the data. One popular solution to creating personalized item suggestions to users is recommender systems. In this work, we propose an approach that integrates community detection with neighborhood-based recommender systems, specifically, the Adsorption algorithm, for recommending items using implicit user preferences. Network communities represent a principled way of organizing real-world networks into densely connected clusters of nodes. We believe that these dense clusters identified by the community detection algorithm will be helpful to construct user neighborhoods for Adsorption algorithm for recommending collaborators and books to users. Through comprehensive experimental evaluations on the *DBLP* co-author dataset and *BookCrossing* dataset, the proposed approach of integrating community detection with the Adsorption algorithm is shown to deliver good performance.**

I. Introduction

Advances in technology and the growth of the Internet over the past decade has resulted in an exponential growth of online content. The available online information is invaluable, however, the sheer volume of information (information overload) and the ambiguity associated with its representation due to multiple formats has limited its usage. Recommender systems have been proposed to address the information overload problem by filtering the relevant information and suggesting items of interest to users, for example, "interesting movies to watch", "books to read", "people to collaborate with", etc.

Typically, recommender systems use either the content of the items (content-based techniques) or the collaboration between users or items (collaborative filtering techniques) to estimate the preference of a user for an item. The content-based techniques profile each item by representing them with a set of features and associate users with matching items based on the profile similarity. For example, to recommend movies to a user, one could represent movies using features genre, actors etc., and compute similarity between movies using the features. A user can be recommended movies similar to the movies highly preferred by him. However, content-based techniques require gathering additional information about users and/or items to generate recommendations.

In this work, we focus on a collaborative filtering (CF) technique that relies only on past user history, specifically, user's implicit behavior corresponding to clicks, likes etc., to provide automated item suggestions to users. One advantage of CF techniques in addition to avoiding the need for collecting additional data is that these techniques are domain independent. Unlike content-based techniques which require domain knowledge to extract features for user and/or item profiles, CF techniques can be applied to any data domain with information about user-item interactions [15], [16], [17], [18], [19], [20].

CF techniques can be further classified into neighborhood-based techniques and model-based techniques. Neighborhood-based techniques predict the preference of a user-item pair based on the preferences given to that item by neighbor users, whereas, model-based techniques learn a predictive model from the user-item preference information to predict unknown user-item preferences. Although both approaches have been extensively used for recommender systems, our focus is restricted to using a neighborhood-based approach because of the following merits: ability to explain recommendations to a user, less parameters to tune, and their intuitiveness [17].

The neighborhood-based techniques have three important characteristics that can have significant impact on the accuracy of a recommender system: *a*) normalization of data, *b*) computation of similarity weights, and *c*) selection of neighbors [17], [18], [20]. Since our work uses implicit user history, data normalization does not apply, however, based on our experience, the other two characteristics namely, computing similarity score and neighborhood selection have proved pivotal in improving the recommendation accuracy. Accordingly, in this work, we study a custom similarity weighting scheme and use it in conjunction with community detection algorithm to select nearest neighbors. The selected neighbors are used to propagate preference information via random-walks on a user graph to recommend co-authors and books.

In general, the n-nearest neighbors u of a user v can be selected based on the similarity between users v and u. Several similarity functions (e.g. Pearson correlation coefficient, cosine similarity) have been proposed in the literature to compute user-user similarity and use the similarity to construct n-nearest neighbors for explicit-feedback datasets [9], [10] , [20]. For implicit feedback datasets, one can use a simple similarity function such as normalized common item count or more complex similarity functions such as the ones used for explicit feedback datasets to select the nearest neighbors. However, selecting neighbors using the aforementioned similarity metrics does not consider the underlying community structure existing between the users. Communities, also called *clusters*, are groups of vertices in a graph which have strong connections with other vertices in the same group compared to vertices

978-1-4799-4274-9/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDMW.2014.159

in other groups. The vertices in a community probably share common properties and play similar roles in the graph [12]. *We hypothesize that the information about communities in the user-user graph can be important for constructing the n-nearest neighbors and consequently for generating recommendations*. To this end, we explore community detection based on modularity optimization proposed by Blondel et al. [11] to construct the n-nearest neighbors. The selected nearest neighbors are used with Adsorption algorithm, a random-walk based label propagation approach, for recommending co-authors and books. To summarize, our main contributions are:

1) We study the application of modularity based community detection for selecting n-nearest neighbors.
   - We also analyze the variation in recommendation accuracy when the sparsity of the graph given as input to the community detection algorithm is varied.
2) We evaluate the effectiveness of community detection techniques for constructing user neighborhoods based on the performance of Adsorption algorithm on two real-world large graph datasets.
3) We investigate the performance of Adsorption algorithm when the neighborhood size is varied.

The remainder of the paper is organized as follows: we review related work in Section II and introduce community detection in Section III. In Section IV, we describe the Adsorption algorithm and community detection based neighborhood selection. Data and experiments are explained in Section V. We present experimental results in Section VI and conclude with future research directions in Section VII.

II. Related Work

Given the large number of works on recommender systems and community detection algorithms, we only review works most relevant to our research.

Community Detection In Graphs: The interest in community detection techniques is ever increasing because many networks in real-world, e.g., WWW, social networks, and citation networks display community structure. Detecting communities from networks is a well-known problem in research community and several approaches have been proposed to address this problem. More information on community detection approaches can be found in [11] [12], [13], [14].

In the recent past, many researchers addressed the item recommendation problem by clustering users into groups. For example, Ying et al. [8] proposed a preference aware community detection approach that groups users based on similarity of their rating and that of their social relation and used the communities to generate recommendations. Sahebi et al. [7] addressed the cold-start problem in recommender systems by identifying user communities in one network and using these communities to generate recommendations in other networks. In their work, the authors assume that the users of the system form a heterogeneous network. Sarwar et al. [6] addressed the performance issues of neighborhood-based approaches by scaling up the neighborhood formation process through the use of clustering. In their work, they used k-means clustering to generate neighborhood for users and applied collaborative filtering on the neighborhood to predict item ratings. Our research differs from the above works as follows: the datasets we use have implicit feedback rather than an explicit rating information, we have a homogeneous network as opposed to a heterogeneous network, and we use community detection approach as opposed to k-means to identify user clusters.

Collaborative Filtering: CF is a popular and widely used approach for recommender systems regardless of the application domain. Among the CF techniques, neighborhood-based approaches are a popular implementation choice for most real-world applications [17], [18], [19], [20]. Well-known neighborhood-based algorithms and sample applications include: user-based and item-based approaches [19] used for applications such as predicting ratings, Adsorption [1], [2] used for applications such as recommending YouTube videos, classification, and sentiment analysis of text data.

Adsorption: The Adsorption algorithm is a random-walk based approach and works by propagating preference information through graphs. The algorithm was first proposed by Baluja et al. [1] to recommend YouTube videos and later was successfully used for tasks such as classification, and sentiment analysis [2], among others. The intuition behind the algorithm is that a user's preference for items is likely to match the items commonly preferred by similar users.

III. Community Discovery in Graphs

Community detection from networks have many practical applications since many datasets can be represented as graphs. Algorithms for community detection aim to find clusters in a graph such that the amount of interaction within a cluster is more than the interaction outside it. For example, in a collaboration network, communities might reveal groups related to similar research; social network communities might reveal real social groupings. In this section, we will first explain basics of community detection based on modularity and later explain the community detection algorithm used in this work.

A. Modularity Optimization

The work by Newman et al. [13] is one of the first works to detect communities in graphs by optimizing modularity. Modularity measures the quality of partitions from a community detection algorithm; the modularity of a partition is a value between -1 and 1 and measures the density of edges within a community as compared to edges between different communities [12], [13]. In Equation (1) we show how modularity is computed for a community structure:

$$Q = \frac{1}{2m} \sum_{i,j}^{n} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta\left(c_i, c_j\right) \tag{1}$$

In this above equation, A_{ij} corresponds to the weight on the edge between nodes i and j in the network, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges through node i, c_i is the community to which the node i is assigned to, the δ-function yields 1 if nodes i and j belong to the same community and 0 otherwise, and $m = \frac{1}{2} \sum_{ij} A_{ij}$.

Modularity is the most used and best known quality function for community detection in networks. Most algorithms for community detection work under the assumption that

high values for modularity indicate good partitions and thus maximize Q [12]. Given the large real-world networks (with millions of nodes and edges) and numerous possibilities to partition a graph, an exhaustive optimization of Q becomes intractable. Also, it has been proved recently by Brandes et al. [14] that modularity optimization is NP-complete, i.e., there probably does not exist a polynomial time algorithm that can find a global maximum for Q. Hence, algorithms focused on modularity optimization are based on approximate optimization methods such as greedy algorithms, simulated annealing. In this work, we use the Louvain method proposed by Blondel et al. [11] to identify communities from the *DBLP* co-author network and the *BookCrossing* network. The approach is described below:

Louvain Method for Community Detection: The louvain method proposed by Blondel et al. [11] is a heuristic method based on modularity optimization for community detection. This method can be applied to large weighted graphs and is known to outperform many other community detection algorithms in terms of computational time. The louvain method is a two phase iterative process. In the first phase, all nodes in the graph are assigned to different communities. At each node i, the algorithm computes the gain in modularity ΔQ, given by Equation (2), when node i is removed from its community and is placed in the community of a neighbor node j. The node i is assigned the community for which the gain in modularity is positive and highest. At the end of this phase, the algorithm identified the first level partitions. In the second phase, a new network is constructed from the communities identified in the first phase; the nodes in the new network are the communities identified in the first phase. Two nodes in the new network are connected if there is at-least one edge between nodes in the corresponding communities from the first phase. These two steps are repeated iteratively until there is no change in modularity yielding hierarchical levels of the original network.

$$\Delta Q = \left[\frac{\sum w_{in} + k_{i,in}}{2m} - \left(\frac{\sum w_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum w_{in}}{2m} - \left(\frac{\sum w_{tot}}{2m}\right)^2 - \frac{k_i}{2w}\right] \tag{2}$$

In the above equation, $\sum w_{in}$ is the sum of the weights of links inside the community C where vertex i is places, w_{tot} is the sum of the weights of the links incident to nodes in C and $k_{i,in}$ is the sum of the weights of the links from i to nodes in C and m is the sum of the weights of all the links in the network. The reader is referred to the work in [11] for more information on the Louvian method.

IV. Adsorption Algorithm

The Adsorption algorithm proposed by Baluja et al. [1] is a general semi-supervised framework for classification. The algorithm can be expressed using three different, yet equivalent, views, namely 'Averaging View,' 'Random-walk View,' and 'Adsorption via Linear Systems'. In this paper, we explain it using the 'Random-walk view'.

A. Adsorption via Random-walk:

We are given an undirected graph $G = (V, E, w)$, where V denotes the set of vertices (users), E denotes the set of edges, and $w_{uv} \epsilon \mathbb{R}_+$ is the weight associated with the edge between two users (u, v), and expresses the similarity between them. Let the set of all possible labels be denoted by $L = \{1 \cdots m\}$ and let m be the size of the set L, i.e. $|L|$. Each user v in the graph is associated with two row-vectors Y_v, $\widehat{Y}_v \epsilon \mathbb{R}_+^m$. Vector Y_v denotes the initial label distribution for user v, i.e., Y_{vx} represents the probability that user v prefers label x. Similarly, the vector $\widehat{Y}_v$ indicates predictions made by the algorithm for user v and encodes a distribution over the m labels. The higher the value of Y_{vx}, the stronger the belief that user v has a high preference for label x a priori. Likewise, the higher the value of $\widehat{Y}_{vy}$, the stronger the a posteriori belief that y corresponds to a good recommendation for user v, $x \neq y$.

In order to generate recommendations, the algorithm uses a controlled random-walk over the graph. At each node, the algorithm is presented with three options: stop and return, in other words *inject* the initial label distribution Y_v of the node, *terminate* or abandon the walk and return an all-zero vector, $\mathbf{0}_m$, or *continue* the walk to neighbor node u chosen according to the probability $Pr[u|v]$, given by Equation (3), and emit predicted labels $\widehat{Y}_u$, given by Equation (4). The injection, termination and continuation steps have probabilities p_{inj}, p_{term}, and p_{cont}, respectively, and the sum of these probabilities should be 1. For a particular problem, these probability values can be selected using cross-validation.

The probability distribution over the neighbors u of a user v is estimated using the following equation:

$$Pr[u|v] = \begin{cases} \dfrac{w_{uv}}{\sum\limits_{u:(u,v)\in E} w_{uv}}, & \text{if } (u,v) \in E \\ 0, & otherwise \end{cases} \tag{3}$$

Furthermore, new labels $\widehat{Y}_v$ for a user v can be computed using the following equation:

$$\widehat{\mathbf{Y}}_v = p_{inj} \times \mathbf{Y}_v + p_{cont} \times \sum_{u:(u,v)\in E} Pr[u|v] \widehat{\mathbf{Y}}_u + p_{term} \times \mathbf{0}_m \tag{4}$$

The random-walk process is initiated at every user vertex in the graph and is repeated until the algorithm converges. Convergence criteria can be a fixed number of iterations or until the change in $\widehat{Y}_v$ is below a threshold.

B. Neighborhood Generation

Theoretically, in the above algorithm, from a user node v we can *continue* the random-walk to all the neighbor users u for which $Pr[u|v] > 0$. However, literature on neighborhood-based approaches suggest that using all neighbors, as opposed to only nearest neighbors, does not always result in great improvements in recommendation accuracy [20]. Furthermore, from an application perspective, using all the neighbors to compute recommendations is often computationally expensive

given the large number of neighbors for a user and large datasets in general. Thus, a popular solution to the problem of large neighborhoods is to only use the n nearest neighbors for each user [3], [19], [20]. For a user, the nearest neighbors can be computed using some similarity metric. Different recommendation applications can use different similarity metrics, depending on the best way to capture similarity in a particular domain [3], [20]. Similar to this line of work in neighborhood-based approaches, we restrict the random-walk from a user node to only the nearest neighbors.

In general, the n-nearest neighbors u of a user v can be selected based on the strength of the edge (similarity) between the users v and u in the graph G. For implicit feedback datasets, the most straightforward way to define edge weights between two users is based on the number of items preferred by both the users. The common item count can be normalized to account for variation in the number of items preferred by each user. However, this approach (like other approaches for similarity) does not take into account the underlying community structure between users in G. Network communities correspond to a principled way of organizing vertices in a graph into densely connected clusters. *We hypothesize that such densely connected clusters of users will result in better neighborhoods compared to the neighborhoods generated in a straightforward way for some applications*. To check our hypothesis, we compare the straightforward approach to select n-nearest neighbors, our baseline or *Variant 1*, with the n-nearest neighbors computed using community detection, *Variant 2*. We describe the two variants more precisely below:

1) **Variant 1 (Baseline):** The weight between two users u and v is defined as the number of preferred items common between the two users u and v normalized by the sum of total number of items each user preferred, i.e., $w_{uv} = \frac{\#common\ items(u,v)}{\#items(u) + \#items(v)}$, $w_{uv} \epsilon [0, 0.5]$. Intuitively, two users who generally prefer many items and have many preferred items in common, should be deemed less similar than two users that have many items in common, but do not prefer many items. Likewise, two users who prefer less items in general, and have a significant number of common preferred items should be considered similar and given a higher weight than the weight between users with a similar number of common items but prefer many items. We select n-nearest neighbors according to the weights.
2) **Variant 2 (Community Detection):** In this variant, we cluster users using a community detection algorithm proposed in [11] and use the clusters to compute neighborhoods. Following are the steps involved in generating the n nearest neighbors for this variant:
   a) Compute the weight between every pair of users (u, v) in the dataset according to the weight equation for variant 1, i.e., $w_{uv} = \frac{\#common\ items(u,v)}{\#items(u) + \#items(v)}$, $w_{uv} \epsilon [0, 0.5]$. To be able to use the weights with community detection algorithm, the range for w is changed from $[0, 0.5]$ to $[0, 1]$. This is done by normalizing the weights between every user pair (u, v) by the highest weight value for the user u.
   b) Extract every user pair (u, v) for which the computed weight w_{uv} is above a threshold t ($0 \leq t \leq 1$). The threshold is used to control the number of edges in the user-user graph. The number of edges will be maximum when $t = 0$ (there will be an edge between every user pair (u, v) for which $w_{uv} > 0$) and the number of edges decreases as t increases.
   c) Construct a user-user graph from the user pairs extracted in the previous step. The weight w_{uv} represents the weight on the edge between users u and v.
   d) Apply the community detection algorithm proposed in [11] on the graph computed in previous step.
   e) The community detection algorithm outputs a list of users for each identified community. The number of communities identified by the algorithm varies from dataset to dataset and is determined at runtime. To construct the n-nearest neighbors for a user v, we select top n users u from the community of user v according to weights w_{uv}. For example, if $u1$, $u2$, $u3$, $u4$, $u5$ are the five nearest neighbors for user v, then users $u1$, $u2$, $u3$, $u4$, $u5$ should belong to the same community as user v and $w_{u1v} \geq w_{u2v} \geq w_{u3v} \geq w_{u4v} \geq w_{u5v}$.

We should note that in the above variants, it is possible that many of the neighbors of a user have the same weight. To resolve ties, we randomly select n users that have the same weights to fill in the n-nearest neighbor positions, following the constraints of the variant in use.

V. Experimental Design

A. Dataset Description and Preprocessing

We have used two datasets to study the usefulness of community detection to recommend collaborators and books to users. The first dataset is the *DBLP* co-author dataset [4]. The dataset has information about user-user collaborations between years 1940 and 2013 and consists of approximately 1.3M users (who can also be seen as items) and 18.9M collaboration records with four columns, specifically, *From_id*, *To_id*, *weight*, and *timestamp*. We have used a subset of this dataset with collaborations between the years 1992 and 2012. This subset has approximately 1M users and 18.3M collaboration records. The second dataset we have used is the *BookCrossing* dataset [5] which has information about books read by members of the *BookCrossing* community. The dataset consists of approximately 100K users, 340K books and 1.1M user-book preferences with three columns, specifically, *user*, *book*, *timesRead*.

Given that timestamps are available for the *DBLP* dataset, we have used the timestamps to generate training and test datasets. Specifically, we have divided the data into seven splits. Each of these seven splits has roughly data from three years. Using these seven splits, we have generated five subsets of training and test data. Table I depicts how the five subsets are generated from the splits. Since timestamp information is not available for the *BookCrossing* dataset, we employed a different approach to create the training and test data for this dataset. We removed users who interacted with less than four books (to ensure that every user had at-least one preference in each split) and divided the data into four splits with roughly 25% of preferences in each split. We use three splits as training data and the fourth split as test data and repeat this four times to ensure that every split is used as test data. Table II depicts how the four subsets are generated from the splits.

TABLE I. TRAINING AND TEST SUBSETS BASED ON 7 SPLITS (DENOTED, $0, \cdots, 6$) OF THE *DBLP* CO-AUTHOR DATASET; $split(...)$ INDICATES THE SPLITS USED TO CREATE A PARTICULAR SUBSET, E.G., SPLIT(0, 1) GIVES A SUBSET THAT IS THE UNION OF SPLITS 0 AND 1.

Subset	Training	Test
0	split(0, 1)	split(2)
1	split(1, 2)	split(3)
2	split(2, 3)	split(4)
3	split(3, 4)	split(5)
4	split(4, 5)	split(6)

TABLE II. TRAINING AND TEST SUBSETS BASED ON 4 SPLITS (DENOTED, $0, \cdots, 3$) OF THE *BookCrossing* DATASET; $split(...)$ INDICATES THE SPLITS USED TO CREATE A PARTICULAR SUBSET, E.G., SPLIT(0, 1, 2) GIVES A SUBSET THAT IS THE UNION OF SPLITS 0, 1, AND 2.

Subset	Training	Test
0	split(0, 1, 2)	split(3)
1	split(1, 2, 3)	split(0)
2	split(2, 3, 0)	split(1)
3	split(3, 0, 1)	split(2)

Once the training and the test sets for the two datasets are created as described above, we perform two preprocessing steps on the test data. Firstly, we remove from the test set all users that do not appear in the corresponding training set, because, if a user is not present in the training set, there is no way to make recommendations for that user. Secondly, given that we want to recommend new unseen items to users, we remove from the test set of a user items that have already been seen by him in the training set. We should not that this is a standard way of creating test datasets for evaluating recommender systems and is widely used in literature [1], [19], [21].

B. Experiments

The experiments designed in this work focused on addressing the following research questions:

1) **How informative are the neighbors constructed from the community detection algorithm for the task of recommending collaborators and books to users?** Our hypothesis is that the performance of Adsorption algorithm using neighborhood constructed from community detection will be better than the neighborhood computed in a straightforward way. This is because, with community detection, we use additional information about community structure in the user-user graph to select the n-nearest neighbors. We verify our hypothesis by comparing the recommendation accuracy form Adsorption algorithm with neighborhood constructed using community detection algorithm with the recommendation accuracy from the Adsorption algorithm with neighborhood constructed using baseline variant, for both the datasets.
2) **How does the performance of Adsorption algorithm vary with the threshold t on the edge weights of user-user graph G (t is used to control the number of edges in G, the input to community detection algorithm)?** By varying t, we aim to control the amount of information encoded in the user-user graph G. Our hypothesis is that the recommendation accuracy increases as t is increased, reaches a maximum for some value of t and then decreases when t is further increased. This is because of the difficulty associated with finding meaningful clusters from dense graphs (i.e when t is close to 0). As we increase t, the density of the graph G decreases which makes it easier to find meaningful clusters. However, when t gets close to 1, graph G becomes very spare and the clusters identified will only have users with high edge weights and does not capture novel information.
3) **What is the impact of the number of neighbors n on the recommendation accuracy?** We expect to see a similar trend in the results as we observed when the threshold t is varied. We believe that the results will improve as the number of users n is increased. However, when n is increased beyond a threshold, i.e., when we consider too many users as neighbors for a user, the recommendations become less personalized decreasing the performance of the algorithm.

C. Evaluation Methodology

The Adsorption algorithm generates a list of $(item,\ preferenceScore)$ tuples as recommendations for each user in the dataset. From this list, tuples referring to items in the corresponding training set of a user are removed, because our goal is to recommend only new items to users. Next, an ordered list of k tuples, sorted from the highest to the lowest $preferenceScore$ is generated. From this list, we compute Average Precision at k values for each user and use these values to compute Mean Average Precision (MAP) at k.

D. Hyper-parameter Values

Choosing hyper-parameters is one of the crucial stages in designing a recommender system given the fact that the accuracy of algorithms vary with these parameters. The Adsorption algorithm has several hyper-parameters, specifically, p_{inj}, p_{term}, p_{cont}, (random-walk probabilities) and n (number of neighbors). We experimented with three combinations of p_{inj}, p_{term}, and p_{cont}: (0, .7, .3), (0, .85, .15), (0, 1, 0) and fixed their values to (0, .85, .15). This is because, several trail runs on the two datasets indicated negligible difference in the MAP values for the p_{inj}, p_{term}, and p_{cont} combinations considered. We experimented with five values for n: (5, 10, 15, 20, 25) and four values for the weight threshold parameter t: (0, 0.25, 0.5, 0.75). We fixed the number of algorithm recommendations i.e., k to 10 as this is a standard value used in most works on recommender systems.

VI. RESULTS

To understand the usefulness of community detection algorithms to identify informative nearest neighbors, we run experiments with the two variants of Adsorption algorithm on both the datasets. Tables III and IV show the MAP scores from the Adsorption variants for the *DBLP* co-author and *BookCrossing* datasets, respectively.

Analysis of *DBLP* Co-Author Dataset: As expected, the recommendation performance of Adsorption algorithm with neighbors constructed using Variant 2 (community detection) outperforms the performance of the algorithm with neighborhood constructed using Variant 1 (baseline). This result is consistent across all the neighborhood sizes considered ($n = 5, 10, 15, 20, 25$) and for many of the thresholds considered

TABLE III. MAP SCORES FROM ADSORPTION VARIANT 1 (BASELINE) AND VARIANT 2 FOR THE *DBLP* CO-AUTHOR DATASET. THE THRESHOLD (t) ON EDGE WEIGHTS FOR VARIANT 2 IS VARIED FROM 0 TO 0.75 WITH A STEP SIZE OF 0.25. THE NEIGHBORHOOD SIZE (n) IS VARIED FROM 5 TO 25 WITH A STEP SIZE OF 5 AND NUMBER OF RECOMMENDATIONS (k) IS SET TO 10. FOR EACH n VALUE, VARIANT WITH THE BEST MAP SCORE IS HIGHLIGHTED.

#Neighbors		variant 1	variant 2			
			$t = 0$	$t = 0.25$	$t = 0.5$	$t = 0.75$
$n = 5$						
	subset 0	0.0175	0.0185	**0.0188**	0.0172	0.0128
	subset 1	0.0182	0.0200	**0.0201**	0.0190	0.0147
	subset 2	0.0180	0.0196	**0.0199**	0.0182	0.0142
	subset 3	0.0180	0.0195	**0.0201**	0.0187	0.0141
	subset 4	0.0174	0.0185	**0.0193**	0.0182	0.0138
$n = 10$						
	subset 0	0.0164	0.0185	**0.0197**	0.0182	0.0135
	subset 1	0.0172	0.0206	**0.0211**	0.0199	0.0152
	subset 2	0.0167	0.0200	**0.0207**	0.0190	0.0148
	subset 3	0.0168	0.0198	**0.0207**	0.0195	0.0146
	subset 4	0.0165	0.0186	**0.0197**	0.019	0.0144
$n = 15$						
	subset 0	0.0150	0.0184	**0.0197**	0.0185	0.0136
	subset 1	0.0158	0.0204	**0.0212**	0.0201	0.0154
	subset 2	0.0153	0.0198	**0.0207**	0.0192	0.0149
	subset 3	0.0156	0.0198	**0.0209**	0.0198	0.0147
	subset 4	0.0153	0.0183	**0.0197**	0.0192	0.0145
$n = 20$						
	subset 0	0.0143	0.0181	**0.0197**	0.0185	0.0136
	subset 1	0.0148	0.0205	**0.0210**	0.0201	0.0154
	subset 2	0.0143	0.0196	**0.0207**	0.0193	0.0148
	subset 3	0.0147	0.0195	**0.0208**	0.0198	0.0148
	subset 4	0.0143	0.0180	**0.0196**	0.0192	0.0145
$n = 25$						
	subset 0	0.0135	0.0180	**0.0197**	0.0185	0.0136
	subset 1	0.0140	0.0201	**0.0210**	0.0201	0.0154
	subset 2	0.0134	0.0195	**0.0206**	0.0193	0.0149
	subset 3	0.0140	0.0194	**0.0204**	0.0198	0.0147
	subset 4	0.0137	0.0179	**0.0196**	0.0192	0.0145

($t = 0, t = 0.25, t = 0.5$) in Table III. In general, the co-author domain can be seen as a social network. Similar to how users in a social network make friends based on *friend of a friend* relationship, authors in the co-author network often collaborate with other authors who have similar research interests and are acquaintances. Thus, there exists a natural community structure between the authors in the co-author domain. Using Variant 2, we capture this underlying community structure between the authors and use it to select the top n neighbors for authors, thus achieving good MAP score. In the case of Variant 1, the intuition is to pick users (authors) who have some items (publications) in common with the current user (author) yet have small degree. Although, such authors are good neighbors to an author in the co-author domain, we ignore the global information i.e., information about the co-authors of the potential neighbor author, while constructing neighborhood. For example, consider three authors *a*, *b*, and *c*. Assume that *a* authored two papers, one with *b* and one with *c* both related to *Data Mining* domain. Also assume that *b* co-authored with 3 other authors in *Network Security* domain and *c* co-authored with six other authors in *Data Mining* domain. To select a neighbor for *a* from *b* and *c*, variant 1 picks the author who authored the least number of papers among *b* and *c*, in this case, *b*. However, in this example, *c* may be more relevant as neighbor to *a* considering their shared interests and interests of co-authors of *c* in *Data Mining* domain. With Variant 2, we believe that authors *a* and *c* along with the co-authors of *c* will be grouped into a community in which majority of authors will be interested in *Data Mining*.

When varying the threshold (t) on edge weights in the user-user graph G, the results in Table III support our hypothesis that recommendation performance improves as we increase t, reach a maximum for some value of t and then decrease when t is further decreased. In fact, this is true for all the n values considered $(5, 10, 15, 20, 25)$. For example, consider the case when the number of neighbors $n = 5$ in Table III, clearly the MAP scores for all subsets increased when t is changed from 0 to 0.25 and decreased when t is further increased to 0.5 and then to 0.75. One possible explanation for this variation in the MAP scores from Adsorption relates to the difficulty associated with finding meaningful clusters from highly dense and highly sparse graphs. When a graph is densest, i.e., each node is connected to every other node, there will only be one community for the entire graph. Similarly, when a graph

is sparsest, i.e., no node is connected to other nodes, each node will be a separate community. In both these cases, the community detection algorithm will not be helpful as it fails to identify any meaningful communities. However, when the density of the graph is varied, the communities identified by the algorithm vary resulting a change in the MAP score.

To study the effect of the number of neighbors (n) on recommendation performance, we compare the MAP scores of subsets 0 through 4 with the corresponding subsets for different n values, for both the variants. The results in Table III suggest that Adsorption performance decreased when n is varied from 5 to 25 for Variant 1. This suggests that Variant 1 can identify a small number of good neighbors but as n increases, the recommendations become less personalized. Conversely, with Variant 2, the MAP scores increased when n is increased, reached a maximum and decreased as n is further increased for all subsets and t values considered, consistent with our hypothesis. For example, for $t = 0.25$, the MAP scores for all the subsets increased when n is varied from 5 to 10 and from 10 to 15. When the value of n is further increased, a decrease in the MAP scores can be observed in Table III. Given these observations, we suggest that Variant 2 is better suitable for constructing neighborhood for the co-author domain.

Analysis of *BookCrossing* Dataset: When analyzing the results for *BookCrossing* dataset in Table IV, we observe that the best results are achieved by Variant 1 (baseline), which outperforms Variant 2 in almost all cases. This result suggest that for the *BookCrossing* dataset, users with smaller degrees and have some items in common with the current user are more reliable neighbors than the neighbors from community detection. One explanation for this might be the high density of the user-user graph given as input to the community detection algorithm. Generally, it is difficult to identify meaningful clusters from dense graphs and the communities identified from such graphs will be few with large number of users in each community. For this dataset, the average number of edges at each user in the user-user graph is approximately 550, 149, 38, and 12 when t is $0, 0.25, 0.5, 0.75$, respectively, compared to approximately 61, 19, 7, and 3 for the *DBLP* dataset. Our reasoning is reinforced by the fact that the average number of users in each community ($35, 32, 12, 7$ for $t = 0, 0.25, 0.5, 0.75$, respectively) for *BookCrossing* dataset is greater than those in *DBLP* dataset ($10, 7, 6, 4$ for $t = 0, 0.25, 0.5, 0.75$, respectively). Also, for $n = 20$ and 25, Variant 2 with $t = 0.75$ has slightly better MAP scores compared to Variant 1 suggesting that for less denser user-user graph, community detection approaches might be helpful.

As can be seen in Table IV, the MAP score increased when the threshold t on the edge weights is varied from 0.0 to 0.75. The increase in the MAP score justifies our hypothesis that the performance increases when t is increased. However, we did not observe a maximum and then a decrease in the MAP score for the t values we considered, suggesting that the optimal value for t is domain dependent. This problem can be addressed by experimenting with t values greater than 0.75.

Finally, the MAP scores for Variant 1 are same or decreased when the number of neighbors n is varied from 5 to 25. This is consistent with the behavior observed for *DBLP* dataset with Variant 1 and reinforces our claim that Variant 1 can identify small number of good neighbors. In the case of Variant 2, while the MAP scores decreased when the n is increased from 5 to 25 for t values 0.0 and 0.25, a slight increase in MAP scores can be observed for t values 0.5 and 0.75. This is because, at small values for t (dense user-user graph), the communities identified are large and selecting too many users from such large communities makes the recommendations less personalized for the current user. However, at larger values for t (sparser user-user graph), the identified communities are crisp and compact and the neighbors selected from such communities might be considered as good neighbors leading to an improvement in the MAP scores.

Discussion: The small MAP scores for the *DBLP* and *BookCrossing* datasets suggest that for many users, the recommendations computed using Adsorption might not be relevant. However, we should note that this type of numbers are commonly reported in the literature for large-scale recommender systems as ours. For example, for the task of recommending YouTube videos, Baluja et al. [1] reported a 10% precision at a 3% recall. Furthermore, recommending only unknown items to a user as opposed to items that the user is already aware of but might be interested in the future, makes the problem more challenging.

VII. Conclusions and Future Direction

In this work, we explored modularity based community detection to generate user neighborhoods for Adsorption, and conducted several experiments to compare it with another approach for neighborhood generation, for two implicit feedback datasets. Experimental results suggested that Variant 2 (community detection) is better for the *DBLP* dataset, while Variant 1 is better for the *BookCrossing* dataset. This lead us to a conclusion that the choice of algorithm for neighborhood selection is domain dependent. Our results also suggested that community detection is most effective for sparse graphs and the parameter controlling the sparsity of a graph (t) can be chosen experimentally. Finally, with community detection, we were able to improve the recommendation accuracy by considering larger neighborhoods. This makes us believe that with the right values for t and n, community detection approach can generate good quality neighborhoods for Adsorption.

For future work, we would like to compare our approach of generating recommendations using community detection with other approaches such as popularity based recommendations and Matrix Factorization. We would also like to experiment with thresholds (t) closer to 1 and larger neighborhood sizes (n) to verify if user neighborhood constructed from the entire community can improve the recommendation accuracy. Furthermore, we would like to apply our approach to datasets in other domains such as music and extend it to accommodate other similarity functions such as cosine similarity.

[1] Because the subsets for this dataset are created in a way similar to cross-validation, we computed the average MAP score across the four subsets.

References

[1] Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., Aly., M.: Video suggestion and discovery for youtube: taking random walks through the view graph. In WWW, 2008.

[2] Talukdar, P. P., Crammer, K.: New regularized algorithms for transductive learning. In ECML/PKDD (2), 2009.

TABLE IV. MAP SCORES FROM ADSORPTION VARIANT 1 (BASELINE) AND VARIANT 2 FOR THE *BookCrossing* DATASET. THE THRESHOLD (t) ON THE EDGE WEIGHTS FOR VARIANT 2 IS VARIED FROM 0 TO 0.75 WITH A STEP SIZE OF 0.25. THE NEIGHBORHOOD SIZE (n) IS VARIED FROM 5 TO 25 WITH A STEP SIZE OF 5 AND NUMBER OF RECOMMENDATIONS (k) IS SET TO 10. FOR EACH n VALUE, VARIANT WITH THE BEST MAP SCORE IS HIGHLIGHTED. [1]

#Neighbors		variant 1	variant 2			
			$t = 0$	$t = 0.25$	$t = 0.5$	$t = 0.75$
$n = 5$						
	subset 0	0.0078	0.0052	0.0052	0.0057	0.0055
	subset 1	0.0085	0.0064	0.0066	0.0069	0.0074
	subset 2	0.0089	0.0065	0.0069	0.0077	0.0077
	subset 3	0.0088	0.0065	0.0067	0.0067	0.0079
	Average	**0.0085**	0.0062	0.0064	0.0068	0.0071
$n = 10$						
	subset 0	0.0082	0.0053	0.0052	0.0061	0.0061
	subset 1	0.0094	0.0061	0.0066	0.0067	0.0077
	subset 2	0.0083	0.0062	0.0060	0.0077	0.0075
	subset 3	0.0083	0.0061	0.0060	0.0067	0.0079
	Average	**0.0085**	0.0059	0.0060	0.0068	0.0073
$n = 15$						
	subset 0	0.0078	0.0053	0.0051	0.0062	0.0063
	subset 1	0.0086	0.0060	0.0067	0.0069	0.0079
	subset 2	0.0077	0.0059	0.0058	0.0076	0.0078
	subset 3	0.0074	0.0062	0.0060	0.0067	0.0082
	Average	**0.0079**	0.0059	0.0059	0.0069	0.0075
$n = 20$						
	subset 0	0.0078	0.0053	0.0052	0.0064	0.0064
	subset 1	0.0082	0.0057	0.0066	0.0069	0.0079
	subset 2	0.0072	0.0059	0.0057	0.0076	0.0079
	subset 3	0.0070	0.0063	0.0060	0.0065	0.0084
	Average	0.0075	0.0058	0.0059	0.0069	**0.0076**
$n = 25$						
	subset 0	0.0081	0.0052	0.0053	0.0065	0.0064
	subset 1	0.0081	0.0054	0.0064	0.0069	0.0080
	subset 2	0.0067	0.0059	0.0056	0.0077	0.0081
	subset 3	0.0068	0.0062	0.0060	0.0066	0.0084
	Average	0.0075	0.0057	0.0058	0.0069	**0.0077**

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans' on Knowledge and Data Eng., 17(6), 2011.

[4] Dblp network dataset – KONECT, Feb. 2014.

[5] Bookcrossing (implicit) network dataset - KONECT, June 2014.

[6] Badrul M. Sarwar and George Karypis and Joseph Konstan and John Riedl, Recommender Systems for Large-scale E-Commerce: Scalable Neighborhood Formation Using Clustering, 2002.

[7] S. Sahebi and W. Cohen: Community-Based Recommendations: a Solution to the Cold Start Problem. In: Workshop on Recommender Systems and the Social Web (RSWEB), 2011.

[8] J. Ying, B. Shi; V.S. Tseng, H. Tsai, K. H. Cheng, S. Lin: Preference-Aware Community Detection for Item Recommendation: Technologies and Applications of Artificial Intelligence (TAAI), 2013.

[9] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In The adaptive web, Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl (Eds.). Lecture Notes In Computer Science, Vol. 4321, pages 291-324, 2007.

[10] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative Filtering Recommender Systems. Found in Trends in Human Computer Interaction. 2011.

[11] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. P10008, 2008.

[12] Fortunato, S. Community detection in graphs. Phys. Rep. 486, 2010.

[13] M. E. J. Newman: Fast algorithm for detecting community structure in networks. Phys. Rev. E 69 (6), 2004.

[14] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, Dorothea Wagner: On Modularity Clustering, IEEE Trans' on Knowledge and Data Engineering, vol. 20, 2008.

[15] Breese, J.S., Heckerman, D., Kadie, C.: Empirical Analysis of Predictive Algorithms for Collaborative Filtering, Proc. 14th UAI Conf., 1998.

[16] Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An Algorithmic Framework for Performing Collaborative Filtering. Int'l ACM SIGIR Conf, 1999.

[17] Bell, Robert M. and Koren, Yehuda. Improved Neighborhood-based Collaborative Filtering . 1st KDDCup'07. San Jose, California, 2007.

[18] Xiaoyuan Su and Taghi M. Khoshgoftaar.: A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence, 2009.

[19] Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms, Proc. 10th WWW Conf., 2001.

[20] Desrosiers, C., Karypis, G.: A Comprehensive Survey of Neighborhood-based Recommendation Methods. Recommender Systems Handbook, pages 107-144, 2011.

[21] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of RecSys*, 2010.