Random Walk based Entity Ranking on Graph for Multidimensional Recommendation

Sangkeun Lee, Sang-il Song, Minsuk Kahng, Dongjoo Lee, and Sang-goo Lee
School of Computer Science and Engineering
Seoul National University
Seoul, Korea
{liza183, danox, minsuk, therocks, sglee}@europa.snu.ac.kr

ABSTRACT

In many applications, flexibility of recommendation, which is the capability of handling multiple dimensions and various recommendation types, is very important. In this paper, we focus on the flexibility of recommendation and propose a graph-based multidimensional recommendation method. We consider the problem as an entity ranking problem on the graph which is constructed using an implicit feedback dataset (e.g. music listening log), and we adapt *Personalized PageRank* algorithm to rank entities according to a given query that is represented as a set of entities in the graph. Our model has advantages in that not only can it support the flexibility, but also it can take advantage of exploiting indirect relationships in the graph so that it can perform competitively with the other existing recommendation methods without suffering from the sparsity problem.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Search and Retrieval – *Information filtering, Retrieval models*

General Terms

Algorithms, Design, Experimentation.

Keywords

Multidimensional, Recommender systems, Context-aware recommender systems, Random walks, Implicit feedback, Usage log, Context-awareness.

1. INTRODUCTION

The performance of a recommender system can be assessed with respect to various criteria including accuracy, diversity, serendipity, and explainability [1]. Many existing studies have focused on improving such performance measures of recommending items to users. More formally, the traditional recommendation problem has been considered as a problem of defining an effective utility function $u: User \times Item \rightarrow Utility$. Typically the utility function u is estimated based on prior user preferences, and then it is used to predict the utility of items to the given user [2]. The typical form of recommendation is recommending top-k items to a given user with highest utility values.

However, in many real world applications, it is not sufficient to assume that there are only user and item dimensions because there is a wide range of other information that can be incorporated into the recommendation effort for better recommendation quality. For example, contextual attributes (e.g. location, time) or metadata attributes (e.g. age, address, genre) can be important factors. Moreover, it is not enough to only consider the typical form of recommendation which is recommending items to users because other types of recommendations can be very useful too. As an example, an online music service may require recommending not only items to users but also items to groups of users, users for items, items to users according to their contexts, users to users, and so on. The capability of handling multidimensional information and various types of recommendation can be called the 'flexibility' of a recommender system [3].

In this paper, we propose a graph-based approach that provides flexibility of recommendation. Considering each dimension in a multidimensional space as a domain of entities, a node in the graph is either an entity or a combination of entities. Weights between nodes are assigned using the entity relationships implied in implicit feedback datasets. Then, the multidimensional recommendation problem can be considered as the problem of ranking the nodes representing the entities of a target domain for a given query that is also represented as a set of nodes.

Exploiting a graph data model for recommendation has several advantages. Firstly, because any type of entities can be modeled as nodes in the graph, we can easily incorporate various types of information into recommendation. Additionally, we can also take advantage of not only direct relationships between entities but also indirect relationships. This can be very helpful for resolving the data sparsity problem which is an important issue in multidimensional recommendation. Lastly, we can adapt various graph ranking algorithms such as HITS [4], PageRank [5], and Personalized PageRank [6]. Specifically, we adapt Personalized PageRank algorithm which provides great flexibility. To the best of our knowledge, our approach is the first graph-based recommendation method which can fully exploit multidimensional information.

Our proposed recommendation method is based on implicit feedback datasets. Although explicit feedbacks better describe users' preferences, gathering users' explicit feedbacks can be burdensome to both users and systems. Especially, in the case of multidimensional recommendation, gathering explicit preferences is infeasible since the rating space becomes extremely large as the number of dimensions increases. Many previous works have shown that exploiting implicit feedback datasets can be a good substitution to the explicit feedback datasets [7][8][9][10]. An important advantage of exploiting implicit feedback datasets is that they often include multidimensional information that can be useful for recommendation. For example, assume that we have a movie rental history dataset that includes time and date of each rental. The time and date information can be useful for recommendation, because they tell us when users actually preferred (implied by the renting actions) the movies. Note that the time information included in explicit ratings cannot be used in this way because they describe when users rated the items but not when they preferred the items. Besides time and date information, various types of information included in implicit feedback datasets can be utilized this way (e.g. location, temperature, weather, etc.). Thus, we believe that implicit feedback datasets are more suitable for multidimensional recommendation than explicit feedback datasets.

Although our initial aim is to achieve flexibility, we should not give up the performance of recommendation. For the evaluation of our method, we compared our method to several other existing methods using two real-world datasets that consist of music listening logs gathered from an online music streaming service *last.fm* [11] and application purchase logs gathered from a mobile application store *LG's Oz Store* [12]. The experimental results show that our method outperforms or performs competitively with several existing algorithms including ItemRank [13] and PureSVD [14] in two-dimensional space. Furthermore, we show that exploiting multidimensional information improves recommendation performance.

The rest of this paper is organized as follows: In the following section, we present related literature especially focusing on multidimensional and graph-based recommendation approaches. In section 3, we formulate the problem of multidimensional recommendation. Section 4 explains the implicit feedback dataset and describes our graph model in detail. In section 5, we present the results of our experiments on two different real world datasets. Section 6 summarizes our work and proposes several application scenarios utilizing the flexibility of our method..

2. RELATED WORK

Existing recommendation methods can be mainly classified into two categories, Content-based Filtering and Collaborative Filtering. Content-based Filtering is one category of recommendation methods which recommend similar items to the items that users previously accessed. Content-based filtering exploits profiles of items (e.g. size, company, category, etc.) for comparing items to find similar items. The other approach is Collaborative Filtering. Collaborative Filtering does not use explicit item profiles, but it relies on users' previously known item preferences. Collaborative Filtering methods have been acknowledged as effective recommendation methods and widely used in many commercial services such as Amazon.com. Collaborative Filtering has several advantages over Content-based Filtering, such that it does not suffer from overspecialization problem and can recommend more novel items by taking advantage of users' relationships [2].

There are many different Collaborative Filtering (CF) methods, and they can be classified into memory-based approaches and model-based approaches in general. Memory-based approaches, such as User-based CF and Item-based CF, are straightforward. User-based CF finds similar users to the active user according to their previous preferences and aggregates the similar users' preferences to infer the given user's unknown item preferences. Item-based CF is similar, but it discovers similar items not users. Memory-based CF approaches directly infer recommendations based on users' previous preferences on items. On the other hand, a model-based approach learns a model from set of previous item access history or users' ratings, and then the model is used to recommend items. Recently, model-based approaches based on latent factor models such as matrix factorization have been introduced, and these approaches are known to provide accurate recommendation performances [14][15]. Our model can be classified as a model-based collaborative filtering method, because we first constructs a graph, in other words, learns a graph using implicit feedback dataset, and then use the graph for recommendation.

The importance of recommender system's flexibility is introduced by Adomavicius *et al.* [3]. The authors proposed a REQUEST that is a SQL-like query language for recommendation and can express various recommendations on OLAP style data. Koutrika *et al.*[16] introduced the FlexRecs framework which is designed to combine and express various types of recommendations using operators. These approaches have concentrated on defining declarative languages or operators that can be used to express recommendations. On the other hand, the capability of handling multidimensional data for recommendation has been also researched by many researchers. Adomavicius *et al.* [17] proposed an approach that is called reduction-based approach, and it is a pre-filtering approach that only utilizes the dataset that matches current contexts. Although the approach seems simple and reasonable, it does not work well with real world datasets due to the sparsity problem caused by filtering out data. To reduce the sparsity problem, several methods have been introduced. Lee *et al.* [8] proposed a disjunctive-based approach, but it has drawbacks that it assumes independency between all dimensions. Also, Kahng *et al.*[18] proposed a method which integrates multidimensional contextual features using a Learning-to-Rank algorithm. Until now, the flexibility and multidimensional recommendation have been considered as separate problems. However, in this paper, we aim to achieve flexibility of recommendation by modeling a multidimensional recommendation method which can support various types of recommendation requests.

So far, several graph-based recommendation methods have been introduced such as [13][19][20][21]. These methods only assume user and item environments in common. In detail, Fouss *et al.*[22] modeled the recommendation problem as measuring dissimilarities between nodes of a people-movie bipartite graph. The authors compared several random walk based quantities (e.g. L+, Commute Time(CT)) for recommendation. However, because the quantities such as L+ and CT are only defined between two nodes, this approach cannot be applied to recommendation types that require measuring relatedness among more than two nodes in graph (e.g. recommending items to a group of users according to current location). Gori *et al.*[13] presented a graph-based method named ItemRank that is a random-walk based scoring algorithm. The method is based on two key properties that are propagation and attenuation. It has shown better performance than Fouss *et al.*[22], however, it cannot deal with multidimensional environments in that the random-walk is performed on the homogeneous item graph. Cheng *et al.*[21] introduced a query-centric random walk method on k-partite graph, which is close to our method in that it can be used for multidimensional recommendation. But, although how to create k-partite graph is very important, the authors do not explain and focus more on speeding up the random walk process. Xiang *et al.* [23] presented a graph-based recommendation which aims to improve recommendation accuracy by mixing users' long-term and short-term preferences. The approach is able to deal with time dimensions, but it cannot incorporate the other types of information.

We use bipartite graph model to inherit the simplicity like [22] [24], but we extend bipartite graph model to utilize various dimensions. Getting hints from methods like [13][21][23], we also adapt *Personalized PageRank* algorithm to inherit its nice properties and scalability. Still, our method is novel in that it is designed to deal with multidimensional information and that how to construct graph and how to apply the *Personalized PageRank* algorithm are different from the other existing methods.

3. PROBLEM FORMULATION

Many existing recommender systems only operate in the two dimensional space $User \times Item$ and do not take other dimensions into consideration. To incorporate various dimensions into recommendation and provide flexibility, we define the multidimensional recommendation problem as follow. Formally, let $D_1, D_2, \ldots, D_n$ be disjoint sets, where each set D_i represents a domain and is defined as a set of entities $e_{i1}, e_{i2}, \ldots, e_{ik_i}$ .Then, we can define the set $\mathcal{E}$ as $D_1 \cup D_2 \cup \ldots \cup D_n$. First, for a given query Q that defined as a subset of $\mathcal{E}$, we let u be a utility function $u(Q, e_{ij})$ that measures the relevance score between query Q and entity e_{ij} . Then, we want to find top-k entities $e_{ij} \in D_T$ that maximize $u(Q, e_{ij})$ where $\mathrm{D_T}$ is the target domain that is of entities we are interested in.

The advantage of considering the problem as we defined is that the definition can express not only the traditional recommendation but also many recommendation variations. For example, the traditional recommendation problem of recommending items to users can be considered as one special case of our problem definition. Assume that we want to recommend items to user $u \in USER$, where $USER$ and $ITEM$ are domains. Then the query $Q = \{u\}$, and D_T is $ITEM$. Because our problem definition allows query as a set of any entities, we can simply express the case of recommending items to more than one user. In this case, the query is defined as $\{u_1, \ldots u_{k_u}\}$ where $u_1, \ldots u_{k_u} \in USER$. Context-aware recommendation is another special case of our definition. Assume that there are four domains $USER, LOCATION, WEATHER$, and $SONG$. When we want to recommend *SONG* to user $\in USER$, the current location is $l \in LOCATION$, and the current weather is $w \in WEATHER$, we can consider the query $Q = \{u, l, w\}$ and D_T is $SONG$.

4. MODELING OUR METHOD

In this section, we explain the implicit feedback dataset and our graph model first, and then we describe our novel recommendation method that is based on random walk process on our graph model.

4.1 Data Model

To solve the problem as we defined in the previous section, we need to define a utility function $u(Q, e_{ij})$ where Q is a query that represented as a set of entities and e_{ij} is an entity. We define the utility function u based on a log table that implies users' implicit preferences. In this section, we discuss the implicit feedback datasets we use and explain how we construct our graph model in detail.

There are various logs such as music listening log, item purchase logs movie rental log, and so on. In our approach, we consider each attribute of such log tables as a domain and the values of the attributes as entities of the domain. Formally, we can define a set of events $L = \{v_1, v_2, \ldots, v_{n_v}\}$ from a log table. Each event v is a tuple $(e_{D_1}, e_{D_2}, \ldots e_{D_n})$, where $e_{D_i} \in D_i$. For example, Table 1 shows an example of movie rental history dataset. In this example, we can define the domains as *USER*={'Matt', 'Jack'}, *MOVIE*={'Superman','300','Rent'}, *DATE*={'09-16-2008','09-17-2008'}, and *LOCATION*={'Seoul'}. Thus, we can define L as a set of tuples ('Matt', 'Superman', '09-16-2008','Seoul'), ('Jack, '300, '09-17-2008', 'Seoul') and ('Matt', 'Rent', '09-17-2008', 'Seoul')

Table 1. Example of Implicit feedback Dataset

USER	MOVIE	DATE	LOCATION
Matt	Superman	09-16-2008	Seoul
Jack	300	09-17-2008	Seoul
Matt	Rent	09-17-2008	Seoul

To simplify the problem, we assume that all the values in the log table are categorical values. But, in real world log data, there are lots of numerical values. To deal with numerical values, we need to transform the numerical values into categorical values. For example, temperature values can be categorized into 'Hot', 'Cold', 'Moderate'. This process is called 'context abstraction' [25], and it should be carefully done with many concerns in that it can affect the performances of recommendation. There have been several related works focusing on this topic [8][26], but it is beyond the scope of this paper.

Log tables do not explicitly describe user preferences. So, we need to interpret the dataset. Several two dimensional existing approaches [7][27] interpret the implicit dataset as binary preferences; a user's access to item is interpreted to rating 1, while a non-access is interpreted to rating 0. On the other hand, some approaches [8][10] use the item access count of users as pseudo-ratings. Because we are dealing with multiple dimensions and consider the problem as answering queries, we need more general relatedness of entities besides user preferences on items. To do so, we firstly build a bipartite graph $G = \{V, E\}$ based on given log table, and we use random walk method to find the relatedness between given query and target entities.

A flexible recommender system should be able to reflect service providers' various requirements. Thus, we use recommendation factors given by service providers when we build our graph data model. We assume that the set $F = \{f_1, f_2, \ldots f_{n_F}\}$ is given by a service provider, where each f_i is a recommendation factor that is defined as a set of one or more than one non-target domains (e.g. {*USER*}, {*USER*, *LOCATION*}). We define w_i as the weight value for f_i , where $w_1 + w_2 + \cdots + w_{n_F} = 1$. Each f_i decides what affects recommendation results, and each w_i decides how much the factor affects the recommendation.

To build our graph model, first, we create a set of nodes $V = V_1 \cup V_2 \cup \ldots \cup V_{n_F} \cup V_T$, where the nodes in V_i correspond to the entity combinations whose domains match the domains of f_i and the nodes in V_T correspond to the entities of target domain D_T.

Then, we define a $|V| \times |V|$ weighted adjacent matrix M for the bipartite graph $G = \{V, E\}$.We assume that edges exist only from the nodes that are not in V_T to the nodes in V_T. This assumption lets the graph G be bipartite.

The Figure 1. shows an example bipartite graph constructed based on the given Table 1, when the set F is given as $\{f_1, f_2, f_3\}$, where $f_1 = \{USER\}, f_2 = \{USER, DATE\}, f_3 = \{USER, LOCATION\}$ and the D_T is *MOVIE*.

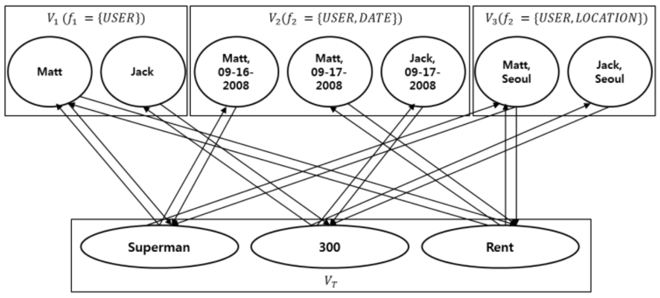

Figure 1. Example of our bipartite graph model.

We use the number of co-occurrences in the same tuples of given log table to assign the weight between two nodes $v_i, v_j \in V_1, \dots, V_{n_F}, V_T$. Element m_{ij} of matrix M can be defined as:

$$m_{ij} = \begin{cases} C(i,j) & , if\ v_i \notin V_T\ and\ v_j \in V_T \\ C(i,j) \times w_k & , if\ v_i \in V_T, v_j \notin V_T, and\ v_j \in V_k \\ 0 & , otherwise \end{cases}$$

where $C(i,j)$ is the co-occurrence number of entities, which v_i and v_j correspond to, and where w_k is the weight of f_k that V_k corresponds to.

Then, we can compute the transition possibility matrix P by normalizing the matrix M as follow:

$$p_{ij} = \begin{cases} \dfrac{m_{ij}}{\sum_{v_k \in outlink\ [v_i]} m_{ik}}, if\ \ outlink[v_i] \neq 0 \\ 0, otherwise \end{cases}$$

In the following section, we will explain how we use the transition probability matrix *T* for ranking entities for recommendation.

4.2 Ranking Entities on Graph

4.2.1 Defining Ranking Function

As we defined problem earlier, we adapt *Personalized PageRank* algorithm [4] for ranking entities on our graph model. *Personalized PageRank* algorithm is a variation of the PageRank algorithm [5]. The PageRank score of a node will get higher score when if a node is connected to more important nodes with less outgoing links. The PageRank score can be calculated as:

$$\vec{r} = cP^T\vec{r} + (1-c)\frac{1}{n}\vec{e}$$

where n is the number of nodes, r_i is the rank score for node v_i, $\vec{e} = (1,1,\dots,1)^T$, and c is a damping factor constant that is normally given as 0.85.

The algorithm is based on a random-walk with restart process, so the random surfer will uniformly choose a random node for restarting with possibility $(1-c)$ during the random walk process.

However, for calculating the *Personalized PageRank* score of a node, we substitute $\frac{1}{n}\vec{e}$ to a personalized teleport $\vec{t}$ that is the personalized bias vector that normally represents a user's interest. Thus, the *Personalized PageRank* is calculated as $\vec{r} = cT\vec{r} + (1-c)\vec{t}$, where t_i is 1 if the node is user's interest, and otherwise t_i is 0. *Personalized PageRank* algorithm calculates the node authority value, but it adjusts the score with a personalized bias. In this work, by getting a hint from the *Personalized PageRank* algorithm, we use a query vector $\vec{q}$ as the teleport vector instead of using the personalized teleport $\vec{t}$

As we explained in section 3, a query Q for a multidimensional recommendation is defined as a set of entities. For the given query $Q = \{e_1, e_2, \dots e_{n_q}\}$, we define a vector $\tilde{\vec{q}}$ as follow:

$$\tilde{q}_i = \begin{cases} 1, if\ \mathcal{E}_{v_i} \subseteq Q \\ 0, otherwise \end{cases}$$

where $\mathcal{E}_{v_i}$ is a set of entities that the node v_i corresponds to.

We normalize the vector $\tilde{\vec{q}}$, and we use the normalized vector $\vec{q}$ as the teleport vector for ranking entities on the graph. So, the ranking score vector is calculated as $\vec{r} = cP^T\vec{r} + (1-c)\vec{q}$. The random walk with restart process makes each node get the higher ranking score if the node is more closely related to the nodes that entities in the query correspond, considering the authority of nodes at the same time.

As result, we can use the rank r_i for each node v_i, and then we use the value as the rank score r_t for entity $e_t \in D_T$ that v_t corresponds to. As a result, we can sort the entities in D_T and find top-*k* entities for the query.

4.2.2 Advantages of Adapting Personalized PageRank

There are several advantages of adapting the *Personalized PageRank* for our problem. First, we can take advantages of propagation and attenuation properties. These properties are introduced in [13]. The propagation property is that the relatedness of the nodes propagates following the links, and the attenuation property is that the propagation strength decreases as the propagation goes further from the starting node. For example, a user node is linked to a song node, and another user node is related to the song node, then these two user nodes are related to each other but less related then directly linked nodes. By using these properties, we can even measure the relatedness between nodes that are not directly linked. Also, this property lets the system handle the sparsity problem that is even more serious in multidimensional recommendation [8]. Second, through a random-walk process, we can find a rank score of target entities by blending the influences of various factors. For instance, the rank score for each movie computed using the graph in Figure 1. can be considered as the blend of the recommendation factors f_1, f_2, and f_3, where $f_1 = \{USER\}$ implies user preferences, $f_2 = \{USER, DATE\}$ implies user preferences depending on the date, and $f_3 = \{USER, LOCATION\}$ implies user preferences depending on the location. Finally, we can take advantages of the nice properties of *Personalized PageRank* algorithm in that the algorithm can be pre-computed and it scales for large size datasets. Our rank score formula can be transformed to $\vec{r} = (1-c)(I - cP^T)^{-1}\vec{q}$. It shows that if we can pre-compute the matrix $(1-c)(I - cP^T)^{-1}$ which is independent from the query vector, then we can simply multiply the matrix and query vector to compute the rank score for each node according to query with reasonable time complexity O(mn). In our method, the query is given as a sparse vector with only several elements. Thus, it can be computed at online times. Although pre-computation requires matrix inversion which has time complexity of $O(n^3)$, [28] has shown that we can estimate the inverse matrix with reasonable cost by using dimension reduction approach. There is another approach by Cheng *et al.*[21] which aims to reduce the cost by using some clustering methods.

5. EXPERIMENTS

We conducted several experiments to evaluate our proposed method by comparing with existing methods using a real world dataset.

5.1 Dataset and Evaluation Measure

For our experiments, we used two real-world datasets. The first dataset consists of logs from a popular online music streaming service called *last.fm*. The dataset contains 673,293 music listening logs of 1,699 users on 3,000 songs from 01-01-2008 to 11-22-2008. The other dataset consists of logs from a mobile application market called *LG Oz Store*. It contains 862,291 application purchase logs of 4,000 users on 4,000 applications from 06-06-2010 to 04-26-2011. For both datasets, the original log tuples only contain $USER$, $ITEM$, and $DATE$ attributes. Therefore, we added several contextual attributes such as DOW (day of week), $HALF$ (the value 1 means the first six month of the year, and the value 2 means the second six month of the year), $QUARTER$ (each value 1,2,3,4 means the 1st, 2nd, 3rd, and 4th quarter of the year) by analyzing the *DATE* attribute. For each dataset, we sorted the logs into ascending order by *DATE* and used the first 80% of the full data as the training set, and we generated test sets from the rest of the data. Each test set consists of randomly chosen 1,000 logs.

Generally, the rating error measures such as *Mean Absolute Error*, *MAE* or *Root Mean Square Error*, *RMSE* have been used to evaluate recommender systems that focus on rating estimation. However, due to the fact that we do not focus on estimating actual ratings of users and only focus on top-k recommendation of entities, we cannot use such measures. Thus, we use a metric called HR@k [8] that is more suitable for measuring the performance of top-k recommendation. We can consider our problem as the problem of predicting hidden entities in test cases. HR@k is defined as $\frac{\#\text{hit}}{n}$, where n is the number of tests and #hit is the number of cases that the hidden entity in the test case is ranked in top-k in the produced ranking list by a recommender system.

5.2 Baseline Methods

We compared the top-k recommendation performance of our method with several existing methods: Popularity-based (POP), User-based CF (UKNN), ItemRank (IRANK), and PureSVD (PSVD). Because UKNN, IRANK, PSVD are designed for two dimensional User-Item matrix, we transformed our training data set into a USER-ITEM pseudo rating matrix by considering the normalized access count of a user on an item as the user's pseudo rating on the item.

Popularity-based (POP): Popularity-based method generates a ranking list based on the popularity of items in the training dataset. The method is not a personalized method, thus it generates the same ranking list for every user.

User-based Collaborative Filtering (UKNN): User-based CF finds the k similar users to the active user and aggregates the ratings of the similar users. We used the aggregation function as follow:

$$r_{u,i} = \overline{r_u} + \frac{\sum_{u' \in \hat{U}} sim(u,u') \times (r_{u',i} - \overline{r_{u'}})}{\sum_{u' \in \hat{U}} sim(u,u')}$$

where $r_{u,i}$ is the rating of user u on item i, $\overline{r_u}$ is the average ratings of user u, $\hat{U}$ is the set of k most similar users, and the $sim(u,u')$ is the cosine similarity of rating vectors of u and u'. We empirically set k=70 for our experiments. The estimated ratings of the active user are used as the rank score for top-k recommendation.

ItemRank (IRANK): As explained in the section 2, ItemRank first creates an item graph based on the given rating matrix and uses a teleport vector that is constructed using the active user's previous ratings for PageRank computation.

PureSVD (PSVD): PureSVD [14] is one of the state-of-the art methods and known to perform very well for top-k recommendation, although the method cannot be used to estimate actual ratings of users. The PureSVD method is based on matrix factorization that reduces the dimension of the matrix, so the number of dimensions d should be decided. We empirically set the $d=30$ for our experiments.

5.3 Experimental Results

Although the initial aim of our research is to achieve flexibility and capability of exploiting various domains' information, we should not give up the recommendation performance. So, we conducted two experiments for evaluating our method. The first experiment is to evaluate the recommendation performance of traditional recommendation case, which is recommending items to users. Then, the second experiment is designed to look at the effects on exploiting multidimensional information that is included in the implicit feedback dataset.

In each experiment, we used two different datasets for testing two recommendation scenarios. The first scenario is a typical case that the recommender system recommends previously unseen items of the users. This scenario is suitable when dealing with contents that are not repeatedly consumed by users. For this scenario, we used the *LG Oz Store* dataset (OZSTORE) in that users do not purchase the same applications repetitively. The second scenario is that the system recommends items to user including the items that have been already consumed by the user. We used the last.fm dataset (LASTFM) for this case because users generally listen to the same songs repeatedly.

For the first scenario, we only sort the applications that have never been purchased by the given user, and we sort all songs to generate the predicted ranking list for the second one. We use the term GFREC (Graph-based Flexible Recommendation) as an abbreviation of our approach. The damping factor c is set to be the typical value 0.85.

5.3.1 Recommending Items to User

We constructed a graph for given each dataset where $F = \{f_1 = \{USER\}\}$ and D_T is $ITEM$. Each user appears in each test log can be considered as an active user. We used $Q = \{e_{user}\}$ as a query for each test case, where e_{user} is the entity that describes the active user.

Table 2 shows the result of our experiment. For the first case, which is the case of recommending previously unseen items to users using OZSTORE dataset, PSVD shows the best performance, but our method outperforms POP, UKNN, and IRANK and shows the competitive performance with PSVD. For the LASTFM case, our method significantly outperforms all of the baseline methods that we compared. We can infer that exploiting

Table 2. Top-*k* Recommendation Performance Comparisons

HR@*k*	SCENARIO 1 (OZSTORE)					SCENARIO 2 (LASTFM)				
	POP	UKNN	PSVD	IRANK	GFREC	POP	UKNN	PSVD	IRANK	GFREC
HR@10	0.027	0.041	**0.050**	0.045	0.048	0.011	0.014	0.050	0.088	**0.096**
HR@20	0.043	0.084	0.090	0.081	0.089	0.024	0.036	0.090	0.151	0.158
HR@30	0.075	0.114	0.123	0.108	0.118	0.036	0.062	0.121	0.181	0.193
HR@40	0.100	0.137	0.152	0.135	0.139	0.044	0.078	0.152	0.215	0.232
HR@50	0.126	0.157	0.179	0.153	0.157	0.053	0.097	0.181	0.241	0.274

Table 3. Effects on Exploiting Multidimensional Information

HR@*k*	SCENARIO 1 (OZSTORE)								SCENARIO 2 (LASTFM)							
	$\mathbf{F_1}$	$\mathbf{F_2}$	$\mathbf{F_3}$	$\mathbf{F_4}$	$\mathbf{F_5}$	$\mathbf{F_6}$	$\mathbf{F_7}$	$\mathbf{F_8}$	$\mathbf{F_1}$	$\mathbf{F_2}$	$\mathbf{F_3}$	$\mathbf{F_4}$	$\mathbf{F_5}$	$\mathbf{F_6}$	$\mathbf{F_7}$	$\mathbf{F_8}$
HR@10	0.048	0.046	0.028	0.008	0.008	0.058	**0.062**	0.057	0.096	0.011	0.015	0.098	0.024	0.097	**0.099**	0.095
HR@20	0.089	0.070	0.060	0.022	0.018	0.100	0.103	0.095	0.158	0.022	0.024	0.160	0.037	0.164	0.164	0.169
HR@30	0.118	0.147	0.091	0.039	0.031	0.125	0.130	0.135	0.193	0.038	0.039	0.208	0.044	0.210	0.210	0.209
HR@40	0.139	0.190	0.106	0.064	0.047	0.159	0.169	0.183	0.232	0.047	0.047	0.250	0.054	0.248	0.249	0.242
HR@50	0.157	0.207	0.138	0.091	0.070	0.195	0.203	0.223	0.274	0.057	0.053	0.294	0.062	0.291	0.292	0.270

indirect relationships among entities through random walk process can resolve the data sparsity and contribute to the recommendation accuracy. So far, although only the first scenario has been considered in many previous works, the second scenario is also very important because there are many applications dealing with repeatedly consumed items (e.g., music, food). In the case of recommending items repeatedly, not only what item to recommend but also in what situation to recommend is important. Note that other baseline methods we compared can be only used for the two dimensional recommendations; however, our method has advantage in that it can be easily extended to incorporate various other contextual attributes (e.g. location, time) into recommendation. We discuss the effects on exploiting multidimensional information and the applications and applications of using flexible queries in the following sections.

5.3.2 *Exploiting Multidimensional Information*

In this experiment, we test our method, varying the settings of utilizing multidimensional information. Each test case (a tuple) in a test dataset can be considered as a set of entities. We hide the item entity in each test case and use the set of the other entities in the test case as a query. Using the query, we predict the rank of the hidden song entity's rank. Table 4 describes our experimental setting variations, and Table 3 shows the experimental result.

We can consider the F_1 as the traditional two dimensional recommendation setup. F_2 and F_3 are the settings for recommending items to users based on contextual information without using user information. We can consider F_4 and F_5 as the cases of utilizing user preferences depending on a specific dimension. From F_6 to F_8 are the examples of merging several factors to produce ranking list. For example, F_6 can be considered as the case of merging user's general preference and user's preference depending on *HALF* dimension. Note that there are virtually infinite settings of utilizing multidimensional information. Our method has such flexibility so that recommendation service providers can setup their recommendation to reflect their requirements.

Table 4. Multidimensionality Setting Variations

Factor Set	weight
$F_1 = \{f_1 = \{USER\}\}$	$w_1 = 1.0$
$F_2 = \{f_1 = \{HALF\}\}$	$w_1 = 1.0$
$F_3 = \{f_1 = \{DOW\}\}$	$w_1 = 1.0$
$F_4 = \{f_1 = \{USER, HALF\}\}$	$w_1 = 1.0$
$F_5 = \{f_1 = \{USER, QUARTER\}\}$	$w_1 = 1.0$
$F_6 = \left\{\begin{matrix} f_1 = \{USER\}, \\ f_2 = \{USER, HALF\} \end{matrix}\right\}$	$w_1 = 0.2$, $w_2 = 0.8$
$F_7 = \left\{\begin{matrix} f_1 = \{USER\}, \\ f_2 = \{USER, HALF\} \\ f_3 = \{USER, QUARTER\} \end{matrix}\right\}$	$w_1 = 0.2$, $w_2 = 0.4$, $w_3 = 0.4$
$F_8 = \left\{\begin{matrix} f_1 = \{USER\}, \\ f_2 = \{HALF\}, \\ f_3 = \{DOW\}, \\ f_4 = \{QUARTER\}, \\ f_5 = \{USER, DOW\}, \\ f_6 = \{USER, HALF\}, \\ f_7 = \{USER, QUARTER\} \end{matrix}\right\}$	$w_1 = 0.15$, $w_2 = 0.05$, $w_3 = 0.05$, $w_4 = 0.05$, $w_5 = 0.1$, $w_6 = 0.3$, $w_7 = 0.3.$

As we can see the result of F_2 and F_3 , recommendation performance decreases when our method does not utilize user information. It is reasonable that user preference is critical for recommendation. For some cases such as F_4,F_5 (OZSTORE) and F_5 (LASTFM), although our method utilizes the combination of *USER* and other dimensions, the recommendation performances decrease. It is because the system often cannot make a recommendation list when there is no matched node for given queries. We can consider F_6, F_7, and F_8 as the solution for this problem, these three settings allow merging the factors so that it resolves the node matching problem. As results, they show better performance than the baseline (F_1). Note that our method provides better accuracy than PSVD for OZSTORE dataset by using additional information, although it does not in two-dimensional space. The experimental result shows that our method can improve the recommendation performance by exploiting various dimensions without suffering from the sparsity problem. Although there can be other settings that can bring on better recommendation performances, we do not focus on this issue here.

Another advantage is that it can increase the diversity of recommendation results, because the system produces different ranking results not only to different users but also to different queries. Even when the system recommends items to the same user, if the additional entities (e.g. location) besides the user entity are in the query, the recommendation will produce different ranking lists. We were not able to test the effects of utilizing various types of information such as location and temperature due to the lack of such information in our implicit feedback data, but they can be utilized also in the same way and may increase the recommendation accuracy.

6. CONCLUSIONS

Research in recommendation systems has been focused mainly in improving accuracy, diversity, and novelty. Relatively little effort has been directed to the flexibility of recommender systems. The flexibility of dealing with multidimensional information is very important not only for the accuracy of recommendation, but also for many aspects. In this paper, we discuss the importance of dealing with flexibility, and we consider the problem as a query answering problem in multidimensional recommendation space. In summary, the main contributions of our paper can be summarized as follows.

First, we presented a novel graph-based multidimensional recommendation method that supports flexibility without suffering the sparsity problem. Service providers can reflect their requirements depending on situation by changing the recommendation factors, queries, and target dimensions. At the same time, exploiting the indirect relationships, i.e., preference propagation in the graph, contributes to resolve the sparsity problem. Our method is scalable because it inherits the nice properties of the *Personalized PageRank* algorithm.

Second, through experiments using two different datasets, we compare our method to existing state-of-the-art approaches and show that our method shows better top-k recommendation performances. Also, we show that exploiting multidimensional information in implicit feedback dataset can improve recommendation quality.

The flexibility of recommender system allows various forms of recommendation; however, it is difficult to numerically measure flexibility. Thus, we propose several applications that can take advantage of the flexibility provided our method.

Recommendation for more than one user: Our method can reflect more than one user's preferences and merge them to produce one ranking list by using a set of user entities as the query. For example, "What are the best movies for Matt and his girlfriend Susan?," can be answered by Q={'Matt','Susan'}, $D_T = MOVIE$)

Promoting items: By using a set of items as the query, our method can find the users that are the most suitable target for the promotion of the items: "Who are the suitable target users for an Indian movie 'My name is Khan'?," - Q={'My name is Khan'}, $D_T = USER$

Recommendation based on items: For an anonymous user with only a list of items accessed, the system can use this set of items as a query: "What are the best songs to be recommended to an anonymous user who has listened to 'Yesterday', 'Yellow Submarine'?," - Q={'Yesterday', 'Yellow Submarine'} , $D_T = SONG$

Friends Recommendation: When a service provider wants to recommend users to users to let them share their favorites, the system can set the target dimension as the user dimension: "Who are the users that have similar taste with 'Matt'?," - Q={'Matt'}, $D_T = USER$

Item Recommendation with Intentional Bias: Sometimes, a recommendation service provider requires biasing the recommendation results for certain marketing or promotional purposes. Such biases can be 'Jazz style songs', 'Obama's favorite songs', 'User's current explicit interests', and more. All these biases can be used as additional entities in the query so that the recommendation result can reflect the bias.

Context-aware Item Recommendation: Our method can incorporate the contextual information of users by considering a user's current context as the query: "What is the most suitable movie for Michael when the weather is Summer and the location is Home?", - Q={'Michael', 'Summer', 'Home'}, $D_T = MOVIE$

As shown by these examples, our method can support various types of recommendations accommodating specific requirements. Although it is required to build different graphs for different target dimensions or different multidimensional settings, it is not a burden in that implicit feedback data can be transformed into graph in reasonable processing time. (e.g., It takes 42.50 seconds average for constructing the graph for 862,291 logs with setting F_8).

Exploiting various information included in implicit feedback dataset can improve the accuracy of recommender systems. We believe that it is possible to learn the weights of each factor by adapting Learning-to-Rank algorithms such as Ranking SVM [29]. This is reserved for our future work. Also, we plan to evaluate our work using real-world lifelog datasets [30].

7. ACKNOWLEDGEMENT

This work was supported by Industrial Strategic technology development program (10038588, Context-aware based Customer Experience Management Platform on App. Store Environment) funded by the Ministry of Knowledge Economy (MKE, Korea)

8. REFERENCES

[1] H. Stack. Training and testing of recommender systems on data missing not at random. In KDD '10: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2010. http://doi.ac.org/10.1145/1835804.1835895

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, Vol. 17(6), pp. 734–749, 2005.

[3] G. Adomavicius, A. Tuzhilin, and R. Zheng. REQUEST: A query language for customizing recommendations. Information System Research. Information Systems Research, Vol. 22(1), pp. 99-117, 2011. http://doi.acm.org/10.1287/isre.1100.0274

[4] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 668-677, 1998.

[5] S. Kamvar , H. Haveliwala, C. D. Manning, and Gene H. Golub. Extrapolation methods for accelerating PageRank computations. In WWW '03: Proceedings of the 12th International Conference on World Wide Web. 2003. http://doi.acm.org/10.1145/775152.775190

[6] T. Haveliwala, S. Kamvar and G. Jeh. An analytical comparison of approaches to personalizing pagerank. Technical Report. Stanford. 2003.

[7] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, pp. 263-272. IEEE. 2008.

[8] D. Lee, S. E. Park, M. Kahng, S. Lee, and S.-g. Lee. Exploiting contextual information from event logs for personalized recommendation. In Computer and Information Science, Studies in Computational Intelligence, pp. 121-139. Springer. 2010.

[9] J. Wang, A. Vries, and M. Reinders. A user-item relevance model for log-based collaborative filtering. In LNCS. In Advances in Information Retrieval. Vol. 3936, pp.37-48. 2006. http://doi.acm.org/10.1007/11735106_5.

[10] T. Lee, Y. Park, and Y. Park. A similarity measure for collaborative filtering with implicit feedback. Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence LNCS, Vol. 4682/2007, pp. 385-397. 2007. http://doi.acm.org/10.1007/978-3-540-74205-0_43

[11] Last.fm, http://www.last.fm/

[12] LG U+ Oz Store, http://ozstore.uplus.co.kr/

[13] M. Gori and A. Pucci. ItemRank, A random-walk based scoring algorithm for recommender engines, In 20th International Joint Conference on Artificial Intelligences. 2007.

[14] P. Cremonesi, Y. Koren, R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In RecSys '10: Proceedings of the fourth ACM Conference on Recommender Systems. 2010. http://doi.acm.org/10.1145/1864708.1864721

[15] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In KDD '08: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 426-343. 2008.

[16] G. Koutrika, B. Bercovitz, and H. Garcia-Molina. FlexRecs: expressing and combining flexible recommendations. In SIGMOD '09: Proceedings of the 35th SIGMOD International Conference on Management of Data . 2009. http://doi.acm.org/10.1145/1559845.1559923

[17] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. ACM Transactions on Information Systems (TOIS), Vol. 23(1), pp. 103–145. 2005.

[18] M. Kahng, S. Lee, and S.-g. Lee. Ranking in context-aware recommender systems. In WWW '11: Proceedings of the 20th International Conference Companion on World Wide Web. 2011. http://doi.acm.org/10.1145/1963192.1963226

[19] Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. Video suggestion and discovery for YouTube: Taking random walks through the view graph. In WWW '08: Proceedings of the 17th International Conference on World Wide Web, pp.895-904. 2008. http://doi.acm.org/10.1145/1367497.1367618

[20] C. Desrosiers and G. Karypis A novel approach to compute similarities and its application to item recommendation.In PRICAI 2010: Trends in Artificial Intelligence. In LNCS. Vol. 6230/2010, pp. 39-51. 2010. http://doi.acm.org/10.1007/978-3-642-15246-7_7

[21] H. Cheng, P. Tan, J. Sticklen, and W. F. Punch. Recommendation via query centered random walk on k-partite graph. In ICDM '07: Proceedings of the Seventh IEEE International Conference on Data Mining. 2007. http://doi.acm.org/ 10.1109/ICDM.2007.8

[22] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Transactions on Knowledge and Data Engineering, Vol. 19(3), pp. 355-369, 2007.

[23] L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, and J. Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In KDD '10: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2010. http://doi.acm.org/ 10.1145/1835804.1835896.

[24] J. Davidson, B. Liebald, and Junning Liu. The YouTube Video Recommendation System. In RecSys '10: Proceedings of the fourth ACM Conference on Recommender Systems. 2010. http://doi.acm.org/ 10.1145/1864708.1864770.

[25] S. Lee, J. Chang, and S.-g. Lee. Survey and Trend Analysis of Context-Aware Systems. Information-An International Interdisciplinary Journal. Vol. 14(2). pp. 527-548. 2011.

[26] D. Shin, J.-w. Lee, J. Yeon, Lee, and S.-g. Lee. Context-aware recommendation by aggregating user context. In CEC 2009: Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing, pp. 423–430. IEEE. 2009.

[27] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In WWW '07: Proceedings of the 16th International Conference on World Wide Web, pp. 271–280. 2007.

[28] H. Tong. C. Faloutsos and J. Pan. Random walk with restart: fast solutions and applications. Knowledge and Information Systems. Vol. 14(3), pp. 327-346. 2006. http://doi.acm.org/10.1007/s10115-007-0094-2

[29] Y. Cao, J. Xu , T.Y. Liu, H. Li, Y. Huang, and H.W Hon. Adapting ranking SVM to document retrieval. In SIGIR '06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 2006. http://doi.acm.org/10.1145/1148170.1148205

[30] S. Lee, G. Gong, and S.-g. Lee. LifelogOn: Log on to your lifelog ontology!. In ISWC '09: Proceedings of the 8th International Semantic Web Conference. 2009.