A General Graph-based Model for Recommendation in Event-based Social Networks

Tuan-Anh Nguyen Pham*, Xutao Li*, Gao Cong*, Zhenjie Zhang†
*School of Computer Engineering, Nanyang Technological University, Singapore 639798
Email: pham0070@e.ntu.edu.sg, lixutao@ntu.edu.sg, gaocong@ntu.edu.sg
†Advanced Digital Sciences Center, Illinois at Singapore Pte. Ltd.
Email: zhenjie@adsc.com.sg

***Abstract*—Event-based social networks (EBSNs), such as Meetup and Plancast, which offer platforms for users to plan, arrange, and publish events, have gained increasing popularity and rapid growth. EBSNs capture not only the online social relationship, but also the offline interactions from offline events. They contain rich heterogeneous information, including multiple types of entities, such as users, events, groups and tags, and their interaction relations. Three recommendation tasks, namely recommending groups to users, recommending tags to groups, and recommending events to users, have been explored in three separate studies. However, none of the proposed methods can handle all the three recommendation tasks. In this paper, we propose a general graph-based model, called HeteRS, to solve the three recommendation problems on EBSNs in one framework. Our method models the rich information with a heterogeneous graph and considers the recommendation problem as a query-dependent node proximity problem. To address the challenging issue of weighting the influences between different types of entities, we propose a learning scheme to set the influence weights between different types of entities. Experimental results on two real-world datasets demonstrate that our proposed method significantly outperforms the state-of-the-art methods for all the three recommendation tasks, and the learned influence weights help understanding user behaviors.**

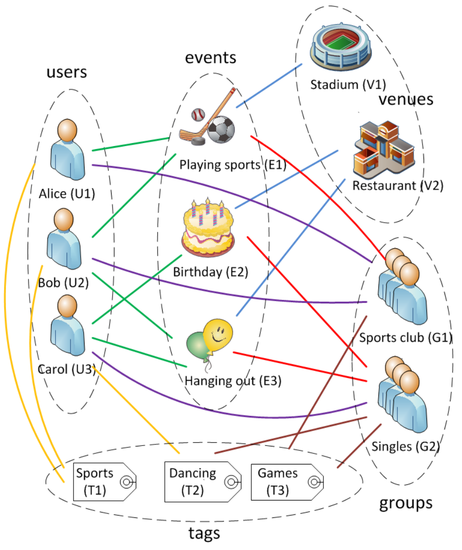


Fig. 1: An example of an event-based social network

I. Introduction

Recent years have witnessed the rapid growth of event-based social network services (*e.g.*, Meetup and Plancast), which provide a new kind of social network that connects people through events. For example, Meetup, currently has 16 million users with more than 300,000 monthly events[1]. On these web services, users can create or join different social events (*e.g.*, dining out, playing sports, parties). To better organize events, these services allow users to form online groups, in which a user can publish and announce events to other group members.

Figure 1 illustrates an example of an event-based social network that consists of five types of entities: users, events, groups, tags and venues, together with their relations. In this example, Alice and Bob join the group "Sports club" while Carol joins the group "Singles". Group "Sports club" held a football match in a stadium, which was participated by both Alice and Bob; meanwhile, Carol took part in two events, namely "Birthday" and "Hanging out", created by group "Singles" in a restaurant. Both groups use tag "Games" to indicate their interests. Alice and Bob use tag "Sports", while Carol uses tag "Dancing".

[1]http://www.meetup.com/about/

With the rich interaction information available, one natural question is how to make use of it to provide better services for users. For example, several problems can be raised: 1) Which groups would a particular user like to join? 2) Which tags might a group choose when constructing its profiles? 3) Who will attend an upcoming event? Answering these questions is necessary in order to predict user activities when they participate in an event-based social network. As a matter of fact, these questions require us to design recommendation systems for three specific tasks: recommending groups to users, tags to groups and events to users.

Several studies have been conducted separately for each of the three recommendation tasks on an event-based social network. Zhang et al. [1] proposed a factorization model that exploits social and location features for event-based group recommendation. Liu et al. [2] introduced a topic model to solve the tag recommendation problem for groups. Liu et al. [3] used a simple graph-based approach to recommending users for an event, which performs the information diffusion over user network from some seed users. We observe that each of those recommendation problems is solved by a distinct approach, which is not applicable to the rest. In other words, a general solution that can resolve all of those problems remains unexplored.

In this paper, we propose a *general-purpose Heterogeneous graph-based Recommendation System* model (HeteRS), which can solve multiple tasks of recommendation for problems that can be modeled as a heterogeneous graph. Particularly, we construct a heterogeneous graph to model the interactions between multiple entities (users, events, groups, and tags etc.) in an event-based social network, where entities are represented as different types of nodes and the interactions between them are represented as different types of edges. Moreover, after analyzing the data in event-based-social networks, we find some useful temporal patterns of user behaviors, and our graph is extended to incorporate them. We then convert the recommendation problem into a node proximity calculation problem to some query nodes on the heterogeneous graph, so as to accomplish the following three tasks: group-to-user recommendation, tag-to-group recommendation and event-to-user recommendation.

The key challenge to evaluate the node proximity lies at that our heterogeneous graph contains multiple types of entities and they influence each other via different types of interactions. It is difficult to know and balance the importance of these influences for proximity calculation. Moreover, the importance of them may vary from one recommendation problem to another.

Random Walk with Restart (RWR) has been applied in many graph-based applications to calculate node proximity for recommendations (*e.g*, [4], [5], [6], [7], [8]). However, RWR is developed on univariate Markov chain for homogeneous graphs (See more details in Section II-A). As a generalization to univariate Markov chain, multivariate Markov chain (MMC) [9] is developed to model the random walk process in a heterogeneous graph [10], [11], [12]. MMC is able to explicitly model the influences between different entities in a well-interpretable way.

In our HeteRS, we employ MMC to calculate the node proximity w.r.t. some query nodes in the event-based heterogeneous graph for recommendations. However, existing MMC based methods need to manually set the influence weights between different types of entities, which is tedious and makes these methods less attractive when multiple types of entities exist, as in our case. To overcome this problem, we propose an optimization framework to automatically learn the influence weights. In particular, our learning scheme tries to find the optimal set of weights such that our model generates the appropriate ranking over recommended items to fit the training data. To the best of our knowledge, we are the first who comes up with the idea of parameter learning in a MMC-based model. In addition, since our model is based on MMC and it may encounter the efficiency problem, we design an approximation algorithm for better efficiency while achieving similar accuracy.

Our contributions in this paper can be summarized as follows:

- We propose a general model, namely HeteRS, to handle multiple recommendation problems in an event-based social network.
- To avoid the issue of manual parameter assignment, we propose a learning framework to find appropriate parameters for the model. Accordingly, the values of learned parameters indicate the importance of different types of entities in different recommendation tasks, giving us better understandings on user behavior in an event-based social network.
- We evaluate our proposed model through comprehensive experiments on real world datasets collected from Meetup.com. The experimental results demonstrate that our model outperforms the state-of-the-art baseline methods [1], [2], [3], which are separately developed for each of the three recommendation tasks, for every recommendation task.

The rest of the paper is organized as follows: after introducing related work in Section II, we present our HeteRS model. Then, in Section IV we describe our optimization method to learn model's parameters and subsequently introduce the approximation algorithm in Section V. The experimental results are shown in Section VI. Finally, we conclude this paper in Section VII.

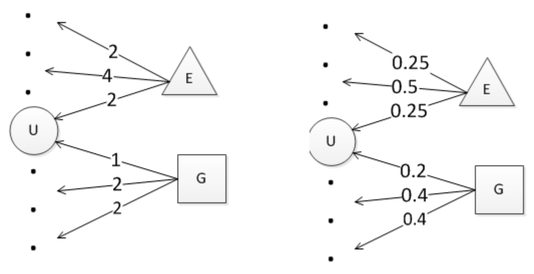
II. Related Work

We review related work on graph-based models used in recommender systems, and the problems of group-to-user recommendation, tag-to-group recommendation and event-to-user recommendation.

A. Graph-based Recommendations

Random Walk with Restart (RWR) is a widely-used graph based recommendation technique. Its main idea is to consider the recommendation problem as a node proximity evaluation problem w.r.t. query inputs. Although RWR is initially proposed for homogeneous graphs, it is also used for recommendation problems in heterogeneous graphs, *e.g.*, [4], [5], [6], [7]. When considering heterogeneous graphs, RWR usually projects them into homogeneous ones by treating all the nodes and edges as the same type. This leads to the neglects of differentiating influences between different types of entities. Although some methods try to address this issue by assigning different weights to edges of different types during the projection [5], [7], these weights are not well-explained in random walk. For example, considering the event recommendation in event-based social networks, assume that past events are twice as important as groups to influence users participating a new event. In RWR, one may want to model this by assigning twice weights to user-event edges than user-group edges as illustrated in Fig. 2. However, after the normalization step required by RWR, these weights lose their original meanings and may not play the expected roles as shown in Fig. 2. In other words, RWR cannot explicitly measure the influence from one type of entity to another, which makes it difficult to differentiate the influences between entities. More importantly, RWR does not provide a principled method to set these weights, which is the challenge.

MMC is developed to model random walk process in heterogeneous graphs [10], [11], [12], where influences between different types of entities can be explicitly modeled. However, in all previous MMC-based methods, the transition parameters between different types of entities are determined manually, which makes them impracticable when there are several types of entities, as in our case. Worse still, each recommendation task may require a distinct parameter setting as roles of types of entities are not the same across different problems.



(a) Before normalization (b) After normalization

Fig. 2: The effect of normalization on edge weights: a) Before normalization, the weight of edge U-E is twice as large as that of edge G-U; b) After normalization, two weights are similar.

In this paper, based on MMC, we propose a new learning scheme to automatically determine the influence weights between different types of entities. Note that our approach is different from the supervised RWR methods proposed in [13], [14], [15] in two aspects: 1) We focus on learning transition parameters in a MMC model while they aim to find the weight of every individual edge. 2) Supervised RWR methods focus on defining features for different types of edges, *e.g.*, the number of common friends for user-user edges [14] or semantic relatedness for tag-tag edges [13], and learning parameters for combining these features. In contrast, our method does not need such features and has a different focus. To the best of our knowledge, no previous work has provided a solution to learning parameters in a MMC-based model.

B. Recommendations in Event-based Social Networks

We review the existing work on the three recommendation tasks in event-based social networks. Since event-based social networks have just emerged recently, there are not many works on this type of social network.

Group-to-user recommendation problem aims to find groups which a user may be willing to join. Zhang et al. [1] proposed an extended factorization model, called PTARMIGAN, for recommending groups to users in an event-based social network. In their method, heterogeneous information such as locations, users' friendships and profile tags is exploited to learn the factorization model and recommendations are made based factorization results.

Tag-to-group recommendation is the problem to recommend tags for user groups. Liu et al. [2] assumed that group members have unequal roles when making a choice for groups. They introduced a topic model called PIT, to take the impact of group members into consideration, and applied it to recommend tags for Meetup groups.

Event-to-user recommendation aims to find the users who will join an upcoming event. Liu et al. [3] dealt with task of event recommendation in an event-based social network, although this is not the main focus of their work. Their simple solution is constructing a user graph based on users' friendship and propagating preferences based on RWR. Their experiments show that this approach beats baseline methods. However, only user relationship information is used in their method. In this paper, we would like to investigate whether we can enhance the recommendation further by taking more heterogeneous information into considerations.

III. Proposed Model

In this section, we present our proposed method called HeteRS. We first define a heterogeneous graph built from an event-based social network. Then, the graph is extended to incorporate implicit patterns of user behaviors from temporal aspect. Finally, we introduce how to accomplish multiple recommendation tasks based on the constructed graph.

A. Event-based Social Network Graph

In this subsection, we give the definition for our event-based social network graph.

Definition 1 (Event-based Social Network Graph). *Let* $U = \{u_1, u_2, ..., u_{|U|}\}$, $E = \{e_1, e_2, ..., e_{|E|}\}$, $G = \{g_1, g_2, ..., g_{|G|}\}$, $T = \{t_1, t_2, ..., t_{|T|}\}$ *and* $V = \{v_1, v_2, ..., v_{|V|}\}$ *be the set of users, events, groups, tags and venues, respectively, and* $\mathcal{R} = \{\langle U,E\rangle, \langle E,G\rangle, \langle E,V\rangle, \langle U,G\rangle, \langle U,T\rangle, \langle G,T\rangle\}$ *be the set of types of relations. The* ***Event-based Social Network Graph*** *is defined as a directed weighted graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$*, where the node set* $\mathcal{V} = U \cup E \cup G \cup T \cup V$ *and the edge set* $\mathcal{E} = \{\langle m,n\rangle | m \in M \wedge n \in N \wedge \{\langle M,N\rangle, \langle N,M\rangle\} \cap \mathcal{R} \neq \emptyset \wedge m$ *has a relation with* $n\}$*. Let* $\mathbf{A}_{MN} \in \mathbb{R}^{|N| \times |M|}$*, (*$\{\langle M,N\rangle, \langle N,M\rangle\} \cap \mathcal{R} \neq \emptyset$*), denote the adjacency matrix representing the relations (interactions) from nodes of type* M *to the ones of type* N*. Then* $\mathbf{A}_{MN}(n,m) = 1$*, if* $m \in M \wedge n \in N \wedge \langle m,n\rangle \in \mathcal{E}$*, and 0 otherwise.*

Note that we only have the adjacency matrices $\mathbf{A}_{MN}$ for every pair of types M and N that have a relation, *i.e.*, $\{\langle M,N\rangle, \langle N,M\rangle\} \cap \mathcal{R} \neq \emptyset$, and the adjacency matrices corresponding to other pairs of types do not exist, *e.g.*, $\mathbf{A}_{UV}$ and $\mathbf{A}_{VU}$.

The graph consists of the explicit information extracted from the event-based social network data; however, after analyzing the dataset, we found that user behaviors follow some hidden temporal patterns. These key observations motivate us to extend the graph to incorporate useful implicit information, which will be introduced in the next subsection.

B. Temporal Factors

Time plays an important role when users decide to join an event. Liu et al. [3] found that Meetup's event time is peaked at around 8pm on weekdays and distributed more evenly at weekends. This is because most of the events in Meetup are informal (*e.g.*, dining out, going to cinema), and users usually join events outside of working hours. To further investigate the temporal factors, we consider the time duration (in days) between two successive events of each user in New York City and then aggregate the durations of all users to depict the histogram in Fig. 3. We note that we have made similar observation on the data in other cities.

From Fig. 3, we have two key observations. First, there is a peak every 7 days in the histogram, *e.g.*, we observe peaks at 7, 14, 21, 28, etc. This means most of users take part in events with a weekly periodic schedule. Second, we

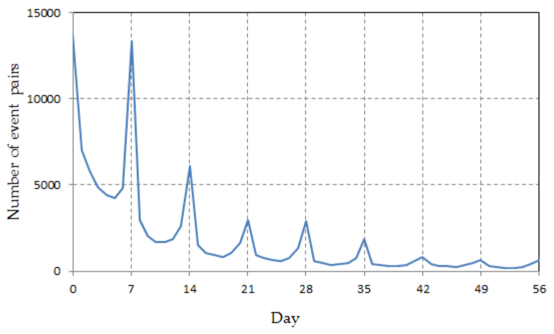


Fig. 3: Duration between two consecutive events of users in New York City

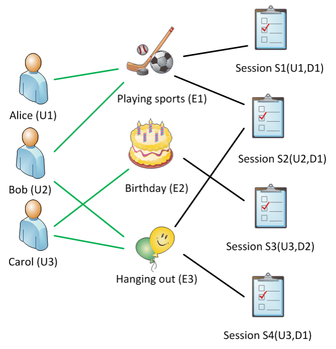


Fig. 4: An example of adding session nodes to the graph (other nodes are not shown)

observe that the number of event pairs deceases as the duration increases in Fig. 3. This means users mostly join two events of their interests in a short time period. Both observations reflect the implicit patterns of user behaviors in event-based social networks, which should be considered in our heterogeneous graph for better modeling the data.

To exploit the weekly periodic patterns, we introduce a new type of nodes, namely session node. In particular, if a user *u* has joined an event in day *d* of week (*e.g.*, Monday), a session node $s(u, d)$ is created and linked to that event. Apparently, a user may have several (at most 7) session nodes, each of which represents his/her activities in one certain day of week. Following this extension, in Fig. 1, assume that events e_1 and e_3 were at the same day d_1, and e_2 was held on day d_2, then for each participant of events we create corresponding session nodes as shown in Fig. 4. Our idea of creating session nodes is motivated by previous works: Xiang et al. [16] created session nodes for exploiting long-term and short-term temporal properties, and Yuan et al. [17] used session nodes to represent users' check-ins interest at different time. Unlike those works, our session nodes are employed to exploit the correlations between events, that is, we consider the events connected to the same session node are similar to each other, because they are shared by the same user at the same day of week. Note that we do not create one session node for a certain day of the week and link it to all events held at that day, because in such case a session node may be shared by many events that are totally unrelated to each other, which will result in noisy edges in our graph.

The short-time period behavior is incorporated by weighting edges. As mentioned above, there is a higher chance that a user participates in an event if he/she already joined an event lately. It means that recent events tend to be more important than earlier ones. To capture this, a decay function $f_t(tm_j)$ is used to define the weight of event *j* taking place at time tm_j:

$$f_t(tm_j) = \exp(-\eta(tm_c - tm_j)), \tag{1}$$

where $tm_c - tm_j$ is the time difference (in days) between the event *j* and the last event *c* in the dataset, and η is a parameter to control the decay rate over time. A larger η will give smaller weights for events in the distant past. This weighting scheme means the all the adjacency matrices related to events should be updated.

Overall, we define a new graph by extending EBSN to utilize the temporal effects on user behaviors.

Definition 2 (EBSN Graph with temporal effects)**.** *Let $S = \{s_1, s_2, ..., s_{|S|}\}$ be the set of session nodes. The* ***EBSN graph with temporal effects*** *$\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is an extension of EBSN graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as follows:*

- *$\mathcal{V}' = \mathcal{V} \cup S$, and $\mathcal{R}' = \mathcal{R} \cup \{\langle E, S\rangle\}$*
- *$\mathcal{E}' = \mathcal{E} \cup \{\langle e, s\rangle, \langle s, e\rangle | e \in E \wedge s \in S \wedge e$ belongs to $s\}$*
- *$\mathbf{A}_{NE}(e, n) = f_t(tm_e)$, if $\langle n, e\rangle \in \mathcal{E}'$, and 0 otherwise.*

When we mention EBSN graph hereinafter, it refers to the EBSN graph with temporal effects, if there is no explanation.

C. Recommendation on EBSN Graphs

We next describe how to perform a recommendation task on our graph. Our idea is to transform the recommendation problem into node proximity calculation problems w.r.t. some query nodes, and then use multivariate Markov chain to solve it. We choose the task of recommending groups to users as the example to introduce our method, and other recommendation problems can be solved in the similar way.

Definition 3 (Group-to-user recommendation)**.** *Given a user u,* ***group-to-user recommendation*** *is the problem to find groups that u is likely to join.*

Now we describe how to use EBSN graph to recommend groups for a user. As our method is based on MMC, we first define the transition matrices.

Definition 4 (Transition matrix)**.** *Transition matrix $\mathbf{P}_{MN}$ $(\{\langle M, N\rangle, \langle N, M\rangle\} \cap \mathcal{R}' \neq \emptyset)$ is obtained by normalizing adjacency matrices $\mathbf{A}_{MN}$ by columns. $\mathbf{P}_{MN}$ handles dangling nodes, i.e., $\mathbf{P}_{MN}(n, m) = 1/|N|, \forall n$ for each m such that $\sum_n^{|N|} \mathbf{A}_{MN}(n, m) = 0$.*

For EBSN graph, there are multiple transition matrices, each of which corresponds to a relation from one to another type of entities. Next, we consider how to construct query vector for relevant node given a user.

Definition 5 (User query vector)**.** *Given a user u , we define the* ***user query vector*** *as $\mathbf{q}_u \in \mathbb{R}^{|U|}$, where $\mathbf{q}_u(i) = 1$ if $i = u$, 0 otherwise.*

For tag-to-group recommendation, we consider to recommend tags given a group g, and the query vector $\mathbf{q}_g \in \mathbb{R}^{|G|}$ is defined similarly. In event-to-user recommendation, given an upcoming event, we aim to find the potential users who are interested in it. Following [3], we assume that we know some seed users who first responded to join the event. Moreover, when a new event is published in EBSN, the corresponding time, venue and group are always accompanied. Therefore, we have multiple query vectors, instead of one as in other two recommendation problems, $\mathbf{q}_u \in \mathbb{R}^{|U|}$, $\mathbf{q}_g \in \mathbb{R}^{|G|}$, $\mathbf{q}_v \in \mathbb{R}^{|V|}$ and $\mathbf{q}_s \in \mathbb{R}^{|S|}$ corresponding to input event's seed users, group, venue and user sessions, where user sessions are constructed from seed users and day of the event. Note that all the query vectors must be probability vectors, *i.e.*, the sum of each vector is 1.

With the transition matrices and user query vector, we are able to simulate a random walk process by using MMC on the heterogeneous graph to calculate the proximity of nodes. We establish the following equations:

$$\begin{aligned}\mathbf{u}^{(t+1)} &= \alpha_{EU}\mathbf{P}_{EU}\mathbf{e}^{(t)} + \alpha_{GU}\mathbf{P}_{GU}\mathbf{g}^{(t)} \\ &\quad + \alpha_{TU}\mathbf{P}_{TU}\mathbf{t}^{(t)} + (1-\alpha_{EU}-\alpha_{GU}-\alpha_{TU})\mathbf{q}_u \quad (2)\\ \mathbf{e}^{(t+1)} &= \alpha_{UE}\mathbf{P}_{UE}\mathbf{u}^{(t)} + \alpha_{GE}\mathbf{P}_{GE}\mathbf{g}^{(t)} \\ &\quad + \alpha_{VE}\mathbf{P}_{VE}\mathbf{v}^{(t)} + (1-\alpha_{UE}-\alpha_{GE}-\alpha_{VE})\mathbf{P}_{SE}\mathbf{s}^{(t)} \quad (3)\\ \mathbf{g}^{(t+1)} &= \alpha_{EG}\mathbf{P}_{EG}\mathbf{e}^{(t)} + \alpha_{UG}\mathbf{P}_{UG}\mathbf{u}^{(t)} \\ &\quad + (1-\alpha_{EG}-\alpha_{UG})\mathbf{P}_{TG}\mathbf{t}^{(t)} \quad (4)\\ \mathbf{t}^{(t+1)} &= \alpha_{UT}\mathbf{P}_{UT}\mathbf{u}^{(t)} + (1-\alpha_{UT})\mathbf{P}_{GT}\mathbf{g}^{(t)} \quad (5)\\ \mathbf{s}^{(t+1)} &= \mathbf{P}_{ES}\mathbf{e}^{(t)} \quad (6)\\ \mathbf{v}^{(t+1)} &= \mathbf{P}_{EV}\mathbf{e}^{(t)} \quad (7)\end{aligned}$$

where

- $\mathbf{u}^{(t)}, \mathbf{e}^{(t)}, \mathbf{g}^{(t)}, \mathbf{t}^{(t)}, \mathbf{s}^{(t)}, \mathbf{v}^{(t)}$ are distribution probability vectors representing the probabilities that users, events, groups, sessions and venues are visited at time t, respectively.
- α_{MN} ($\{\langle M,N\rangle, \langle N,M\rangle\} \cap \mathcal{R}' \neq \emptyset$, $\alpha_{MN} > 0$ and $\sum_M \alpha_{MN} \leq 1$) denotes the transition weight (or influence weight) from nodes of type M to nodes of type N.

Equations (2)-(7) model how the probabilities change for different types of nodes after each step of random walk transition on our EBSN graph. We can see from the equations that α_{MN} explicitly controls how much probability (influence) one type of nodes receive from the other types of nodes. For example, in Eq. (2), user nodes receive α_{EU} probability from event nodes, and α_{GU} probability from group nodes, α_{TU} probability from tag nodes and $(1-\alpha_{EU}-\alpha_{GU}-\alpha_{TU})$ from query user node. The explicit controllability benefits from the fact that $\mathbf{P}_{EU}$, $\mathbf{P}_{GU}$ and $\mathbf{P}_{TU}$ are transition matrices and $\mathbf{e}^{(t)}$, $\mathbf{g}^{(t)}$ and $\mathbf{t}^{(t)}$ are probability distribution vectors, and as a result their products are still probability distribution vectors. This explicit controllability is not only useful to explain the model, but important to reveal the roles of different entities in different recommendation tasks after learning α_{MN}. However, if we project the graph into homogeneous graph and apply traditional univariate random walk, as shown in Section II, we cannot obtain the explicit controllability.

After solving the stationary probability vectors from Eqs. (2)-(7), the recommendations can be made by ranking the groups based on $\mathbf{g}$. We note that, without considering the query inputs, the model degenerates into the original MMC in [9]. The query vectors are introduced into the model to produce "personalized" results, the idea of which is similar to RWR. When considering tag-to-group recommendation, we incorporate the group query vector $\mathbf{q}_g$ into Eq. (4) and remove the user query vector $\mathbf{q}_u$ from Eq. (2). For event-to-user recommendation, we can perform similar modifications to incorporate query vectors $\mathbf{q}_u, \mathbf{q}_g, \mathbf{q}_v, \mathbf{q}_s$ into Eqs. (2)-(7).

Theorem 1. *If the EBSN graph is connected, probability vectors in Eqs.* (2)-(7) *uniquely converge to stationary vectors as t goes to infinity.*

Proof: First, we rewrite Eqs. (2)-(7) as follows:

$$\mathbf{r}^{(t+1)} = \mathbf{M}\mathbf{r}^{(t)} + \mathbf{q} \tag{8}$$

where:

$$\mathbf{r} = \left(\mathbf{u}^\top, \mathbf{e}^\top, \mathbf{g}^\top, \mathbf{t}^\top, \mathbf{s}^\top, \mathbf{v}^\top\right)^\top \tag{9}$$

$$\mathbf{M} = \begin{pmatrix} \mathbf{0} & \alpha_{EU}\mathbf{P}_{EU} & \cdots & \mathbf{0} \\ \alpha_{UE}\mathbf{P}_{UE} & \mathbf{0} & \cdots & \alpha_{VE}\mathbf{P}_{VE} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{P}_{EV} & \cdots & \mathbf{0} \end{pmatrix} \tag{10}$$

$$\begin{aligned}\mathbf{q} &= \left(\alpha_{UU}\mathbf{q}_u^\top, \mathbf{0}^\top, \mathbf{0}^\top, \mathbf{0}^\top, \mathbf{0}^\top, \mathbf{0}^\top\right)^\top \quad (11)\\ \alpha_{UU} &= 1-\alpha_{EU}-\alpha_{GU}-\alpha_{TU}\\ \alpha_{SE} &= 1-\alpha_{UE}-\alpha_{GE}-\alpha_{VE}\\ \alpha_{TG} &= 1-\alpha_{EG}-\alpha_{UG}\\ \alpha_{GT} &= 1-\alpha_{UT}\end{aligned}$$

Note that $\mathbf{M}$ is not a stochastic matrix. Let Λ be the matrix containing parameters from $\mathbf{M}$:

$$\Lambda = \begin{pmatrix} 0 & \alpha_{EU} & \cdots & 0 \\ \alpha_{UE} & 0 & \cdots & \alpha_{VE} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 \end{pmatrix}$$

We can see that Λ is a sub-stochastic matrix, and thus the *spectral radius* (maximum of the absolute eigenvalues) of Λ is strictly smaller than 1, denoted by $\rho(\Lambda) = |\lambda| < 1$. We can also know $\lambda > 0$ because Λ is a nonnegative and irreducible matrix (there is always a path from one type to another type of entities, so Λ is irreducible). Then by Perron-Frobenius Theorem, there exists a positive vector $\mathbf{z} = (z_U, z_E, z_G, z_T, z_S, z_V)^\top$ such that $\mathbf{z}^\top\Lambda = \lambda\mathbf{z}^\top$. We note that $\mathbf{1}_{|N|}^\top\mathbf{P}_{MN} = \mathbf{1}_{|M|}^\top$ where $\mathbf{1}_{|M|}$ is the vector with size $|M|$ of all ones. Then it is easy to show that $(z_U\mathbf{1}_{|U|}^\top, z_E\mathbf{1}_{|E|}^\top, \ldots, z_V\mathbf{1}_{|V|}^\top)\mathbf{M} = \lambda(z_U\mathbf{1}_{|U|}^\top, z_E\mathbf{1}_{|E|}^\top, \ldots, z_V\mathbf{1}_{|V|}^\top)$, hence λ is an eigenvalue of $\mathbf{M}$ and its corresponding eigenvector is positive. Since the EBSN graph is connected, $\mathbf{M}$ is a nonnegative and irreducible matrix. Based on Perron-Frobenius Theorem, we know that only eigenvectors associated with the spectral radius of a nonnegative and irreducible matrix are positive, and thus we have $\rho(\mathbf{M}) = \lambda$.

Suppose $\mathbf{r}^{(0)} = \pi$, we have $\mathbf{r}^{(1)} = \mathbf{M}\pi + \mathbf{q}, \mathbf{r}^{(2)} = \mathbf{M}^2\pi + \mathbf{M}\mathbf{q} + \mathbf{q}, \ldots, \mathbf{r}^{(t)} = \mathbf{M}^t\pi + \sum_{k=0}^{t-1}\mathbf{M}^k\mathbf{q}$. Since $\rho(M) = \lambda < 1$, we have $\lim_{t\to\infty}\mathbf{M}^t = \mathbf{0}$ and $\lim_{t\to\infty}\sum_{k=0}^{t-1}\mathbf{M}^k = (\mathbf{I}-\mathbf{M})^{-1}$. So $\mathbf{r}^{(t)}$ finally converges to $\mathbf{r}^* = (\mathbf{I}-\mathbf{M})^{-1}\mathbf{q}$.

■

Generally, our constructed graph may not be connected. However, there usually exist dangling nodes in our graph, for example, considering the user-group interaction, if a user does not join any group, he/she is then a dangling node. According to definition 4, we consider under this circumstance that he/she is connected to all group nodes. This operator for dangling nodes usually makes our EBSN graph be connected. Therefore we can obtain the stationary probability vectors by iteratively running Eqs. (2)-(7) until it converges.

IV. Optimization Approach for Parameter Learning

From Eqs. (2)-(7), we can see the transition weights α_{MN} play an important role in producing the final solution, as they control how much probability one type of entities receives from others. In this section, we consider to learn them from training data. To this end, we first design an objective function on α_{MN}, and then propose a learning algorithm to optimize it. Again, we use group-to-user recommendation task as the example to illustrate our learning method, and for other recommendation problems the learning process is similar.

A. Objective Function

We follow the Bayesian Personalized Ranking (BPR) optimization framework [18] to construct our objective function. In group-to-user recommendation, given a user, we consider groups which he/she already joined as positive groups, denoted by the set PG, while the ones that the user did not join as negative groups, denoted by NG. In other words, the whole group set consists of two parts: $G = PG \cup NG$. Then the appropriate parameters α_{MN} of Eqs. (2)-(7) should rank all the positive groups higher than negative groups, *i.e.*, positive groups should have higher probability than negative ones. To model this, we design the following AUC (Area Under the ROC Curve) objective to be maximized, where we assume there are *m* instances $\{< u_k, PG_k >\}_{k=1}^{m}$:

$$\max_{\boldsymbol{\alpha}} Obj(\boldsymbol{\alpha}) = \sum_{k=1}^{m} \frac{\sum_{i\in PG_k}\sum_{j\in NG_k}\mathbb{1}(\mathbf{g}(i)-\mathbf{g}(j))}{|PG_k|\,|NG_k|} \ , \tag{12}$$

where $NG_k = G - PG_k$ and $\mathbb{1}(.)$ is an indicator function that equals to 1 if $\mathbf{g}(i) > \mathbf{g}(j)$, and 0 otherwise. As the indicator function $\mathbb{1}(.)$ is not differentiable, it is usually approximated by sigmoid function:

$$\sigma(x;\beta) = \frac{1}{1+e^{-\beta x}} \ , \tag{13}$$

where the parameter β controls the approximate error, and is set empirically during the training process. Substituting it into Eq. (12) and considering in log form as in [1], we obtain a new objective function as follows:

$$\max_{\boldsymbol{\alpha}} Obj(\boldsymbol{\alpha}) = \sum_{k=1}^{m} \frac{\sum_{i\in PG_k}\sum_{j\in NG_k}\ln\sigma(\mathbf{g}(i)-\mathbf{g}(j))}{|PG_k|\,|NG_k|} \ . \tag{14}$$

B. Solving the Optimization Problem

To find parameters $\boldsymbol{\alpha}$ that maximize the objective function in Eq. (14), we adopt stochastic gradient descent (SGD) algorithm. In SGD, the parameters are step by step updated based on one training instance, instead of all training data as in gradient descent. Therefore, SGD is more suitable when the training data is large. Specifically, for each training instance, its derivative is calculated and the parameters $\boldsymbol{\alpha}$ are updated by moving along the ascending direction of the gradient as follows:

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + lr\frac{\partial Obj_k(\boldsymbol{\alpha})}{\partial\boldsymbol{\alpha}} \ , \tag{15}$$

where $Obj_k(\boldsymbol{\alpha})$ is the objective function for *k*-th training instance:

$$Obj_k(\boldsymbol{\alpha}) = \frac{\sum_{i\in PG_k}\sum_{j\in NG_k}\ln\sigma(\mathbf{g}(i)-\mathbf{g}(j))}{|PG_k|\,|NG_k|} \ . \tag{16}$$

Accordingly, we have the partial derivatives of $Obj_k(.)$ w.r.t. $\boldsymbol{\alpha}$ as:

$$\frac{\partial Obj_k(\boldsymbol{\alpha})}{\partial\boldsymbol{\alpha}} = \frac{\sum_{i\in PG_k}\sum_{j\in NG_k}\frac{\partial\ln\sigma(\delta_{ij})}{\partial\delta_{ij}}\left(\frac{\partial\mathbf{g}(i)}{\partial\boldsymbol{\alpha}} - \frac{\partial\mathbf{g}(j)}{\partial\boldsymbol{\alpha}}\right)}{|PG_k|\,|NG_k|} \ , \tag{17}$$

where $\delta_{ij} = \mathbf{g}(i) - \mathbf{g}(j)$. From Eq. (13), we easily have $\partial\ln\sigma(\delta_{ij})/\partial\delta_{ij} = \beta(1-\sigma(\delta_{ij}))$. In the implementation, we set $\partial\ln\sigma(\delta_{ij})/\partial\delta_{ij} = (1-\sigma(\delta_{ij}))$ because β can be integrated into learning rate *lr*.

The remaining issue is how to compute the derivative $\partial\mathbf{g}(i)/\partial\boldsymbol{\alpha}$. Taking the derivatives w.r.t. each parameter α_{MN} on the both sides of Eq. (4), we can get:

$$\frac{\partial\mathbf{g}}{\partial\alpha_{EG}} = \mathbf{P}_{EG}\mathbf{e} - \mathbf{P}_{TG}\mathbf{t} \tag{18}$$

$$\begin{aligned}\frac{\partial\mathbf{g}}{\partial\alpha_{UE}} &= \alpha_{EG}\mathbf{P}_{EG}\frac{\partial\mathbf{e}}{\partial\alpha_{UE}} \\ &= \alpha_{EG}\mathbf{P}_{EG}(\mathbf{P}_{UE}\mathbf{u} - \mathbf{P}_{SE}\mathbf{s}) \\ &\propto \mathbf{P}_{EG}(\mathbf{P}_{UE}\mathbf{u} - \mathbf{P}_{SE}\mathbf{s})\end{aligned} \tag{19}$$

$$\begin{aligned}\frac{\partial\mathbf{g}}{\partial\alpha_{EU}} &= \alpha_{UG}\mathbf{P}_{UG}\frac{\partial\mathbf{u}}{\partial\alpha_{EU}} \\ &= \alpha_{GU}\mathbf{P}_{UG}(\mathbf{P}_{EU}\mathbf{e} - \mathbf{q}_u) \\ &\propto \mathbf{P}_{UG}(\mathbf{P}_{EU}\mathbf{e} - \mathbf{q}_u)\end{aligned} \tag{20}$$

$$\begin{aligned}\frac{\partial\mathbf{g}}{\partial\alpha_{UT}} &= \alpha_{TG}\mathbf{P}_{TG}\frac{\partial\mathbf{t}}{\partial\alpha_{UT}} \\ &= \alpha_{TG}\mathbf{P}_{TG}(\mathbf{P}_{UT}\mathbf{u} - \mathbf{P}_{GT}\mathbf{g}) \\ &\propto \mathbf{P}_{TG}(\mathbf{P}_{UT}\mathbf{u} - \mathbf{P}_{GT}\mathbf{g})\end{aligned} \tag{21}$$

Due to the space limitation, we only show derivatives $\partial\mathbf{g}/\partial\boldsymbol{\alpha}$ w.r.t. some parameters α_{MN}; however, the remaining ones can be derived in the similar way. Note that, by definition, parameters $\boldsymbol{\alpha}$ must be positive, so we set a lower bound γ for all transition parameters $\boldsymbol{\alpha}$. Whenever a parameter α_{MN} becomes smaller than γ, which is 0.01 in our implementation, its value is set to γ. Moreover, when the sum of all transition parameters α_{MN} to type N is not 1, we normalize those parameters so that their sum equals to 1.

Based on the gradients derived above, we can learn the optimal parameters $\boldsymbol{\alpha}$, and the learning scheme is summarized in Algorithm 1.

Algorithm 1: Learning process

```
Input: $m$ training instances, and learning rate $lr$
Output: optimal $\boldsymbol{\alpha}$
begin
    t = 0;
    Initialize $\boldsymbol{\alpha}^{(0)}$;
    while Obj($\boldsymbol{\alpha}$) has not converged do
        Randomly shuffle the $m$ training instances;
        foreach training instance $k$ do
            Compute stationary vectors $\mathbf{u}, \mathbf{e}, \mathbf{g}, \mathbf{t}, \mathbf{s}, \mathbf{v}$ by
            iteratively executing Eqs. (2)-(7);
            Compute $\partial \mathbf{g} / \partial \boldsymbol{\alpha}$ based on Eqs. (18)-(21);
            Update $\boldsymbol{\alpha}$ : $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + lr \frac{\partial Obj_k(\boldsymbol{\alpha}^{(t)})}{\partial \boldsymbol{\alpha}}$;
            t = t+1;
```

V. Approximation Algorithm

Another major issue that HeteRS may encounter is efficiency. To make a recommendation, our model needs to iteratively execute Eqs. (2)-(7) until probability vectors converge. The time complexity of this process is $O(tn)$, where n is number of edges (represented by nonzero values in transition matrices) and t is number of iterations. Obviously, when the graph becomes too large, the computation is expensive, which is a common problem of random walk based approaches. To overcome this problem, we propose an approximation algorithm for HeteRS, which could yield much better efficiency without sacrificing much accuracy.

In Theorem 1, we prove that $\mathbf{r}^{(t)} = \mathbf{M}^t \pi + \sum_{k=0}^{t-1} \mathbf{M}^k \mathbf{q} = \sum_{k=0}^{t-1} \mathbf{M}^k \mathbf{q}$. In other words,

$$\mathbf{r} = (\mathbf{I} + \mathbf{M} + \mathbf{M}^2 + \mathbf{M}^3 + \dots)\mathbf{q} \tag{22}$$

Equation (22) gives us an idea to design an approximation algorithm. Since $\mathbf{p}^{(k)} = \mathbf{M}^k \mathbf{q}$ represents the preferences of all nodes in the graph after k steps of preference propagation starting from query nodes, and $\mathbf{p}^{(k)}$ can be computed directly from $\mathbf{p}^{(k-1)}$ ($\mathbf{p}^{(k)} = \mathbf{M}\mathbf{p}^{(k-1)}$), we may simulate the process of propagating preferences over our constructed graph to estimate $\mathbf{r}$, which is shown in Algorithm 2.

Particularly, we maintain a queue for nodes to be visited, and it initially contains query nodes (line 4). Then, each time a node i is taken from the queue, we send to each of i's neighbors j an amount of preference $\mathbf{M}(j,i) \times \mathbf{p}(i)$, where $\mathbf{p}(i)$ is current preference of i (line 10-11), and put j into the queue if it is not in (line 13-14). Each time a node j receives preference from other node, we update the accumulating preference vector $\mathbf{r}$ in the corresponding position so that $\mathbf{r}(j)$ contains the total amount of preference that j receives during the process (line 12). When the preference of a node is below a threshold ϵ, we ignore this node and do not send any value to its neighbors (line 8). The algorithm is terminated when there is no any node to be visited, *i.e.*, the queue is empty, and $\mathbf{r}$ is returned as the result.

Compared to the iterative algorithm, the approximation one is more efficient because iterative method takes into account all nodes, most of which are irrelevant nodes, while approximation

Algorithm 2: Approximation algorithm

```
Input: Matrix $\mathbf{M}$ and query nodes $\{q_i\}$
Output: Score vector $\mathbf{r}$
begin
1       Initialize $\mathbf{q}$ for query nodes $\{q_i\}$ as in Eq. (11);
2       $\mathbf{r} = \mathbf{q}$; \\accumulating preference vector;
3       $\mathbf{p} = \mathbf{q}$; \\current preference vector;
4       $\mathbf{Q} := \{q_i\}$; \\queue of nodes to be visited;
5       while $\mathbf{Q}$ is not empty do
6           $i$ = $\mathbf{Q}$.pop();
7           if $\mathbf{p}(i) < \epsilon$ then
8               continue;
9           foreach $j$ of $i$'s neighbors do
10              $w = \mathbf{M}(j,i) \times \mathbf{p}(i)$;
11              $\mathbf{p}(j) = \mathbf{p}(j) + w$;
12              $\mathbf{r}(j) = \mathbf{r}(j) + w$;
13              if $j \notin \mathbf{Q}$ then
14                  $\mathbf{Q}$.push($j$);
15          $\mathbf{p}(i) = 0$;
16      return $\mathbf{r}$;
```

algorithm exploits local propagation, and it rarely goes too far from query nodes. Through our experiments, it is shown that the approximation algorithm is significantly faster than iterative method while achieving similar accuracy.

Note that although our approximation algorithm is similar to the Particle Filtering heuristic proposed in [19], our algorithm is motivated from the analysis of theoretical solution in Eq. (22), where we cutoff small terms according to Eq. (22) to obtain an approximation solution, and Particle Filtering is derived from Monte Carlo simulation of random walk.

VI. Experiments

A. Dataset Description

We crawl meetup.com to construct two datasets for New York City (NYC) and state of California (CA) respectively during the period from Jan. 1st to Dec. 31st 2012. For preprocessing, we keep users who joined at least 5 events, events with at least 5 participants, and groups with more than 20 events. The dataset statistics for both regions are summarized in Table I[2].

Subsequently, we randomly select 20% groups of each user and 20% tags of each group as test sets for the tasks of group-to-user and tag-to-group recommendation, respectively. The remaining data will be used to form our EBSN graph. To learn the parameters for HeteRS in group-to-user recommendation problem, we further select 10% groups from each user as tuning set and remove corresponding user-group edges from the EBSN graph. The tuning set for tag-to-group recommendation is built in the same way. For event-to-user recommendation, the construction is different because events are time related. In particular, we remove events in the last 3 months of training set, *i.e.*, Oct. 1st - Dec. 31st, to build the tuning set. For the test set, we extract from the data events held after training

[2]The datasets are available at http://www.ntu.edu.sg/home/gaocong/datacode.htm

TABLE I: DATASET STATISTICS

Dataset	NYC	CA
# Events	9,549	15,588
# Users	46,895	59,989
# Groups	398	631
# Tags	15,819	21,228
# Venues	2,396	4,507
Avg. Events per month	795.75	1299
Avg. Participants for an event	17.65	13.68
Avg. Events for a user	3.59	3.55
Avg. Members for a group	382.71	256.76
Avg. Groups joined by a user	3.16	2.74
Avg. Tags for a group	10.62	10.17
Avg. Tags for a user	14.44	18.42

set's last timestamp (Dec 31st 2012) and joined by at least 5 users in the training set. We use events in first month (1st-month test data) to compare the performances of different event-to-user recommendation methods, and use test data sets in different months, *e.g.*, 1st-month, 3rd-month and 6th-month test data, to illustrate the effect of temporal factor on user's event participation behaviors.

B. Evaluation Metrics

For evaluation, we use two metrics: precision and recall, denoted by p@N and r@N, respectively, where N is number of recommendation results. In particular, the precision and recall of each test case are computed and the overall precision and recall are obtained by averaging these scores of all test cases.

C. Baseline Methods

Besides common baseline methods, we compare HeteRS with the state-of-the-art methods developed for each of the three recommendation problems.

Group-to-user recommendation and *Tag-to-group recommendation*:

- **CF**: This is user-based Collaborative Filtering method. For group-to-user recommendation, the user similarity is calculated based on the matrix $\mathbf{A}_{GU}$. Similarly, for tag-to-group, the group similarity is based on $\mathbf{A}_{TG}$.
- **BPR**: The Bayesian Personalized Ranking [18] based matrix factorization method is performed on matrices $\mathbf{A}_{GU}$ and $\mathbf{A}_{TG}$, respectively for the two tasks, and recommendations are made based on the factorization results.
- **RWR**: For group-to-user recommendation, Random Walk with Restart is run on a group-group interaction graph weighted by number of common users, where the groups of each test user are treated as query nodes. The tag-to-group recommendation is performed similarly.
- **PTARMIGAN**: As discussed in Section II, this is the state-of-the-art method [1] for group-to-user recommendation.
- **PIT**: As discussed in Section II, this is the state-of-the-art method [2] for tag-to-group recommendation.

Event-to-user recommendation:

- **CF**: CF calculates the similarity between a candidate user and the given event's seed users based on $\mathbf{A}_{UE}$, then returns top-N most similar users as results.
- **BPR**: Based on BPR criteria, matrix factorization method is performed on a user-user matrix constructed by the number of common events between users. Similarity between two users is computed by using their latent features.
- **RWR**: This method is the state-of-the-art for this task [3], which performs RWR on a weighted user-user graph to make recommendation, and seed users are used as query nodes.
- **tRWR**: This is a variation of RWR where the event time information is incorporated into edges using Eq. (1).

Finally, we consider two other methods derived from our proposed model:

- **full_RWR**: This method performs RWR on our constructed heterogeneous graph, and it treats the different types of edges (and nodes) in the same way.
- **uni_HeteRS**: This method is HeteRS without parameter learning part. In this method, parameters in the same equation, *i.e.* $\{\alpha_{iN}\}_{\forall i \in \{U,E,G,T,V,S\}}$, are assigned with values uniformly. We compare with this variation of our method to evaluate the effectiveness of our learning process.

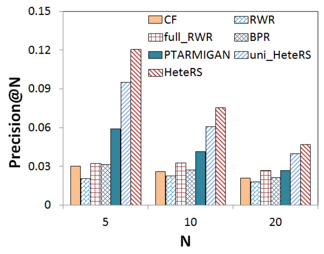
D. Parameter Settings

There are three parameters in HeteRS: time-decay η, error approximation β and learning rate lr. Time-decay parameter η defines the decrement rate of an event's importance in terms of their time spans to now, and is set to 0.03 in all experiments. Parameter β in Eq. (13) controls the error of approximating $\mathbb{1}(\cdot)$. The bigger β is, the better Eq. (13) approximates the indicator function. However, when β gets too large, the gradient at 0 tends to be infinity, which causes a numerical problem. When β gets too small, Eq. (13) fails to approximate the indicator function, and maximizing Eq. (14) would not lead to the optimization of the AUC. One suggestion is to set the value proportional to the total number of items to be recommended, because more items always mean bigger $|PG||NG|$ and the gradient Eq. (17) is divided by $|PG||NG|$. In such case, using bigger β is more reasonable. By using tuning set, we set β to 10^3 for event-to-user and group-to-user recommendation, and set β to 10^4 for tag-to-group recommendation. Learning rate lr relies on β, and is used to balance between convergence and the speed of learning. We set lr to 0.1 for event-to-user and group-to-user recommendation and to 1 for tag-to-group recommendation. For each recommendation task, we train our model with 50 iterations, as we observe that after 30-35 iterations the objective function becomes stable.

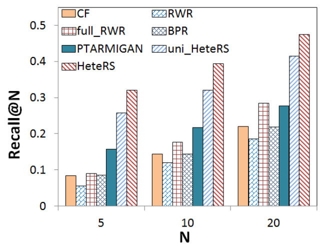
All the parameters of baseline methods are empirically set to optimal values on the tuning sets.

E. Group-to-user Recommendation

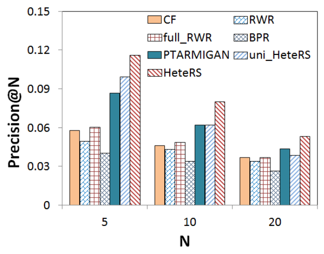
Figure 5 shows the results on group-to-user recommendation problem. In this problem, the input of each test is a user. We observe that our method HeteRS outperforms state-of-the-art PTARMIGAN, which also exploits heterogeneous information, significantly on both datasets. For example, HeteRS outperforms PTARMIGAN by 34% and 39% for p@5 and r@5, respectively, on CA dataset, and the improvement is statistical significant (p-value < 0.01). This may be because HeteRS represents the interactions between different entities



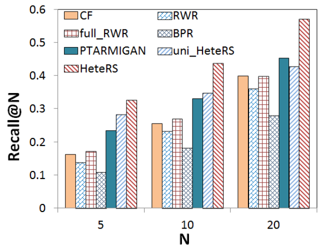
(a) Precision@N - NYC



(b) Recall@N - NYC

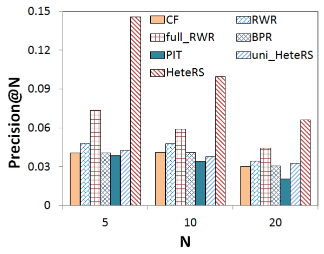


(c) Precision@N - CA

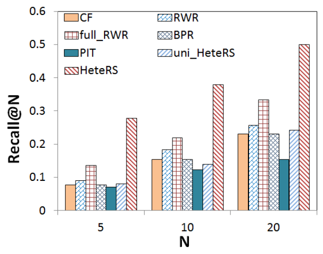


(d) Recall@N - CA

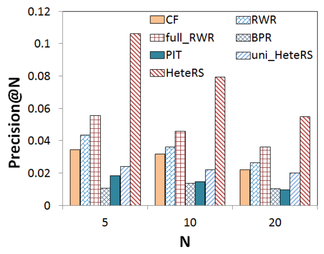
Fig. 5: Performance of methods in group-to-user recommendation



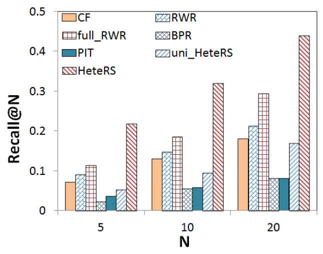
(a) Precision@N - NYC



(b) Recall@N - NYC

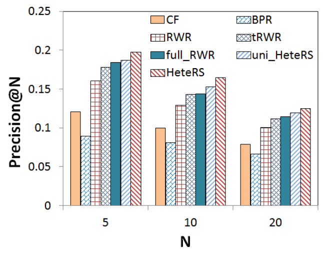


(c) Precision@N - CA

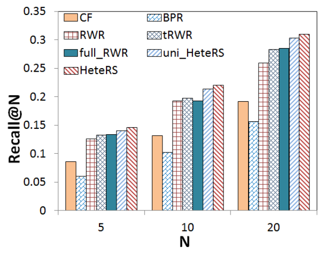


(d) Recall@N - CA

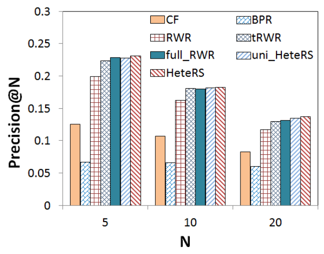
Fig. 6: Performance of methods in tag-to-group recommendation



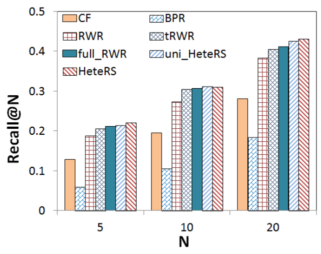
(a) Precision@N - NYC



(b) Recall@N - NYC



(c) Precision@N - CA



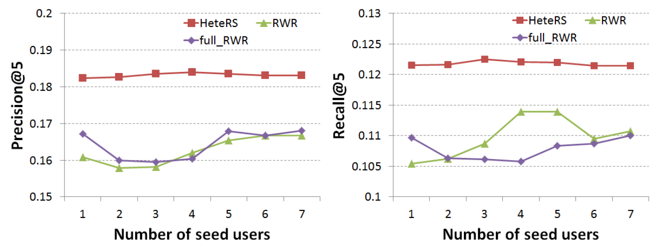
(d) Recall@N - CA

Fig. 7: Performance of methods in event-to-user recommendation (1st-month test set)

better than the linear latent factor model in PTARMIGAN. Moreover, HeteRS is significantly better than full_RWR on both datasets, which demonstrates the effectiveness of our model on heterogeneous graphs. On the other hand, HeteRS always achieves better results than uni_HeteRS, which means that our learning scheme is necessary for our model to obtain the better performance. Finally, we can see that full_RWR outperforms RWR on both datasets, because the former is performed on a heterogeneous graph. The performance of CF and BPR is also poor, and comparing all approaches we find the methods that can take advantage of additional information, *e.g.* tags, events, etc., are able to produce better results than others.

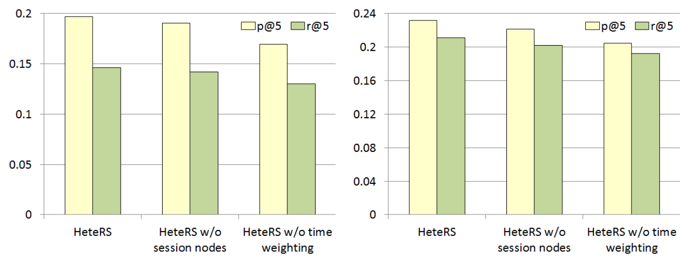
F. Tag-to-group Recommendation

This task takes a group as input and returns the most likely tags the group may use. Figure 6 shows the results of different methods for this recommendation problem. From the figure, we observe that HeteRS outperforms the state-of-the-art baseline PIT and other methods, by at least 90% on both datasets in terms of p@5 and the p-value of t-test is smaller than 0.01. This could be because HeteRS benefits from exploiting additional information including tags from group members or from participants of group events, while the other baseline methods except for full_RWR do not. This also explains why full_RWR yields good performance compared to the other baselines since full_RWR also uses all the different types of information to make recommendations. We also observe that PIT performs worse than other baselines. It could be because groups in Meetup are usually large, in which most users have limited historical data (*e.g.*, only participate in one group). Therefore, the impacts learned from PIT for these users are not reliable, which results in the poor performance of PIT. The results of uni_HeteRS are significantly worse than those of HeteRS and even worse than the baseline method full_RWR. This means that if we do not carefully set parameters for our model, its performance could be diminished dramatically.



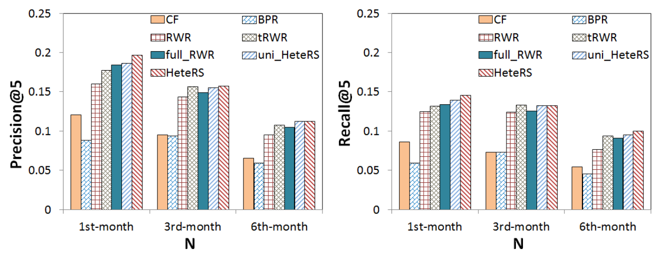
(a) Precision@5 - NYC (b) Recall@5 - NYC

Fig. 8: Performance of three event-to-user recommendation methods RWR, full_RWR and HeteRS with different number of seed users on NYC dataset



(a) NYC (b) CA

Fig. 10: Performance of HeteRS and its variants in event-to-user recommendation task



(a) Precision@5 - NYC (b) Recall@5 - NYC

Fig. 9: Performance of event-to-user recommendation methods on test sets in different months on NYC dataset

G. *Event-to-user Recommendation*

The seed users, group, venue and day of the week of each test event are taken as inputs, and users who are interested in the event are targets.

1) Recommendation Results: The results are shown in Fig. 7. In this experiment, the number of seed users is set to 5. We observe from this figure that, the state-of-the-art baseline RWR always has better performance than CF and BPR. On the other hand, tRWR, full_RWR, uni_HeteRS and HeteRS always outperform CF, BPR and RWR, because they take the events' importance into account while the others do not. In particular, HeteRS outperforms RWR by 23% and 16% on NYC and CA data, respectively, in terms of p@5. Moreover, HeteRS always achieves better results than tRWR, which cannot exploit the periodical behavior patterns of users as HeteRS does. The effect of this factor is analyzed in more detail in Section VI-H. In this experiment, full_RWR performs well, which again confirms the benefits of using additional information to produce better recommendation results. On NYC dataset, HeteRS outperforms the second best method full_RWR with p-value < 0.05, which is statistically significant. However, the difference between two methods on CA dataset is less significant, which could be because user behaviors in CA are less periodical than in NYC (more details in Section VI-H).

2) Number of seed users: In this experiment, we investigate how the number of seed users affects the recommendation results. Particularly, we vary the number of seed users for each event from 1 to 7 and test the performances of three recommendation methods, namely RWR, full_RWR and HeteRS. To use the same test set for all cases, we remove first 7 users of each user and keep the remaining users as the test set. The results of p@5 and r@5 on NYC dataset are shown in Fig 8. As we can see, the performance of HeteRS is almost unchanged for different numbers of seed users, *i.e.*, HeteRS is not sensitive to the number of seed users. In contrast, the results of the other two methods change considerably and they perform the best in 5-seed user case.

3) Temporal effect: This set of experiments is to study the temporal effect on event-to-user recommendation.

First, we report p@5 and r@5 on the 1st-, 3rd- and 6th-month test sets of the NYC dataset in Fig. 9; the results on CA dataset are qualitatively similar and are ignored due to the space limitation. From Fig. 9, we observe that as time goes on, the performance of all methods decrease. The results are as expected since the users' interests may change over time and they may know more friends over time and join their group events. As a result, it is more difficult to predict participants for events far from now.

Second, we investigate how our two modifications to the original EBSN graph in Section III-B affect the performance of HeteRS. In particular, we test HeteRS on two modified graphs: one graph does not have session nodes but still uses the time-weighted function Eq. (1) for edges connecting to events, and the other uses session nodes but does not weight events' edges. The comparison results are shown in Fig. 10. We observe the performances of HeteRS on two new graphs are worse than on full EBSN graph, which demonstrates HeteRS is able to take advantages of two temporal aspects of user behaviors. The improvement of HeteRS is significant than other two variants with p-value less than 0.01.

H. *Interpretation of Transition Parameters*

In all these experiments, HeteRS works better than uni_HeteRS. This demonstrates that our parameter learning process is necessary and useful. In this subsection, we investigate the transition parameters α_{MN} to examine the roles of different types of entities in our recommendation problems. Due to the space limitation, we only show the values of learned parameters for event-to-user recommendation problem on both NYC and CA datasets and for group-to-user recommendation problem on NYC dataset in Table II.

TABLE II: TRANSITION PARAMETERS

α_{MN}		Recommendation problems		
M	N	Event-to-user rec. NYC	Event-to-user rec. CA	Group-to-user rec. NYC
G	E	0.34	0.44	0.38
S	E	0.44	0.37	0.25
U	E	0.11	0.10	0.12
V	E	0.11	0.09	0.25
E	G	0.01	0.01	0.27
T	G	0.01	0.01	0.31
U	G	0.01	0.01	0.42
q_g	G	0.97	0.97	N/A
E	S	0.88	0.73	N/A
q_s	S	0.12	0.27	N/A
G	T	0.72	0.83	0.65
U	T	0.28	0.17	0.35
E	U	0.33	0.37	0.11
G	U	0.50	0.43	0.11
T	U	0.16	0.19	0.05
q_u	U	0.01	0.01	0.73
E	V	0.99	0.80	N/A
q_v	V	0.01	0.20	N/A

From Table II, for event-to-user recommendation problem on NYC, we observe that the value of α_{SE} is the largest, 0.44, among those of transition parameters from other types to event type E, which means that users in NYC tend to join events periodically. On the other hand, parameter α_{SE} has smaller value in CA dataset, which indicates that the user activities in CA is less periodical than in NYC. This could be the reason why the improvement we achieved in event-to-user recommendation on CA dataset is less significant than on NYC dataset. On both datasets, the location information plays a minor role as the weight α_{VE} is small, 0.11 for NYC and 0.09 for CA dataset. By analyzing data, we observe that two events held in the same venue always belong to the same group. This means that the role of locations is already contained by that of groups. Moreover, most of the new events take place in new venues, and this makes the location information less useful. For group-to-user recommendation, we can see that the weights for nodes of type G largely come from those of U. Hence, we mostly rely on similar users when recommending groups to a target user. Note that the value of α_{TG} is high, *i.e.* tags have a large role in group-to-user recommendation. Also, we can see that the weight of T is mainly contributed by both G (α_{GT} is 0.65) and U (α_{UT} is 0.35). These observations are because users usually choose groups with the same topics (represented by tags) as their current groups or their interests.

I. Approximation Method

In Section V, we propose an approximation method for HeteRS. We evaluate the efficiency of HeteRS with approximation method (Approximation HeteRS) by comparing with the one using iterative method (Iterative HeteRS) in this section. Note that in previous experiments, we use iterative method for HeteRS.

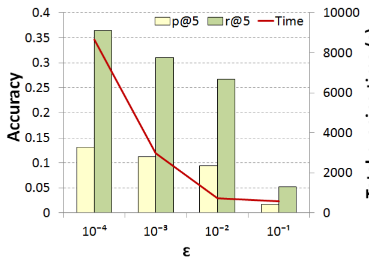
1) Parameter Setting: The approximation method has only one parameter: the preference threshold ϵ, which is the minimum preference a node needs to propagate to its adjacent nodes. If a node has preference smaller than ϵ, it will be ignored and no preferences are sent to its neighbors. There is a trade-off between the accuracy and the running time when setting ϵ: if we set ϵ large, the preference propagation process stops early, and only a small fraction of nodes get values; hence, the running time is short but the accuracy may be low. Figure 11 shows the performance of Approximation HeteRS in terms of p@5, r@5 and total running time in three recommendation problems on CA dataset with different preference threshold ϵ. We can see that when ϵ becomes larger, the running time is shorter, while the accuracy decreases. The precision and recall of Approximation HeteRS become stable when ϵ is smaller than 10^{-3} and 10^{-2} in tag-to-group and event-to-user recommendation, respectively. The accuracy of Approximation HeteRS in group-to-user recommendation still increases when ϵ is below 10^{-3}; however, it suffers from significant increase in running time. In the rest of the experiments, we set ϵ to 10^{-3} because Approximation HeteRS balances well between accuracy and running time.

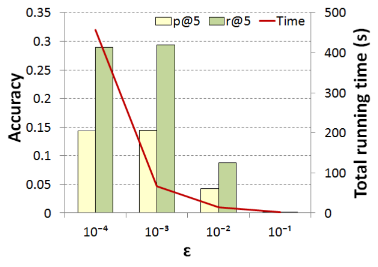
2) Experiment for Scalability: To evaluate the scalability of the proposed algorithms with the size of EBSN graph, we use training sets of different sizes, *i.e.*, 6-month, 12-month and 18-month training sets that include events in last 6, 12 and 18 months respectively. Figure 12 shows the results of both Iterative and Approximation HeteRS in terms of p@5 and total running time on CA of different sizes. The total number of nodes/edges of EBSN graphs corresponding to 6-month, 12-month and 18-month training sets is 48K/1M, 102K/2.4M and 136K/3.3M, respectively. From Fig. 12, it is observed that Approximation HeteRS is always at least 3 times faster than Iterative HeteRS in all recommendation problems, and the improvement is almost an order of magnitude on the largest EBSN graph. When the size of training data, *i.e.*, the EBSN graph, becomes larger, the computation time of both algorithms scales well, but Approximation HeteRS scales much better. We also observe that Approximation HeteRS achieves similar accuracy as Iterative HeteRS, and it even produces better results in tag-to-group recommendation problem. This improvement probably comes from the locality property of our approximation method: it rarely reaches nodes that are far from query node, hence reducing the number of irrelevant nodes in final results. Overall, the proposed approximation method greatly improves the efficiency of HeteRS while achieving similar accuracy as Iterative HeteRS.

Compared to Random Walk-based methods, *e.g.*, full_RWR, uniHeteRS, the Approximation HeteRS achieves similar efficiency improvement as the running time of those methods are roughly equal to that of Iterative HeteRS. Other baselines, *e.g.*, CF, BPR and RWR, always have shortest response time; their accuracy, however, is considerably worse, as shown in Figs. 5-7. Due to the space limitation, we do not show the comparison results of Approximation HeteRS with other methods.

VII. Conclusion

In this paper, we introduce a general model to solve multiple recommendation problems in heterogeneous networks like event-based social networks. First, we transform the recommendation problems into node proximity calculation problem w.r.t. query inputs on a heterogeneous graph, and employ multivariate Markov chain to solve it. Then, we propose an optimization scheme to learn transition parameters for our model. The learned parameters not only enable our model to


(a) Group-to-user recommendation


(b) Tag-to-group recommendation

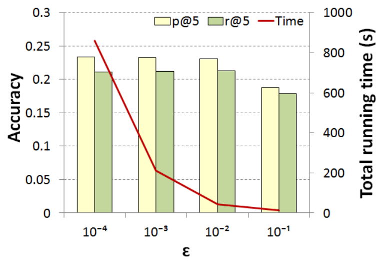

(c) Event-to-user recommendation

Fig. 11: Performance of Approximation HeteRS with different parameter ϵ on CA dataset

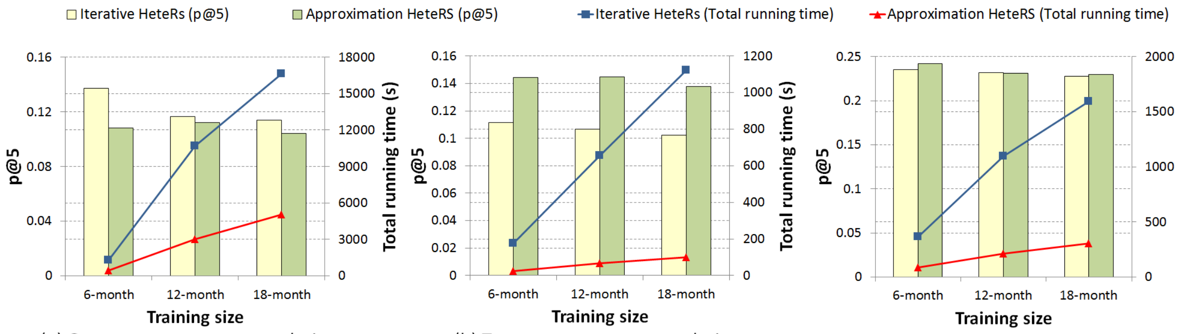

(a) Group-to-user recommendation (b) Tag-to-group recommendation (c) Event-to-user recommendation

Fig. 12: Comparison between Iterative and Approximation HeteRS in terms of effectiveness and efficiency on CA dataset

achieve superior performance, but also help us to understand roles of different types of entities in each recommendation problem. The experimental results on two real-world datasets demonstrate the proposed model outperforms the state-of-the-art methods and other baselines in all recommendation tasks. In the future, we will evaluate the performance of the proposed model for the recommendation problem in other heterogeneous networks.

Acknowledgment

This work is supported in part by a grant awarded by a Singapore MOE AcRF Tier 2 Grant (ARC30/12).

References

[1] W. Zhang, J. Wang, and W. Feng, "Combining latent factor model with location features for event-based group recommendation," in *KDD*. ACM, 2013, pp. 910–918.

[2] X. Liu, Y. Tian, M. Ye, and W.-C. Lee, "Exploring personal impact for group recommendation," in *CIKM*. ACM, 2012, pp. 674–683.

[3] X. Liu, Q. He, Y. Tian, W.-C. Lee, J. McPherson, and J. Han, "Event-based social networks: linking the online and offline social worlds," in *KDD*. ACM, 2012, pp. 1032–1040.

[4] I. Konstas, V. Stathopoulos, and J. M. Jose, "On social networks and collaborative recommendation," in *SIGIR*. ACM, 2009, pp. 195–202.

[5] S. Lee, S.-i. Song, M. Kahng, D. Lee, and S.-g. Lee, "Random walk based entity ranking on graph for multidimensional recommendation," in *RecSys*. ACM, 2011, pp. 93–100.

[6] J. Tang, S. Wu, J. Sun, and H. Su, "Cross-domain collaboration recommendation," in *KDD*. ACM, 2012, pp. 1285–1293.

[7] H. Wang, M. Terrovitis, and N. Mamoulis, "Location recommendation in location-based social networks using user check-in data," in *SIGSPATIAL*. ACM, 2013, pp. 364–373.

[8] H. Cheng, P.-N. Tan, J. Sticklen, and W. F. Punch, "Recommendation via query centered random walk on k-partite graph," in *ICDM 2007*. IEEE, 2007, pp. 457–462.

[9] W.-K. Ching and M. K. Ng, "Multivariate markov chains," *Markov Chains: Models, Algorithms and Applications*, pp. 141–169, 2006.

[10] X. Jiang, X. Sun, and H. Zhuge, "Towards an effective and unbiased ranking of scientific literature through mutual reinforcement," in *CIKM*. ACM, 2012, pp. 714–723.

[11] H. Sayyadi and L. Getoor, "Futurerank: Ranking scientific articles by predicting their future pagerank," in *SDM*. SIAM, 2009, pp. 533–544.

[12] M. Jiang, P. Cui, F. Wang, Q. Yang, W. Zhu, and S. Yang, "Social recommendation across multiple relational domains," in *CIKM*. ACM, 2012, pp. 1422–1431.

[13] W. Feng and J. Wang, "Incorporating heterogeneous information for personalized tag recommendation in social tagging systems," in *KDD*. ACM, 2012, pp. 1276–1284.

[14] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *WSDM*. ACM, 2011, pp. 635–644.

[15] B. Gao, T.-Y. Liu, W. Wei, T. Wang, and H. Li, "Semi-supervised ranking on very large graphs with rich metadata," in *KDD*. ACM, 2011, pp. 96–104.

[16] 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*. ACM, 2010, pp. 723–732.

[17] Q. Yuan, G. Cong, and A. Sun, "Graph-based point-of-interest recommendation with geographical and temporal influences," in *CIKM*. ACM, 2014, pp. 659–668.

[18] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *UAI*. AUAI Press, 2009, pp. 452–461.

[19] N. Lao and W. W. Cohen, "Fast query execution for retrieval models based on path-constrained random walks," in *KDD*. ACM, 2010, pp. 881–888.