Mobile App Cross-Domain Recommendation with Multi-Graph Neural Network

YI OUYANG and BIN GUO, Northwestern Polytechnical University
XING TANG and XIUQIANG HE, Noah's Ark Lab, Huawei
JIAN XIONG, Tencent
ZHIWEN YU, Northwestern Polytechnical University

With the rapid development of mobile app ecosystem, mobile apps have grown greatly popular. The explosive growth of apps makes it difficult for users to find apps that meet their interests. Therefore, it is necessary to recommend user with a personalized set of apps. However, one of the challenges is data sparsity, as users' historical behavior data are usually insufficient. In fact, user's behaviors from different domains in app store regarding the same apps are usually relevant. Therefore, we can alleviate the sparsity using complementary information from correlated domains. It is intuitive to model users' behaviors using graph, and graph neural networks have shown the great power for representation learning. In this article, we propose a novel model, Deep Multi-Graph Embedding (DMGE), to learn cross-domain app embedding. Specifically, we first construct a multi-graph based on users' behaviors from different domains, and then propose a multi-graph neural network to learn cross-domain app embedding. Particularly, we present an adaptive method to balance the weight of each domain and efficiently train the model. Finally, we achieve cross-domain app recommendation based on the learned app embedding. Extensive experiments on real-world datasets show that DMGE outperforms other state-of-art embedding methods.

CCS Concepts: • **Information systems** → **Recommender systems**; • **Computing methodologies** → *Neural networks*; Multi-task learning;

Additional Key Words and Phrases: Mobile app, cross-domain recommendation, graph neural network, multi-task learning, transfer learning

ACM Reference format:
Yi Ouyang, Bin Guo, Xing Tang, Xiuqiang He, Jian Xiong, and Zhiwen Yu. 2021. Mobile App Cross-Domain Recommendation with Multi-Graph Neural Network. *ACM Trans. Knowl. Discov. Data* 15, 4, Article 55 (April 2021), 21 pages.
https://doi.org/10.1145/3442201

Xing Tang is the co-first author. This work was done when he worked at Tencent.
Xiuqiang He, this work was done when he worked at Tencent.
This work was partially supported by the National Key R&D Program of China (2017YFB1001800), the National Science Fund for Distinguished Young Scholars (62025205), and the National Natural Science Foundation of China (No. 61960206008, 61772428, and 61725205).
Authors' addresses: Y. Ouyang, B. Guo (corresponding author), and Z. Yu, Northwestern Polytechnical University, No. 127, Youyi-West Rd., Xi'an, Shaanxi, 710072, China; emails: ouyangyi@mail.nwpu.edu.cn, {guob, zhiwenyu}@nwpu.edu.cn; X. Tang and X. He, Noah's Ark Lab, Huawei, Wuhe Avenue, Shenzhen, Guangdong, 518129, China; emails: tangxing01@hotmail.com, hexiuqiang1@huawei.com; J. Xiong, Tencent, No. 10000, Shennan Avenue, Shenzhen, Guangdong, 518057, China; email: janexiong@tencent.com.

1556-4681/2021/04-ART55 $15.00
https://doi.org/10.1145/3442201

1 INTRODUCTION

Recently, with the rapid development of mobile internet technology, as well as the widespread adoption of smartphones, mobile apps have grown greatly popular, and become the indispensable companions in people's daily lives. The emergence of mobile apps has changed people's lifestyles, including work, socialize, entertainment, and consumption, and made our life more intelligent and convenient.

Mobile app stores [11], such as Google Play and Apple App Store, provide a large number of mobile apps to meet the need of different users. However, the explosive growth of mobile apps with diverse categories and functionalities makes it difficult for users to find apps that are relevant to their interests from the huge amount of apps. Therefore, it is significant and necessary to provide accurate mobile app recommendation for users, and it is helpful to understand user preferences and improve user experience. Besides, accurate recommendation of mobile app will improve the popularity of mobile apps, promote the app economy, and bring the economic benefits to app developers [28, 29].

There are some recent studies about the mobile app recommendation, which can be mainly divided into three categories: *Context-aware app recommendation* [16, 47], which aims to use the current mobile context information (e.g., time and location) of users to facilitate the mobile app recommendation. *Privacy protection-based app recommendation* [22, 46], which aims to protect user privacy in mobile app recommendation, as mining and understanding user preferences may also leak user privacy information. *Cold-start app recommendation* [21], which aims to recommend newly released apps to users, as newly released apps may not have user ratings and lead to the cold-start problem. These works don't focus on leveraging users' behaviors in mobile app store (i.e., users' app download records) to recommend personalized mobile apps to users. Besides, there are usually a relative small portion of active users in mobile app store, and a majority of non-active users often download only a small number of apps, users' behavior data are thus lacking or insufficient, which makes it difficult to accurately recommend personalized apps to users [37].

In fact, though data from a single domain are sparse, user behaviors from correlated domains with the same apps are usually complementary [48]. In the mobile app store, there are two ways users interact (e.g., download) with apps. One is downloading apps recommended on the homepage (or category page, etc.) of app store (i.e., *recommendation domain*), and the other is by searching (i.e., *search domain*). Specifically, search domain refers to the search page of app store. Users can search and download apps directly in this domain. Recommendation domain refers to the homepage (or category page, etc.) of app store. When users search, click, or download apps, the app store can record users' long-term behaviors, and then recommends related apps on the homepage based on users' historical behaviors. User behaviors in search domain reflect user's current needs or intention, while that in recommendation domain reflect user's relative general interests. Leveraging the interaction data from search domain can improve the performance of recommendation. On the other hand, interaction data from recommendation domain can also help to explore user's personalized interests and thus optimizing the ranking in search domain. Thus, the information from recommendation and search domain are complementary, and we are motivated to leverage the complementary information from correlated domains in mobile app store to alleviate the sparsity.

Cross-domain recommender systems (RSs) [3] aim to fuse information from correlated domains to improve the performance of recommendation. Based on the information shared by different domains, existing works on cross-domain recommendation mainly fall into two categories, i.e., *user-shared* and *item-shared*. In *user-shared cross-domain recommendation* [14], user embeddings or profiles are usually shared across domains. However, user information in different domains may not be directly accessible due to privacy protection or the absence of user profiles, thus it is difficult

to directly obtain the shared user information. In *item-shared cross-domain recommendation* [24, 45], matrix factorization (MF) techniques [19, 31] are often used to factorize user-item interaction matrix into user and item embeddings, then item embeddings are shared or transferred across domains. However, these works only consider the user-item interaction, while ignore the item dependency (i.e., the item co-occurrences in users' behaviors). The item[1] dependency is also useful to capture item-item similarity and reflect users' preferences [36]. Thus, we are motivated to learn cross-domain app embedding based on the app co-occurrences in users' behaviors, and then use the cross-domain app embedding to achieve cross-domain app recommendation.

The key is to generate cross-domain app embedding. However, it is challenging to learn cross-domain app embedding due to the following three key issues.

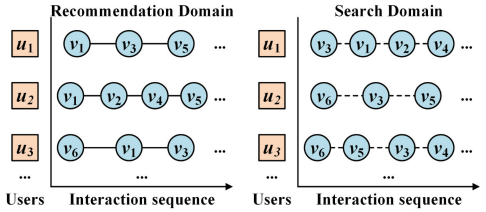
- —*How to correlate different domains in mobile app store, and what information about mobile apps should be transferred (or shared) across domains?*
- —Though existing work [36] have applied the state-of-art embedding technique [30] to learn user/item embedding for recommendation, these methods are developed for a single domain, which fail to generate effective cross-domain item embedding. *How to transfer information across domains and generate cross-domain app embedding?*
- —In mobile app store, all domains contribute to the effectiveness of app embedding. However, the weight (or contribution) of each domain is different, the effectiveness of app embedding and the performance of app recommendation is sensitive to the weight selection. *How to adaptively balance the weight of each domain to generate effective cross-domain app embedding?*

To address these challenges, in this article, we propose a novel embedding model, named, Deep Multi-Graph Embedding (DMGE). First, to correlate different domains in mobile app store, we construct a cross-domain app graph (i.e., multi-graph) based on users' behaviors from different domains. In the multi-graph, nodes represent mobile apps, and edges represent app co-occurrences in users' historical app download records. Thus, learning cross-domain app embedding is formulated as learning node embedding in the multi-graph. Second, to utilize the power of Graph Neural Networks (GNNs) [44] on graph embedding, we propose a multi-graph neural network inspired by multi-task learning, which extends GNNs to learn cross-domain app embedding. Specifically, we design the domain-shared embedding layers to generate shared embedding of apps, which is used to transfer across domains, and design the domain-specific embedding layers to generate specific embedding of apps for each domain based on the shared embedding. Third, to balance the weight of different domains, we present an adaptive method to dynamically adjust the weight, and train the model efficiently. Finally, we use the learned cross-domain app embedding to achieve cross-domain app recommendation.

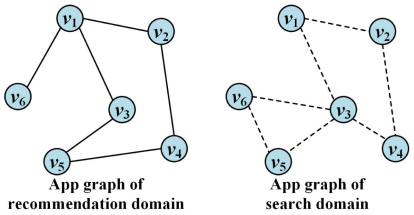
The contributions of our work are summarized as follows:

- —We aim to leverage the complementary information from correlated domains in app store to facilitate mobile app cross-domain recommendation. Innovatively, we correlate different domains by modeling the app dependency using a multi-graph, and learn cross-domain app embedding for cross-domain recommendation.
- —We propose DMGE, which is a graph neural network based on multi-task learning, to learn domain-shared and domain-specific embedding. Particularly, it can adaptively balance the weight of different domains.

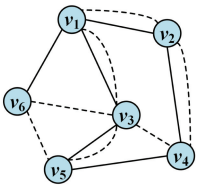
[1]In this article, the item refers to the mobile app.



(a) Users' interaction sequences in different domains.



(b) App graph in different domains.



(c) Cross-domain app graph.

Fig. 1. Illustration of constructing the app graph: (a) Users' interaction sequences in different domains, in which the square denotes user, the circle denotes mobile app, the solid line denotes app co-occurrence in recommendation domain, and the dashed line denotes app co-occurrence in search domain. (b) App graph in different domains, in which the node represents app, the edge represents co-occurrence of two apps, and the weight of edge is the number of co-occurrence of two apps in the interaction sequences. (c) Cross-domain app graph, i.e., the multi-graph, which contains the app dependency in different domains.

—We evaluate DMGE on large-scale real-world datasets, and the results show that DMGE outperforms other state-of-the-art embedding methods.

2 PRELIMINARIES

In this section, we present several fundamental definitions of our work.

2.1 Users' Behaviors in Different Domains

We first introduce two important domains in app store, and users' behaviors in these domains.

Definition 1 (Recommendation Domain). It refers to the homepage (or category page, etc.) of app store. It often recommends relevant apps according to user's historical app download records.

Definition 2 (Search Domain). It refers to the search page of the app store. Users can search and download apps in this domain according to their current needs or intention. Both domains share the same set of mobile apps.

Definition 3 (Interaction Sequence). Suppose there are $D\,(D \geq 2)$ domains. In each domain, for each user, his/her behaviors (i.e., $< user, app, time >$) are sequential, and can be sorted by the timestamps when he/she downloaded the app, as shown in Figure 1(a).

2.2 App Graph Construction

Generally, users' sequential behaviors in each domain can be modeled by graph intuitively [36]. Specifically, in the app graph, apps can be modeled as nodes, and app co-occurrences can be

Table 1. A Summary of the Description of Notations

Notation	Description
v_i	A mobile app.
u	A user.
D	The number of domains.
$\mathcal{G}$	The multi-graph.
$\mathcal{V}, \mathcal{E}$	The node/edge set of $\mathcal{G}$.
$\mathcal{G}_d$	The sub-graph, i.e., domain d.
$\mathbf{X}$	The app feature matrix.
$\mathbf{A}$	The adjacency matrix of multi-graph $\mathcal{G}$.
$\mathbf{A}_d$	The adjacency matrix of subgraph $\mathcal{G}_d$.
$\mathbf{X}_s$	The shared embedding of apps.
$\mathbf{X}_d$	The specific embedding of apps in domain d.
Θ_s	The parameter of shared embedding layers.
Θ_d	The parameter of specific embedding layers in domain d.

modeled as edges. The app graph contains abundant information about users' behaviors. We then introduce the construction of app graph.

Definition 4 (App Graph). In domain d ($d = \{1, \ldots, D\}$), the app graph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E}_d)$ is constructed to represent the dependency (i.e., co-occurrences) between apps, as shown in Figure 1(b). The node set $\mathcal{V}$ denotes the set of apps. The edge set $\mathcal{E}_d$ denotes the app co-occurrences. The weight of edge $e_{i,j}$ is the number of co-occurrence of app i and j in the interaction sequences.

Definition 5 (App Multi-Graph). As these D domains are correlated by the share apps $\mathcal{V}$, the cross-domain app graph can be constructed as the app multi-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which contains the node set $\mathcal{V}$ with N nodes, and the edge set $\mathcal{E}$ with D types of edges (i.e., $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_D\}$. Each type constructs a subgraph, and each type of edge represents the app co-occurrence in a domain, as shown in Figure 1(c).

2.3 Problem Definition

To achieve mobile app recommendation, we first learn app embedding in the cross-domain app graph (i.e., app multi-graph). Then we can generate the user embedding in each domain by aggregating the embeddings of his/her historical downloaded apps. Finally, we measure user-app similarity by computing the distance between user embedding and item embedding. Based on the user-app similarity, we recommend candidate top-K apps for each user in each domain.

Specifically, learning app embedding in the cross-domain app graph can be formulated as learning node embedding in the multi-graph, which can be stated as follows: with an undirected weighted multi-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$, representing the input for each node as an M-dimensional vector, our goal is to learn a set of embeddings for all nodes in each subgraph $\mathcal{G}_d$, i.e., $\mathcal{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_D\}$ ($\mathbf{X}_d \in \mathbb{R}^{N \times E}$ is the node embedding in subgraph $\mathcal{G}_d$, with each node has an E-dimensional embedding). A summary of the definition of notations is given in Table 1.

3 DEEP MULTI-GRAPH EMBEDDING

In this section, we elaborate the DMGE model to learn cross-domain app embedding. We first propose a multi-graph neural network to learn node embedding, and introduce the definition and

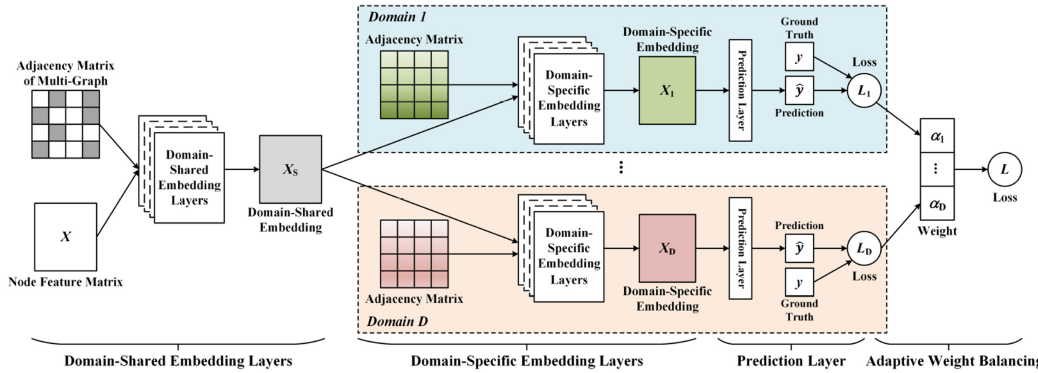


Fig. 2. The architecture of the multi-graph neural network.

architecture of the multi-graph neural network (in Section 3.1). Then, we introduce the key components of the multi-graph neural network (from Section 3.2 to Section 3.4). Finally, we present an adaptive method to learn the weight of each domain (in Section 3.5).

3.1 Multi-Graph Neural Network

In the cross-domain app graph $\mathcal{G}$, the app set $\mathcal{V}$ is shared in all subgraphs.[2] In Figure 1(c), each app v_i has different neighbors in different subgraph, i.e., each app may co-occur with different apps in different domains, thus it is likely that app v_i has different embeddings in different subgraph [23]. Moreover, all these embeddings belong to the same app v_i, thus they are inherently related, and contain the shared information about app v_i. Here, we present two types of app embedding in the cross-domain app graph $\mathcal{G}$ as follows:

Definition 4 (Domain-Shared Embedding). Each app v_i has a shared embedding $\mathbf{x}_s$, which denotes the shared information in the multi-graph $\mathcal{G}$. We denote the shared embedding of the multi-graph $\mathcal{G}$ as $\mathbf{X}_s$.

Definition 5 (Domain-Specific Embedding). Each app v_i has a specific embedding $\mathbf{x}_d$ in subgraph $\mathcal{G}_d$, which encodes the specific information in subgraph $\mathcal{G}_d$. We denote the specific embedding of subgraph $\mathcal{G}_d$ as $\mathbf{X}_d$.

GNNs [39, 44] have recently emerged as a powerful approach for representation learning on graphs, such as graph convolutional network (GCN) [18]. Existing GNNs are mostly developed for single graph, where only one type of edge exists between a pair of nodes. However, the multi-graph may contain multiple types of edges between a pair of nodes, which brings additional complexity, thus existing GNNs fail to learn effective multi-graph embedding.

To learn multiple types of node embeddings, we propose a multi-graph neural network, which extends GNNs to generate multi-graph node embedding. The architecture is presented in Figure 2, which follows the multi-task learning regime [4, 32], and each subgraph (i.e., domain) is viewed as a task. There are four key components in the architecture: (1) the *domain-shared embedding layers* to learn shared embedding of nodes in the multi-graph; (2) the *domain-specific embedding layers* to learn specific embedding of nodes in each subgraph; (3) the *prediction layer* to predict the probability that a link existing between a pair of nodes based on the specific embedding; and (4) the *adaptive weight balancing module* to automatically adjust the weight of different domains.

[2] A subgraph represents a domain.

The multi-graph neural network has the following characteristics:

— The inputs of the model the node features $\mathbf{X}$ (i.e., app features) and adjacency matrix $\mathbf{A}$ of the multi-graph $\mathcal{G}$.
— The outputs of the model are the app embedding $\mathbf{X}_d$ in all domains. After obtaining the app embedding, we can achieve mobile app cross-recommendation based on the app embedding (as illustrated in Section 4).
— The architecture of the multi-graph neural network is built upon the most commonly used multi-task structure with hard parameter sharing [4, 32], where the domain-shared embedding layers are shared across all the domains, and then each domain has the domain-specific embedding layers on top of the domain-shared embedding. The information can be transferred (or shared) across domains from the domain-shared embedding layers.
— The domain-shared embedding layers are designed to generate shared information by encoding the node attributes and multi-graph structure, which can be transferred (or shared) across domains. The output of these layers is the shared node embedding, which contains the shared information among all domains, and can be transferred to learn specific embedding in different domains.
— The domain-specific embedding layers are designed to generate specific embedding of nodes in each domain, based on the domain-shared embedding. Then, the specific embedding in each domain can be used to solve the task in corresponding domain.
— All the domains are correlated by the domain-shared embedding layers.

3.2 Domain-Shared Embedding Layer

The input of the domain-shared embedding layers are the adjacency matrix of multi-graph and node feature matrix, and these layers are designed to generate shared embedding by encoding node attributes and multi-graph structure. The graph convolution operator in GCNs [18] can efficiently learn node embedding based on neighborhood aggregation scheme, thus we adopt the graph convolution operator in the domain-shared embedding layers:

$$\mathbf{X}_s^{(l+1)} = ReLu\left(\tilde{\mathbf{W}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{W}}^{-\frac{1}{2}}\mathbf{X}_s^{(l)}\Theta_s^{(l)}\right), \tag{1}$$

where $\mathbf{X}_s^{(l+1)} \in \mathbb{R}^{N\times E_s}$ is the shared embedding of multi-graph $\mathcal{G}$ in l-th layer, and $\mathbf{X}_s^{(0)} = \mathbf{X} \in \mathbb{R}^{N\times M}$ is the node feature matrix. $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N \in \mathbb{R}^{N\times N}$ is the adjacency matrix of graph $\mathcal{G}$ with added self-connections. $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix, and $\mathbf{A}(i,j) = 1$ if there are any links between node v_i and v_j in each domain. The adjacency matrix A can be seen as the union of link information in all domains, i.e., the adjacency matrix A encodes different types of edges between mobile apps across all domains. $\mathbf{I}_N \in \mathbb{R}^{N\times N}$ is the identity matrix. $\tilde{\mathbf{W}} \in \mathbb{R}^{N\times N}$ is a diagonal matrix, and $\tilde{\mathbf{W}}(i,i) = \sum_j \tilde{\mathbf{A}}(i,j)$. $\Theta_s^{(l)} \in \mathbb{R}^{N\times E_s}$ is the shared weight matrix of l-th layer. The output of the l-th shared embedding layer is the shared node embedding $\mathbf{X}_s$, which contains the shared information among all subgraphs, and can be transferred to learn specific embedding in different subgraphs.

3.3 Domain-Specific Embedding Layer

Based on the shared embedding $\mathbf{X}_s$, the domain-specific embedding layers are designed to generate specific embedding of nodes on each subgraph by Equation (2), and the inputs of these layers are

the shared embedding and the adjacency matrix of each subgraph.

$$\begin{cases} \mathbf{X}_1^{(l+2)} = ReLu\left(\tilde{\mathbf{W}}_1^{-\frac{1}{2}}\tilde{\mathbf{A}}_1\tilde{\mathbf{W}}_1^{-\frac{1}{2}}\mathbf{X}_1^{(l+1)}\Theta_1^{(l+1)}\right) \\ \quad \cdots \\ \mathbf{X}_D^{(l+2)} = ReLu\left(\tilde{\mathbf{W}}_D^{-\frac{1}{2}}\tilde{\mathbf{A}}_D\tilde{\mathbf{W}}_D^{-\frac{1}{2}}\mathbf{X}_D^{(l+1)}\Theta_D^{(l+1)}\right), \end{cases} \tag{2}$$

where $\mathbf{X}_d^{(l+2)} \in \mathbb{R}^{N\times E}$ is the specific embedding of subgraph $\mathcal{G}_d$ in l+1-th layer, and $\mathbf{X}_d^{(l+1)} = \mathbf{X}_s$ is the shared embedding. $\tilde{\mathbf{A}}_d = \mathbf{A}_d + \mathbf{I}_N \in \mathbb{R}^{N\times N}$. $\mathbf{A}_d \in \mathbb{R}^{N\times N}$ and $\mathbf{A}_d(i,j)$ is the weight of edge (v_i, v_j), and the weight of the edge is the number of co-occurrence of app i and j. $\tilde{\mathbf{W}}_d \in \mathbb{R}^{N\times N}$ and $\tilde{\mathbf{W}}_d(i,i) = \sum_j \tilde{\mathbf{A}}_d(i,j)$. $\Theta_d^{(l+1)} \in \mathbb{R}^{E_s\times E}$ is the specific weight matrix of l + 1-th layer. The outputs of the specific embedding layers is the set of node embedding $\mathcal{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_D\}$.

3.4 Prediction Layer

In this layer, we use the domain-specific embeddings to predict the linkage between two nodes in a subgraph (i.e. the probability that a link existing between two nodes), and train the model by modeling the graph structure. The probability that there exists an edge between node v_i and node v_j in subgraph $\mathcal{G}_d$ is defined in Equation (3):

$$\hat{y}_{ij} = \sigma\left(\mathbf{x}_{d,i}^T \cdot \mathbf{x}_{d,j}\right), \tag{3}$$

where $\mathbf{x}_{d,i}$ is the i-th row of $\mathbf{X}_d$, which is the embedding of node v_i in subgraph $\mathcal{G}_d$. $\sigma(\cdot)$ is the sigmoid function.

For each subgraph $\mathcal{G}_d$, the loss function L_d is defined as the cross-entropy of the prediction $\hat{y}$ and the ground truth y, which can be formulated as follows:

$$\begin{aligned} L_d &= -y_{ij}\log(\hat{y}_{ij}) - (1-y_{ij})\log(1-\hat{y}_{ij}) \\ &= -\sum_{v_j\in\mathcal{S}_{d,p}} \log\sigma\left(\mathbf{x}_{d,i}^T \cdot \mathbf{x}_{d,j}\right) - \sum_{v_k\in\mathcal{S}_{d,n}} \log\sigma\left(-\mathbf{x}_{d,i}^T \cdot \mathbf{x}_{d,k}\right), \end{aligned} \tag{4}$$

where $\mathcal{S}_{d,p}$ is the set of positive samples in subgraph $\mathcal{G}_d$, which contains the tuples (v_i, v_j, d) with an edge between node v_i and v_j in subgraph $\mathcal{G}_d$. $\mathcal{S}_{d,n} = \{v_k | k = 1, \ldots, S\}$ is the set of negative samples in subgraph $\mathcal{G}_d$, and $\mathcal{S}_{d,n}$ is sampled from node set $\mathcal{V}$ by negative sampling [25, 26], which contains the tuples (v_i, v_k, d) with no edge between node v_i and v_k in subgraph $\mathcal{G}_d$. Specifically, the negative samples in $\mathcal{S}_{d,n}$ are generated based on the degree of the node, i.e., sampling the nodes with large degree values. Because these nodes usually contain sufficient structural and semantic information in the graph.

Therefore, the objective function of the model is defined as Equation (5):

$$\begin{cases} L_1 = -\sum_{v_j\in\mathcal{S}_{1,p}} \log\sigma\left(\mathbf{x}_{1,i}^T \cdot \mathbf{x}_{1,j}\right) - \sum_{v_k\in\mathcal{S}_{1,n}} \log\sigma\left(-\mathbf{x}_{1,i}^T \cdot \mathbf{x}_{1,k}\right) \\ \quad \cdots \\ L_D = -\sum_{v_j\in\mathcal{S}_{D,p}} \log\sigma\left(\mathbf{x}_{D,i}^T \cdot \mathbf{x}_{D,j}\right) - \sum_{v_k\in\mathcal{S}_{D,n}} \log\sigma\left(-\mathbf{x}_{D,i}^T \cdot \mathbf{x}_{D,k}\right). \end{cases} \tag{5}$$

3.5 Adaptive Weight Balancing

In fact, the multi-graph neural network is difficult to train, because multiple subgraphs (i.e., domains) need to be solved jointly, different domains need to be properly balanced to train the shared and specific embedding that are useful to all domains. Generally, a naive approach to train the

model is to perform a weighted sum of the loss function in Equation (5): $L = \sum_{d=1}^{D} \alpha_d L_d$, which is the dominant approach in multi-task learning [4]. However, there are some issues of this method. First, the weight α_d controls the information transfer across domains, the effectiveness of embedding and the model performance is dependent on the weight selection. Second, it is time-consuming and computationally expensive to tune the weight α_d manually, makes the model learning prohibitive in practice. Therefore, it is desirable to find an adaptive method to balance the weight automatically and train the model efficiently.

We derive the weight of each domain from the perspective of multi-objective optimization. To benefit all domains, we need to optimize all the objectives as follows:

$$\begin{cases} \min\limits_{\Theta_s, \Theta_1} L_1(\Theta_s, \Theta_1) \\ \cdots \\ \min\limits_{\Theta_s, \Theta_D} L_D(\Theta_s, \Theta_D), \end{cases} \tag{6}$$

where Θ_s is the domain-shared parameter, Θ_d ($d = \{1, \ldots, D\}$) is the domain-specific parameter.

To solve Equation (6), we first state the Karush–Kuhn–Tucker (KKT) conditions [20], which is a necessary condition for the optimal solution of multi-objective optimization.

$$\begin{cases} \sum\limits_{d=1}^{D} \alpha_d \frac{\partial L_d(\Theta_s, \Theta_d)}{\partial \Theta_s} = 0 \\ \frac{\partial L_d(\Theta_s, \Theta_d)}{\partial \Theta_d} = 0 \\ \sum\limits_{d=1}^{D} \alpha_d = 1 \\ \alpha_d \geq 0 \end{cases} \tag{7}$$

where α_d is the weight of objective $L_d(\Theta_s, \Theta_d)$.

As proved in [8], the solution to Equation (8) falls into two cases, one is 0 and the result satisfies the KKT conditions Equation (7); the other is that the solution gives a descent direction that improves all objectives in Equation (6). Thus, solving Equation (7) is equivalent to optimizing Equation (8).

$$\begin{aligned} &\min_{\alpha_1, \ldots, \alpha_D} \left\| \sum_{d=1}^{D} \alpha_d \frac{\partial L_d(\Theta_s, \Theta_d)}{\partial \Theta_s} \right\|_2^2 \\ &s.t. \begin{cases} \sum\limits_{d=1}^{D} \alpha_d = 1 \\ \alpha_d \geq 0 \end{cases} \end{aligned} \tag{8}$$

To clearly illustrate how to derive α_d, we consider the case of two domains, thus Equation (8) can be simplified as:

$$\begin{aligned} &\min_{\alpha} \left\| \alpha \frac{\partial L_1(\Theta_s, \Theta_1)}{\partial \Theta_s} + (1 - \alpha) \frac{\partial L_2(\Theta_s, \Theta_2)}{\partial \Theta_s} \right\|_2^2 \\ &s.t.\ 0 \leq \alpha \leq 1 \end{aligned} \tag{9}$$

where α is the weight of loss function $L_1(\Theta_s, \Theta_1)$.

Equation (9) is a unary quadratic equation of α, and the solution to Equation (9) is:

$$\alpha = \begin{cases} 0, & \mathtt{sum}(\mathbf{U}^T\mathbf{V}) \geq \mathtt{sum}(\mathbf{V}^T\mathbf{V}) \\ 1, & \mathtt{sum}(\mathbf{U}^T\mathbf{V}) \geq \mathtt{sum}(\mathbf{U}^T\mathbf{U}) \\ \frac{(\mathbf{V}-\mathbf{U})^T\mathbf{V}}{\|\mathbf{U}-\mathbf{V}\|_2^2}, & else \end{cases} \tag{10}$$

ALGORITHM 1: Deep Multi-Graph Embedding

Input: A multi-graph $\mathcal{G} = (\mathcal{V}, \{\mathcal{E}_1, \ldots, \mathcal{E}_D\})$, and the node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$
Parameter: Θ_s, Θ_d $(d = \{1, \ldots, D\})$, and α_d
Output: Node embedding $\mathcal{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_D\}(\mathbf{X}_d \in \mathbb{R}^{N \times E})$

1: Initialize parameters Θ_s, Θ_d, and α_d.
2: Operate convolution on multi-graph $\mathcal{G}$ by Equation (1) and Equation (2).
3: **for** $d \in [1, D]$ **do**
4: **for** $(v_i, v_j) \in \mathcal{E}_d$ **do**
5: Sample a set of negative samples $\mathcal{S}_{d,n}$.
6: Optimize the loss function Equation (5), and automatically tune α_d by using Equation (10).
7: **end for**
8: **end for**
9: **return** A set of node embedding $\mathcal{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_D\}$

where $\mathbf{U} = \frac{\partial L_1(\Theta_s, \Theta_1)}{\partial \Theta_s}$, $\mathbf{V} = \frac{\partial L_2(\Theta_s, \Theta_2)}{\partial \Theta_s}$, $\mathtt{sum}(\mathbf{U}^T\mathbf{V}) = \sum_i \sum_j (\mathbf{U}^T\mathbf{V})_{ij}$. The weight α updates at each training step, and it is dynamically changing in the training process.

Finally, the loss function can be defined as follows:

$$L = \sum_{d=1}^{D} \alpha_d L_d. \tag{11}$$

The weight α_d and the loss function are trained simultaneously. After training the model, we can obtain the app embedding in each domain, which will be further used for cross-domain app recommendation (as illustrated in Section 4).

Finally, we summarize the learning procedure of DMGE in Algorithm 1. In DMGE, the input includes the multi-graph $\mathcal{G}$ and the node feature matrix $\mathbf{X}$. In line 1, we first initialize the parameter sets Θ_s, Θ_d and the weight α_d of each domain. Then, we operate convolution on the multi-graph $\mathcal{G}$ in line 2. For each subgraph $\mathcal{G}_d$, we sample a set of negative samples in line 5. We use the link information to train DMGE, optimize the loss function Equation (5), and automatically tune α_d by using Equation (10) simultaneously in line 6. Finally, we return a set of node embedding in line 9. The time complexity of the model is $O(NE_s + D(NE_s + NE_sE))$, and the memory complexity is $O(2NE_s + D(NE + E_sE))$. The complexity is linear in the number of nodes (i.e., apps) N and the number of domains D.

4 CROSS-DOMAIN APP RECOMMENDATION

In this section, we will introduce how to achieve mobile apps cross-recommendation based on the app embedding.

Generally, users' preferences can be characterized by their historical downloaded apps, thus we represent users by aggregating embeddings of their downloaded apps [43]. We apply mean aggregator here, and represent users by using the average app embeddings:

$$\mathbf{u}_d = \frac{1}{I_d} \sum_{i=1}^{I_d} \mathbf{x}_{d,i}, \tag{12}$$

where $\mathbf{u}_d$ is the embedding of user $u \in \mathcal{U}$ in domain d, I_d is the number of apps user u has downloaded, and $\mathbf{x}_{d,i}$ is the embedding of app i in domain d.

Table 2. Statistics of Datasets

Dataset	Domain/relation	Nodes	Edges
Tencent App Store	Recommendation	18,229	548,930
	search	18,229	936,065
Youtube	Friendship	15,088	76,765
	co-friends	15,088	1,940,806

Then, for each domain, we measure user-app similarity by computing the cosine distance between user embedding and item embedding. Finally, based on the user-app similarity, we recommend candidate top-K app for each user in each domain.

5 EXPERIMENTS

In this section, we first present the research questions about DMGE. Then, we introduce the datasets and experimental settings. Next, we present the experimental results to demonstrate the effectiveness of DMGE. Finally, we make a discussion of the deep insights of our work.

We first present the following four research questions:

- **RQ1:** How does DMGE perform in the mobile app cross-domain recommendation task compared with other state-of-the-art embedding methods for recommendation?
- **RQ2:** How does the parameter sensitivity affect the performance of DMGE for app recommendation?
- **RQ3:** Does the app embedding generated by DMGE can reflect the similarity between apps?
- **RQ4:** How does DMGE perform in the link prediction task on graph compared with other state-of-the-art graph embedding methods?

5.1 Datasets

We evaluate our model on two real-world datasets, the details of datasets are as follows and the statistics of datasets are presented in Table 2.

- **Tencent App Store:** It is the mobile app download records from a company app store, which contains recommendation domain and search domain. The time span of the dataset is 31 days, the number of apps is 18,229, and the number of user is 1,011,567. Based on the download records, we construct the app graph for each domain, and the statistics of graph is presented in Table 2. We use this dataset for mobile app cross-domain recommendation task.
- **Youtube**[3]**:** The dataset is used to investigate the *RQ4*. The dataset [42] is a multi-dimensional network consists of various type of interactions between users. We utilize two types of relation among users, i.e., friendship and co-friends. The friendship relation means two users are friends, and the co-friends means two users have shared friends. We use this dataset for link prediction task in the multi-graph.

The two datasets are used differently in the experiments. As *RQ1* described, the app store dataset is used to evaluated the app recommendation performance of different methods. As *RQ4* described, the YouTube dataset is used to evaluate the link prediction performance of different graph embedding methods. The goal of our work is mobile app cross-domain recommendation, thus we focus on evaluating the app recommendation performance of different methods. The link prediction task is

[3]http://socialcomputing.asu.edu/datasets/YouTube.

just an auxiliary experiment to evaluate the performance of the method in the classic task on graph (e.g., link prediction). Thus, we perform app recommendation on app store dataset, and perform link prediction on YouTube dataset.

5.2 Experimental Settings

5.2.1 Baseline Methods. For both tasks, we choose the following state-of-the-art graph embedding methods as baselines:

- **DeepWalk** [30]: It applies random walk on graph to generate node sequences, and uses Skip-Gram algorithm to learn embedding. We apply DeepWalk to each subgraph separately.
- **LINE** [34]: It learns node embedding through preserving both local and global graph structures. We apply LINE to each subgraph separately.
- **node2vec** [10]: It designs a biased random walk to explore diverse neighbors. We apply node2vec to each subgraph separately.
- **GCN** [18]: It operates convolution on graph, and generates node embedding based on neighbors. We apply GCN to each subgraph separately.
- **mGCN** [23]: It applies GCNs for multi-graph embedding. It can generate both general embeddings to capture the information for nodes over the entire graph and dimension-specific embeddings to capture the information for nodes in each subgraph.
- **DMGE(α)**: It is a variant of DMGE. It tunes the weight in Equation (6) manually, and α is the weight of the first domain.

For the mobile app cross-domain recommendation task, besides the above baselines, we also compare with MF [19], which factorizes user-item matrix into user embedding and item embedding, respectively. We apply MF to each domain separately.

5.2.2 Evaluation Metrics. To evaluate the performance of app recommendation, we compare the recommended top-K list R_u with the ground truth list T_u for each user u, and use the following metrics to evaluate the top-K recommended results:

- **Recall@K:** It calculates the fraction of the ground truth (i.e., the user downloaded apps) that are recommended by different algorithms using Equation (13), where $\mathcal{U}$ is the user set, h_u denotes the number of downloaded apps hits in the candidate top-K app list R_u for user u, and t_u denotes the number of downloaded app list T_u of user u. A larger value of recall@K means better performance.

$$Recall@K = \frac{\sum_{u \in \mathcal{U}} h_u}{\sum_{u \in \mathcal{U}} t_u}. \tag{13}$$

- **MRR@K:** Mean Reciprocal Rank (MRR) uses the multiplicative inverse of the rank of the first hit app among top-K app list to evaluate the performance of rank by Equation (14), where r_u is the rank of the first hit app. A larger value of MRR@K means better performance.

$$MRR@K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{r_u}. \tag{14}$$

To evaluate the performance of link prediction, we use the metrics of classification: *AUC* and *F1*.

5.2.3 Model Parameters. The parameters of DMGE are set as follows:

- *Network architecture.* The number of shared and specific embedding layers are both 1, the shared hidden size is 64, and the specific hidden size is 16.

Table 3. Recall@K Performance of Different Methods in Recommendation Domain

Domain	Recall@K	10	20	30	40	50	60	70	80	90	100	1000
Single	MF	0.0301	0.0453	0.0565	0.0658	0.0739	0.0812	0.0880	0.0942	0.1002	0.1065	0.2932
	DeepWalk	0.0730	0.1104	0.1363	0.1558	0.1720	0.1853	0.1975	0.2082	0.2186	0.2273	0.4744
	LINE	0.0622	0.0908	0.1128	0.1311	0.1487	0.1627	0.1756	0.1880	0.1987	0.2085	0.5032
	node2vec	0.0345	0.0579	0.0773	0.0936	0.1080	0.1207	0.1324	0.1436	0.1534	0.1630	0.4574
	GCN	0.0773	0.1139	0.1402	0.1642	0.1843	0.1981	0.2134	0.2267	0.2389	0.2487	0.5693
Cross	mGCN	0.0431	0.0835	0.1273	0.1677	0.2002	0.2142	0.2261	0.2383	0.2505	0.2627	0.6323
	DMGE(0.5)	0.1019	0.1607	0.2069	0.2436	0.2762	0.3035	0.3260	0.3471	0.3660	0.3826	**0.7016**
	DMGE	**0.1024**	**0.1661**	**0.2109**	**0.2455**	**0.2767**	**0.3042**	**0.3277**	**0.3484**	**0.3669**	**0.3831**	0.6929

— *Initialization*. The node feature matrix can be initialized randomly, or by other embedding methods, we initialize it as the identity matrix.
— *Gradient normalization*. We normalize the gradient of shared parameter Θ_s of each domain, and then use the normalized gradient to calculating α in Equation (10). The normalized gradient of domain d is $\mathbf{G}_d / (\|\mathbf{G}_d\|_2 \cdot L_d)$, where $\mathbf{G}_d = \frac{\partial L_d(\Theta_s, \Theta_d)}{\partial \Theta_s}$ is the unnormalized gradient.
— *Other hyper-parameters*. The number of negative samples is 2; the embedding dimension is 16; the dropout of shared embedding layers is 0.3 and that is 0.1 of specific embedding layers; the batch size is 256 and we train the model using Adam [17].

The parameters of baselines are fine tuned, and set as follows:
(1) MF, it is implemented using LibMF.[4] *(2) DeepWalk*, the length of context window is 5; the length of random walk is 20; the number of walks per node is 50. *(3) LINE*, the number of negative samples is 2. *(4) node2vec*, p is 1 and q is 0.25; other parameter settings are the same as DeepWalk. *(5) GCN*, the number of negative samples is 2. *(6) mGCN*, the initial general embedding size is 64, other parameter settings are the same as [23]. *(7) DMGE(α)*, considering that both domains are important, we set the weight α to 0.5; the other parameter settings are the same as DMGE.

5.3 Mobile App Cross-Domain Recommendation

To demonstrate the performance of DMGE in app recommendation task (*RQ1*), we compare DMGE with other state-of-the-art embedding methods. The intuition is that better embeddings will achieve better performance of recommendation.

For the app store dataset, we follow the commonly-used data splitting method in recommendation task [37, 43], i.e., splitting the data by time. We use data in consecutive 26 days to train app embedding, and measure the performance of app recommendation in the next 5 days by the metric Recall@K and MRR@K. The performance of different methods for recommendation domain and search domain is presented in Tables 3–6. (Note that the best results are indicated by the bold font.)

Based on the results, we have the following observations:

— We first compare the performance of single-domain methods, including: MF, DeepWalk, LINE, node2vec, and GCN. We can observe the graph embedding methods outperforms MF, as MF only takes into account the explicit user-app interactions, while ignoring app dependency in users' behaviors, which can reflect users' preferences.
— The overall performance of cross-domain methods (i.e., mGCN, DMGE(α), DMGE) is better than the single domain methods, which demonstrates that fusing information from

[4] https://www.csie.ntu.edu.tw/~cjlin/libmf/.

Table 4. Recall@K Performance of Different Methods in Search Domain

Domain	Recall@K	10	20	30	40	50	60	70	80	90	100	1000
Single	MF	0.0150	0.0251	0.0335	0.0408	0.0474	0.0533	0.0589	0.0642	0.0691	0.0739	0.2679
	DeepWalk	0.0638	0.1043	0.1338	0.1571	0.1761	0.1924	0.2064	0.2185	0.2291	0.2387	0.4676
	LINE	0.0546	0.0834	0.1044	0.1210	0.1348	0.1472	0.1584	0.1685	0.1774	0.1861	0.4439
	node2vec	0.0289	0.0471	0.0622	0.0753	0.0870	0.0982	0.1082	0.1174	0.1260	0.1346	0.4176
	GCN	0.0724	0.1089	0.1342	0.1534	0.1684	0.1812	0.1923	0.2023	0.2115	0.2201	0.5132
Cross	mGCN	0.0478	0.0939	0.1454	0.1938	0.2218	0.2328	0.2399	0.2480	0.2565	0.2653	0.5920
	DMGE(0.5)	0.0823	0.1363	0.1784	0.2134	0.2415	0.2652	0.2857	0.3037	0.3206	0.3360	0.6254
	DMGE	**0.0885**	**0.1467**	**0.1900**	**0.2238**	**0.2517**	**0.2759**	**0.2971**	**0.3162**	**0.3328**	**0.3473**	**0.6263**

Table 5. MRR@K Performance of Different Methods in Recommendation Domain

Domain	MRR@K	10	20	30	40	50	60	70	80	90	100	1000
Single	MF	0.0149	0.0170	0.0180	0.0185	0.0188	0.0191	0.0193	0.0194	0.0196	0.0197	0.0208
	DeepWalk	0.0510	0.0549	0.0563	0.0571	0.0575	0.0578	0.0581	0.0582	0.0584	0.0585	0.0594
	LINE	0.0532	0.0561	0.0573	0.0580	0.0585	0.0588	0.0590	0.0592	0.0594	0.0595	0.0607
	node2vec	0.0265	0.0290	0.0302	0.0308	0.0312	0.0315	0.0317	0.0319	0.0321	0.0322	0.0334
	GCN	0.0641	0.0681	0.0695	0.0703	0.0708	0.0711	0.0713	0.0715	0.0717	0.0718	0.0729
Cross	mGCN	0.0264	0.0311	0.0338	0.0354	0.0364	0.0367	0.0370	0.0372	0.0373	0.0375	0.0389
	DMGE(0.5)	0.0697	0.0756	0.0780	0.0793	0.0801	0.0807	0.0811	0.0814	0.0816	0.0817	0.0829
	DMGE	**0.0699**	**0.0761**	**0.0785**	**0.0797**	**0.0805**	**0.0810**	**0.0814**	**0.0817**	**0.0819**	**0.0821**	**0.0832**

Table 6. MRR@K Performance of Different Methods in Search Domain

Domain	MRR@K	10	20	30	40	50	60	70	80	90	100	1000
Single	MF	0.0120	0.0134	0.0141	0.0145	0.0148	0.0150	0.0151	0.0152	0.0153	0.0154	0.0164
	DeepWalk	0.0468	0.0515	0.0534	0.0543	0.0549	0.0553	0.0556	0.0558	0.0560	0.0561	0.0571
	LINE	0.0466	0.0500	0.0514	0.0522	0.0526	0.0530	0.0532	0.0534	0.0536	0.0537	0.0548
	node2vec	0.0237	0.0260	0.0271	0.0278	0.0283	0.0286	0.0289	0.0290	0.0292	0.0293	0.0307
	GCN	0.0558	0.0600	0.0616	0.0625	0.0630	0.0633	0.0635	0.0637	0.0639	0.0640	0.0651
Cross	mGCN	0.0324	0.0382	0.0415	0.0435	0.0443	0.0446	0.0448	0.0449	0.0450	0.0452	0.0465
	DMGE(0.5)	0.0592	0.0653	0.0678	0.0691	0.0700	0.0705	0.0709	0.0712	0.0714	0.0716	0.0728
	DMGE	**0.0629**	**0.0693**	**0.0718**	**0.0731**	**0.0739**	**0.0744**	**0.0748**	**0.0751**	**0.0753**	**0.0755**	**0.0766**

correlated domains is helpful to learn better app embedding, and can improve the performance of recommendation in both domains. When K is less than 40 in recommendation domain and K is less than 30 in search domain, the Recall of mGCN is worse than the single domain methods, the possible reason is that the weight between within-domain and across-domain in mGCN is a hyper-parameter to be tuned, and cannot be adaptively learned by the importance of each domain. Both DMGE and DMGE(α) outperform the single domain methods.

— Comparing the cross-domain methods, both DMGE(α) and DMGE outperform mGCN, which indicates that our model is effective to learn better app embeddings.

— DMGE outperforms DMGE(α). We find that the average of α in DMGE is 0.4409, thus when $\alpha = 0.5$, DMGE(α) can also achieve good performance. However, in DMGE(α), it is

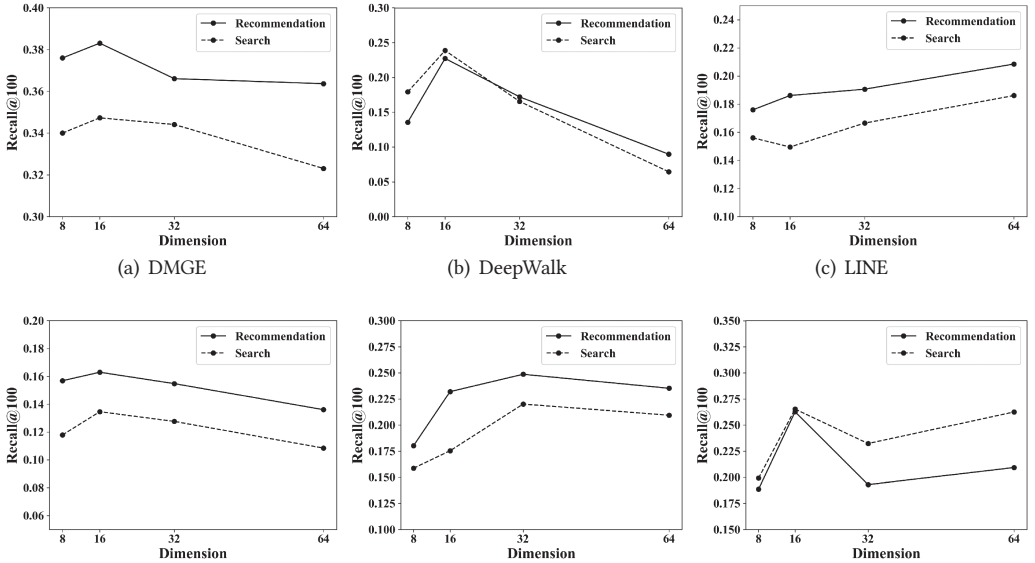


Fig. 3. The parameter sensitivity analysis of embedding dimension.

time-consuming and computationally expensive to tune the hyper-parameter α manually to obtain the optimal result. While in DMGE, α is a trainable parameter. Thus, we recommend to use the adaptive method to train the model.

Overall, the proposed DMGE outperforms the state-of-the-art embedding methods, and improves the performance of app recommendation in both domains.

5.4 Parameter Sensitivity

The key parameter that affects the performance of app recommendation is the dimension size of app embedding (*RQ2*), we analyze how does the dimension size of app embedding affect the performance of different graph embedding methods. In particular, we test the dimension size = $\{8, 16, 32, 64\}$. Figure 3 show the results of different embedding dimension in recommendation and search domain, and the evaluation metric is Recall@100.

As shown in Figure 3(a), in both domains, when the dimension of app embedding is 16, DMGE performs the best. Therefore, we set the dimension of app embedding as 16.

Figure 3(b)–(f) shows the results of the graph embedding methods; we present the optimal dimension of different methods as follows: the dimension in DeepWalk is set to 16; the dimension in LINE is set to 64; the dimension in node2vec is set to 16; the dimension in GCN is set to 32; and the dimension in mGCN is set to 16.

5.5 Case Study: App-App Similarity

To evaluate whether the app embedding generated by DMGE can reflect the similarity between apps (*RQ3*), we conduct a case study, and measure the app–app similarity by computing cosine distance between the embeddings of two apps. Table 7 presents the seed app and their top three similar apps in recommendation domain. (Note that inside the bracket is the category of the app.)

We can observe that the seed app Youku Video (Taobao, Cross Fire, or B612) has the same category and functionality as its top-3 similar apps. Considering the seed app WeChat, though the

Table 7. App-App Similarity in Recommendation Domain

App	Top-3 similar apps
WeChat (*Social*)	QQ (*Social*), Tencent Video (*Video*), Kuaishou (*Social*)
Youku Video (*Video*)	Baidu Video (*Video*), iQiYi Video (*Video*), Sohu Video (*Video*)
Taobao (*Shopping*)	JD.com (*Shopping*), Pinduoduo (*Shopping*), TMall (*Shopping*))
TouTiao (*Reading*)	Tencent News (*Reading*), Xigua Video (*Video*), Kuaibao (*Reading*)
Cross Fire (*Game*)	Happy Poker (*Game*), Wangzhe Rongyao (*Game*), Clash Royale (*Game*)
B612 (*Photography*)	Facue (*Photography*), BeautyCam (*Photography*), Meitu XiuXiu (*Photography*)

category of Tencent Video is different from it, both of them are developed by the same developer, besides, users can also share the interesting videos in Tencent Video to their WeChat friends. So, though the category of Tencent Video is *Video*, it still has social properties to some extent. Considering the seed app Toutiao, though the category of Xigua Video is different from it, both of them are developed by the same developer, besides, both of these two apps share some of the same video content. Although the category of these apps are different, they are still relevant.

We find that the app embedding generated by DMGE can capture the similarity between apps.

5.6 Link Prediction

To demonstrate the performance of DMGE in link prediction task (*RQ4*), we compare DMGE with other state-of-the-art graph embedding methods. The intuition is that better embeddings will achieve better performance of link prediction.

In the multi-graph, we perform link prediction in different subgraph separately. In each subgraph, we randomly remove 30% of edges, and we aim to predict whether these removed edges exist. We formulate the link prediction task as a binary classification problem by using the embeddings of two nodes, and there are two types of combination: element-wise addition, element-wise multiplication.

In training set, we use the remaining node pairs as positive samples, and randomly sample an equal number of not connected node pairs as negative samples. In testing set, we use the removed node pairs as positive samples, and randomly sample an equal number of not connected node pairs as negative samples. We train a binary classifier using logistic regression on the training set, and evaluate the performance of link prediction on the testing set. For each method, we choose the optimal combination of embeddings and present the best results. The average performance of different methods are presented in Figure 4. For metric AUC, the standard deviations of all methods are as follows: DeepWalk is 0.0440, LINE is 0.0388, node2vec is 0.0367, GCN is 0.0038, mGCN is 0.0053, DMGE(α) is 0.0088, and DMGE is 0.0031. For metric F1, the standard deviations of all methods are as follows: DeepWalk is 0.0502, LINE is 0.0662, node2vec is 0.0434, GCN is 0.0115, mGCN is 0.0011, DMGE(α) is 0.0066, and DMGE is 0.0064.

Based on the results, we have the following observations:

- —The multi-graph embedding methods (i.e., mGCN, DMGE(α), and DMGE) outperform the single graph embedding methods (i.e., DeepWalk, LINE, node2vec, and GCN), which indicates that using multiple relations in the multi-graph is helpful to learn better embedding.
- —DMGE(α) and DMGE outperform mGCN, which indicates that our proposed graph neural network is effective to learn better embeddings.
- —The average performance of DMGE is better than DMGE(α), which indicates the effectiveness of training the model in the adaptive way.

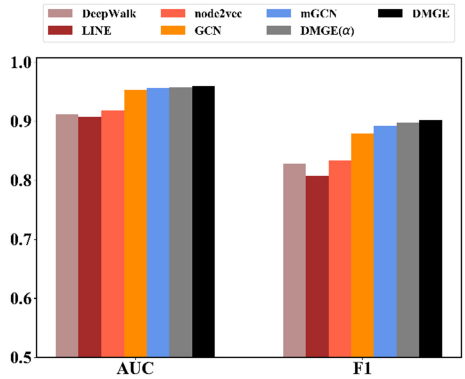


Fig. 4. The performance of different methods in link prediction.

5.7 Discussion

We next discuss the potential future directions to improve this work.

5.7.1 Node Features. To take full advantage of the information of user–app interaction data, aside from using them for graph construction, we can also use them as node features. Using the user–app interaction data as additional features, we can represent the node features by encoding both its own information (i.e., the one-hot features) and the user–app interaction information. In particular, the user–app interaction information can reveal users' preferences for apps. Thus, it is helpful to learn effective node embeddings by using the user–app interaction data as the additional features.

5.7.2 The Usage of Embedding. The embeddings of DMGE can be used for candidate apps generation in the recall stage. Through calculating the pairwise similarities between the embeddings of users and apps, we can generate a candidate set of apps which users may like, and the candidate set can be further used in the ranking stage to generate the final recommendation set of apps [5]. Besides, the embeddings can also be used for transfer learning [27] and alleviating the sparsity [43].

5.7.3 Cross-Domain Representation Learning. In app store, there are many domains, in addition to the recommendation and search domain studied in this article, there are game domain (i.e., the game page in app store), ranking domain (i.e., the app ranking page in app store), and so on. When there are multiple domains, using the Frank–Wolfe algorithm [15, 33], we can efficiently obtain the weight α_d of each domain in Equation (8). With the learned weight α_d, we can optimize the loss function in Equation (11) to train the model.

5.7.4 Scalability. The training time of the model is 30 minutes per epoch on average, and testing time is 35 minutes per epoch on average. The embedding layers in DMGE adopt the graph convolution operator in GCN [18]. However, GCN requires the full graph Laplacian, thus it is computationally expensive to apply GCN for large-scale graph embedding. To apply DMGE for large-scale multi-graph embedding, we have the following strategy: we can adopt GraphSAGE [12] as the embedding layers in DMGE, as GraphSAGE generates embeddings by sampling and aggregating features from a node's local neighborhood, and only requires local graph structures.

6 RELATED WORK

In this section, we review the relevant works in four areas that best line up with our research.

6.1 Cross-Domain Recommendation

Cross-domain RSs [3] aim to fuse information from correlated domains to improve the performance of recommendation. Based on the information shared by different domains, existing works on cross-domain recommendation mainly fall into two categories, i.e., *user-shared* and *item-shared*. In *user-shared cross-domain recommendation* [14], user embeddings or profiles are usually shared across domains. However, user information in different domains may not be directly accessible due to privacy protection or the absence of user profiles, thus it is difficult to directly obtain the shared user information. In *item-shared cross-domain recommendation* [24, 45], MF techniques [19, 31] are often used to factorize user-item interaction matrix into user and item embeddings, then item embeddings are shared or transferred across domains. However, these works only consider the user-item interaction, while ignore the item dependency (i.e., the item co-occurrences in users' behaviors). The item dependency has been proved to be also important to capture item-item similarity and reflect users' preferences [36, 43], we are thus motivated to achieve cross-domain recommendation based on the app dependency in users' behaviors.

6.2 Mobile Apps Recommendation

Mobile app recommendation has attracted an increasing number of attention of researchers, which aims to recommend each user with a personalized set of apps. Existing works on mobile app recommendation can be mainly divided into three categories. *Context-aware app recommendation* [16, 47], which aims to recommend relevant apps to user based on his/her current mobile context information (e.g., time and location). A tensor is often used to represent user-app-context interaction in users' mobile app usage history. Then tensor decomposition methods [16] are commonly used to represent user' preferences, app information, and context information to facilitate context-aware app recommendation. *Privacy protection-based app recommendation* [22, 46], which aims to protect user privacy in mobile app recommendation, as mining and understanding user preferences may also leak users' privacy information. Specifically, the trade-off between users' privacy preferences and apps' attributes (e.g., popularity and functionality) is considered to achieve mobile app recommendation. *Cold-start app recommendation* [21], which aims to recommend newly released apps with no ratings to users. Relevant information from Twitter about the apps is used to solve the cold-start app recommendation problem. Different from the above studies, we focus on users' download behaviors in mobile app store, and aim to leverage the complementary information from correlated domains in mobile app store to facilitate cross-domain app recommendation.

6.3 Embedding Methods for Recommendation

Representation learning [1] is one of the most fundamental problems in deep learning. As a practical application, effective embedding has been proven to be useful and achieve significant performance in RSs including: E-commerce [36, 43], search ranking [9], and social media [40]. The embedding methods in RSs can be divided into two categories: word embedding-based methods and graph embedding-based methods. The former methods [9, 43] learn embedding by modeling the item co-occurrence in users' behavior sequences. Specifically, they model the items as words and user's behavior sequences as sentences, and apply the word embedding methods [25, 26] to represent items in a low-dimensional space. While the latter methods [36, 40] construct item graph based on users' behaviors, they model the items as nodes and item co-occurrences as edges, and apply the graph embedding methods [6, 12, 13, 30] to learn embedding. However, these methods are developed to learn embedding in a single domain, which fail to learn effective cross-domain embedding.

6.4 Graph Neural Network

GNNs [38, 39] have emerged as a powerful approach for representation learning on graphs recently, such as GCN [18], GraphSAGE [12], and GAT [35]. Through a recursive neighborhood aggregation scheme, GNNs can generate node embedding by aggregating features of neighbors. In this part, we focus on reviewing related works about the convolution based GNNs, which can be categorized as spectral approaches and non-spectral approaches.

The spectral approaches depend on the theory of spectral graph convolutions. Bruna et al. [2] first propose a generalization of convolutional neural networks to graphs, however, it is computationally expensive. Defferrard et al. [7] design K-localized convolutional filters on graphs based on spectral graph theory, which is more computationally efficient. Kipf et al. [18] limit the layer-wise convolution operation to $K = 1$ to avoid overfitting, and propose the GCN to encode both local graph structure and features of nodes by layer-wise propagation. The non-spectral approaches operate spatial convolutions on the graph. Hamilton et al. [12] propose GraphSAGE to generate node embeddings by sampling and aggregating features from a node's local neighborhood. However, these GNNs are developed for single graph embedding, which fail to learn effective multi-graph embedding, because these GNNs can only generate the node embeddings, which encode the information from all domains, i.e., domain-shared embeddings. However, directly applying the same embeddings learned by GNNs to all domains may not be applicable, because each domain has specific information, and the same embedding cannot capture the specific characteristic of each domain. The main difference between DMGE and GNNs is that, DMGE learns both the domain-shared embedding and the domain-specific embedding, while GNNs can only generate the shared node embedding.

There are also GNNs for learning multi-graph embedding, such as: Graph Transformer Network (GTN) [41], which is designed to learn node representation in the heterogeneous graph. The main difference between GTN and DMGE is that, the functionalities of these two methods are different. GTN can only generate one type of embedding for each node, and cannot generate multiple types of embeddings for each node to adaptive to multiple domains. We cannot directly apply the same embedding of a node learned by GTN to all domains, because each domain has specific information, and the same embedding cannot capture the specific characteristic of each domain. While DMGE first utilizes the domain-shared embedding layers to learn shared embedding of nodes in the multi-graph. Then, it further feeds the domain-shared node embeddings into different domain-specific embedding layers to learn specific embedding of nodes in different domain. The learned domain-specific embedding can be used to solve the app recommendation task in the domain.

7 CONCLUSION

In this article, we aim to leverage the complementary information from correlated domains in app store to facilitate mobile app cross-domain recommendation. We propose the DMGE model, which is a graph neural network based on multi-task learning. We construct the cross-domain app graph as a multi-graph based on users' behaviors from different domains in app store, and then design a multi-graph neural network to learn multi-graph embedding. Particularly, we present an adaptive method to balance the weight of different domains and efficiently training the model. Finally, we achieve cross-domain app recommendation based on the learned app embedding. We evaluate our approach on large-scale real-world datasets, and the experimental results show that DMGE outperforms other state-of-the-art embedding methods.

REFERENCES

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 8 (2013), 1798–1828.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *Proceedings of the 2nd International Conference on Learning Representations*. 1–14.
[3] Iván Cantador, Ignacio Fernández-Tobías, Shlomo Berkovsky, and Paolo Cremonesi. 2015. Cross-domain recommender systems. In *Recommender Systems Handbook*. Springer, 919–959.
[4] Rich Caruana. 1997. Multitask learning. *Machine Learning* 28, 1 (1997), 41–75.
[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for Youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.
[6] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* 31, 5 (2018), 833–852.
[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the Advances in Neural Information Processing Systems*. 3844–3852.
[8] Jean-Antoine Désidéri. 2012. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus Mathematique* 350, 5–6 (2012), 313–318.
[9] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time personalization using embeddings for search ranking at Airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 311–320.
[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.
[11] Bin Guo, Yi Ouyang, Tong Guo, Longbing Cao, and Zhiwen Yu. 2019. Enhancing mobile app user understanding and marketing with heterogeneous crowdsourced data: A review. *IEEE Access* 7 (2019), 68557–68571.
[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the Advances in Neural Information Processing Systems*. 1024–1034.
[13] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin* 40, 3 (2017), 52–74.
[14] Guangneng Hu, Yu Zhang, and Qiang Yang. 2018. Conet: Collaborative cross networks for cross-domain recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 667–676.
[15] Martin Jaggi. 2013. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning*. 427–435.
[16] Alexandros Karatzoglou, Linas Baltrunas, Karen Church, and Matthias Böhmer. 2012. Climbing the app wall: Enabling mobile app discovery through context-aware recommendations. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. ACM, 2527–2530.
[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of 3rd International Conference on Learning Representations*. 1–14.
[18] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*. 1–14.
[19] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[20] H. W. Kuhn and A. W. Tucker. 1951. Nonlinear programming. In *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probability*. 481–492.
[21] Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. 2013. Addressing cold-start in app recommendation: Latent user models constructed from twitter followers. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 283–292.
[22] Bin Liu, Deguang Kong, Lei Cen, Neil Zhenqiang Gong, Hongxia Jin, and Hui Xiong. 2015. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*. ACM, 315–324.
[23] Yao Ma, Suhang Wang, Chara C. Aggarwal, Dawei Yin, and Jiliang Tang. 2019. Multi-dimensional graph convolutional networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 657–665.
[24] Tong Man, Huawei Shen, Xiaolong Jin, and Xueqi Cheng. 2017. Cross-domain recommendation: An embedding and mapping approach. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 2464–2470.
[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of the 1st International Conference on Learning Representations*. 1–11.
[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Advances in Neural Information Processing Systems*. 3111–3119.
[27] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. 2018. Perceive your users in depth: Learning universal user representations from multiple e-commerce tasks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 596–605.

[28] Yi Ouyang, Bin Guo, Tong Guo, Longbing Cao, and Zhiwen Yu. 2018. Modeling and forecasting the popularity evolution of mobile apps: A multivariate Hawkes process approach. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 182.

[29] Yi Ouyang, Bin Guo, Xinjiang Lu, Qi Han, Tong Guo, and Zhiwen Yu. 2019. Competitivebike: Competitive analysis and popularity prediction of bike-sharing apps using multi-source data. *IEEE Transactions on Mobile Computing* 18, 8 (2019), 1760–1773.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 701–710.

[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 452–461.

[32] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[33] Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. In *Proceedings of the Advances in Neural Information Processing Systems*. 527–538.

[34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of the International Conference on Learning Representations*. 1–12.

[36] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in Alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 839–848.

[37] Yaqing Wang, Chunyan Feng, Caili Guo, Yunfei Chu, and Jenq-Neng Hwang. 2019. Solving the sparsity problem in recommendations via cross-domain item embedding based on co-clustering. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. ACM, 717–725.

[38] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2019. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? In *Proceedings of the International Conference on Learning Representations*. 1–17.

[40] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[41] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. 2019. Graph transformer networks. In *Proceedings of the Advances in Neural Information Processing Systems*. 11960–11970.

[42] R. Zafarani and H. Liu. 2009. Social Computing Data Repository at ASU. Retrieved from http://socialcomputing.asu.edu.

[43] Kui Zhao, Yuechuan Li, Zhaoqian Shuai, and Cheng Yang. 2018. Learning and transferring ids representation in e-commerce. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1031–1039.

[44] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

[45] Feng Zhu, Yan Wang, Chaochao Chen, Guanfeng Liu, Mehmet Orgun, and Jia Wu. 2018. A deep framework for cross-domain and cross-system recommendations. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[46] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. 2014. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 951–960.

[47] Konglin Zhu, Lin Zhang, and Achille Pattavina. 2017. Learning geographical and mobility factors for mobile application recommendation. *IEEE Intelligent Systems* 32, 3 (2017), 36–44.

[48] Fuzhen Zhuang, Yingmin Zhou, Fuzheng Zhang, Xiang Ao, Xing Xie, and Qing He. 2017. Sequential transfer learning: Cross-domain novelty seeking trait mining for recommendation. In *Proceedings of the 26th International Conference on World Wide Web Companion*. 881–882.

Received March 2020; revised July 2020; accepted December 2020