Deoscillated Graph Collaborative Filtering

Zhiwei Liu[*1], **Lin Meng**[*2], **Jiawei Zhang**[2], **Philip S. Yu**[1]
[1]University of Illinois at Chicago, IL, USA {zliu213, psyu}@uic.edu
[2]Florida State University, FL, USA {lin, jiawei}@ifmlab.org

Abstract

Collaborative Filtering (CF) signals are crucial for a Recommender System (RS) model to learn user and item embeddings. High-order information can alleviate the cold-start issue of CF-based methods, which is modelled through propagating the information over the user-item bipartite graph. Recent Graph Neural Networks (GNNs) propose to stack multiple aggregation layers to propagate high-order signals. However, the oscillation problem, varying locality of bipartite graph, and the fix propagation pattern spoil the ability of multi-layer structure to propagate information. The oscillation problem results from the bipartite structure, as the information from users only propagates to items. Besides oscillation problem, varying locality suggests the density of nodes should be considered in the propagation process. Moreover, the layer-fixed propagation pattern introduces redundant information between layers. In order to tackle these problems, we propose a new RS model, named as **D**eoscillated **G**raph **C**ollaborative **F**iltering (DGCF). We introduce cross-hop propagation layers in it to break the bipartite propagating structure, thus resolving the oscillation problem. Additionally, we design innovative locality-adaptive layers which adaptively propagate information. Stacking multiple cross-hop propagation layers and locality layers constitutes the DGCF model, which models high-order CF signals adaptively to the locality of nodes and layers. Extensive experiments on real-world datasets show the effectiveness of DGCF. Detailed analyses indicate that DGCF solves oscillation problem, adaptively learns local factor, and has layer-wise propagation pattern. Our code is available online[1].

Introduction

A Recommender System (RS) (Rendle et al. 2009; Rendle 2010; He et al. 2017; Wang et al. 2019; Zheng, Noroozi, and Yu 2017; Ying et al. 2018) can predict the interests of users towards items, where a typical method is to model the Collaborative Filtering (CF) signals (Rendle et al. 2009; He et al. 2017; Zheng et al. 2018; Wang et al. 2019). Assuming similar users share relevant items, CF signals help to learn the user and item embeddings (Rendle et al. 2009; He et al. 2017; Ying et al. 2018; Zheng et al. 2018) upon given user-item interactions. One major issue regarding CF methods is the cold-start problem (Zheng et al. 2018; Yang et al. 2018; Zhang and Chen 2020), where embeddings of users or items with few interactions are hard to learn. To remedy this issue, we can model *high-order signals* (Zheng et al. 2018; Yang et al. 2018; Liu et al. 2019; Wang et al. 2019) that propagate information from multiple hops away over the user-item bipartite graph. Recently, owing to the development of Graph Neural Network (GNN) (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017b; Wang et al. 2019), we can easily propagate high-order signals (Wang et al. 2019; Liu et al. 2019) by stacking multiple *aggregation layers* (Kipf and Welling 2017; Zheng et al. 2018; Wang et al. 2019). Therefore, users (items) can aggregate the first-order signal from direct neighbors at the first layer and high-order signals from indirect neighbors at deep layers. However, via experimental analyses on real-world datasets, we observe three common performance problems of existing GNNs on RS, which are named as the **oscillation** problem, **varying locality** problem, and **fixed propagation pattern** problem in this paper, respectively.

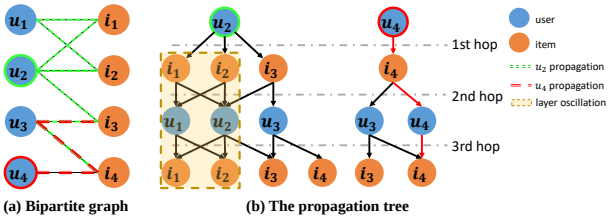


Figure 1: The left part is an example of bipartite graph. The green and red color labels the edges covered by 3-hop propagation of u_2 and u_4, respectively. The right part presents the corresponding 3-hop propagation tree of u_2 and u_4.

Firstly, stacking multiple layers leads to the oscillation problem, which in turn hinders the information to propagate. Before formally defined, intuitively, oscillation problem occurs if there is an information gap between the propagation of successive hops. It results from the bipartite structure. As illustrated in the left part of Figure 1, the direct neighbors of users are all items. This implies that by aggregating the direct neighbors, users only receive the information from items, and vice versa. It turns out that the information oscillates between users and items due to the bipartite graph

[*]Equal contributions
Preprint. Under review
[1]https://github.com/JimLiu96/DeosciRec

structure. One quick solution is adding self-loops (Kipf and Welling 2017; Wang et al. 2019) to break the bipartite structure. However, we theoretically prove that it cannot alleviate the oscillation problem. Additionally, adding self-loops is equivalent to not propagating the information, which even exacerbates the cold-start issue.

Besides the oscillation problem, the varying locality problem of bipartite graph also limits the propagating ability of the multi-layer structure (Xu et al. 2018). It refers to the density of local structures over the bipartite graph. For example, in the left-hand side of Figure 1, u_4 has a lower local density compared with u_2 as the former only directly connects to i_4 while the later connects to i_1, i_2 and i_3. The 3-hop propagation starting from u_4 (red edges in Figure 1) only covers 3 edges in the bipartite graph. In contrast, the 3-hop propagation of u_2 (green edges in Figure 1) covers almost every edges in the bipartite graph. Specifically, we present the corresponding propagation tree of u_2 and u_4 on the right-hand side of Figure 1. There are fewer possible propagation paths for u_4 compared with u_2.

Last but not the least, the fixed propagation pattern at different layers of existing multi-layer RS models (He et al. 2020; Wang et al. 2019; Berg, Kipf, and Welling 2017) induces redundant information propagation between layers. We present an example of the redundancy as the yellow-shaded block on the right-hand side of Figure 1. The first hop propagation distributes the information of u_2 to i_1 and i_2. Then, the information from i_1 and i_2 is propagated to u_1 and u_2 at the second hop propagation, which is reversely propagated back at the third hop propagation. This is due to the propagation pattern is fixed at different layers. One possible solution is to sample layer-dependent neighbors (Zou et al. 2019). However, current sampling strategy requires additional computation and is not differentiable, which spoils the training procedure.

In order to tackle the aforementioned problems, we design a new GNN framework specifically for bipartite graphs, named deep **D**eoscillated **G**raph **C**ollaborative **F**iltering (DGCF). We propose a Cross-Hop Propagation (CHP) layer which remedies the bipartite propagating structure to resolve the oscillation problem. Compared with existing methods that aggregates only direct neighbors, CHP layer also propagates the information from cross-hop neighbors. As a result, users (items) can also receive the information from cross-hop users (items). Moreover, we design a Locality-Adaptive (LA) layer which controls the amount of information to propagate. It learns an influence factor for each node, which is adaptive to the varying locality of the graph. Multiple CHP layers and LA layers constitutes the DGCF model. It propagates the high-order CF signal to train user embeddings and item embeddings, which is adaptive to the locality of nodes. The weights of LA layers and CHP layers are different from layer to layer, thus being a layer-wise propagation pattern. The contributions of our paper are:

- **Oscillation Problem:** We formally define the graph oscillation problem and prove its existence and boundary in bipartite graph. We also provide empirical evidence that oscillation occurs with a few layers.
- **Innovative modules:** We design two innovative layers, i.e., CHP layer and LA layer. CHP layer is designed to overcome oscillation problem, while LA layer is for varying locality problem. Stacking multiple LA layers resolves the fixed aggregation pattern problem.
- **Extensive experiments:** In addition to overall comparison experiment, we conduct three types of detailed analyses regarding three aforementioned problems. We study the oscillation problem when stacking too many layers. Additionally, we discuss the varying locality problem and how our model resolves it. Moreover, we observe the variations of the propagation pattern.

Preliminary and Related Work

Nowadays, Graph Neural Networks (GNNs) (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017b; Wu et al. 2019) are widely used in recommender system (RS) (He et al. 2020; Ying et al. 2018; Wang et al. 2019). In this section, we review the basic concepts of GNNs and their association with RS.

Graph Neural Network

Given a simple graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ denote the nodes and edges, respectively. The input feature of node v can be denoted as $\mathbf{x}_v \in \mathbb{R}^d$ and the hidden feature learned at l-th layer is denoted by $\mathbf{h}_v^{(l)} \in \mathbb{R}^h$, where the d is dimension of the input feature and h is the dimension of the hidden features. We define $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. Generally, GNN models have aggregation layer that aggregates features of selected neighboring nodes (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017a; Liu et al. 2020; Zou et al. 2019; Chen, Ma, and Xiao 2018; Gao, Wang, and Ji 2018; Derr, Ma, and Tang 2018). The information is propagated from the selected nodes. Note that 'aggregation' describes the information diffusion from the view of target nodes, while 'propagation' is from the view of source nodes. Hence, we use them interchangeably in the remaining parts. Formally, the general propagation layer can be denoted by:

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu} \mathbf{h}_u^{(l-1)} \mathbf{W}^{(l-1)}\right), \qquad (1)$$

where the set $\mathcal{N}_v = \{u \in \mathcal{V} | (u, v) \in \mathcal{E}\}$ is the selected neighboring node set of node v, $\mathbf{W}^{(l)}$ denotes a linear transformation, and σ denotes a non-linear activation function (e.g. ReLU). α_{uv} denotes the coefficient between node v and neighboring node u. Particularly, GCN (Kipf and Welling 2017) directly aggregates all the neighboring nodes:

$$\mathbf{H}^{(l)} = \sigma\left(\tilde{\mathbf{A}} \mathbf{H}^{(l-1)} \mathbf{W}^{(l-1)}\right), \qquad (2)$$

where $\tilde{\mathbf{A}}$ is defined as the Laplacian normalized adjacency matrix $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\mathbf{D}^{-\frac{1}{2}}$, and $\mathbf{H}^{(0)} = \mathbf{X}$. The GCN layer can learn its 1-st hop neighbors, which can be also seen as one step information propagation from the starting node. By stacking L layers, node features propagate to the nodes on the L-th hop. Note that $\tilde{A}$ can also be random walk

normalization, i.e., $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$ (Zhang and Meng 2019). The nonlinear activation between layers and redundant linear transformations are removed later in SGC (Wu et al. 2019), $\mathbf{H}^{(l)} = \tilde{\mathbf{A}}^L\mathbf{X}\mathbf{W}$. Instead of stacking L layers, SGC directly calculates the L powers of $\tilde{\mathbf{A}}$ as multi-hop aggregation, which greatly reduces the number of linear transformation matrix to one rather than L.

Recommender System

In typical Recommender Systems (RS), user-item interaction can be represented as a user-item bipartite graph. Meanwhile, GCNs shows unprecedented representation power on many areas including RS, where the collaborative filtering signal can be modelled via the high-order information propagation. Recently, NGCF (Wang et al. 2019) is proposed specifically for recommendation, which successfully models the collaborative signal in bipartite graphs. Different from GCN layers, a NGCF propagation layer additionally includes dot product to model the propagation messages, which can be denoted as:

$$\mathbf{E}^{(l)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}_1^{(l)} + \hat{\mathbf{A}}\mathbf{E}^{(l-1)} \odot \mathbf{E}^{(l-1)}\mathbf{W}_2^{(l)}\right) \quad (3)$$

where $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}$ are linear mapping matrix of the hidden embeddings. $\mathbf{E}^{(l)}$ denotes the embeddings for both users and items at $l-$th layer. And $\hat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, which is different from $\tilde{\mathbf{A}}$ by the self-loop. To reduce the time and space complexity, LightGCN (He et al. 2020) is proposed, which shows that a propagation layer without dot product and redandunt linear transformations even improves performance. Formally, a propagation layer of LightGCN is:

$$\mathbf{E}^{(l)} = \hat{\mathbf{A}}\mathbf{E}^{(l-1)}. \quad (4)$$

There are other works related to GCN based RS. GC-MC (Berg, Kipf, and Welling 2017) predicts the links between users and items by applying the graph convolutional network as the graph encoder. GraphSage (Hamilton, Ying, and Leskovec 2017b) learns the graph embedding by aggregating the information of neighbors, which is extended as a large-scale recommender system, namely PinSage (Ying et al. 2018). SpectralCF (Zheng et al. 2018) designs a spectral convolutional filter to model the CF signals in user-item bipartite graphs. Although existing works are effective in RS, we find that those models fail to notice the oscillation problem when applying the GNN to bipartite graph.

Comparison of GNN and Graph CF

In this paper, we focus on combining Graph CF method (Wang et al. 2019; He et al. 2020) with GNN research area (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017b; Wu et al. 2019). The information propagation process is similar in both areas. Therefore, we can use off-the-shelf GNN research findings to design the graph-based CF model. However, most existing GNN models are built upon attribute graphs, while the CF method mainly focuses on learning the latent embeddings purely from bipartite graph structure. This leads to the difference in *trainable parameters* and *final embeddings* of GNN and Graph CF model. The trainable parameters in most GNNs are the convolutional weights. For example, the GCN model only has $\mathbf{W} \in \mathbb{R}^{d\times d}$ being the trainable weight. The initial embeddings $\mathbf{H}^{(0)}$ is the node features. However, existing CF models require trainable initial embeddings of all nodes, i.e., $\mathbf{E}^{(0)} \in \mathbb{R}^{N\times d}$ is the trainable parameters (Zheng et al. 2018; He et al. 2020). Since $d \ll N$, therefore, the trainable parameters for CF-based models are in general much more than which of GNN models. The final embeddings of GNN models are output from the last layer. However, for the CF-based model, since the initial embeddings $\mathbf{E}^{(0)}$ are trainable and contain important structure information, existing methods all consider combining embeddings output from all layers. For example, NGCF concatenates the embeddings from 0 layer to L layers. LightGCN averages the embeddings of L layers. In our paper, we use the same idea as in LightGCN that averages the embeddings from all layers. Hence, the final embeddings in graph CF models already employ the graph residual structure (Zhang and Meng 2019) like in JKNet (Xu et al. 2018).

Information Propagation on Bipartite Graph

In this section, we introduce the information propagation over bipartite graph and prove the existence of oscillation problem. Intuitively, the oscillation is caused by the information of user nodes only propagates to item nodes, and vice versa. Before studying the oscillation problem on bipartite graph, we present how the information propagates on an irreducible and aperiodic graph.

Definition 1. *(Regular Graph): A regular graph is irreducible and aperiodic. Given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{G}$ is irreducible iff for any two nodes v_i, v_j, they are accessible to each other. Meanwhile, $\mathcal{G}$ is aperiodic iff it is non-bipartite.*

Lemma 1. *Given an unweighted regular graph $\mathcal{G}$, if its corresponding matrix is asymmetric, starting from any initial distribution vector $\mathbf{x}_0 \in \mathbb{R}^n$ with $\mathbf{x}_0 \geq 0$ and $\|\mathbf{x}_0\|_1 = 1$, the random walk propagation over the graph has a unique stationary distribution vector $\boldsymbol{\pi}$, i.e. $\lim_{t\to\infty} \tilde{\mathbf{A}}^t\mathbf{x}_0 = \boldsymbol{\pi}$ where $\boldsymbol{\pi}(i) = \frac{d(v_i)}{2|\mathcal{E}|}$. $d(v_i)$ represents the degree of node v_i. $|\mathcal{E}|$ denotes the total number of edges.*

Definition 2. *(Bipartite Graph): A bipartite user-item graph is defined as $\mathcal{B} = (\mathcal{U}, \mathcal{I}, \mathcal{E})$, where $\mathcal{U}$ and $\mathcal{I}$ are two joint sets of nodes, i.e., $\mathcal{U} \cap \mathcal{I} = \phi$, denoting users and items, respectively. Every edge $e \in \mathcal{E}$ has the form $e = (u, i)$, where $u \in \mathcal{U}$ and $i \in \mathcal{I}$. The corresponding adjacent matrix $\mathbf{A} = \{0,1\}^{(|\mathcal{U}|+|\mathcal{I}|)\times(|\mathcal{U}|+|\mathcal{I}|)}$ of the bipartite graph is defined as:*

$$\mathbf{A} = \begin{bmatrix} 0 & \mathbf{R} \\ \mathbf{R}^\top & 0 \end{bmatrix}, \quad \textit{where} \quad \mathbf{R}_{u,i} = \begin{cases} 1 & \textit{if } (u,i) \in \mathcal{E} \\ 0 & \textit{otherwise}. \end{cases} \quad (5)$$

If taking account of only user (item) nodes, and creating an edge between users (items) if they are connected by a common item (user), we construct a user (item) side graph of the original bipartite graph. The associated side graph of a bipartite graph is defined as:

Definition 3. *(Side Graph): Given a bipartite graph $\mathcal{B} = (\mathcal{U}, \mathcal{I}, \mathcal{E})$, a user side graph of it is defined as $\mathcal{G}_u = (\mathcal{U}, \mathcal{E}_u)$, where an edge $e \in \mathcal{E}_u$ has the form (u_1, u_2) and $u_1, u_2 \in \mathcal{U}$. The edge (u_1, u_2) is constructed from original bipartite graph $\mathcal{B}$, where $\exists i \in \mathcal{I}$ s.t. $(u_1, i) \in \mathcal{E} \wedge (u_2, i) \in \mathcal{E}$. Similarly, we can define an item side graph of $\mathcal{B}$ as $\mathcal{G}_i = (\mathcal{I}, \mathcal{E}_i)$.*

Lemma 1 illustrates that the propagation on a regular graph has a stationary distribution. However, the propagation over a bipartite graph never converges to a stationary distribution. Consider $\mathbf{x}_0$ to be a distribution only on one side of the bi-partition, i.e., only on $\mathcal{U}$ or only on $\mathcal{I}$. Then, the distribution of $\mathbf{x}_1$ at the first step random walk moves entirely to the other side and thus, by induction, $\mathbf{x}_t$ is different for ts with different parity. This phenomenon is *oscillation*.

Definition 4. *(Graph Oscillation): Given an input graph $\mathcal{G}$, starting from any initial distribution, if the random walk over $\mathcal{G}$ has two stationary distributions respectively on even steps and odd steps, i.e., $\lim_{t\to\infty} \tilde{\mathbf{A}}^{2t}\mathbf{x}_0 = \boldsymbol{\pi}_1^*$, $\lim_{t\to\infty} \tilde{\mathbf{A}}^{2t+1}\mathbf{x}_0 = \boldsymbol{\pi}_2^*$, and $\boldsymbol{\pi}_1^* \neq \boldsymbol{\pi}_2^*$, there is an oscillation problem associated with $\mathcal{G}$.*

Theorem 1. *Given a bipartite graph $\mathcal{B} = (\mathcal{U}, \mathcal{I}, \mathcal{E})$, if its associated user and item side graph are both regular graphs, there is an oscillation problem associated with $\mathcal{B}$.*

We provide the proof and the boundary of oscillation in the Appendix. Theorem 1 indicates that oscillation problem exists on bipartite graph. This suggests that stacking many layers results in oscillation problem. Later, we will show oscillation occurs as few as 4 layers in experiments.

Proposed Model

In this section, we present the structure of our proposed deep **D**eoscillated **G**raph **C**ollaborative **F**iltering (DGCF) model. DGCF is a multi-layer GNN model which propagates the CF signals over the bipartite graph. Instead of propagating information to direct neighbor, it has Cross-Hop Propagation (CHP) layers that resolve the oscillation problem. Moreover, its Locality-Adaptive (LA) layers learn the propagating factor for each node. Stacking multiple these layers adaptively controls the information to propagate at different layers. The framework of aggregating the user embedding is present on the right-hand side in Figure 2.

Cross-Hop Propagation Layer

CHP layer propagates the information for both direct and cross-hop neighbor nodes. The information is constructed from both direct neighbor nodes and cross-hop neighbor nodes. Concretely, the node information for user u and item i at $l-$th layer are $\mathbf{e}_u^{(l)} \in \mathbb{R}^d$ and $\mathbf{e}_i^{(l)} \in \mathbb{R}^d$, respectively, where d is the embedding dimension. The additional cross-hop interactions are exactly from user and item side graphs of the bipartite graph. As illustrated on the left-hand side of Figure 2, the red dash lines represent additional cross-hop propagation of DGCF, compared with only direct propagation of NGCF model. CHP layer outputs the embedding of each node by aggregating the information of neighboirs, e.g. the embedding of a user u at $l-$th layer:

$$\mathbf{e}_u^{(l)} = \sum_{j \in \tilde{\mathcal{N}}_u} \alpha_j^{(l)} p_j \mathbf{e}_j^{(l-1)}, \tag{6}$$

where $\tilde{\mathcal{N}}_u$ contains both direct and cross-hop neighbors of user u, as illustrated in the left part in Figure 2. For example, regarding the bipartite graph in Figure 1, both i_4 and u_3 are included in $\tilde{\mathcal{N}}_u$. p_j is the normalizing factor and $\alpha_j^{(l)}$ denotes the locality weight of the $l-$th LA layer, both of which will be introduced later. Analogously, we calculate the item embeddings at different layers by using CHP layers.

Locality-Adaptive Layer

DGCF has an LA layer before a CHP layer, which adaptively controls the propagation process for each node. It assigns a locality weight for each node at $l-$th layer, thus denoted as $\alpha_j^{(l)}$ as in Eq. 6. Since the value of it should be from 0 to 1, we use sigmoid activation to train the weights, i.e.:

$$\boldsymbol{\alpha}^{(l)} = \sigma(\mathbf{w}_{LA}^{(l)}), \quad \text{where} \quad \mathbf{w}_{LA}^{(l)} \in \mathbb{R}^{|\mathcal{U}|+|\mathcal{I}|}. \tag{7}$$

$\mathbf{w}_{LA}^{(l)}$ is the trainable parameter vector for the $l-$th LA layer. $|\mathcal{U}|$ and $|\mathcal{I}|$ denote the number of users and items, respectively. Intuitively, LA layer assigns an influence factor to all nodes before propagation, which is illustrated in Figure 2. Thus, during the aggregation process, it learns the importance of nodes. Since nodes exist in different local structures with varying density, it turns out that the influence factor α controlling the propagation process should be adaptive to the locality. Moreover, as we learn different locality weights from layer to layer, the LA layers also adjust the propagation process in a layer-wise adaptive manner.

Matrix Form

In this section, we present the matrix form propagation of Eq. (6) for all nodes. We use $\mathbf{E}^{(l)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|)\times d}$ to denote embeddings for all user and item nodes at $l-$th layer. Before proceeding to the CHP layer, LA layer applies an importance to each node. Hence, the weighted embedding matrix is:

$$\tilde{\mathbf{E}}^{(l)} = \boldsymbol{\Omega}^{(l)}\mathbf{E}^{(l)}, \tag{8}$$

where $\boldsymbol{\Omega}^{(l)}$ is diagonal and $\boldsymbol{\Omega}^{(l)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|)\times(|\mathcal{U}|+|\mathcal{I}|)}$. $\boldsymbol{\Omega}^{(l)}$ contains the importance factor $\boldsymbol{\alpha}^{(l)}$ in its elements, i.e., $\boldsymbol{\Omega}^{(l)}(j,j) = \boldsymbol{\alpha}_j^{(l)}$. The direct and cross-hop neighbors can be inferred by using the adjacent matrix $\mathbf{A}$ and the cross-hop matrix $\mathbf{C} = \mathbf{A}^2$, respectively. We adopt the Laplacian normalization (Kipf and Welling 2017) of both matrices regarding the p_i in Eq. (6). The Laplacian matrix $\mathcal{L}$ and the cross-hop Laplacian matrix $\mathcal{L}_c$ are defined as:

$$\mathcal{L} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \quad \text{and} \quad \mathcal{L}_c = \mathbf{D}_c^{-\frac{1}{2}}\mathbf{C}\mathbf{D}_c^{-\frac{1}{2}}, \tag{9}$$

respectively. $\mathbf{D}$ and $\mathbf{D}_c$ are both the diagonal degree matrices, by taking the row-sum of $\mathbf{A}$ and $\mathbf{C}$, respectively. Therefore, the final matrix form of the propagation is:

$$\mathbf{E}^{(l)} = (\mathcal{L} + \mathcal{L}_c)\,\tilde{\mathbf{E}}^{(l-1)}. \tag{10}$$

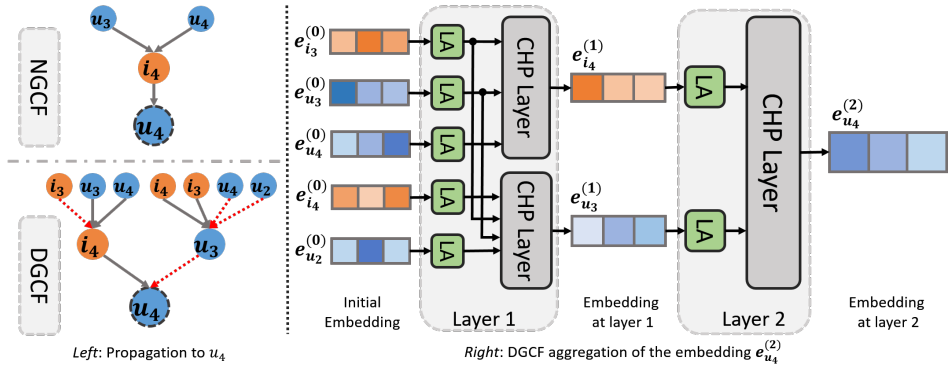


Figure 2: *Left:* Comparison of NGCF and DGCF on the information propagation to u_4. Red dash lines are the cross-hop propagation. The original bipartite graph is in Figure 1. *Right:* The DGCF framework to learn embedding of user u_4 at layer 2. Green 'LA' blocks and gray 'CHP layer' blocks are the locality-adaptive layer and cross-hop propagation layer, respectively.

After $L-$layer propagation, we averages the embeddings (He et al. 2020) at each layer to construct the final embedding for prediction, i.e.,

$$\mathbf{E}^* = \frac{1}{L+1}\sum_{l=0}^{L}\mathbf{E}^{(l)}. \tag{11}$$

Optimization

The initial embedding is generated from Xavier initializer (Glorot and Bengio 2010). The final prediction between users and items is estimated by using the inner product, i.e., $\hat{y}(u,i) = \mathbf{e}_u^{*\top}\mathbf{e}_i^*$. We use BPR (Rendle et al. 2009) loss to optimize the trainable weights:

$$\mathcal{L} = -\sum_{(u,i,j)\in\mathcal{S}} \log\sigma\big(\hat{y}(u,i) - \hat{y}(u,j)\big) + \lambda\|\Theta\|_2^2, \tag{12}$$

where the first term denotes the BPR interaction loss and the second term is the regularization for parameters (λ is a hyper-parameter). $\mathcal{S}$ is generated from the rule that we sample, for each user u, a positive item i and a negative item j. We optimize the model by using the mini-batch Adam (Kingma and Ba 2014) optimizer in Tensorflow.

Model Analysis

The intuition of using **cross-hop** information to enhance the training process of GNN is also discussed in previous works (Xu et al. 2018; Yang et al. 2018; Abu-El-Haija et al. 2019; Chen et al. 2020). In this section, we compare DGCF model with existing models to clarify the difference.

Taking residual model JKNet (Xu et al. 2018) for example, it adds additional skip connections from intermediate layers to the last layer of the GNN model. To simplify the comparison, we eliminate the convolutional weights and only keep the embedding propagation of JKNet [2] as:

$$\mathbf{E}^*_{JK} = \text{CONCAT}\{\mathbf{E}^{(0)}_{JK}, \mathbf{E}^{(1)}_{JK}, \ldots, \mathbf{E}^{(L)}_{JK}\}, \tag{13}$$

[2]Original JKNet also ignores the initial embeddings, which is rather important for RS.

where $\mathbf{E}^{(l)}_{JK}$ is the intermediate embeddings of JKNet and $\mathbf{E}^*_{JK}$ is the final output embeddings of JKNet. Since $\mathbf{E}^{(l)}_{JK} = \mathcal{L}\mathbf{E}^{(l-1)}_{JK}$, we can rewrite Eq. (13) as:

$$\mathbf{E}^*_{JK} = \text{CONCAT}\{\mathbf{E}^{(0)}_{JK}, \mathcal{L}\mathbf{E}^{(0)}_{JK}, \ldots, \mathcal{L}^L\mathbf{E}^{(0)}_{JK}\}. \tag{14}$$

Compared with DGCF in Eq. (11), which is rewritten as:

$$\mathbf{E}^* = \frac{1}{L+1}\sum_{l=0}^{L}(\mathcal{L}+\mathcal{L}_c)^l\,\tilde{\mathbf{E}}^{(0)}. \tag{15}$$

Other than the difference between *concat* and *average* operation, JKNet only considers $\mathcal{L}$ operators. It still suffers from the oscillation problem because $\mathcal{L}$ only propagates the information from one side graph to the other.

Regarding the **complexity** of DGCF, as we mentioned before, all the initial embeddings are trainable in RS, which contains dN parameters if given total of N user and item nodes. LA layer only introduces LN parameters to train. Since $L \ll d$, e.g., $L = 4$ and $d = 64$ in our experiments, the model complexity is comparable to existing models. During implementation, we use the broadcasting mechanism in Tensorflow and thus simply use an LA vector to store the value rather than a matrix.

Experiment

In this section, we conduct extensive experiments to show the effectiveness of our proposed model on four benchmark datasets that include ML100K (Harper and Konstan 2015), ML1M (Harper and Konstan 2015), Amazon Movies and TV (He and McAuley 2016) and Gowalla (Liang et al. 2016). All the dataset are split as training, validation, and testing data. In the experiments, we use the validation set to select the model for testing. More details are in supplementary materials.

Experimental Setting

To evaluate the model performance, several state-of-the-art methods are used for performance comparison, including BPR-MF (Rendle et al. 2009), GCN (Kipf and Welling

Table 1: Overall Performance Comparison.

Dataset	ML1M		Amazon		Gowalla		ML100K	
Metric@20	Recall	NDCG	Recall	NDCG	Recall	NDCG	Recall	NDCG
BPR-MF	0.2653	0.2149	0.0762	0.0588	0.1043	0.0541	0.2894	0.1907
GCN	0.2628	0.2086	0.0701	0.0569	0.0825	0.0430	0.3025	0.1919
GCN+JK	0.2723	0.2184	0.0611	0.0490	0.0800	0.0420	0.3086	0.1917
GC-MC	0.2611	0.2069	0.0578	0.0475	0.0729	0.0370	0.2966	0.1883
NGCF	0.2693	0.2164	0.1117	0.0886	0.1193	0.0606	0.3146	0.1978
LightGCN	0.2888	0.2334	0.1130	0.0893	0.1307	0.0676	0.3399	0.2137
DGCF	**0.3075**	**0.2501**	**0.1351**	**0.1083**	**0.1375**	**0.0708**	**0.3536**	**0.2290**
DGCF-chp	0.2975	0.2420	0.1241	0.0991	0.1327	0.0683	0.3368	0.2159

2017), GCN with JKNet (GCN+JK) (Xu et al. 2018), GC-MC (Berg, Kipf, and Welling 2017), NGCF (Wang et al. 2019) and LightGCN (He et al. 2020). The evaluation metrics are Recall@K and NDCG@K, where K=20 by default.

All models are validated on the performance of Recall@20. The embedding size d is searched from $\{32, 64, 128\}$ for all models. The learning rate for all models is searched from $\{10^{-5}, 10^{-4}, \ldots, 10^{-1}\}$. The regularization factor λ is searched from $\{10^{-5}, 10^{-4}, \ldots, 10^{-1}\}$. We treat each interaction as a positive and pair it with a negative item that the user have no interactions. The layer number L (except for BPR-MF) is search from 1 to 4. The best hyper-parameters are searched based on the validation results. Early stop, i.e., the performance value on validation data not growing for 10 times, is applied. More implementing details are presented in Appendix.

Overall Evaluation

In this section, we present the overall comparing performance. The overall performance are shown in Table 1. The best performance values on different datasets are in bold and the second best values are underlined. Clearly, our DGCF model outperforms others in all cases, since it has CHP layer to remedy oscillation problem and LA layers in it adaptively learn influence factors for nodes.

For those baselines, GC-MC performs the worst compared with other models. The poor performance of GC-MC can be the result of the final MLP layer, which harms the effectiveness of structural regularization and thus overfit the training data (Liu et al. 2019). GCN and GCN+JK have similar performance while GCN gets better results on large datasets (e.g, Amazon and Gowalla), which indicates directly appying graph residual structure cannot benefit the RS. Compared with other GCN models, LightGCN performs the best. The reason is that LightGCN removes redundant parameters (e.g., linear transformation) and nonlinear activation. According to other research work (Cui et al. 2020), those redundant parameters lead to entanglement problem. Moreover, by averaging the outputs of different layers, the expression ability of LightGCN are enhanced by a large margin. However, LightGCN still suffers from the oscillation problem and is not able to adapt to the locality of nodes, which are solved by our DGCF model.

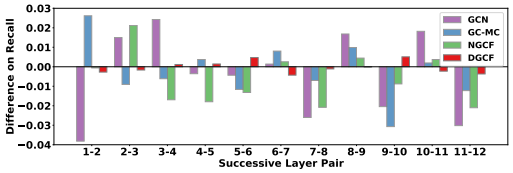


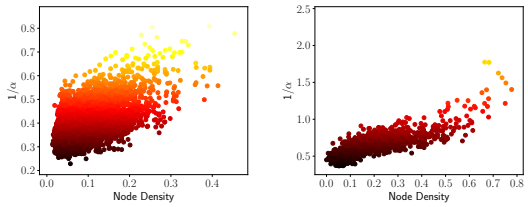
Figure 3: Difference of Recall on successive layer pairs.

Discussion on Oscillation Problem

To understand the oscillation problem, we conduct experiments on ML100K. All models run 10 times for $1 - 12$ layers. We only use embeddings output from the final layer to predict. The differences of the average performance on Recall between successive layers are reported in Figure 3. We observe that the performance goes up and down cyclically (i.e., oscillation), which matches the definition. Though oscillation theoretically appears with infinite layers, it occurs as few as 4 layers in practice as the performance shows in Figure 3. We notice that the oscillation amplitude of DGCF is the smallest one compared with other methods, which shows the effect of deoscillation of DGCF. Additionally, we conduct experiments on ML1M by using only the embeddings at the last layer, which is illustrated in Table 2. LightGCN has good performance in layer 1 and layer 2. But the performance drops quickly in layer 3 and layer 4. It is even worse than GCN and GCMC on layer 4. LightGCN only aggreagtes neighbor embeddings repeatly for deep layers, which is vulnerable to both oversmoothness and oscillations. GCN performs better in layer 4 than in layer 1 and layer 2, which suggests it requires multiple layers. But it is observed that layer 3 is worse than layer 2 and layer 4, which is the result of oscillation. DGCF consistently yields good performance on all depths, which indicates its robustness.

Discussion on Varying Locality

In this section, we discuss the correlation between learned influence factor $\boldsymbol{\alpha}$ and the density of nodes. Intuitively, the LA layer balances the information propagation for different nodes, i.e., nodes with lower density should be assigned higher influence factor α. We train DGCF on ML1M dataset. The distributions of nodes' density and the reciprocal value of α are illustrated in Figure 4. To better view the corre-



(a) Users. $\rho = 0.578$ (b) Items. $\rho = 0.835$

Figure 4: Correlation coefficient of LA layer and the density of nodes on ML1M. Left: users. Right: items.

Recall@20	Layer 1	Layer 2	Layer 3	Layer 4
GCN	0.2589	0.2577	0.2332	0.2628
GCMC	0.2574	0.2602	0.2328	0.2646
NGCF	0.2107	0.2111	0.1767	0.1759
LightGCN	0.2930	0.2844	0.2543	0.1968
DGCF	**0.3026**	**0.3036**	0.2950	0.2939
DGCF-chp	0.3025	0.2980	**0.2975**	**0.2954**

Table 2: Performance Comparison on Single Layer.

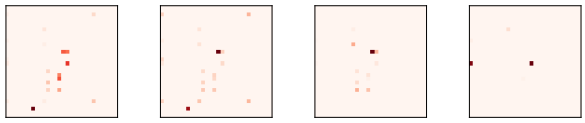
lation, node's density is calculated as normalization of the logarithm of its degree. The correlation coefficients ρ w.r.t. users and items are 0.578 and 0.835, respectively. It indicates $1/\alpha$ is positively correlated with the density of nodes, which proves that LA layer balances the influence of nodes regarding the varying locality problem. We also discuss in appendix about the performance of nodes w.r.t. their density.

Discussion on Propagation Pattern

In this section, we conduct experiments to show how DGCF learns layer-wise aggregation patterns. Existing GNN RS models only has fix propagation pattern. For example, NGCF only propagates information to direct neighbors at different layers. In constrast, DGCF has LA layer that assigns each node an influence factor which controls the propagation pattern. Since DGCF has L different LA layers, it has L different propagation patterns. We implement a $4-$layer DGCF model and train it on ML1M data. We present the propagation patterns on each layer in Figure 5. Due to the sparsity of adjacency matrix, we zoom in a local patch (30×30) to view the variations. Each pixel denotes the normalized value of the product of influence factor and Laplacian matrix, i.e., $\Omega(\mathcal{L}+\mathcal{L}_c)$. We observe that the propagation pattern varies on different layers. On the first layer, it is a matrix containing almost every original edges, which suggests information is propagated out widely. Then, on layer 2, 3, and 4, the value on each edge varies and shrinks to a few important edges, which implies the propagation pattern adaptively concentrates on important substructures.

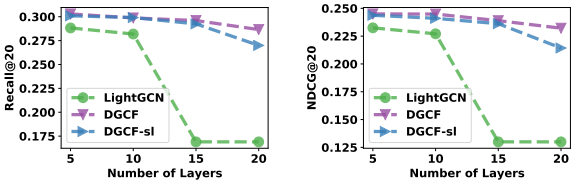
Variants Analysis

In this section, we conduct the ablation study and discuss other variants of DGCF. There is one major variant DGCF-chp which only considers CHP layers while without LA layers. We report the overall performance of DGCF-chp Table 1. It is observed DGCF-chp yields better performance over other baselines under most cases. It indicates the effectiveness of adding the CHP layer. But it is still worse than DGCF as it has fixed propagation patterns if without LA layers. We also use the last layer of DGCF-chp to make prediction. The results are presented in Table 2. We can observe that DGCF-chp has the highest values on both layer 3 and layer 4, which suggests the effectiveness of the CHP layer. Additionally, in order to study the effect of deep depth in RS, we conduct experiments on ML1M dataset with layers ranging from 5 to 20 layers. We create a new variant that shares LA layers, named as DGCF-sl. The performance comparison is presented in Figure 6. We observed that LightGCN drops dramatically with deep layers, which is the result of oversmoothing. DGCF and DGCF-sl both perform better than LightGCN, though also drop with deep depth. Compared with DGCF-sl, the better performance of DGCF shows the effectiveness of its layer-wise LA layers.



(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 5: Propagation pattern varies on different layers. Darker pixels are greater values.



(a) Recall@20 (b) NDCG@20

Figure 6: Performance with deep depth.

Conclusion

In this paper, we study the oscillation problem, varying locality problem, and fixed aggregation pattern when applying multi-layer GNN on bipartite graph. We formally define the graph oscillation problem and prove its existence on bipartite graph. To tackle the problems, we propose a new model DGCF which stacks multiple LA layers and CHP layers. The overall experiments on four real-world datasets proves the effectiveness of DGCF. Moreover, we conduct detailed analyzing experiments. The experiment on the depth of model implies the oscillation on existing models occurs while DGCF has smallest amplitude. Also, the experiment regarding the varying locality indicates that DGCF adaptively learn the influence factor correlated with the density. Additionally, the experiment on the aggregation pattern suggests that DGCF automatically adjusts the aggregation pattern in a layer-wise manner.

Ethics Statement

In general, this paper studies how to design a new recommender system model. It learns user and item embeddings, which helps to make a further recommendation for users. This model can be applied to many real-world applications, such as the recommender system in E-commerce company. The datasets that we use in this paper are all public available data. The original data may contains the user profile information, which is a little bit sensitive. However, the proposed model and conducted experiments are all solely based on the interaction records, which is of no sensitivity.

On the other hand, since the proposed model trains the user embedding, it can also be applied as a personalising model. It finds users' interests towards items. If a company deploys this model based on their customer' interaction records, it can help the users to find their potential interests. However, the recommendation result may cause privacy concerns of users.

References

Abu-El-Haija, S.; Perozzi, B.; Kapoor, A.; Alipourfard, N.; Lerman, K.; Harutyunyan, H.; Steeg, G. V.; and Galstyan, A. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. *arXiv preprint arXiv:1905.00067* .

Berg, R. v. d.; Kipf, T. N.; and Welling, M. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* .

Chen, J.; Ma, T.; and Xiao, C. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Chen, L.; Wu, L.; Hong, R.; Zhang, K.; and Wang, M. 2020. Revisiting Graph Based Collaborative Filtering: A Linear Residual Graph Convolutional Network Approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 27–34.

Cui, G.; Zhou, J.; Yang, C.; and Liu, Z. 2020. Adaptive Graph Encoder for Attributed Graph Embedding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 976–985.

Derr, T.; Ma, Y.; and Tang, J. 2018. Signed graph convolutional networks. In *2018 IEEE International Conference on Data Mining (ICDM)*, 929–934. IEEE.

Gao, H.; Wang, Z.; and Ji, S. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1416–1424.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *IJCAI*, 249–256.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017a. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40(3): 52–74.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017b. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 1024–1034.

Harper, F. M.; and Konstan, J. A. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5(4): 1–19.

He, R.; and McAuley, J. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, 507–517.

He, X.; Deng, K.; Wang, X.; Li, Y.; Zhang, Y.; and Wang, M. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *arXiv preprint arXiv:2002.02126* .

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, 173–182.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Liang, D.; Charlin, L.; McInerney, J.; and Blei, D. M. 2016. Modeling user exposure in recommendation. In *Proceedings of the 25th international conference on World Wide Web*, 951–961.

Liu, Z.; Dou, Y.; Yu, P. S.; Deng, Y.; and Peng, H. 2020. Alleviating the Inconsistency Problem of Applying Graph Neural Network to Fraud Detection. In *Proceedings of the 43nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Liu, Z.; Zheng, L.; Zhang, J.; Han, J.; and Yu, P. S. 2019. JSCN: Joint Spectral Convolutional Network for Cross Domain Recommendation. *ArXiv* abs/1910.08219.

Noble, B.; Daniel, J. W.; et al. 1988. *Applied linear algebra*, volume 3. Prentice-Hall New Jersey.

Rendle, S. 2010. Factorization machines. In *ICDM*, 995–1000.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*, 452–461.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T. 2019. Neural Graph Collaborative Filtering. In *SIGIR*, 165–174.

Wu, F.; Jr., A. H. S.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. Q. 2019. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 6861–6871.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.; and Jegelka, S. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML*

2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, 5449–5458.

Yang, J.-H.; Chen, C.-M.; Wang, C.-J.; and Tsai, M.-F. 2018. HOP-rec: high-order proximity for implicit recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 140–144.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In Guo, Y.; and Farooq, F., eds., *SIGKDD*, 974–983.

Zhang, J.; and Meng, L. 2019. GResNet: Graph Residual Network for Reviving Deep GNNs from Suspended Animation. *ArXiv* abs/1909.05729.

Zhang, M.; and Chen, Y. 2020. Inductive Matrix Completion Based on Graph Neural Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Zheng, L.; Lu, C.-T.; Jiang, F.; Zhang, J.; and Yu, P. S. 2018. Spectral collaborative filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 311–319. ACM.

Zheng, L.; Noroozi, V.; and Yu, P. S. 2017. Joint Deep Modeling of Users and Items Using Reviews for Recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*, 425–434.

Zou, D.; Hu, Z.; Wang, Y.; Jiang, S.; Sun, Y.; and Gu, Q. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In *Advances in Neural Information Processing Systems*, 11249–11259.

Appendix

Proof of Lemma 1

Proof. Assume $\boldsymbol{\pi}$ is a stationary distribution vector such that $\tilde{\mathbf{A}}\boldsymbol{\pi} = \boldsymbol{\pi}$. According to the definition of $\tilde{\mathbf{A}}$ and the matrix multiplication rule, $\boldsymbol{\pi}(i)$ can be represented as

$$\sum_j \tilde{\mathbf{A}}(i,j)\boldsymbol{\pi}(j) = \sum_j \frac{w_{i,j}}{d(j)}\boldsymbol{\pi} = \boldsymbol{\pi}(i), \tag{16}$$

where $w_{i,j}$ denotes the initial weight between node v_i and v_j and $d(i)$ denotes the rough degree that sums the weight of edges connected to the nodes in the graph, i.e., $d(i) = \sum_j w_{i,j}$. Therefore, we can obtain $\boldsymbol{\pi_j} \propto d_j$ since $d(i)$ respects to $w_{i,j}$. Specifically, we can set

$$\boldsymbol{\pi}(j) = \frac{d(j)}{\sum_k d(k)} = \frac{d(j)}{2|\mathcal{E}|} \tag{17}$$

Thus,

$$\sum_j \tilde{\mathbf{A}}(i,j)\boldsymbol{\pi}(j) = \sum_j \frac{w_{i,j}}{d(j)}\frac{d(j)}{2|\mathcal{E}|} = \sum_j \frac{w_{i,j}}{2|\mathcal{E}|} = \frac{d(i)}{2|\mathcal{E}|} \tag{18}$$

holds for any regular graph (note that $d(i)$ equals to $d(v_i)$). □

Proof of Theorem 1

Proof. Given a bipartite graph $\mathcal{B} = \{\mathcal{U}, \mathcal{I}, \mathcal{E}\}$, where $\mathcal{U}$ and $\mathcal{I}$ represent two groups of nodes and $\mathcal{E}$ represents the corresponding edges between the nodes. The associated normalized adjacent matrix is: Given a bipartite graph $\mathcal{B} = \{\mathcal{U}, \mathcal{I}, \mathcal{E}\}$, where $\mathcal{U}$ and $\mathcal{I}$ represent two groups of nodes and $\mathcal{E}$ represents the corresponding edges between the nodes. The associated normalized adjacent matrix is:

$$\hat{\mathbf{A}} = \begin{bmatrix} 0 & \mathbf{R} \\ \mathbf{R}^\top & 0 \end{bmatrix}, \tag{19}$$

where $\mathbf{R} \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$ is the normalized link matrix from node set $\mathcal{U}$ to node set $\mathcal{I}$. For simple analysis, we assume the initial probability distribution on the nodes is $\mathbf{x}^{(0)} = [\underbrace{x_{u_1}^{(0)}, x_{u_2}^{(0)}, ..., x_{u_{|\mathcal{U}|}}^{(0)}}_{\mathbf{x}_u^{(0)\top}}, \underbrace{x_{i_1}^{(0)}, \ldots, x_{i_{|\mathcal{I}|}}^{(0)}}_{\mathbf{x}_i^{(0)\top}}]^\top$ for all the associated nodes, which is the normalized node feature. We apply one step of random walk on the graph $\mathcal{B}$, which is:

$$\mathbf{x}^{(1)} = \hat{\mathbf{A}}\mathbf{x}^{(0)} = \begin{bmatrix} \mathbf{R}\mathbf{x}_i^{(0)} \\ \mathbf{R}^\top\mathbf{x}_u^{(0)} \end{bmatrix}. \tag{20}$$

And two steps random walk is:

$$\mathbf{x}^{(2)} = \hat{\mathbf{A}}\mathbf{x}^{(1)} = \begin{bmatrix} \mathbf{R}\mathbf{R}^\top\mathbf{x}_u^{(0)} \\ \mathbf{R}^\top\mathbf{R}\mathbf{x}_i^{(0)} \end{bmatrix}. \tag{21}$$

Let $\mathbf{V} = \mathbf{R}\mathbf{R}^\top$ and $\mathbf{T} = \mathbf{R}^\top\mathbf{R}$, and if we continue the random walks for $2k$ steps, we have

$$\begin{aligned} \mathbf{x}^{(2k)} &= \hat{\mathbf{A}}\hat{\mathbf{A}}\mathbf{x}^{2(k-1)} \\ &= \begin{bmatrix} 0 & \mathbf{R} \\ \mathbf{R}^\top & 0 \end{bmatrix}\begin{bmatrix} 0 & \mathbf{R} \\ \mathbf{R}^\top & 0 \end{bmatrix}\mathbf{x}^{2(k-1)} \\ &= \begin{bmatrix} \mathbf{V} & 0 \\ 0 & \mathbf{T} \end{bmatrix}\mathbf{x}^{2(k-1)} \\ &= \begin{bmatrix} \mathbf{V}^k\mathbf{x}_u^{(0)} \\ \mathbf{T}^k\mathbf{x}_i^{(0)} \end{bmatrix}. \end{aligned} \tag{22}$$

We can find that $\mathbf{V} \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{U}|}$ is the corresponding normalized adjacent matrix of user side graph. $\mathbf{T} \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{I}|}$ is the corresponding item side graph. Since $\mathbf{V}$ and $\mathbf{T}$ are both associated with regular graphs, when $k \to \infty$, the final probability is the stationary distribution $\boldsymbol{\pi}$. Hence, we have:

$$\lim_{k\to\infty} \mathbf{x}^{(2k)} = \boldsymbol{\pi} = \begin{bmatrix} \boldsymbol{\pi}_u \\ \boldsymbol{\pi}_i \end{bmatrix}, \tag{23}$$

where $\boldsymbol{\pi}_u$ and $\boldsymbol{\pi}_i$ are the stationary probability for user side graph and item side graph, respectively. Similarly, for the random walks with $2k+1$ steps, we have the

$$\begin{aligned} \lim_{k\to\infty} \mathbf{x}^{(2k+1)} &= \lim_{k\to\infty-} \hat{\mathbf{A}}\mathbf{x}^{2k} = \hat{\mathbf{A}}\boldsymbol{\pi} \\ &= \begin{bmatrix} 0 & \mathbf{R} \\ \mathbf{R}^\top & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{\pi}_u \\ \boldsymbol{\pi}_i \end{bmatrix} = \begin{bmatrix} \mathbf{R}\boldsymbol{\pi}_i \\ \mathbf{R}^\top\boldsymbol{\pi}_u \end{bmatrix} = \boldsymbol{\pi}'. \end{aligned} \tag{24}$$

From Eq. (23) and Eq. (24), we know that multi-step propagation on a bipartite graph would reduce the representation of the node from a column normalized feature to two different stationary distributions w.r.t. the parity of the number of steps. It suggests that the oscillation problem is a result of the parity of the number of layers of a multi-layer GNN model. Note the oscillation vanishes if and only if $\mathbf{R}\boldsymbol{\pi}_i = \boldsymbol{\pi}_u$. □

Boundary of Oscillation

In addition to the layer-parity-related stationary distribution, we further analyze whether there exists a bound between successive layers regarding the distribution. The bound is a theoretical proof that the final representation learned from the GCN model oscillates within a shallow range. Hence, oscillation would not affect to much on the tuning process of the GCN model w.r.t. the number of layers. Assuming that we are at even step which denoted as $2k$-th step, from Eq. (22), we have the following difference of the probability between $2k$-th step and $2k+1$-th step as:

$$\left\|\mathbf{x}^{(2k)} - \mathbf{x}^{(2k+1)}\right\|_1 = \left\| \begin{bmatrix} \mathbf{V}^k\mathbf{x}_u^{(0)} \\ \mathbf{T}^k\mathbf{x}_i^{(0)} \end{bmatrix} - \begin{bmatrix} \mathbf{R}\mathbf{T}^k\mathbf{x}_i^{(0)} \\ \mathbf{R}^\top\mathbf{V}^k\mathbf{x}_u^{(0)} \end{bmatrix} \right\|_1, \tag{25}$$

where the $\|\cdot\|_1$ is the 1-norm of the vector. Since we have $\mathbf{R}\mathbf{T}^k = \mathbf{V}^k\mathbf{R}$ and $\mathbf{R}^\top\mathbf{V}^k = \mathbf{T}^k\mathbf{R}^\top$, hence we have

$$\left\|\mathbf{x}^{(2k)} - \mathbf{x}^{(2k+1)}\right\|_1 = \left\| \begin{bmatrix} \mathbf{V}^k\left(\mathbf{R}\mathbf{x}_i^{(0)} - \mathbf{x}_u^{(0)}\right) \\ \mathbf{T}^k\left(\mathbf{R}^\top\mathbf{x}_u^{(0)} - \mathbf{x}_i^{(0)}\right) \end{bmatrix} \right\|_1. \tag{26}$$

Table 3: Datasets statistics

Dataset	#Users	#Items	Density	#Interactions		
				Train	Validation	Test
MovieLens 100k	779	1,169	0.02285	14,585	2,068	4,152
MovieLens 1M	4,627	1,840	0.02490	148,299	21,362	42,347
Amazon Movies&TV	14,432	28,242	0.00155	441,771	63,654	126,085
Gowalla	29,858	40,981	0.00084	718,418	103,862	205,090

Since $\mathbf{R}\,\mathbf{x}_i^{(0)}$ and $\mathbf{x}_u^{(0)}$ is a constant given as the initial value of the graph, we denote $\mathbf{R}\mathbf{x}_i^{(0)} - \mathbf{x}_u^{(0)}$ and $\mathbf{R}^\top \mathbf{x}_u^{(0)} - \mathbf{x}_i^{(0)}$ as ε and ξ, respectively. Then, the Eq. (26) becomes:

$$\begin{aligned}\left\|\mathbf{x}^{(2k)} - \mathbf{x}^{(2k+1)}\right\|_1 &= \left\| \begin{bmatrix} \mathbf{V}^k & 0 \\ 0 & \mathbf{T}^k \end{bmatrix} \begin{bmatrix} \epsilon \\ \xi \end{bmatrix} \right\|_1 \\ &\leq \left\| \begin{bmatrix} \mathbf{V}^k & 0 \\ 0 & \mathbf{T}^k \end{bmatrix} \right\|_1 \cdot \left\| \begin{bmatrix} \epsilon \\ \xi \end{bmatrix} \right\|_1 .\end{aligned} \tag{27}$$

Since the 1-norm of the matrix is the maximum column sum (Noble, Daniel et al. 1988), the final inequality in Eq. (27) is:

$$\begin{aligned}&\left\|\mathbf{x}^{(2k)} - \mathbf{x}^{(2k+1)}\right\|_1 \leq \\ &\max\left\{ \max_{1\leq j\leq|\mathcal{U}|} \sum_{i=1}^{|\mathcal{U}|} |v_{ij}|, \max_{1\leq j\leq|\mathcal{I}|} \sum_{i=1}^{|\mathcal{I}|} |t_{ij}| \right\} \cdot \left\| \begin{bmatrix} \epsilon \\ \xi \end{bmatrix} \right\|_1,\end{aligned} \tag{28}$$

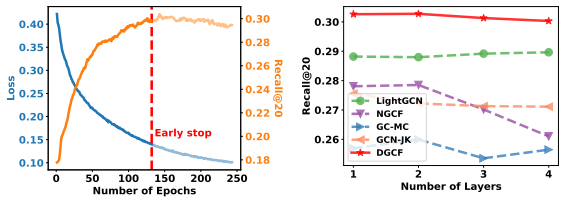
where the v_{ij} and t_{ij} is the corresponding entries of the matrix $\mathbf{V}^k$ and matrix $\mathbf{T}^k$ respectively. Here, we prove that the difference between the the odd steps and the even steps is bounded by the maximum value in the corresponding matrix. And since the maximum value in the matrix is always less than 1 due to probability property, we can safely increase the layer number while the amplitude will not exceed the bound.

Experimental Details

Dataset Description

To evaluate the effectiveness of our proposed model, we conduct experiments on four public benchmark datasets: MovieLens 100k, MovieLens 1M, Amazon Movies and TV and Gowalla. After pre-processing, we summarize the statistics of three datasets in Table 3.

- **MovieLens 100K**: This is a commonly used benchmark dataset (Harper and Konstan 2015). For this dataset, we maintain users with at least 5 interactions.
- **MovieLens 1M**: This movie rating dataset (Harper and Konstan 2015) has been widely used in the recommendation scenarios. To ensure the quality, we only retain ratings equal to 5 and apply 10-core setting adopted in (Wang et al. 2019), retaining the users and items with at least 10 interactions. Thus, we get 4,627 users and 1,840 items as well as 212,008 interactions.
- **Amazon Movies and TV**: Amazon Movies and TV is another widely used dataset (He and McAuley 2016) for product recommendation. Here, we also hold ratings equal to 5 and adopt 10-core setting on users and items, which results in 14,432 users and 28,242 items with 631,510 interactions.
- **Gowalla**: This is a checkin dataset (Liang et al. 2016) obtained from Gowalla, where the users share locations by checkins. After filtered by 10-core setting, the dataset contains 1,027,370 interactions between 29,858 users and 40,981 items.



(a) Loss and validation recall. (b) Recall w.r.t. # of layers.

Figure 7: Training and tuning the number of layers of DGCF on ML1M dataset.

For each dataset, we randomly choose 70% interactions as the training set, 10% as the validation set, and the rest 20% as the testing set. For each interactions in the training set, we treat it as a positive interaction and randomly select one negative interaction by negative sampling strategy.

Hyperparameters

In this section, we introduce the details of the implementation. For the overall evaluation, we first train the model on the training data. During training, we validate all models based on the performance of Recall@20 on validation data. We also use early-stopping to prevent models from overfitting if it reaches highest value without growing for 5 times. The best models are selected based on their performance on validation data. Finally, we test the selected models on testing data. For all models, we use the Xavier initializer for trainable parameters. We also randomly dropout partial edges to prevent oversmoothing.

Hyper-parameters of DGCF are chosen as following: For ML1M dataset, we use 2-layer DGCF. The learning rate is 0.001 and the regularization factor $\lambda = 0.01$. The embedding dimension is 64. For Movies and TV dataset, we use 4-layer DGCF. The learning rate is 0.0001 and the regularization factor $\lambda = 0.01$. The embedding dimension is 128. For Gowalla dataset, we use 3-layer structure DGCF. The learning rate is 0.0001 and $\lambda = 0.001$. The embedding dimension is 128. For ML100K, we use 2-layer DGCF. The learning rate is 0.001 and the regularization factor $\lambda = 0.01$.

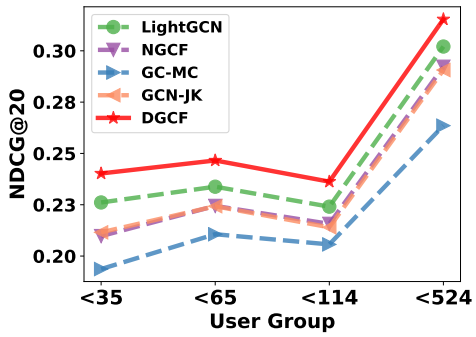


Figure 8: Performance w.r.t. different density groups of users on ML1M dataset.

The embedding dimension is 64. Moreover, due to large size of cross-hop adjacency matrices of Movies and TV dataset and Gowalla dataset, we apply filtering technique to drop cross-hop weights less than ϵ, i.e, dropping $\mathcal{L}_c^2[i,j] < \epsilon$, where $\epsilon = 0.01$ and $\epsilon = 0.004$ for Movies and TV dataset and Gowalla dataset, respectively. The training loss and validation performance on ML1M are presented in Figure 7a. The tuning process of the number of layers of DGCF on ML1M dataset are illustrated in Figure 7b.

For other baseline methods, we set learning rate to 0.001 and $\lambda = 10^{-5}$ for both ML100k and ML1M for all baselines. For Amazon and Gowalla, GCN and GC-MC set learning rate to 10^{-3} and $\lambda = 10^{-5}$, while NGCF and LightGCN use learning rate 10^{-4} and $\lambda = 10^{-5}$. To make a fair comparison, we also use the same embedding size as DGCF. In table 1, we choose the best score obtained from one layer to four layers for each GCN-based baseline. All baselines are implemented based on code [3].

Implementation

Our model is build upon tensorflow-gpu framework. The GPU machine we use is GeForce GTX 1080Ti. The CPU is Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz. The memory size is 64 GB in total. In average, the running times of the 4-layer DGCF are 3.4 s/epoch, 22.0 s/epoch, 62.1 s/epoch, and 100.6 s/epoch on ML100K dataset, ML1M dataset, Movies and TV dataset and Gowalla dataset, respectively .

Comparison on Density Levels

In this section, we discuss the model performance under different users' sparsity levels based on ML1M dataset. We split test users into four groups according to the number of interactions they have, and each group has same number of interactions. The reported NDCG@20 are shown in Figure 8. It is observed that DGCF consistently achieves the best performance, which indicates that DGCF can effectively propagate the information on bipartite graph. Moreover, we can find that the improvement that DGCF made on sparse user groups (i.e., < 35 and < 65) is higher (up to 24%) than that (up to 19%) for the dense groups (i.e., < 114 and < 524), demonstrating the DGCF can alleviate the cold-start issue by incorporating the CHP layers. Additionally, we observe that the performance of all models on the densest group are the best, which suggests that GCN-based models should perform better on a relative denser graph.

[3] https://github.com/xiangwang1223/neural_graph_collaborative_filtering