SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2022, Vol. 65 112101:1–112101:15
https://doi.org/10.1007/s11432-020-3274-6

# LECF: recommendation via learnable edge collaborative filtering

Shitao XIAO[1,2], Yingxia SHAO[1,2*], Yawen LI[1], Hongzhi YIN[3], Yanyan SHEN[4] & Bin CUI[5,6]

[1]*School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing* 100876*, China;*
[2]*Beijing Key Lab of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing* 100876*, China;*
[3]*School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane* Qld 4072*, Australia;*
[4]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai* 200240*, China;*
[5]*Department of Computer Science and Technology, Peking University, Beijing* 100871*, China;*
[6]*National Engineering Laboratory for Big Data Analysis and Applications, Peking University, Beijing* 100871*, China*

Received 12 October 2020/Revised 12 December 2020/Accepted 9 April 2021/Published online 23 December 2021

**Abstract** The core of recommendation models is estimating the probability that a user will like an item based on historical interactions. Existing collaborative filtering (CF) algorithms compute the likelihood by utilizing simple relationships between objects, e.g., user-item, item-item, or user-user. They always rely on a single type of object-object relationship, ignoring other useful relationship information in data. In this paper, we model an interaction between user and item as an edge and propose a novel CF framework, called learnable edge collaborative filtering (LECF). LECF predicts the existence probability of an edge based on the connections among edges and is able to capture the complex relationship in data. Specifically, we first adopt the concept of line graph where each node represents an interaction edge; then calculate a weighted sum of similarity between the query edge and the observed edges (i.e., historical interactions) that are selected from the neighborhood of query edge in the line graph for a recommendation. In addition, we design an efficient propagation algorithm to speed up the training and inference of LECF. Extensive experiments on four public datasets demonstrate LECF can achieve better performance than the state-of-the-art methods.

**Keywords** collaborative filtering, recommender system, implicit feedback, edge-edge similarity, line graph

**Citation** Xiao S T, Shao Y X, Li Y W, et al. LECF: recommendation via learnable edge collaborative filtering. Sci China Inf Sci, 2022, 65(1): 112101, https://doi.org/10.1007/s11432-020-3274-6

## 1 Introduction

Recommender systems can help users find what they are interested in from an unprecedented number of items, which is vital to enhancing user satisfaction and loyalty. In the era of information explosion, they play an increasingly important role in online services such as E-commerce, news feed, and video subscription [1–4], and attract considerable attention of researchers. In this work, we focus on the recommender systems based on implicit data (e.g., bookmarks, click-through, purchases), where collaborative filtering (CF) is widely used, assuming that behaviorally similar users like similar items. We treat both users and items as objects in the recommender systems. Existing CF methods mostly utilize the object-object relationship to estimate scores representing the likelihood. Figures 1(a)–(c) give three examples of object-object relationships. Some methods use the interaction function to directly model the relationship between user object and item object. For example, matrix factorization (MF) [5] maps objects to a low dimensional space and uses the inner product function to capture the user-item interaction

* Corresponding author (email: shaoyx@bupt.edu.cn)

 

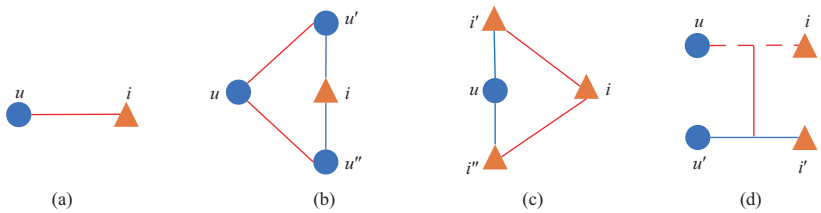


**Figure 1** (Color online) Illustrations of user-item relationship (a), user-user relationship (b), item-item relationship (c), and edge-edge relationship (d). The blue line represents an observed interaction, and the red line is the similarity need to learn. The red dotted line is the query edge.

relationship. NeuMF [6] estimates the score by a nonlinear function based on the user and item embedding. Some other methods utilize the similarity relationship between user objects (user-based methods) or item objects (item-based methods) to predict scores. Sparse linear model (SLIM) [7] calculates the sum of similarity between query item and the items that have been interacted by query user as the score. UserKNN [8] considers whether there are some interactions between the query item and other users who are similar to the query user.

However, the above methods that rely on the object-object relationship cannot make full use of complex relationships in data. Specifically, the models that directly factorize user-item interactions do not consider utilizing the similarity among users (items), are poor in detecting strong associations among closely related users (items) [9]. The methods based on object-object similarity relationship only consider the interactions of query user or item, difficult to get enough information to achieve satisfactory performance [2, 9]. The majority of other useful interaction information is ignored. For example, the interactions between users, which are similar to the query user, and items, which are similar to the query item, should be helpful for predicting the scores between the query user and item [10]. More concretely, when predicting the score between a user B and the movie *The Avengers*, the historical interaction between a user A, who has similar interest with user B, and the movie *Iron Man*, which has the same genre with the movie *The Avengers*, is benefit to the prediction. But the methods based on object-object similarity relationship cannot leverage such historical interactions if there is no interaction between user A and *The Avengers* or the one between user B and *Iron Man*.

In this work, we propose a new recommendation framework learnable edge collaborative filtering (LECF). LECF models the interaction as an edge between user object and item object. For a query pair $(u, i)$, also called query edge, it estimates the probability of the edge to exist between object $u$ and object $i$. According to similar things always share the same characteristics (e.g., label) [11, 12], if the query edge is similar to the observed edges, the query edge should also tend to exist. Therefore, LECF uses the edge-edge similarity to represent the level of observed edges supporting the existence of query edge and outputs a weighted sum of these similarities as the estimated score. Compared with the methods only modeling the user-item interactions, LECF further learns the edge-edge similarities in the historical interactions. As shown in Subsection 6.6, LECF can also capture the object-object similarity in data. Compared with the methods that utilize the user-user or item-item similarity, LECF can take any past interactions into consideration when predicting, not only focus on the local interactions directly related to query user or item. LECF is capable of capturing sufficient and complex relationship information among data. In addition, the existing algorithms that rely on the object-object relationship can be viewed as the reduced versions of LECF, which is detailed analyzed in Subsection 3.4.

In LECF, we model user-item interactions as edges and then learn edge-edge similarities between the query edge and other observed edges. Hence, there are two important issues that need to be addressed. First, we need to define a proper form to represent the edges, and design a function able to learn the edge-edge similarity. Second, we need to consider which edges should be used to predict the score and how to treat them differently. Since there are a huge number of historical interactions in a recommender system, it is not feasible to utilize all the observed edges. Furthermore, we should consider the different importance of the observed edges with regard to a query edge when predicting the score. It is because active users (the users who have many interactions) are generally the source of noise, the interactions of inactive users should be considered more reliable in comparison with the interactions of active users [11].

To address the first issue, we introduce a binary function that takes the user and item embedding as the input to generate representations of edges and use the inner product to learn the similarity between edges. For the second issue, we transform the bipartite graph $G$ built from historical interactions between

users and items to a line graph $L(G)$. Each node of the line graph represents an interaction edge, and we call the node the observed edge in this paper. Then the observed edges within the $l$-hop neighborhood of the query edge in the line graph are selected to predict the score. We further design a biased random walk model over the line graph to compute the importance of each observed edge. Finally, to fast train LECF, we develop an efficient propagation algorithm.

The main contributions of this work are summarized as follows:

• We present a new recommendation framework, called LECF, which generates a recommendation based on the relationship between edges.

• We design an efficient propagation algorithm to speed up the training and inference of LECF.

• We discuss the connection between LECF and existing CF models that rely on the object-object relationship. The existing models can be viewed as a certain simplified type of LECF.

• We conduct extensive experiments on four real-world datasets. The results demonstrate that LECF can achieve better model accuracy than the start-of-the-art models without reducing the training efficiency.

## 2 Preliminaries

### 2.1 Implicit collaborative filtering

In this paper, we focus on the CF with implicit feedback (e.g., purchase, bookmarks, clicks, likes). We use the symbols $u$, $i$ to represent user and item, respectively. The implicit interaction matrix is defined as follows.

**Definition 1** (Implicit interaction matrix)**.** Implicit interaction matrix is a binary matrix $\boldsymbol{Y}$, where $y_{ui}$ is one if the user $u$ has interaction with item $i$, and zero otherwise. Formally,

$$y_{ui} = \begin{cases} 1, & \text{if interaction } (u, i) \text{ exists,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

A value of 0 for entry $y_{ui}$ does not imply that user $u$ does not like $i$. In most cases, the user is unaware of the item, so the value of this entry can be viewed as a missing value. This leads to the task of recommendation with implicit feedback: estimating the scores of the unobserved entries in $\boldsymbol{Y}$. This recommendation task also can be viewed as a prediction problem — with a user and an item as input, predicting the probability $y_e$ of a query edge $e$ between $u$ and $i$.

### 2.2 Bipartite graph

We can model the implicit interaction matrix as a bipartite graph, which is defined as follows.

**Definition 2** (Bipartite graph)**.** A user-item bipartite graph with $|V|$ vertices and $|E|$ edges for recommendation is denoted by $G = \{U, I, E\}$, where $U$ is user node set, $I$ is item node set, $V$ is the union of $U$ and $I$, and $E$ is the set of edges between user nodes and item nodes. Every edge $e(u, i) \in E$ indicates that user $u$ has interacted with item $i$.

Given a node $v \in V$, we use $E(v)$ to represent the set of edges that directly are adjacent to the node $v$, and $|E(v)|$ represents the number of edges in $E(v)$.

### 2.3 Line graph

Next, we give the concept of a line graph corresponding to the bipartite graph that is defined by a recommendation task. The detailed definition of the line graph is shown as follows.

**Definition 3** (Line graph)**.** Given a bipartite graph $G$, its line graph $L(G)$ is defined as follows: each node of $L(G)$ represents an edge of $G$; two nodes of $L(G)$ are adjacent if and only if their corresponding edges share a common node in $G$.

Over a line graph $L(G)$, we use $t(e_0 \to e_1 \to \cdots \to e_k \to \cdots \to e_l)$ to represent a walk of length $l$, where $e_k$ is the $k$th node in the walk $t$. In addition, we use $T^l : e \rightsquigarrow e'$ to represent all walks of length $l$ between nodes $e$ and $e'$ in $L(G)$.

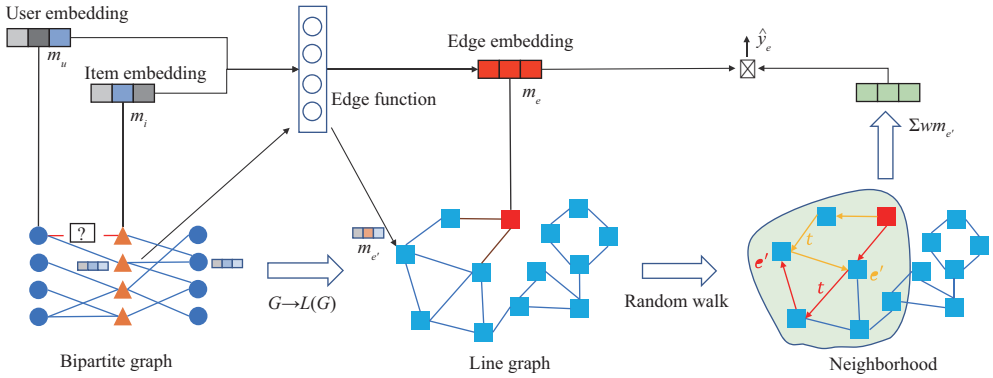


**Figure 2** (Color online) The framework of LECF.

**Table 1** Three binary functions to represent an edge. || is the concatenation operation and ⊙ is the element-wise product

| Function | Definition |
|---|---|
| Sum | $f(\boldsymbol{m}_u, \boldsymbol{m}_i) = \boldsymbol{m}_u + \boldsymbol{m}_i$ |
| Concat | $f(\boldsymbol{m}_u, \boldsymbol{m}_i) = \boldsymbol{m}_u \| \| \boldsymbol{m}_i$ |
| Hadamard | $f(\boldsymbol{m}_u, \boldsymbol{m}_i) = \boldsymbol{m}_u \odot \boldsymbol{m}_i$ |

## 3 LECF: learnable edge collaborative filtering

Figure 2 shows the overview of LECF. In LECF, the historical interactions between users and items are modeled as a bipartite graph $G$. Users and items are represented by latent dense vectors through looking up the embedding matrix. During the prediction, to estimate the score $\hat{y}_{ui}$ of a query edge $e(u,i)$, LECF constructs a line graph $L(G)$ on the basis of the bipartite graph, and the latent representation of the node in $L(G)$ is obtained from a binary edge function. Then LECF extracts a subgraph within the $l$-hop neighborhood of node $e(u,i)$ in $L(G)$ and computes the importance weights of the nodes in subgraph with respect to the query edge $e(u,i)$ via the biased random walk. Note that in $L(G)$ except the node corresponding to the query edge, the other nodes are the observed edges (i.e., historical interactions) in the bipartite graph $G$. Finally, LECF calculates the similarity between query edge and the observed edges, and output the weighted sum of similarities as the predicting score $\hat{y}_{ui}$.

### 3.1 User, item and interaction embeddings

#### 3.1.1 *User and item embeddings*

To encode user's preferences and item's attributes, in LECF, we use latent dense vectors $\boldsymbol{m}_u$ and $\boldsymbol{m}_i$ as the intrinsic representations of users and items. Therefore, we build a parameter matrix $\boldsymbol{M} \in \mathbb{R}^{(|U|+|I|)\times d}$ as an embedding look-up table, where $d$ denotes the embedding size, $|U|$ and $|I|$ are the total numbers of users and items, respectively.

#### 3.1.2 *Interaction embeddings*

Different from previous models that optimize the user and item embeddings by learning an object-object relationship, LECF learns the similarity among interactions based on the interaction representations. Note that we call both observed interactions and unobserved interactions as edges for simplicity. Inspired by the link prediction task [13], we introduce a binary function with the related user embedding $\boldsymbol{m}_u$ and item embedding $\boldsymbol{m}_i$ to represent an edge $e(u,i)$, that is

$$\boldsymbol{m}_{e(u,i)} = f(\boldsymbol{m}_u, \boldsymbol{m}_i). \tag{2}$$

The goal of the binary function is to generate an edge embedding $\boldsymbol{m}_{e(u,i)}$ by utilizing $\boldsymbol{m}_u$ and $\boldsymbol{m}_i$. In this work, we consider three options for this binary function, and they are sum, concat, and Hadamard functions, whose definitions are summarized in Table 1.

On top of the edge embeddings, we use the inner product to evaluate the edge-edge similarity. The similarity between the query edge $e(u,i)$ and the observed edge implies the support of observed interaction

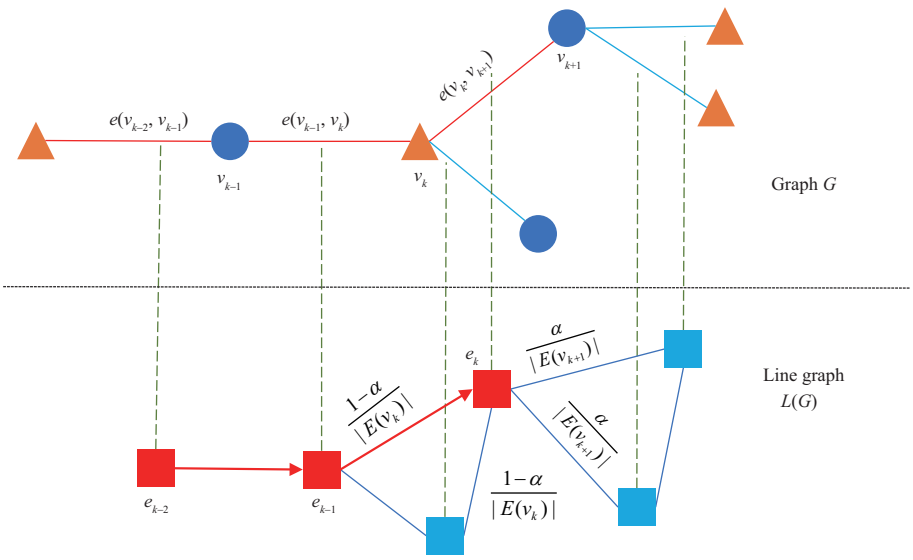


**Figure 3** (Color online) Illustration of the random walk procedure with parameter $\alpha$. Each square vertex in $L(G)$ represents an edge in $G$, while the red color indicates this vertex has been traversed.

for recommending the query item $i$ to query user $u$. The more similar the query edge to the observed edges, the higher score of the query edge for prediction.

### 3.2 Biased random walk for neighborhood searching

Given a query edge $e(u,i)$, it is impractical to involve all observed edges to compute the edge-edge similarity. Furthermore, the importance of an observed edge varies with respect to the query edge. Here we propose a biased random walk model over the line graph $L(G)$ to compute the importance within an $l$-hop neighborhood.

Concretely, we simulate a biased random walk of length $l$ starting at the query edge $e$ in the line graph. Assuming that $e_k = e(v_k, v_{k+1})$ denotes the $k$th node in the walk, the transition probability for the next node $e_{k+1}$ is defined as

$$\boldsymbol{P}(e_{k+1}|e_k) = \begin{cases} \dfrac{1-\alpha}{|E(v_k)|}, & \text{if } e_{k+1} \in E(v_k), \\ \dfrac{\alpha}{|E(v_{k+1})|}, & \text{if } e_{k+1} \in E(v_{k+1}), \end{cases} \tag{3}$$

where the $E(v)$ denotes the set of edges that related with the node $v$ in the bipartite graph, and the $|E(v)|$ is the number of edges in $E(v)$. By introducing the $|E(v)|$, we give the edges of high degree node lower importance since they are generally the source of noise [11]. Figure 3 shows an example of a random walk that just traversed $e_{k-1}$ and now resides at $e_k$.

With the help of the biased random walk model, we only use the edges reached by an $l$-step random walk to compute the prediction score. In other words, the edges within the $l$-hop neighborhood of query edge $e$ in the line graph are used. Then we adopt the reaching probability as the importance of the observed edges [14, 15], and the formula is

$$w_e^l(e') = \sum_{t \in T^l : e \rightsquigarrow e'} P[t], \tag{4}$$

where $T^l : e \rightsquigarrow e'$ denotes the set contains all walks of length $l$ starting at $e$ and ending at $e'$, and $P[t]$ is the probability of walk $t$.

#### 3.2.1 *Outward parameter* $\alpha$

In (3), we introduce $\alpha$ to control the direction of random walk, i.e., the walker selects the next step from $E(v_{k+1})$ with the probability $\alpha$, and from $E(v_k)$ with the probability $1-\alpha$. In other words, $\alpha$ controls the affinity of the walk leaving the neighborhood of query edge $e$ in the line graph. Recall the walk in

Figure 3, if $\alpha > 0.5$, the walk tends to select the edge from the $E(v_{k+1})$, which becomes more far away from the edge $e_{k-1}$. Such behavior is similar to the traversing order in depth-first search (DFS) which explores as far as possible along each branch. In contrast, if $\alpha < 0.5$, the random walk is biased towards edges in $E(v_k)$, which are close to $e_{k-1}$. Such walks are similar to the breadth-first search (BFS) behavior which explores the immediate neighbor nodes at the present depth before moving to the nodes at the next depth level. Hence, the parameter $\alpha$ allows the random walk in line graph approximately interpolate between BFS and DFS, which is similar to the random walk search strategy of node2vec [13]. In LECF, we can adjust the importance of edges by tuning the parameter $\alpha$ to get a trade-off between BFS and DFS.

### 3.3 Model prediction and optimization

Finally, LECF computes the score of a query edge $e(u, i)$ as a weighted sum of edge-edge similarity:

$$\hat{y}_e = \sum_{e' \in \mathcal{O}_e^l} w_e^l(e')(\boldsymbol{m}_e^{\mathrm{T}} \boldsymbol{m}_{e'}), \tag{5}$$

where $\mathcal{O}_e^l$ denotes the set of edges that can reach by an $l$-step random walk starting from the query edge $e$. The weight $w_e^l$ represents the importance of the observed edges. It also can be viewed as a normalization term which helps avoid the scale of scores increasing with the number of the observed edges.

We leverage the Bayesian personalized ranking (BPR) optimization criterion to learn model parameters [16], which is widely adopted in CF models. Specifically, it tries to maximize the difference between the scores of the positive and negative samples. The loss function can be described as

$$\mathcal{L} = \sum_{(u,i,j) \in D_s} - \ln \sigma(\hat{y}_{e(u,i)} - \hat{y}_{e(u,j)}) + \lambda ||\Theta||_2^2, \tag{6}$$

where $D_s = \{(u, i, j) | e(u, i) \in E, e(u, j) \notin E\})$ denotes the training set, $E$ is the set of edges, i.e., all observed interactions, $\sigma$ is the sigmoid function, $\Theta$ denotes the model parameters, and $\lambda$ is the parameter to control the $L_2$ regularization strength.

### 3.4 The connection with existing methods

In this subsection, we analyze the connection between LECF and existing studies that rely on object-object relationships.

#### 3.4.1 *Methods that rely on similarity relationships*

These methods utilize the user-user similarity (user-based) or item-item similarity (item-based) to predict the score (e.g., Figures 1(b) and (c)). The user-based methods calculate the score based on the similarity between query user and other users that have interacted with the query item, while the item-based methods are based on the similarity between query item and other items that have been interacted by the query user. In the following, we mainly describe the item-based algorithms, while the user-based algorithms are analogous. Specifically, the predicting function of item-based methods is

$$\hat{y}_{ui} = \sum_{i' \in I_u^+} s_{ii'}, \tag{7}$$

where $I_u^+$ denote all items $i'$ that query user $u$ has interacted with, and $s_{ii'}$ is a similarity measure. ItemKNN [17] selects the items $i'$ from the intersection of $I_u^+$ and nearest neighbors of query item $i$ to limit the number of items. The learnable models usually model the similarity as the inner product between embeddings [2, 9]:

$$s_{ii'} = \boldsymbol{m}_i^{\mathrm{T}} \boldsymbol{m}_{i'}. \tag{8}$$

We can redefine these methods in the framework of LECF by only considering the item-item similarity to the one-hop neighbors in the line graph. Specifically, we set the length $l$ of random walk to be one, and the edge function to be $\boldsymbol{m}_e = \boldsymbol{m}_i$, then the output score of LECF is

$$\hat{y}_{e(u,i)} = \sum_{e'(u,i') \in E(u)} w_e^1(e') \boldsymbol{m}_i^{\mathrm{T}} \boldsymbol{m}_{i'} + \sum_{e'(u',i) \in E(i)} w_e^1(e') \boldsymbol{m}_i^{\mathrm{T}} \boldsymbol{m}_i. \tag{9}$$

Obviously, by setting the weight of edges in $E(i)$ to be zero, the second term is unavailable and we can exactly recover an item-based model. Moreover, the user-based model also can be recovered in the framework of LECF, by setting $\boldsymbol{m}_e = \boldsymbol{m}_u$, $l = 1$ and the weight of $E(u)$ to be zero in LECF.

#### 3.4.2 *Methods that model user-item interactions*

Besides using the similarity relationship between users or items to predict the score, the other approach is to directly model the interaction relationship between a user and an item (e.g., Figure 1(a)). For example, the BPR [16] and NGCF [1] use the inner product between user embedding and item embedding to predict the interaction relationship:

$$\hat{y}_{ui} = \boldsymbol{m}_u^{\mathrm{T}} \boldsymbol{m}_i. \tag{10}$$

These models based on the user-item relationship can be viewed as a special case of LECF which discards the edge-edge similarity component. In particular, we use hadamard function: $\boldsymbol{m}_e = \boldsymbol{m}_u \odot \boldsymbol{m}_i$ to represent query edge, and replace the embeddings of edges in $\mathcal{O}_e^l$ with vectors whose entries are all equal to one to only keep the interaction information, i.e., $\sum_{e' \in \mathcal{O}_e^l} w_e^l(e') \boldsymbol{m}_{e'} \to \boldsymbol{q}$. Here $\boldsymbol{q}$ is a $d$-dimensional vector whose entries are all equal to one. In this setting, we remove the neighbor information and get

$$\hat{y}_{e(u,i)} = \boldsymbol{m}_e^{\mathrm{T}} \sum_{e' \in \mathcal{O}_e^l} w_e^l(e') \boldsymbol{m}_{e'} = (\boldsymbol{m}_u \odot \boldsymbol{m}_i)^{\mathrm{T}} \boldsymbol{q} = \boldsymbol{m}_u^{\mathrm{T}} \boldsymbol{m}_i. \tag{11}$$

## 4 Efficient training via propagation

In LECF, a main operator is to search the observed edges and calculate their weights. A very intuitive method is to simulate all possible $l$-step random walks starting from each query edge in the line graph, but it is time-consuming when $l$ is large. Therefore, we propose a novel propagation algorithm to calculate the importance of the observed edges with respect to the query edge. The propagation algorithm accelerates both the training and inference. In the end of this section, we analyze the time complexity of the propagation algorithm.

### 4.1 Propagation rule

**Takeaway.** We summarize our propagation rule as follows:

$$\hat{y}_e = \boldsymbol{m}_e^{\mathrm{T}} (\boldsymbol{H}_i^l + \boldsymbol{H}_u^l), \tag{12}$$

where

$$\begin{cases} \boldsymbol{H}_i^1 = \displaystyle\sum_{e' \in E(i)} \frac{0.5}{|E(i)|} \boldsymbol{m}_{e'}, & \text{if } l = 1, \\ \boldsymbol{H}_i^l = (1-\alpha) \boldsymbol{H}_i^{l-1} + \dfrac{\alpha}{|E(i)|} \displaystyle\sum_{e(i,u') \in E(i)} \boldsymbol{H}_{u'}^{l-1}, & \text{otherwise}, \end{cases} \tag{13}$$

and

$$\begin{cases} \boldsymbol{H}_u^1 = \displaystyle\sum_{e' \in E(u)} \frac{0.5}{|E(u)|} \boldsymbol{m}_{e'}, & \text{if } l = 1, \\ \boldsymbol{H}_u^l = (1-\alpha) \boldsymbol{H}_u^{l-1} + \dfrac{\alpha}{|E(u)|} \displaystyle\sum_{e(u,i') \in E(u)} \boldsymbol{H}_{i'}^{l-1}, & \text{otherwise}. \end{cases} \tag{14}$$

The intuition of the above propagation rule is that at each iteration, $\boldsymbol{H}_i^l$ and $\boldsymbol{H}_u^l$ aggregate information from their neighbor edges $E(i)$ and $E(u)$ respectively, and as the iteration goes on, $\boldsymbol{H}_i^l$ and $\boldsymbol{H}_u^l$ incrementally gain information from the whole graph. Finally, we predict the score according to (12). The pseudocode of the propagation rule is listed in Algorithm 1.

**Detailed analysis.** Given walks $T^l : e \to e_1 \rightsquigarrow e_l$ starting from the query edge $e(u,i)$ to $e_l$, where $e_1$ is the first step away from the query edge $e(u,i)$ and can be selected from either $E(i)$ or $E(u)$. For the first step of random walks, we treat $E(i)$ and $E(u)$ equivalently, which means the probability of choosing $e_1$ from $E(i)$ or $E(u)$ is 0.5. Then, in the cases of $e_1 \in E(i)$, the probability of traveling $e_1$ as the first

**Algorithm 1** LECF propagation algorithm

**Require:** Embedding matrix $\boldsymbol{M}$; bipartite graph $G(V,E)$; the length of random walk $l$; outward parameter $\alpha$; edge function $f$; the set of query edges $E_q$.

**for** $k=1,\ldots,l$ **do**
  **if** $k=1$ **then**
    **for** $v$ in $V$ **do**
      $\boldsymbol{H}_v^1 \leftarrow \sum_{e(v,v')\in E(v)} \frac{0.5}{|E(v)|} f(\boldsymbol{m}_v, \boldsymbol{m}_{v'})$;
    **end for**
  **else**
    **for** $v$ in $V$ **do**
      $\boldsymbol{H}_v^k \leftarrow (1-\alpha)\boldsymbol{H}_v^{k-1} + \sum_{e(v,v')\in E(v)} \frac{\alpha}{|E(v)|} \boldsymbol{H}_{v'}^{k-1}$;
    **end for**
  **end if**
**end for**
**for** $e(u,i)$ in $E_q$ **do**
  $\boldsymbol{m}_{e(u,i)} = f(\boldsymbol{m}_u, \boldsymbol{m}_i)$;
  $\hat{y}_{e(u,i)} = \boldsymbol{m}_{e(u,i)}^{\mathrm{T}}(\boldsymbol{H}_i^l + \boldsymbol{H}_u^l)$;
**end for**

step is $\boldsymbol{P}(e_1|e) = \frac{0.5}{|E(i)|}$; in the cases of $e_1 \in E(u)$, the probability becomes $\boldsymbol{P}(e_1|e) = \frac{0.5}{|E(u)|}$. Therefore, we can rewrite (4) as follows:

$$\begin{aligned} w_e^l(e') &= \sum_{t\in T^l: e\rightsquigarrow e'} p[t] \\ &= \sum_{e_1\in E(i)} \sum_{t\in T^l: e\to e_1\rightsquigarrow e'} p[t] + \sum_{e_1\in E(u)} \sum_{t\in T^l: e\to e_1\rightsquigarrow e'} p[t] \\ &= \sum_{e_1\in E(i)} \frac{0.5}{|E(i)|} \sum_{t'\in T^{l-1}: e_1\rightsquigarrow e'} p[t'] + \sum_{e_1\in E(u)} \frac{0.5}{|E(u)|} \sum_{t'\in T^{l-1}: e_1\rightsquigarrow e'} p[t'] \\ &= \sum_{e_1\in E(i)} \frac{0.5}{|E(i)|} w_{e_1}^{l-1}(e') + \sum_{e_1\in E(u)} \frac{0.5}{|E(u)|} w_{e_1}^{l-1}(e'). \end{aligned} \tag{15}$$

Consequently, Eq. (5) is rewritten as

$$\begin{aligned} \hat{y}_e &= \boldsymbol{m}_e^{\mathrm{T}} \sum_{e'\in\mathcal{O}_e^l} w_e^l(e')\boldsymbol{m}_{e'} \\ &= \boldsymbol{m}_e^{\mathrm{T}} \left( \sum_{e_1\in E(i)} \sum_{e'\in\mathcal{O}_{e_1}^{l-1}} \frac{0.5}{|E(i)|} w_{e_1}^{l-1}(e')\boldsymbol{m}_{e'} + \sum_{e_1\in E(u)} \sum_{e'\in\mathcal{O}_{e_1}^{l-1}} \frac{0.5}{|E(u)|} w_{e_1}^{l-1}(e')\boldsymbol{m}_{e'} \right). \end{aligned} \tag{16}$$

We further define $\boldsymbol{H}_i^l$ as the information learned from the walks of length $l$ which have $e_1 \in E(i)$, and $\boldsymbol{H}_u^l$ as the information learned from the walks of length $l$ which has $e_1 \in E(u)$:

$$\boldsymbol{H}_i^l = \sum_{e_1\in E(i)} \sum_{e'\in\mathcal{O}_{e_1}^{l-1}} \frac{0.5}{|E(i)|} w_{e_1}^{l-1}(e')\boldsymbol{m}_{e'}, \tag{17}$$

$$\boldsymbol{H}_u^l = \sum_{e_1\in E(u)} \sum_{e'\in\mathcal{O}_{e_1}^{l-1}} \frac{0.5}{|E(u)|} w_{e_1}^{l-1}(e')\boldsymbol{m}_{e'}. \tag{18}$$

Finally, we can simplify (16) as

$$\hat{y}_e = \boldsymbol{m}_e^{\mathrm{T}}(\boldsymbol{H}_i^l + \boldsymbol{H}_u^l). \tag{19}$$

Next, we analyze the recursive version of $\boldsymbol{H}_i^l$ and $\boldsymbol{H}_u^l$.

(1) When $l=1$, we can easily get

$$\boldsymbol{H}_i^1 = \sum_{e'\in E(i)} \frac{0.5}{|E(i)|} \boldsymbol{m}_{e'}, \tag{20}$$

$$\boldsymbol{H}_u^1 = \sum_{e'\in E(u)} \frac{0.5}{|E(u)|} \boldsymbol{m}_{e'}. \tag{21}$$

(2) When $l>1$, for $\boldsymbol{H}_i^l$, the first step $e_1$ is selected from $E(i)$, denoted by $e_1(i,u')$. Similarly, the second step $e_2$ can also be selected from $E(i)$ and $E(u')$ with the probability of $\frac{1-\alpha}{|E(i)|}$ and $\frac{\alpha}{|E(u')|}$, respectively. Then we can further split $\boldsymbol{H}_i^l$ according to the second step $e_2$:

$$\begin{aligned}
\boldsymbol{H}_i^l &= \frac{0.5}{|E(i)|}\sum_{e_1\in E(i)}\sum_{e'\in\mathcal{O}_{e_1}^{l-1}} w_{e_1}^{l-1}(e')\boldsymbol{m}_{e'} \\
&= \frac{0.5}{|E(i)|}\sum_{e(i,u')\in E(i)}\left(\sum_{e_2\in E(i)}\sum_{e'\in\mathcal{O}_{e_2}^{l-2}}\frac{1-\alpha}{|E(i)|}w_{e_2}^{l-2}(e')\boldsymbol{m}_{e'}+\sum_{e_2\in E(u')}\sum_{e'\in\mathcal{O}_{e_2}^{l-2}}\frac{\alpha}{|E(u')|}w_{e_2}^{l-2}(e')\boldsymbol{m}_{e'}\right) \\
&= \frac{1-\alpha}{|E(i)|}\sum_{e(i,u')\in E(i)}\sum_{e_2\in E(i)}\sum_{e'\in\mathcal{O}_{e_2}^{l-2}}\frac{0.5}{|E(i)|}w_{e_2}^{l-2}(e')\boldsymbol{m}_{e'} \\
&\quad+\frac{\alpha}{|E(i)|}\sum_{e(i,u')\in E(i)}\sum_{e_2\in E(u')}\sum_{e'\in\mathcal{O}_{e_2}^{l-2}}\frac{0.5}{|E(u')|}w_{e_2}^{l-2}(e')\boldsymbol{m}_{e'} \\
&= \frac{1-\alpha}{|E(i)|}\sum_{e(i,u')\in E(i)}\boldsymbol{H}_i^{l-1}+\frac{\alpha}{|E(i)|}\sum_{e(i,u')\in E(i)}\boldsymbol{H}_{u'}^{l-1} \\
&= (1-\alpha)\boldsymbol{H}_i^{l-1}+\frac{\alpha}{|E(i)|}\sum_{e(i,u')\in E(i)}\boldsymbol{H}_{u'}^{l-1}.
\end{aligned} \tag{22}$$

For $\boldsymbol{H}_u^l$, we can get

$$\boldsymbol{H}_u^l = (1-\alpha)\boldsymbol{H}_u^{l-1}+\frac{\alpha}{|E(u)|}\sum_{e(u,i')\in E(u)}\boldsymbol{H}_{i'}^{l-1}. \tag{23}$$

### 4.2 Matrix form of the rule

We provide the matrix form of the propagation rule for LECF. Taking the LECF with the sum edge function as an example,

$$\begin{aligned}
\boldsymbol{H}_i^1 &= 0.5\left(\boldsymbol{m}_i+\sum_{e'(i,u')\in E(i)}\frac{1}{|E(i)|}\boldsymbol{m}_{u'}\right), \\
\boldsymbol{H}_u^1 &= 0.5\left(\boldsymbol{m}_u+\sum_{e'(u,i')\in E(u)}\frac{1}{|E(u)|}\boldsymbol{m}_{i'}\right).
\end{aligned} \tag{24}$$

We can aggregate the first-hop neighbor edges in the matrix form equivalent to (24):

$$\boldsymbol{H}^1 = 0.5(\widehat{A}+T)\boldsymbol{M}, \tag{25}$$

where $T$ is an identity matrix. And $\widehat{A}=D^{-1}A$, where $A$ denote the adjacency matrix of bipartite graph, $D$ is the diagonal degree matrix. $\boldsymbol{H}^1$ is a matrix contains all $\boldsymbol{H}_v^1$, i.e., $[\boldsymbol{H}_{u_1}^1,\ldots,\boldsymbol{H}_{u_{|U|}}^1,\boldsymbol{H}_{i_1}^1,\ldots,\boldsymbol{H}_{i_{|I|}}^1]$. The LECF with other edge functions in Table 1 can be transformed to matrix form similarly:

$$\boldsymbol{H}^1 = \begin{cases} 0.5(\widehat{A}\boldsymbol{M}\odot\boldsymbol{M}), & \text{if } \boldsymbol{m}_e=\boldsymbol{m}_u\odot\boldsymbol{m}_i, \\ 0.5(\widehat{A}\boldsymbol{M}||\boldsymbol{M}), & \text{if } \boldsymbol{m}_e=\boldsymbol{m}_u||\boldsymbol{m}_i. \end{cases} \tag{26}$$

When $l>1$, no matter which edge function is applied, LECF can explore the high-hop neighbor edges by the matrix form of (22) and (23):

$$\widetilde{A} = \alpha\widehat{A}+(1-\alpha)T, \tag{27}$$

$$\boldsymbol{H}^l = \widetilde{A}\boldsymbol{H}^{l-1} = \widetilde{A}^{l-1}\boldsymbol{H}^1. \tag{28}$$

Then we can retrieve the $\boldsymbol{H}_u^l$ and $\boldsymbol{H}_i^l$ from the $\boldsymbol{H}^l$ to predict the score according to (12). The matrix form-based propagation algorithm allows LECF to optimize parameters and infer results efficiently.

### 4.3 Time complexity analysis

Here we analyze the computational complexity of LECF with the sum edge function. For the $k$-th step of walk, according to (20)–(23), the computational complexity is $O(|A+T|d)$, where the $|A+T|$ denotes the number of nonzero entries in the matrix $A+T$, and $d$ is the embedding size. For the prediction function, i.e., (19), the time complexity in a training epoch is $O(|E|d)$. Therefore, during the training, the time complexity of an epoch is $O(\sum_1^l |A+T|d+|E|d)$.

With regard to the time complexity of inference, LECF is comparable to the traditional MF. After training, we get two matrices: $\boldsymbol{M}$ and $\boldsymbol{H}^l$. We only need to retrieve $\boldsymbol{m}_u$ and $\boldsymbol{m}_i$ from $\boldsymbol{M}$, and retrieve $\boldsymbol{H}_u^l$ and $\boldsymbol{H}_i^l$ from $\boldsymbol{H}^l$; then we calculate the score according to (19). In this way, for each query edge, the time complexity of predicting its score is $O(d)$.

## 5 Related work

### 5.1 Methods based on similarity relationship

Both user-based methods and item-based methods belong to this category. Specifically, user-based methods predict the score of $(u,i)$ based on the similarity of $u$ to all other users that have interacted with the item $i$, while item-based methods are based on the similarity of $i$ to all other items that user $u$ has interacted with [18]. UserKNN and ItemKNN [17] are the typical examples of these methods, integrated with the nearest neighbor search. More recently, Wang et al. [19] proposed a semantic similarity based on the information in the linked open data. Bag et al. [20] argued that considering only co-rated items is not proper in sparse datasets and developed the relevant Jaccard similarity. Besides, many research efforts have been devoted to applying similarity models in sequential recommendations [21, 22]. However, these methods only consider the first-order interactions, ignoring the majority of other complex interaction information.

### 5.2 Model-based methods

In model-based methods, users and items are converted to vectorized representations, and the user-item interactions are captured by an interaction function based on the representations. For example, MF [23] utilizes the inner product between the user and item representation to model the user-item interaction relationship. Recent years have witnessed a surge of interest in applying neural networks to recommendation tasks [24]. NeuMF [6] is a prominent neural model that jointly learns an MF and a feed forward neural network. CMN [2] enhances the MF by leveraging the memory network to store the users' neighborhood information. Additionally, the methods to generate vectorized representations also have attracted much researchers' attention. In [25, 26], the authors proposed models mapping users and items into the hyperbolic spaces. To capture the learning uncertainties, PMLAM describes the users and items with Gaussian distributions [27]. These methods directly model the interaction relationships and underutilize the user-user similarity and item-item similarity in data.

### 5.3 Graph-based methods

Recently, a lot of studies enhance the CF by integrating it with graph convolutional networks. SpectralCF [28] uses a spectral convolution operation in a bipartite graph, but its feature decomposition leads to high computational complexity. NGCF [1] considers the importance of high-order connectivity and only encodes it in the embedding layer. FNBE [29] improves the performance of social recommendation by folding the user-item bipartite graph to explore the implicit higher-order relations. Several studies also introduce graph networks to integrate auxiliary data. Cui et al. [30] adopted user social networks and POI multimodal contents as hypergraph models to alleviate the data sparseness in CF. In [31], heterogeneous information networks are constructed for cold-start recommendation. LightGCN [32] is the state-of-the-art graph convolutional network for the implicit CF, which simplifies the GCN [33] and only remains the component of neighborhood aggregation. Although these GCN-based models also have been proposed to leverage the high-order interactions, they usually compress user/item and neighbors into single user/item embeddings, in which the interaction information is mixed and the similarity relationship is hidden. Furthermore, they mostly adopt a simple interaction function between user and item embeddings

**Table 2** Statistics of the datasets

| Dataset | Interactions | Users | Items | Sparsity (%) | Average degree |
|---|---|---|---|---|---|
| Pinterest | 1500809 | 55187 | 9916 | 99.73 | 46.11 |
| MovieLens | 1000209 | 6040 | 3706 | 95.53 | 205.26 |
| Citeulike-a | 204987 | 5551 | 16980 | 99.78 | 18.20 |
| Amazon-video | 21809 | 1372 | 7957 | 99.80 | 4.68 |

to predict scores. In contrast, we distinguish the contribution of each interaction and explicitly utilize the similarity relationship to predict scores.

## 6 Experiments and results

### 6.1 Datasets

Table 2 summarizes the statistics of four datasets.

• **Pinterest** [34]**.** This dataset is collected for image recommendation, containing the information of the images that users save or pin to their board.

• **MovieLens** [35]**.** This dataset has been widely used to evaluate CF algorithms, and it contains about one million movie ratings.

• **Amazon-video** [36]**.** This dataset contains the purchasing history of instant video. To ensure the quality of the dataset, we filtered the dataset by retaining only users with at least 10 interactions.

• **Citeulike-a** [37]**.** This dataset is collected from CiteULike, an online service that allows users to manage and discover research papers.

### 6.2 Experimental settings

**Evaluation metrics.** We split the dataset based on the leave-one-out evaluation method, which is widely used in previous work [2, 6]. Specifically, for each user, a positive item and 100 negative items (i.e., the items that the user has not interacted with) are randomly sampled to form the validation set. The test set is constructed in the same way. And the remained positive items are shuffled to form the train set. We use two popular evaluation metrics: hit ratio (HR) and normalized discounted cumulative gain (NDCG) [38], to evaluate the performance of top-$K$ recommendations.

**Baselines.** We compare LECF with the following algorithms:

• **UserKNN.** UserKNN is a traditional user-based CF method, which predicts the score by calculating the user-user similarity.

• **ItemKNN** [17]**.** It computes item-item similarity to provide recommendations.

• **HybridKNN.** This hybrid method ranks items by averaging the scores of ItemKNN and UserKNN.

• **SLIM** [7]**.** Sparse linear model learns a sparse coefficient matrix in which entries represent the similarity between items, and generates the recommendation based on the item-item similarity.

• **BPR** [16]**.** This is a strong CF baseline that adopts the Bayesian personalized ranking loss.

• **NeuMF** [6]**.** This is a popular neural CF baseline, jointly training the MF and multi-layer perception model to predict the score.

• **CMN** [2]**.** It is a memory network based on CF, which uses the memory-augmented neural network to capture the user-user similarity.

• **RP**$^3_\beta$ [39]**.** It ranks items based on the three-step random walk in a bipartite graph.

• **NGCF** [1]**.** This model improves the user and item representations by integrating the high-order connectivity into the embedding learning process.

• **LightGCN** [32]**.** This is a GCN-based recommendation that simplifies the original GCN significantly and only remains the component of neighborhood aggregation.

**Parameter settings.** The models are implemented with MindSpore Lite tool[1)]. During the training, we check the performance on the validation set and select the model with the best performance for testing. For all models, the dimension of the user and item embedding is fixed to 50. We performed a grid search over each model's learning rate from $\{10^{-2}, 10^{-3}, 10^{-4}\}$ and regularization terms $\lambda$ from $\{10^{-1}, 10^{-2}, \ldots, 10^{-5}\}$. For NGCF and LightGCN, we tune the number of propagation layers from 1 to

1) https://www.mindspore.cn/.

**Table 3** Overall performance comparison[a)]

| | Amazon-video | | Citeulike-a | | MovieLens | | Pinterest | |
|---|---|---|---|---|---|---|---|---|
| | HR@5 | NDCG@5 | HR@5 | NDCG@5 | HR@5 | NDCG@5 | HR@5 | NDCG@5 |
| UserKNN | 0.4585 | 0.3666 | 0.7244 | 0.6230 | 0.4113 | 0.3145 | 0.6095 | 0.4596 |
| ItemKNN | 0.4745 | 0.3833 | 0.7485 | 0.6233 | 0.6500 | 0.4850 | 0.6910 | 0.5153 |
| HybridKNN | 0.4934 | 0.3540 | 0.7920 | 0.6587 | 0.6634 | 0.5088 | 0.7064 | 0.5181 |
| SLIM | 0.4723 | 0.3842 | 0.7469 | 0.6199 | 0.6817 | 0.5177 | 0.6803 | 0.5022 |
| BPR | 0.5029 | 0.3880 | 0.8088 | 0.6730 | 0.6836 | 0.5209 | 0.7159 | 0.5115 |
| NeuMF | 0.4206 | 0.3099 | 0.7849 | 0.6496 | 0.6927 | 0.5229 | 0.7126 | 0.5032 |
| CMN | 0.4913 | 0.3811 | 0.8090 | 0.6589 | 0.6010 | 0.4407 | 0.7265 | 0.5219 |
| $RP^3_\beta$ | 0.4679 | 0.3778 | 0.7599 | 0.6571 | 0.6649 | 0.5045 | 0.6566 | 0.4901 |
| NGCF | 0.5138 | 0.4006 | 0.8000 | 0.6556 | 0.6951 | 0.5257 | 0.7316 | 0.5258 |
| LightGCN | 0.5364 | 0.4212 | 0.8108 | 0.6717 | 0.6912 | 0.5256 | 0.7442 | 0.5406 |
| LECF (sum) | **0.5568** | **0.4315** | **0.8330** | **0.6886** | **0.7141** | **0.5461** | **0.7599** | **0.5550** |
| LECF (concat) | 0.5481 | 0.4160 | 0.8230 | 0.6840 | 0.7025 | 0.5348 | 0.7551 | 0.5524 |
| LECF (Hadamard) | 0.5066 | 0.3952 | 0.8098 | 0.6629 | 0.7031 | 0.5323 | 0.7419 | 0.5359 |
| LECF-no-walk | 0.4910 | 0.3678 | 0.8061 | 0.6543 | 0.6715 | 0.5062 | 0.7267 | 0.5196 |
| $p$-value | 4.75E−3 | 1.46E−2 | 1.17E−3 | 2.06E−4 | 1.18E−3 | 1.77E−5 | 2.36E−2 | 5.42E−3 |

a) Best results are highlighted in bold. p-value is evaluated between the LECF (sum) and LightGCN.

10 and node dropout ratio from 0.0 to 0.8. For CMN, we use pretrained GMF model [6] as suggested and vary the CMN's hop from $\{1, 2, 3\}$. Without explicit mention, we show the result of LECF with a sum edge function. The default outward parameter $\alpha$ is 0.5. And the default length of random walk is set to 10 for Amazon-video datasets and 5 for Citeulike-a, MovieLens, and Pinterest datasets.

### 6.3 Baseline comparison

Table 3 shows the results of LECF and baselines for HR and NDCG with cut offs at 5 on four datasets. We denote LECF with sum edge function as 'LECF (sum)', with concat edge function as 'LECF (concat)', and with Hadamard edge function as 'LECF (Hadamard)'. Besides, to verify the effectiveness of the biased random walk component, we replace the importance weight of edges in LECF with the average operator, and denote such variant of LECF as 'LECF-no-walk'. We have the following observations.

• Encouragingly, LECF (sum) achieves the best performance on all the datasets, and this demonstrates the effectiveness of our proposed LECF. Furthermore, we conduct one-sample t-tests of LECF (sum) and the best baseline (i.e., LightGCN), and the p-values $<0.05$ indicate that the improvements of LECF (sum) over LightGCN are statistically significant. Compared with the UserKNN, ItemKNN, HybridKNN and SLIM which only consider the interactions of query user/item. LECF can employ more interaction information for predicting scores by increasing the length of random walk to utilize the high-hop neighbor edges. Compared with BPR, NeuMF, NGCF, and LightGCN which directly model the user-item interaction, LECF (sum) can explicitly utilize a similarity relationship in the data to predict scores by learning the edge-edge similarity. LECF (concat) also consistently yields better performance than baselines. LECF (Hadamard) achieves higher accuracy than most of the baselines except on Amazon-video dataset. This is because Amazon-video is really sparse and has an average degree of 4.68. Hadamard edge function might not well model such sparse data.

• The performance of LECF-no-walk is poor, sometimes even worse than some baselines such as NeuMF. This observation highlights the significance of considering the importance of edges which is introduced by the biased random walk component. The main reason is that there is much noise in neighbor edges, and we need a weight mechanism to alleviate this problem.

• The methods (UserKNN, ItemKNN, and SLIM) that rely on user-user or item-item similarity achieve poor performance on all datasets, and perform worse than BPR and NeuMF which directly model the user-item interaction. Since only a few interaction information from the data is employed to predict the score, they are hard to estimate a good score for ranking items.

• CMN outperforms NeuMF on most datasets. Although both use the deep learning architecture, CMN considers not only the user-item interaction relationship but also the user-user similarity, while the NeuMF only models the user-item interaction. It verifies the benefit of combining the similarity relationship and user-item interaction function for predicting scores.

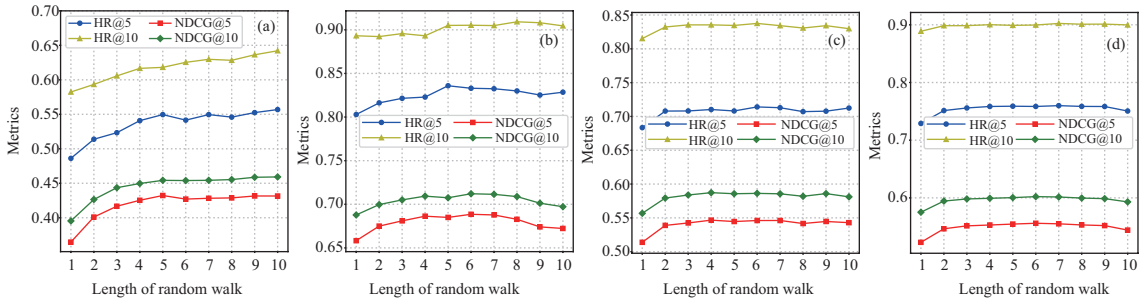


**Figure 4** (Color online) The performance of LECF with different lengths of random walk. (a) Amazon-video; (b) Citeulike-a; (c) MovieLens; (d) Pinterest.

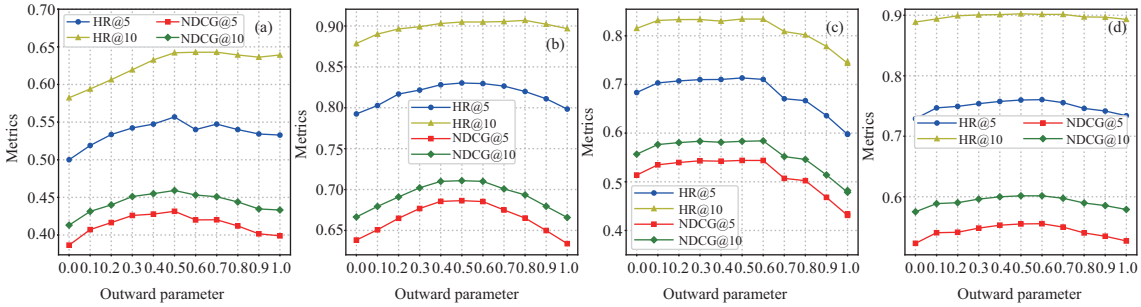


**Figure 5** (Color online) The performance of LECF with different outward parameters. The length of random walk is the default value. (a) Amazon-video; (b) Citeulike-a; (c) MovieLens; (d) Pinterest.

### 6.4 Impact of the length of random walk

To investigate the impact of the length of random walk, we vary the parameter $l$ of LECF from 1 to 10. Figure 4 shows the experimental results. The following observations are made from the results:

• When the length of the random walk is one, the model gives unfavorable results across all the datasets. This verifies that only considering the interaction directly related to query users or items is insufficient for predicting the score.

• When the model considers the further observed edges (e.g., $l = 2$) in the line graph, we can see that the performance increases drastically. And on the Citeulike-a, MovieLens, and Pinterest datasets, the performance of LECF remains relatively stable when $l$ is greater than 5. These results mean that considering further observed edges is beneficial for performance. However, stacking too many edges does not significantly improve accuracy. It is not necessary to consider all the observed edges.

### 6.5 Impact of outward parameter $\alpha$

The search strategy in LECF is both flexible and controllable through the outward parameter $\alpha$. In this experiment, we investigate its impact on the performance by tuning $\alpha$ in the range of $0 \sim 1$. The results are shown in Figure 5.

As analyzed in Subsection 3.2, the parameter $\alpha$ allows our neighborhood search procedure to consider the trade-off between BFS and DFS. When $\alpha = 0$, the random walk will be restricted within the first-order neighbor edges. In this case, the performance is poor on all the datasets. When high-hop neighbor edges are considered, the performance increases drastically. However, when the parameter $\alpha$ continues to increase, the performance goes down again on four datasets. The best results are achieved when $\alpha \approx 0.5$.

### 6.6 Analysis of edge-edge similarity

To verify that our LECF captures the meaningful edge-edge similarity, we choose the attributes in the MovieLens dataset, which include the age, occupation, and gender of users and the title of movies, to compute the ground-truth similarity between edges. In other words, two edges $(u_1, i_1)$ and $(u_2, i_2)$ are similar (or attribute match) if $u_1$ and $u_2$ have similar attributes, and $i_1$ and $i_2$ have similar attributes. In the experiment, for each query edge, we find its most similar observed edge in the bipartite graph and compute the attribute matches between the users (items) associated with the edges. Table 4 reports the

**Table 4** Ratio of attribute matches between the query edge and the most similar observed edge

| Attribute | Top (%) | Random (%) | Diff (%) |
|---|---|---|---|
| User job | 76.97 | 8.12 | **68.85** |
| User age | 81.33 | 22.80 | **58.53** |
| User gender | 91.90 | 63.41 | **28.49** |
| Movie genre | 56.95 | 39.31 | **17.64** |

**Table 5** Training time (s) and inference efficiency (ms per query) of different models

| | Amazon-video | | Citeulike-a | | MovieLens | | Pinterest | |
|---|---|---|---|---|---|---|---|---|
| | Train | Inference | Train | Inference | Train | Inference | Train | Inference |
| BPR | 5 | 1.24 | 43 | 1.24 | 224 | 1.32 | 372 | 1.27 |
| NeuMF | 10 | 3.50 | 80 | 3.42 | 389 | 3.48 | 620 | 3.34 |
| CMN | 78 | 2.29 | 1658 | 3.60 | 2785 | 7.14 | 12875 | 3.34 |
| NGCF | 92 | 1.39 | 933 | 1.32 | 1566 | 1.45 | 1960 | 1.32 |
| LightGCN | 26 | 1.38 | 295 | 1.28 | 1628 | 1.34 | 2749 | 1.29 |
| LECF | 35 | 1.38 | 326 | 1.29 | 1436 | 1.40 | 2854 | 1.31 |

ratio of attribute matches for 100 M query edges. Note that we did not use the attributes in the training of LECF.

To check whether the result of LECF is significant or not, for each query edge, we computed the match probability between the query edge and a randomly selected edge. We observed that the percentage of attribute matches between similar edges is always higher than that of randomly selected edges. These results show that the edge-edge similarity between similar user-item pairs is high in LECF. In this way, if a similar user likes a similar item, this edge will provide a high similarity, and then the model will tend to recommend the query item to the query user, which is consistent with the idea of CF. It also verifies that LECF can utilize the correlation among data, including both user-user similarity and item-item similarity.

### 6.7 Efficiency comparison

Table 5 shows the training and inference time of different recommendation models with the optimal hyper-parameter setting. BPR and NeuMF train faster than LECF and the other two graph-based models because of their simple model architectures, but they sacrifice the model accuracy on four datasets. Although LECF has a similar training efficiency compared to the graph-based models — NGCF and LightGCN, it achieves a better model performance than the ones of NGCF and LightGCN.

The inference efficiency of LECF is comparable to the BPR and the other two graph-based models. With the help of our propagation algorithm, given the query edge, LECF does not need to explicitly conduct random walks to search the observed edges for the computation of the weights. It only needs to retrieve $\boldsymbol{m}_u$ and $\boldsymbol{m}_i$ from $\boldsymbol{M}$, and retrieve $\boldsymbol{H}_u^l$ and $\boldsymbol{H}_i^l$ from $\boldsymbol{H}^l$, then simply calculates the score according to (12).

## 7 Conclusion

In this work, we proposed a novel variant of CF, which generates a recommendation based on the similarity between edges, rather than the simple relationship between objects (users and items) commonly used in previous work. Based on this idea, we designed a new framework LECF, which calculates the weighted sum of edge-edge similarity as the score to rank items. Comprehensive experiments on four datasets demonstrate the effectiveness of LECF.

**Acknowledgements** This work was supported by National Key Research and Development Program of China (Grant No. 2018YFB1004403), National Natural Science Foundation of China (Grant Nos. U1936104, 61902037, 61832001), ARC Discovery Project (Grant No. DP190101985), CAAI-Huawei MindSpore Open Fund, Beijing Academy of Artificial Intelligence (BAAI), PKU-Baidu Fund (Grant No. 2019BD006), and Fundamental Research Funds for the Central Universities (Grant No. 2020RC25).

**References**

1 Wang X, He X N, Wang M, et al. Neural graph collaborative filtering. In: Proceedings of the 42nd International ACM SIGIR Conference, Paris, 2019. 165–174

2 Ebesu T, Shen B, Fang Y, et al. Collaborative memory network for recommendation systems. In: Proceedings of the 41st International ACM SIGIR Conference, Ann Arbor Michigan, 2018. 515–524
3 Hosseini B, Montagne R, Hammer B. Deep-aligned convolutional neural network for skeleton-based action recognition and segmentation. Data Sci Eng, 2020, 5: 126–139
4 Chen J H, Chen W, Huang J J, et al. Co-purchaser recommendation for online group buying. Data Sci Eng, 2020, 5: 280–292
5 Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems. Computer, 2009, 42: 30–37
6 He X N, Liao L Z, Zhang H W, et al. Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web Companion, Perth, 2017. 173–182
7 Ning X, Karypis G. SLIM: sparse linear methods for top-n recommender systems. In: Proceedings of the 11th IEEE International Conference on Data Mining, Vancouver, 2011. 497–506
8 Herlocker J L, Konstan J A, Borchers A, et al. An algorithmic framework for performing collaborative filtering. In: Proceedings of the 22nd International ACM SIGIR Conference, Berkeley, 1999. 230–237
9 Koren Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proceedings of the 14th ACM SIGKDD International Conference, Las Vegas, 2008. 426–434
10 Wang J, de Vries A P, Reinders M J T. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In: Proceedings of the 29th Annual International ACM SIGIR Conference, Seattle, 2006. 501–508
11 Nandanwar S, Murty M N. Structural neighborhood based classification of nodes in a network. In: Proceedings of the 22nd ACM SIGKDD International Conference, San Francisco, 2016. 1085–1094
12 Desrosiers C, Karypis G. A comprehensive survey of neighborhood-based recommendation methods. In: Recommender Systems Handbook. Berlin: Springer, 2011. 107–144
13 Grover A, Leskovec J. Node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference, San Francisco, 2016. 855–864
14 Ying R, He R N, Chen K F, et al. Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference, London, 2018. 974–983
15 Eksombatchai C, Jindal P, Liu J Z, et al. Pixie: a system for recommending 3+ billion items to 200+ million users in real-time. In: Proceedings of the 27th International Conference on World Wide Web Companion, Lyon, 2018
16 Rendle S, Freudenthaler C, Gantner Z, et al. BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, Montreal, 2009. 452–461
17 Deshpande M, Karypis G. Item-based top-n recommendation algorithms. ACM Trans Inf Syst, 2004, 22: 143–177
18 Ghazanfar M A, Prügel-Bennett A, Szedmak S. Kernel-mapping recommender system algorithms. Inf Sci, 2012, 208: 81–104
19 Wang R Q, Cheng H K, Jiang Y L, et al. A novel matrix factorization model for recommendation with LOD-based semantic similarity measure. Expert Syst Appl, 2019, 123: 70–81
20 Bag S, Kumar S K, Tiwari M K. An efficient recommendation generation using relevant Jaccard similarity. Inf Sci, 2019, 483: 53–64
21 Zeng Z J, Lin J, Li L, et al. Next-item recommendation via collaborative filtering with bidirectional item similarity. ACM Trans Inf Syst, 2020, 38: 1–22
22 Bayer I, He X N, Kanagal B, et al. A generic coordinate descent framework for learning from implicit feedback. In: Proceedings of the 26th International Conference on World Wide Web, Perth, 2017. 1341–1350
23 He X N, Zhang H W, Kan M Y, et al. Fast matrix factorization for online recommendation with implicit feedback. In: Proceedings of the 39th International ACM SIGIR Conference, Pisa, 2016. 549–558
24 Cheng H T, Koc L, Harmsen J, et al. Wide & deep learning for recommender systems. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, Boston, 2016. 7–10
25 Feng S S, Tran L V, Cong G, et al. HME: a hyperbolic metric embedding approach for next-POI recommendation. In: Proceedings of the 43rd International ACM SIGIR Conference, New York, 2020. 1429–1438
26 Mirvakhabova L, Frolov E, Khrulkov V, et al. Performance of hyperbolic geometry models on top-n recommendation tasks. In: Proceedings of ACM Conference on Recommender Systems, New York, 2020. 527–532
27 Ma C, Ma L H, Zhang Y X, et al. Probabilistic metric learning with adaptive margin for top-k recommendation. In: Proceedings of the 26th ACM SIGKDD International Conference, New York, 2020. 1036–1044
28 Zheng L, Lu C T, Jiang F, et al. Spectral collaborative filtering. In: Proceedings of the 12th ACM Conference on Recommender System, Vancouver, 2018. 311–319
29 Chen H X, Yin H Z, Chen T, et al. Social boosted recommendation with folded bipartite network embedding. IEEE Trans Knowl Data Eng, 2020. doi: 10.1109/TKDE.2020.2982878
30 Cui C R, Shen J L, Nie L Q, et al. Augmented collaborative filtering for sparseness reduction in personalized POI recommendation. ACM Trans Intell Syst Technol, 2017, 8: 1–23
31 Lu Y F, Fang Y, Shi C. Meta-learning on heterogeneous information networks for cold-start recommendation. In: Proceedings of the 26th ACM SIGKDD International Conference, New York, 2020. 1563–1573
32 He X N, Deng K, Wang X, et al. Lightgcn: simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd International ACM SIGIR Conference, Virtual Event, 2020. 639–648
33 Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th International Conference on Learning Representations, Toulon, 2017
34 Geng X, Zhang H W, Bian J W, et al. Learning image and user features for recommendation in social networks. In Proceedings of the 15th IEEE International Conference on Computer Vision, Santiago, 2015. 4274–4282
35 Harper F M, Konstan J A. The movielens datasets: history and context. ACM Trans Interact Intell Syst, 2015, 5: 1–19
36 He R N, McAuley J. 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, Montreal, 2016. 507–517
37 Wang H, Wang N Y, Yeung D Y. Collaborative deep learning for recommender systems. In: Proceedings of the 21st ACM SIGKDD International Conference, Sydney, 2015. 1235–1244
38 He X N, Chen T, Kan M Y, et al. Trirank: review-aware explainable recommendation by modelingaspects. In: Proceedings of the 24th ACM International Conference on Information and Knowledge Management, Melbourne, 2015
39 Paudel B, Christoffel F, Newell C, et al. Updatable, accurate, diverse, and scalable recommendations for interactive applications. ACM Trans Interact Intell Syst, 2016, 7: 34