Neural Graph Matching based Collaborative Filtering

Yixin Su
University of Melbourne, Australia
yixins1@student.unimelb.edu.au

Sarah M. Erfani∗
University of Melbourne, Australia
sarah.erfani@unimelb.edu.au

Rui Zhang†
www.ruizhang.info, China
rayteam@yeah.net

Junhao Gan
University of Melbourne, Australia
junhao.gan@unimelb.edu.au

ABSTRACT
User and item attributes are essential side-information; their interactions (i.e., their co-occurrence in the sample data) can significantly enhance prediction accuracy in various recommender systems. We identify two different types of attribute interactions, inner interactions and cross interactions: inner interactions are those between only user attributes or those between only item attributes; cross interactions are those between user attributes and item attributes. Existing models do not distinguish these two types of attribute interactions, which may not be the most effective way to exploit the information carried by the interactions. To address this drawback, we propose a neural Graph Matching based Collaborative Filtering model (GMCF), which effectively captures the two types of attribute interactions through modeling and aggregating attribute interactions in a graph matching structure for recommendation. In our model, the two essential recommendation procedures, characteristic learning and preference matching, are explicitly conducted through graph learning (based on inner interactions) and node matching (based on cross interactions), respectively. Experimental results show that our model outperforms state-of-the-art models. Further studies verify the effectiveness of GMCF in improving the accuracy of recommendation.

1 INTRODUCTION
Collaborative Filtering (CF) is one of the most frequently used algorithms in recommender systems. It performs the predictions based on an assumption that similar users will have common preferences on similar items. For example, matrix factorization based recommender systems [14, 19, 36] learn user and item embeddings from user-item interactions (e.g., click, purchase) to exhibit similarities between similar users and between similar items. One of the most important challenges faced by CF is how to consider user and item attributes (e.g., user genders, item colors) to enhance the prediction performance. To this end, recent CF models conduct attribute embedding to capture fine-grained collaborative information and reveal the similarities between attributes [1, 29]. While learning attribute embeddings, considering the co-occurrence of attributes inside each data sample, i.e., attribute interactions, have been proven essential in providing useful information for more accurate predictions [25, 28]. For example, in a movie recommendation system, the director Nolan is a master of producing sci-fi movies. In this scenario, considering the attribute interaction <Nolan, sci-fi> is more effective than considering the two attributes separately. Generally, an attribute-interaction-aware CF model takes the attribute interactions into account (by modeling and aggregating them) to jointly decide the final predictions. Factorization Machine (FM) [25] models each attribute interaction as a dot product of two embedded vectors and aggregates all the modeling results linearly. With the development of Graph Neural Networks (GNNs), Li et al. [21] and Su et al. [30] leverage the relation modeling ability of GNNs to capture more sophisticated attribute interaction information and aggregate the information through graph learning.

While these models capture the co-occurrence between attributes by modeling attribute interactions, we argue that they may not be sufficient to yield a satisfactory joint decision. The key reason is that these models simply treat all the attribute interactions equally, and hence, model and aggregate them in the same way. However, different attribute interactions should have different impacts on the final prediction. Specifically, when the attribute interactions are exploited to perform joint decisions, they could play different roles on similar items. For example, matrix factorization based recommendation algorithms in recommender systems. It performs the predictions based on an assumption that similar users will have common preferences on similar items. For example, matrix factorization based recommender systems [14, 19, 36] learn user and item embeddings from user-item interactions (e.g., click, purchase) to exhibit similarities between similar users and between similar items. One of the most important challenges faced by CF is how to consider user and item attributes (e.g., user genders, item colors) to enhance the prediction performance. To this end, recent CF models conduct attribute embedding to capture fine-grained collaborative information and reveal the similarities between attributes [1, 29]. While learning attribute embeddings, considering the co-occurrence of attributes inside each data sample, i.e., attribute interactions, have been proven essential in providing useful information for more accurate predictions [25, 28]. For example, in a movie recommendation system, the director Nolan is a master of producing sci-fi movies. In this scenario, considering the attribute interaction <Nolan, sci-fi> is more effective than considering the two attributes separately. Generally, an attribute-interaction-aware CF model takes the attribute interactions into account (by modeling and aggregating them) to jointly decide the final predictions. Factorization Machine (FM) [25] models each attribute interaction as a dot product of two embedded vectors and aggregates all the modeling results linearly. With the development of Graph Neural Networks (GNNs), Li et al. [21] and Su et al. [30] leverage the relation modeling ability of GNNs to capture more sophisticated attribute interaction information and aggregate the information through graph learning.

While these models capture the co-occurrence between attributes by modeling attribute interactions, we argue that they may not be sufficient to yield a satisfactory joint decision. The key reason is that these models simply treat all the attribute interactions equally, and hence, model and aggregate them in the same way. However, different attribute interactions should have different impacts on the final prediction. Specifically, when the attribute interactions are exploited to perform joint decisions, they could play different roles for recommendation. For example, without loss of generality, the goal of movie recommendation is to predict the preference of a user (represented by the user’s attributes) on a movie (represented by the item’s attributes). Therefore, the attribute interactions are naturally
user and each item as an attribute graph, and model and aggregate attribute interactions in a graph matching structure.

• We conduct extensive experiments. Experimental results show that (i) our model outperforms state-of-the-art baselines in terms of accuracy; (ii) GMCF is able to effectively model the two types of attribute interactions to produce accurate predictions.

2 RELATED WORK

In this section, we discuss more related work on attribute-aware CF models, graph neural networks, and graph matching methods.

2.1 Attribute-aware CF models

Factorization machine (FM) [25] is one of the most frequently used collaborative filtering algorithms that takes attribute interactions into account. FM performs interaction modeling between each pair of attributes and sums up all the modeling results to make the final prediction. Some extensions of FM further calculates an attention score [39] or a gate [22] for each interaction modeling result to differentiate each interaction’s importance using the attention mechanism. However, these models consider the structural information linearly, which is ineffective in leveraging attribute interaction to make a joint decision. NFM [13] and DeepFM [11] add a multilayer perceptron (MLP) on top of attributes or attribute interactions, aiming to implicitly capture the structural information in a non-linear way. However, the use of MLP to model the interactions between input variables has been proven less effective [4, 28]. GMCF models and aggregates attribute interactions explicitly in a structure of graph matching, which is more effective in attribute interaction modeling and structural information capturing.

2.2 Graph Neural Networks

Graph neural networks (GNNs) facilitate learning about entities and their relations [18, 23, 40, 43]. Existing work leverages GNNs in various domains, such as molecular property prediction [9, 10] and object relation learning physical systems [3, 6]. Recently, GNNs attract attention in the domain of recommender systems. Some work considers the user-item interactions as a bipartite graph, where an edge between a user and an item indicates an interaction (e.g., click or rate) [15, 31, 35]. These models only consider user-item interactions in GNNs. Other work leverages GNNs to model knowledge graphs for recommendation [33, 34, 38, 42]. These models regard edges as predefined relations between attributes and items (users) instead of between attributes. Therefore, they do not consider the attribute interactions. Li et al. [21] and Su et al. [30] leverage GNNs to perform attribute interaction modeling and aggregation as a graph learning procedure. However, these models analyze all attribute interactions equally, which are ineffective in capturing useful structural information of attribute interactions to make a joint decision. Our model differentiates inner interactions and cross interactions, and models and aggregates these interactions in a graph matching structure that is considered as more suitable for recommendation.

2.3 Graph Matching

Graph matching is a long-standing research topic in computer science such as database and data mining domains [8, 41]. The goal of graph matching is to discover the similarity between two

---

“Figure 1: Illustration of the differences between GMCF (left) and existing work (right). GMCF treats attribute interactions differently in a structure of graph matching, while existing work treats all attribute interactions equally.”

---

1 Our implementation of the GMCF model is available at: https://github.com/ruizhang-ai/GMCF_Neural_Graph_Matching_based_Collaborative_Filtering.
graph form representations. Traditional graph matching algorithms are based on heuristic rules, such as minimal graph edit distance [24, 37], or based on graph kernel methods, such as random walks inside graphs [16, 32] and graph sub-structures [26, 27]. With the development of GNNs, recent work explores neural-based graph matching. Bai et al. [2] fuses the graph-level embeddings learned by GNNs and a node-level matching embedding learned by a pairwise node comparison method. Li et al. [20] uses GNNs to learn two graphs’ embeddings in a Siamese network framework [5], with a node-level matching performed in the fusing procedure. However, the neural graph matching methods have not been fully explored in recommender systems yet. We are the first to represent each user and each item in a graph form and leverage the framework of neural graph matching for preference matching.

3 PROBLEM STATEMENT & BACKGROUND

In this section, we introduce the problem definition, the representations of attribute interactions, and the basic idea of GNNs.

3.1 Problem Statement

Denote by \( J^U \) the universe of the user attributes and by \( J^I \) the universe of the item attributes. An attribute-value pair is a name-value pair, denoted by \((att, val)\), where \(att\) is the name of the attribute and \(val\) is the value on this attribute. For example, \((Male, 1)\) and \((Female, 1)\) mean the user’s gender is Male and Female, respectively, where \(Male \in J^U\) and \(Female \in J^U\) are considered as two user attributes. Let \( D \) be a set of \( N \) training data pairs, denoted by \( D = \{(x_n, y_n)\}_{1 \leq n \leq N} \). In each training data \( x_n, y_n \) ∈ \( D \),

- \( x_n \) is called a data sample, which consists of:
  - a set \( \mathcal{C}^U_n = \{ (e^U = (att, val))_{att \in J^U_n} \} \) of attribute-value pairs with respect to a set of user attributes \( J^U_n \subseteq J^U \), and
  - a set \( \mathcal{C}^I_n = \{ (e^I = (att, val))_{att \in J^I_n} \} \) of attribute-value pairs with respect to a set of item attributes \( J^I_n \subseteq J^I \).

Moreover, \( \mathcal{C}^U_n \) and \( \mathcal{C}^I_n \) are respectively called the characteristic of the user and the item specified in this data sample.

- \( y_n \in \mathbb{R} \) is the implicit feedback (e.g., watched, liked) of the user on the item.

It should be noted that the number of attribute-value pairs in each data sample may be different, as the information of some attributes may be missing, or may contain multiple attributes of the same type, e.g., a movie may belong to multiple genres.

The Recommendation Problem. The goal of the Recommendation Problem studied in this paper is to design a predictive model \( F(x_n) \) such that for an input data sample \( x_n \) (specifying the characteristics of a user and an item), the output of the model, \( F(x_n) \), is a prediction on the true feedback, \( y_n \), of the user on the item.

Our Solution. In this paper, we propose a Neural Graph Matching based Collaborative Filtering (GMCF) model \( F_{GMCF}(x_n) \). Our GMCF models the user and the item characteristics as two graphs, respectively, and leverages these user and item graph representations to predict \( y_n \) by graph matching techniques.

3.2 Representing Attributes and Interactions

Each attribute \( att \in J^U \cup J^I \) is embedded as a vector \( v \) in \( d \)-dimensional space \( \mathbb{R}^d \). This process can be seen as building a parameter matrix as an embedding lookup table. All the data samples share the same vector embedding \( \mathbf{u}_{att} \) of the same attribute \( att \), but they may have different scalar on the vector due to the potentially different values \( val \)'s. More specifically, for an attribute-value pair \((att, val)\), the corresponding vector, \( \mathbf{u}_{att} \), is computed as \( \mathbf{u}_{att} = \text{val} \cdot \mathbf{u}_{att} \), and such \( \mathbf{u}_{att} \) is called the representation of the attribute-value pair. Initially, the \( \mathbf{u}_{att} \) of each attribute is set as a random vector. In the following, when the context of the attribute \( att \) is clear, we omit the subscript from \( v \) and \( u \) for simplicity.

The co-occurrence of two attributes \( att_1 \) and \( att_2 \) in a data sample is defined as an interaction between \( att_1 \) and \( att_2 \). Such an interaction is modeled by a function \( f(u_1, u_2) : \mathbb{R}^{2d} \rightarrow \mathbb{R}^t \), where \( u_1 \) and \( u_2 \) are the representations of the attribute-value pairs of \( att_1 \) and \( att_2 \) (in the same data sample), and \( t \) is the dimensionality of the output. Since the set of attributes, \( J^U_n \) and \( J^I_n \), in each data sample may be different, the interactions specified in different data samples could be different. And because each attribute may appear (with different values) in multiple data samples, the collaborative information of the interactions in different data samples would further help discover the interactions between attributes that have never co-occurred. Therefore, \( f(\cdot, \cdot) \) actually learns attribute embeddings that capture the collaborative information between the attributes (i.e., similar attributes would have similar embeddings) [25].

Our proposed model GMCF categorizes attribute interactions in a data sample into two types, inner interactions and cross interactions. More specifically, the interactions between user attributes only and between item attributes only are defined as inner interactions. However, on the other hand, those between one user attribute and one item attribute are cross interactions. As we will see shortly in Section 4, our GMCF model deploys different functions, \( f(\cdot, \cdot) \), to model these two kinds of attribute interactions, and respectively uses them for different purposes: (i) user and item characteristic learning, and (ii) recommendation.

3.3 Graph Neural Networks

Consider a graph \( G = (V, E) \), where \( V = \{v_i\}_{1 \leq i \leq k} \) is the set of \( k \) nodes, each of which is represented by a vector representation \( v_i \), and \( E \) is the set of edges which indicate the neighborhood information between nodes: two nodes are neighbors if they are linked by an edge. A Graph Neural Network (GNN) learns the vector representation of each node by message passing, a procedure of aggregating neighborhood information. Specifically, the message passing procedure for node \( i \) first aggregates the vector representations of all its neighbors. Then, it gets the fused representation of node \( i \) by fusing \( v_i \) and the aggregated vector representation. Formally, the fused vector representation of node \( i \), \( v'_i \), through the graph modeling is:

\[
v'_i = f_{fuse}(v_i, \text{Aggregate}_v(v_j)_{j \in N(i)}),
\]

where \( f_{fuse} \) is the fusing function, \( \text{Aggregate}_v(\cdot) \) is an aggregation function that aggregates the neighborhood embeddings into a fixed dimension representation (e.g., element-wise sum) and \( N(i) \) is the set of all the neighbors of node \( i \).
If it is necessary, the graph representation can be computed as the
aggregation of the vector embeddings of all the nodes: \( v_G =
\text{Aggregate}_{GC}(v'_{ij}) \), where \( v_G \) is the graph representation and
\( \text{Aggregate}_{GC}(\cdot) \) is a node aggregation function that is similar to the
one for aggregating nodes’ neighbors.

### 4 OUR APPROACH

In this section, we describe our model in detail. First, we give an
overview of our GMCF model. Then, we unveil the details of each part
of the model. Finally, we discuss the relations of our model to
existing work and applicable situations.

#### 4.1 GMCF Overview

Our proposed model GMCF mainly consists of three modules: (i)
User and Item Graph Construction Module, (ii) Node Matching
based GNN Module, and (iii) Graph Representation Matching Module.
Figure 2 shows an overview of the GMCF model. In particular,
for an input data sample \( x = \{c_{ij} = (\text{att}, \text{val})\}_{i,j} \in \mathcal{SP} \cup \{c'_{ij} =
(\text{att}, \text{val})\}_{i,j} \in \mathcal{SP}' \}, \) each module works as follows.

The User and Item Graph Construction Module, GMCF constructs
a user attribute graph and an item attribute graph respectively based
on the user and item characteristic specified in the data sample \( x \). More specifically, the user attribute graph (resp., item attribute
graph) is a complete graph, where each node corresponds to an
attribute-value pair \((\text{att}, \text{val})\) in the user (resp., item) characteristic
and is represented by the representation \( u = \text{val} \cdot \alpha \) of the pair.

The Node Matching based GNN Module. For each node \( i \) with rep-
resentation \( u_i \) in the user (resp., item) attribute graph, this module
first computes the message passing information \( z_i \) and the node
matching information \( s_i \) with the item (resp., user) attribute graph.
It then fuses \( u_i, z_i \) and \( s_i \) to obtain a fused node representation \( u'_i \)
for the node \( i \). Finally, a graph representation \( v_G \) is obtained by
aggregating the fused node representation \( u'_i \) for all nodes.

The Graph Representation Matching Module. Our GMCF performs
a graph matching between the user and item attribute graph rep-
sentations. The final prediction is obtained from matching result.

Next, we introduce the details of the three modules.

#### 4.2 User & Item Graph Construction

We represent each user and each item as an attribute graph, with
their attributes as nodes and their inner interactions as edges. Specif-
ically, for each data sample \( x \), the user attributes are the nodes of
the user attribute graph. Each node \( i \) that represents an attribute
takes the attribute representation \( u_i^U \) as the node representation.
Therefore, we represent the node set of the user attribute graph as
\( V^U = \{u_i^U\}_{i \in \mathcal{U}} \). Then, each pair of nodes in the graph are con-
nected with an edge to indicate the pairwise interaction between
the two attributes. In summary, each user attribute graph is represen-
ted as \( G^U = (V^U, E^U) \), where \( E^U \) is the edge set that contains
all edges in the graph. Note that since we consider all pairwise
attribute interactions, the user attribute graph is a complete graph.
We perform the same transformation for item attributes and get
the item graph \( G^I = (V^I, E^I) \). In practice, we perform the graph
construction by simply dividing attribute interactions as edges and
matching pairs for different modules, which does not take additional
effort than other explicit pairwise interaction modeling methods
(e.g., AutoInt, Fi-GNN).

#### 4.3 Node Matching based GNN

We propose a node matching based GNN, \( f_G \) that considers both
inner interactions through message passing and cross interactions
through node matching. In this section, we describe how \( f_G \) out-
puts the graph representation through modeling the two types of
interactions. Note that since the modeling on user attribute graphs
and item attribute graphs are symmetric, the notations for \( f_G \) in
the following subsections is generic (i.e., we omit the superscript \( U \)
and \( I \)) that can apply to both user and item attribute graphs.

##### 4.3.1 Neural Interaction based Message Passing

In our model, the message passing method models the inner interactions for character-
istic learning. Inspired by \( [3, 30] \), we use an MLP to model each
inner interactions. Then, we aggregate the interaction modeling
results as the message passing information. Specifically, an MLP
function \( f_{\text{neural}} \in \mathbb{R}^{2\cdot d} \rightarrow \mathbb{R}^d \) takes the embeddings of the two
nodes as input and output the interaction modeling results:

\[
\ z_{ij} = f_{\text{neural}}(u_i, u_j),
\]

where \( z_{ij} \) is the interaction modeling results of the node pair \((i, j)\).

Then, all the interaction modeling results corresponding to each
node will be aggregated as the message passing information. In
GMCF, we use the element-wise sum to aggregate the interaction
modeling results \( z_i = \sum_{j \in \mathcal{N}_i} z_{ij} \), where \( z_i \in \mathbb{R}^d \) is the message
passing results of node \( i \) and \( \mathcal{N}_i \) is a set of node \( i \)’s neighbors.

\( f_{\text{neural}} \) explicitly models the interaction information between
two attributes, which effectively unifies the interaction modeling
for recommendation and the message passing in graph learning.
In addition, inner interactions are used for capturing user (item)
characteristics and are inherently complex. A high inner interaction
result does not mean that the two attributes should be similar
when determining the characteristics of a user (an item), which are
different from cross interaction model results that reveal similarity
(whether will be discussed in the next section). Therefore, a neural method
(e.g., MLP) that non-linearly models the two attributes are desired.

##### 4.3.2 Bi-Interaction based Node Matching

GMCF conducts node matching between two graphs through modeling cross interactions.
Intuitively, we expect that an attribute \( c_{ij}^U \) will have a high matching score with an attribute \( c_{ij}^F \), if \( c_{ij}^F \) shows a high preference on \( c_{ij}^U \). For example, if male users prefer sci-fi movies, the node matching score
of the node pairs \(<\text{male}, \text{sci-fi}>\) should be high. In collaborative fil-
tering, if a user attribute has a high preference for an item attribute,
their embeddings should be similar after training. To achieve this,
we use the Bi-interaction \( [13] \) for node matching, which keeps the
monotonically increasing correlation between interaction modeling
results and the attribute similarities. As a result, if a user attribute
has a high matching score on an item attribute, they have similar
attribute representations. Specifically, the Bi-interaction algorithm
models the attribute interactions as:

\[
 s_{ij} = u_i \odot \bar{u}_j,
\]

where \( u_i \) is the embedding of node \( i \) in one graph, \( \bar{u}_j \) is the embed-
ning of node \( j \) in the other graph, \( s_{ij} \) is the node matching result of
the node pairs from different graphs, and \( \odot \) is the element-wise product.
While performing graph matching, we get the vector representation of each graph as aggregated graph matching results. Specifically, the node fusing function $f_{\text{fuse}}$ considers $[u_i, z_i, s_i]$ as input, and get the fused node representation. Formally, we have $u'_i = f_{\text{fuse}}(u_i, z_i, s_i)$, where $u'_i$ is the fused node representation of node $i$.

We can use any method for $f_{\text{fuse}}$, e.g., element-wise addition or recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best. We use GRU, an effective recurrent neural network. Through testing, we find that the recurrent neural networks perform the best.
We first describe the datasets and the baselines used in our experiments. The rows $\sum$ additionally calculates an attention value for each interaction that can take user attributes and item attributes into account:

5.1.2 Baselines. We compare our model with competitive baselines.

5.1.1 Datasets. We run GMCF and baselines on three datasets, that contain both user and item attributes. Table 1 shows their statistic information. Below are the descriptions of the datasets:

**MovieLens 1M** [12] contains users’ ratings on movies. Each data sample contains a user and a movie with their corresponding attributes. We further collect movies’ other attributes, such as directors and casts from IMDB to enrich the dataset. **Book-crossing** [45] contains users’ implicit and explicit ratings of books. Each data sample contains a user and a book with their corresponding attributes. The reprocessed words in the book titles are also regarded as attributes of the book. **Taobao** [44] is a dataset that collects the log of click on display advertisement displayed on the website of Taobao. Each log contains a user with corresponding attributes such as gender and age level, a displayed advertisement with attributes such as category and brand of the advertised item.

MovieLens 1M and Book-crossing contain explicit ratings. We transfer the explicit ratings to implicit feedback. We regard the ratings greater than 3 as positive ratings for MovieLens 1M and regard all rated explicit ratings as positive ratings for Book-crossing due to its sparsity. Then, we randomly select the same number of negative samples equal to the number of positive samples for each user. To ensure the datasets’ quality, we select the users with more than 10 positive ratings for MovieLens 1M and have more than 20 positive ratings for Book-crossing and Taobao.

| Dataset          | #Data     | #User     | #Item     | #User attr | #Item attr |
|------------------|-----------|-----------|-----------|------------|------------|
| MovieLens 1M     | 1,149,238 | 5,950     | 3,514     | 30         | 6,944      |
| Book-Crossing    | 1,050,834 | 4,873     | 53,168    | 87         | 43,157     |
| Taobao           | 2,599,463 | 4,532     | 371,760   | 36         | 434,254    |

5.2 Overall Performance

We compare the performance of GMCF with the baselines. Table 2 shows the prediction performance of each model. The best results for each dataset are in bold, and the best baseline results are in underline. The rows Improv and $p$-value show the improvement and statistical significance test results (through Wilcoxon signed-rank test) of GMCF and the best baseline results, respectively. From Table 2, we observe that:

- GMCF outperforms all the baselines significantly, with the $p$-value of all metrics rejecting the null hypothesis with a level of
We focus on (i) the effectiveness of modeling inner and cross interactions, which show similar trends with AUC and NDCG@10, respectively. We focus on three questions: 1) what is the effectiveness of node-attribute interactions using different methods; (ii) the comparison of using different algorithms as the fusing function; (iii) the performance variations that use different methods: <inner interaction model, cross interaction model>. To answer the three questions, we run GMCF with different method combinations in each variation. For example, we represent the original GMCF as <MLP, Bi> to indicate an MLP-based message passing and a Bi-interaction-based node matching.

We run GMCF using the combination <MLP, None>, where None indicates that we do not perform node-level matching (i.e., no cross interaction modeling). Then, we run two variations that use the same method to model all interactions: <Bi, Bi>, <MLP, MLP>. Note that the second variation uses the same MLP (i.e., the same neural architecture with shared parameters). Finally, we further run two variations that use different methods: <MLP1, MLP2> (the same architecture with different parameters) and <Bi, MLP>. Figure 3 shows the results of using different variations. We omit the results of Book-crossing due to the space limitation, which shows similar trend with MovieLens 1M (the same as the remaining figures).

For question 1), <MLP, None> gains the worst performance. The reason is that the attribute graphs are inherently complex. Fusing this information only in the last step makes it difficult to match users' preferences on items through attributes. The node matching method provides the explicit attribute communication through the cross interactions between the two graphs, which results in a fine-grained analysis for a more accurate preference matching.

For question 2), the combinations (<Bi, Bi> and <MLP, MLP>) gain relatively worse results than the original setting (<MLP, Bi>). This is because the inner interactions and cross interactions are inherently different that should be modeled differently that fit their roles. The inner interactions are used for characteristic learning and do not indicate the similarity information (i.e., two attributes having strong interaction does not mean that they are similar). Meanwhile, in collaborative filtering, two attributes are expected to show similarity information if their cross interaction is strong (high preference). The Bi-interaction algorithm fits for the requirement of cross interactions. For question 3), even using different methods, <MLP1, MLP2> and <Bi, MLP> still gain worse results than the original GMCF. Although an MLP seems more powerful than the Bi-interaction algorithm, MLP cannot to effectively learn the measurement for the cross interactions. The method combinations in each variation. For example, we represent the original GMCF as <MLP, Bi> to indicate an MLP-based message passing and a Bi-interaction-based node matching. For question 1), <MLP, None> gains the worst performance. The reason is that the attribute graphs are inherently complex. Fusing this information only in the last step makes it difficult to match users' preferences on items through attributes. The node matching method provides the explicit attribute communication through the cross interactions between the two graphs, which results in a fine-grained analysis for a more accurate preference matching.

For question 2), the combinations (<Bi, Bi> and <MLP, MLP>) gain relatively worse results than the original setting (<MLP, Bi>). This is because the inner interactions and cross interactions are inherently different that should be modeled differently that fit their roles. The inner interactions are used for characteristic learning and do not indicate the similarity information (i.e., two attributes having strong interaction does not mean that they are similar). Meanwhile, in collaborative filtering, two attributes are expected to show similarity information if their cross interaction is strong (high preference). The Bi-interaction algorithm fits for the requirement of cross interactions. For question 3), even using different methods, <MLP1, MLP2> and <Bi, MLP> still gain worse results than the original GMCF. Although an MLP seems more powerful than the Bi-interaction algorithm, MLP cannot to effectively learn the measurement for the cross interactions. The method combinations in each variation. For example, we represent the original GMCF as <MLP, Bi> to indicate an MLP-based message passing and a Bi-interaction-based node matching.

5.3 Study of Neural Graph Matching

In this section, We evaluate the neural graph matching of GMCF. We focus on (i) the effectiveness of modeling inner and cross interactions using different methods; (ii) the comparison of using different algorithms as the fusing function; (iii) the performance of GMCF when the user and item attributes are not available. Due to space limitation, we omit the results of Logloss and NDCG@5, which show similar trends with AUC and NDCG@10, respectively.

5.3.1 Evaluation of Inner and Cross Interaction modeling. We evaluate the inner and cross interaction modeling methods. Specifically, we focus on three questions: 1) what is the effectiveness of node-level graph matching (i.e., modeling cross interactions)? 2) whether we should use different methods to model the two types of interactions? 3) whether sophisticated nonlinear algorithms are always suitable for modeling the two types of interactions?

To answer the three questions, we run GMCF with different interaction modeling methods. For clear demonstration, we use the pair <inner interaction model, cross interaction model> to indicate the method combinations in each variation. For example, we represent the original GMCF as <MLP, Bi> to indicate an MLP-based message passing and a Bi-interaction-based node matching.

We run GMCF using the combination <MLP, None>, where None indicates that we do not perform node-level matching (i.e., no cross interaction modeling). Then, we run two variations that use the same method to model all interactions: <Bi, Bi>, <MLP, MLP>. Note that the second variation uses the same MLP (i.e., the same neural architecture with shared parameters). Finally, we further run two variations that use different methods: <MLP1, MLP2> (the same architecture with different parameters) and <Bi, MLP>. Figure 3 shows the results of using different variations. We omit the results of Book-crossing due to the space limitation, which shows similar trend with MovieLens 1M (the same as the remaining figures).

For question 1), <MLP, None> gains the worst performance. The reason is that the attribute graphs are inherently complex. Fusing this information only in the last step makes it difficult to match users’ preferences on items through attributes. The node matching method provides the explicit attribute communication through the cross interactions between the two graphs, which results in a fine-grained analysis for a more accurate preference matching.

For question 2), the combinations (<Bi, Bi> and <MLP, MLP>) gain relatively worse results than the original setting (<MLP, Bi>). This is because the inner interactions and cross interactions are inherently different that should be modeled differently that fit their roles. The inner interactions are used for characteristic learning and do not indicate the similarity information (i.e., two attributes having strong interaction does not mean that they are similar). Meanwhile, in collaborative filtering, two attributes are expected to show similarity information if their cross interaction is strong (high preference). The Bi-interaction algorithm fits for the requirement of cross interactions. For question 3), even using different methods, <MLP1, MLP2> and <Bi, MLP> still gain worse results than the original GMCF. Although an MLP seems more powerful than the Bi-interaction algorithm, MLP cannot to effectively learn the measurement for the cross interactions. The method combinations in each variation. For example, we represent the original GMCF as <MLP, Bi> to indicate an MLP-based message passing and a Bi-interaction-based node matching.
results to get the fused node representation. We evaluate the effectiveness of using different algorithms as the fusion algorithm. Except the GRU, we further use the element-wise sum on the three vectors as the fused node representation (SUM) and the MLP that concatenates the vectors as input and outputs the fused node representation (MLP). We use the MLP that has one hidden layer, with the number of neurons being 4d.

Table 3 shows the experimental results of using the three algorithms as the fusioning method. From the table, we can see that using GRU results in the best performance on all datasets. It shows the ability of GRU to effectively aggregate the message passing information and the node matching information into the fused node representation for accurate predictions. Summing up the results (SUM) gains the worst results in most situations, which indicates that the message passing and node matching information are complex. Powerful algorithms are required to fuse them.

5.3.3 Evaluation of the User and Item Attributes. In some situations, user or item attributes are not available. We evaluate how GMCF performs in these situations. GMCF is a flexible framework that is applicable when the user or item attributes are not available. In these situations, the user (or item) ID graphs are reduced into a single node indicating user (or the item) ID. We run GMCF and the best baseline L₀-SIGN on the situations that user or item attributes are not available. Figure 4 shows the results. Specifically, in the x-axis, “None” indicates neither user or item attributes are available, “User” indicates only user attributes are available, “Item” indicates only item attributes are available, and “Both” indicates both user and item attributes are available.

From the figure, we observe that: 1) Both models perform better when user and item attributes become available (from left to right).

It shows that both user and item attributes are useful for performance gain. Although the performance gain from user attributes (User) seems not that significant compared to item attributes (Item), the user attributes provide useful information for potential explanations of the prediction results, which will be discussed in Section 5.5. 2) The two models have similar performance when no attributes are available (None). However, GMCF performs much better when user and item attributes are available. The performance gain compared to L₀-SIGN when the user and item attributes are available resulted from the generated graph matching structure, which captures more useful structural information for accurate predictions.

5.4 Parameter study

In this section, we evaluate GMCF with different hyper-parameter settings. Specifically, we evaluate the node representations’ dimension and the depth of the MLP used in our model.

Figure 6 shows the results of our model and best-performed baselines on different node representation dimensions (d). From the figure, we observe that GMCF constantly outperforms baselines on different node representation dimensions, which shows the robustness of our model in delivering superior prediction accuracy. Then, when the dimension is 64, our model and most of the baselines gain the best performance. This indicates that a higher dimension does not necessarily result in better prediction accuracy. This is because that a larger dimension means more parameters to fit, and thus is prone to cause the overfitting problem.

Then, we evaluate how the different number of hidden layers in the MLP affects our model’s performance. In our model, we use an MLP with 1 hidden layer to analyze the inner interactions while message passing. Now we evaluate our model with different number of layers. Specifically, we run our model with 0, 1, 2, 3, 4 hidden layers respectively (GMCF-0,...,GMCF-4). Note that GMCF-0 means that the MLP only performs a linear transformation from the node representations to the interaction modeling results. We use the same number of units (4d) for each hidden layer.

Table 4 shows the results of using different MLP layers in GMCF. We can see that using the MLP with 0 hidden layer (GMCF-0) gains much worse results than other settings. It shows that a powerful non-linear algorithm helps extract useful information from inner
interactions as a linear transformation. This result is consistent with the results in section 5.3.1. When the number of the hidden layer is 1, GMCF gains the best performance. This illustrates that deeper MLP does not necessarily increase the performance due to the overfitting \cite{11, 13}, and one hidden layer is enough in our models to analyze the inner interactions.

5.5 Case Study

In this section, we conduct case studies to evaluate whether our model learns collaborative information between attributes and whether the attributes show semantic meaning that provide potential explanations of the predictions. Specifically, we use the learned attribute embeddings from the MovieLens 1M dataset. We first calculate the embeddings’ cosine similarities between user age groups (e.g., 1-18, 19-24) and between movie genres. Then, we calculate the node matching results between the two types of attributes. Figure 5 shows the similarity results (the left and central figures) and the node matching results (the right figure). Note that the darker the color, the higher the similarity or the node matching value. The age labels indicate the age groups (e.g., age_18 means age 18-24).

From the left and central figures, we can see that similar user attributes and similar item attributes have similar embeddings after training. For example, in the left figure, the age group attributes are clearly divided into two groups at the age of 45, which means that the similar age attributes are grouped. It indicates that the same age group users may have a similar preference for movies in terms of age. In the central figure, Fantasy, Sci-fi, and Mystery have similar embeddings in the latent space. Semantically, these genres are similar. Similar results are also shown for Crime, Horror, and Thriller. The above observations show that similar attributes are successfully learned to have similar embeddings in GMCF.

The right figure shows the node matching between age groups and genres. We observe that different age groups have different preferences for movie genres. For example, younger users (under 45) have higher node matching values (preference) on fantasy, sci-fi, and mystery movies more than older users (over 45). In comparison, the older users seem to like musical and Documentary (Doc) movies more. These observations show our model’s ability to provide potential explanations about the prediction results at attribute level, e.g., we recommend Interstellar (a sci-fi movie) to a 16-year-old user because younger people have a high chance to prefer a sci-fi movie.

6 CONCLUSION

User and item attribute interactions provide useful information in recommender systems for accurate predictions. While existing work treat all the attribute interactions equally, in this work, we identify two types of attribute interactions: inner interactions and cross interactions. We propose a neural Graph Matching based Collaborative Filtering (GMCF) model. The GMCF models and exploits the two types of interactions for different purpose in a graph matching structure. Specifically, GMCF explicitly performs user and item characteristic learning with the inner interactions, and performs preference matching for recommendation based on the cross interactions. Experimental results show that our GMCF model is effective and outperforms all state-of-the-art baselines in terms of accuracy on three widely used datasets. In future work, we will consider higher-order interactions in our model architecture, which may contain edge matching and sub-graph exploration.

ACKNOWLEDGMENTS

This work is supported by the China Scholarship Council (CSC). In this research, Junhao Gan was in part supported by Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) DE190101118.
