IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 3, 2021, accepted March 13, 2021, date of publication March 29, 2021, date of current version April 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3068570

A Service Recommendation Algorithm Based on Knowledge Graph and Collaborative Filtering

BO JIANG[1], JUNCHEN YANG[1], YANBIN QIN[1], TIAN WANG[1], MUCHOU WANG[2], AND WEIFENG PAN[1]

[1]School of Computer Science and Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, China
[2]Wenzhou University Library, Wenzhou University, Wenzhou 325000, China

Corresponding authors: Bo Jiang (nancybjiang@zjgsu.edu.cn), Junchen Yang (19020100041@pop.zjgsu.edu.cn), and Muchou Wang (wzuwmc@163.com)

This work was supported in part by the Natural Science Foundation of Zhejiang Province under Grant LY21F020002, and in part by the Key Research and Development Program Project of Zhejiang Province under Grant 2019C01004.

ABSTRACT With the rapid development of the Internet, the number of Web APIs is increasing. How to recommend accurate and appropriate Web APIs to mashups has become a focus and difficulty in the field of service computing. The existing methods are mainly based on collaborative filtering technology, but these methods have problems such as the data sparsity and cold start, which leads to poor recommendation effects. This paper proposes a service recommendation model based on knowledge graph and collaborative filtering. In this model, the knowledge graph connects the APIs and mashups related information to mine the potential relations between mashups and APIs, hence reducing the impact of data sparsity. All the API entities in the service knowledge graph are embedded into the low-dimensional space through the representation learning algorithm, then the distances between the API vectors are calculated to recommend the related APIs. In addition, in order to solve the cold-start problem of recommending APIs to target mashups that have no APIs usage, the similarities of functional sets extracted from mashups are calculated to recommend APIs for target mashups. At the same time, the model obtains the Mashup-API usage record, using the technology of collaborative filtering to recommend appropriate APIs to target mashups. Finally, the similarities of the above recommended APIs are normalized and sorted to form the final recommendation result. The experimental results show that our proposed model significantly improves the accuracy of service recommendation.

INDEX TERMS Web API recommendation, knowledge graph, representation learning, collaborative filtering.

I. INTRODUCTION

Web APIs are applications that do not depend on the specific operating system. Developers can complete their requirements by using specific Web APIs [1]. With the rapid development of the Internet, the number of Web APIs has increased rapidly in recent years, which has brought great trouble to developers [2]. In this case, mashup technology becomes popular [3]. Mashups can combine Web APIs effectively according to existing web resources [4]. In addition, mashups integrate different functions together, which has the advantages of fast development and strong scalability. For example, the *best shop guide* is a shopping-themed mashup, which consists of three Web APIs: *shopping.com*, *comamazon product advertising*, and *ebay*. This mashup implements the function of shopping through the combination of three APIs. It can be seen that the emergence and development of mashups not only solve the shortcomings of a single Web API function, but also make codes related to APIs reusable, which greatly reduces the workload of developers. Therefore, mashup technology has been applied in many fields, and has been combined with cloud computing, Internet of things [5] and other advanced technologies in recent years.

However, the number of Web APIs and mashups is too large. For example, the number of Web APIs published on ProgrammableWeb (PWeb for short) has exceeded 18000, while that of mashups has exceeded 7700, and the number is growing every year. How to find suitable Web APIs for a mashup from so many Web APIs has become a difficult problem in the Service Computing field. There are two ways

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Marozzo.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

to find required APIs: recommendation and search. In most cases, developers cannot clearly express the specific information of the required API, so it is difficult to obtain the required API through service search. On the contrary, the service recommendation system can personally recommend suitable APIs based on fuzzy requirements and historical usage records, which makes service recommendation gradually become a research hotspot. There are many researches on service recommendation. Among them, the most widely used method is collaborative filtering [6], including mashup-based collaborative filtering [7], service recommendation combining collaborative filtering and text content [8]. Based on collaborative filtering, matrix decomposition [9], [10] is usually used to improve the quality of recommendation through historical usage records between mashups and APIs. In addition, researchers have also tried to combine NLP [11], knowledge graph [12], neural network [13] and other technologies with collaborative filtering.

Since only a few of the APIs have been used, the service recommendation based on collaborative filtering has the problem of sparsity. In order to alleviate sparsity, [14] proposed a service recommendation model based on knowledge graph and representation learning. The model analyzes the relations between APIs to construct a knowledge graph about users and services. Then representation learning is used to embed the entities in the knowledge graph into the low-dimensional space. Finally, the collaborative filtering is used to recommend suitable APIs for users. However, this method relies on the description information of APIs. In reality, the information of the API is often inaccurate and incomplete, which seriously affects the recommendation effect. Inspired by the above model, we applied some of the ideas of the model to recommend APIs for mashups. The development of a mashup is often the work of a large team, and it is difficult to accurately recommend APIs based on the usage records of a single user. Therefore, we added the relevant information of mashups to the knowledge graph to make the service knowledge graph more complete. In addition to recommending APIs through knowledge graph and representation learning, we also recommended APIs based on the similarities between the functions of mashup and Mashup-API invocation matrices. In this way, we solved the problem of relying only on API information through the hybrid recommendation.

In view of the shortcomings of the existing recommendation methods, we propose a service recommendation model based on knowledge map and collaborative filtering. In this model, in order to reduce the impact of data sparsity, our model embeds information related to APIs and mashups into the knowledge graph to explore the potential relations between APIs and mashups. Then our model uses the representation learning algorithm to convert API entities into low-dimensional vectors to calculate the distances between API vectors, and uses collaborative filtering to recommend suitable APIs. In addition, the model recommends APIs related to the target mashup by calculating the similarities between the extracted functional sets of all mashups, which solve the cold start problem. At the same time, the historical record of mashups using APIs is used to recommend APIs for target mashups to enrich the recommended result. Finally, the similarities of above recommended APIs are normalized to form the final recommendation list.

The main contributions of this paper are as follows:

- We propose a service recommendation model (KGCF-SR) based on knowledge graph and collaborative filtering, which can effectively alleviate sparsity and solve the problem of cold start.
- We embed mashups and APIs into neo4j graph database to build a complete service knowledge graph. Compared with the existing service knowledge graphs, the service knowledge graph we constructed contains the complete information of APIs and mashups, and fully considers the diverse relations between entities. With this knowledge graph, the potential relations between mashups and APIs can be explored.
- We have collected more than 18000 web APIs and more than 7700 mashups on PWeb. Experiments on this dataset showed that our proposed method is significantly better than existing related methods.

The rest of the paper is organized as follows. Section 2 discusses some representative related work. Section 3 introduces the service recommendation model of this paper in detail. Section 4 evaluates our model through a comprehensive experiment. In section 5, we make a summary of our work; meanwhile we put forward the shortcomings of our method and prospects for the future.

II. RELATED WORK

A. API RECOMMENDATION FOR MASHUPS

In this section, we discuss some representative research work on the existing methods that recommend APIs for mashups.

In the existing related researches, the most widely used method is collaborative filtering technology, which provides recommendations through similarities of mashups or similarities of APIs. Cao *et al.* [7] proposed a collaborative filtering method based on the semantic similarities of mashups. Depending on the semantic similarities between mashups, they recommended the APIs used by mashups with similar description information to the target mashup. However, this method has high requirements for the description information of mashups. When the description information is incomplete or inaccurate, it will greatly affect the final recommendation result. In addition, some researchers used the technology of matrix factorization [9], and combined the relations between APIs and mashups to make the recommended APIs more diverse. However, this method has the problem of data sparsity. In order to alleviate the sparsity problem of collaborative filtering, many scholars have proposed effective methods. Yu *et al.* [15] proposed a two-side Cross Domain Collaborate Filtering model. Firstly, the intrinsic features of users and items are extracted from the auxiliary domain. Then the problem of recommending items to users is changed into a

classification problem. Finally, SVM [16] is used to solve the classification problem. However, the extracted intrinsic features are usually not related to the domain, which leads to poor recommendation effects. Considering this shortcoming, Yu *et al.* [17] improved the previous model and proposed a cross-domain collaborative filtering algorithm with expanding user and item features via the latent factor space of auxiliary domains. Based on the previous model, Funk-SVD is used to extract additional user and item features from the auxiliary domain to expand the two-dimensional location feature vector. Finally, the model uses the C4.5 decision tree algorithm to predict missing ratings to achieve better experimental results.

Xiong *et al.* [8] proposed a hybrid service recommendation method based on deep learning. They first obtained the historical usage records between mashups and APIs. Then they put the historical usage records and their functions into the neural network. In this way, researchers explored the potential relations between mashups and APIs. However, this approach relies on the historical usage records between mashups and APIs. Unfortunately, historical usage data is very sparse. Most mashups use no more than three APIs, and only 10% of the existing APIs are used by mashups, which makes it difficult to recommend appropriate APIs when the scale of data is very large.

B. KNOWLEDGE GRAPH

Knowledge graph is a directed graph composed of entities and relationships between entities [18]. It can store large-scale data and describe the relations between data. At present, there are many knowledge graph databases, the most well-known ones are DBpedia [19] based on Wikipedia, and the huge knowledge resource library YAGO [20].

In recent years, knowledge graph has gradually become a hot spot in academic research, especially in the field of recommendation algorithms [21]. Many researchers applied knowledge graph to recommend APIs for mashups [14], which has a good effect in solving the problem of sparsity.

Wang *et al.* [12] proposed an API recommendation method by using knowledge graph and random walking. They first obtained the information of APIs and mashups through the knowledge graph. Then they mined the potential relations between the APIs and the target mashup through the random walking algorithm. Finally, they recommended appropriate APIs for the target mashup through potential relationships. Although this method can alleviate sparseness, it has high requirements for description information of mashups. Once the developers describe the requirements of mashups inaccurately, it will affect the final experimental results. In addition, the quality of screening features directly affects the final results.

C. KNOWLEDGE REPRESENTATION LEARNING

In the field of knowledge, feature engineering [22] is often used to extract features. However, feature engineering needs to process data manually, which makes the workload become huge. Considering the shortcomings of feature engineering, Mikolov *et al.* [23] proposed the word2vec representation learning model, which is the earliest research of knowledge representation learning.

As more and more scholars pay attention to knowledge representation learning, many models of representation learning appear. Nickel *et al.* [24] proposed the method of e factorization of a three-way tensor, and Socher *et al.* [25] proposed a single-layer neural network-based representation learning model. Among these models, the translation model [26] has received attention because of its high accuracy.

With the development of knowledge graph technology, the application of the translation model in knowledge graph has become extensive [26]–[29]. These methods transform the relations and entities in the knowledge graph into vectors, and the representative one is the TransE model [26]. Although the TransE method has a good effect on a single relation vector, when the entities have one-to-many or many-to-many relations, the expected effect cannot be achieved. Therefore, in 2014, Wang *et al.* [27] proposed TransH, which can handle one-to-many or many-to-many relations in the knowledge graph. Since then, many scholars improved models on the basis of TransE, and successively proposed TransA [28], TransR [29], TransD [30].

In recent years, the method of combining representation learning and knowledge graph has been used in service recommendation. Cao *et al.* [14] proposed an API recommendation algorithm based on knowledge graph and representation learning. They first determined the competitive or cooperative relations between APIs. Then they embedded API entities and user entities into the low-dimensional space by the representation learning algorithm. Finally, they recommended APIs to users through the distances between API entities and user entities. But this recommendation model is dependent on the quality of the information in the API description. Generally, if the quality of the API description is not high, then the triples extracted from the knowledge graph will be inaccurate, which seriously affects the effectiveness of a specific recommendation approach.

III. THE PROPOSED METHOD: KGCF-SR

Considering the problem of data sparsity and cold start in existing service recommendation models, this paper proposes a service recommendation model based on knowledge graph and collaborative filtering. In this model, in order to explore the hidden relations between APIs and mashups, the related information of APIs and mashups is transformed into entities to construct the service knowledge graph. Then the TransH algorithm is used to embed the APIs into the low-dimensional space to calculate the similarities between APIs. The K neighbor APIs of target APIs are selected as the recommended set, which is defined as AS1. For the purpose of solving the cold start problem, the model calculates the similarities between the extracted functional sets of all mashups, and then selects K neighbor mashups of the target mashup. The APIs used by these neighbor mashups form a recommended set, which

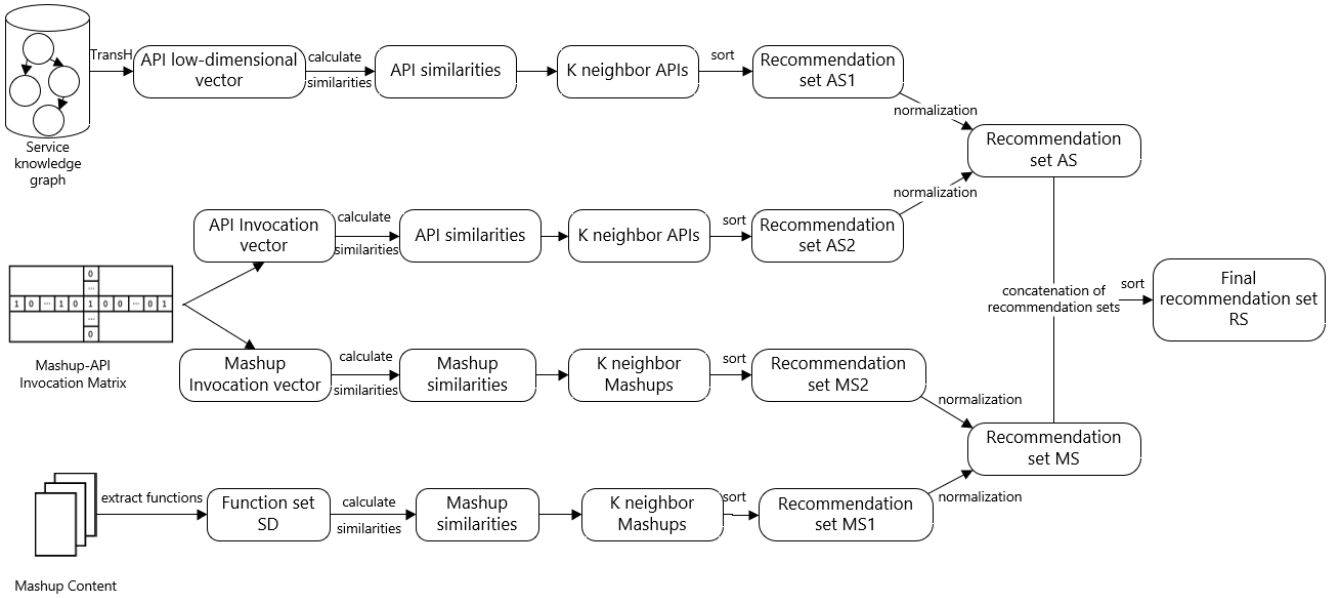


FIGURE 1. The main steps of the proposed method.

is defined as (MS1). In addition, the similarities between APIs and the similarities between mashups are calculated according to the record of mashups using APIs, and then the collaborative filtering is used to obtain the API-based recommendation set (AS2) and the Mashup-based recommendation set (MS2). After the four recommendation lists are obtained, the similarities of the recommended APIs in (AS1) and (AS2) are normalized and the two sets are merged into the API-based recommendation list AS. Similarly, (MS1) and (MS2) are merged into the Mashup-based recommendation set (MS). Finally, the (AS) and (MS) are combined and sorted to form the final recommendation list RS.

The specific steps of the proposed method are shown in Figure 1.

A. SERVICE KNOWLEDGE GRAPH CONSTRUCTION

At present, the historical data of mashups using APIs is very sparse. Among nearly 20000 APIs, more than 90% of mashups use less than five APIs, which makes it difficult to obtain the relations between the target mashups and a large number of APIs. The knowledge graph is a directed graph, which can connect information related to entities with complex relations. So that knowledge graph can mine the potential relations between entities. Therefore, our model constructs a service knowledge graph to mine the potential relations between the target mashups and a large number of APIs.

In order to build a service knowledge graph, we first obtained over 18,000 APIs and 7,732 mashups as service entities from PWeb, and embedded them into the knowledge graph. Then we defined the relations between entities. The relation between API and Mashup was ''used'', the relation between API and Category was ''belong_to'', and the relation between Tag and API was ''tag''. In addition, the relation between Mashup and Category is ''belong_to'', and the relation between Mashup and Tag is ''Tag''.

When the functions between the two APIs are very similar, the two APIs are in competition. In order to explore the potential relations between APIs, we calculated the similarities between APIs by extracting functions. Based on the functional similarities, we set the 20% APIs that were closest to each API as the API's competing APIs.

The relations in the knowledge graph are shown in Figure 2.

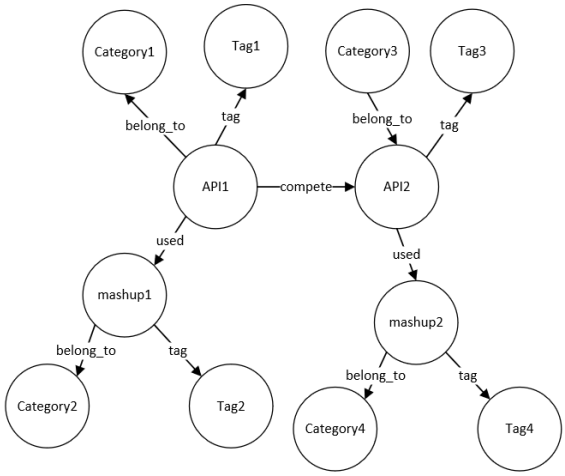


FIGURE 2. Relations in the knowledge graph.

Neo4J [31] is an open source graph database, which can embed entities and relations into graph databases by Cypher language. We embedded the entities and relations into the

Neo4J graph database to construct a knowledge graph about services.

B. EMBEDDING SERVICE ENTITIES INTO LOW DIMENSIONAL SPACE

After constructing the service knowledge graph, although the API entities are connected with their related attributes and entities, the distance between the APIs cannot be calculated, so we need to embed the APIs into the vector space. In order to obtain the vectors of APIs from the knowledge graph, the representation learning algorithm is used to embed API entities into low dimensional space.

TransE is a classical algorithm in representation learning algorithms. It obtains the low-dimensional vector of each API by the triples in the knowledge graph. In the proposed model, the triples (h, r, t) extracted from the service knowledge graph form the training set M. h and t belong to the set E which is the set of entities. And r belongs to the set R which is the set of relations. The vectors of h, t, r are defined as v_h, v_r, v_t. The purpose of the TransE algorithm is to satisfy that the sum of v_h and v_r is approximately equal to v_t, as shown in Figure 3.

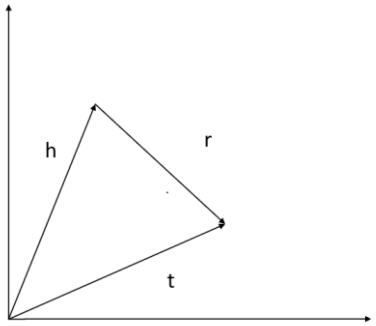


FIGURE 3. Representation of the triple vector in the TransE.

v_h, v_r and v_t satisfy Eq. (1):

$$v_h + v_r \approx v_t \tag{1}$$

The TransE algorithm constructs an incorrect set M', which is composed of the correct triples replaced by an incorrect head entity or an incorrect tail entity. Its construction method is as follows:

$$M' = \{(h, r, t')|t' \in E\} \cup \{(h', r, t)|h' \in E\} \tag{2}$$

The loss function on the training set is

$$L = \sum_{(h,r,t)\in M} \sum_{(h',r,t')\in M'} [\eta + d(v_h + v_r, v_t) - d(v_{h'} + v_r, v_{t'})]_+ \tag{3}$$

where η is a boundary parameter whose value is greater than zero. $[X]_+$ represents the positive part of x.

In order to make the entities and relations in the existing triples satisfy Eq. (1), the stochastic gradient descent algorithm is usually used to optimize the method.

Although the TransE algorithm has a good effect when the relation between the entities in the knowledge graph is one-to-one, the entities cannot be well converted into low-dimensional vectors when the relations between the entities is complex. However, the relations between entities in the service knowledge graph are all one-to-many or many-to-many. For example, there are more than 30 competing APIs with the API named *Yahoo Travel*, and *Yahoo Travel* is used by 7 mashups.

Considering the complex relations between entities in the service knowledge graph, the TransH algorithm is used to embed API entities into the low-dimensional space. TransH is an improved representation learning algorithm based on TransE. It solves the problem that TransE can only work when the relations in the knowledge graph is one-to-one. The vector representation of the TransH algorithm is shown in Figure 4.

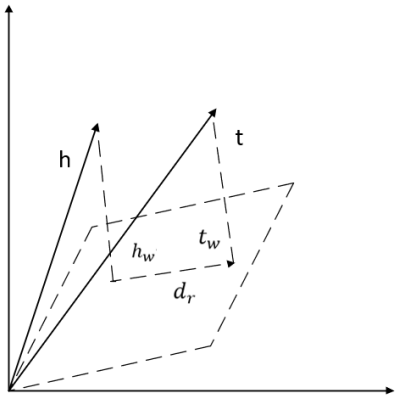


FIGURE 4. Vector representation in the TransH.

For a triple (h, r, t) in the knowledge graph, its corresponding vectors are v_h, v_r, v_t. In the TransH algorithm, each relation r will have a hyperplane, and the normal vector of the hyperplane is defined as w_r. The v_h and v_t are respectively multiplied by the normal vector w_r to map the two vectors to the hyperplane to obtain v_{hw} and v_{tw}. The formulas of v_{hw} and v_{tw} are as follows:

$$v_{tw} = v_t - w_r^T v_t w_r \tag{4}$$

$$v_{tw} = v_t - w_r^T v_t w_r \tag{5}$$

The score function is defined as:

$$d(v_{hw} + v_r, v_{tw}) = \left\| v_h - w_r^T v_h w_r + d_r - v_t + w_r^T v_t w_r \right\|_2^2 \tag{6}$$

In order to distinguish correct triples from wrong triples, we use negative sampling to maximize the scores of wrong triples and minimize the scores of correct triples. The loss function sets some constraints on the basis of TransE:

$$L = \sum_{(h,r,t)\in M} \sum_{(h',r,t')\in M'} [\eta + d(v_h + v_r, v_t) - d(v_{h'} + v_r, v_{t'})]_+ + C\left\{ \sum_{e\in E} [||e||_2^2 - 1]_+ + \sum_{r\in R} \left[\frac{(w_r^T d_r)^2}{||d_r||_2^2} \right]_+ \right\} \tag{7}$$

where C is a weight. The second term is the maximum of the difference between the square of the lengths of all entity vectors and 1, which restricts the length of the entity to be no more than 1. The accumulation in the third term constrains that the spaces represented by different relations are uncorrelated.

C. SIMILARITY CALCULATION

The model proposed in this paper uses the collaborative filtering algorithm to recommend related APIs for target mashups. In order to obtain the neighbor mashups of the target mashup or the neighbor APIs of the target API, a variety of similarities need to be calculated, including the similarities between API entities in the low-dimensional space, the similarities between the invocation matrices of mashups or APIs, and Functional similarities between mashups.

1) CALCULATING THE SIMILARITIES OF API ENTITIES IN THE LOW-DIMENSIONAL SPACE

After embedding the entities in the service knowledge graph into the low-dimensional space, we can get the vectors of all API entities. Then the similarities between APIs can be calculated by cosine similarity. The cosine similarities between API vectors are defined as:

$$Sim(API, API') = \frac{\sum_{i=1}^{d} (API_i \times API_i')}{\sqrt{\sum_{i=1}^{d} API_i^2} \times \sqrt{\sum_{i=1}^{d} API_i'^2}} \quad (8)$$

where d is the dimension of the API vector. The dimension of vector can be set in the TransH algorithm. In this model, the dimension of API vector is 100.

2) CALCULATING THE SIMILARITIES BETWEEN INVOCATION MATRICES

We obtained a historical record of mashups using APIs. From the historical record, a total of 1569 APIs are used by 7732 mashups. Based on the record, mashup-based invocation matrices and API-based invocation matrices are constructed, and then the similarities of the two types of matrices are calculated respectively.

According to records of Mashup-API invocation, matrices of mashups can be constructed. The construction method is as follows:

$$Mashup_i = (API_1, API_2, \ldots, API_j, \ldots, API_{1569}) \quad (9)$$

The value of API_j depends on whether API_j has been used by $Mashup_i$. If it has been used, API_j is 1, otherwise it is 0. Its value is shown in Eq. (10):

$$API_j = \begin{cases} 0 & if\ API_j\ is\ not\ used\ by\ Mashup_i \\ 1 & if\ API_j\ is\ used\ by\ Mashup_i \end{cases} \quad (10)$$

After obtaining the matrices of mashups, the Jaccard similarity coefficient is used to calculate the similarities between the matrices. The similarities are calculated as follows:

$$Jaccard(x, y) = \frac{Mashup_x \cap Mashup_y}{Mashup_x \cup Mashup_y} \quad (11)$$

where $Mashup_x \cap Mashup_y$ represents the number of APIs used by two Mashups at the same time. $Mashup_x \cup Mashup_y$ means the number of APIs used by $Mashup_x$ or $Mashup_y$.

Similarly, the matrices of APIs are constructed as follows:

$$API_i = (Mashup_1, Mashup_2, \ldots, Mashup_j, \ldots, Mashup_{7732}) \quad (12)$$

When API_i has been used by $Mashup_j$, the value of $Mashup_j$ is 1. Otherwise, the value of $Mashup_j$ is 0. Its value is as shown in Eq. (13)

$$Mashup_j = \begin{cases} 0 & if\ API_i\ is\ not\ used\ by\ Mashup_j \\ 1 & if\ API_i\ is\ used\ by\ Mashup_j \end{cases} \quad (13)$$

The Jaccard similarity coefficient is used to calculate the similarities between matrices of APIs. The formula for calculating similarity is as follows:

$$Jaccard(x, y) = \frac{API_x \cap API_y}{API_x \cup API_y} \quad (14)$$

where $API_x \cap API_y$ means the number of mashups that have used both API_x and API_y, and $API_x \cup API_y$ means the number of mashups that have used API_x or API_y.

3) CALCULATING THE SIMILARITIES BETWEEN MASHUPS BASED ON FUNCTIONS

In most related researches, the similarities between mashups are represented by the similarities between description texts of mashups. However, the description texts of mashups are very short, which makes the semantic similarities between the texts unable to accurately represent the similarities between the mashups.

In actual development, when the functions in the description text of two mashups are similar, the APIs they use are also similar. Therefore, in the model proposed in this paper, the functions in the description texts of mashups are extracted to form function sets. Based on the functional sets of mashups, the similarities between mashups are calculated.

Stanford Parser is a popular tool of natural language processing. It can identify the parts of speech of words and generate the corresponding sets Stanford dependence (*SD*) by analyzing the grammatical relationships of words [32]. An *SD* is a two-tuple expressed as *sdtype* (*gov*, *dep*). Where *gov* is the dominant word, including verbs, prepositions, etc., while *dep* is a subsidiary word, including nouns, noun phrases, etc. The *sdtype* is the type of *SD* between two words. In this paper, Stanford Parser is used to extract the functions of mashups' description text. For example, the description information of the mashup whose name is *search video* is *Search for videos through Google*. The *SD* method of direct conversion is used to convert the description information of this mashup into a two-tuple *dobj* (*search for*, *video*), which is the extracted function of the mashup. Since the number of functions of different mashups is different, the Jaccard similarity coefficient is used to calculate the similarities between the functional sets

of mashups. The formula is as follows:

$$Jaccard(P, Q) = \frac{|P \cap Q|}{|P \cup Q|} \tag{15}$$

where P and Q are function sets with different lengths. The similarities between mashups are calculated as shown in Eq. (16):

$$S_m(m_1, m_2) = \frac{\sum_{i=0}^{l} S_n(f_{1i}, f_{2i})}{k} \tag{16}$$

where m_1 and m_2 are two mashups. f_{1i} is the ith function in m_1, and f_{2i} is the ith function in m_2. k is the larger number of functions in the mashup between m_1 and m_2. l is the number of functions of the mashup with fewer functions. $S_n(f_{1i}, f_{2i})$ is the similarity between each function in m_1 and each function in m_2. The formula for calculating the similarities of functions is as follows:

$$S_n(f_1, f_2) = w_1 \times S_{word}(V_1, V_2) + w_2 \times \frac{\sum_{i=1}^{l} S_{word}(N_{1i}, N_{2i})}{k} \tag{17}$$

where V_1 is the verb contained in function f_1, and V_2 is the verb contained in function f_2. N_{1i} is the ith noun in function f_1, and N_{2i} is the ith noun in function f_2. $S_{\text{word}}(V_1, V_2)$ is the similarity between V_1 and V_2, and $S_{\text{word}}(N_{1i}, N_{2i})$ is the similarity between N_{1i} and N_{2i}. w_1 is the proportion of verb similarity, w_2 is the proportion of noun similarity, $w_1 + w_2 = 1$.

In this model, the WordNet [33] is used to calculate the similarities of words. The formula of $S_{word}(d_1, d_2)$ is as follows:

$$S_{word}(d_1, d_2) = \frac{2 \times E(F(d_1) \cap F(d_2))}{E(F(d_1)) + E(E(d_2))} \tag{18}$$

where $F(D)$ is the feature set of word D, while $E(D)$ is the number of information of functional set D. The formula of $E(D)$ is as Eq. (19):

$$E(D) = -\sum_{t \in D} \log P(t) \tag{19}$$

where $P(t)$ represents the probability of feature t. When there is no intersection between the feature set of d_1 and the feature set of d_2, $S_{\text{word}}(d_1, d_2)$ is 0. When the feature set of d_1 and the feature set of d_2 are the same, $S_{\text{word}}(d_1, d_2)$ is 1.

D. CONCATENATION OF RECOMMENDATION SETS

For a target Mashup, all the APIs used by the target mashup are the target APIs. Equation (8) is used to calculate the similarities between APIs in the low-dimensional space, and then K neighbor APIs of each target API are selected to form the recommended set AS1. In addition, the formula (14) calculates the similarities of Mashup-API invocation matrices of APIs, and then the K neighbor APIs of the target APIs form the recommended set (AS2). In order to merge (AS1) and (AS2), the API similarities in the two sets are normalized. The normalized formula is as follows:

$$M_i = \frac{s_i - s_{min}}{s_{max} - s_{min}} \tag{20}$$

where s_i is the similarity of the original ith API in the set, s_{min} is the lowest API similarity value in the set, and s_{max} is the highest API similarity value in the collection.

After the set (AS1) and set (AS2) are normalized, they become the union of the two sets, and then the scores for the recommended APIs in the two sets are determined. If the API was included in the set before, the score of the recommended API is the normalized similarity of the API. Otherwise, the score of this recommended API is 0. In this way, two sets of the same length are obtained, and the recommended APIs in two sets are the same.

Next, set (AS1) and set (AS2) are merged into an API-based recommendation set (AS). The score of each API in (AS) is calculated as follows:

$$Sr_i = \begin{cases} s_1 & if\ s_2 = 0 \\ s_2 & if\ s_1 = 0 \\ \dfrac{s_1 + s_2}{2} & if\ s_1 \neq 0\ and\ s2 \neq 0 \end{cases} \tag{21}$$

Where s_1 is the score of the ith API in the set AS1, AND s_2 is the score of the ith API in the set (AS2).

Eq. (11) is used to calculate the functional similarities between the target mashup and other mashups, and then all APIs used by the K neighbor mashups of the target mashup form the recommended set (MS1). The similarities between the target mashup and the recommended APIs are the similarities between the neighbor mashups and the APIs. At the same time, Eq. (11) is used to calculate the similarities between the Mashup-API invocation matrices of mashups. The APIs used by the K neighbor mashups of the target mashup form the recommended set (MS2).

Considering that the similarity range of recommended API in (MS1) is different from that in (MS2), Eq. (20) is used to normalize the two sets. Similar to the formation of the API-based recommendation set, (MS1) and (MS2) are transformed into two sets of the same length, and then (MS1) and (MS2) are combined into the Mashup-based recommendation set (MS) through Eq. (21).

Finally, Eq. (21) is used to merge API-based recommendation set (AS) and mashup-based recommendation set (MS) into one set and take Top-N APIs as final recommendation set. When obtaining Top-N APIs, it is possible that multiple APIs have the same score, which leads to the number of recommended APIs exceeds N. Since a large number of experiments have proved that the Mashup-based recommendation is more reliable, if the score is the same, we prefer to choose the APIs from set MS. If APIs with the same score are all from MS, we choose the API that has been used more often.

IV. EXPERIMENTAL EVALUATION

In our model, the selection of K which is the number of neighbors is very important, so we analyzed the influence of K on the experimental results. In order to verify the accuracy of our algorithm, we used real data obtained from PWeb to compare our model with several traditional collaborative

filtering recommendation models. Experiments show that our service recommendation model is better than the existing service recommendation models based on collaborative filtering in *Recall*, *Precision* and $F1$.

A. DATASET

For the authenticity of the experiment, we obtained 18,536 real APIs and 7,732 real mashups from PWeb. We selected all the mashups that use three APIs as the test set, and remove the mashups without text description. Finally, the number of mashups selected in the test set is 769.

In order to evaluate the effectiveness of the recommendation method, we removed one of the three APIs from each target mashup in the test set. According to the two APIs that have not been removed, recommendation model recommends APIs for the target mashup. If the removed API is in the Top-N recommendation set, the recommendation is successful. Since a target mashup in the test set will have three different sets of data depending on the different removed API, the evaluation metric of the target mashup is the average of three experimental results.

B. EVALUATION METRICS

In the experiment, we used the *Precision*, *Recall* and $F1$ to evaluate the effectiveness of the recommendation model. *Precision* and *Recall* are widely used when comparing the effects of methods [34], and $F1$ is a combination of *precision* [35] and *Recall* [36]. The metrics are defined as follows.

Precision@N [37] is a popular type of *Precision* for evaluating recommendation systems. It indicates how many of all recommended APIs are marked as originally belonging to the target mashup. The formula is as follows:

$$Precision@N = \frac{|\{Real\ \ APIs\} \cap \{Recommend\ \ APIs\}|}{N} \tag{22}$$

where *Real APIs* represents the APIs set marked in the target mashup, and *Recommend APIs* represents the Top-N recommended APIs. Since *Precision@N* refers to the ratio of the number of recommended correct APIs to the total recommended APIs, the larger N is, the smaller *Precision@N* will be.

Recall@N indicates how many of the APIs marked by the target mashup are in the Top-N recommended API set. The calculation formula is as follows:

$$Recall@N = \frac{|\{Real\ \ APIs\} \cap \{Recommend\ \ APIs\}|}{|\{Real\ \ APIs\}|} \tag{23}$$

$F1@N$ considers both the *Recall* and the *Precision*. Therefore, the $F1$ is often used as an important indicator of the recommendation system evaluation. $F1@N$ is defined as equation (24):

$$F1@N = \frac{2Precision@N \times Recall@N}{Precision@N + Recall@N} \tag{24}$$

C. THE INFLUENCE OF THE PARAMETER K

There is an important parameter K in our model, which is the number of neighbors of the target mashup or target API in the model. K has a great influence on the experimental results. When K is set too small, the recommended APIs will be too few to compare the experimental results under different N which is the number of recommended APIs. When the value of K is set too large, it will seriously affect the running speed of the model. In this section, we study the influence of K on the recommendation results through experiments.

We set K in the KGCF-SR algorithm to 5, 7, 9, 11 respectively, and then calculated the *Precision*, *Recall* and $F1$ of Top-N under different K. N is the number of APIs recommended by the recommendation model. The value of N in the relevant research is generally evenly distributed Between 0 and 20. We chose the value of N as 5, 10, 15 and 20. The experimental result is shown in Figures 5-7.

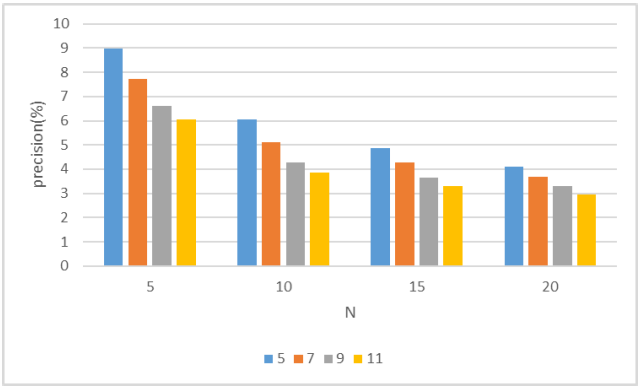


FIGURE 5. ***Precision*** **of KGCF-SR under different** ***K*** **settings.**

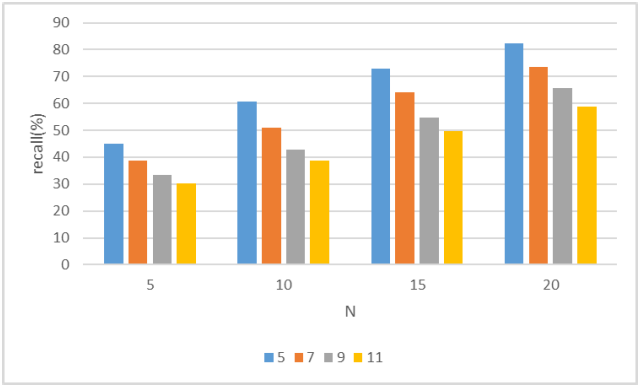


FIGURE 6. ***Recall*** **of KGCF-SR under different** ***K*** **settings.**

It can be seen from Figure 5 that the *Precision* of our recommending model decreases with the increase of the K, and the smaller the API recommended number N is, the greater the decline of *Precision* with the increase of K. When the N is 5, the *Precision* of top-5 when K is 7 is 14% lower than the *Precision* of top-5 when K is 5. When N reaches 20, the change in K has little effect on the *Precision* of the top-20.

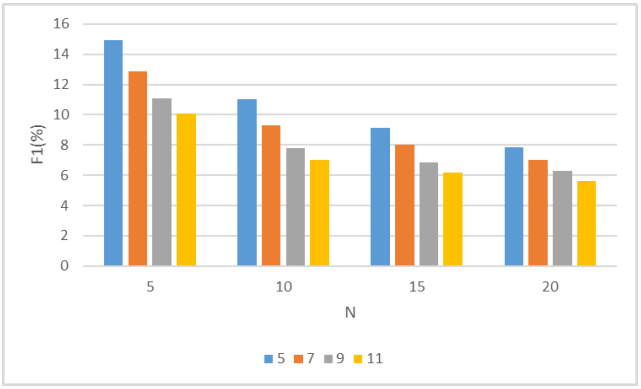


FIGURE 7. *F1* of KGCF-SR under different *K* settings.

Fig. 6 shows that the Top-*N* *Recall* of the KGCF-SR model also decreases with the increase of K, and the change rule is similar to that of the *Precision*. Affected by the *Precision* and *Recall*, the larger the K is, the smaller the *F1* is.

Experiments showed that the value of K has an impact on the final experimental results, especially when N is small, the choice of K has a huge impact on the recommended performance. Generally speaking, when the value of K is larger, the recommendation effect of API is worse. Therefore, the value of K selected in our model should be as small as possible. But when K is too small, the final recommended APIs are too few. Considering comprehensively, the parameter K in our model is set to 5 in all the following comparison experiments.

D. APPROACH COMPARISON

To evaluate the performance of our model, we compared our method with existing service recommendation methods. These methods are briefly explained as follows.

1) Function-Based Recommendation (FBR): The purpose of this method is to recommend functionally similar APIs for the target mashup. First, the Stanford Parser tool is used to extract the verbs and objects of the description information in mashups and APIs. Then the extracted verbs and objects form a functional set. The similarities between the target mashup and APIs are calculated through functions. Finally, the most similar Top-*N* APIs are recommended to the target mashup.
2) API-Based Recommendation though LDA (ABR-LDA): This method first obtains the description information of all APIs, then LDA [38] is used to model APIs as topic probabilistic proportion vectors. Finally, the most similar Top-*N* APIs of each API used by the target mashup are recommended.
3) Mashup-Based Recommendation through Function (MBRF): Similar to method 1, this method first uses the Stanford Parser tool to extract the functions of all mashups. Based on the similarities of the functions, the model recommends the APIs used by similar mashups of the target mashup.
4) Mashup-Based Recommendation through VSM (MBR-VSM): This method uses the VSM algorithm to map the description information of the mashups into feature vectors. Then the similarities between mashups are calculated. Finally, the APIs used by the K neighbor mashups that are most similar to the target mashup form the final recommendation list.
5) Service recommendation based on knowledge map and collaborative filtering (KGCF-SR): our function.

Figures 8-10 show the comparison of our method and other recommended methods. These three figures respectively show the *Precision*@*N*, *Recall*@N, *F1*@N of five methods. In general, our method is obviously better than the other four methods.

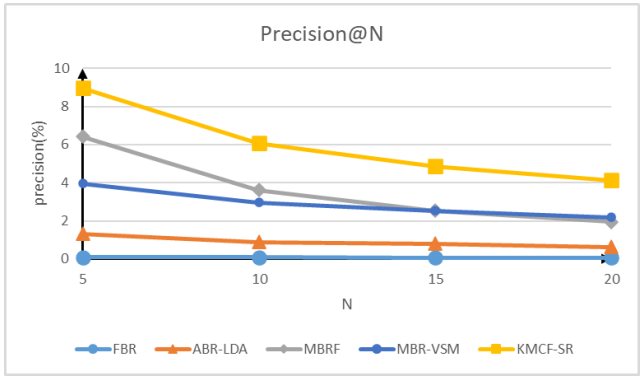


FIGURE 8. The comparison on the overall *Precision*@*N*.

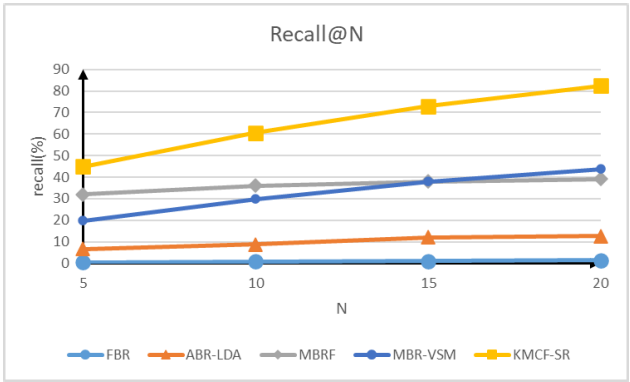


FIGURE 9. The comparison on the overall *Recall*@N.

Experiments show that the effect of API-based collaborative filtering recommendation through the LDA is very poor. The main reason is that two APIs with greater correlation cannot cooperate well with each other in the same mashup. Generally, two similar APIs often have a competitive relationship and can replace each other. The effect of the function-based recommendation method is also very bad, mainly because the description information of mashups and APIs is not accurate, and a lot of description information is even incomplete.

Compared with the above methods, the two methods of Mashup-based collaborative filtering recommendation have

TABLE 1. Evaluation results of five recommendation algorithms.

	N=5	N=10	N=15	N=20
Precision				
FBR	0.09%	0.08%	0.07%	0.07%
ABR-LDA	1.33%	0.88%	0.8%	0.63%
MBRF	6.41%	3.6%	2.53%	1.95%
MBR-VSM	3.95%	2.97%	2.53%	2.18%
KGCF-SR	8.97%	6.06%	4.86%	4.12%
Recall				
FBR	0.48%	0.78%	1.00%	1.34%
ABR-LDA	6.67%	8.75%	12.00%	12.61%
MBRF	32.03%	36.02%	37.97%	39.10%
MBR-VSM	19.77%	29.74%	37.97%	43.69%
KGCF-SR	44.86%	60.60%	72.91%	82.36%
F1				
FBR	0.16%	0.15%	0.13%	0.13%
ABR-LDA	2.22%	1.59%	1.50%	1.20%
MBRF	10.68%	6.55%	4.74%	3.72%
MBR-VSM	6.59%	5.41%	4.74%	4.16%
KGCF-SR	14.95%	11.02%	9.12%	7.84%

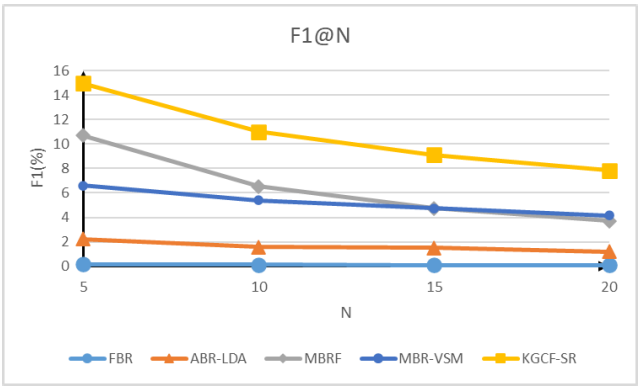


FIGURE 10. The comparison on the overall *F1*@N.

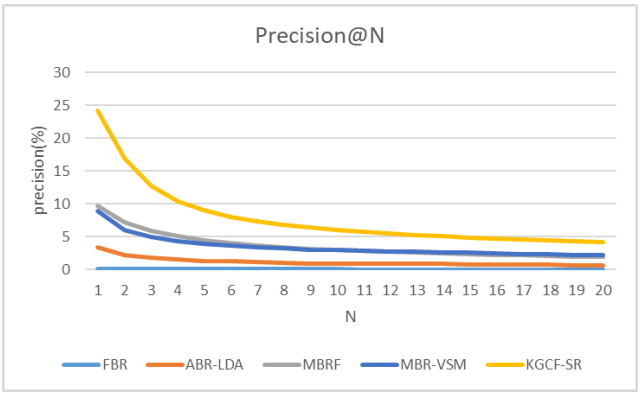


FIGURE 11. *Precision@N from top-1 to top-20.*

better performance. Because only a few of APIs have been used by mashups, and API usage frequency varies greatly. The frequently used APIs are used more than 2000 times, while the rarely used APIs have been used no more than 10 times. The invocations of the 200 most frequently used APIs even accounts for more than 90% of the historical records of invocations. It can be seen that mashups tend to use popular APIs, which makes two mashups with similar functional requirements often use the same API. Therefore, MBRF and MBR-VSM have better performance.

Our method KGCF-SR not only considers the functional information of mashups, but also considers the Mashup-API invocation matrices and potential relations between the APIs in the service knowledge graph. Experiments showed that our model has better effect. It can be seen from Figure 9 that when N is 5, the gap between our method KGCF-SR and other methods is not large. As N increases, the advantage of our method becomes more and more obvious. When the number of recommended APIs increases to 20, the *Recall* of our method has reached 0.82, which is far better than other methods. In addition, the *Recall* tends to be stable with the growth of n.

The values of N selected in Top-N above are not continuous, which makes the experimental results not rigorous. In order to make the experiment more rigorous, we calculated all the *Precision*, *Recall* and *F1* of the five models from top-1 to top-20. The *Precision*, *Recall*, and *F1* from Top-1 to top-20 are shown in Figures 11-13.

TABLE 2. The cases of recommendation.

Target Mashup	Method	Top 3 Recommendations of APIs		
Acme georss	FBR	Active financial	stupeflix	data.washington
	ABR-LDA	kelkoo	shipstation	Twilio sms
	MBRF	Google maps	youtube	**Yahoo maps**
	MBR-VSM	foursquare	instagram	google calendar
	KGCF-SR	Yahoo search	gowalla	Syndicate plus
songs	FBR	bing-maps-locations	battlecell	noaa-national-weather-service
	ABR-LDA	identi.ca	feedly	wiziq-virtual-classroom
	MBRF	flickr	Yahoo boss	Yahoo local search
	MBR-VSM	youtube	lyricsfly	Yahoo local search
	KGCF-SR	backtype	flats	google-ajax-language
amanav	FBR	betfair	big property list	ustream.tv
	ABR-LDA	bitfinex	gigjunkie	boxcar
	MBRF	twitter	**Amazon product advertising**	Google earth
	MBR-VSM	last.fm	lyricwiki	coindesk
	KGCF-SR	**Amazon product advertising**	Rpm software	google-friend-connect
actually	FBR	windows-live-contacts	facebook-real-time-updates	best-buy-reviews
	ABR-LDA	cloudmade	imdbapi.org	geogratis
	MBRF	google-ajax-search	google-ajax-language	google-o3d
	MBR-VSM	discogs	**500px**	twitter
	KGCF-SR	rhymebrain	bit.ly	google-ajax-search
App review tube	FBR	sports-power	layar	flixster
	ABR-LDA	gastro	Google maps	4-1-search
	MBRF	Google maps	twitter	Yahoo geocoding
	MBR-VSM	Amazon product advertising	cnet	netflix
	KGCF-SR	**Google search**	bizvizz	Realtime register

It can be seen from the figure that when N takes continuous values, the effect of our method is still significantly better than other methods

E. CASE STUDY

In this section, we will use actual cases to show the results of the five recommended API recommendation methods. We randomly selected 5 target mashups from the mashup data set, and each mashup in the data set was removed with a used API. The five selected mashups are *Acme Georss*, *Songs*, *Amanav*, *Actually*, *App Review Tube*. Then five methods were used to recommend top-3 APIs for target mashups. The final recommendation results are shown in Table 2. The bolded API is the one that has been removed from APIs used by the target mashup. If there is a bolded API in the recommended list of top-3, it means that the recommended method hits the correct API.

In the five randomly selected samples, KGCF-SR hits twice, MBRF also hits twice, and MBR-VSM hits once. In contrast, ABR-LDA and FBR did not perform well, none of their recommended top-3 APIs hits. From the results, both KGCF-SR and MBRF hit twice. KGCF-SR did not show an advantage, because it can be seen from the previous part that the top-3 *Precision* of KGCF-SR and MBCF are both low and similar, and the target mashups were all randomly selected. In fact, the superiority of our method has been fully reflected in the experimental comparison of the previous part. The purpose of this part is to show real cases of recommendation.

It can be seen from the table that some popular APIs appear frequently and rank very high in the recommended list even if they are completely unrelated to the target mashup.

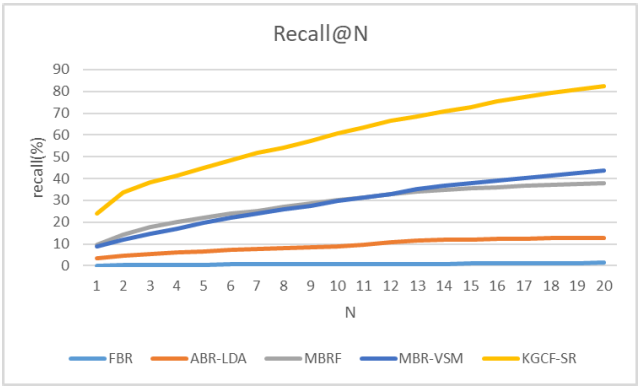


FIGURE 12. ***Recall@N*** **from top-1 to top-20.**

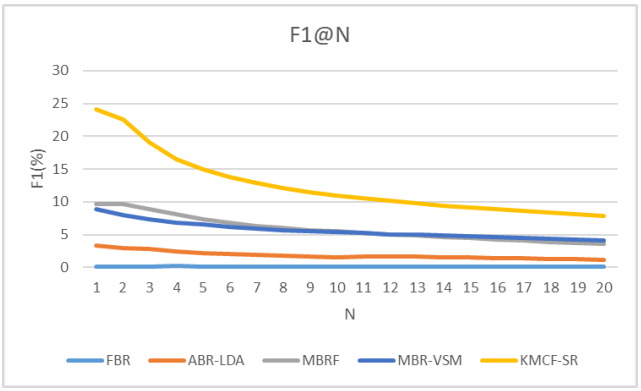


FIGURE 13. ***F1@N*** **from top-1 to top-20.**

For example, some popular APIs released by Google and Yahoo. It shows that these methods have the Matthew effect. Those APIs that are used more frequently will have a higher ranking.

V. CONCLUSION

In this paper, we propose a new method of recommending APIs for mashups, which is based on knowledge graphs and collaborative filtering. In the proposed model, in order to mine the potential relations between APIs and mashups, TransH is used to embed the entities in the service knowledge graph into a low-dimensional space. In addition, the Mashup-API invocation matrices and functions of mashups are also considered.

Experiments showed that our method can provide more accurate recommendation results compared with existing methods.

However, our method has some limitations. The main problem is the Matthew effect, that is, APIs that are frequently used by mashups rank high in the recommended list. For example, *Google maps* has been used by mashups for more than 2000 times, which leads to a high ranking of Google maps, even if it is completely unrelated to the target mashup.

Since APIs used by mashups account for a small proportion of all APIs, the Matthew effect has a greater impact on the recommendation results. In addition, the incomplete description information of mashup leads to inaccurate extracted functions, which also affects the final recommendation effect.

In the future, we will improve our method in view of the above shortcomings. First of all, in order to reduce the impact of the Matthew effect, the weight of popular APIs should be reduced to make their ranking in the recommended list more reasonable. For the problem of incomplete description information of mashups, it is necessary to use natural language processing technology to complete the description text. In this way, the functions of the mashups can be extracted accurately.

REFERENCES

[1] W. Pan, X. Xu, H. Ming, and C. K. Chang, "Clustering mashups by integrating structural and semantic similarities using fuzzy AHP," *Int. J. Web Services Res.*, vol. 18, no. 1, pp. 34–57, Jan. 2021.

[2] T. Espinha, A. Zaidman, and H.-G. Gross, "Web API growing pains: Stories from client developers and their code," in *Proc. Softw. Evol. Week - IEEE Conf. Softw. Maintenance, Reeng., Reverse Eng. (CSMR-WCRE)*, Feb. 2014, pp. 84–93.

[3] V. Hoyer and M. Fischer, "Market overview of enterprise mashup tools," in *Proc. 6th Service-Oriented Comput. (ICSOC)*, Sydney, NSW, Australia, 2008, pp. 708–772.

[4] D. Benslimane, S. Dustdar, and A. Sheth, "Services mashups: The new generation of Web applications," *IEEE Internet Comput.*, vol. 12, no. 5, pp. 13–15, Sep. 2008.

[5] W. Pan and C. Chai, "Structure-aware mashup service clustering for cloud-based Internet of Things using genetic algorithm based clustering algorithm," *Future Gener. Comput. Syst.*, vol. 87, pp. 267–277, Oct. 2018.

[6] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, Jan. 2004.

[7] B. Cao, M. Tang, and X. Huang, "CSCF: A mashup service recommendation approach based on content similarity and collaborative filtering," *Int. J. Grid Distrib. Comput.*, vol. 7, no. 2, pp. 163–172, Apr. 2014.

[8] R. Xiong, J. Wang, N. Zhang, and Y. Ma, "Deep hybrid collaborative filtering for Web service recommendation," *Expert Syst. Appl.*, vol. 110, pp. 191–205, Nov. 2018.

[9] M. M. Rahman, X. Liu, and B. Cao, "Web API recommendation for mashup development using matrix factorization on integrated content and network-based service clustering," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2017, pp. 225–232.

[10] G. Tian, J. Wang, K. He, C. Sun, and Y. Tian, "Integrating implicit feedbacks for time-aware Web service recommendations," *Inf. Syst. Frontiers*, vol. 19, no. 1, pp. 75–89, Feb. 2017.

[11] Y. Zhong, Y. Fan, W. Tan, and J. Zhang, "Web service recommendation with reconstructed profile from mashup descriptions," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 468–478, Apr. 2018.

[12] X. Wang, H. Wu, and C.-H. Hsu, "Mashup-oriented API recommendation via random walk on knowledge graph," *IEEE Access*, vol. 7, pp. 7651–7662, 2019.

[13] T. K. Paradarami, N. D. Bastian, and J. L. Wightman, "A hybrid recommender system using artificial neural networks," *Expert Syst. Appl.*, vol. 83, pp. 300–313, Oct. 2017.

[14] Z. Cao, X. Qiao, S. Jiang, and X. Zhang, "An efficient knowledge-graph-based Web service recommendation algorithm," *Symmetry*, vol. 11, no. 3, p. 392, Mar. 2019.

[15] X. Yu, Y. Chu, F. Jiang, Y. Guo, and D. Gong, "SVMs classification based two-side cross domain collaborative filtering by inferring intrinsic user and item features," *Knowl.-Based Syst.*, vol. 141, pp. 80–91, Feb. 2018.

[16] X. Yu, J. Yang, and Z. Xie, "Training SVMs on a bound vectors set based on Fisher projection," *Frontiers Comput. Sci.*, vol. 8, no. 5, pp. 793–806, Oct. 2014.

[17] X. Yu, F. Jiang, J. Du, and D. Gong, "A cross-domain collaborative filtering algorithm with expanding user and item features via the latent factor space of auxiliary domains," *Pattern Recognit.*, vol. 94, pp. 96–109, Oct. 2019.

[18] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2724–2743, Dec. 2017.

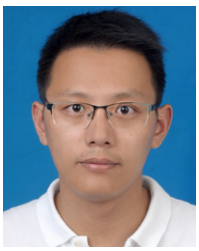
[19] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "DBpedia—A large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.
[20] F. M. Suchanek, G. Kasneci, and G. Weikum, "YAGO: A core of semantic knowledge unifying WordNet and wikipedia," in *Proc. 16th Int. Conf. World Wide Web*. New York, NY, USA, May 2007, pp. 697–706.
[21] Q. Li, X. Tang, T. Wang, H. Yang, and H. Song, "Unifying task-oriented knowledge graph learning and recommendation," *IEEE Access*, vol. 7, pp. 115816–115828, 2019.
[22] C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, and C. Zhang, "Feature engineering for knowledge base construction," 2014, *arXiv:1407.6439*. [Online]. Available: http://arxiv.org/abs/1407.6439
[23] T. Mikolov, L. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2, Dec. 2013, pp. 3111–3119.
[24] M. Nickel, V. Tresp, and H. P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, Washington, DC, USA, Jun. 2011, pp. 809–816.
[25] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, Dec. 2013, pp. 926–934.
[26] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modelingmulti-relational data," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, Dec. 2013, pp. 2787–2795.
[27] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proc. 28th AAAI Conf. Artif. Intell.*, Jul. 2014,.pp 1112-1119.
[28] H. Xiao, M. Huang, Y. Hao, and X. Zhu, "TransA: An adaptive approach for knowledge graph embedding," 2015, *arXiv:1509.05490*. [Online]. Available: https://arxiv.org/abs/1509.05490
[29] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proc. 28th AAAI Conf. Artif.Intell.*, Austin, TX, USA, Jan. 2015, pp. 2181–2187.
[30] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *Proc. 53rd Annu. Meeting Assoc. Comput. Linguistics 7th Int. Joint Conf. Natural Lang. Process. (Long Papers)*, vol. 1, 2015, pp. 687–696.
[31] J. Partner, A. Vukotic, N. Watt, T. Abedrabbo and D. Fox,*Neo4j in Action Pearson Schweiz Ag*, 2014, p. 304.
[32] M. Mozgovoy and R. Efimov, "WordBricks: A virtual language lab inspired by scratch environment and dependency grammars," *Hum.-Centric Comput. Inf. Sci.*, vol. 3, no. 1, pp. 1–9, Dec. 2013.
[33] G. A. Miller and A. George, "WordNet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.
[34] W. Pan, H. Ming, C. Chang, Z. Yang, and D.-K. Kim, "ElementRank: Ranking java software classes and packages using a multilayer complex network-based approach," *IEEE Trans. Softw. Eng.*, early access, Oct. 8, 2019, doi: 10.1109/TSE.2019.2946357.
[35] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of java software systems by weightedK-core decomposition," *Future Gener. Comput. Syst.*, vol. 83, pp. 431–444, Jun. 2018.
[36] H. Li, T. Wang, W. Pan, M. Wang, C. Chai, P. Chen, J. Wang, and J. Wang, "Mining key classes in java projects by examining a very small number of classes: A complex network-based approach," *IEEE Access*, vol. 9, pp. 28076–28088, 2021.
[37] N. Craswell, L. Liu, and M. T. Zsu, "Precision at N," *Encyclopedia Database Syst.*, to be published.
[38] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

BO JIANG received the Ph.D. degree from the School of Computer, Zhejiang University, China. She is currently a Professor and a M.S. Supervisor with the School of Computer Science and Information Engineering, Zhejiang Gongshang University. She has published more than 30 articles in international journals and conferences. Her current research interests include service computing and complex networks. She is also a member of China Computer Federation (CCF) and CCF Service Computing Association.

JUNCHEN YANG is currently pursuing the M.S. degree with the School of Computer Science and Information Engineering, Zhejiang Gongshang University. His research interests include service computing and software engineering.

YANBIN QIN is currently pursuing the M.S. degree with the School of Computer Science and Information Engineering, Zhejiang Gongshang University. His research interests include service computing and complex networks.

TIAN WANG is currently pursuing the M.S. degree with the School of Computer Science and Information Engineering, Zhejiang Gongshang University. Her research interests include software engineering and complex networks.



MUCHOU WANG received the bachelor's degree from the Department of Computer Science, Zhejiang University of Technology (ZJUT), in 2005, and the master's degree from the Huazhong University of Science and Technology (HUST), in 2008. He is currently working with Wenzhou University. His research interests include service-oriented software engineering and digital library.

WEIFENG PAN received the Ph.D. degree from the School of Computer, Wuhan University, China, in 2011. He has been a Visiting Scholar with Western Michigan University. He is currently an Associate Professor and a M.S. Supervisor with the School of Computer Science and Information Engineering, Zhejiang Gongshang University. He has published more than 50 articles in international journals, such as IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *Future Generation Computer Systems*, and *Cluster Computing*. His current research interests include software engineering, service computing, complex networks, and intelligent computation. He is also a member of China Computer Federation (CCF) and CCF Service Computing Association.