Received November 22, 2018, accepted December 19, 2018, date of publication December 28, 2018,
date of current version January 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2890156

Mashup-Oriented API Recommendation via Random Walk on Knowledge Graph

XIN WANG[1], HAO WU[1], AND CHING-HSIEN HSU[2], (Senior Member, IEEE)
[1]School of Information Science and Engineering, Yunnan University, Kunming 650500, China
[2]Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi 62102, Taiwan

Corresponding author: Hao Wu (haowu@ynu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61562090, Grant 61472345, and Grant 61562092, and in part by the China Postdoctoral Science Foundation under Grant 2016M592721.

ABSTRACT With the growing prosperity of the Web API economy, mashup-oriented API recommendation has become an important requirement. Various methods based on different principles of technology have been used to deal with this issue. In recent years, the Web API ecosystem has accumulated a wealth of knowledge that can be used to enhance the recommendation models, and however, current concerns in this regard still remain. To cope with this issue, we present a graph-based algorithmic framework for the task of mashup-oriented API recommendation. Especially, we design a concise schema of the knowledge graph to encode the mashup-specific contexts and model the mashup requirement with graphic entities. We then exploit random walks with restart to assess the potential relevance between the mashup requirement and the Web APIs according to the knowledge graph. In addition, we propose the query-specific weighting strategies to enhance the knowledge graph construction. The experimental results demonstrate that our proposed method is much superior to some state-of-the-art methods, also achieves robust effects on reducing computational overhead, and suppresses the negative Matthew effect in APIs' recommendation.

INDEX TERMS Mashup development, API recommendation, random walks with restart, knowledge graph.

I. INTRODUCTION

Web APIs are application programming interfaces through which web applications can realize storage services, message services, computing services and other capabilities. The number of accessible Web APIs has grown consistently over the past years. ProgrammableWeb, the largest online API registry, has tracked more than 20,000 Web APIs recently. As Web APIs become the backbone of the Web, cloud, mobile and machine learning applications, API ecosystem has gradually formed and an API economy is emerging [1]. However, in the face of rapid development of the information society and the emergence of a large number of additional requirements, the functions of existing APIs have become increasingly unable to respond to complex business needs. Regarding this issue, mashup has emerged as a technology for today's challenges by integrating multiple services to match requirements of users even with users who have little programming skills [2].

Unfortunately, with the rapid growth of the number of Web APIs, quickly selecting the right Web APIs from a large number of candidates covering a wide range of functionalities has become increasingly challenging for inexperienced developers. Therefore, it is necessary to develop recommendation techniques and help developers to better identify relevant Web APIs satisfying the need of mashup developments in a shorter amount of time [3], [4]. In recent years, numerous efforts have been made to address this issue. Existing works can be coarsely classified into two categories, one focuses on the principle of collaborative filtering [5]–[8], and the other focuses on estimating the relevance between the mashup requirements and the candidate APIs [9]–[13]. Various technologies, e.g., matrix factorization [7], [8], topic modeling [9], [10], link analysis [11] and various features, e.g., texts, tags, topics and popularity are exploited to enhance the accuracy of recommendations [13]–[17].

Nevertheless, most of the existing works which use single source information are vulnerable to data sparsity. For example, methods based on text similarity or topic mining will get worse effect once the text description is poor or insufficient [7]. In contrast, knowledge graph usually contains much more fruitful facts and connections of APIs, mashups and other items. A knowledge graph is a type of directed heterogeneous graph in which nodes correspond to *entities* and edges correspond to *relations*. These semantics can help us to understand mashup patterns accurately. The Web API ecosystem has accumulated a wealth of knowledge that can be used to enhance recommendation models [18], however, the current concerns in this regard are still limited. Particularly, there is the absence of a unified and easy-to-use way. As to take advantage of these knowledge, they usually combine different technologies, resulting in the relatively complex algorithms which are hard to understand and practice. Moreover, how to exploit key knowledge instead of all the knowledge to benefit recommendation of APIs and how to effectively express users' requirements need to be further explored.

Inspired by these, we propose a simple yet effective recommendation framework based on random walks on the knowledge graph for mashup developments. In this approach, we use knowledge graph to capture the most relevant information which is related to Web APIs and mashups. Then, we use Random Walks with Restart(RWR) to estimate the relatedness between the mashup requirements and the candidate APIs. Also, we propose three query-specific weighting strategies to improve the relatedness estimation. In this way, we can effectively address the problem of data sparsity, providing an elegant way to utilize the abundant knowledge in the API ecosystem and achieving better recommendation performance.

The main contributions of this paper can be summarized as follows:

- We have proposed an effective schema of knowledge graph to capture the most useful knowledge for mashup-oriented API recommendation and proposed three query weighting strategies to enhance the recommendation effectiveness of RWR.
- Through comprehensive experimental analyses, our proposed method can provide highly accurate recommendation results in comparison with the state-of-the-art methods. Particularly, RWR using our graph schema promotes that case of using a full graph schema by at least 5.9-9.5% on *Recall*@{5,10} and at least 4.5-5.7% on *NDCG*@{5,10}. With query-specific weighting strategies, we further improve the recommendation effects of RWR by at least 11.9-17.7% on *Recall*@{5,10} and at least 12.9-13.3% on *NDCG*@{5,10}.

The rest of the paper is structured as follows. Section II provides some backgrounds to API-specific knowledge graph and the random walk method. Section III presents our enhanced model by introducing refined knowledge graph and query weighting strategies. We evaluate our methods via a set of experiments in Section IV. We review some works which are most relevant to ours in Section V. Finally, we conclude our works and point to future works.

II. PRELIMINARY MODEL

A. KNOWLEDGE GRAPH FOR APIS RECOMMENDATION

In the past decades, the use of Web API has increased significantly. However, the lack of semantic descriptions of Web APIs limits their discovery, sharing, integration and consumption [18]. To cope with this problem, Dojchinovski and Vitvar [18] have presented the *Linked Web APIs dataset* with semantic descriptions of Web APIs to capture the provenance, temporal, technical, functional, and non-functional aspects. Specially, the Linked Web APIs ontology is designed to capture the most relevant information which is related to Web APIs and mashups. This provides us with a knowledge base (graph) to guide APIs recommendation.

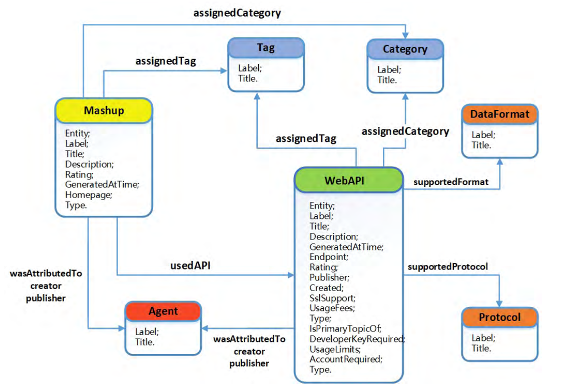


FIGURE 1. A full schema of knowledge graph for mashup-oriented API recommendation.

A full schema of knowledge graph is presented as Fig.1, which includes key entities: *Tag*, *Category*, *Mashup*, *WebAPI*, *Agent*, *Protocol*, *DataFormat* as well as relationships among them and attributes associated with, such as *isPrimaryTopicof*, *created*, *creator*, *rating*, *title*, *name*, *type*, *label*, *publisher*, *homepage*, *wasAttributedTo*, *supportedProtocol*, *supportedDataFormat*, *sslSupport*, etc. There are totally 3286 tag entities and 66 category entities. Here, we specially distinguish entities of *Tag* from entities of *Category* considering that tags are usually user-generated in the free way while categories are assigned adopting to a standard vocabulary.

B. RANDOM WALKS WITH RESTART

PageRank is an ordering node technique based on *Markovian* walks in a directed graph $G = (V, E)$, where $V(|V| = n)$ is the node set and E is the edge set. The surfer jumps from one node to another with a consistent probability of α (*damping factor*) or gets bored, and then jumps to a random node with a probability of $1 - \alpha$. Assuming P is the ranking vector for all nodes in G, the PageRank value of P_i is the surfer's probability at a given node of i. The method of fast computing PageRank is to use the *power iteration method*,

$$P^{(t)} = \alpha M^T P^{(t-1)} + (1-\alpha)e \quad (1)$$

where M is a row-stochastic matrix ($n \times n$). Beginning with an arbitrary vector $P^{(0)}$, the solving of Eq.1 is to apply the operator $\widehat{M^T} = \alpha M^T + (1-\alpha)e$ in succession, until $|P^{(t+1)} - P^{(t)}| < \epsilon$. When setting the personalized vector e to prefer a subset of V [19], the PageRank model is usually called as Random Walks with Restart(RWR) [20], [31].

RWR has been used as a measure of relatedness in various recommendation scenes and been proved to achieve superior performance with the ability to alleviate data sparsity [20], [21]. It can be adapted to recommend Web APIs for the mashup development as follows: a) Given a knowledge graph, we treat the edges with different types as a bidirectional link with a unified weight of 1; b) set e to prefer the node representing a mashup; c) find the vector $P^{(t)}$ (where t is the state after convergence) using Eq.1; d) sort APIs by their rankings in $P^{(t)}$ and generate the top-N recommendations [22]. Specially, given the source node i, the target node j and the number of links from i to j can be expressed as $L(i, j)$. $L(i, k)$ is the same, in which, k represents the node connected to node i. The matrix M is initialized as follows:

$$M_{ij} = \begin{cases} \frac{L(i,j)}{\sum_k L(i,k)} & \textit{if } L(i,j) > 0, \\ 0 & \textit{otherwise} \end{cases} \tag{2}$$

RWR is simple and effective, however, it has a couple of drawbacks which lead to unsatisfying results in mashup-oriented API recommendation. One problem concerns the computational efficiency. As the computational complexity of RWR is $O(n^2t)$, the computational cost will be high when the number of entities in the knowledge graph is larger. Also, not all the information in the knowledge graph contributes to the accuracy of recommendations. Thus we are required to refine the knowledge graph to reduce the computational cost. Another problem is about the negative *Matthew effect* which means that the APIs of high popularity will always achieve higher ranking values [23]. This negative effect will reduce the diversity of recommendation lists and lower the accuracy of recommendations [22]. For instance, *Google Maps API* has been used more than 2000 times in mashup developments and frequently ranks higher in the recommendation lists, even if it does not match any requirement.

III. ENHANCED MODEL

A. REFINING KNOWLEDGE GRAPH

To improve RWR-based recommendation model, one feasible approach is to initialize a walking boundary to reduce unnecessary random surfing. We customize the query boundaries based on the mashup requirements: (I) only nodes of typed *Tag* or *Category* with rich semantic information and a strong transitivity ability are used to represent the mashup requirements. Other nodes are either unique or lack sufficient feature information; (II) only the nodes and edges within the boundary will be used to create the knowledge graph, and the other nodes and edges are omitted. Figure 2 specifies the walking boundary. Formally, we use the following notations to model requirements of mashups:

Defintion 1: A *mashup requirement*is specified by $Q = Q_{cat} \cup Q_{tag}$, where Q_{cat} is the set of entities typed *Category*, and Q_{tag} is the set of entities typed *Tag*.

Defintion 2: A *query node q* represents a mashup to be established (i.e., *test* node in Figure 2).

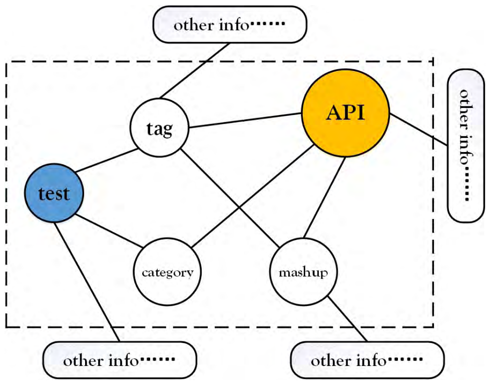


FIGURE 2. A refined schema of knowledge graph where the boundary of the dotted box is used to create a data graph. The node named *test* represents the node of target mashup.

The APIs recommendation corresponds to repeat a spread process from the query node q to the nodes in Q, and in turn to other nodes, until a stable global state is achieved. By employing this strategy, the number of visited nodes by RWR is greatly reduced, and thus the computational cost will be less. By the way, the negative *Matthew effect* will also be suppressed to some extent. According to our experiments, the number of visited nodes can be reduced by 98%, while the recommendation accuracy can be increased by around 10%. To distinguish from the original **RF** model using the full knowledge graph, we call this approach using the refined knowledge graph as **RR**.

It is worth noting that we use tags and categories to describe user needs for the following reasons: (1) Tags/categories can be regarded as precise summaries of API functions, so they carry more rich information. (2) Tags/categories have been recognized as successful in organizing and sharing resources in information systems, especially in the era of social Web, such as Flickr, YouTube, Delicious and other websites. This is also true for service/API repositories, e.g., ProgrammableWeb and Seekda.

B. QUERY-SPECIFIC WEIGHTING

Up to now, we use uniform weight for different links when constructing knowledge graph, and do not consider the impact of different weights of links on the recommendation effects. It is almost impossible to find the global optimal configuration of all link weights. For this reason, we introduce some simple but easy-to-operate heuristic strategies to adjust weights of specific links, to reflect the influence of user preferences on mashup developments.

Q1: In a common sense, the APIs contain more information related to Q, the more important they should be. Therefore, we strengthen the weights of links between Q and those APIs to reflect this principle.

Q2: In many cases, sets Q_{cat} and Q_{tag} contain the same keywords. For example, a mashup requirement may include both *social_tag* and *social_category*. In our dataset, 95.5% keywords in the Q_{cat} appears in the Q_{tag}, so we should give more weights to the APIs they point together.

Algorithm 1 *RRQ*

```
Input: $E$, $V$, mashup requirement $Q$, query node $q$, damping factor $\alpha$, weight ratio $\beta$, threshold $\epsilon$;
Output: The ranking vector in $G_r$;
  (1) Get the graph $G_r$ according to Fig.2 and let all link weights as 1;
   ## weighting strategy Q1
  (2) for $i \in Q$ do
  (3)     for $j \in E[i]$ do
  (4)         if $j$ is $API$ && $E[j] \in Q$ then
  (5)             $L(i,j) = L(i,j) + 1$;
  (6)         end if
  (7)     end for
  (8) end for
   ## weighting strategy Q2
  (9) for $i \in Q_{cat}$ and $j \in Q_{tag}$ do
  (10)    if $i$ shares the same keyword with $j$ then
  (11)        for $k \in E[i] \cap E[j]$ && $k$ is $API$ do
  (12)            $L(i,k) = L(i,k) + 1$;
  (13)            $L(j,k) = L(j,k) + 1$;
  (14)        end for
  (15)    end if
  (16) end for
   ## weighting strategy Q3
  (17) for $i \in Q_{cat}$ do
  (18)     $L(q,i) = L(q,i) + 1$;
  (19) end for
  (20) if $\frac{\sum_{i \in Q_{cat}} L(q,i)}{\sum_{j \in Q} L(q,j)} < \beta$ then;
  (21)     repeat (17)-(19)
  (22) end if
   ## performing RWR
  (23) Update $G_r$ with new weights;
  (24) Initialize the matrix $M$ by Eq.2;
  (25) for $j \in V$ do
  (26)     if $j == q$ then
  (27)         $\mathbf{e}_j = 1$;
  (28)     else
  (29)         $\mathbf{e}_j = 0$;
  (30)     end if
  (31) end for
  (32) repeat
  (33)     Compute $\mathbf{P}^{(t)} = \alpha M^T \mathbf{P}^{(t-1)} + (1-\alpha)\mathbf{e}$
  (34)     $t = t + 1$
  (35) until
  (36)     $|\mathbf{P}^{(t)} - \mathbf{P}^{(t-1)}| < \epsilon$
  (37) return  $\mathbf{P}$ ;
```

Q3: On the analyses of experimental datasets, we find that the category information is more important than that of tag due to the large number of tag entities. This results in two types of entities carrying different amounts of information. To clarify this point, we assign a weight β (0.3 in default) to the entities of category and $1 - \beta$ to the entities of tag.

Let $|Q|$ be the number of entities in the Q and $E = \{source\ node_j : [target\ node_1, ..., target\ node_k], ...\}$ be the link dictionary of all entities in the knowledge graph, detailed query edge weighting strategies are outlined in Algorithm 1. As query weighting strategies work by strengthening weights of specific links without modifying the algorithm of RWR, it is not hard to follow and operate in practice. Formally, we call this improved model as **RRQ**.

TABLE 1. Statistics of dataset.

Statistics	Timelines		
	2012	2013	2014
#mashups in the test dataset	516	401	138
#mashups in the training dataset	6,273	6,789	7,190
#APIs in the training dataset	8,279	10,664	11,339
#entities in the training knowledge graph	159,603	162,333	164,644

IV. EXPERIMENTS

In this section, we conducted extensive experiments aiming to answer the following three research questions:

1) RQ1 How do parameter settings affect the recommendation performance w.r.t random walks with restart on different schemas of knowledge graph?
2) RQ2 Are the proposed query-specific weighting strategies effective in mashup-oriented API recommendation?
3) RQ3 How do the proposed methods perform in comparison with the state-of-the-art methods in the task of mashup-oriented API recommendation?

A. DATASETS

For our experiments, we use the *Linked Web APIs dataset*[1] of which semantic descriptions of Web APIs retrieved from ProgrammableWeb.com. It contains 11,339 Web APIs descriptions, 7,415 mashups and almost 7,717 mashup developers' profiles, over half million of RDF triples. The average number of APIs used per mashup is 2.1.

For our purpose, we remove 87 mashups without containing any APIs usage and divide the dataset into three training datasets and three test datasets with timelines (years 2014, 2013 and 2012). For example, when setting the timeline in the year 2014, the historical data of mashups before the year 2014 is used for training and the historical data of mashup in the year 2014 is used for testing. Also, we respectively build the full knowledge graph and the refined knowledge graph according to the schema in Figure 1 and Figure 2. In addition, for graph-based recommendation algorithms, no textual information of APIs and mashups is used. For text-based recommendation algorithms, the textual descriptions of APIs and mashups are preprocessed by removing punctuations and stop words, and then used for similarity calculation and topic modeling analysis. Statistics of dataset are given in Table 1.

B. EVALUATION METRICS

Since we concentrate on recommending top-N Web APIs, we adopt three commonly used metrics, namely Recall, Normalize Discounted Cumulative Gain (NDCG) and Hamming

[1]http://linked-web-apis.fit.cvut.cz/

Distance (HD), to evaluate the recommendation performance. For these three metrics, bigger values indicate better performance.

Given a ranking list of recommendations, *Recall*@N is the ratio of retrieved real APIs to all real used APIs in the mashup development:

$$Recall@N = \frac{|\{real\ APIs\} \cap \{top\text{-}N\ APIs\}|}{|\{real\ APIs\}|}. \quad (3)$$

DCG@N is a measure to give more weights to high-ranking APIs, combining different APIs with different gain values. One of the commonly used descriptions is:

$$DCG@N = \sum_{i=1}^{N} \frac{2^{rel(i)} - 1}{log_2(i+1)}. \quad (4)$$

Here, $rel(i)$ takes a binary value to indicate whether a candidate API had been used in the target mashup or not. If it is true, $rel(i) = 1$; otherwise, $rel(i) = 0$. *NDCG*@N is achieved by normalizing *DCG*@N with the ideal DCG of the ranking lists of candidates: $NDCG@N = \frac{DCG@N}{IDCG@N}$, where *IDCG*@N is pre-calculated by counting all real APIs participated into the development of target mashup.

HD@N is a measure to evaluate the aggregate diversity of recommendation lists for different users. Given two ranking lists of APIs, it is defined as:

$$HD@N = 1 - \frac{overlap@N}{N}, \quad (5)$$

where overlap@N is the number of common APIs in the top-N position of two lists. For our experiments, this metric is helpful to quantify the performance of *Matthew effect* on different recommendation algorithms. If an algorithm has a significant *Matthew effect*, popular APIs are often appeared and highly ranked in different recommendation lists generated by this algorithm, which results in the lower uniqueness of different recommendation lists and the lower overall *HD*@N. Therefore, we calculate *HD*@N for all pairs of tested mashups and employ the averaged *HD*@N to investigate the impact of *Matthew effect*.

C. EVALUATION METHODS

Currently, there are no universal baseline models for mashup-oriented API recommendation. For this reason, we select some widely-used methods in related works and some newer state-of-the-art approaches as the baseline models [5]–[10], [28].

1) POPULARITY-BASED MODEL (Pop)

It is the most commonly used baseline model in the field of recommender systems. For each tag or category, a queue of APIs is created through sorting APIs by their frequencies in the mashup developments. For each tag or category of Q, one API is selected to enter into the recommendation list according to its queue order in each iteration, and such a process repeats several times until top-N candidates are completed.

2) TEXT-BASED COLLABORATIVE FILTERING (CF) [5]

This method resembles item-based collaborative filtering where items referred to mashups which are further represented by their text documents. We first calculate similarities between the target mashup and the existing mashups in the training dataset, to find top-similar mashups. Then, we recommend APIs used by the top-similar mashups. Formally, let $sim(i, j)$ be the cosine similarity between the documents of two mashups i and j, $N(i)$ be the top-similar mashups of i, $rel(j, api)$ be the relevance between the mashup j and api (For simple, $rel(j, api) = 1/m$ if there are m APIs in the mashup j), then the recommendation score of api for the target mashup i can be calculated as follows:

$$score(i, api) = \sum_{j \in N(i)} sim(i, j) * rel(j, api) \quad (6)$$

To estimate $sim(i, j)$, we merge the tag/category and the text description of mashups into documents, and then discard those words that are meaningless to the recommendation task, including punctuations and stop words.

3) TOPIC-MATCH MODEL(TM)

CF relies on text description to estimate the similarity may suffer from insufficient information or semantics, so we exploit Latent Dirichlet Allocation (LDA) [24] to analyze the documents of APIs and generate the topic-based semantic representation for each API. We then calculate $sim(i, api)$ based on topic-based representations of the API and the mashup i; and generate the recommendation list by $score(i, api) = sim(i, api)$. This resembles the retrieval-based APIs recommendation method [12] and achieves competitive results with CF-based methods according to our experiments.

4) SERVICE PROFILE RECONSTRUCTION MODEL (SPR) [10]

The key idea of it is to leverage descriptions and structures of mashups to discover the important lexical features of APIs and bridge the vocabulary gap between mashup developers and service providers. The implementation is to jointly model mashup descriptions and component services using author topic model (ATM) [30] to reconstruct service profiles. With exploiting word features derived from reconstructed service profiles, a new mashup-oriented API recommendation algorithm is developed.

5) RANDOM WALKS WITH RESTART

We respectively evaluate RWR on full knowledge graph (**RF**), RWR on the refined knowledge graph (**RR**), RWR on the refined knowledge graph with query weighting (**RRQ**). Also, a RWR model combined with latent semantic analysis, proposed by Liang *et al.* [28] to recommend tags for APIs, is also used as a baseline model. In this method [28], both graph structure information and semantic similarity (LSI) are exploited to automatically assign tags to unlabeled APIs. We adjust this method to make it suitable for APIs recommendation by exchanging the roles of APIs and tags, and evaluate it against our proposed methods based on knowledge

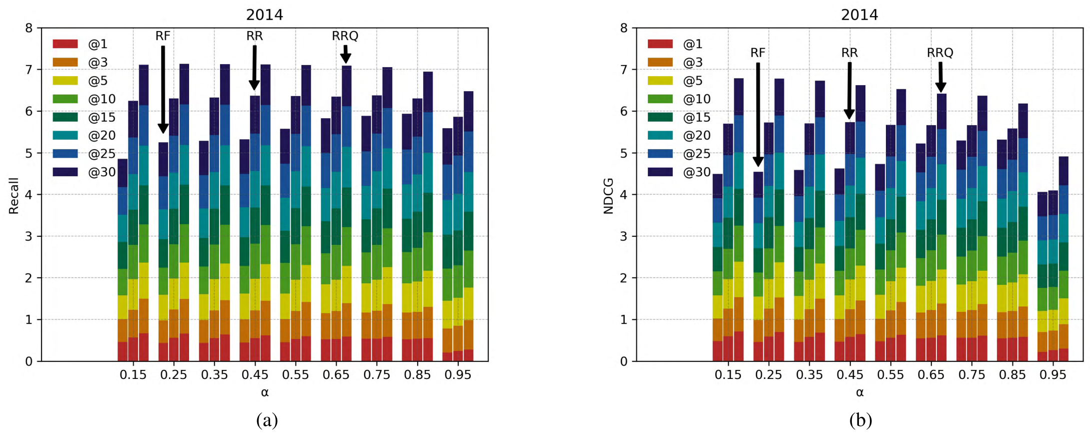


FIGURE 3. Performance on Recall and NDCG of three RWR models with different damping factors. For each column, the corresponding metric values of the legends are accumulated for the sake of compactness.

graph. Because the task is different, there is no need to refine the knowledge graph in their task. For the sake of fairness, we also use weighting strategies in this method. (we name it as **RRQ-L** for convenience).

We treat each keyword in Q (*tag* or *category*) as a single individual object in **Pop**, **SPR**, **RF**, **RR** and **RRQ**. We add keywords of *Tag* and *Category* to the textual description of mashups (also APIs) to calculate the text or topic similarities in **CF** and **TM**. In **RRQ-L**, we calculate the similarities between the text information of all associated APIs and the requirement information (Q), then multiply the corresponding terms of the similarity vector and the strength vector of the association obtained through RWR for each API to get a score vector of APIs. Finally, we reorder the score of each API to generate a recommendation list.

D. CONFIGURATION ANALYSIS

1) IMPACT OF DAMPING FACTOR

The recommendation algorithm based on RWR is significantly affected by the settings of the damping factor [20]. Therefore, we conduct experiments in this section to observe how the damping factor affect the accuracy of recommendations w.r.t variants of the RWR. As the difference is small, we choose to stack the data from top-1 to top-30 for an intuitive feel. The experimental results are presented in Figure 3. As for RF, we find that the optimal setting of the damping factor is close to 0.85, where both NDCG and Recall achieve the best values. As for RR and RRQ, recommendation performances in both NDCG and Recall change little as the damping factor ranges from 0.15 to 0.85. Essentially, what the damping factor represents is the chance that the process of searching APIs will stop on the current node and restart from the node of mashup. To some extent, damping factor determines the length of the path to locate the required APIs. In this sense, using the refined knowledge graph can find the desired APIs at a lower cost, as it performs the random walk in the neighboring entities of q, while utilizing the full knowledge graph costs higher as it goes a long way from q.

2) EFFECTIVENESS OF QUERY-SPECIFIC WEIGHTING

To investigate the effectiveness of our proposed query weighting strategies, we conduct experiments by applying each strategy independently with the RR model. The experimental results are presented in Figure 4. According to the experimental results, all three strategies significantly improve the accuracy of recommendations, the strategy Q1 and Q3 are better than the strategy Q2. For the strategy Q1, if an API matches more the requirement facets of mashup development, it should be ranked higher. Strengthening the connection weight between the API and the corresponding tag will guide a random walk to locate the API with a greater probability. The same explanation can be applied to strategy Q2. In this way, the accessing probability of popular APIs which do not match the requirements will be reduced, thus suppressing the *Matthew effect*.

As for the strategy Q3, the significant effect is that describing user needs with the standard classification terminology provided by the system helps to locate the required API. On one hand, the number of category entities is much smaller than the number of tag entities, so the former carries much more information than the latter. On the other hand, because the tag entities are usually generated by users, their ability to describe requirements is greatly reduced compared with the category entities. This also inspires us to pay more attention to the category information when developing recommendation algorithms.

We further let the β range from 0.3 to 0.7 to study its influence on the accuracy of recommendations. Figure 5 shows the model performance of RR when β takes different values. As the value of β increases, the recommendation performance changes little. Considering that the number of entities in Q_{tag} is much larger than the number of entities in Q_{cat}, we take $\beta = 0.3$ for subsequent experiments.

E. QUANTITATIVE COMPARISON

In this section, we conduct intensive experiments to observe the pros and cons of selected methods in both

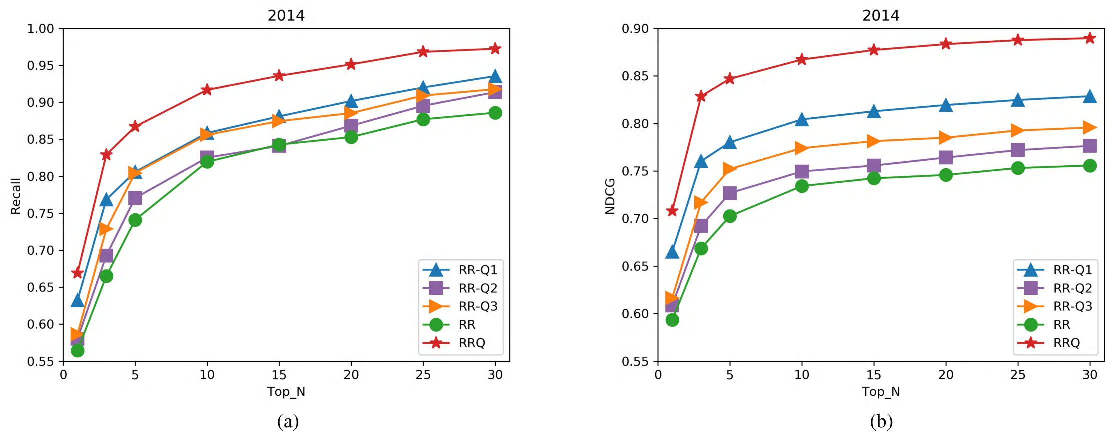


FIGURE 4. Performance comparison on Recall and NDCG of the RR model with different weighting strategies.

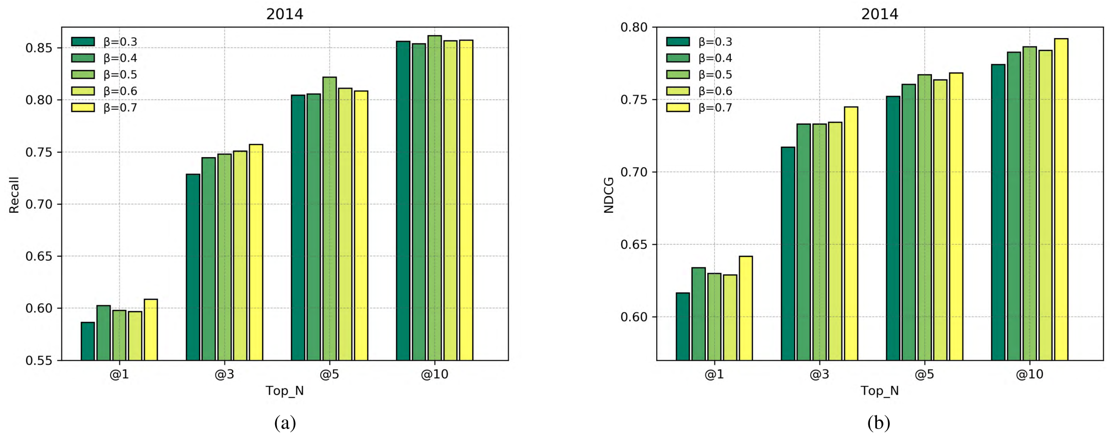


FIGURE 5. Performance comparison on Recall and NDCG of the RR model with different value of β.

recommendation effects and efficiency. The experimental results are correspondingly given in Table 2 and Figure 6.

1) PERFORMANCE OF RECOMMENDATIONS

From Table 2, the accuracy of recommendations is quite different on different subsets of experimental data w.r.t the baseline models Pop, CF and TM. For example, Pop works well on the 2012 dataset, CF does best on the 2013 dataset, and TM performs better on the 2014 dataset. These three methods use keywords, text similarities, and topical similarities respectively to match APIs, but ignore other important knowledge of mashup development, such as the common invocation pattern of APIs. It confirms the former observations that most of the existing works using single source information are vulnerable to data sparsity, and thus leading to relatively modest recommendation performance.

SPR basically performs better than Pop, CF and TM on both metrics of recommendation accuracy as it takes advantages of both co-invocation pattern of APIs and semantic relation between APIs and textual description of mashup. As for the RRQ-L model, its performance is not so good as that of RRQ. The main reason may be that the quality factors of text descriptions lead to unfortunate results of LSI analysis, harming the effect of the linear fusion with the random walk based relevance measure.

As for RF, it does not do as robust as expected and even worse than the baseline models. This shows that the information in the knowledge graph is not always advantageous for APIs recommendation, and the existence of some link information may become noise, which affects the estimation of relevance. In contrast, RWR utilizing refined knowledge graph almost outperforms all the additional baseline models. This proves that our new schema of knowledge graph effectively avoids the influence of noise information.

When introducing query weighting strategies, our recommendation model is further enhanced. According to Table 2, the gains reach about 17.0-22.4% in *Recall*@5, 11.9-15.1% in *Recall*@10 compared with other methods (except SPR in 2013), 9.5-27.2% in *NDCG*@5 and 11.9-23.4% in *NDCG*@10. On one hand, the information used to assess the relevance between requirements and candidate APIs is strengthened through query weighting. On the other hand, it effectively utilizes the preferences of

TABLE 2. Gains of RRQ for top-5 and top-10 APIs recommendation.

Time	Methods	Recall@5	Recall@10	NDCG@5	NDCG@10	HD@5	HD@10
2014	Pop	0.5058 (+71.4%)	0.6843 (+34.0%)	0.3289 (+135.3%)	0.4030 (+100.8%)	0.8804 (+3.5%)	0.8711 (+4.4%)
	CF	0.3587 (+141.7%)	0.4131 (+121.9%)	0.3455 (+124.0%)	0.3606 (+124.4%)	0.8966 (+1.7%)	0.8477 (+7.2%)
	TM	0.5991 (+44.7%)	0.7373 (+24.3%)	0.4780 (+61.9%)	0.5306 (+52.5%)	**0.9470** (-3.8%)	0.9099 (-0.1%)
	SPR	0.6716 (+29.1%)	0.7989 (+14.8%)	0.4772 (+62.2%)	0.5191 (+55.9%)	0.9005 (+1.2%)	0.8832 (+2.9%)
	RRQ-L	0.7278 (+19.1%)	0.7625 (+20.2%)	0.7064 (+9.5%)	0.7233 (+11.9%)	0.9214 (-1.1%)	**0.9149** (-0.6%)
	RF	0.6998 (+23.9%)	0.7482 (+22.5%)	0.6557 (+18.0%)	0.6752 (+19.9%)	0.8621 (+5.7%)	0.8671 (+4.8%)
	RR	0.7410 (+17.0%)	0.8195 (+11.9%)	0.6851 (+12.9%)	0.7140 (+13.3%)	0.9103 (+0.1%)	0.9074 (+0.2%)
	RRQ	**0.8671**	**0.9168**	**0.7738**	**0.8093**	0.9114	0.9090
2013	Pop	0.4524 (+46.6%)	0.6300 (+16.5%)	0.3187 (+98.4%)	0.3929 (+69.1%)	0.9242 (+1.7%)	0.9125 (+2.8%)
	CF	0.5747 (+15.4%)	0.6300 (+16.5%)	0.5173 (+22.2%)	0.5442 (+22.1%)	0.9034 (+4.1%)	0.8698 (+7.9%)
	TM	0.4474 (+48.3%)	0.5797 (+26.6%)	0.3518 (+79.7%)	0.4027 (+65.0%)	**0.9787** (-4.0%)	**0.9489** (-1.1%)
	SPR	0.5557 (+19.4%)	**0.7436** (-1.3%)	0.4061 (+55.7%)	0.4324 (+53.7%)	0.9156 (+2.7%)	0.8967 (+4.6%)
	RRQ-L	0.5169 (+28.3%)	0.5748 (+27.6%)	0.4450 (+42.1%)	0.4728 (+40.5%)	0.9491 (-1.0%)	0.9441 (-0.6%)
	RF	0.4775 (+38.9%)	0.5482 (+33.8%)	0.4135 (+52.9%)	0.4420 (+50.3%)	0.8944 (+5.1%)	0.8991 (+4.3%)
	RR	0.5671 (+17.0%)	0.6375 (+15.1%)	0.5096 (+24.1%)	0.5396 (+23.1%)	0.9372 (+0.3%)	0.9362 (+0.2%)
	RRQ	**0.6633**	0.7337	**0.6323**	**0.6645**	0.9400	0.9382
2012	Pop	0.5308 (+49.2%)	0.7484 (+14.4%)	0.3636 (+111.8%)	0.4569 (+75.2%)	0.8862 (+8.2%)	0.8682 (+10.3%)
	CF	0.4046 (+95.7%)	0.4849 (+76.6%)	0.4161 (+85.1%)	0.4584 (+74.6%)	0.8938 (+7.2%)	0.8643 (+10.8%)
	TM	0.4744 (+66.9%)	0.5589 (+53.2%)	0.4286 (+79.7%)	0.4633 (+72.7%)	**0.9876** (-2.9%)	**0.9802** (-2.3%)
	SPR	0.4931 (+60.6%)	0.6636 (+29.0%)	0.3549 (+117.0%)	0.4151 (+92.8%)	0.9069 (+5.7%)	0.8895 (+7.7%)
	RRQ-L	0.4786 (+65.4%)	0.5758 (+48.7%)	0.4540 (+69.6%)	0.4882 (+63.9%)	0.9563 (+0.2%)	0.9489 (+1.0%)
	RF	0.3987 (+98.6%)	0.4687 (+82.7%)	0.3812 (+102.0%)	0.4095 (+95.4%)	0.9184 (+4.4%)	0.9371 (+2.2%)
	RR	0.6467 (+22.4%)	0.7469 (+14.6%)	0.6054 (+27.2%)	0.6483 (+23.4%)	0.9443 (+1.5%)	0.9409 (+1.8%)
	RRQ	**0.7918**	**0.8561**	**0.7701**	**0.8003**	0.9586	0.9580

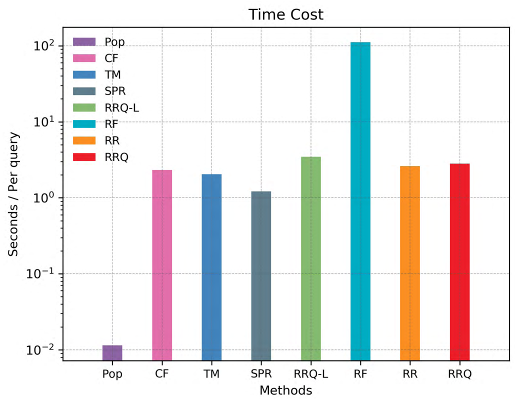


FIGURE 6. Average computational overheads of selected methods, where y-axis is in logarithmic scale.

developers and the historical knowledge in mashup developments (e.g., co-invocation pattern of APIs).

Finally, we examine the HD metric. In the experiment, Pop, CF and RF performed poorly, followed by SPR, which accords with the historical observations. For the first three methods, the principles of the algorithms where popular APIs are easier to get high rankings determines that the *Matthew effect* has a significant negative impact on the recommendation results. For SPR, the data base of its model training is the records of mashup. Because popular APIs have been involved in mashup developments, they are more likely to appear at the top positions of the recommendation lists. The *Matthew effect* should also be remarkable in the principle. However, due to the adoption of topic modeling technology(as TM outperforms all other methods in *HD*@5 and *HD*@10), this effect is somewhat neutralized.

When comparing RRQ with RF, we can find that RRQ improves RF about 4.4-5.7% on *HD*@5 and 2.2-4.8% on *HD*@10. The *Matthew effect* is suppressed to some extent. Of course, this should be attributed mainly to the use of simplified knowledge graph rather than the full knowledge graph, on which useless entities and connections tend to aggravate the negative transmission of the *Matthew effect*.

2) COMPUTATIONAL OVERHEAD

To investigate the computational overhead of selected methods. We conduct experiments on a computer equipped with i5-8300H CPU and 8-GB memory. Also, all the methods are implemented in Python. Depending on experimental results from Figure 6, our proposed methods achieve comparable time overhead to the baseline models. Especially when compared with RF, the time cost is substantially reduced, yet the recommendation effect is improved in RRQ.

F. CASE STUDY

In this section, we show actual cases on mashup-oriented API recommendation to provide a qualitative comparison of selected methods. *Groundtruth* represents the APIs used in the actual development process. Table 3 shows the top-3 recommendations for six randomly chosen target mashups covering different years, namely, **Place My Past** (2014), **Respin** (2014), **Healthy Lemur** (2013), **Mvbanana** (2013), **Pixurl Social Search** (2012) and **Flickoutr** (2012) where our method RRQ outperforms other counterparts in global performance by respectively achieving one hit, two hits, two hits, two hits, three hits and two hits.

Place My Past is a family history mapping application where the keywords include *Family*, *History*, *Genealogy*,

TABLE 3. Case study on APIs recommendation where APIs get hit are marked in bold font.

Target Mashup	Methods	Top 3 Recommendations of APIs		
Place My Past	Pop	Twitter	Family Echo Family Tree	Internet Archive
	CF	Google Maps	Google App Engine	Foursquare
	TM	Global Biodiversity Information Facility	43 Places	Wordstream Keyword Tool
	SPR	Google Places	Google Geocoding	**Openstreetmap**
	RRQ-L	Family Tree Maker	Family Echo Family Tree	Family Photoloom
	RF	Google Maps	Genealogycloud Onegreatfamily	Geni
	RR	Google Maps	Myheritage Family Graph	Family Echo Family Tree
	RRQ	Google Maps	**Openstreetmap**	Mapquest Open
	Groundtruth	**Openstreetmap**	**Freebase**	**Facebook Social Plugins**
Respin	Pop	Twitter	Google Maps	Youtube
	CF	**Spotify Metadata**	**Last.fm**	Youtube
	TM	**Last.fm**	Uk Street Level Crime	Google Street View Image
	SPR	Grooveshark	Lyricsfly	**Rdio**
	RRQ-L	Inlight Radio	Ubuntu One Music	Nokia Mixradio
	RF	Dubset Mixscan	Meemix	Metallizer
	RR	Dubset Mixscan	Soundrop	Metallizer
	RRQ	**Last.fm**	**Rdio**	Inlight Radio
	Groundtruth	**Spotify Metadata**	**Rdio**	**Last.fm**
Healthy Lemur	Pop	Iheartquotes	Strava	Google Health
	CF	Twilio	Sendgrid	Heroku
	TM	Myspace	Thrutu	**Foursquare**
	SPR	Twitpic	Facebook Social Plugins	Linkedin
	RRQ-L	**Facebook**	Kit Social	Social Networks Software
	RF	Google Maps	Life Fitness	Ergdb
	RR	Strava	Myfitnesspal	Ergdb
	RRQ	Strava	**Facebook**	**Twitter**
	Groundtruth	**Twitter**	**Foursquare**	**Facebook**
Mvbanana	Pop	Musicbrainz	Twitter	Google Maps
	CF	**Youtube**	Discogs	Spotify Metadata
	TM	Ooyala Video Player	Vzaar	Google Latitude
	SPR	Echo Nest	Last.fm	**Youtube**
	RRQ-L	**Musixmatch**	Lololyrics	**Lyricsfly**
	RF	Google Maps	**Youtube**	**Musixmatch**
	RR	**Youtube**	Qobuz	**Musixmatch**
	RRQ	**Youtube**	**Musixmatch**	Last.fm
	Groundtruth	**Youtube**	**Musixmatch**	**Lyricsfly**
Pixurl Social Search	Pop	Ebay	**Twitter**	**Youtube**
	CF	**Flickr**	Friendster	Bebo
	TM	Svpply	Yahoo Video Search	Google Ajax Feeds
	SPR	Myspace	Google Picasa	Google Custom Search
	RRQ-L	Revver	Veoh	Vmix Media Platform
	RF	Google Maps	**Twitter**	**Youtube**
	RR	**Twitter**	**Youtube**	**Flickr**
	RRQ	**Youtube**	**Twitter**	**Flickr**
	Groundtruth	**Youtube**	**Twitter**	**Flickr**
Flickoutr	Pop	**Twitter**	Youtube	Amazon Product Advertising
	CF	Youtube	**Twitter**	Twitpic
	TM	Face.com	Wuala	Brightkite
	SPR	Twitpic	Google Picasa	Myspace
	RRQ-L	Kit Social	**Flickr**	Social Networks Software
	RF	Google Maps	**Twitter**	**Flickr**
	RR	**Twitter**	**Flickr**	Youtube
	RRQ	**Twitter**	**Flickr**	500px
	Groundtruth	**Twitter**	**Flickr**	**Facebook Graph**

Mapping, *Social*, *Database*, *Openstreetmap*, *Display*, *Places*, *Viewer* and *Reference*. In the top-3 recommendations, none of the selected methods achieves satisfactory results. On one hand, the text description is quite insufficient; On the other hand, requirement information in Q is diverse, so top-recommendation consequences have various combinations and are not easy to hit. However, RRQ and SPR outperform other counterparts by achieving one hit.

Respin allows music lovers to import any of their hand crafted playlists into Rdio, where the keywords include *Music*, *Social*, *Playlists*, *Rdio* and *Streaming*. Since the requirement information is very clear and specific, and the APIs used are more common. CF achieves two hits on the top-3 recommendations. Depending on the music attribute emphasized in the requirement information, TM and SPR can also get one hit. In the case of sparse text descriptions, the textual information is rarely good enough, or even noise, so RRQ-L fails in the top-3 recommendations. RRQ achieves two hits just by keyword, although both RF and RR fail in this task, which confirms the advantage of weighting strategies.

Healthy Lemur is an app that lets you publicly *link your foursquare account*, *select the number of days per week that you're going to exercise*, *link your Facebook and/or Twitter accounts and you're all set*, where the keywords mainly include *Exercise*, *Self-being*, *Fun*, *Social*, *Gym*, *Health*, *Mobile*, *Search*, *Photo*, *Location*, *Mapping* and *Webhooks*. Since there is a sufficient text description for this mashup, TM achieves one hit on the top-3 recommendations. Pop, CF, SPR, RF and RR fail in the top-3 recommendations. In contrast, RRQ achieves two hits by using query weighting strategies followed by RRQ-L where achieves one hit.

Mvbanana is a radio station that plays music videos provides a fun way of discovering new or forgotten music where the keywords mainly include *Music*, *Video*, *Aggregate*, *Streaming*, *Metadata*, *Lyrics* and *Media*. In this recommendation task, the RWR-based models outperform other methods and produce good results.

Pixurl Social Search uses pixurl to search your favorite social platforms all from one web page. Its keywords include *Search* and *Social*, *Video*, *Photos*, *Media* and *Microblogging*. In the top-3 recommendations, both RR and RRQ achieve three hits and followed by Pop and RF, where both achieve two hits. As the text description of this mashup is also insufficient, CF just achieve one hit, TM and SPR do not perform well.

Flickoutr allows you to painlessly share your Flickr images by tag or set across Facebook and Twitter. Group sharing is supported and multiple accounts are supported. Its keywords mainly included *Photos*, *Social*, *Search*, *Tools*, *Microblogging* and *Video*. Due to the insufficient text description, TM and SPR are failure in the top-3 recommendations and RRQ-L just achieves one hit. RF, RR and RRQ achieve two hits and followed by Pop and CF, where both achieve one hit.

Regarding the negative *Matthew effect*, the recommendation lists generated by RF for *Mvbanana*, *Pixurl Social Search* and *Flickoutr* still put *Google Maps API* in the first place even if we do not have the requirement of mapping function. On the contrary, RR and RRQ can effectively suppress this effect and generate highly accurate APIs recommendation. Surely, the refinement of full knowledge graph cuts off the association with APIs that have high popularity but do not comply with the requirement.

V. RELATED WORKS

Service recommendation has been an active area of research for years [25]. In most of the cases, services are considered as a kind of item and a feedback matrix which contains ratings or quality of services in invocations from users to services are used for training and prediction. This obeys the general case of traditional recommender systems which have been widely used in e-commerce platform and social network services. However, an important feature of Web APIs is to participate in mashup (service composition) and achieve value-added applications [26]. How to effectively suggest APIs to match mashup requirements is quite different from the objective of these recommendation tasks. Consequently, we will summarize the related works which focus on providing solutions for mashup developments. On the surface, existing works are based on different principles of technology. However, these works can be coarsely classified into two categories, one focuses on the principle of collaborative filtering, and the other focuses on estimating the relevance between the mashup requirements and the candidate APIs.

As for the first category, Cao *et al.* [5] present a mashup service recommendation approach based on content similarity and collaborative filtering to rank and recommend top-k mashup services by combining the user interest value and the rating prediction value. Xu *et al.* [6] propose a coupled matrix model to describe the multi-dimensional social relationships among users, mashups, and services. They then design a factorization algorithm to predict unobserved relationships in the model to assist more accurate service recommendations. Rahman *et al.* [7] suggest a matrix factorization method based on integrated content and network-based service clustering. This ensures that the recommendation can be made within a comparable short list of interrelated services, with the latent relationship taken into account. Recently, Yao *et al.* [8] propose a matrix factorization with implicit correlation regularization to solve the recommendation problem and enhance the recommendation diversity. They conjecture that the co-invocation of APIs in mashups is driven by both the explicit textual similarity and implicit correlations of APIs such as the similarity or the complementary relationship of APIs.

Regarding the second category, various techniques are used to estimate the relevances between the requirements and the candidate APIs, such as link analysis and topic modeling. Also, different features such as API text description, network information, tags and popularity are utilized [13]–[17]. Xia *et al.* [9] propose a three-steps approach, including service clustering for each category, relevant categories identification, and category-aware service recommendation, to enhance the recommendation for the mashup creation. This work shows the importance of fully leveraging the meta-information in service profiles to achieve effective recommendations. Cao *et al.* [13] propose an integrated content and network-based service clustering and Web APIs recommendation method for mashup developments. They develop a two-level topic model by using the relationship among mashup services to mine the latent useful and novel topics for better service clustering accuracy. They also design a CF-based recommendation algorithm by exploiting the implicit co-invocation relationship between APIs inferred from the invocation history. Similarly, Li *et al.* [16] firstly exploit the enriched tags and topics information about mashups and Web APIs to calculate the similarity between Web APIs and the similarity between mashups. Then, they use the invocation times and category information of Web APIs to derive their popularity. Finally, multi-dimensional information is modeled by factorization machines to predict and recommend APIs for a target mashup. To leverage mashup descriptions and structures to discover important word features of services and bridge the vocabulary gap between mashup developers and service providers, Zhong *et al.* [10] jointly model mashup descriptions and component service using author topic model to reconstruct service profiles. Considering that different attributes may contribute differently to a service linkage in calculating the semantic distances among services, Bao *et al.* [11] suggest that we should simultaneously train separate models for individual attributes and develop a latent attribute modeling method to reveal context-aware attribute distribution. In addition, considering that different from individual service recommendation scenarios, some works are focused on optimizing the candidate APIs set to reduce the redundancy and improve the diversity of candidate APIs list [14], [27].

Some emerging recommendation methods exploit deep learning techniques as support, for example, Xue *et al.* [29] cluster APIs and use deep learning techniques to automatically generate API combinations based on requirement information. Compared with the evaluation methods in this paper, the main problem of the deep neural model is that its black box characteristics make the recommendation results lack explanatory. Also, it may be hard to learn representations of newly created APIs with novel features (e.g, new words) in textual descriptions, and thus suffer from a cold-start problem. In this regard, we need to explore in depth towards this technical direction.

Compared with the existing works, our methods can be grouped into the second category, but the difference is that we provide a simple yet effective framework to use the rich information (precise summaries of API functions and invocation pattern of APIs) in the knowledge graph of API ecosystem, which effectively overcomes the problem of data sparsity, thus achieving better recommendation accuracy.

VI. CONCLUSION

We have proposed a novel approach for mashup-oriented API recommendation based on random walks on knowledge graph. Specially, we enhance the preliminary model by refining the knowledge graph and suggesting query-specific weighting strategies on the graph construction. The new method can not only provide accurate recommendation results but also reduce computational overhead and suppress the negative *Matthew effect* in APIs recommendation.

However, there still exist some limitations w.r.t our method. On the one hand, describing requirements of mashups with keyword combinations is too high-level to judge the APIs used in a mashup, as this treatment requires developers to describe their needs accurately and completely, which depends on professional skills and experience. On the other hand, the proposed method can work well in the evaluation is mainly because that ProgrammableWeb defines a limited number of category keywords, and uses such keywords for both APIs and its mashups. If there are a large number of mashups used different keywords, it may be hard to be applied to recommend APIs for those mashups. To overcome these shortcomings, it is necessary to introduce natural language processing technology to accurately summarize those freestyle documents of requirement descriptions and align them to standard categories/tags. This would decrease the difficulty of requirement description and make our methods easier to be applied to actual scenarios.

As for future works, we want to explore other datasets to deeply evaluate our proposed methods. Also, we will study more simple yet elegant query weighting strategies to enhance the model. Additionally, as the negative *Matthew effect* always exists in the RWR model, we still need to pursuit advanced solution for it. Finally, adding other technologies, such as deep understanding of textual documents, to study the more widely used recommendation method is also our future goal.

REFERENCES

[1] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar, “From the service-oriented architecture to the Web API economy,” *IEEE Internet Comput.*, vol. 20, no. 4, pp. 64–68, Jul./Aug. 2016.

[2] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso, “Information quality in mashups,” *IEEE Internet Comput.*, vol. 14, no. 4, pp. 14–22, Jul./Aug. 2010.

[3] A. Bouguettaya *et al.*, “A service computing manifesto: The next 10 years,” *Commun. ACM*, vol. 60, no. 4, pp. 64–72, 2017.

[4] X. Liu, G. Huang, Q. Zhao, H. Mei, and M. B. Blake, “iMashup: A mashup-based framework for service composition,” *Sci. China Inf. Sci.*, vol. 57, no. 1, pp. 1–20, 2014.

[5] 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, 2014.

[6] W. Xu, J. Cao, L. Hu, J. Wang, and M. Li, “A social-aware service recommendation approach for mashup creation,” in *Proc. IEEE 20th Int. Conf. Web Services (ICWS)*, Jun./Jul. 2013, pp. 107–114.

[7] 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.

[8] L. Yao, X. Wang, Q. Z. Sheng, B. Benatallah, and C. Huang, “Mashup recommendation by regularizing matrix factorization with API co-invocations,” *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2018.2803171.

[9] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, “Category-aware API clustering and distributed recommendation for automatic mashup creation,” *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 674–687, Sep. 2015.

[10] 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.

[11] Q. Bao *et al.*, “A fine-grained API link prediction approach supporting mashup recommendation,” in *Proc. IEEE Int. Conf. Web Services*, Jun. 2017, pp. 220–228.

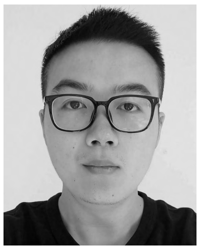
[12] D. Bianchini, V. Antonellis, and M. Melchiori, “WISeR: A multi-dimensional framework for searching and ranking Web apis,” *ACM Trans. Web*, vol. 11, no. 3, p. 19, 2017.

[13] B. Cao, X. Liu, M. M. Rahman, B. Li, J. Liu, and M. Tang, “Integrated content and network-based service clustering and Web APIs recommendation for mashup development,” *IEEE Trans. Services Comput.*, to be published.

[14] Q. Gu, J. Cao, and Q. Peng, “Service package recommendation for mashup creation via mashup textual description mining,” in *Proc. IEEE Int. Conf. Web Services*, Jun./Jul. 2016, pp. 452–459.

[15] Z. Yu, R. K. Wong, and C.-H. Chi, “Efficient role mining for context-aware service recommendation using a high-performance cluster,” *IEEE Trans. Services Comput.*, vol. 10, no. 6, pp. 914–926, Nov./Dec. 2017.

[16] H. Li, J. Liu, B. Cao, M. Tang, X. Liu, and B. Li, "Integrating tag, topic, co-occurrence, and popularity to recommend web APIs for mashup creation," in *Proc. IEEE Int. Conf. Services Comput.*, Jun. 2017, pp. 84–91.
[17] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, "Time-aware service recommendation for mashup creation in an evolving service ecosystem," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2014, pp. 25–32.
[18] M. Dojchinovski and T. Vitvar, "Linked Web APIs dataset," *Semantic Web*, vol. 9, no. 4, pp. 381–391, 2018.
[19] T. H. Haveliwala, "Topic-sensitive PageRank: A context-sensitive ranking algorithm for Web search," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 4, pp. 784–796, Jul. 2003.
[20] I. Konstas, V. Stathopoulos, and J. M. Jose, "On social networks and collaborative recommendation," in *Proc. 32nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2009, pp. 195–202.
[21] H. Wu, K. Yue, X. Liu, Y. Pei, and B. Li, "Context-aware recommendation via graph-based contextual modeling and postfiltering," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 8, p. 613612, 2015.
[22] H. Wu, X. Cui, J. He, B. Li, and Y. Pei, "On improving aggregate recommendation diversity and novelty in folksonomy-based social systems," *Pers. Ubiquitous Comput.*, vol. 18, no. 8, pp. 1855–1869, 2015.
[23] H. Wang, Z. Wang, and W. Zhang, "Quantitative analysis of matthew effect and sparsity problem of recommender systems," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2018, pp. 78–82.
[24] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.
[25] H. Wu, K. Yue, B. Li, B. Zhang, and C. H. Hsu, "Collaborative QoS prediction with context-sensitive matrix factorization," *Future Gener. Comput. Syst.*, vol. 82, pp. 669–678, May 2018.
[26] K. Huang, Y. Fan, and W. Tan, "An empirical study of programmable Web: A network analysis on a service-mashup system," in *Proc. IEEE 19th Int. Conf. Web Services (ICWS)*, Jun. 2012, pp. 552–559.
[27] W. Gao and J. Wu, "A novel framework for service set recommendation in mashup creation," in *Proc. IEEE Int. Conf. Web Services*, Jun. 2017, pp. 65–72.
[28] T. Liang, L. Chen, J. Wu, and A. Bouguettaya, "Exploiting heterogeneous information for tag recommendation in API management," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun./Jul. 2016, pp. 436–443.
[29] Q. Xue, L. Liu, W. Chen, MC. Chuah, "Automatic generation and recommendation for API mashups," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2017, pp. 119–124.
[30] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth, "The author-topic model for authors and documents," in *Proc. 20th Conf. Uncertainty Artif. Intell.*, 2004, pp. 487–494.
[31] L. Guo, X. Cai, F. Hao, D. Mu, C. Fang, and L. Yang, "Exploiting fine-grained co-authorship for personalized citation recommendation," *IEEE Access*, vol. 5, pp. 12714–12725, 2017.



XIN WANG received the B.Sc. degree in electronic science and technology from the Kunming University of Science and Technology, Kunming, China, in 2016. He is currently pursuing the master's degree in computer science with Yunnan University. His current research interests mainly include service computing and recommender systems.

HAO WU received the bachelor's degree in computer science from Zhengzhou University, in 2001, and the master's and Ph.D. degrees in computer science from the Huazhong University of Science and Technology, in 2004 and 2007, respectively. He is currently an Associate Professor with the School of Information Science and Engineering, Yunnan University, China. He has published more than 50 papers in peer-reviewed international journals and conferences, such as the IEEE TSC, JASIST, FGCS, the *Journal of Supercomputing*, KBS, and PUC. He has co-authored a monograph published with World Scientific. He has served as a Reviewer and a PC Member for many venues. His research interests include service computing, information filtering, and recommender systems.

CHING-HSIEN HSU is currently a Professor with the Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan. He has been acting as an author/co-author or an editor/co-editor of 10 books from Elsevier, Springer, IGI Global, World Scientific, and McGraw-Hill. His research interests include high-performance computing, cloud computing, parallel and distributed systems, big data analytics, ubiquitous/pervasive computing, and intelligence. He has published 100 papers in top journals, such as the IEEE TPDS, the IEEE TSC, the IEEE TCC, the IEEE TETC, the IEEE SYSTEMS, the IEEE NETWORK, and ACM TOMM, and book chapters in these areas. He is a Fellow of IET and a Senior Member of the IEEE. He received the Distinguished Award for Excellence in Research, nine times, from Chung Hua University. He is the Vice Chair of the IEEE TCCLD, an Executive Committee Member of the IEEE TCSC, and the Taiwan Association of Cloud Computing. He is serving on the Editorial Boards of a number of prestigious journals, including the IEEE TSC and the IEEE TCC.