Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 4960439, 10 pages
https://doi.org/10.1155/2020/4960439

Research Article

Automating Mashup Service Recommendation via Semantic and Structural Features

Wei Xiong,[1,2] **Zhao Wu**,[1] **Bing Li**,[2] **and Bo Hang**[1]

[1]*Hubei University of Arts and Science, Xiangyang 441000, China*
[2]*International School of Software, Wuhan University, Wuhan 430072, China*

Correspondence should be addressed to Zhao Wu; wuzhao73@163.com

Received 4 February 2020; Revised 9 June 2020; Accepted 6 July 2020; Published 12 August 2020

Guest Editor: Chunlai Chai

Increasing physical objects connected to the Internet make it possible for smart things to access all kinds of cloud services. Mashup has been an effective way to the rapid IoT (Internet of Things) application development. It remains a big challenge to bridge the semantic gap between user expectations and application functionality with the development of mashup services. This paper proposes a mashup service recommendation approach via merging semantic features from API descriptions and structural features from the mashup-API network. To validate our approach, large-scale experiments are conducted based on a real-world accessible service repository, ProgrammableWeb. The results show the effectiveness of our proposed approach.

1. Introduction

Internet of Things (IoT) was firstly introduced to the community in 1999 for supply chain management. Now, we will arrive to the post-cloud era, where there will be large amounts of smart things to access all kinds of cloud services, and the capabilities of smart things can be enhanced by interacting with other functional entities through the interfaces of cloud services. Since the functionality of an individual service is too simple to satisfy the complex requirements of users, in an IoT application, people prefer to combine several cloud services together. In recent years, mashup technology has gained great attention since it can support the development of IoT applications by composing existing cloud services in the form of web APIs.

Several platforms, including Seekda1, Google, and ProgrammableWeb [1], enabled vibrant software ecosystems between service providers and developers, where developers utilize one or more query items such as keywords and category. However, keyword-based service recommendation mechanisms usually have low accuracy [2]. NLP techniques are increasingly applied in the software engineering domain, which have been shown to be useful in requirements engineering [3], usability of API documents [4], and other areas [5]. Thus, semantic queries may get more accurate results.

Most of API descriptions and the tags are all text. Then, these approaches use similar semantic metrics that capture the service similarity, such as the similarity of service descriptions and tags. Indeed, most of similarity metrics, which are proposed to quantify the similarity of service descriptions and the similarity of tags, are based on the semantic information in the text. Some works utilized structural metrics to quantify the structural similarity where the metrics reached the topological information extracted from methods, attributes, classes and their couplings, etc [6–8]. However, there are very few works in which structural similarity has been introduced to guide the service ranking.

We present AMSRSSF (automating mashup service recommendation via semantic and structural features), a framework that utilizes NLP and graph-embedding techniques to recommend services for developers in this study. AMSRSSF takes as input the developers' personalized requirements and structural semantic and determines which services can be recommended for developers. Furthermore, we show that the structural semantics can be generated from a two-mode graph, which can describe mashups, web APIs, and their relations. We evaluate AMSRSSF against a dataset

of description documents including 10050 web-based services and 7155 mashups. Our experiments demonstrate that our approach can rank mashup services efficiently, and its performance is better than semantic-based mashup service ranking approaches alone.

This paper makes the following main contributions:

(1) We propose a mashup service recommendation approach via merging semantic features from API descriptions and structural features from the mashup-API network

(2) We conduct comprehensive experiments on a real-world dataset, demonstrating the effectiveness of our approach

The remainder of this paper is organized as follows: Section 2 gives a motivating scenario and presents our approach. Section 3 describes the experiments in detail. Section 4 discusses the related works, and Section 5 concludes the paper with future work directions.

2. Automating Service Recommendation via Semantic Features and Structural Features

In this section, we first present a scenario to illustrate the motivation of our work in Section 2.1. Then, we discuss the issues of mashup service recommendation in Section 2.2; next, we propose a mashup service recommendation approach via semantic and structural features in Sections 2.3–2.8.

2.1. Motivation. Under Internet scenario, more structures are presented between data objects. A typical scenario is a knowledge graph that consists of a global relationship diagram of user behavior data and an item with more attributes. Structural and semantic similarities characterize different aspects of the service similarity. Indeed, two similarities are orthogonal, which motivate our work. We expected that better service recommendation approaches might be proposed by using structural features. In particular, we propose a simple but effective approach AMSRSSF to rank mashup services by using semantic and structural features. First, it applies a two-mode graph to describe mashups, web APIs, and their relations formally. Second, we quantify the structural similarity between every pair of mashup services based on the two-mode graph. The structural similarity results from the structural context in which web APIs are used by mashup services. Finally, we introduce a merging embedding vectors algorithm that only considers the pairwise similarities between mashup services to rank them effectively.

According to the above, it is certainly valuable to introduce structural features and then improve the accuracy of recommendation based on structural semantics.

Finally, there are some other issues which need to be addressed:

(1) How to extract semantic from natural language API descriptions?

(2) How to generate structural semantics of the mashup-mashup network?

(3) How to merge multiple embedding vectors for better accuracy?

(4) How do we design experiments for performance evaluation?

2.2. Problem Description. Our work considers the task of recommendation via representation learning as follows: given a requirement q described in multidimensional information, which includes semantic and structural features, the corresponding recommended services are given via similarity calculation, which exists in terms of the representation learning model.

Our model consists in learning a function $S(\cdot)$, which can score requirement-service pairs (q, t). Hence, finding the top-ranked answer $t(q)$ to a requirement q is directly carried out by

$$\hat{t}(q) = \arg\max_{t' \in K} S(q, t'), \tag{1}$$

and to handle multiple recommended services, we present the results as a ranked list rather than taking the top service.

2.3. Overview of Our Framework. In this section, we will present an overview of our approach. As shown in Figure 1, our approach mainly consists of four components: semantic extraction based on NLP pipeline, structural embedding generation of the mashup-mashup network, merging of multiple embeddings, and recommendation based on fusion embedding. More specifically, the descriptions of web-based services will be crawled from ProgrammableWeb.

We firstly preprocess natural language API descriptions using NLP pipeline and extract text semantic embedding. We then generate structural embedding of the mashup-mashup network; next, we merge semantic embedding and structural embedding. Finally, the developers give multi-dimensional information of requirements, which will be parsed into semantic and structural features and passed to similarity calculation for ranking (Figure 2).

2.4. Semantic Extraction Based on the NLP Pipeline. This section preprocesses the sentences in API descriptions using NLP pipeline named as NLTK [9], which resembles the following high-level flow in Figure 3.

For purposes of illustration, we introduce a sample as follows:

> "The Twitter microblogging service includes two RESTful APIs. The Twitter REST API methods allow developers to access core Twitter data."

We have the following steps.

2.4.1. EOS (End of Sentence) Detection. Breaking the texts into paragraphs might be important. However, it may be unlikely to help EOS detection, which marks the boundary of a sentence from texts and breaks them into sentences. We

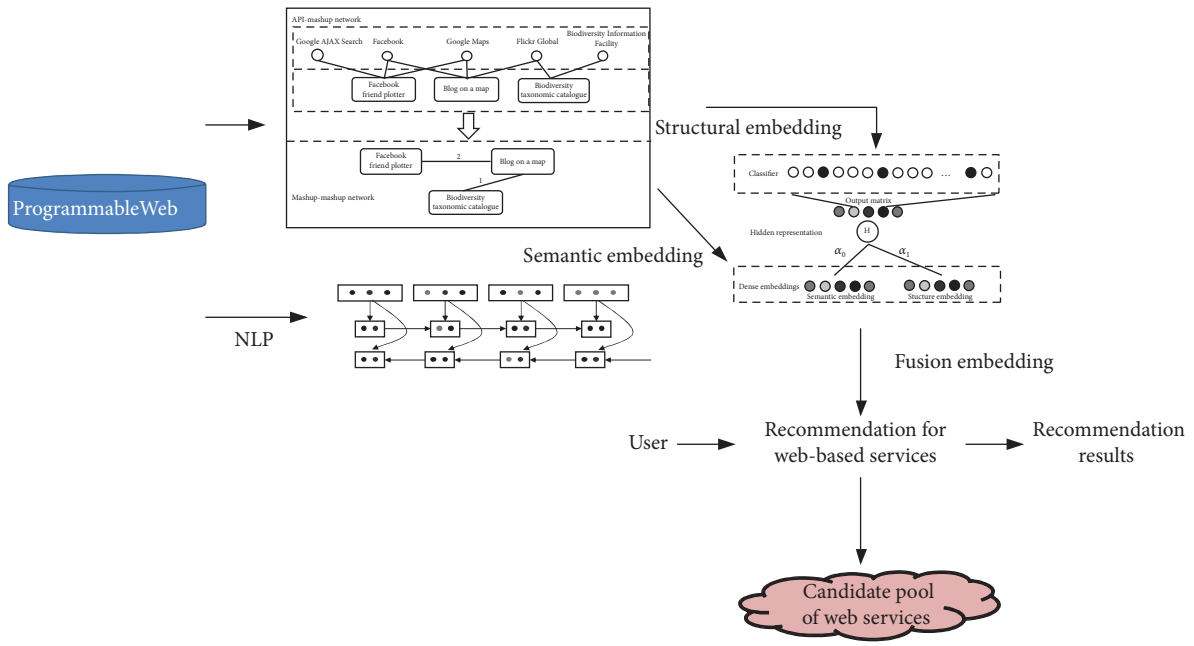

Figure 1: Overview of our approach.

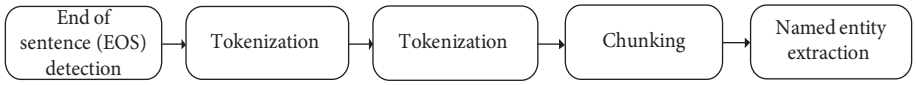

Figure 2: Overview of the NLP pipeline.

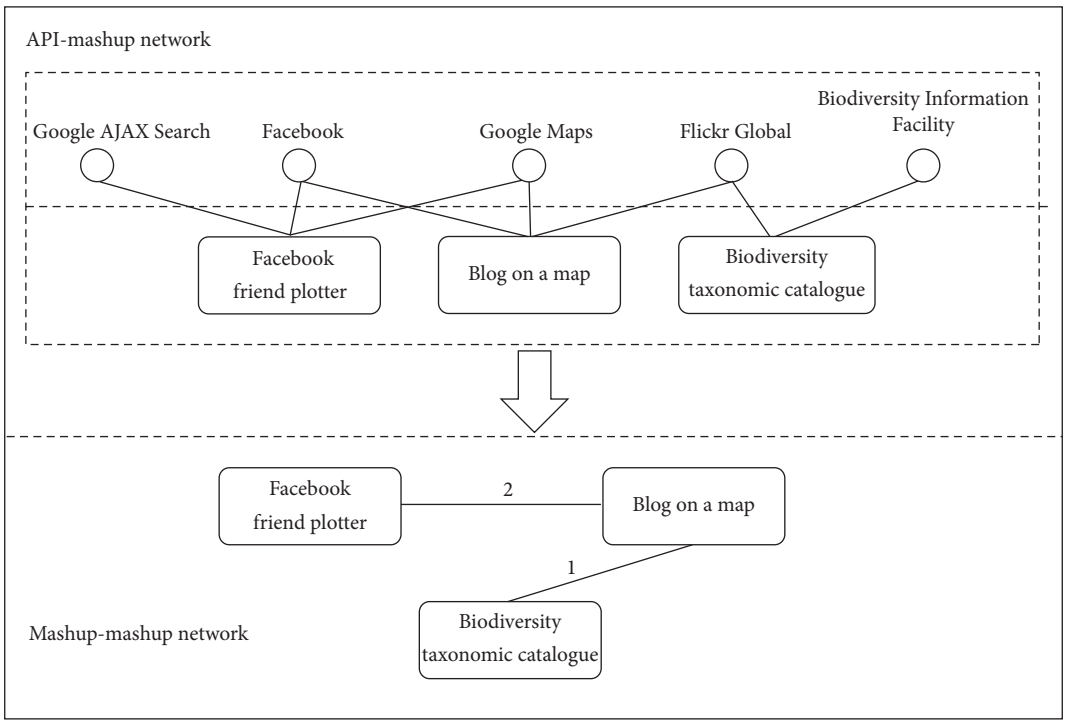

Figure 3: Mashup-API network and mashup-mashup network.

utilize them to represent logical units of thought and tend to give a predictable syntax for further analysis.

We will obtain the following results by parsing a sentence with NLTK:

> ["The Twitter micro-blogging service includes two RESTful APIs. The Twitter REST API methods allow developers to access core Twitter data."]

A period "." is utilized to mark the end of a sentence.

However, there are other legal usages of the period such as decimal (periods between numbers), ellipsis "...," and shorthand notations just like "Mr." and "Dr.". We can overcome this difficulty by looking up abbreviations from WordNet and detecting decimals including period character by regular expressions.

Moreover, there are instances where an enumeration list is used to describe data type, such as the following:

"This service where you can obtain the following data: (a) $abc\ldots$, (b) $xyz\ldots$."

It is prone to understand the implication for a human, but it is arduous to seek appropriate boundaries as far as a machine is concerned. An improvement over breaking on characters "." is leveraging the Syntax information:

(1) Placements of tabs

(2) Bullet points (numbers, characters, roman numerals, and symbols)

(3) Delimiters such as ":" to detect appropriate boundaries

We further improve the EOS detection using the following patterns:

(1) We remove the leading and trailing " $*$ " and "-" characters in a sentence.

(2) We consider the following characters as sentence separators: "–", "-", "ϕ", "§", "¥", "◇", "}", "|", "∼", and "★" ....

(3) For an enumeration sentence, we split the sentence into short ones for each enumerated item.

Currently, sentence detection has occurred for arbitrary text.

2.4.2. Tokenization. We then split individual sentences into tokens, leading to the following results:

> [["The", "Twitter", "microblogging", "service", "includes", "two", "RESTful", "APIs", "."], ["The", "Twitter", "REST", "API", "methods", "allow", "developers", "to", "access", "core", "Twitter", "data","."]]

Here, tokenization appeared to split on whitespace, and it tokenized out EOS correctly as well. We further improve the discrimination by determining whether a period is the end of sentence marker or part of an abbreviation.

2.4.3. POS (Parts of Speech) Tagging. Part of speech of each token in a sentence means "word tagging" or "grammatical tagging," where the state-of-the-art approaches have been shown to achieve 97% accuracy in classifying POS tags for well-written news articles. We can obtain the following results:

> [[("The", "DT"), ("Twitter", "NNP"), ("micro-blogging", "NNP"), ("service", "NNP"), ("includes", "VBZ"), ("two", "NUM"), ("RESTful","'JJ"), ("APIs","'NNP")], [("The", "DT"), ("Twitter", "NNP"), ("REST", "NNP"), ("API", "NNP"), ("methods", "NNP"), ("allow", "VB"), ("developers", "NNP"), ("to", 'IN"), ("access", "VB"), ("core", "JJ"), ("Twitter", "NNP"), ("data", "NNP"), (".", ".")]]

"NNP" manifests a noun as the part of a noun phrase, "VBD" manifests a verb in simple past tense, "DT" manifests an article, "IN" manifests a preposition, "NUM" manifests a numeral, and "JJ" manifests an adjective.

After finishing POS tagging, we can chunk nouns as part of noun phrases and then try to deduce the types of entities (e.g., people, places, organizations, etc.).

2.4.4. Chunking. Chunking means to analyze tagged tokens from sentence and identify logical concepts, such as "Pandora Internet radio" and "Google Maps." We annotate such phrases as single lexical units such as a noun phrase and a verb phrase. The state-of-the-art approaches achieve around 97% accuracy in classifying phrases and clauses over well-written news articles[4]. We develop chunking with more complex grammars to substitute default one in NLTK.

2.4.5. Named Entity Extraction. Named Entity Extraction means analyzing chunks and further tagging the chunks as named entities, such as people, organizations, and objects. We will obtain the following results:

> [Tree("S", [Tree("OBJ", [("The", "DT"), ("Twitter", "NNP"), ("microblogging", "NNP"), ("service", "NNP")]), ("includes", "VBZ"), Tree("OBJ", [("two", "NUM"), ("RESTful", "JJ"), ("APIs", "NNP"), (".", ".")]), Tree("S", [Tree("OBJ", [("The", "DT"), ("Twitter", "NNP"), ("REST"" "NNP"), ("PI", "NNP"), ("methods", "NNP")]), ("allow", "VB"), Tree("-PERSON", [("developers", "NNP")]), ("to", "IN"), ("access", "VB"), ("core", "JJ"), ("Twitter", "NNP"), ("data", "NNP"), (".", ".")]

We can identify that "The Twitter microblogging service," "two RESTful APIs," and "The Twitter REST API methods" have been tagged as an object and "developers" has been tagged as a person.

In addition, abbreviations usually lead to incorrect parsing for sentence; for example, "Application Programming Interfaces (APIs)" is treated as single lexical unit. We can work it out looking up abbreviations from WordNet [10].

2.4.6. Semantic Embedding Generation. In order to convert named entity into feature vectors and utilize them in latter models, we utilize word2vector to encode named entities from API descriptions.

Word2vector is a powerful method to extract meaningful grammatical and semantic features, which can transform a word into a fixed-length vector. Each named entity from API description can be reduced to embedding vector V_{text} , and semantic embedding of API description can then be represented as a fixed-length vector as follows:

$$V_m = \text{average}\,(V_{\text{text}i}),$$
$$I = \{1, 2, \ldots, N_m\}, \tag{2}$$

where N_m is the number of named entities of API description.

2.5. Two-Mode Graph of Mashup Services and Web APIs. The first step of structural embedding generation is to generate the mashup-mashup network. The network can be generated through mashup behavior sequence. The information, such as the same APIs between mashups, can be utilized to establish the edges between mashups as well, so as to generate structural graph. The embedding vectors based on this graph can be called structural embedding vectors.

In this article, two networks are utilized to represent all relationships of mashup services and APIs: one is the network between API and mashup services, and the other is the network between mashup services.

Definition 1. MAN (mashup-API network) is a network of 2 modes, where the nodes represent the API and the mashup services, and edges represent the relationship between them.

Definition 2. MMN (mashup-mashup network) is a weighted network between mashups, where each node represents a mashup service and edges represent the relationship between them; the weight of edge represents the relevancy between them. MMN can be represented with a triple:

$$\text{MMN} = \{N, E, W\}, \tag{3}$$

where N is a node set of all mashup services; E is a set of all edges in MMN network; W is an edge weights matrix. For example, if Node$i$ and Node$j$ are linked, then Wij is defined as the number of API intersections of two mashup services.

2.6. Structural Embedding Generation of the Mashup-Mashup Network. Structural embedding is designed to represent a network in a low-dimensional space and retain as much information as possible. By using structural embedding, each node in the network can be represented as a vector.

The traditional sequence embedding is inadequate for network structure. We can conduct a random walk on the graph structure to generate a large number of node sequences, which were input into word2vec as training samples and obtained structural embedding. Figure 4 illustrates the random walk process graphically.

As shown in Figure 4, random walk process firstly selects random mashup as starting point and produces the sequence, where the hop probability of the random walk needs to be formally defined, that is, the probability of traversing vj after reaching node vi. If the graph is an undirected graph, then the hop probability from node vi to node vj is defined as follows:

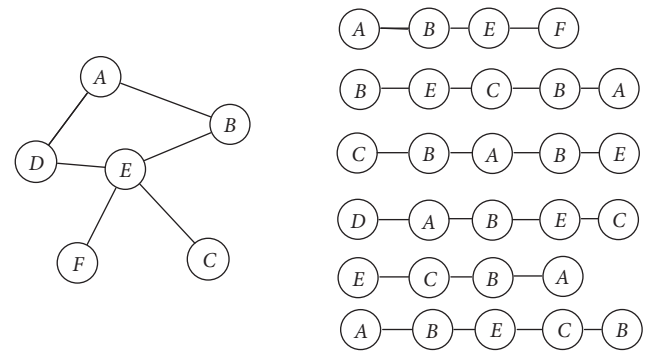

Figure 4: Random walk on network.

$$P\left(v_i \,\middle|\, v_j\right) = \begin{cases} (1-d) + d\left(\dfrac{w_{ij}}{\sum_{j \in N(v_i)} w_{ij}}\right), & \\ 0, \quad e_{ij} \notin \varepsilon, & \end{cases} \tag{4}$$

where $N(v_i)$ is the set of all outgoing edges of node v_i and w_{ij} is the weight of edge from nodevi to node v_j.

We further tradeoff the homogeneity and structural equivalence of network embedding by adjusting the weight of random walk.

Specifically, the homogeneity of the network means that the embeddings of nodes that are close to each other should be as similar as possible. As shown in Figure 5, the embedding expression of node u should be close to the nodes which are connected to s1, s2, s3, and s4. That is the embodiment of homogeneity.

Structural equivalence means that the embeddings of nodes with similar structures should be as close as possible. In Figure 4, both node u and node s6 are the central node of their local networks, respectively, which are similar in structure. Their embedding expressions should be similar, which is the embodiment of structural equivalence.

Random walk needs to be inclined to the process of breadth first search (BFS) in order to express the homogeneity of the network. There will be more homogeneous information in generated sequence because BFS prefers access to direct connection. On the other hand, depth-first search (DFS) can grasp the overall structure of the network through multiple hops. However, it is important to trade off the tendency of BFS and DFS. Formally, the hop probability from node v to node x is enhanced as follows:

$$P(v|x) = \lambda_{pq}(t, x) \cdot P(v|x), \tag{5}$$

where $\lambda_{pq}(t, x)$ is defined as follows:

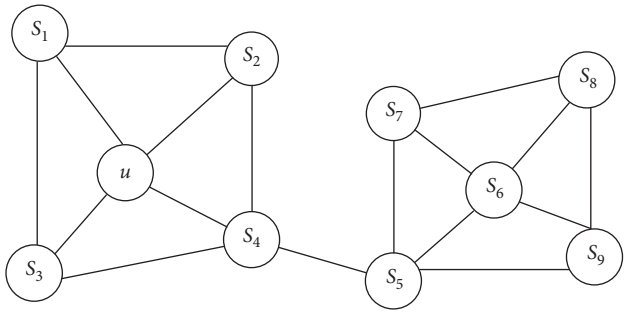

Figure 5: Homogeneity and structural equivalence of the network.

$$\lambda_{pq}(t,x) = \begin{cases} \dfrac{1}{p_r}, & \text{if } d_{tx} = 0, \\ 1, & \text{if } d_{tx} = 1, \\ \dfrac{1}{p_{io}}, & \text{if } d_{tx} = 2, \end{cases} \tag{6}$$

where d_{tx}refers to the distance from node t to node x and the parameters p_r and p_{io} jointly control the tendency of random walk. Parameter p_r is known as return parameter, which controls the probability of accessing the node visited repeatedly. Parameter p_{io} is called in-out parameter, which controls the outward or inward tendency. Random walk tends to access nodes close to node t if $p_{io} \geq 1$ and tends to access nodes far from node t if $p_{io} \leq 1$. Our approach enhances Node2vec [11] by replacing w_{ij} as $P(v|x)$.

Figure 6 shows the hop probability of our approach while jumping from node t to node v in the next step.

The homogeneity and structure equivalence of the network embodied by equation (5) can be intuitively explained in the recommendation system. Items with the same homogeneity are likely to be items of the same category and attributes, while items with the same structure are items with similar trends or structural attributes. There is no doubt that both of them are very important features in recommendation systems.

Next, we should encode above node sequences into structural embedding. Our model named sequence2-vector is derived from skip-gram, where the node sequence also utilizes central node sequence to predict the context, and its data structure can be represented by a triplet $\{s_{t-1}, s_t, s_{t+1}\}$. s_t is input, and $\{s_{t-1}, s_{t+1}\}$ is output. The neural network structure of the sequence2vector model is the common encoder-decoder architecture using the GRU model. Therefore, word vector is utilized while training sequence vector; the output of encoder is embedding vector of last word of node sequences. The formulas for the realization of encoding stage are as follows:

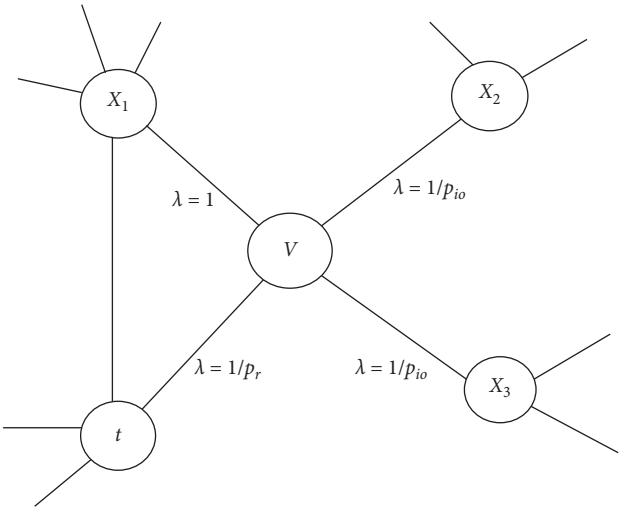

Figure 6: Hop probability of the network.

$$\begin{aligned} r^t &= \sigma\left(W_r x^t + U_r h^{t-1}\right), \\ z^t &= \sigma\left(W_z x^t + U_z h^{t-1}\right), \\ \overline{h^t} &= \tan h\left(W^d x^t + U\left(r^t \odot h^{t-1}\right)\right), \\ h^t &= \left(1 - z^t\right) \odot h^{t-1} + z^t \odot \overline{h^t}, \end{aligned} \tag{7}$$

where h^trepresents the output of the hidden layer at the time of t. While we take s_{t+1} as an example, the formula for the realization of decoding stage is as follows:

$$\begin{aligned} r^t &= \sigma\left(W_r^d x^{t-1} + U_r^d h^{t-1} + C_r h_i\right), \\ z^t &= \sigma\left(W_z^d x^{t-1} + U_z^d h^{t-1} + C_z h_i\right), \\ \overline{h^t} &= \tan h\left(W^d x^{t-1} + U^d\left(r^t \odot h^{t-1}\right) + C h_i\right), \\ h_{i+1}^t &= \left(1 - z^t\right) \odot h^{t-1} + z^t \odot \overline{h^t}, \end{aligned} \tag{8}$$

where C_r , C_z, and C are used as vector bias for reset gate, update gate, and hidden layer, respectively. Sequence2vector can even be combined into the subsequent deep learning network to retain different features of objects (Algorithm 1).

2.7. Embedding Fusion. From the above, we have got semantic embedding of API description and structural embedding through mashup behavior sequence. Then it is important to integrate multiple embedding vectors to form the final embedding of mashup. The simplest approach is to add the average pooling layer into the deep neural network to average different kinds of embedding. On this basis, we have added weights to each embedding, as shown in Figure 7. The hidden representation layer is the layer that performs the weighted average operation on different kinds of embedding. After the weighted average embedding vector is obtained, it is directly input into the softmax layer. In this way, the weighted $\alpha_i\,(i = 0, \ldots, 1)$ can be trained.

Input
MMN
Output
structure embedding vector h
Body
Random select a starting point for random walk using (5) and produce sequence s;
h = sequence2vector (s);
return h

Algorithm 1: Structure embedding algorithm.

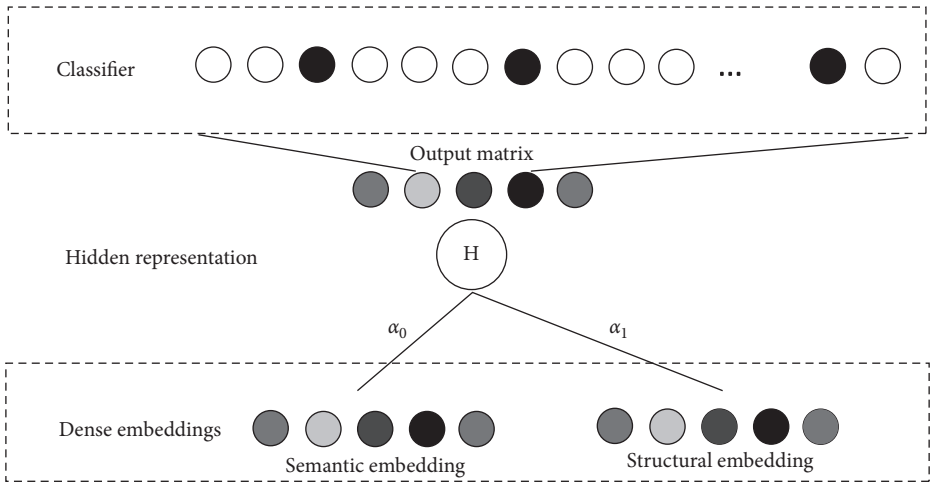

Figure 7: Embedding fusion.

In the actual model, we adopted $e^{\alpha i}$ instead of α_i as the weight of corresponding embedding, which can avoid the weight being 0 and own good mathematical properties during the process of gradient descent.

Our model finally inputs semantic and structure embedding into the model to generate the final fusion embedding.

2.8. Interfering Semantic-Related Services. In this section, we utilize fusion embedding from the above to calculate the similarity between mashups as follows:

$$\mathrm{sim}\left(h_i, h_j\right) = \exp\left(-\left\|h_i - h_j\right\|_1\right), \tag{9}$$

where $\| \|_1$ is the Manhattan distance. Finally, we recommend the top-k mashups to developers.

3. Experiments

In this section, we conduct experiments to compare our approach with state-of-the-art method. Our experiments are intended to address the following questions:

(1) Does structural embedding improve the effectiveness with respect to semantic similarities?

(2) What is the performance of top-k rankings of our embedding models?

3.1. Data Set Description. We adopt a real-world dataset, ProgrammableWeb, which was crawled from all registered services in publically available repository until Nov. 25, 2015. The dataset contains description documents including 12250 web-based services and 7875 mashups, which have attributes such as name, tags, summary, and description.

3.2. Metrics. In statistical classification [12], Precision is defined as the ratio of the number of true positives to the total number of items reported to be true, and Recall is defined as the ratio of the number of true positives to the total number of items that are true. F-score is defined as the weighted harmonic mean of Precision and Recall. Accuracy is defined as the ratio of sum of true positives and true negatives to the total number of items. Higher values of precision, recall, F-score, and accuracy indicate higher quality of identifying the semantic relationship. Based on the total number of TPs, FPs, TNs, and FNs, we computed the precision, recall, and F-score as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$
$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{10}$$
$$F - \text{score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}},$$

where TP denotes true positives, FP denotes false positives, FN denotes false negatives, and TN denotes true negatives.

MAP (mean average precision) is the average of precisions computed at the point of each of the relevant documents in the ranked sequence, whose equation is as follows:

$$\text{MAP} = \frac{1}{|Q_R|} \sum_{q \in QR} \text{AP}(q), \tag{11}$$

where AP is average precision and Q_R denotes relevant documents.

We utilize the normalized discounted cumulative gain (NDCG) [13] metric as well, which is a popular metric for evaluating ranking results. Given ideal rankings (used as ground truth) and the derived rankings, the NDCG value of top-k ranked services can be calculated by the following formula:

$$\text{NDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k}, \tag{12}$$

where DCG_k and IDCG_k denote the discounted cumulative gain (DCG) values of derived rankings and ideal rankings of top-k services.

The value of DCG_k can be calculated by the following formula:

$$\text{DCG}_k = \text{rel}_1 + \sum_{i=2}^{k} \frac{\text{rel}_i}{\log_2 i}, \tag{13}$$

where rel_i is the score of the service at position i of rankings. The premise of DCG is that a high-quality service appearing lower in a ranking list should be penalized as the score is reduced logarithmically proportional to the position of the result divided by $\log_2 i$.

3.3. Baseline Approaches. In this section, to show the effectiveness of our approach, we compare our approach with the following approaches:

(1) SimpleGraph [14]: this approach extracts features of subgraphs of question-answer pairs and utilizes a logistic regression classifier to predict the probability that a candidate answer is correct

(2) AvgPara: the semantics for a given question in this model are directly averaged to predict the answers

(3) DataAugment: this approach employs semantics for data augmentation during training which utilize the question-answer pairs and semantics to automatically generate new training samples

(4) SepPara [15]: this approach is the semantic scoring model, which is trained on classification data, without considering question-answer pairs

(5) Para4QA [15]: this approach results in learning semantics, where semantic scoring and QA models are trained end-to-end on question-answer pairs

3.4. Evaluation Protocols. In this section, we have designed the experiments to evaluate our approach. We then created a set of (q, t, l) triples, where q is a API description, t is a mashup-mashup network, and l is a label (classes of mashups). We manually tagged each pair with a label l. This resulted in a set of approximately 20 human judgments. The union of them composes set T1, which is considered as the baseline to evaluate the precision of mashup recommendation.

Five authors pick out 30 API descriptions of mashups as well; top-20 rankings are artificially extracted by two authors at least after analyzing and augmenting them, which is considered as the baseline to evaluate the rankings of mashup recommendation. The union of them composes set T2.

3.5. Training Details. We utilized encoder-decoder architecture using the GRU model with 2 layers, with 800 cells at each layer and 800 dimensional word embeddings. The complete training details are given below:

(1) We used word2vector to initialize the word embedding matrix. Out-of-vocabulary words were replaced with a special unknown symbol. We also augmented API descriptions with start-of-sequence and end-of-sequence symbols. Word vectors for these special symbols were updated during training. Model hyperparameters were validated on set T1. Each of the structural and semantic features is represented by a vector of 64 dimensions. The dropout rate was selected from {0.2, 0.3, 0.4}.

(2) Parameters were randomly initialized from a uniform distribution U(−0.08,0.08). The learning rate was 0.7. After 5 epochs, we began halving the learning rate every half epoch. We trained our models for a total of 20 epochs. The batch size was set to 150. To alleviate the exploding gradient problem, the gradient norm was clipped to 5.

(3) We used the DNN model to train weighted $\alpha_i (i = 0, \ldots, 1)$. The DNN model takes the combination of semantic and structural features as input, and each feature is represented by a vector of 16 dimensions. The total input vector dimension is 32 (16 + 16), and output vector dimension vector is the tag. To convert the input into a neural node, the hidden layer utilizes a linear transformation and then compresses the nonlinearity. The DNN model utilizes back-propagation and adjusts the weight α_i according to the training set. Finally, there is a softmax layer that translates the issue into classification problems.

Table 1: Performance of variants from the T1 test set.

Method	Precision (%)	Recall (%)	*F*-score (%)	Map (%)
SimpleGraph	86.32	26.55	40.60	16.54
AvgPara	77.99	32.13	45.51	22.62
DataAugment	71.66	35.55	47.52	32.97
Our approach	66.43	42.58	51.89	73.65

Table 2: Performance of top-k from the T2 test set.

Method	NDCG1 (%)	NDCG10 (%)	NDCG20 (%)
SimpleGraph	43.33	57.94	66.72
AvgPara	57.56	63.44	78.54
DataAugment	68.93	76.68	79.34
SepPara	72.30	80.84	82.49
Para4QA	81.73	82.70	87.84
Our approach	83.27	85.83	88.76

3.6. Results

3.6.1. Answer to RQ1. Table 1 presents the results of the accuracy experiments. AvgPara achieves better accuracy compared with SimpleGraph, since semantic embeddings contain more information than the features of subgraphs. DataAugment achieves better accuracy compared with AvgPara, since data augmentation works. Our approach achieves better accuracy compared with DataAugment, since structural embedding and semantic embedding are fused together.

3.6.2. Answer to RQ2. Table 2 presents the results of the ranking experiments. Our approach obtains the best prediction accuracy (largest NDCG values) under all the experimental settings consistently, since our approach considers fusion embedding of semantic and structural features, which improved the accuracy of similarity calculation.

3.7. Threats to Validity. The threats to external validity are related to the representativeness of the dataset. To evaluate the quality of semantic extraction and service recommendation, we applied our approach on real-world dataset, ProgrammableWeb, which is suitable for our study due to public availability and activeness. This threat has been partially mitigated by the fact that we collected description documents including 12250 web-based APIs and 7875 mashups. Furthermore, we applied our approach on top-10 subjects of them as well.

The threats to internal validity of our study are related to the results of the pairwise comparison on the criteria (see Section 3.5). There is a possibility that the results are subjective, as people may be biased. It may affect the weighted α_i which is assigned to the semantic and structural embedding. This threat has been partially mitigated by the fact that five authors elicit 10 service description documents from each of top-10 subjects independently to evaluate the quality of semantic relationship extraction and query, where we manually achieve top-10 rankings as the baseline in our evaluations.

4. Related Works and Discussion

Our approach touches some areas such as recommendation, DL artifacts, and requirements engineering. We next discuss relevant works pertinent to our proposed approach in these domains.

Most of recommendations focused on keyword-based searching, where the keywords are usually tokenized from the interface of web-based services, such as input, output, precondition, and postcondition. For example, Junghans et al. give an oriented functionalities and requests formalization method [16]. OWLS-MX is proposed to calculate the similarity of concept in OWL-S for recommendation [17]. Chen et al. give a clustering approach named as WTCluster using WSDL documents and service tags [18]. Mueller et al. give an approach based on nature-inspired swarm intelligence [19].

Semantic parsing offers NLP-based techniques, such as syntax directed [20], parse trees [21], combinatory categorical grammars [22, 23], and dependency-based semantics [24, 25]. These approaches typically have high precision but lower recall, which are sensitive to grammatically incorrect or ill-formed descriptions. Several approaches utilize either NLP or a merging of NLP and machine learning algorithms (SVM driven, etc.) to infer specifications, such as API descriptions [26–29].

Generally, most ranking approaches ranked services by utilizing the text in service profiles (mashup descriptions, API descriptions, WSDL documents, tags, etc.) to calculate the semantic similarity between every pair of services. In contrast, our proposed approach utilized structural similarities to guide the ranking activities, which can reduce false positives and reach higher performance.

5. Conclusion and Future Work

This paper introduces a mashup service recommendation approach via merging semantic features from API descriptions and structural features from the mashup-API network. Our approach can significantly outperform previous works after encoding semantic and structural features.

In spite of these promising results, some exciting challenges remain, especially in order to scale up this model to more dimensions. Furthermore, many more modes have to be carried out to encode the complex semantics into the embedding space.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The work described in this study was fully supported by the National Key Research and Development Program of China (no. 2017YFB1400602), the National Natural Science Foundation of China (nos. 61572371, 61702377, 61728202, 61572137, and 61873309), Science and Technology Development Program of Central Guide to Local Government of China (no. 2019ZYYD043), International Science and Technology Cooperation Program of Hubei Province (no. 2019AHB059), the Natural Science Foundation of Hubei Province in China (no. 2019CFC870), and Hubei Superior and Distinctive Discipline Group of "Mechatronics and Automobiles" (XKQ2020016).

References

[1] https://www.programmableweb.com/.

[2] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender system application developments: a survey," *Decision Support Systems*, vol. 74, pp. 12–32, 2015.

[3] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 3, pp. 277–330, 2005.

[4] U. Dekel and J. D. Herbsleb, "Improving API documentation usability with knowledge pushing," in *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, Vancouver, BC, Canada, May 2009.

[5] G. Little and R. C. Miller, Keyword programming in Java, vol. 16, Springer US, Boston, MA, USA, 2007.

[6] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of java software systems by weighted *k*-core decomposition," *Future Generation Computer Systems*, vol. 83, pp. 431–444, 2018.

[7] W. Pan, B. Song, K. Li, and K. Zhang, "Identifying key classes in object-oriented software using generalized *k*-core decomposition," *Future Generation Computer Systems*, vol. 81, pp. 188–202, 2018.

[8] W. Pan, H. Ming, C. K. Chang, Z. Yang, and D. K. Kim, "ElementRank: ranking java software classes and packages using multilayer complex network-based approach," *IEEE Transactions on Software Engineering*, 2019.

[9] http://www.nltk.org.

[10] https://wordnet.princeton.edu/.

[11] A. Grover and J. Leskovec, "node2vec: scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864, New York, NY, USA, August 2016.

[12] D. Olson, *Advanced Data Mining Techniques*, Springer-Verlag, Berlin, Germany, 2008.

[13] K. Kekäläinen and J. Kekalainen, "Cumulated gain-based evaluation of IR techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.

[14] S. Reddy, O. Täckström, M. Collins et al., "Transforming dependency structures to logical forms for semantic parsing," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 127–140, 2016.

[15] L. Dong, J. Mallinson, S. Reddy et al., "Learning to paraphrase for question answering," 2017, http://arxiv.org/abs/1708.06022.

[16] M. Junghans, S. Agarwal, and R. Studer, "Towards practical semantic web service discovery," in *Proceedings of the Extended Semantic Web Conference*, pp. 15–29, Heraklion, Greece, June 2010.

[17] M. Klusch, B. Fries, and K. Sycara, "OWLS-MX: a hybrid semantic web service matchmaker for OWL-S services," *Journal of Web Semantics*, vol. 7, no. 2, pp. 121–133, 2009.

[18] L. Chen, L. Hu, Z. Zheng et al., "Utilizing tags for web services clustering," in *Proceedings 2011 International Conference on Service-Oriented Computing*, pp. 204–218, Paphos, Cyprus, 2011.

[19] C. Mueller, N. Kiehne, and N. Kiehne, "Keyword-based service matching in a cloud environment using nature-inspired swarm intelligence," *Lecture Notes on Software Engineering*, vol. 3, no. 4, pp. 299–302, 2015.

[20] R. J. Mooney, "Learning for semantic parsing," *Computational Linguistics and Intelligent Text Processing*, Springer, Berlin, Germany, pp. 311–324, 2007.

[21] R. Ge and R. J. Mooney, "A statistical semantic parser that integrates syntax and semantics," in *Proceedings of the 9th Conference on Computational Natural Language Learning*, pp. 9–16, Stroudsburg, PA, USA, 2005.

[22] L. Zettlemoyer and M. Collins, "Learning to map sentences to logical form: structured classification with probabilistic categorial grammars," in *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pp. 658–666, Edinburgh, UK, 2005.

[23] L. S. Zettlemoyer and M. Collins, "Online learning of relaxed CCG grammars for parsing to logical form," in *Proceedings of the EMNLP-CoNLL*, pp. 678–687, Prague, Czech Republic, June 2007.

[24] P. Liang, M. I. Jordan, and D. Klein, "Learning dependency-based compositional semantics," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 590–599, Portland, OR, USA, June 2011.

[25] H. Poon, "Grounded unsupervised semantic parsing," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pp. 933–943, Sofia, Bulgaria, August 2013.

[26] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/∗icomment," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 145–158, 2007.

[27] R. J. Kate and R. J. Mooney, "Using string-Kernels for learning semantic parsers," in *Proceedings of the 21st International Conference on Computational Linguistics*, pp. 913–920, Sydney, Australia, July 2006.

[28] W. Xiong, Z. Lu, B. Li, B. Hang, and Z. Wu, "Automating smart recommendation from natural language API descriptions via representation learning," *Future Generation Computer Systems*, vol. 87, pp. 382–391, 2018.

[29] W. Xiong, Z. Wu, B. Li, Q. Gu, L. Yuan, and B. Hang, "Inferring service recommendation from natural language API descriptions," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pp. 316–323, San Francisco, CA, USA, June 2016.